卒業論文

題目

ヘテロジニアスマルチコアプロセッサ用のオンチップ同時多対多通信及びオフチップ通信のためのバスシステム自動生成に関する研究

指導教員

佐々木 敬泰　助教

2013年

三重大学 工学部 情報工学科
計算機アーキテクチャ研究室

川島 弘晃 (410814)

内容梗概

近年，ホモジニアスマルチコアプロセッサからヘテロジニアスマルチコアプロセッサの研究へと注目が集まっている．ヘテロジニアスマルチコアプロセッサは特徴の異なるアプリケーションに対して複数種類のコアを使い分けることで，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，コアやそれに付随するキャッシュ，バスの設計・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生成するツールセットとして FabHetero を提案している．本論文では，FabHetero のフレームワークの内，様々な構成のバスを自動生成する FabBus について述べる．FabBus[1)] が生成するシステムバスの構成はメモリのと通信が効率的に行えず，周辺機器との接続も考慮されていない．そこで本論文では，同時多対多通信と周辺回路との接続を可能にするより実用的なツールを提案する．

Abstract

Recently, heterogeneous multi-core processor has gotten attention from researchers as an approach to achieve higher-performance and lower-energy consumption. Using a proper superscalar core for characteristic in a program contributes to reduce energy consumption and improve performance. However, designing a heterogeneous multi-core processor requires a large design and verification effort which is multiplied by the number of different core types. To solve this problem, we propose FabHetero to automatically generate diverse heterogeneous multi-core processors. This paper mainly treat FabBus. A construct of system bus which generated by existing FabBus cannot perform efficiency communication. In addition, FabBus cannot connects peripheral circuit. Thus, this paper expand FabBus to perform multipoint-to-multipoint communication and connect peripheral circuit.

目 次

図 目 次

表 目 次

1　はじめに

近年，高性能・低消費電力の両立のためのアプローチとして，構成の異なる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．ヘテロジニアスマルチコアプロセッサは，実行するプログラムの特徴に合わせて選択的にコアを使い分けることで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，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を自動生成するツールセットとして FabScalar[2] が提案されており，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場に短縮している．しかし，FabScalar が自動生成するのはプロセッサコア部分のみであり，キャッシュシステムやバスシステムを自動生成する機構は実装されていない．そこで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生成するツールセットとして FabHetero を提案している．FabHetero はコア自動生成ツール FabScalar に，キャッシュを自動生成する FabCache，バスを自動生成する FabBus を追加したものである．本論文では主に FabBus について述べる．既存の FabBus が生成するシステムバスの構成は共有バスシステムが採用されて

いる．共有バスシステムは，バス帯域幅を複数のユニット間で共有することでバス上にある複数のユニット間の通信を可能にしている．しかし，このシステムでは複数のマスターのアクセス先スレーブが異なっている場合でもどれか一つを選択して通信するので，スレーブのリソースが余ってしまいデータアクセス効率が悪い．加えて，FabBus はチップ外の周辺機器との接続が考慮されていない．そこで本論文では，データアクセス効率化のために FabBus を同時多対多通信ができるより効率的なプロトコルに改良する．さらに周辺機器との接続を容易にする汎用的な変換器の実装を行う．加えて、FabBus のプロトコルを AMBA2 から AMBA4 へと移行する。

2　ヘテロジニアスマルチコアプロセッサ

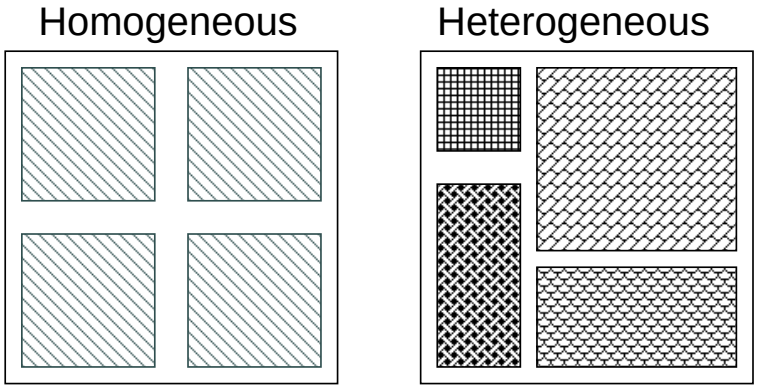

図 2.1: ホモジニアスマルチコアとヘテロジニアスマルチコア

現在，図 2.1(左) のような同種の CPU コアを１つのチップに複数個搭載したホモジニアスマルチコアプロセッサが広く用いられている．ホモジニアスマルチコアプロセッサは特徴の異なる様々なアプリケーションに対しても同一アーキテクチャのコアで処理するため，ハードウェアリソースの過不足が起こりやすく，電力効率の低下の一因となっている．そこで図 2.1(右) のような異種のアーキテクチャのコアを複数個搭載したヘテロジニアスマルチコアプロセッサを用いて，様々なアプリケーションに対して適切なコアを使い分けることで高性能と低消費電力の両立を図る．しかし，ヘテロジニアスマルチコアプロセッサを研究する上で各コアの構成やコア数，キャッシュシステム，バスシステムなどの組み合わせ

の膨大さや設計・検証にかかる時間が大きな障害となっている．

2.1 FabScalar

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ーコアを自動生成するツールセットとしてFabScalarがノースカロライナ州立大学で提案されている．FabScalarはフェッチ幅やパイプライン段数，各ユニット数などの各種パラーメータを与えることで様々な構成のスーパースカラコアを自動生成するツールである．FabScalarを用い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，FabScalarが自動生成するのはプロセッサコア部分のみであり，それに付随するキャッシュシステムやバスシステムを自動生成する機構が実装されていない．

2.2 FabHetero

ヘテロジニアスマルチコアプロセッサの構成例を図2.2に示す．ヘテロジニアスマルチコアプロセッサではシステムを構成する個々のプロセッサコアの特徴によって最適なキャッシュ構成やバスシステムが異なる．しかし，最適なキャッシュ構成やバスシステムを手動で設計・評価するのは組合

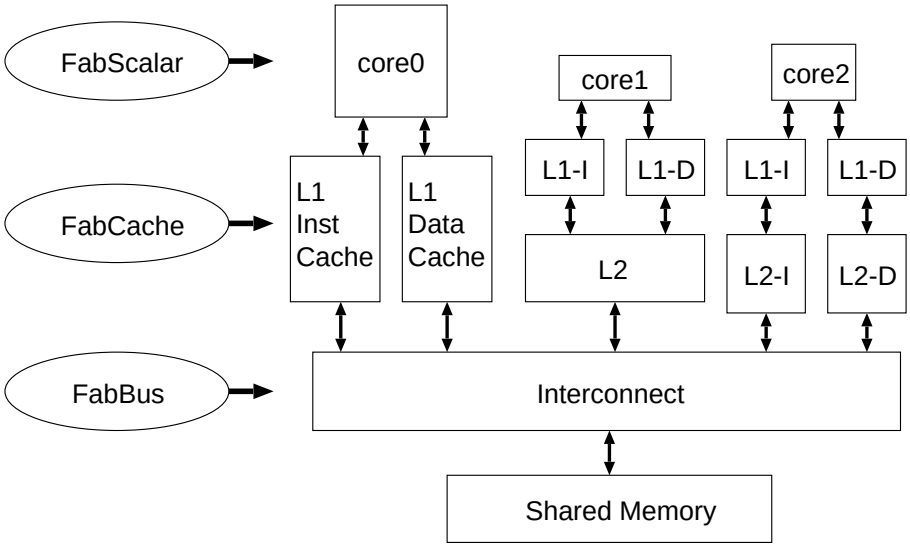

図 2.2: FabHetero

わせの膨大さから非常に困難である．そこで著者らの研究グループではこの問題を解決するために，コア自動生成ツール FabScalar に，キャッシュ自動生成ツール FabCache，バス自動生成ツール FabBus を加えた FabHetero を提案している．

3 既存の FabBus

3.1 FabBus の構成

FabHetero によって，生成される各ユニット間の接続には，柔軟なバス構成が必要とされる．そこで可変構成のマルチコア及びキャッシュシステムに対応したバスフレームワークの自動生成ツールとして FabBus が提案されている．FabBus が生成するバスシステムは，AMBA2.0 の仕様に準拠したものである．FabBus が AMBA2.0 を採用した理由としてハードウェア規模が小さく仕様がシンプルで改良が容易であるということが挙げられている．また FabBus にはキャッシュコヒーレンシ制御機構としてスヌープバスが実装されている．スヌープバスは，ヘテロジニアスマルチコアプロセッサ環境への適用を考え，バス仕様と分離したキャッシュ機能ユニットの生成が可能となっている．基本的な処理内容は入力アドレスの優先度処理とアクセス認可のみであるが，優先度処理に関してはラウンドロビン及び固定優先度のアルゴリズムを使用している．以下に FabBus が持つ二つの問題点を述べる．

3.2 FabBus の問題点

一つ目は，共有バスシステムを採用しているため同時多対多通信が出来ない点である．このシステムは，ハードウェア規模は比較的小さく実装できるが，性能面において限界がある．例えば，複数のマスターのアクセス先スレーブが異なっている場合でも，どれか一つのマスターを選択して通信するのでスレーブのリソースが余ってしまう．そこで各ユニットをクロスバースイッチを用いて接続することで，同時多対多通信を実現する．

二つ目はチップ内の各ユニットの接続のみに焦点を当てているため周辺回路との接続が考慮されていない点である．実チップや FPGA を用いて評価をとる場合、様々な周辺機器を接続することが想定される．周辺機器は様々なデータ幅を持つため，プロセッサ内部のデータ幅と異なる場合がある．ビット幅の異なるバスの接続は不可能なため，ユーザーによる修正が必要となってくるが，様々な周辺機器を接続する場合ユーザーの負担が増加する．そこでビット幅の異なるバスの接続を容易にする変換器の実装を行う．

4 AMBA

AMBA プロトコルとは，ARM 社によって開発されたマルチプロセッサ用のシステムバスの規格である．ARM プロセッサに限らず MIPS コア等の様々なシステムに搭載されており，実際のハードウェアシステムにおいて現在きわめて広く採用されているバス仕様である．AMBA プロトコルには現在いくつかのバージョンが存在し，その中でも Rev2.0 と 4.0 が一般的なシステムに用いられている．

4.1 AMBA2.0

AMBA2.0 プロトコルは主にキャッシュ等の内部メモリとプロセッサユニットとの間を接続する高性能な AHB バスと，低周波数帯域で動作し周辺装置間の接続を行う APB バスに分けられる．AHB 仕様のバス構造を図 4.3 に示す．AHB は，以下のような複数のバスとそのほかの信号で構成される．

・アドレス・バス（HADDR）

・読み出しデータ・バス（HRDATA）

・書き込みデータ・バス（HWDATA）

・制御信号

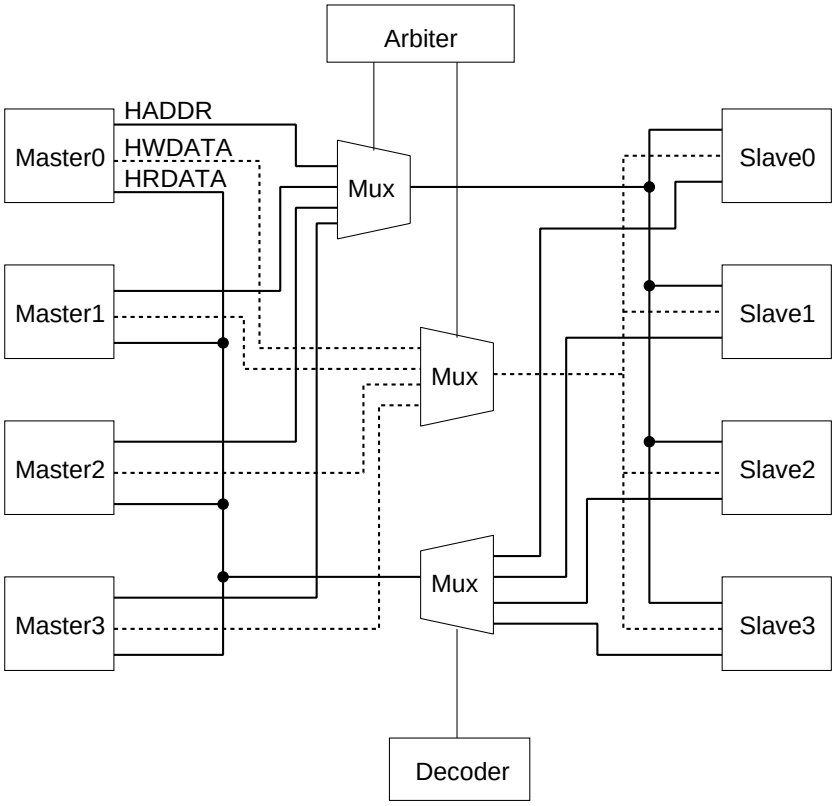

図 4.3: AHB 仕様のバス構造

・応答信号

AHB のバスはそれぞれの情報を載せるという役割しかなく，バス間の動作タイミングは厳密に定義されている．よって一つの転送を行うために各バスが協調して動作しなければならず，独立して動作することはできない．AHB 型のバスの問題点は，遅いデバイスとのデータ転送が存在すると後続のデバイスとのデータ転送が待たされることである．例えば，アドレスをアドレス・バスに載せた次のサイクルでは，必ずそのアドレス

に対するデータをデータ・バスで転送する必要がある．よって，遅いデバイスがデータの準備に手間どると，その間アドレス・バスは次に進めず，バスの効率が下がる．また，高速なデバイスが後続の転送を行う場合，その高速性が生かせない．

4.2 AMBA4.0

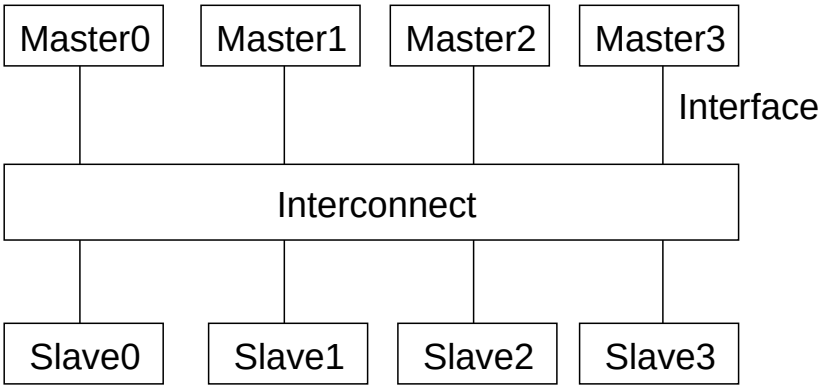

図 4.4: AXI 仕様の構造

AMBA4.0 プロトコルは主に AMBA2.0 の AHB，APB との後方互換性を持つ AXI とコヒーレンシ制御を扱う ACE に分けられる．本論文では AXI プロトコルについて述べる．AXI 仕様の構造を図 4.4 に示す．AMBA4.0 は新たにチャネル方式を導入し従来の AMBA バスから大きな変更が行われた．AXI 仕様の信号構成を以下に示す．

・読み出しアドレス・チャネル

・書き込みアドレス・チャネル

・読み出しデータ・チャネル

・書き込みデータ・チャネル

・書き込み応答チャネル

従来「xxx バス」と称していたものが「xxx チャネル」と変更されている．AHB が各バスが強調して動作しなければ成らなかったのに対して，AXI では各チャネルごとに独立して情報を送出できる．例えば，AHB のような「アドレスを送出したら次のサイクルで対応するデータを必ず転送しなければならない」などの，チャネル間の手順の制約が存在しない．このため，ほかのチャネルのタイミングに関係なく情報を次々に転送できる．このように，AXI 仕様ではバス間が独立し，それだけで情報ストリームを送出できるので,「チャネル」という呼びかたをしている．なお，AXI はインターフェース仕様であり，バス仕様ではない．したがって，インターコネクトの部分は仕様の範囲外である．インターコネクトとしては，以下のような選択肢が考えられる．

・ポイント・ツー・ポイント

・シングルレイヤ

・マルチレイヤ

・アドレスはシングルレイヤ，データはマルチレイヤ

このように AXI 仕様は柔軟なバス構成を取ることができ，FabBus の要求とも合致する．

5　実装

5.1　同時多対多通信

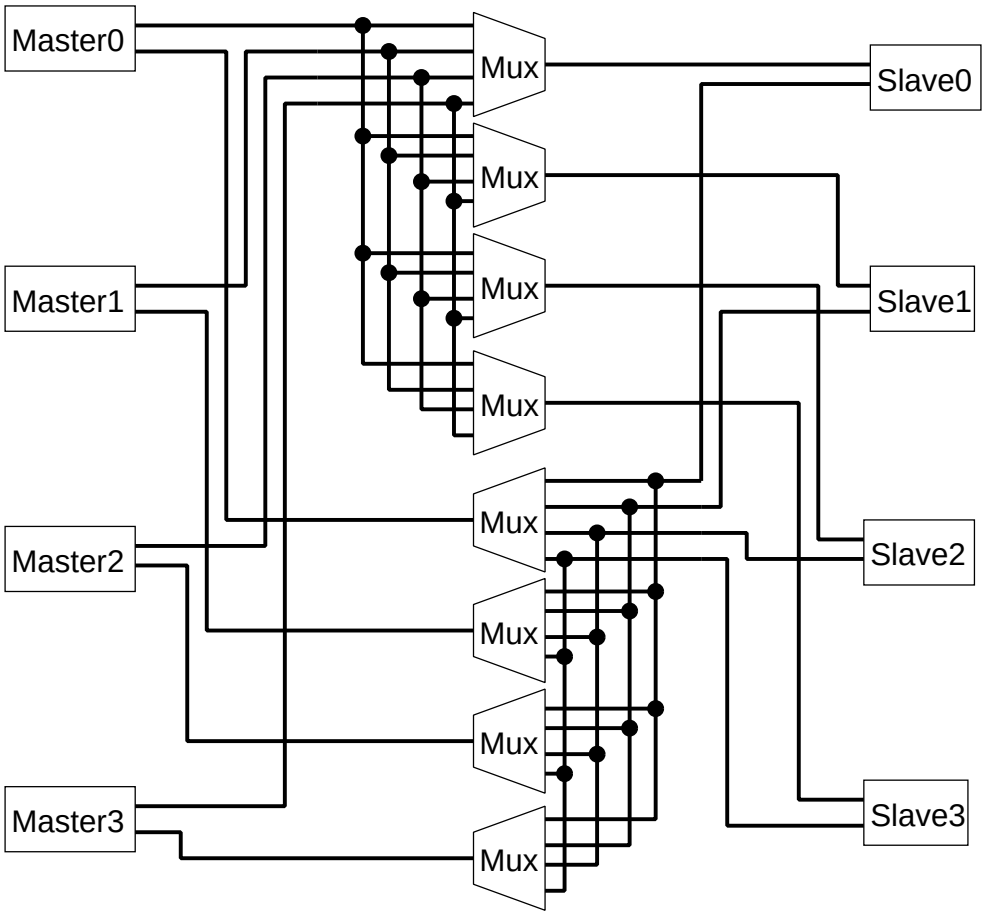

図 5.5: 多対多通信の概念図

既存の FabBus が生成していたバス構成を図 4.3，今回実装したバス構成を図 5.5 に示す．既存の FabBus の実装方式の場合，複数のマスターから同時にアクセス要求が発行されたとき１つのマスターを選択して通信

を行っていた．年々マスターとなるコア数は増加しており，コアの待ち時間が増加してしまう．従来方式の問題点として，各マスターのアクセス先スレーブが異なるにもかかわらず同時に通信することができないという点が挙げられる．そこで，図 4.3 のバス構成を実装した．従来の方式はバス帯域幅を複数のスレーブ間で共有しているのに対して，今回実装した方式はそれぞれのスレーブに対して専用のバスが用意されている．よって，複数のマスターのアクセス先スレーブが異なっている場合同時多対多通信が可能となる．図 5.5 のマルチプレクサはラウンドロビンアルゴリズムを用いてバスマスターのアービトレーションを行った．

5.2 周辺回路との接続

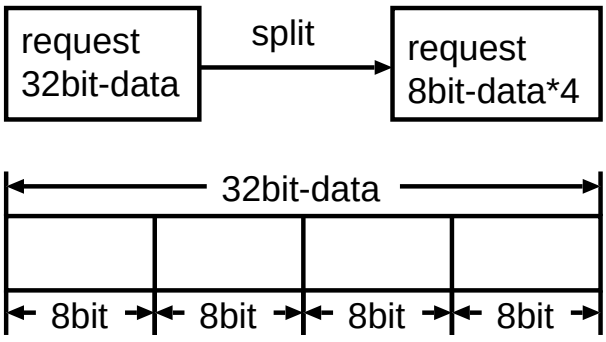

図 5.6: トランザクションの分解

本論文では，周辺機器の接続を容易にする汎用的な変換器の実装を行った．通常，多ビットバスから少ビットバスへの変換は図 5.6 のように 1 つ

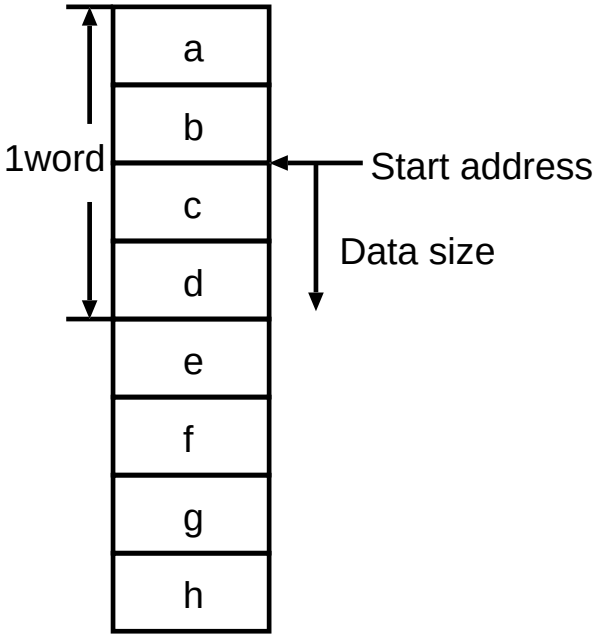

図 5.7: 周辺機器との接続を考慮した実装

の読み出し動作を複数のトランザクションに分解することで実現するが，単純にそれを行うと FIFO を持つようなメモリの場合，不必要なデータアクセスを行ってしまい FIFO の中身を破壊してしまう危険がある．そこで，今回実装した変換器ではこの問題に対してプロセッサからのワード単位のアクセスを必要な領域のみへのアクセスに変換することで対応した．変換器の動作を図 5.7 を用いて説明する．プロセッサから送られてくるワード単位のアドレスから読み出したい領域の先頭アドレスに変換する．そのアドレスからどれだけのデータを読み出すかという情報を用いて読み出す領域を決定する．この操作により必要な領域のみへのアク

セスを実現できる.

6　評価・検証

本論文で実装した同時多対多通信可能なバスモジュールが基本的なシステムバス通信を達成できているかを確認するためにSPEC2000INTからbzip，gzip，gap，gcc，mcfの５つのベンチマークを実行した．各ベンチマークは最初の１億命令を実行完了し，正しい通信が行われていることが確認できた．次に実装したバスモジュールを論理合成しNAND換算の総ゲート数と動作周波数を算出した．表6.1に結果を示す。

対象	総ゲート数	動作周波数
従来のFabBus	1,174	454.5MHz
今回実装したバス	4,882	476.1MHz

表 6.1: 総ゲート数と動作周波数

以上の結果より，同時多対多通信を行うインターコネクトのペナルティとしてハードウェア量の増加が確認できた．ハードウェア量の増加は主にアービターの複雑化と多重化によるものだと考えると，妥当な結果であると言える．

7　おわりに

本論文では，ヘテロジニアスマルチコアプロセッサ環境を対象としたオンチップ同時多対多通信及びオフチップ通信を行えるバスの実装を行った．ベンチマークの実行結果から実装した回路は正しく動作していることが確認できた．しかし，現時点ではコヒーレンシ機能が未実装である．今後の展望としてコヒーレンシ機能に加え，より柔軟なバスフレームワークの自動生成対応していくことが挙げられる．

謝辞

本研究を進めるに当たり，ご指導を頂いた佐々木敬泰助教，並びに近藤利夫教授，研究室の皆様に感謝の意を表します．また，事務を担当して頂いた田中さんに感謝します．

参考文献

[1] 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，SWoPP 2012，2012年8月

[2] N. K. Choudhary，et. al. ”FabScalar: Composing Synthesizable RTL Designs of Arbitrary Cores within a Canonical Superscalar Template ”. ISCA-38, pp. 11-22, June 2011.

[3] ARM: AMBA

http://www.arm.com/

A　プログラムリスト

B　評価用データ