修士論文

題目

ヘテロジニアスマルチプロセッサに対応したコヒーレンシシステムの設計手法に関する研究

指導教員

近藤　利夫　教授

2016年

三重大学 大学院 工学研究科
博士前期課程 情報工学専攻
計算機アーキテクチャ研究室

三好　聖二 (413M526)

内容梗概

近年，高性能と低消費電力を両立する手法として，構成の異なる複数のプロセッサコアによって構成される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．従来のホモジニアスマルチコアプロセッサでは，構成の同じコアを搭載しているため，各アプリケーションに対して全て同じリソースが割り当てられ，性能の過不足により，無駄な消費電力が発生する．ヘテロジニアスマルチコアプロセッサでは，各アプリケーションに対して，最適な構成を持つコアを割り当てることによって，このような無駄な消費電力を抑えて，より高い電力効率を達成することができる．一方で，ヘテロジニアスマルチコアプロセッサは，構成の異なるコア，キャッシュとそれぞれに対応したバスシステムが必要であり，それぞれに別の設計・検証が必要になることから，従来手法と比較して設計コストが高く，ヘテロジニアスマルチコアプロセッサを研究・開発する上での大きな障害となっている．この問題を解決するために，自動設計技術が研究されており，著者らの研究グループでは，様々な構成のヘテロジニアスマルチコアプロセッサを自動設計するツールとして，FabHeteroを提案している．

FabHeteroは，並列度やキャッシュサイズのような要求仕様をパラメータとして与えることで，ヘテロジニアスマルチコアプロセッサを自動設計することを目指している．しかし，現状のFabHeteroはキャッシュコヒーレンシ機構が実装されていない．そのため，本研究ではFabHetero上にキャッシュコヒーレンシ機構を実装することで，より現実的な設計ツールを実現する．しかしながら，自動設計ツールにおいてキャッシュコヒーレンシをサポートするバスシステムの設計は，その構成や回路が要求仕様や設計条件等により多様に変化するため，自動化のための大きな課題の一つとなる．この問題の一つとしては，キャッシュやコアがそれぞれの仕様に従って内部で完結するのに対して，バスシステムはそれらを相互接続するために，それぞれのキャッシュやコアの異なる仕様に起因するプロトコル・バス幅の違いなどの問題を解決する必要があることが挙げられる．更に，ただ１つの仕様の変更に伴ってバスシステム全体の設計が変わることから，接続するコアが増えたり，仕様の可変要素が増えることに伴って，解決する必要がある要素が指数関数的に増えていく．このため，全ての組み合わせに対応して設計データを作る，といった手法は事実上

不可能であり，どの仕様の違いを設計のどの部分で解決するか，という課題を解決する必要がある．本研究では，これらの問題を解決するために，ヘテロジニアスマルチコアプロセッサ上でのキャッシュコヒーレンシをサポートするバスシステムの設計のためのデザインフレームワークを提案する．具体的には，上述の問題に対して，設計のどの部分を共通化するか，仕様の違いを設計のどの部分で吸収するかのベースとなる構成をフレームワークとして与えることで，組み合わせ問題の分割，設計データの共有化率の向上を図り，ヘテロジニアスマルチコアプロセッサの設計，自動設計ツールの開発の効率化を目指すものである．更に，本論文では提案フレームワークの有効性を示すために，提案フレームワークを含むバスシステムの自動設計ツールを実装し，その動作検証を行った．

その結果，ソースコードの重複量や修正する必要のあるコード量について，FabHetero を用いて構成した典型的な 2 階層のキャッシュシステムで約 22 %削減できた．又，SPEC2000INT ベンチマークを用いて，提案したフレームワークをベースに，FabHetero 上に実装したキャッシュコヒーレンシ機構の動作検証を行い，正しく動作していることを確認した．

Abstract

Single-ISA heterogeneous multi-core processor architecture which is composed of diverse cores, cache systems, and share bus system is promising technique to achieve higher energy efficiency. However, because a designer must design and verify each of cores, caches and shared bus system of heterogeneous multi-core processor (HMP), an effort of implementing HMP is multiplied by the number of kinds of each of component. This limits an amount of microarchitectural diversity of commercial or research products which can be practically implemented due to a limitation of a resource and time to implement.

In order to reduce the efforts, many researches have focused on design automation technology. Then, we proposes FabHetero which is an automatic generation toolset of HMP. However, the current FabHetero does not have a cache coherency mechanism. Therefore, this study implements a cache coherency logic on FabHetero for developing it. Nevertheless, generating of shared bus interconnection with supporting cache coherency mechanism of HMP is one of the major challenge to implement an automatic generation system of HMP. As a reason for this, a bus system of HMP have to resolve differences of connecting elements caused by micro-architectural differences of each core and cache. In addition, a number of this differences increases exponentially along with a number of connecting elements or modification of specification. And then, a number of combination is too enormous to support all of cases. This is a barrier to develop and reserach of HMP and its design tool.

In this study, to the barrier, this paper proposes a framework which shows a implementation strategy or policy of cache coherent system and bus system under HMP environment for making efficiency of developing an automation design tool and implementing a HMP.

As the first step to develop and verify this framework, this papr also implemented an automatic generating system of snoop-based interconnection using FabHetero ported to ARM AMBA4 and ACE protocol.

In the result, this study succeeds to reduce about 22% redundant HDL codes of HMP generated by FabHetero, and therefore, this study can re-

duce implementation effort about 2 to 4 man-month out of developing entire multi-core processor. In addition, this study verify the implemented system correctly works on SPEC2000INT benchmarks.

目 次

1 はじめに **1**

2 背景 **4**
2.1 ホモジニアスマルチコアプロセッサとヘテロジニアスマルチコアプロセッサ 4
2.2 ヘテロジニアスマルチコアの問題点 5
2.3 FabHetero：ヘテロジニアスマルチコアプロセッサの自動生成ツール 6

3 キャッシュコヒーレンシとその実装手法 **7**

4 ヘテロジニアスマルチコアプロセッサにおけるキャッシュコヒーレンシの実装と問題点 **12**
4.1 ヘテロジニアスマルチコアにおけるバスシステム 12
4.2 ヘテロジニアスマルチコアプロセッサにおけるキャッシュコヒーレンシ 14

5 関連研究 **14**

6 フレームワークの提案 **15**
6.1 キャッシュのラッパー 16
6.2 キャッシュ内インターフェース 17
6.3 バスインターフェース 18

7 ケーススタディ **19**
7.1 ケース 1 20
7.2 ケース 2 22
7.3 ケース 3 24

8 バスシステム自動生成ツールの実装 **25**

9 結論 **26**

10 謝辞 **27**

参考文献 **28**

図 目 次

2.1 Homogeneous multi-core vs. Heterogeneous multi-core. . . 4
2.2 FabHetero. 6
3.3 An Example of cache coherency: Step 0. 8
3.4 An Example of cache coherency: Step 1. 9
3.5 An Example of cache coherency: Step 2. 9
3.6 Outline of Snooping bus. 10
3.7 Directory based Protocol. 11
3.8 An Example of Conceivable Heterogeneous multi-core System. 12
6.9 Cache Wrapper. 17
6.10 Inner-cache Interface. 18
6.11 Bus-side Coherent Interface. 19
7.12 Case 1. 21
7.13 Case 2. 22
7.14 Case 3. 24
8.15 An Example of Wave of HDL Simulation. 26

1　はじめに

近年，高性能と低消費電力を両立する手法として，構成の異なる複数のプロセッサコアによって構成される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．従来のホモジニアスマルチコアプロセッサでは、構成の同じコアを搭載しているため、各アプリケーションに対して全て同じリソースが割り当てられ、性能の過不足により、無駄な消費電力が発生する。ヘテロジニアスマルチコアプロセッサでは、各アプリケーションに対して、最適な構成を持つコアを割り当てることによって、このような無駄な消費電力を抑えて、より高い電力効率を達成することができる。一方で、ヘテロジニアスマルチコアプロセッサは、構成の異なるコア、キャッシュとそれぞれに対応したバスシステムが必要であり、それぞれに別の設計・検証が必要になることから、従来手法と比較して設計コストが高く、ヘテロジニアスマルチコアプロセッサを研究・開発する上での大きな障害となっている。この問題を解決するために、自動設計技術が研究されており、著者らの研究グループでは、様々な構成のヘテロジニアスマルチコアプロセッサを自動設計するツールとして、FabHeteroを提案している。

FabHeteroは、並列度やキャッシュサイズのような要求仕様をパラメー

タとして与えることで、ヘテロジニアスマルチコアプロセッサを自動設計することを目指している。しかし、現状の FabHetero はキャッシュコヒーレンシ機構が実装されていない。そのため、本研究では FabHetero 上にキャッシュコヒーレンシ機構を実装することで、より現実的な設計ツールを実現する。しかしながら、自動設計ツールにおいてキャッシュコヒーレンシをサポートするバスシステムの設計は、その構成や回路が要求仕様や設計条件等により多様に変化するため、自動化のための大きな課題の一つとなる。この問題の一つとしては、キャッシュやコアがそれぞれの仕様に従って内部で完結するのに対して、バスシステムはそれらを相互接続するために、それぞれのキャッシュやコアの異なる仕様に起因するプロトコル・バス幅の違いなどの問題を解決する必要があることが挙げられる。更に、ただ１つの仕様の変更に伴ってバスシステム全体の設計が変わることから、接続するコアが増えたり、仕様の可変要素が増えることに伴って、解決する必要がある要素が指数関数的に増えていく。このため、全ての組み合わせに対応して設計データを作る、といった手法は事実上不可能であり、どの仕様の違いを設計のどの部分で解決するか、という課題を解決する必要がある。本研究では、これらの問題を解決するために、ヘテロジニアスマルチコアプロセッサ上でのキャッシュコヒーレンシをサ

ポートするバスシステムの設計のためのデザインフレームワークを提案する。具体的には、上述の問題に対して、設計のどの部分を共通化するか、仕様の違いを設計のどの部分で吸収するかのベースとなる構成をフレームワークとして与えることで、組み合わせ問題の分割、設計データの共有化率の向上を図り、ヘテロジニアスマルチコアプロセッサの設計、自動設計ツールの開発の効率化を目指すものである。更に、本論文では提案フレームワークの有効性を示すために、提案フレームワークを含むバスシステムの自動設計ツールを実装し、その動作検証を行った。

2 背景

2.1 ホモジニアスマルチコアプロセッサとヘテロジニアスマルチコアプロセッサ

近年，より高い電力効率を達成する目的で，同様の構造を持つ CPU コアから構成されるホモジニアスマルチコア（図 2.1 左）から，異なる構成の CPU コアから構成されるヘテロジニアスマルチコア（図 2.1 右）へのパラダイムシフトが発生している [1]〜[4].

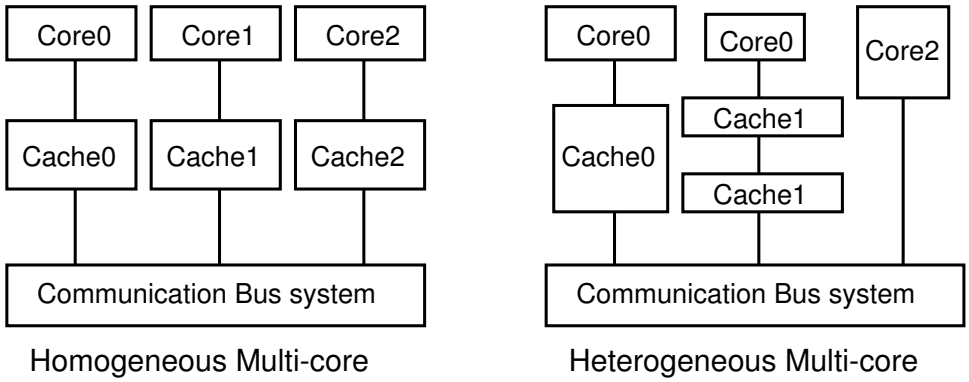


図 2.1: Homogeneous multi-core vs. Heterogeneous multi-core.

従来のホモジニアスマルチコアは，同様の構造を持つコア群から構成される．そのため，各アプリケーションに対して一様なハードウェアリソースの割当しかできず，リソースの過不足が生じ，これが無駄な消費電力を発生させている．

これに対して，ヘテロジニアスマルチコアは，様々な構造のコア群か

ら構成される．これによって，各アプリケーションに対して，最適な構成（ハードウェアリソース）を持つコアを割り当てることによって，上記した無駄な消費電力を抑えることが出来，ホモジニアスマルチコアに比べて高い電力効率を達成することが出来る．

2.2 ヘテロジニアスマルチコアの問題点

ヘテロジニアスマルチコアの問題として，従来のホモジニアスマルチコアに比べて設計コストが高いことが挙げられる．これは，ホモジニアスマルチコアは図 2.1 左のように，同じ構成のコア，キャッシュから構成されているために，一組のコアとキャッシュを設計すればそれを複製して並べることで，マルチコアの設計が可能なのに対して，ヘテロジニアスマルチコアでは，図 2.1 右のように，それぞれのコア，キャッシュの構成が異なるために，それぞれのコアとキャッシュを一から設計する必要があることに起因する．このように，従来のホモジニアスマルチコアに比べて，設計コストが幾倍にも大きくなることが，ヘテロジニアスマルチコアを研究・開発する上での障害となっている．

2.3 FabHetero：ヘテロジニアスマルチコアプロセッサの自動生成ツール

著者らの研究グループはヘテロジニアスマルチコアプロセッサにおける設計コストの問題を解決するために，ヘテロジニアスマルチコアプロセッサの自動生成ツール FabHetero を提案している．[12] FabHetero は，コア部を自動生成する FabScalar[9][11]，キャッシュシステムを自動生成する FabCache[13]，バスシステムを自動生成する FabBus[14][15] から構成され，要求する仕様をパラメータとして FabHetero に与えることで，FabHetero は自動的にヘテロジニアスマルチコアプロセッサの論理合成可能な RTL コードを生成する．

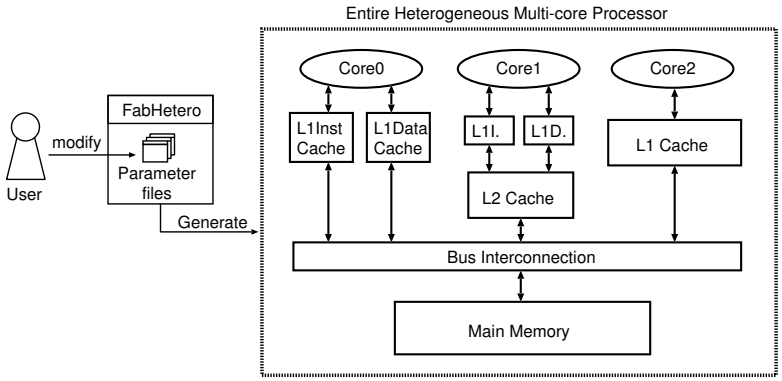


図 2.2: FabHetero.

しかし，現在の FabHetero ではキャッシュコヒーレント機構が実装されていないため，マルチコアプロセッサ上でマルチコアプログラムを動作

させる検証が不可能である．

そこで，本研究では，FabHeteroのような自動生成ツールに適したキャッシュコヒーレンシ実装のためのデザインフレームワークを提案する．又，提案フレームワークをベースとしたキャッシュコヒーレンシ機構をFabHetero上に実装し，その動作検証を行う．次章では，通常のマルチコアプロセッサにおけるキャッシュコヒーレンシとその実装手法について述べる．

3　キャッシュコヒーレンシとその実装手法

キャッシュコヒーレンシとは，マルチコアプロセッサによって実行されるアプリケーションにおいて，コア間でデータの更新情報をやりとりすることによって，コア間でのデータの一貫性を保つ機能を指し，マルチコアアプリケーションを実行する際には必須の機能となるため，近年の商用のマルチコアプロセッサのほとんど全てに搭載されている．

図3.3から図3.5にマルチコアプロセッサにおける，キャッシュコヒーレンシの例を示す．

図3.3では，コア０とコア１がそれぞれ個別のキャッシュを持ち，各キャッシュは共有メモリに繋がっており，メモリ上にデータ'A'が存在する．又，図3.3右はコア０，コア１の実行するプログラムを示し，コア

０が先にデータ’A’ をロードし，更新する．続いて，コア１が同様にデータ’A’ をロードし，更新する．この一連の動作では，対象となるデータ’A’ は同一のものを扱うため，コア０とコア１にはデータの依存が存在し，図 3.3 の右側の波線矢印のように，コア０でのデータ’A’ の更新と，コア１でのデータ’A’ のロードには依存関係が存在する．

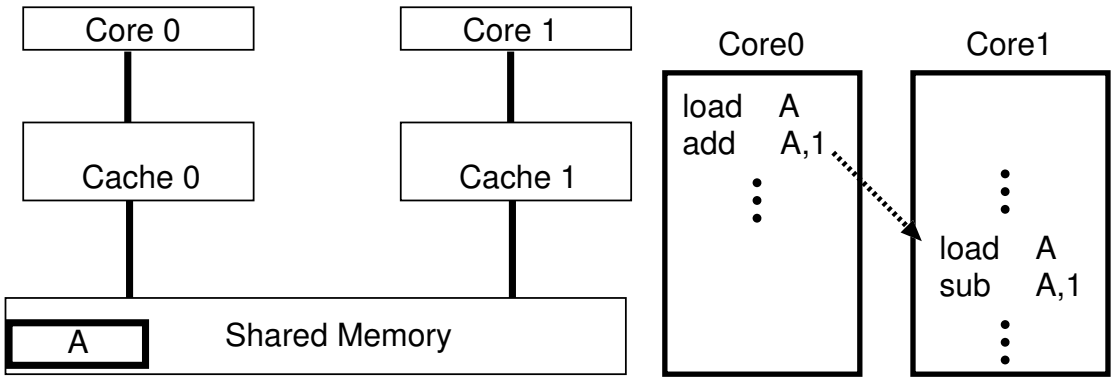


図 3.3: An Example of cache coherency: Step 0.

まず，先行してコア０によってデータ’A’ がロードされ，更新される．その状況を図 3.4 で示す．コア１に繋がるキャッシュ１にはデータ’A’ は存在せず，コア０に繋がるキャッシュ０にのみ，更新されたデータ’A+1’ が存在する．

次に，コア１によってデータ’A’ をロードする必要がある．しかし，ここでコア１がロードする必要のあるデータは，コア０のキャッシュ０上に存在する一方で，キャッシュ０上の新しいデータは共有メモリに更新され

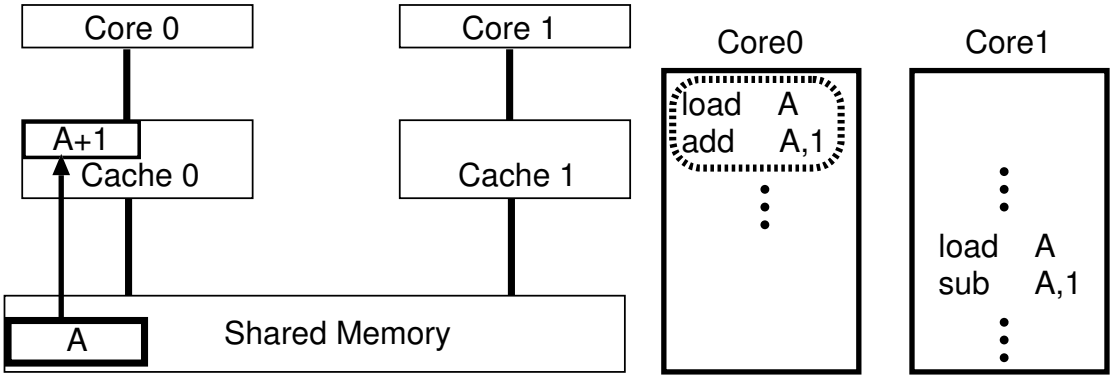


図 3.4: An Example of cache coherency: Step 1.

ていないため，内容が異なる．このため，キャッシュ 0 はキャッシュ 1 に更新内容を通知する必要がある．

この更新データの通知を通して，キャッシュ間でのデータの整合性を保つ機能をキャッシュコヒーレンシと呼ぶ.

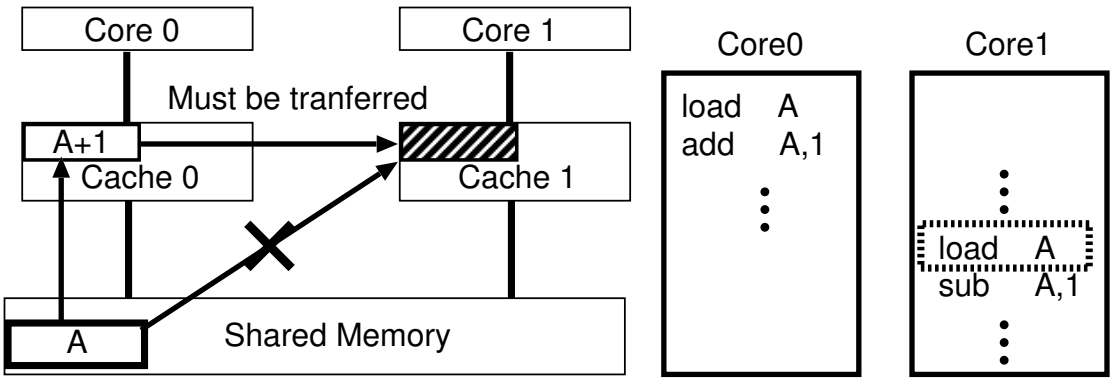


図 3.5: An Example of cache coherency: Step 2.

このようなキャッシュコヒーレンシを実際のマルチコアプロセッサ上に実装する際には，スヌーピング・ディレクトリベースの 2 つのプロトコ

ルが広く用いられるが,

- スヌーピングプロトコルスヌーピングプロトコルとはキャッシュコヒーレンシを実装する際によく使われる手法の一つである.

  図 3.6 にスヌーピングプロトコルを実装する際の例を示す．スヌーピングプロトコルでは，各コアによるデータの更新情報をスヌーピングバスと呼ばれる特別なバスを通して，他の全てのコアに伝達することで，キャッシュコヒーレンシを保つ．実装が非常にシンプルで，性能的にも優れているが，コアの数が増すにつれてスヌーピングバスが輻輳を起こし，劇的に性能が低下することから，概して小規模なマルチコアのバスシステムにおいて採用される．

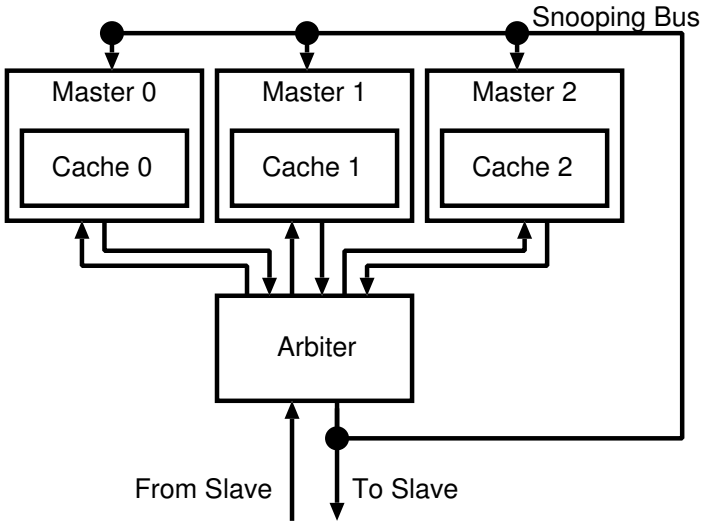


図 3.6: Outline of Snooping bus.

- ディレクトリベースプロトコルディレクトリベースプロトコルとはキャッシュコヒーレンシを実装する際に広く用いられる手法の一つである．

  図 3.7 にディレクトリベースプロトコルの実装例を示す．ディレクトリベースプロトコルでは，各コアにディレクトリと呼ばれる表を備える．各ディレクトリは，キャッシュ上のデータが，他のどのコアによって保持されているかを記録したものであり，データの更新情報を伝達する際に，どのコアに伝達すればよいかを示す．必ずディレクトリを参照する必要があるため，スヌーピングプロトコルに比べて最高性能に劣るが，コア数が増加した際にも性能が低下しにくいという特徴を持つ．

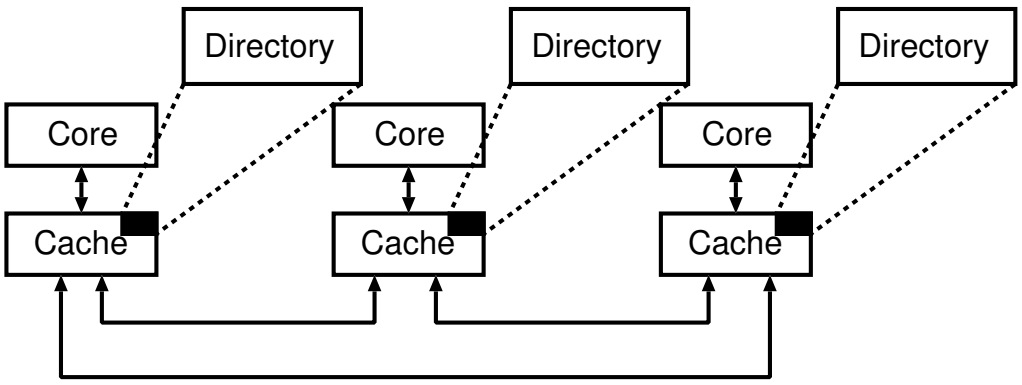


図 3.7: Directory based Protocol.

本研究では，コア数の多い大型のプロセッサを想定しないこと，実装す

る際により簡易であることから，スヌーピングプロトコルを実装するシステムにおいて採用する．

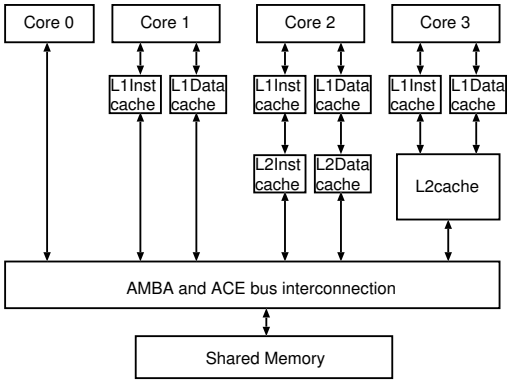


図 3.8: An Example of Conceivable Heterogeneous multi-core System.

4　ヘテロジニアスマルチコアプロセッサにおけるキャッシュコヒーレンシの実装と問題点

キャッシュコヒーレンシのための機構は，キャッシュシステム，バスシステム双方に関連が深く，実装する際にも両者との連携を保ちながら実装する必要がある．

4.1　ヘテロジニアスマルチコアにおけるバスシステム

バスシステムとは，プロセッサ間，もしくはプロセッサとメモリ間を接続する信号線の束の総称であり，図 2.1 においては，Communication Bus system と記述された部位に相当する．近年では，プロセッサ単体での性

能が頭打ちになる中で，バスシステムの性能がマルチコアプロセッサ全体の性能を左右する重要な要素になりつつあり，バスシステムの設計が重要度を増している．

従来のホモジニアスマルチコアプロセッサにおいて，接続される各コア・キャッシュは，同一の構造・仕様のため，それらを相互接続するバスシステムは一様な（対称的な）構造で問題なかった．

対して，ヘテロジニアスマルチコアプロセッサでは，各コア・キャッシュの構成が異なるために，最適なバスシステムを設計するのは困難で，それぞれの仕様の違いを吸収する必要がある．各コア・キャッシュの構成の組み合わせも膨大なものとなるため，バスシステムの設計・検証に要する時間も膨大なものとなる．そのため，ヘテロジニアスマルチコアプロセッサを設計する際の障害の１つとなっている．

バスシステムには，大きくわけて，メモリアクセスとキャッシュコヒーレンシ管理の２つの役割があり，中でもキャッシュコヒーレンシに関する機構は，バスシステムの実装を難しくする大きな要因である．

4.2 ヘテロジニアスマルチコアプロセッサにおけるキャッシュコヒーレンシ

図3.8のように，ヘテロジニアスマルチコアプロセッサにおいて，各コアに付属するキャッシュ構造はそれぞれ異なる．そのため，キャッシュのコヒーレンシプロトコルや，キャッシュの書き換えポリシーがキャッシュ毎に異なる状況が発生する．キャッシュコヒーレンシ機構の設計は，各キャッシュの仕様に大きく左右され，ヘテロジニアスマルチコアプロセッサでは各キャッシュの仕様の組み合わせが膨大となることが，ヘテロジニアスマルチコア上へのキャッシュコヒーレンシ機構の実装を難しくしており，バスシステムを実装する際の最大の障害の１つとなっている．

5 関連研究

本研究はヘテロジニアスマルチコアプロセッサを対象とした，キャッシュコヒーレント機構の実装についてであり，キャッシュシステムとバスシステムについて関係がある．そこで，この２つの領域について関連研究を挙げる．

Sungjoo Yooらの研究では[5]，ラッパーと呼ばれるアーキテクチャを提案している．これは，ヘテロジニアスマルチコアプロセッサにおいて，

キャッシュとバスの間に特別なモジュールを追加することで，各コア・キャッシュシステムのキャッシュコヒーレントに関わる仕様の違いを吸収・隠蔽する．D. Lyonnard らの研究では [6]，このラッパーアーキテクチャを自動生成するデザインフローについて述べている．これらの研究で提案されているラッパーアーキテクチャは，各バスプロトコルに対応するモジュールと，各プロセッサの仕様に対応するモジュール，及びそれらを相互に選択的に接続するバスから構成される．しかし，これらの研究はキャッシュコヒーレンシについては触れておらず，キャッシュシステム内の設計やバスシステムの設計指針にも言及していない．

本研究では，キャッシュシステム内外を問わず，ヘテロジニアスマルチコアプロセッサにおけるキャッシュコヒーレント機構全般についてのフレームワークを述べる．又，著者らの研究グループが提案している自動生成ツール FabHetero をベースに提案するフレームワークを元にバスの自動生成ツールを実装する．

6　フレームワークの提案

4 章で述べた通り，ヘテロジニアスマルチコアプロセッサにおける各キャッシュに合わせてバスシステムを設計する事は，手での設計でも自動

設計であっても非常に難しく，これがヘテロジニアスマルチコアプロセッサを研究・開発する上での問題となっている．本研究では，この問題に対してバスシステムのデザインフレームワークを提案する．フレームワークの基本的なコンセプトとして，バスシステムの設計時の問題を分割し，対応する実装部をモジュール化して再利用が可能にすることで，ヘテロジニアスマルチコアプロセッサの開発効率の向上を目指す．

提案フレームワークは以下の 2 つによって構成され，

- キャッシュのラッパー
- キャッシュ内インターフェース
- バスインターフェース

各要素については以下の各節で述べる．

6.1 キャッシュのラッパー

通常，バスシステムの設計はバスの要求仕様とキャッシュの仕様に左右されるので，これらの間に相違があると，設計者が手作業でつなぎ合わせる必要がある．図のように，ラッパーはキャッシュをカプセル化することで，キャッシュの構造，仕様の違いをバス側から隠す．これによって，

バス側の問題とキャッシュ側の設計を分離することで，設計を単純化することができる．

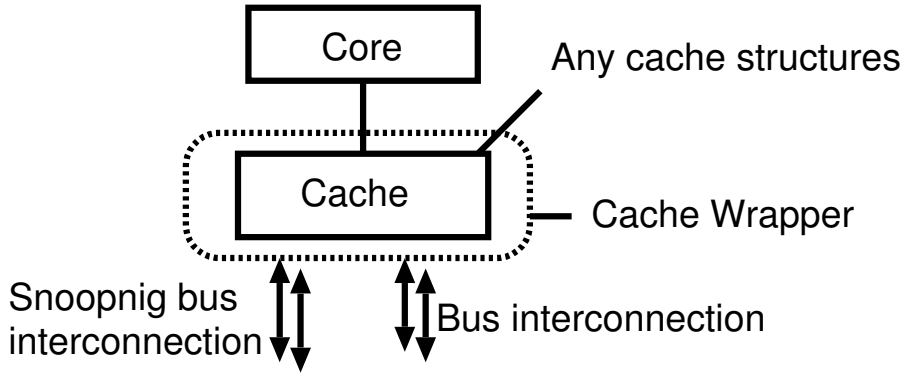


図 6.9: Cache Wrapper.

6.2 キャッシュ内インターフェース

本節のインターフェースでは，キャッシュ構造内部についてのキャッシュコヒーレントに関わるバスシステムを扱う．このインターフェースは以下の 2 点から構成される．

1. 統合インターフェース:キャッシュシステムに付属するインターフェースで，図 6.9 の右側の部分である．ここでは，キャッシュシステムに代わってバス側のコヒーレントバスと接続され，バス幅やプロトコルの違いを吸収する．又，バスシステムがビジー状態の際に，リクエストを保持するバッファの役割も兼ねる．

2. レベルインターフェース：各レベルのキャッシュに付属するインターフェースである．各レベルのキャッシュをカプセル化することで，各レベルのキャッシュ間の違いを吸収する．又，上述の統合インターフェースと接続して，メモリバスからコヒーレントバスを独立させることで，実装・変更を容易にする．

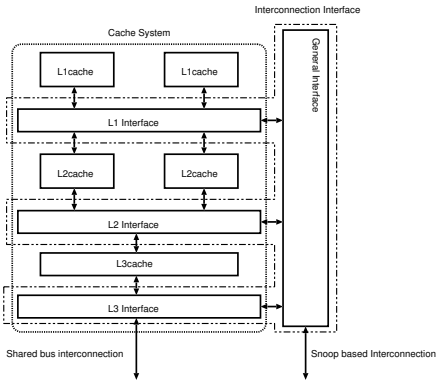


図 6.10: Inner-cache Interface.

6.3 バスインターフェース

バスインターフェースでは，ラッパーによって分割されたうちバス側に対応し，図のように，通常のメモリバスから独立してコヒーレントバスを扱う．独立させることによって，コヒーレントバスの仕様を統一することができ，実装・変更が容易になる．

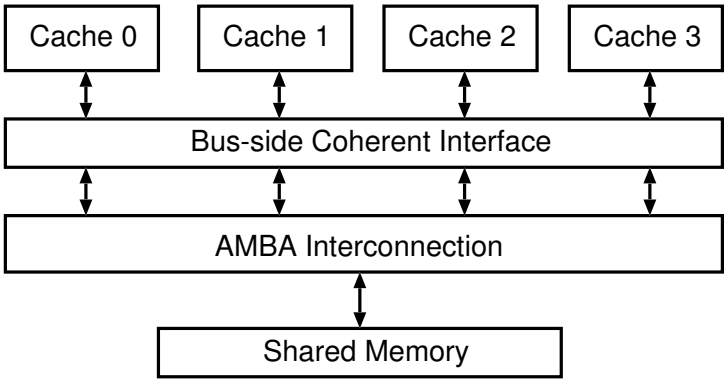


図 6.11: Bus-side Coherent Interface.

7　ケーススタディ

ヘテロジニアスマルチコアプロセッサにおいては，各キャッシュシステムは様々な違いを持つ．通常，違う仕様のコア，キャッシュについては最適なバス設計が異なり，相違点の数に応じてバスシステムの組み合わせは指数関数式に増えていく．本章では，いくつかの例を元に，ヘテロジニアスマルチコアプロセッサにおけるバスシステムの組み合わせの問題と，提案するフレームワークがどのように関わるかを論じる．

簡単化のため，各キャッシュシステムにおける相違点を以下の点に限定する

1. キャッシュの書き戻しポリシー：ライトバック（WB）もしくはライトスルー（WT）

2. キャッシュのコヒーレンシプロトコル：アップデート方式（WU）か，インバリデイト方式（WI)

3. 自己書き換えプログラム：自己書き換えをサポートする（SM）か，サポートしないか（NSM)

4. 包摂的／排他的キャッシュ：包摂的キャッシュ（IC) であるか，排他的キャッシュか（EC)

7.1 ケース 1

このケースでは図 7.12 のように，1 つのコアに L1 命令キャッシュと L1 データキャッシュをそれぞれ備える．プロセッサの実装では最も単純な構成であるが，先述した要素では 1〜3 が関わり，各要素は図のように (1)L1 キャッシュ全体の書き戻しポリシー，(2) キャッシュ全体のコヒーレンシプロトコルの方式，(3)L1 命令キャッシュにおいて自己書き換えプログラムをサポートするか，という 3 点で仕様の組み合わせ数は 2^3 で 8 通りである．

このケース単体ではマルチコアではないが，この構成のプロセッサはマルチコアプロセッサを設計する際の要素として採用され得るので，キャッシュコヒーレンシを考慮する必要がある．

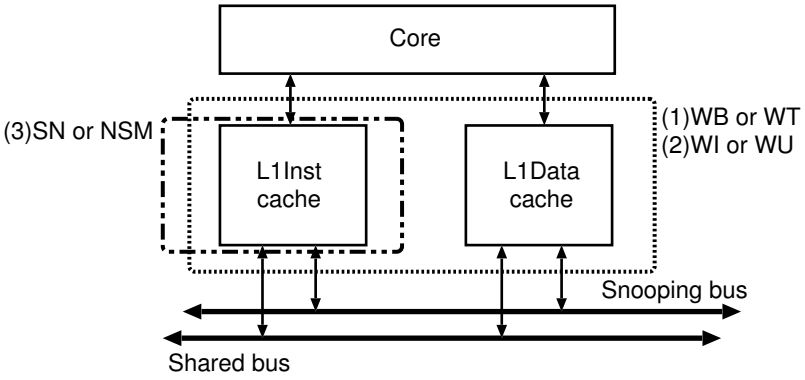


図 7.12: Case 1.

このケースのプロセッサの内部の設計では，6.1 によってカプセル化されることで，他のマルチコアプロセッサへの移植が簡単になる．又，プロセッサ内部に追加の実装を行う際にも，6.2 のレベルインターフェースによって L1 キャッシュより上の要素が下の要素から隠蔽されるため，インターフェースに合わせて設計を行うことで既存の設計データへ考慮した設計が必要でなくなる．

加えて，6.1 によってキャッシュとバスの仕様を分離し，6.3 によってコヒーレントバスの仕様を統一することによって，マルチコアプロセッサ全体でのキャッシュコヒーレントの設計を易化する．又，コヒーレントバスの仕様が独立させられることから，設計資源として他のマルチコアプロセッサの設計に使い回すことが可能となる．

これらの要素によって，既存の設計に配慮した設計が必要でなくなる

とともに，設計資源としての使い回しが簡単になるため，全体の設計コストを低減させることができる．

7.2 ケース2

図7.13のように，ケース1のコアに，統合型のL2キャッシュを加えたものである．これもまた，最も単純な構成の一つであるが，先述した1〜4の点全てについて考慮する必要があり，各点は図のように(1)L1キャッシュ全体の書き戻しポリシー，(2)キャッシュ全体のコヒーレンシプロトコルの方式，(3)L1命令キャッシュにおいて自己書き換えプログラムをサポートするか，(4)L2キャッシュの書き戻しポリシー，(5)L2キャッシュが排他的キャッシュであるか包摂的キャッシュか，という点で関わり組み合わせ数は2^5から32通りとなる．

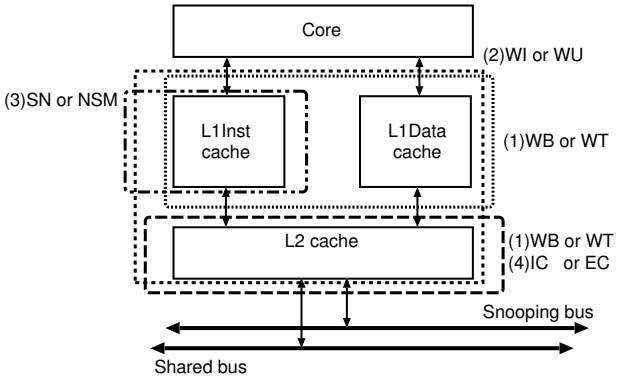


図 7.13: Case 2.

組み合わせ数の増加に加えて，キャッシュ構造の変化に伴って内部のバ

ス構造も変化しているが，セクション 7.1 で述べたように，6.1 によってカプセル化されているためこのプロセッサの外側への影響をなくすことが出来る．

又，このケースはケース 1 に統合型の L2 キャッシュを付加したものであることから，ケース 1 の設計資源を使い回すことが考えられるが，6.2 のレベルインターフェースによって L1 キャッシュ以上の要素が隠蔽されているため，設計者はケース 1 の設計資源に L2 キャッシュを追加するだけでケース 2 を設計することが可能になる．

これによって上述した (1)〜(4) について考慮する必要がなくなる．(5) については，L1 キャッシュの動作にも影響を与えることから完全に排除することが不可能だが，全体として設計者の考慮する必要がある組み合わせ数を 32 通りから 2 通りまで削減することが出来る．

又，L3 キャッシュを追加するなど，キャッシュ階層を増やす・複雑化する際にも，これらの隠蔽や，カプセル化による設計資源の使い回しが可能なため，設計コストの増加を防ぐことができる．

7.3 ケース 3

このケースでは，図 7.14 のように 2 つのコアを含むマルチコア構成で，それぞれのコアはケース 1 とケース 2 のものである．これらを接続する場合には，バスの組み合わせ数は単純に乗算となり $32 * 8$ の 256 通りとなる．

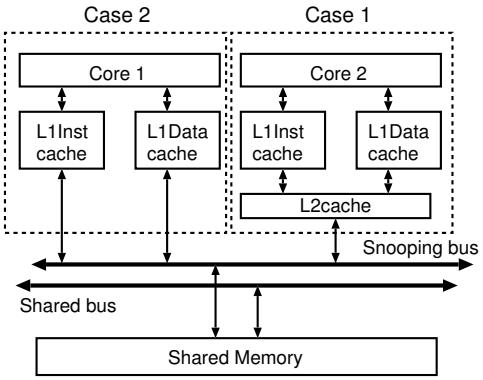


図 7.14: Case 3.

セクション 7.1 とセクション 7.2 で述べたように，キャッシュ内部の問題は解決している．特に，ケース 2 の部分については，ケース 1 の設計資源の使い回しによってほとんどの設計コストを排除できるため，全体で必要となる設計は基礎となるケース 1 の設計，及び共有されるバス側の設計となる．

これについても，6.3 によって独立な統一された仕様をもつバスシステムを提供するので，バス側の実装はマルチコア用のバスを一度設計すれば

よい．それぞれのコアとバスの接続についても 6.2 の統合インターフェースによって隠蔽されるため，設計者の配慮を必要としない．

又，3 コア，4 コアとコア数を増やす際には，共有バスが既に設計されていることから，より少ない設計コストでマルチコアプロセッサを設計可能となる．

以上のように提案するフレームワークを用いることで，ほとんどの部分での設計データの使い回しを可能にすることで，新規でマルチコアプロセッサを設計する際の設計負担を低減することができる．

8　バスシステム自動生成ツールの実装

本研究では，後術する FabHetero を元にして，提案したフレームワークを採用したバスシステムの自動生成ツールを実装した．実装したツールは図のようにユーザーがパラメータを指定することで，様々な構成の論理合成可能なバスシステムの RTL（Register Transfer Level）コードを与える．実装にあたっては AMBA4.0 及び ACE プロトコルを採用した．

生成された設計データの動作検証には Cadence 社の NC-Verilog を利用し，動作させるプログラムには SPEC2000INT ベンチマークを使用した．図に RTL シミュレーションでの波形図を示す．動作確認でのプロセッサ

構成はケース１をベースにコヒーレントリクエストを発行する試験用モジュールを備え付けたもので，L1 キャッシュはライトバックおよびMOESI プロトコル，自己改造プログラムはサポートしないものとする．

実行結果及び出力された波形図より正しく動作できていることが確認できたことから，FabHetero がキャッシュコヒーレント機構を含むヘテロジニアスマルチコアプロセッサを自動生成できることを確認できた．ユーザーは要求デザインが FabHetero のスーパーセットコードの含まれている限りはほとんど設計努力を払わずに要求したデザインを得ることができる．

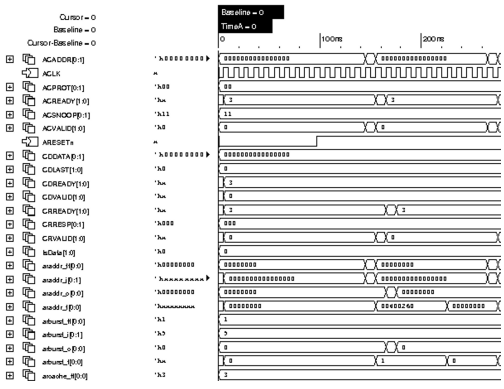


図 8.15: An Example of Wave of HDL Simulation.

9　結論

本研究では，ヘテロジニアスマルチコアプロセッサにおけるキャッシュコヒーレント機構を実装するためのフレームワークを提案した．又，ヘテロ

ジニアスマルチコアプロセッサの自動生成ツール FabHetero を AMBA4.0 & ACE プロトコルに移植し，提案したフレームワークを実装したバスシステムの自動生成ツールを実装・動作確認を行った．その結果，自動生成ツールによって生成された設計データが正しく動作することが確認できた．又，提案したフレームワークによって FabCache，及び FabBus の設計データのうち 22%の共有モジュール化に成功したことから，全体として 2〜4 人月ほどの設計コストを削減できると見積もることが出来る．今後は，実装したバスシステムの自動生成ツールの改良，及び消費電力・ハードウェア規模の評価を行う．

10　謝辞

本研究を行うにあたり，多数の助言を頂きました近藤利夫教授，深澤研究員，並びにご指導を頂きました佐々木敬泰助教に深く感謝いたします．また，計算機アーキテクチャ研究室院生・学生のメンバーには常に刺激的な議論を頂き，精神的にも支えられました．また，本研究は日本学術振興会の科学研究費補助金の 24700047 及び 15K00074，Synopsys 社 CAD ツールによる東京大学 VDEC，Rohm 社 VDEC の支援により実施されたことを並びに感謝します．

参考文献

[1] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, K. I. Farkas. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. *31st International Symposium on Computer Architecture (ISCA31)*, pp. 64-75, June 2004.

[2] H. H. Najaf-abadi, E. Rotenberg. Configurational Workload Characterization. *International Symposium on Performance Analysis of Systems and Software 2008 (ISPASS-2008)*, pp. 147-156, April 2008.

[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] 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.

[5] S. Yoo, G. Nicolescu, D. Lyonnard, A. Baghdadi, A. A. Jerraya. A Generic Wrapper Architecture for Multi-Processor SoC Cosimula-

tion and Design, *Proceedings of the Ninth International Workshop on Hardware/Software Codesign Codes/CASHE*, 2001, pp. 195-200

[6] D. Lyonnard, S. Yoo, A. Badhdadi, A.A. Jerraya. Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip, *Proceedings of the 38th Design Automation Conference (DAC 01)*, ACM Press, 2001, pp. 518-523.

[7] F. Gharsalli, D. Lyonnard, S. Meftali, F. Rousseau, A.A. Jerraya. Unifying Memory and Processor Wrapper Architecture in Multiprocessor SoC Design, *The 15th International Symposium on System Synthesis*, Feb. 2002, pp.26-31

[8] 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. *38th IEEE/ACM International Symposium on Computer Architecture (ISCA-38)*, pp. 11-22, June 2011.

Rationale for a 3D Heterogeneous Multi-core Processor. *Proceedings of the 31st IEEE International Conference on Computer Design (ICCD-31)*, pp. 154-168, Oct. 2013.

[9] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. FabScalar: Automating Superscalar Core Design. *Micro, IEEE (Volume:32 , Issue: 3)*, pp. 48-59, June 2012.

[10] 中林智之, 佐々木敬泰, Eric Rotenberg, 大野和彦, 近藤利夫, 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, SACSIS2012.

[11] N. K. Choudhary, B. H. Dwiel, E. Rotenberg. A physical design study of fabscalar-generated superscalar cores. *VLSI and System-on-Chip (VLSI-SoC), 2012 IEEE/IFIP 20th International Conference on* , pp. 165-170, Oct. 2012.

[12] T. Nakabayashi, T. Sasaki, E. Rotenberg, K. Ohno and T. Kondo. Research for Transporting Alpha ISA and Adopting Multi-processor to FabScalar. *Symposium on Advanced Computing Systems and*

Infrastructures 2012 (SACSIS2012), pp. 374-381, May 2012. (in Japanese)

[13] T. Okamoto, T. Nakabayashi, T. Sasaki, T. Kondo. FabCache: Cache Design Automation for Heterogeneous Multi-core Processors. *Proceedins of the 1st International Symposium on Computing and Networking*, pp.602-606, Dec. 2013.

[14] 瀬戸　勇介，佐々木　敬泰，大野　和彦，近藤　利夫, ヘテロジニアスマルチプロセッサ環境を対象としたAMBA バスフレームワークの設計と評価,SWOPP2012.

[15] Y. Seto, T. Nakabayashi, T. Sasaki, and T. Kondo. FabBus: A Bus Framework for Heterogeneous Multi-core processor. *28th International Technical Conferench on Circuits/Systems, Computers and Communications (ITC-CSCC2013)*, pp. 254-257, July 2013.