FabScalar の Alpha 21264 命令セット対応とマルチプロセッサ環境フレームワークの構築

中林 智之[†] 佐々木 敬泰[†] Eric Rotenberg[††]
大野 和彦[†] 近藤 利夫[†]

近年，特徴の異なるプログラムやプログラム中のフェーズを効率的に実行するために，構成の異なるスーパースカラコアを複数個用いたヘテロジニアスマルチコアプロセッサが注目を集めている．構成の異なるスーパースカラコアをプログラムの特徴に合わせて使い分けることは，計算性能の向上や消費電力の低減に大きく貢献する．しかしながら，各コアの構成やコア数，キャッシュコヒーレンシのプロトコル，バスシステムなどの組合せの膨大さや，設計検証にかかる時間がヘテロジニアスマルチコアプロセッサを研究する上で大きな障害となっている．この問題を解決するために，様々な構成のスーパースカラコアの Verilog HDL コードを自動生成するツールセットとして FabScalar が提案されている．FabScalar を使用することにより，構成の異なるスーパースカラコアを短時間で設計することが可能となり，ヘテロジニアスマルチコアプロセッサの設計・検証にかかる時間を大幅に短縮できる．しかしながら，FabScalar は SimpleScalar で使用されている PISA 命令セットを採用している．PISA は 64bit アーキテクチャをサポートしておらず，またマルチスレッドのアプリケーションにも対応していない．それに加え，現在の FabScalar にはキャッシュシステムが実装されておらず，共有バスやキャッシュコヒーレンシといったマルチコアプロセッサの設計に必須な機能も実装されていない．これらの問題を解決するための第一歩として，我々は FabScalar によって生成されたスーパースカラコアを Alpha 21264 互換プロセッサへと移植した．また本論文では，FabScalar でマルチコアシステムを実現するためのフレームワークについても提案する．我々の行った検証の結果，移植したスーパースカラコアで SPEC2000 INT ベンチマークが正常に動作していることが確認できた．

Resaerch for Transporting Alpha ISA and Adopting Multi-processor to FabScalar

Tomoyuki Nakabayashi,[†] Takahiro Sasaki,[†] Eric Rotenberg,[††]
Kazuhiko Ohno[†] and Toshio Kondo[†]

Single-ISA heterogeneous multi-core processors are increasing importance in the processor architecture. However, designing a single-ISA heterogeneous multi-core requires design and verification effort which is multiplied by the number of different cores. FabScalar automatically generates diverse superscalar cores and contributes to research of heterogeneous multi-core processor. But, the current FabScalar supports PISA instruction set used in SimpleScalar, which does not support 64-bit architecture and multi-thread applications. In addition, necessary functions for designing multi-core processor such as cache coherency and shared bus are not implemented. In this paper, we modified the canonical superscalar core generated by FabScalar to implement Alpha compatible core that supports 64-bit architecture and multi-thread applications. We also present a framework to design multi-core processor. Our verification result shows that the customized core correctly executes some SPEC2000 INT benchmarks.

1. はじめに

近年，モバイルコンピューティングからハイパフォーマンスコンピューティングに至る幅広い分野で低消費電力かつ高性能を実現するプロセッサが求められている．この要求を実現するために，同一の命令セットアーキテクチャ (ISA: Instructions Set Architecture) で構成の異なるプロセッサコアを 1 チップ上に複数搭載したシングル-ISA ヘテロジニアスマルチコアプロセッサが注目を集めている[1)~3)]．シングル-ISA ヘテロジニアスマルチコアプロセッサでは，プログラムごとの特徴やプログラム中のフェーズごとの特徴によって，

† 三重大学大学院工学研究科
Graduate School of Engineering, Mie University
†† ノースカロライナ州立大学
North Carolina State University

搭載されているコアのうち最適な構成のコアを使用することにより，プログラムの実行効率を高める．しかしながら，チップ内で使用されるコアは，フェッチ幅，発行幅，パイプラインステージ数，命令スケジューリング（インオーダ，アウトオブオーダー），実行ユニット数などが異なるため設計および検証は各コアごとに行う必要がある．その結果，シングル-ISA ヘテロジニアスマルチコアプロセッサのコアの設計検証に要する時間は使用するコアの種類数に比例して増加してしまう．このことが，シングル-ISA ヘテロジニアスマルチコアプロセッサを設計する上で大きな障害となっており，実際の設計時のマイクロアーキテクチャの選択肢を狭くしている．

この問題を解決するために，ノースカロライナ州立大学では FabScalar[4] を提案している．FabScalar はスーパースカラコアの Verilog HDL コードを自動生成するツールセットであり，スーパースカラコアの 3 つの主要なパラメータ，すなわち，スーパースカラ幅，パイプライン段数，命令ウィンドウのサイズをはじめとした様々なテーブルサイズ，が異なるコアを短時間で設計することが可能である．ここでいうスーパースカラ幅とは，フェッチ幅だけでなくリネーム幅，発行幅，リタイア幅などのスーパースカラプロセッサ内のあらゆる幅を含んでいる．そのため，FabScalar はシングル-ISA ヘテロジニアスマルチコアプロセッサの設計および検証にかかる時間を大きく短縮することができる．

しかしながら，現在の FabScalar は命令セットとして SimpleScalar で使用されている Portable ISA (PISA) 命令セットを採用している．PISA は 64bit アーキテクチャをサポートしておらず，マルチスレッドアプリケーションにも対応していない．また，現在の FabScalar にはキャッシュシステムが実装されておらず，共有バスやキャッシュコヒーレンシといったマルチコアプロセッサの設計に必須な機能が実装されていない．これは，現在の FabScalar がシングルスレッドプログラムを対象としており，プログラムの実行フェーズにより最適な構成のコア上で実行することを想定しているためである．しかしながら，近年マルチスレッドプログラムが広く用いられるようになりつつあり，多数のコアを同時に効率的に使用することが望ましいと考えられる．

そこで，我々は FabScalar で Alpha 21264 命令セットをサポートすることを提案する．Alpha 命令セットは 64bit アーキテクチャであり，マルチスレッドアプリケーションにも対応しているため，FabScalar が現在抱えている問題を解決することが可能である．FabScalar で Alpha 命令セットをサポートするための第一歩として，我々は FabScalar によって生成されたスーパースカラコアに Alpha 21264 命令セットを移植することに成功した．また本論文では，FabScalar でマルチコアシステムを実現するためのフレームワークについても提案する．我々の行った検証の結果，移植したスーパースカラコアで SPEC2000 INT ベンチマークが正常に動作していることが確認できた．

以降，本論文は次のように構成される．まず，第 2 章で関連研究について述べ，次に第 3 章で FabScalar について説明し，第 4 章では現在の FabScalar の問題点についてまとめる．次に，第 5 章で FabScalar への Alpha 命令セット移植と SPEC INT を用いた検証結果について述べ，第 6 章で FabScalar でマルチコアシステムを実現するためのフレームワークを提案し，最後に第 7 章でまとめる．

2. 関 連 研 究

Illinois Verilog Model (IVM) は 4-way スーパースカラプロセッサの Verilog HDL を提供している[5]．現在の IVM の欠点としては，論理合成が不可能，または論理合成結果の周波数が非常に低いことである．IVM は現在 2 つのバージョンがリリースされており，バージョン 1.0 は L1 キャッシュやメモリインタフェースを含んだシミュレーションが可能であるが，論理合成をすることができない．また，バージョン 1.1 では，一部の機能を取り除くことにより論理合成を可能としているが，その合成結果の周波数は非常に低い．さらに，IVM ではスーパースカラの幅とパイプライン段数を変更することができないという問題点があり，この問題を解決することは困難である．それに対して，FabScalar では望まれたフェッチ幅やパイプライン段数を実現した Verilog HDL を生成することが可能である．

Strozek と Brooks は組込み向けの非常にシンプルで合成可能なフレームワークを提案している[6]．また，Tensilica の Xtensa Configurable Processors は命令のカスタマイズ，実行ユニット数や VLIW のデータパスまでの設計を自動化している[7]．しかし，FabScalar は複雑な構成のスーパースカラプロセッサを自動生成できるという点でこれらの研究と異なる．

Palacharla, Jouppi と Smith はスーパースカラプロセッサでクリティカルとなるパイプラインステージ（リネーミング，発行とフォワーディング）の遅延時間を見積るモデルを提案している[8]．また，Li らは幅

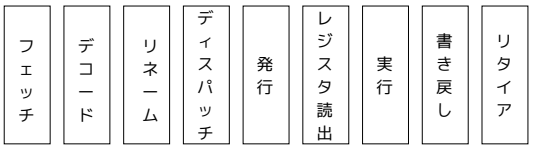

図 1 FabScalar の基本構成
Fig. 1 canonical superscalar processor

広く使用できる電力，面積およびタイミングをモデル化するマルチコアシステムのためのフレームワークを提案している[9]．これらに対して，FabScalar はコア全体の遅延モデルを提供することができ，さらに，実際に RTL で実装されたコアを提供することができる．FabScalar は研究のためのモデルを与えるだけでなく，実際のヘテロジアスマルチコアチップを設計するためのツールとして使用することができる．

3. FabScalar

本章では FabScalar ツールセットについて述べる．

3.1 Canonical superscalar processor と CPSL

図 1 に FabScalar の最も基本的な構成を示す．FabScalar ではこの構成を canonical superscalar processor と呼び，FabScalar で生成されるスーパースカラコアは全てこの構成を元にしたものとなる．canonical superscalar processor は 9 つのパイプラインステージにより構成される，すなわち，フェッチ，デコード，リネーム，ディスパッチ，発行，レジスタ読出し，実行，書戻しとリタイアである．また，それぞれのパイプラインステージを構成する要素を canonical pipeline stage library (CPSL) と呼んでいる．

CPSL は各パイプラインステージに対して，スーパースカラ幅やパイプライン段数などが異なる，種々の合成可能な RTL 設計を有している．表 1 に現在の CPSL で利用可能な構成を示す．表の 1 列目はパイプラインステージを，2 列目は各ステージで変更可能な幅および段数の範囲を示している．フェッチからディスパッチまでの全てのフロントステージでは，1-way から 8-way までのスーパースカラプロセッサとして構成が可能である．表の 2 列目では，サブパイプラインの段数の範囲についても記述されている．例えば，フェッチステージはフェッチ 1 とフェッチ 2 の 2 つのサブパイプラインに分られており，フェッチ 1 ステージは 1 あるいは 2 ステージのパイプラインとして構成することが可能である．表の 3 列目は各ステージ固有の構成要素で，そのサイズは RTL に与えるパラメータによって変更可能が可能である．これらについては，特に範囲の指定に制限はない．

表の 4 列目は，その他に考えられるアーキテクチャレベルのアプローチの選択肢や現在の FabScalar が採用しているマイクロアーキテクチャについて示している．

3.2 ファンクションシミュレータとタイミングシミュレータ

前節で説明した canonical superscalar processor は Verilog HDL で実装された RTL レベルのシミューレタであり，FabScalar ではこれをタイミングシミュレータと呼んでいる．一般的なプログラムにおいて，その最初のフェーズはプログラムの初期化ルーチンであり，そのプログラムの本質を表していない．また，初期化ルーチンは大規模なプログラムでは，それ相応のサイクル数を必要とする．例えば SPEC INT ベンチマークでも，大規模なデータセットを用いた場合，SimpleScalar の fast-sim を用いても数十分以上かかる．RTL シミュレーションは SimpleScalar と比べて大幅に遅いため，この初期化ルーチンをタイミングシミュレータで実行すると，プログラムの本質となるフェーズに至るまでに数時間以上を要してしまう．

そこで，FabScalar ではタイミングシミュレータとは別に，C 言語で記述されたファンクションシミュレータを有している．このファンクションシミュレータはプログラムの任意の時点までを高速にシミュレートし，必要な情報（プログラムカウンタやレジスタファイルの内容）をタイミングシミュレータへと引き継ぐ役割を果す．この機能により，FabScalar は評価を行いたいフェーズのみを高速にシミュレートすることが可能である．また，現在の FabScalar では実装の簡単化のために，システムコールが発生した際の処理をファンクションシミュレータで行っている．

3.3 現在の FabScalar の状況

文献[4] において，N. K. Choudhary らは FabScalar で生成された 12 個のスーパースカラコアについて，SPEC INT ベンチマークを用いた IPC の評価を行っている．また，合成された FabScalar コアのタイミング解析を行い同様の構成の商用プロセッサとクロックサイクルタイムを比較している．その結果，FabScalar で生成されたスーパースカラコアが商用プロセッサと比較して遜色のない動作速度を実現していることを明かとしている．

4. 現在の FabScalar の問題点

現在の FabScalar は命令セットとして，SimpleScalar で用いられている Portable ISA (PISA) を使用して

表 1 CPSL で使用可能な構成
Table 1 Overview of designs available in the CPSL

ステージ	幅，段数	ステージ固有の構成で変更可能なもの	マイクロアーキテクチャ
フェッチ	幅=1〜8, 段数=2〜5 フェッチ 1:1〜2 サブステージ フェッチ 2:1〜3 サブステージ	分岐 or パターンヒストリテーブル 分岐ターゲットバッファ（BTB） リターンアドレススタック（RAS）	分岐予測アルゴリズム 2 ウェイインタリーブ ブロックベース BTB vs. インタリーブ BTB ブロックアヘッド予測
デコード	幅=1〜8, 段数=1〜3	命令キュー	
リネーム, リタイア	幅=1〜8, 段数=1〜3 幅=1〜8, 段数=2	リネームマップテーブル（RMT） アーキテクチャマップテーブル（AMT） チェックポイント数:0 か 4 フリーリスト アクティブリスト	アーキテクチャマップテーブル 分岐予測ミス回復 ・チェックポイント ・例外と同様に扱う 例外回復 ・AMT から RMT を回復 レジスタの解放 ・AMT から前のマッピングを読込, AMT はフリーリストをプッシュ
ディスパッチ	幅=1〜8, 段数=1	発行キュー（IQ）, フリーリスト	フリーリストベース IQ
発行	幅=4〜8, 段数=1〜3	発行キュー（IQ）	アウト・オブ・オーダ
レジスタ読出し	幅=4〜8, 段数=1〜4	物理レジスタファイル	
実行	幅=4〜8, 段数=演算器ごと simple ALU:1〜5 個, 段数=1 complex ALU:1 個, 段数=3 分岐:1 個, 段数=1 ロードストア:1 個, 段数=2	ロードキュー（LQ） ストアキュー（LQ）	ストア-ロードフォワーディング
書戻し	幅=4〜8 段数=読出しと同じ		フォワーディング

いる．PISA は MIPS をベースに開発された命令セットで，MIPS ISA から遅延ロード，遅延分岐を排除したシンプルな命令セットを構成している．しかしながら，PISA は近年主流となりつつある 64bit アーキテクチャに対応していない．また，PISA はマルチスレッドアプリケーションへの対応もなされていない．

もう一つの問題点として，現在の FabScalar では，キャッシュシステムが実装されておらず，共有バスやキャッシュコヒーレンシといったマルチコアプロセッサシステムを構成するために必要な機能が実装されていない．これは，現在の FabScalar がシングルスレッドプログラムを対象としており，プログラムの実行フェーズに適したプロセッサコア上でプログラムを実行することを想定しているためである．しかしながら，近年マルチスレッドプログラムが広く用いられるようになり，多数のコアを同時に効率的に使用することが求められている．そのような，複数のコアが同時に活性化するシングル-ISA ヘテロジニアスマルチコアプロセッサにおいて，共有バスや共有キャッシュの実装は必須である．

そこで我々は，これらの問題を解決するために，FabScalar に Alpha 21264 命令セットを移植した上でマルチコアプロセッサを設計するフレームワークの提案を行う．

5. FabScalar への Alpha 命令セットの移植

本章では，Ahpha 21264 命令セットの FabScalar への移植について述べる．前述の通り，FabScalar でマルチアコプロセッサシステムを実装するための第一歩として，我々は FabScalar への Alpha 21264 命令セットの移植を行っている．Alpha 21264 命令セットは 64bit アーキテクチャであり，マルチスレッドアプリケーションにも対応しているため，現在の FabScalar が抱えている問題を解決することができる．

5.1 PISA 命令セットとの違い

FabScalar は現在進行中のプロジェクトであり，命令セットの移植は可能な限りコードに互換性を持たせることが望ましい．そこで本節では，Alpha と PISA 命令セット間の主な違いと，互換性を保つために行った実装方法について述べる．Alpha 21264 命令セットも PISA 命令セットも RISC プロセッサをベースとしており，多くの命令は一対一で対応できる．バス幅については，FabScalar は大部分のコードがパラメータ化されているため，比較的容易に 64 ビット化できる．Alpha 21264 命令セットと PISA 命令セットで大幅に異なる点として，1)Alpha は条件付転送命令を提供

している，2)Alpha では FPU の有無にかかわらず浮動小数点レジスタを必要とする，という 2 点が挙げられる．以下にその詳細と我々の採用した解決方法を述べる．

5.1.1 条件付転送命令

条件付転送命令とは以下のようなコードによって表される命令である．

$$CMOVXX\ S\_R_A,\ S\_R_B,\ D\_R_C \qquad (1)$$

ここで，"XX"は評価する条件（例えばイコールゼロ），"S_R_A"と"S_R_B"はソースレジスタ，"D_R_C"はディスティネーションレジスタである．つまり，"S_R_A"を"XX"で評価した結果が真ならば，"D_R_C"に"S_R_B"を代入し，偽ならば"元の R_C"の値を保持するという命令である．この条件付転送命令は，イン・オーダ実行のプロセッサで実装する場合は通常通り 2 ソースオペランド命令として実装することができる．しかしながら，アウト・オブ・オーダ実行のプロセッサに実装する場合には 3 つのソースオペランドを必要とする命令になってしまう．なぜならば，レジスタリネーミングによって"元の R_C"の値とは関係のない予測不能な値が代入されている物理レジスタがディスティネーションレジスタとして割当てられるからである．つまり，"R_A"と"R_B"に加えて"元の R_C"を読出し，条件が偽の場合には"元の R_C"をディスティネーションの"R_C"に書込む必要がある．そのため，通常の実装では発行する命令ごとに 3 ポートの読出しポートが必要となる．しかし，PISA では 2 ソースオペランドのみの命令を提供しており，3 ソースオペランドの命令を追加することは，オリジナルの FabScalar のレジスタファイルを変更するだけでなく，リネームや発行ロジックを大きく変更することに繋がってしまう．そこで，我々は後述の分裂命令 (fission instruction) 機構を活用し，条件付転送命令を実装している．

PISA では 1 命令で 2 つのディスティネーションに書込を行う 2 ディスティネーション命令を提供している．通常，この命令を実装するためには，レジスタの書込ポートを増加する必要があるが，SRAM の面積はポート数の 2 乗に比例するため，ハードウェア量が大幅に増加してしまう．オリジナルの FabScalar では，このような分裂命令は，2 ディスティネーション命令を 2 つの 1 ディスティネーション命令に分割することで，レジスタの書込ポートの増加なしに 2 ディスティネーション命令を実装するための分裂命令機能を備えている．我々はこの機能を，3 つのソースオペランドを必要とする命令である条件付転送命令を実装するために活用している．

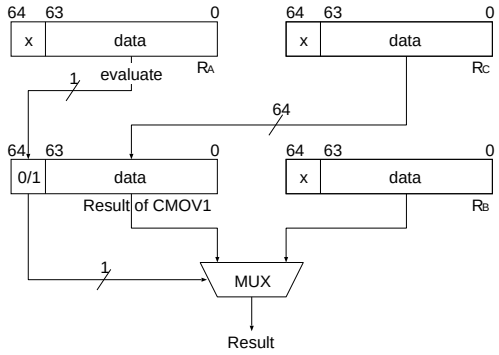

図 2 条件付転送命令の実装方法
Fig. 2 Implementation for conditional move.

図 2 に我々が行った条件付転送命令の実装方法を示す．分割後の条件付転送命令をそれぞれ CMOV1, CMOV2 と呼ぶ．CMOV1 は"R_A"と"元の R_C"を読出し，"R_A"を評価した結果をレジスタの最上位ビットに書込む．それと同時に"元の R_C"の値を下位 64 ビットに書込む．CMOV2 は CMOV1 で書込まれた結果と"R_B"を読出し，CMOV1 の結果の最上位ビットを使用して，最終的な書込結果を決定する．この命令の実装にあたり，レジスタファイルを 1 ビット拡張する必要があるが，これはレジスタファイルのポート数を増やすことと比較すると，小さなコストしかかからない．また，Verilog HDL のコードでは，データパスを定義するビットの指定を変更するだけで実装が可能であり，十分な互換性が保たれていると言える．

5.1.2 浮動小数点レジスタ

現在の FabScalar では，浮動小数点ユニットが実装されていない．しかしながら，Alpha 21264 命令セットでは，プログラム中で浮動小数点命令を使用するかしないかに関らず浮動小数点レジスタを有している必要がある．これは，浮動小数点制御用レジスタの読出しと書込に使用される．

我々は，要求される浮動小数点レジスタを，整数レジスタの論理番号 32 番から 63 番を仮想的に浮動小数点レジスタとして使用することで実装している．Alpha 命令セットでは，整数レジスタと浮動小数点レジスタが共に 64 ビット幅であるため，機能的な問題は発生しない．また，コードの実装も，64 本の論理整数レジスタを持つアーキテクチャとして実装するだけで良いため，オリジナルの FabScalar からの変更点はリネーミングテーブルとアーキテクチャマップテーブルのパラメータを変更するだけで実現できる．

5.1.3 Verilog HDL ジェネレータへの移植

これまでに述べたように，我々は FabScalar に Al-

表 2 生成したスーパースカラコアのパラメータ（コア 1）
Table 2 Parameters for generated superscalar core (core 1).

パラメータ	値
フェッチ/デコード/リネーム/ディスパッチ/発行/レジスタ読出/書戻し/リタイア幅	4
実行ユニット（Simple, Complex, 分岐, ロードストア）	1, 1, 1, 1
フェッチキュー*	16
アクティブリスト*	128
物理レジスタ*	128
発行キュー*	32
ロードキュー/ストアキュー*	32/32
分岐ヒストリテーブル*	8K エントリ
分岐ターゲットバッファ*	4K エントリ
リターンアドレススタック*	16
パイプライン段数	10

pha 21264 命令セットを移植するにあたり，FabScalar 本来の構成を大きく変更しないしように慎重に移植を行っている．このことにより，容易にオリジナルの FabScalar で用いられている Verilog HDL ジェネレータに今回の移植成果を反映することが可能である．なぜならば，FabScalar の Verilog HDL ジェネレータはパラメータ化可能な機能を除いて，CPSL の各マイクロアーキテクチャを一度 Verilog HDL で実装したものを生成時に Perl スクリプトで選択しているからである．PISA との違いである条件付転送命令と浮動小数点レジスタの使用方法とを前述のように実装したことにより，一部の機能ユニットの置換えとパラメータの調整を行うだけで Alpha 命令セットに対応した FabScalar の Verilog HDL ジェネレータを実装可能である．

5.2 SPEC による IPC の評価と論理合成による面積評価

我々は FabScalar で最も基本となる構成のスーパースカラコアの RTL を生成し，Alpha 21264 命令セットの移植を行った．生成したスーパースカラコアのパラメータを表 2 にまとめる．また，表 2 の中で*がついているパラメータは，現在の Alpha プロセッサで変更することができる項目である．

SPEC2000 INT ベンチマークの中から，BZIP, GZIP，GAP，MCF の 4 つのプログラムについて，最初の 1 億命令を Alpha 命令セット用に移植したファンクションシミュレータでスキップした後，1 億命令をタイミングシミュレータで実行し，その Instruction per cycle (IPC) を評価した．なお，入力データには reference のデータセットを用いた．評価に使用したスーパースカラコアは，表 2 と同一のパラメータのものと（コア 1），変更可能なパラメータを全て表 2 から 2 倍にしたもの（コア 2）の 2 つである．表 3 にその結果を示す．これらの全ての結果について，プログラムが正しく実行されていることを我々は確認している．

表 3 1 億命令実行した結果の IPC
Table 3 Results of executing 100 million instructions.

ベンチマーク	コア 1	コア 2
BZIP2	0.53	0.58
GZIP	0.46	0.46
GAP	0.84	0.98
MCF	0.40	0.39

表 3 から，一般的な 4-way スーパースカラコアの基準と比較すると IPC が低い水準になっていることがわかる．この原因としては，ロードの投機的実行ミス（競合するストア以前にロードが発行される）や分岐予測ミス発生時の回復手段として，非常に保守的な実装，すなわち，例外回復と同様の手法でプロセッサの回復を行っているためであると考えられる．また，GZIP や MCF では各種のテーブルサイズを 2 倍にしたにも関らず，性能が同一または若干低い値となっている．この現象は頻繁なロードの投機的実行ミスと分岐予測ミスの発生が原因であると考えられる．コア 2 はコア 1 より大きいウィンドウサイズを有しており，インフライトな命令がより多くプロセッサ内に存在する．そのため，例外回復が頻繁に必要となった場合，例外回復により多くのサイクル数がかかってしまい，性能が低下してしまう．

また，我々はハードウェア規模を見積もるために，Synopsys の Design compiler を使用し，移植を行った Alpha プロセッサのコア 1 の論理合成を行った．テクノロジにはルネサスエレクトロニクス株式会社の 40nm テクノロジを使用した．論理合成の結果，Alpha 命令セットを移植した FabScalar の面積は $1.713mm^2$ であることがわかった．また，我々の行った移植が論理合成可能な実装であることが明かとなった．現在の段階では，合成したネットリストに対する詳細なタイミング評価は行えていないため，今後詳細なタイミング評価を行う必要がある．

6. FabScalar のマルチプロセッサ対応のためのフレームワークの提案

一般にマルチコアプロセッサでは，共有メモリバス，マルチコアに対応したキャッシュシステムが必要になる．これらの資源は実行するプログラムの特性や利用できるチップサイズなどの要因により，複雑なトレードオフ問題が発生する．とくにヘテロジニアスマルチコアシステムでは，コア数に加え，搭載するコアの

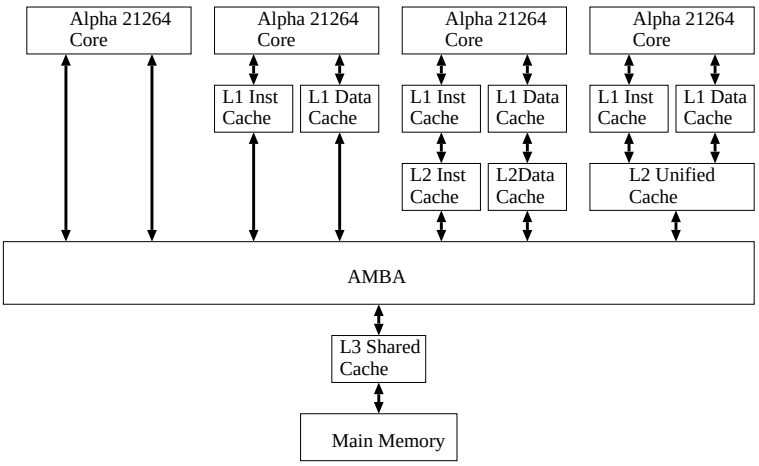

図 3 生成できるヘテロジニアスマルチコア構成の例
Fig. 3 Configurable Heterogeneous Multicore System.

アーキテクチャにより最適なバス構成やキャッシュ構成は変わってくるため，全ての手法を網羅的に評価するのは困難である．そこで，本章では現在設計している共有バス構成，キャッシュメモリ構成を自動的に生成できるツールについて説明する．

図 3 に構成可能なヘテロジニアスマルチコアの例を挙げる．L1 キャッシュは命令/データ分離，L2 キャッシュは命令/データ分離，あるいは統合キャッシュとして構成できる．また，L1 及び L2 キャッシュを外すこともできる．L3 キャッシュは共有キャッシュであるが，外すこともできるようにする．このようにキャッシュシステムの構成を可変にした場合，コアーキャッシュ間やキャッシューキャッシュ間などのモジュール間の接続方法が問題となる．そこで，各モジュール間を接続するための FabBus-Alpha を設計する．

特に組込み分野においては，キャッシュ階層の一部，あるいは全てを省略する場合がある．そのため，キャッシュ階層を自由に変更できるように Alpha 21264 コアやキャッシュなどの全てのモジュール間接続は AMBA[14),15)] バスを用いることにする．

6.1 FabBus-Alpha

FabBus-Alpha は，与えられたパラメータに従い各コアと共有メモリを接続するための共有バス一式を自動生成するツールである．チップ内バスの仕様は CoreConnect[13)] や Wishbone[12)]，AMBA[14),15)] など，様々なものが提案されているが，我々は仕様が公開されており，組込み分野などでも広く用いられている AMBA を採用する．

AMBA バスには様々なバージョンがあるが，我々は実装が容易な AMBA2[14)] をベースにした．AMBA2 はマルチプロセッサ向けチップ内バスの仕様の一つで，ARM 社により提案されているものである．AMBA2 はスプリットバストランザクションをサポートしており，また，64bit バスにも対応しているため，Alpha 版 FabScalar への適用が容易である．しかしながら，AMBA2 はキャッシュコヒーレンシをサポートしていないという問題がある．キャッシュコヒーレンシは AMBA4[15)] ではサポートされているが，AMBA4 は実装が困難であるという問題がある．また，ヘテロジニアスマルチコアでは，最適なキャッシュプロトコルがコアによって異なるため，バスの仕様とは切り離せた方が望ましい．そこで，我々は実装の容易な AMBA2 バスにキャッシュコヒーレンシのための制御バスを追加するという手法を採用する．キャッシュコヒーレンシのための制御バスは，後述のキャッシュシステムに適したバス構成を自動生成することができる．また，コヒーレンシ処理に必要なスヌープバスの本数も可変パラメータとした．

6.2 FabCache-Alpha

FabCache-Alpha は Alpha 21264 用のキャッシュシステムジェネレータである．ヘテロジニアスマルチコアは実行するアプリケーションの特性に加え，システムを構成する個々のコアにより最適なキャッシュ構成が異なる．しかしながら，MOESI, MESI をはじめとした様々なキャッシュシステムを手設計し，評価するのは非常に困難である．そのため，与えられたパラメータに基づき，マルチプロセッサ用キャッシュシステムを自動生成するツールを構築する．

また，FabScalar ではラインの境界を跨いだ命令フェッチができるように，命令キャッシュをインター

表 4 変更可能なキャッシュ構成
Table 4 Design parameters of cache system.

パラメータ	値
ウェイ数	1 ～ 32
セット数	1 ～ 262,144
ラインサイズ	4×64(Byte)
コヒーレンシプロトコル	MOESI, MESI, MOSI
インターリーブ	有り/無し
アクセスレイテンシ	タグメモリ，データメモリに依存
スヌープ専用パス数	0 ～ CPU コア数-1

リーブすることもできる．そのため，FabBus-Alpha と連係することにより，キャッシュメモリのインターリーブアクセスもできるようにする．表 4 に変更可能なパラメータを示す．

7. おわりに

本論文では，FabScalar でマルチコアプロセッサシステムを実現するために，FabScalar で生成されたスーパースカラコアへの Alpha 21264 命令セットの移植および FabCache-Alpha，FabBus-Alpha のフレームワークについて提案した．我々の行った検証の結果，移植した Alpha コアはいくつかの SPEC2000 ベンチマークが正常に動作することが明かとなった．

今後の研究として，我々は FabScalar で対応している全ての構成の Alpha コアを Verilog HDL ジェネレータで自動生成できるように，Verilog HDL ジェネレータの改造を行う．同時に，FabScalar で生成されたスーパースカラコアを搭載したチップの試作を行っていく予定である．また，本論文で示した FabCache-Alpha, FabBus-Alpha の実装を行っていく予定である．

謝辞 本研究の LSI 設計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，ルネサスエレクトロニクス株式会社，シノプシス株式会社の協力で行われたものである．

参 考 文 献

1) R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan and D. M. Tullsen. Single-ISA Heterogeneous Multi-core Architectures: The Potential for Processor Power Reduction. Int'l Symposium on Microarchitecture, Dec. 2003.
2) H. H. Najaf-abadi, N. K. Choudhary and E. Rotenberg. Core-Selectability in Chip Multiprocessors. 18th Int'l Conference on Parallel Architectures and Compilation Techniques, Sep. 2009.
3) P. Greenhalgh. Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7. ARM WHITE PAPER: http://www.arm.com/ja/files/downloads/big.LITTLE_Final.pdf.
4) N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi and E. Rotenberg. FabScalar: Composing Synthesizable RTL Designs of Arbitrary Cores within a Canonical Superscalar Template. Proceeding of the 38th IEEE/ACM Int'l Symposium on Computer Architecture (ISCA-38), pp. 11-22, June 2011.
5) N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel. Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline. Int 'l Conference on Dependable Systems and Networks (DSN), 2004.
6) L. Strozek, D. Brooks. Efficient Architectures through Application Clustering and Architectural Heterogeneity. Int 'l Conference on Compilers, Architecture, and Synthesis for Embedded Systems, 2006.
7) http://www.tensilica.com/products/xtensa-customizable.htm
8) S. Palacharla, N. P. Jouppi, J. E. Smith. Complexity-effective Superscalar Processors. Int 'l Symposium on Computer Architecture, June 1997.
9) S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, N. P. Jouppi. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. 42nd Int 'l Symposium on Microarchitecture, Dec. 2009.
10) Mark S. Papamarcos, Janak H. Patel. A low-overhead coherence solution for multiprocessors with private cache memories, In Proceedings of 25 Years ISCA: Retrospectives and Reprints'1998, pp.39 41, 1998.
11) P. Sweazey, A. J. Smith. A class of compatible cache consistency protocols and their support by the IEEE Futurebus, Proceedings of the 13th annual international symposium on Computer architecture Vol. 14 No. 2 pp. 414-423, 1986.
12) OpenCores: WISHBONE System-on-chip (SoC) Interconnection Architecture for Portable IP Cores, http://www.opencores.org/
13) IBM: CoreConnect Bus Archirecture, http://ibm.com/chips/products/coreconnect/
14) ARM: AMBA 仕様書 (Rev 2.0), http://www.arm.com/
15) ARM: AMBA AXI and ACE Protocol Specification, http://www.arm.com/