修士論文 2008 年度（平成 20 年度）

IP 化を実現するバスブリッジの設計と実装

慶應義塾大学 大学院 政策・メディア研究科
氏名：松谷 健史

指導教員
主査：村井 純　（慶應義塾大学　環境情報学部）
副査：中村 修　（慶應義塾大学　環境情報学部）
副査：Rodney Van Meter　（慶應義塾大学　環境情報学部）

平成２０年７月１日

修士論文要旨 2008 年度（平成 20 年度）

IP 化を実現するバスブリッジの設計と実装

論文要旨

コンピュータの処理能力に影響する CPU、メモリ、拡張機器などのハードウェア構成は，物理的な接続によって静的に決定されている。

コンピュータ内部のハードウェアはバスによって接続されているが、バスには距離、デバイスの数、並列トランザクション数、再構成可能性の点で制限がある。

特別に設計された I/O ネットワークでさえ、それら制限の一部分を解決するにすぎない。

この問題を解決するために、本論文では IP レベルネットワークをメモリと I/O デバイスのバス相互接続に用いる手法を提案する。

そして提案内容の一部を実現するために、バスのプロトコルとインターネットプロトコルとを相互に変換する IP バスブリッジを設計し、実装、評価した。

IP バスブリッジは MIPS アーキテクチャ CPU とメモリと拡張機器を Ethernet によって接続しており、IP を使用した接続と使用しない接続の両方に対応している。

キャッシュのある状態，およびない状態の両方で本システムを 200 ミリ秒までの遅延環境で評価した。

その結果メモリと CPU を IP によって接続し、演算ができることを確認した。

キーワード:

1. バスブリッジ 2. バス 3. CPU 4. FPGA 5. TLB

慶應義塾大学 大学院 政策・メディア研究科

松谷 健史

Abstract of Master's Thesis - Academic Year 2008

The Design and Implementation of An IP Bus Bridge

Summary

What a computer can do is limited by its hardware. The hardware configuration of a computer is statically determined by the physical attachment of devices, such as CPUs, memories and expansion peripherals. Device connections are done using a bus, but buses are limited in distance, number of devices and concurrent transactions, and re-configurability. Even specialized I/O networks only partially eliminate these problems. However, the hardware resources required by applications vary. Improved flexibility in dynamic system configuration is necessary.

This thesis proposes the most flexible and scalable alternative possible: using an IP-level network as the interconnect for both memory and I/O devices. An IP network can improve flexibility, cost and scalability.

A bus bridge that provides protocol translation between the computer bus and Internet Protocol has been designed, implemented, and evaluated. The IP Bus Bridge connects the memory and peripherals of a MIPS-based system over Ethernet, with or without IP. The performance of the system with and without cache is evaluated for latencies up to 200 msec, demonstrating that it is possible to connect memory via IP.

Key words:

1. BusBridge 2. Bus 3. CPU 4. FPGA 5. TLB

Takeshi Matsuya

Keio University, Graduate School of Media and Governance

目次

図目次

表目次

第1章　序論

本章では、本研究の背景としてコンピュータバスとメモリバス、そして LAN の速度の推移とアプリケーションに応じたハードウェア構成変更の必要性について述べ、最後に本研究の目的と成果について述べる。

1.1　本研究の背景

近年の技術の進歩により、コンピュータの高速化、低価格化が進み、インターネットをはじめとするネットワークインフラストラクチャが整備されることにより、様々なコンピュータがネットワークに接続されるようになった。

コンピュータの高速化は、ムーアの法則[1]に従い、単一 CPU のトランジスタの集積度の向上と CPU の動作クロックを向上させることでおこなわれてきたが、最近では単一 CPU あたりの性能向上から複数の CPU コアを使うことで並列処理を行い高速化がおこなわれている。

他にもコンピュータの性能を測る指針のひとつにコンピュータバスの能力があげられる。現在のコンピュータアーキテクチャの基本系としてノイマン型[2]の計算機がある。図 1 にノイマン型アーキテクチャの概念図を示す。CPU はバスを介してメモリから命令を順次読み込み、必要に応じて結果をメモリに書き出す。このことから CPU の性能はバスの能力に依存することがわかる。

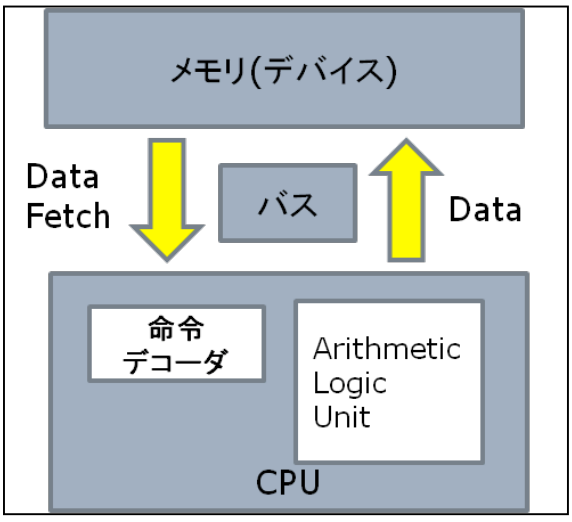


図 1：ノイマン型アーキテクチャ

1.1.1 PC バスと LAN の速度差の減少

バス[4]の性能はコンピュータの性能向上とともに飛躍的に向上している。一方、LAN の性能向上も著しい。表 1 に PC バスと LAN そしてメモリ[3]の転送速度の推移を示す。

図 2 の折れ線グラフから 1990 年代までは PC バスが LAN に対して 10 倍以上の転送能力があったのに対し、2000 年半ばになるとその差が縮まっていることがわかる。このことより転送速度にのみ着目すると LAN を PC バスとして利用することも可能といえる。

表 1：PC バス・LAN・Memory の速度推移

年代	規格	バス幅	速度	転送速度/s
1.PCバス				
1981年	ISA	8ビット	8.33MHz	8Mbytes
1990年	PCI 1.0	32ビット	33MHz	133Mbytes
1995年	PCI 2.1	64ビット	66MHz	533Mbytes
2000年	PCI-X 1.0	64ビット	133MHz	1.06Gbytes
2002年	PCI Express 1.0(X8)		2.5GHz	2Gbytes
2007年	PCI Express 2.0(X16)		5GHz	8Gbytes
2.LAN				
1983年	IEEE802.3　10BASE5		10Mbps	1.25Mbytes
1995年	IEEE802.3u　100BASE-TX		100Mbps	12.5Mbytes
1999年	IEEE802.3ab 1000BASE-T		1Gbps	125Mbytes
2004年	IEEE802.3ak 10GBASE-CX4		10Gbps	1.25Gbytes
200X年	100GBASE		100Gbps	12.5Gbytes
3.MEMORY				
1993年	EDO DRAM (x-2-2-2)	64ビット	66MHz	264Mbytes
1997年	SDR-SDRAM(PC133)	64ビット	133MHz	1.066Gbytes
2001年	DDR-SDRAM(PC-2100)	64ビット	266MHz	2.133Gbytes
2004年	DDR2-SDRAM(PC2-6400)	64ビット	400MHz	6.4Gbytes
2006年	DDR3-SDRAM(PC3-14400)	64ビット	900MHz	14.4Gbytes

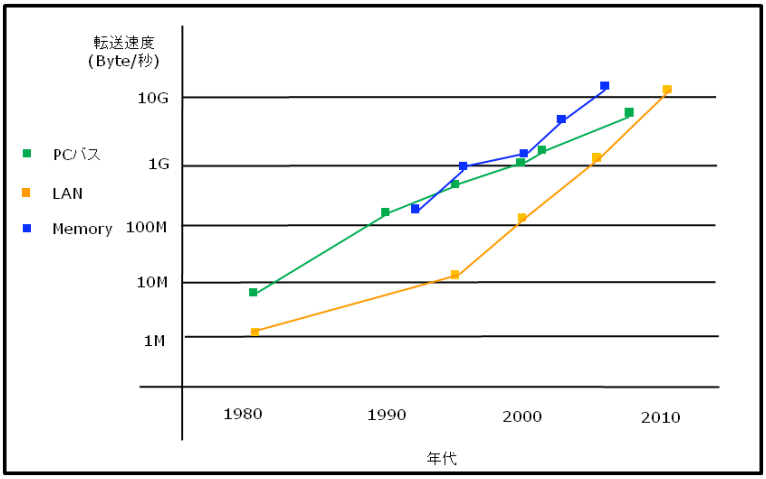


図 2：PC バス・LAN・Memory の速度推移

1.1.2 アプリケーションによって異なるハードウェア資源要求の違い

コンピュータの利用範囲の広がりとともに、アプリケーションによるハードウェア資源の要求範囲が広くなってきている。

メールやワープロ、表計算アプリケーションなどを利用するときは多くのハードウェア資源を必要としないが、HD ビデオの編集アプリケーションではビデオキャプチャーボードなどのデバイス資源や動画圧縮伸長処理などに多くの CPU やメモリ資源が要求される。

デバイスや CPU、メモリ等のハードウェア資源の増強は、コンピュータ内のバスに挿すことによって可能ではあるが、一部のハードウェア要求が多いソフトウェアのために常時最大限のハードウェアを準備しておくことはコスト面において難しい。

また、同一匡体で増設できるハードウェア資源は、バスの仕様上の制限や物理的な拡張バスの数の上限によって、制限されてしまう。

1.2 本研究の目的

本研究の目的はアプリケーションやユーザの要求に応じてハードウェアを動的に割り当てるコンピュータアーキテクチャを実現するために、バスの距離と数の制限に関する問題を解決することである。

そのためにハードウェアの拡張をおこなう際に重要となる、バスの距離と数の制限を解決する必要がある。

距離の制限とは、CPU やメモリ、周辺装置を増設するバスは主に匡体の中にあるため、接続できる距離に限界があることを示す。

数の制限とは、物理的なバスの接続数や仕様によってハードウェア増設できる数が制限されてしまうことを示す。

本研究ではネットワーク技術を使い、IP とハードウェアのバスレベルのプロトコル変換ができる専用のバスブリッジ[23][24][25][26]を設計することにより、バスの距離と数の制限に関する問題を解決する。

1.3 本論文の構成

本論文は全 6 章で構成されている。第 2 章では本研究に関連する技術とその問題点について述べる。第 3 章では，その問題点を解決するための本研究のアプローチとその有用性について述べる。第 4 章では本システムの実装について述べる。第 5 章では本システムに対する定量的・定性的評価の結果と考察を述べ，最後に第 6 章で本研究によって得られた結論を述べる。

第2章　関連技術

前章ではハードウェア構成は静的であり、アプリケーションの要求に応じてハードウェア構成を動的に変えることが困難なことを述べた。本章では、関連する既存の接続技術で実現できることと問題点について整理する。

2.1　相互接続の様々なレベル

コンピュータの相互接続を実現する手法には図 3 で示すように様々な接続レベルがある。そして近年の傾向として下記に示す方向に向かっている。

・より多くのデバイスへの接続（バスあたり、システムあたり）
・より遠くとの接続
・より階層化されたアーキテクチャ(ISO によって策定された OSI 参照モデルのように)
・動的な再構成

2.3 節以降ではそれぞれの相互接続レベルに応じて実現できることと問題点について述べる。

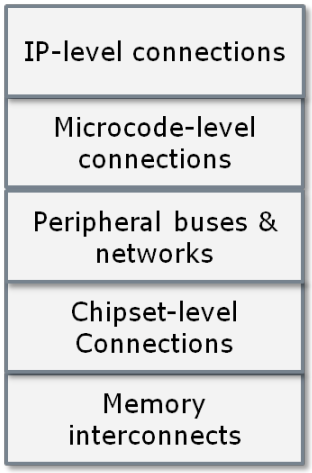


図 3：様々な相互接続レベル

2.2　PC アーキテクチャで使われているバス

本節では、PC の内部のアーキテクチャに関して解説する。

PC の内部は単一のバスではなく複数の種類バスによって構成され、ブリッジによって接続している。PC 内のバスとブリッジの接続例を図 4 に示す。

上部から CPU、ノースブリッジ、サウスブリッジが配置される。

CPU は高速な FSB(Front Side Bus)を経由してノースブリッジと接続する。

ノースブリッジは直接CPUと接続され高速転送が必要なグラフィックチップやメモリのバスと接続され､Intel ではノースブリッジ内にグラフィック機構を内蔵したものを GMCH（Graphics and Memory Controller Hub）と呼ぶ。

サウスブリッジは比較的低速な I/O デバイスと接続され、Intel では ICH(I/O Controller HUB)とも呼ばれる。

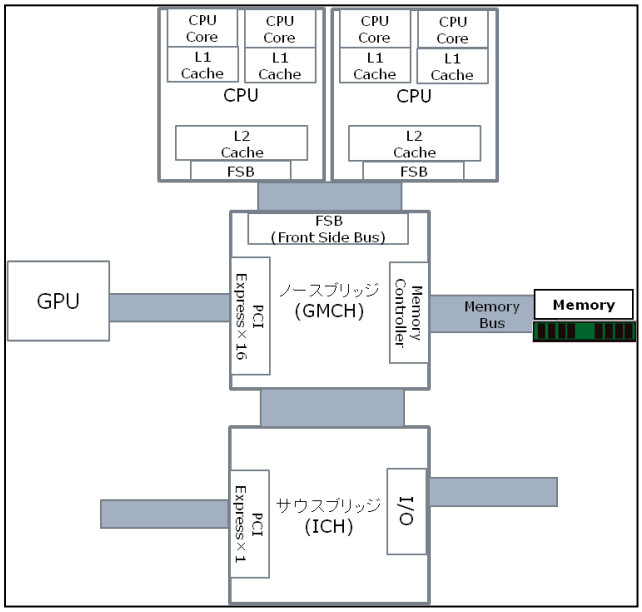


図 4：PC アーキテクチャ・コンピュータバス

2.3　Memory Interconnects

本節では、メモリの相互接続手法による CPU の並列処理の特徴について述べる。

2.3.1　NUMA (Non-Uniform Memory Access)

NUMA[5][12][13]はすべての CPU が共有メモリを持つ密結合なマルチプロセッサモデルで、それぞれの CPU がメモリへアクセスする速度は距離に依存し一定ではない。

CPU と Memory 間の接続は、プロセッサ間を高速バスで接続するか、もしくは現在主流の CPU の様にチップ内で NUMA アーキテクチャを実現しているものが多い。

アクセス速度が不均等な点以外は通常の SMP システムと同じ扱いのため、CPU 追加によるアプリケーションの高速化が容易である。

この方式では CPU の増設が可能であるが、高速な専用バスによる接続が必要になる。

2.3.2 分散メモリ (Distributed Memory)

CPU が互いに独立したメモリ空間を持つプロセッサモデルでメッセージのやりとりによって並列計算を進めていく。

共有メモリを持たずネットワークにより接続されている PC などで容易に実現できる並列処理コンピューティング環境である。並列実行用ライブラリとして MPI[20] (Message Passing Interface)や PVM[21] (Parallel Virtual Machine)などがある。

追加した CPU は独立したメモリ空間を持つため、MPI や PVM などに対応したアプリケーション以外では高速化が難しい。

この方式では CPU の増設が可能だが、メッセージパッシングに対応したアプリケーションが必要となるという問題がある。

2.4 Chipset-level Connections

本節ではチップセットレベルで実現される相互接続について述べる。

2.4.1 vPro

vPro は、Intel よりクライアント PC 管理のソリューションとして提供されているブランド名である。

仮想化の Intel VT (Virtualization Technology)とクライアント PC 管理の Intel AMT (Active Managemtn Technology)の 2 つの技術から構成され、vPro 対応の CPU、チップセット (ノースブリッジとサウスブリッジ)、LAN コントローラを揃えることによって実現できる。

図 5 は vPro が導入されたクライアント PC を管理する様子を示している。

管理者用 PC は、Web ブラウザを使って下記についてクライアントを管理することができる。

1. Fan や CPU の識別などのハードウェア情報の取得
2. 電源のオンおよびオフ
3. ネットワーク経由でのシリアルコンソール操作(BIOS 画面や一部のシリアルコンソール対応 OS)

4. リモートディスクからの起動

これらは、チップセット内の機能で実現しており、コンピュータの電源が入っていない場合や、OS が起動していない状態でも実行することができる。

本技術に関してのみいえば、クライアント PC の管理機能のみに特化しており本研究での流用は不可能である。

しかし、チップセットはコンピュータシステムにおける重要なバスがつながっている中枢部であり、本レベルでの相互接続は CPU、メモリ、デバイスを含むすべての拡張が可能である。

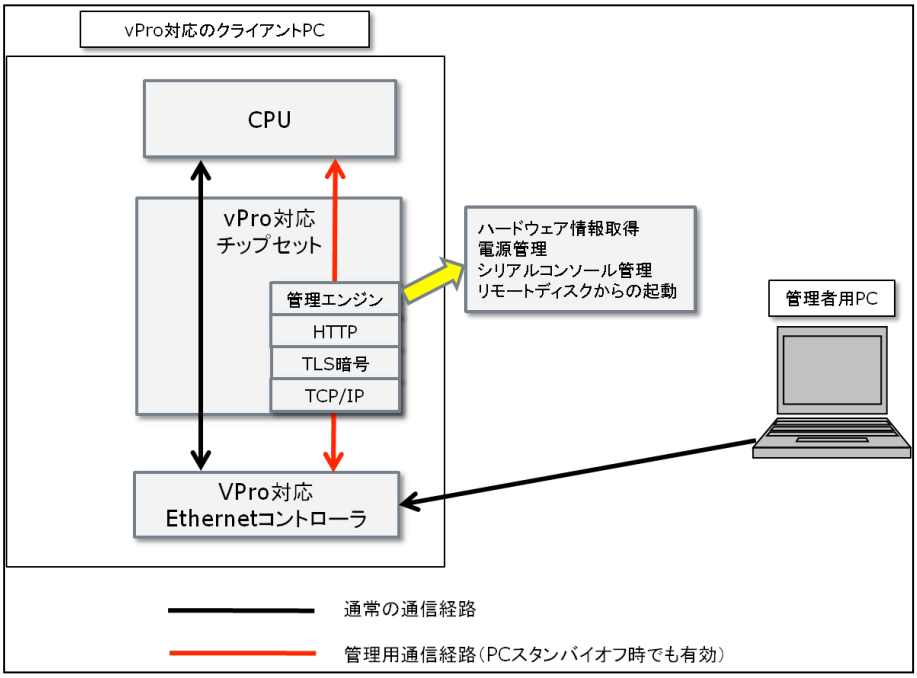


図 5：AMT の機能と仕組

2.5 Peripheral buses & networks

本節では、既存バスの概要を述べたのちに、バスレベルでの相互接続技術について解説する。

2.5.1 PCI バス (Peripheral Component Interconnect)

PCI SIG (Special Interest Group)によって策定されたバス規格。1992 年 PCI1.0[8][14][17]として仕様が決まり、後述の PCI Express[15]が使われるようになるまで、多くの PC に採

用されてきた。表 2 に仕様を示す。

表 2：PCI バス仕様

クロック周波数	同期バス 33MHz/66MHz
アドレスバス	32/64bit
データバス	32bit
転送速度	133MB/s～533MB/s
転送方式	パラレル通信

この手法ではデバイスやメモリの増設が可能であるが、内部専用のバスのため距離や数の制限がある。

2.5.2 PCI Express BUS

PCI Express BUS は、2002 年に PCI-SIG によって策定された PCI バスの後継にあたるバスである。

前身のバスとして Intel を中心に開発した NGIO (Next Generation I/O)があったが、PCI バスと互換性がなかったため、インテルが 3GIO (Third Generation I/O)と開発を進め PCI-Express となった。

NGIO は HP、IBM が開発していた Future I/O と統合され次節の InfiniBand として仕様化された。表 3 に仕様を示す。

表 3：PCI Express 1.1 バス仕様

クロック周波数	同期バス 2.5GHz (×1 レーン)
アドレスバス	32/64bit
データバス	32bit
転送速度	250MB/s(×1)～4GB/s(×8) ※片方向
転送方式	8b/10b シリアル通信

PCI-Express はシリアル通信で、図 6 で示す下記の 5 つの層から構成され、ネットワーク通信と類似している。

- ソフトウェア層
- トランザクション層

TLP (Transaction Layer Packet)の生成と処理を行う

下記に TLP コマンドの概要を一覧する。

1) メモリリクエスト

2) I/O リクエスト

3) コンフィグレーションリクエスト

4) コンプリッション

リクエストに対する応答やデータ

5) メッセージ

割り込みやパワーマネージメント

・データリンク層

トランザクション層のデータを確実に転送するために、リンクを管理しエラー検知およびエラー訂正をする。

ACK/NACK などの要求。

・物理層

8b/10b 変換、スクランブルデスクランブル等の電気的信号を定義する。

・メカニカル層

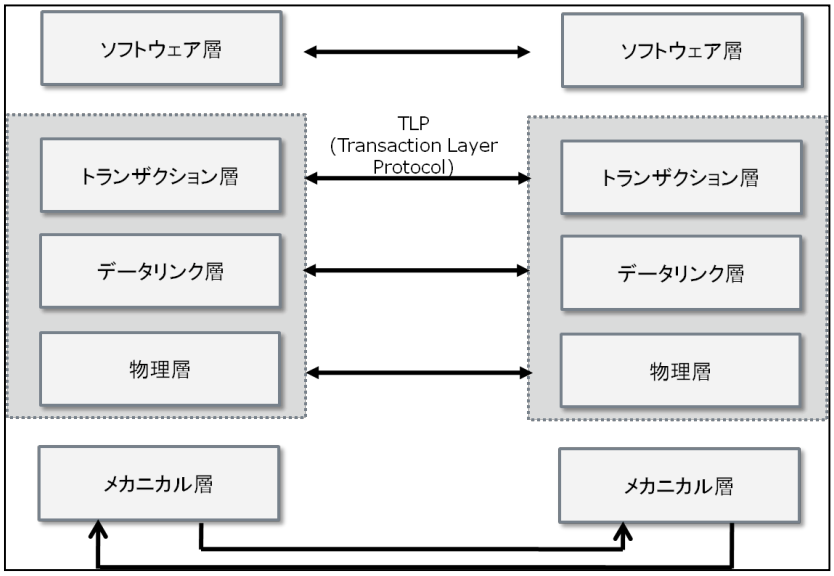


図 6：PCI Express レイヤ

PCI Express のフレーム構成を図 7 に示す。ヘッダーサイズはアドレスバスを 32bit 利用する場合は 12 バイト、64bit 利用する場合は 16 バイトになる。

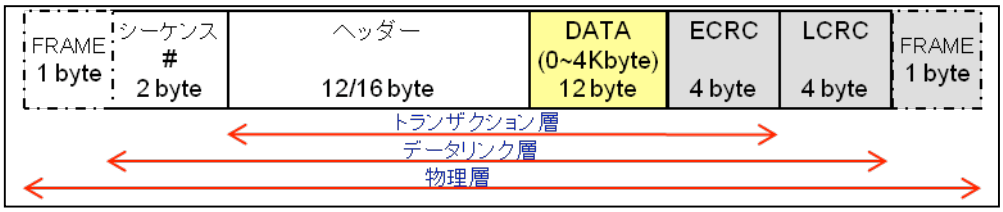


図 7：PCI Express フレーム構成

この手法ではデバイスやメモリの増設が可能だが、内部専用のバスのため距離や数の制限がある。

2.5.3 InfiniBand

InfiniBand はコンピュータの種類や接続デバイスなどに一切依存しない汎用の I/O アーキテクチャであり、スーパーコンピュータのような密結合でのプロセッサ同士の相互接続やコンピュータとストレージもしくはネットワーク機器を接続するために用いられる。

InfiniBand は一方向あたりの信号帯域が 2.5Gbps、8B/10B 符号化方式を採用しているため、データ転送速度は最大 250MB/s である。複数チャンネルで接続することにより 4X では 10Gbps、12X では 30Gbps の信号帯域幅を確保できる。ほとんどのシステムで 4X 以上の構成をとっている。

HPCC (High Performance Computing Clusters)やデータセンタ内のクラスタシステムなどで用いられ、PC バスと比べて対応しているデバイスが少ない。

この手法では CPU やデバイス、メモリの増設が可能であるが、距離の制限がある。

2.5.4 RDMA (Remote Direct Memory Access)

高速なネットワークにおいての転送では、CPU による TCP/IP プロトコルスタック処理やデータコピーがスループットの低下の原因となる。

RDMA[32]では CPU を介さず、ハードウェアによるプロトコルスタックのサポートとネットワークとのメモリの転送を直接行うことによりスループットを上げることができる。

この手法は、ネットワークデータとローカルメモリ間の転送を高速化するもので、リモート PC のメモリを直接利用するものではない。他のサービスとの併用が考えられる。

2.5.5 BUS over Ethernet

BUS over Ethernet は Ethernet 上にコンピュータバスをカプセリングすることで、ネットワークを使ってバスを延長することができる。

PCI-Express バスに対応したものとして ExpEther[18]がある。PCI-Express バス上のフレームを Ethernet にカプセリングすることにより PCI-Express バスの延長を行う。

PCI-Express の I/O コマンドに対応するため、I/O ボードに対応できる。メモリコマンドには対応していない。

この方式では拡張ボードの延長はできるが、システムバス間のブリッジはできない。（これはコンピュータ同士のアドレス空間におけるコンフリクト問題の為である）

例えば、図 8 のように Node-1 と Node-2 のシステムバスが接続されると、Node-1 のデバイス 1 と Node-2 のデバイス 2 が$20000 の同じアドレスが割り当てられることになる。
Node-1 の CPU がアドレス$20000 の回路へ接続を行うと、二つの回路へ接続され回路の動作や応答などがコンフリクトしてしまう。

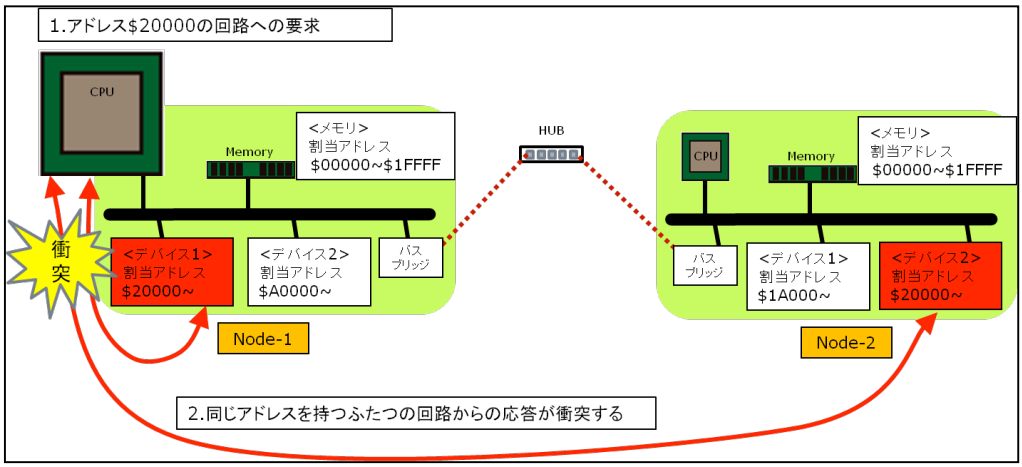


図 8：アドレスコンフリクト問題

2.6 Microcode-level connections

本節では CPU 内のマイクロコードレベルで実現される接続について解説する。

2.6.1 Virtualization

Virtualization はメモリや I/O を仮想化することにより、ハードウェアへのアクセスを捕捉しソフトウェアで処理することにより仮想マシン上からはあたかもハードウェア構成が変わったようにみせられる手法である。

リモート PC 上のメモリをローカルのメモリ回路に割り当てる場合やローカルメモリへのアクセスがあった場合に、ネットワーク経由で該当コンピュータへメモリ読み出し指示を行うなどの処理を行う。

この方式の問題として以下の点が挙げられる。

1. ハードウェア処理と比べると動作が遅い。
2. 仮想化を実現するために CPU、OS、ハイパーバイザーが必要。

2.7 IP-level connections

本節では IP レベルでの接続技術について解説する。[28][29][30][31]

2.7.1 iSCSI & USB/IP

iSCSI や USB/IP は、デバイスのインターフェイスプロトコルを IP パケットでカプセリング化することにより、ネットワーク経由でデバイスを接続する技術。

SCSI デバイスに対応した iSCSI[19][22] (Internet Small Computer System Interface)や USB デバイスに対応した USB over IP 等がある。

図 9 に本技術の通信レイヤを示す。

上位層であるアプリケーションからは SCSI または USB デバイスとして認識できるため、既存の該当インターフェイス向けのアプリケーションを実行することができる。

API レベルで対応しているため、上位レイヤでのチューニングを行うことが可能である。

問題として下記の点があげられる

1. 対応させるインターフェイス毎にソフトウェアドライバやプロトコル変換を行うブリッジが必要。
2. 電気的なリピータと異なりインターフェイスレベルでの仕様を完全に満たすわけではないので、対応するインターフェイスであっても対応できないデバイスが存在する。

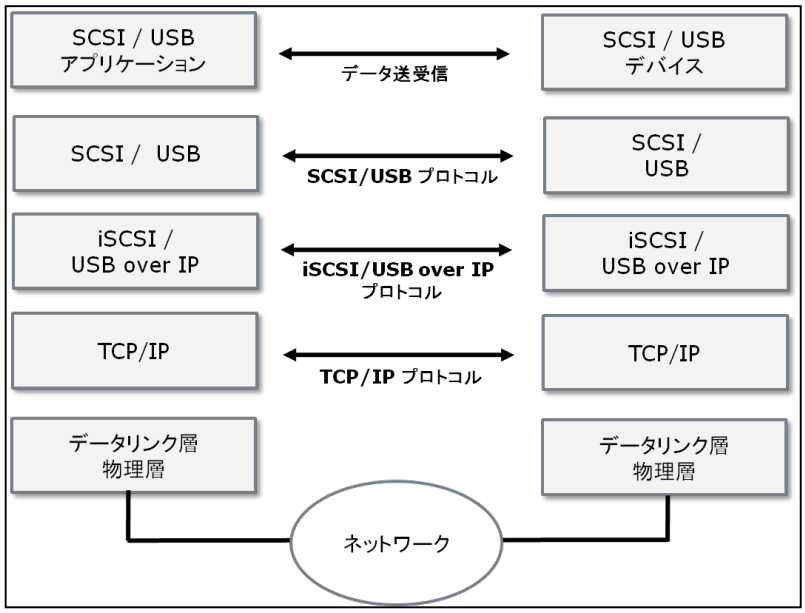


図 9：Peripheral over IP の通信レイヤ

2.8 必要機能要件と既存研究の対応状況

前節までは、既存技術を本研究に適用することについて検討した。

本節では本研究の必要機能件を下記にあげ、既存技術の対応状況を表 4 に示す。

1) CPU 増設ができるか
2) I/O 増設ができるか
3) メモリ増設ができるか
4) IP ネットワーク対応
5) 共有／占有の可否

InfiniBand が IP ネットワーク対応以外の項目を除き、多くの要件を満たした。

しかし、バスにおける距離の制限を取り除くために、IP ネットワークへの対応が必要であることと、HPCC 用のハードウェア設計が前提であるため、本研究のベースとして利用せず、別の手法を検討する。

表 4：既存技術との比較

	接続レベル／手法	CPU増設	I/O増設	メモリ増設	IP対応	共有可否
1	Memory Interconnects	○	×	○	×	○
2	NUMA	○	×	○	×	○
3	分散メモリ	○※1	×	×	×	○
4	Chipset-level Connections	○	○	○	×	×
5	vPro	×	△※2	×	○	×
6	Peripheral buses	×	○	○	×	×
7	PCI, PCI Express	×	○	○	×	×
8	InfiniBand	○	○	○	×	○※3
9	RDMA	×	×	×	○	—
10	ExpEther	×	○※4	○	△※5	×
11	Microcode-level connections	—	—	—	—	—
12	Virtualization	×	○	○	○	○
13	IP-level connnections	×	○※4	×	○	×
14	iSCSI/USB over IP	×	○	×	○	△

※1 メッセージングパッシング対応アプリケーションが必要

※2 クライアント管理用のシリアルコンソール機能が利用できる

※3 専用のハード設計が必要

※4 サポートできないデバイスもある

※5 Ethernet Frame に対応

第3章　IP バスブリッジ

本章では２章までに述べた問題意識に基づき、提案する IP バスブリッジの設計について述べる。

3.1　ネットワークセントリック・コンピュータアーキテクチャ

現状のコンピュータアーキテクチャは CPU を中心にメモリ、入力回路、出力回路が接続され、DMA など一部を除き処理は CPU を経由する。本研究ではこのように CPU を中心としたコンピュータアーキテクチャを CPU セントリックと定義し、図 10 に概念図を示す。

CPU セントリック・コンピュータアーキテクチャにおいて、LAN 装置は CPU の周辺ペリフェラル装置である。ネットワーク通信をおこなうためには LAN 装置単独では機能できず、CPU およびオペレーティングシステムとネットワーク、アプリケーションレイヤのソフトウェアが必要になり、CPU を経由したデータ転送が頻繁に発生する。

例えば図 10 にある Node-1 の CPU が Node-2 の入力回路 2A の情報を参照する場合、CPU と入力回路は直接通信することができない。赤色で示された周辺の回路間で頻繁にアクセスが発生し、多くのハードウェア資源が消費されることになる。

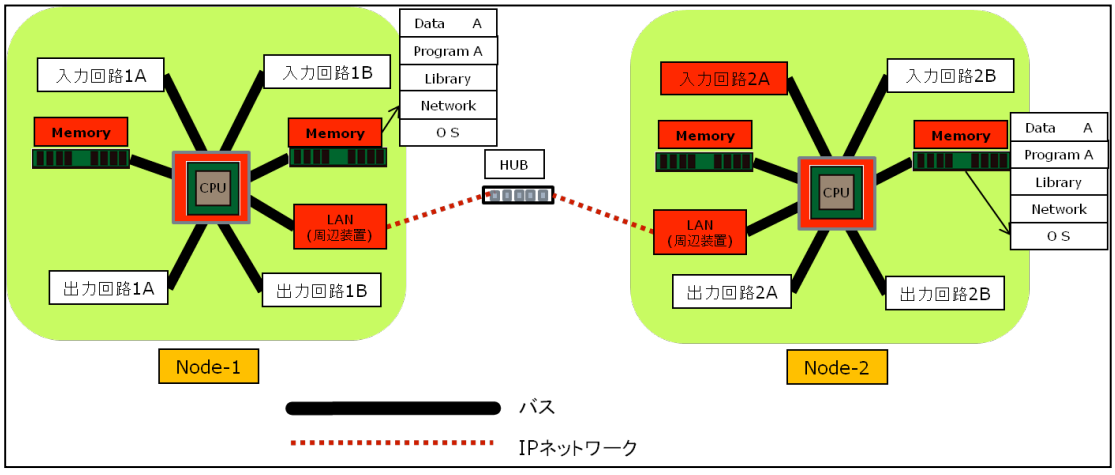


図 10：CPU セントリック・コンピュータアーキテクチャ概念図

次にネットワークを中心としたネットワークセントリック・アーキテクチャについて提

案する。

本アーキテクチャではバスは既存ネットワークを用いてグローバルかつフレキシブな接続が可能である。バスを通じた回路の接続を固定ではなく、CPU や外部ネットワークからの要求をもとに回路同士を接続できるようにバスネットワークを構築する。

LAN 装置はバスと LAN ネットワークのバスブリッジとして機能する。これにより、回路からみると別ネットワーク上にあるコンピュータバスもローカル上にあるコンピュータバスと同様に接続することができる。このときの概念図を図 11 に示す。

Node-1 の CPU が Node-2 の入力回路 2A につながっているセンサーの情報を参照する場合、２つのコンピュータバスは LAN 装置によってブリッジ接続される。

Node-1 の CPU と入力回路 2A は Node-2 の CPU を一切介さず直接アクセスすることでハードウェア資源の消費量を下げるとともにバスブリッジにより転送パフォーマンスをあげることが可能である。この時に資源を消費する回路を赤色で示す。

本アーキテクチャでは、イーサーネットやＩＰ技術のアドバンテージにより別ネットワークにある回路同士でもバスブリッジを介して直接接続できる。

ネットワークの特性を考慮する必要があるが、このことは目的に応じて効率を考慮した回路接続ができることをあらわし、ハードウェア構成のフレキシビリティ向上・回路のスケーラビリティ改善による性能向上、デバッグ・メンテナンス性の改良が期待できる。

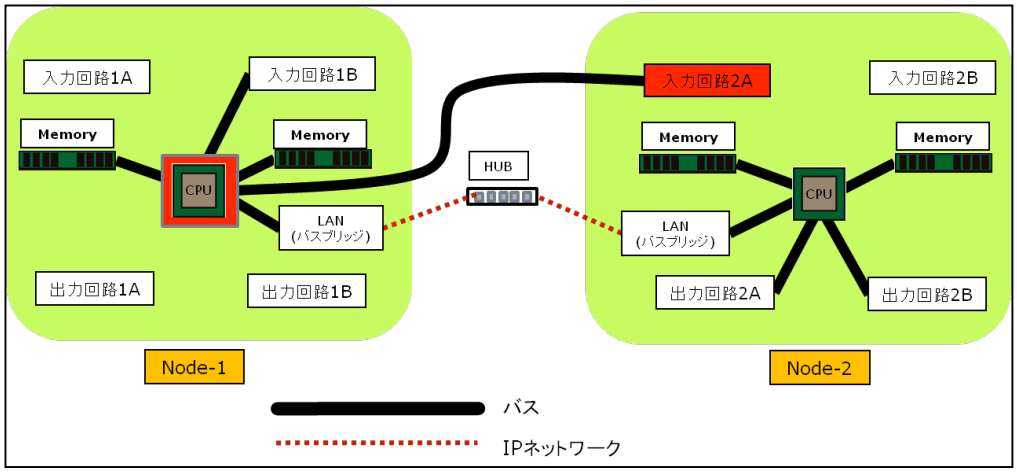


図 11：ネットワークセントリック・コンピュータアーキテクチャ概念図

3.2

3.3 Use Cases

本節ではネットワークセントリック・コンピュータアーキテクチャが実現されることに

よって実現できる活用例について述べる。

本アーキテクチャの特徴を下記にあげる。

・コンピュータバスのグローバル相互接続

・オンデマンドな回路構成変更

・ソフトウェアとハードウェアのダイレクト接続

・異なるクロック回路の混在利用

コンピュータバスのグローバル相互接続は、本機構のブリッジ回路によりネットワークを用いてバスを延長、拡張することにより実現できる。ネットワークによるバス延長の利用を図 12 に示す。

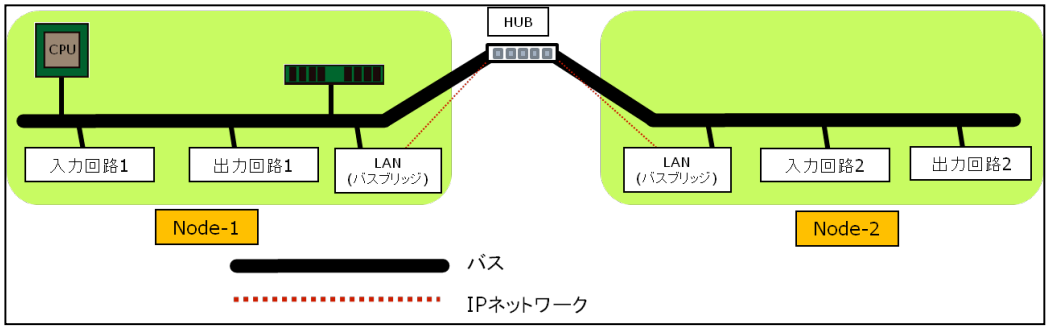


図 12：ネットワークによるバス延長

図 13 に CPU セントリック・コンピュータアーキテクチャによる分散処理の例をあらわす。Node-1 と Node-2 は NORA (No Remote Memory Access)アーキテクチャ型で、Message Passing により並列処理を行う。

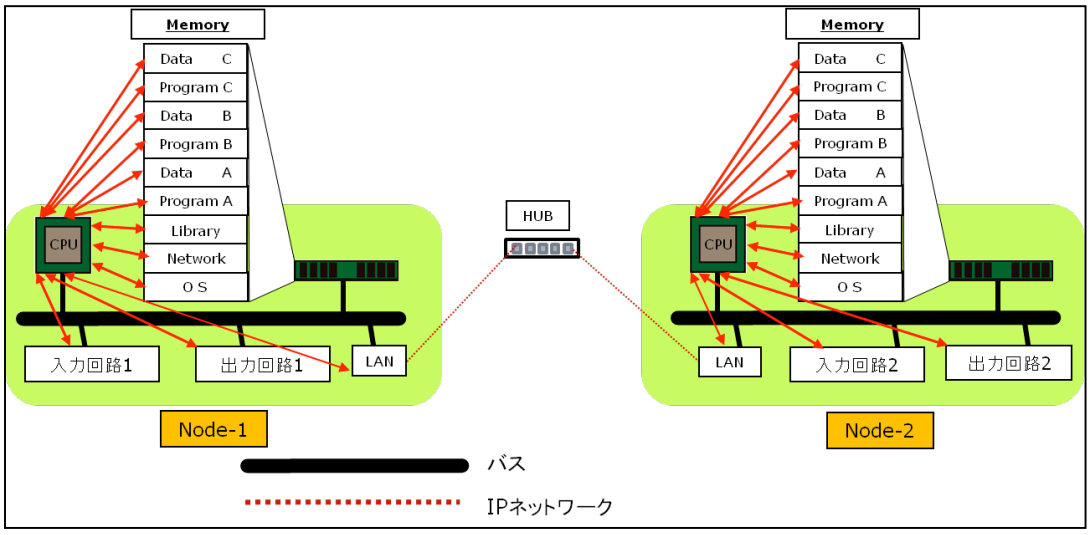


図 13：CPU セントリック・コンピュータアーキテクチャによる分散処理

ネットワークセントリック・コンピュータアーキテクチャの場合、必要に応じて NORA の構成から NUMA の構成へ回路を切り替えることができる。こうすることで、ローカルメモリを大容量キャッシュとした分散共有メモリ構成となり、ネットワーク回線におけるボトルネックを削減できる。このときの構成を図 14 に示す。

CPU とメモリ間が専用バスで接続される NUMA 構成に対して本アーキテクチャで実現する NUMA を IP-NUMA と定義する。

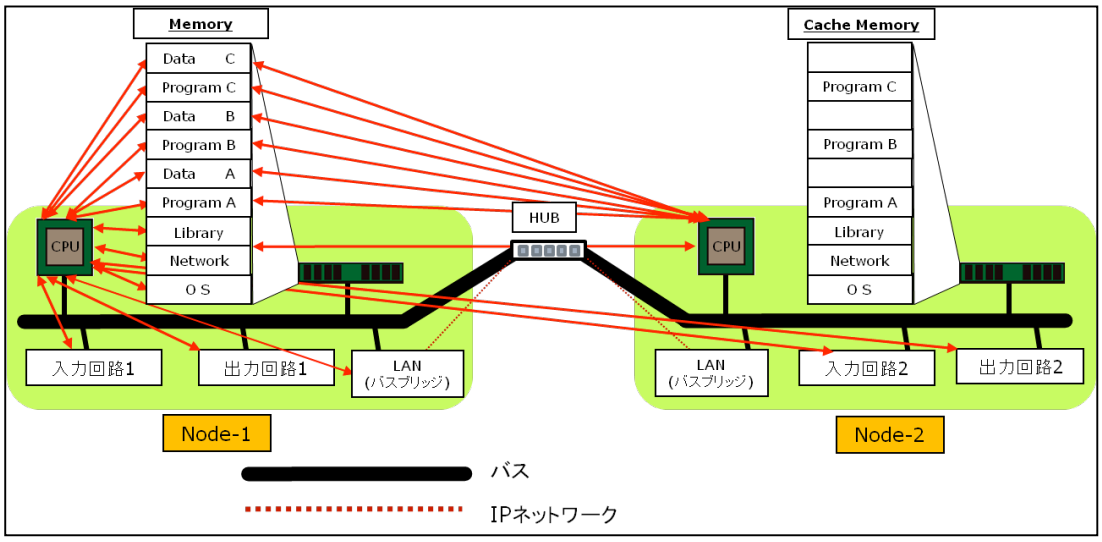


図 14：IP-NUMA ON DEMAND

ネットワークセントリック・コンピュータアーキテクチャでは、回路間の接続にネットワークプロトコルを利用するため、ソフトウェアとハードウェア回路をダイレクトに接続

することができる。

ソフトウェアからの出力を直接ハードウェア回路で処理することで、ソフトウェア同士のプロセス間通信などと比べて処理能力を向上することが可能である。

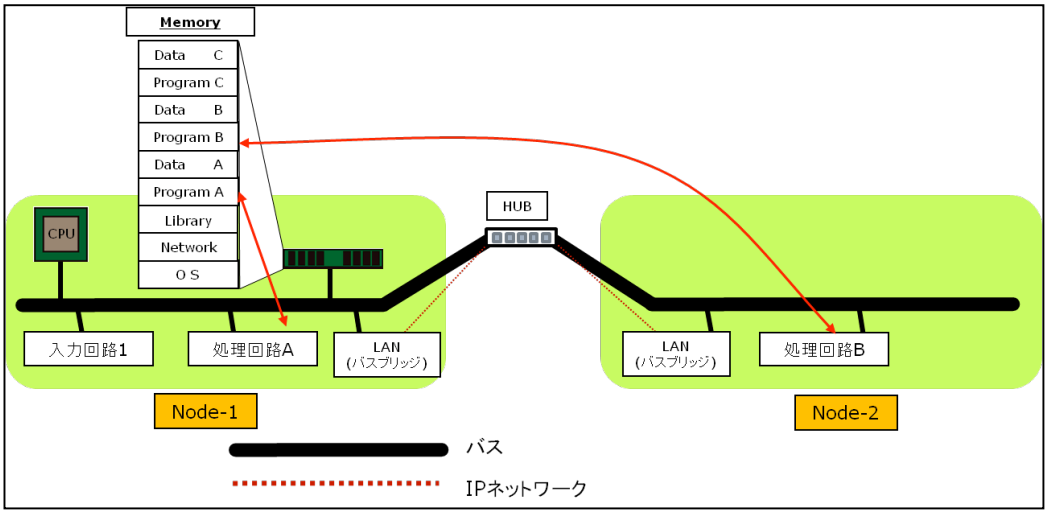


図 15：ソフトウェアとハードウェアのダイレクト接続

図 16 で示すように本アーキテクチャでは大幅に異なるクロック環境が混在する環境でも利用できる。高速な共有メモリアクセスが必要なハードウェア資源は近距離ネットワーク上の資源を利用し、低速でも構わない I/O 資源は遠距離ネットワークの資源を活用するなどが可能である。

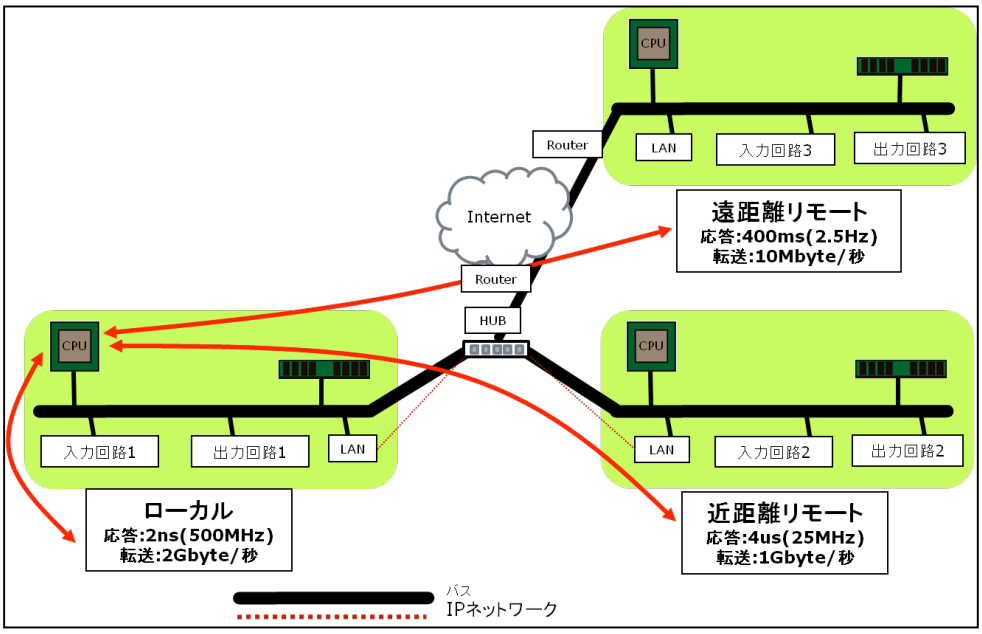


図 16：様々な周波数が混在するコンピュータシステム

3.4　設計要件

本研究での目的は、コンピュータバスの制限を取り除き、グローバルでかつ柔軟性のある接続がおこなえるバス接続の実現である。

下記の点に留意し設計を行う。

1.グローバル接続
2.透過的アクセス
3.低遅延
4.リモート接続におけるメモリ遅延対策
5.柔軟性のある回路接続
6.非オーナーシップ
7.独自バス

1．グローバル接続

コンピュータバスには距離や接続数の制限がある。本機構ではバス信号を既存の通信ネットワークプロトコルに変換する機能をもつことによりグローバル接続を実現する。

2．透過的アクセス

接続先の回路がローカル上にある場合でも、ネットワークを超えたリモート上にある場合でも意識することなく透過的に利用できる必要がある。また、回路が直接通信ネットワークプロトコルで接続することは回路規模の増加や互換性の低下につながるため、回路間にブリッジ機構を設けることにより実現する。

3．低遅延

リモート回路への接続ではネットワークプロトコルへ変換されるため遅延の増加がみこまれる。本機構では低遅延化を実現するために、ネットワークプロトコルへの変換にソフトウェア処理は一切おこなわず、電子回路のデータフロー処理によってネットワーク処理を実現できる専用の LAN コントローラを設計する。

4．リモート接続におけるメモリ遅延対策

メモリ回路がリモートにある場合において、ワード単位でネットワークを経由して頻繁にアクセスを行うと遅延の影響は大きく、使用用途によっては現実的ではない。

本機構では、ブリッジ内にキャッシュを設計することで実メモリとのアクセス回数を極力減らし、遅延の問題を軽減化させる。

5. 柔軟性のある回路接続

通常、バスに接続される回路は同一バス内の CPU やメモリとしか接続が行われることがなく、ハードウェアの構成変更がない限り接続先は固定となる。本設計ではネットワークを経由して様々な回路と動的に接続するための機構が必要になる。

6. 非オーナーシップ

本ブリッジで接続した回路は、物理的な場所による優先度は設けない。ローカルからのアクセスおよびネットワーク越しでのリモートアクセスでも同等にあつかう。

7. 独自バス

本研究の評価には、独自バスアーキテクチャを持つコンピュータシステムを使うため、汎用の PC バスには対応せず、独自バス仕様とする。

3.4.1 設計の概要

前節の設計方針に基づいてバスとデバイス回路の間にブリッジ回路を設計した場合の概要を図 17 に示す。

CPU またはデバイスからの要求は必ずブリッジを経由する。このとき、ローカルに存在する回路か、リモートに存在する回路かの判定は要求されたアドレス値に基づいて TLB[3][6][10]内のテーブルによって行う。

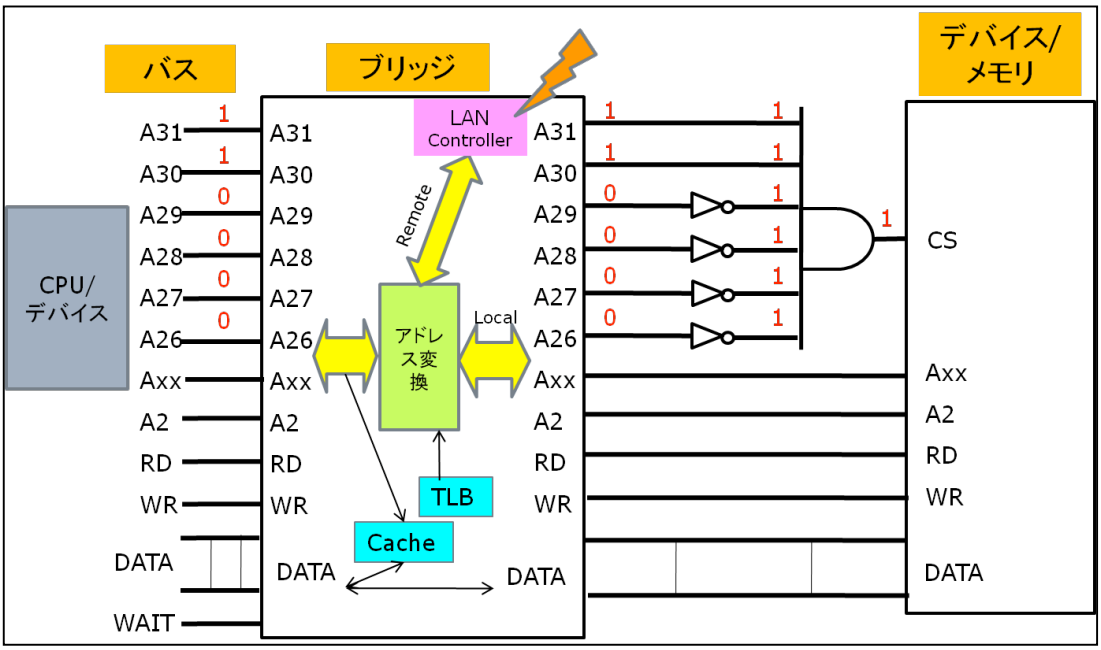


図 17：ブリッジ回路

CPU がブリッジを経由してローカル回路へアクセスするケースについて図 18 を用いて解説する。

1. CPU がブリッジのバスに対して、アクセスしたいアドレスを出力し、データ要求を行う。

2. ブリッジは CPU からの要求を受け取り、要求したアドレスが TLB 内とメモリキャッシュ内でヒットするかを同時に参照する。

3. TLB で照合した結果、該当アドレスの回路はローカルであったと判断された場合、ローカル回路のバスに対して要求を行う。アドレスは TLB によって変換された物理アドレスを使用する。

4. ローカル回路から応答されたデータをブリッジが受け取る。

5. 応答されたデータを CPU のバスに対して応答する。

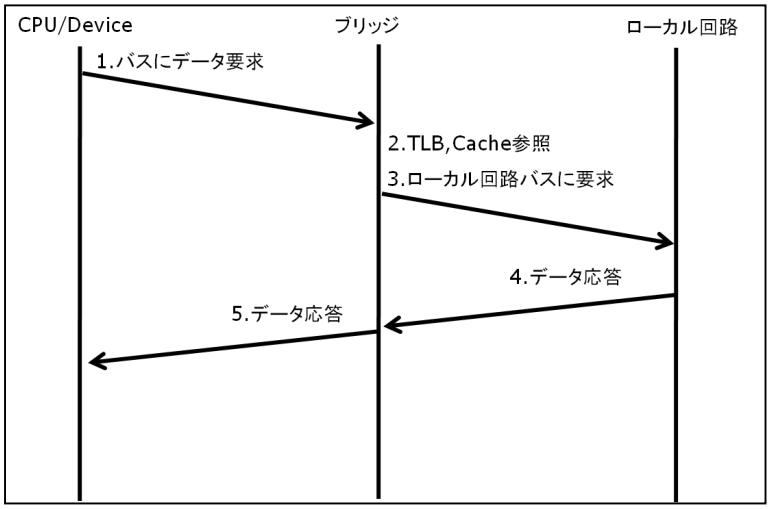


図 18：TLB がローカル回路を示す場合

次に CPU がブリッジを経由してリモート回路へアクセスするケースについて図 19 を用いて解説する。

1. CPU がブリッジのバスに対して、アクセスしたいアドレスを出力し、データ要求を行う。

2. ブリッジは CPU からの要求を受け取り、要求したアドレスが TLB 内とメモリキャッシュ内でヒットするかを同時に参照する。

3. TLB へヒットした結果、該当アドレスの回路がリモートあったと判断された場合、CPU のバスに対してウェイト要求を発行する。

4. LAN コントローラ回路へデータ要求を行う。LAN コントローラには TLB 上のホスト ID と物理アドレスを送信する。

5. LAN コントローラ回路から応答されたデータをブリッジが受け取る。

6. 応答されたデータを CPU のバスに対して応答する。

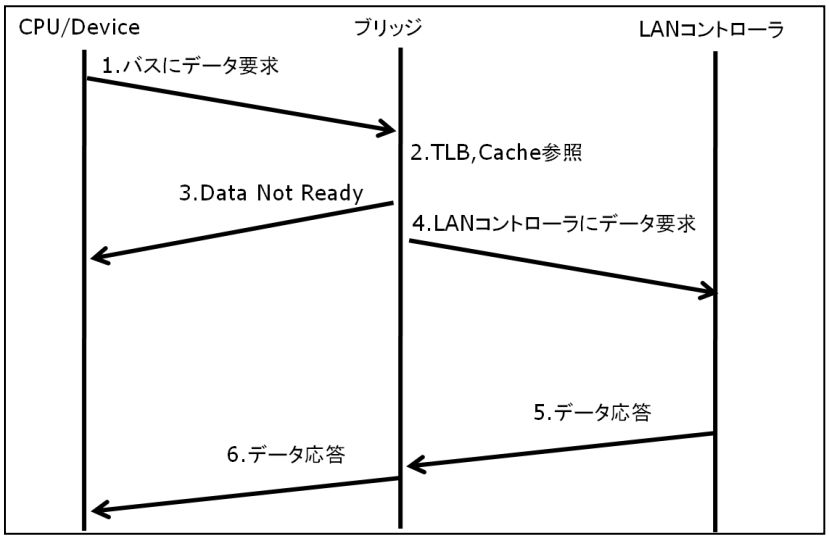


図 19：TLB がリモート回路を示す場合

3.5 バスの設計

本ブリッジにはバスの調停回路を設けず、バス要求を行うデバイス毎にバスを占有させる。

図 20 ではハーバードアーキテクチャの CPU と Memory および、バスマスタになりうるデバイスがふたつ存在する回路を本ブリッジで接続した場合の接続例を示す。

ハーバードアーキテクチャの CPU はインストラクション用のバスとデータ用バスが必要になるため、バス 1 とバス 2 を割り当てる。メモリ回路はバス 3 に割り当て、バス要求を行うふたつのデバイスをバス 4 とバス 5 に割り当てる。

バス要求を行わないデバイスの場合、デコーダ回路を用意してひとつのバスに複数接続する。

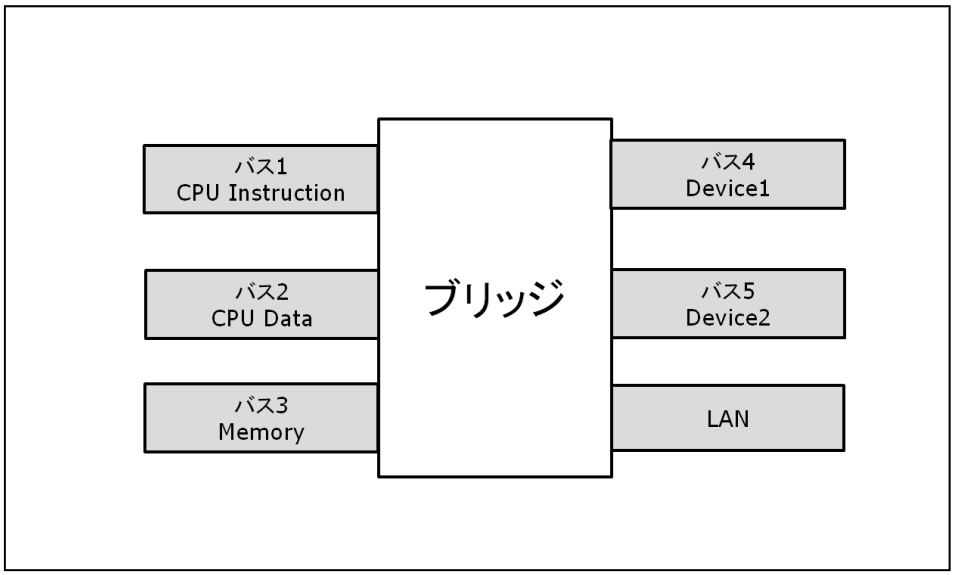


図 20：バス接続例

3.5.1 信号

本設計で想定するバス内の信号を下記に示す。

・CLOCK

クロック

・ADDR

１６～３２ビットのアドレス信号

・DATA

データ信号

・RD

読み込み要求信号

・WR

書き込み要求信号

・BUSREQ

デバイスのバスの使用要求

・BUSACK

BUSREQ に対する使用許可

・~READY (WAIT)

データ準備ができたことを示す

3.5.2 バスの種類

本設計では4種類のバスを想定している。

・CPU Instruction バス

CPU のインストラクションを読み込む。リードオンリー。

・CPU Data バス

CPU のデータ読み込み、書き込み要求。

・メモリバス

非同期メモリ用バス。

・デバイスバス

スレーブまたはマスタデバイス用。

バスの種類毎に割り当てる信号の対応を表 5 に示す。

表 5：信号とバスの対応

信号名	用途	ビット幅	CPU I バス	CPU D バス	メモリバス	デバイスバス
CLOCK	クロック	1	○ 同期	○ 同期	× 非同期	○ 同期
ADDR	アドレス	16~32	○	○	○	○
DATA	データ	8～32	○	○	○	○
RD	読み込み信号	1	○	○	○	○
WR	書き込み信号	1	×	○	○	○
BUSREQ	バス要求	1	×	×	×	○
BUSACK	バス要求応答	1	×	×	×	○
~READY (WAIT)	データの準備完了	1	○	○	×	○

3.6 ローカル回路とリモート回路の切り替え

本節ではローカル回路とリモート回路の判別について述べる。

本設計ではローカルおよびリモートの回路が透過的に扱われ、見掛け上の差がない。

ブリッジ内には論理メモリアドレス空間とホストアドレスおよび物理メモリアドレス空間の対応テーブルが存在し、これらの組み合わせにより回路が選定される。

図 21 に論理メモリアドレスと物理メモリアドレスの設定例を示す。

Node1 と Node2 はそれぞれ別ネットワーク上にある本機構のブリッジで接続された回路の集合をあらわす。

Node1 の論理メモリアドレス空間の$0000000 番地からはローカル回路上のメモリに割り当てられ、$100000 番地以降は Node2 のリモート回路上のメモリとデバイスが割り当てられ接続されている。

論理アドレスと物理アドレスの対応テーブルは CPU などのバスマスタになりうるデバイス、またはネットワーク経由の指示によって書き換えられる。テーブル形式の詳細については 4.6.1 節で述べる。

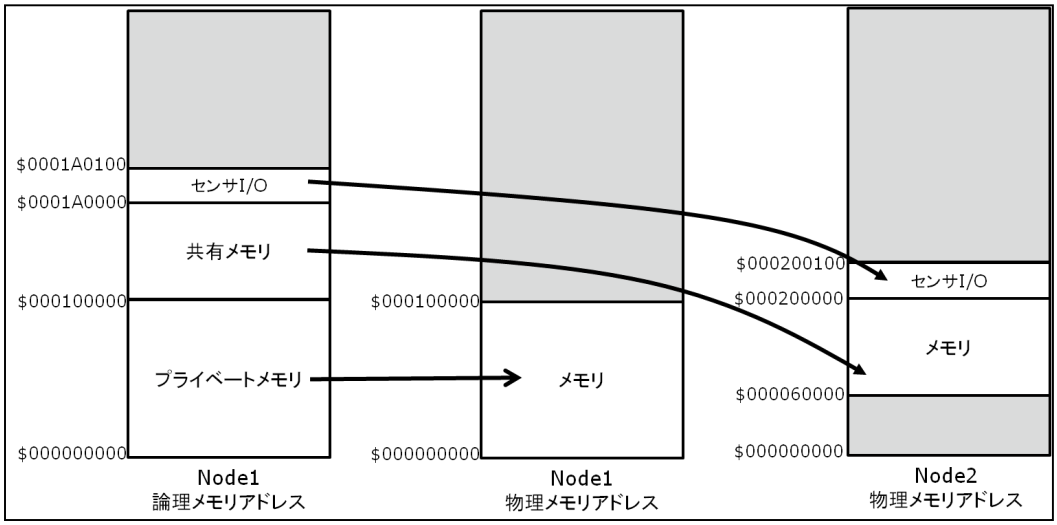


図 21：論理メモリアドレスと物理メモリアドレス

3.7 プロトコル

本節では、ネットワークプロトコルに関して検討する。

Ethernet Frame と比べて IP はフレームヘッダが大きく、遅延面において不利だと考えられる。

実際にフレーム形式を比較するため 100Base-T における Ethernet フレーム、IPv4 UDP、IPv6 UDP と参考のため PC バスの PCI-Express フレームの形式を図 22 に示す。データ部のサイズは本実装の最小データ長である 12 バイトとした。

フレーム長が短いほうが遅延を削減できるが、いずれも Ethernet Frame 先頭のプレアンブル部が多く、フレームヘッダが短い Ethernet Frame と IPv4 UDP での差は 2 倍もない。

また、PC バスとして広く使われている PCI Express と比べても IPv4 のフレームサイズ

は 2.3 倍程度の大きさである。データ長が増えた場合は、差はさらに少なくなるので GB Ethernet や 10GB Ethernet を使用した場合における、IP のバス利用は速度面においても十分可能であると考えられる。参考までに PCI Express×1 の速度は片方向通信 2.5Gbps (250MB/s)である。

以上を踏まえ、いずれのプロトコルも差異が多くないため、ここにあげた Ethernet Frame、IPv4 UDP、IPv6 UDP は本評価で利用し比較の対象とする。

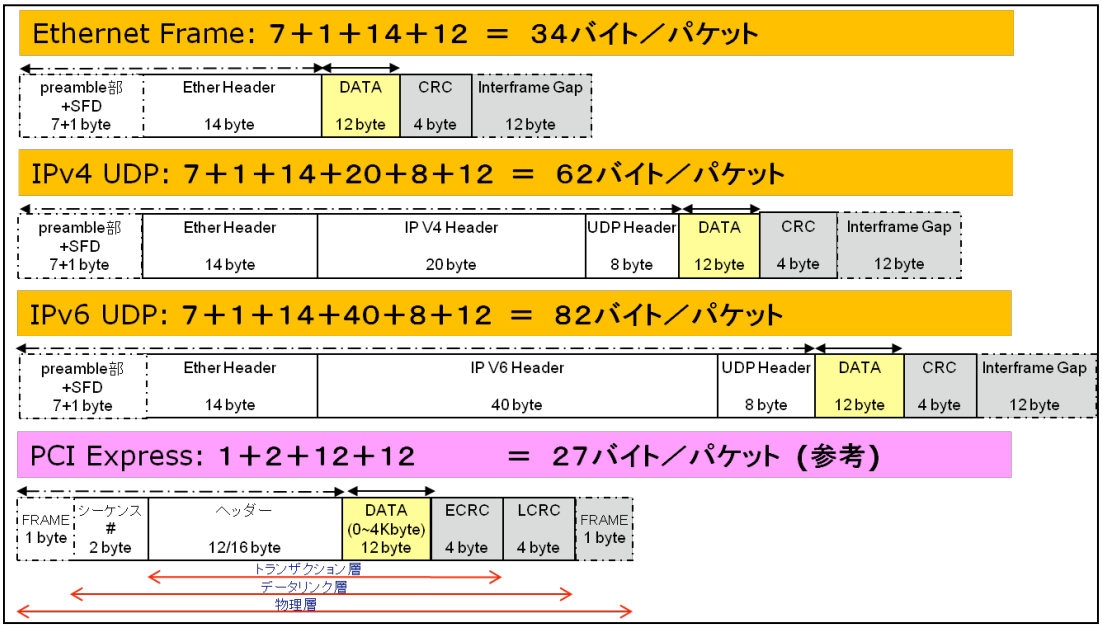


図 22：プロトコル比較

3.8 詳細設計

3.8.1 リモート回路の低遅延機構

本節ではブリッジとリモート回路とのアクセス遅延の軽減化を考慮した設計について述べる。

CPU またはデバイスからの要求がリモート回路あての場合、速やかに要求パケットをネットワーク上に送信させることが望ましい。

本研究では専用の LAN コントローラを設計することによって遅延の軽減化を図る。

・LAN コントローラ送信 FIFO

LAN コントローラには一時的な送信バッファと受信バッファが必要になるが、本回路では外部メモリを使用しない。送信と受信用の非同期型の FIFO を内部にもつことにより高

速化をはかり、また異なるクロックを持つコンピュータ内と外部 LAN 間での高速非同期通信をおこなう。

次に送信 FIFO 内のデータの流れを図 23 をもとに解説する。

1. ブリッジがリモート回路よりアクセス要求を受け付ける。
2. ブリッジは最初のクロックで送信 FIFO に$101 を書き込む。この 9bit コードはフレームのプリアンプル部生成の指示を意味する。
3. クロックに応じて順次、データリンクレイヤ、IP レイヤ、UDP レイヤのデータを FIFO に書き込む
4. CRC32 の作成指示である$103 を書き込む
5. フレームの終了指示である$102 を書き込み、送信処理を終える。

また、送信 FIFO のデータは”9B/8B”に読み出され 8 ビットに変換、$100 以上の 9 ビットで指定されるコントロールコードは該当するデータに置き換わり、ニブル単位に変換され、Ethernet の電気信号レベルで必要なエンコード処理を経て RJ-45 インターフェイスに送信される。

これらがパイプライン処理されることによって、リモート回路への要求発生から少ないクロック数でネットワークへの送信が可能になる。

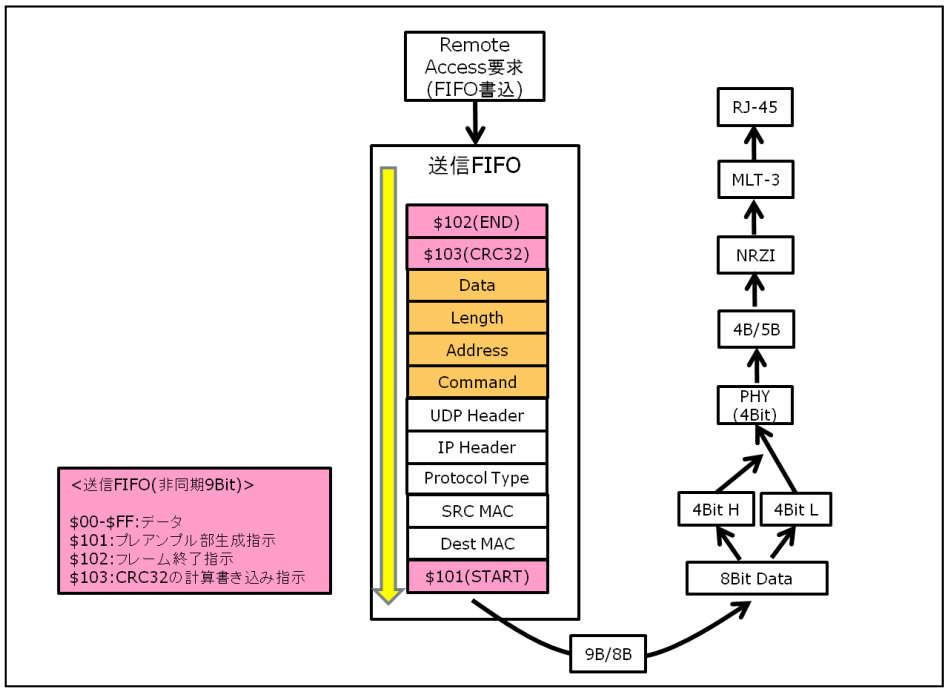


図 23：送信 FIFO フロー

・LAN コントローラ

LAN コントローラはコマンド系とサービス系の処理を持つ。

コマンド系は、ブリッジ回路からリモート回路への要求や応答パケットが流れ、サービス系では、ネットワークから要求される自分の回路リソースに対しての要求や応答パケット[7]が処理される。それぞれに送信用の FIFO と受信用の FIFO が用意される。図 24 にその様子を示す。

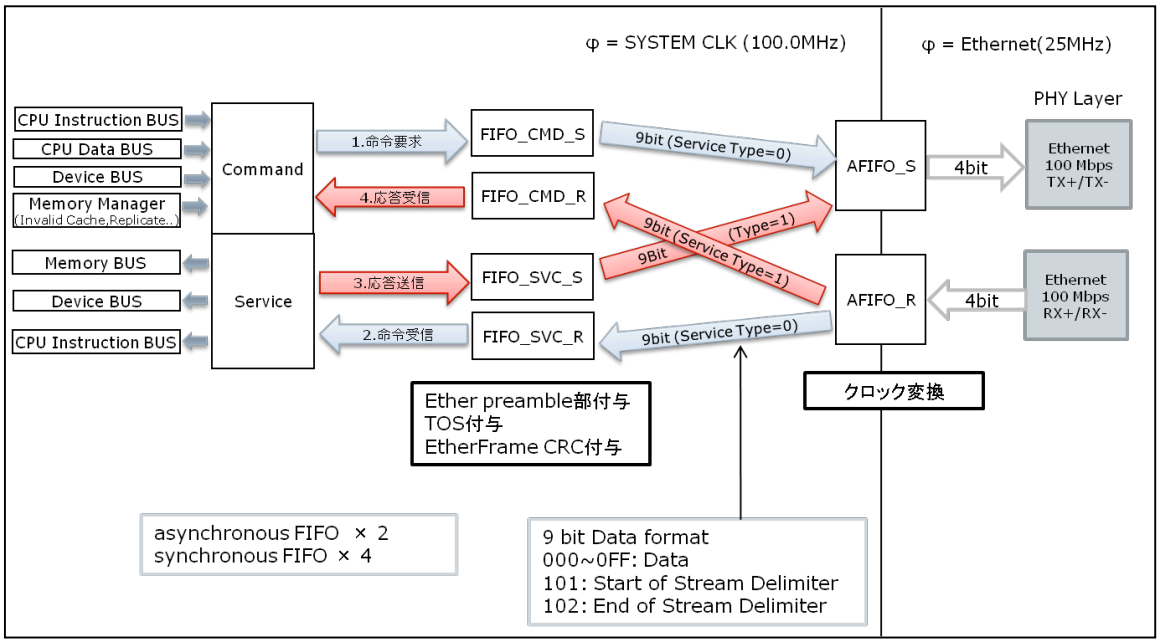


図 24：LAN コントローラ回路

3.8.2 ローカル回路の低遅延機構

本節では、ブリッジ回路とローカル回路間の信号遅延を低減化することについて検討する。

図 25 にブリッジを経由せず、バスに直接接続されている回路間での読み込みのタイミング例を示す。最初のクロックで読み込みを行うアドレスの設定を行い、次のクロックで該当するデータが準備され読み込みを完了する。

本機構では、回路同士の接続には必ずブリッジを経由する。また、アドレスは論理アドレスであり、実アクセスをするためには論理アドレスから物理アドレスへの変換が必要になり、その時点ではじめて該当する回路の存在するバスが決定される。

論理アドレスから物理アドレスの変換は、TLB 回路によって行われる。ローカル回路と

接続する前に TLB 回路に 1 サイクルが消費されることは望ましくない。本設計では TLB とキャッシュのヒット判定回路を組み合わせ回路で実行し、半サイクル以下で実行することにより、物理アドレスへの変換とローカル回路が存在するバスへの要求転送を最小化する。

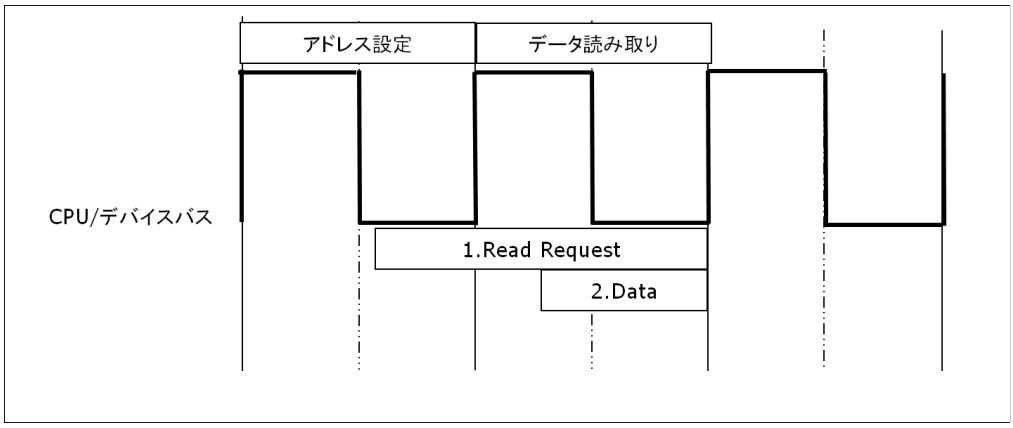


図 25：ブリッジを経由しない読み込みのバスタイミング

図 26 に本ブリッジを経由してバス要求が該当回路の存在するバスへ転送される様子を示す。

1. バスマスタデバイスがブリッジに対して読み込み要求を行う。

2.ブリッジは TLB 回路とキャッシュヒット回路を同時に使い、物理アドレスへの変換と、キャッシュバッファの存在の有無を調べる。

3. 該当回路が存在するバスに対して、物理アドレスを指定して読み込み要求を行う。

4. 回路からのデータを取得する。

5. 4.で取得したデータをバスマスタデバイスがあるバスへ転送する。

回路のデータ準備が間に合わない場合は、バスの~READY(WAIT)信号をアサートする。

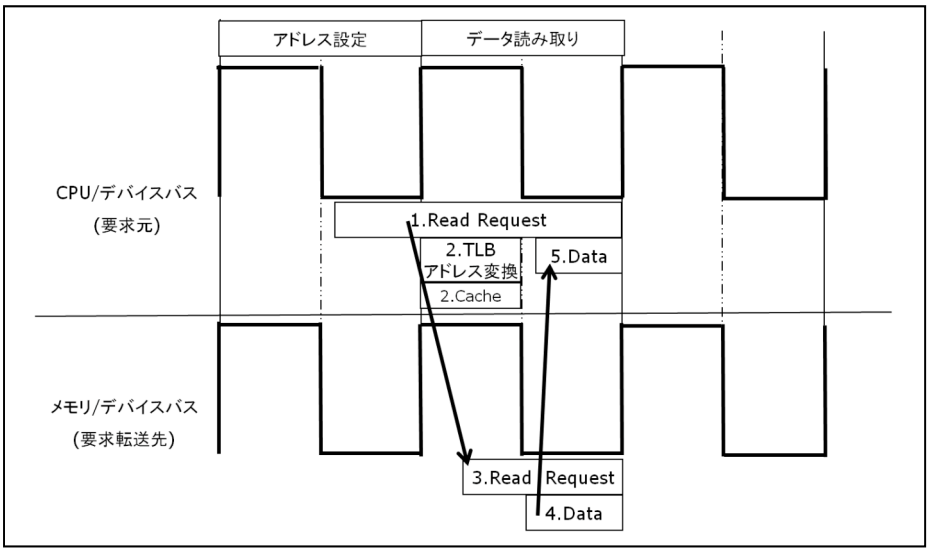


図 26：ローカル回路における読み込み要求転送のバスタイミング

3.8.3 高遅延対策

本節では、遅延が多い環境においてリモートのメモリ回路へのアクセスを低減化するための、キャッシュ機構について述べる。

ブリッジにキャッシュを内蔵させることにより、リモートメモリを利用する CPU やデバイスからキャッシュを利用することができる。

キャッシュの設計要素として WAY 数、1 ラインあたりのバイト数、Index ビット長がある。

WAY 数は同じ Index 値になるアドレスを識別できる数である。例えば 1 WAY 方式の場合アドレス$0000 と$1000 を区別できないため$1000 のアドレスをアクセスすると$0000 用のキャッシュが無効になってしまう。4 WAY 方式では$0000、$1000、$2000、$3000 を区別することができこれらのメモリ空間をアクセスしてもそれぞれのキャッシュは有効である。

WAY 数は増える毎に比較回路が必要になり、関連する回路の規模も大きくなるため設計上のポイントになる。本設計では 2 WAY とし、Index 値と Tag 値が同じになった場合は LRU (Least Recently Used)アルゴリズムにより利用が古い WAY を上書きする。

1 ラインあたりのバイト数はキャッシュの記憶単位で、ミスキャッシュ時のメモリからの読み込み単位でもある。CPU の命令長によって決定されることが多い。本設計では命令長が 4 バイト(32bit)の CPU を評価に用いるため、キャッシュラインを 64 バイトとし 16 命令をキャッシュに格納できるようにする。

Index ビット長はキャッシュのライン数を指定するものでキャッシュ容量を決める大きな要素である。本実装では 6bit（64 ライン／Way）を指定する。

図 27 に本研究で実装する 2 ウェイセットアソシアティブ方式のキャッシュ回路を示す。

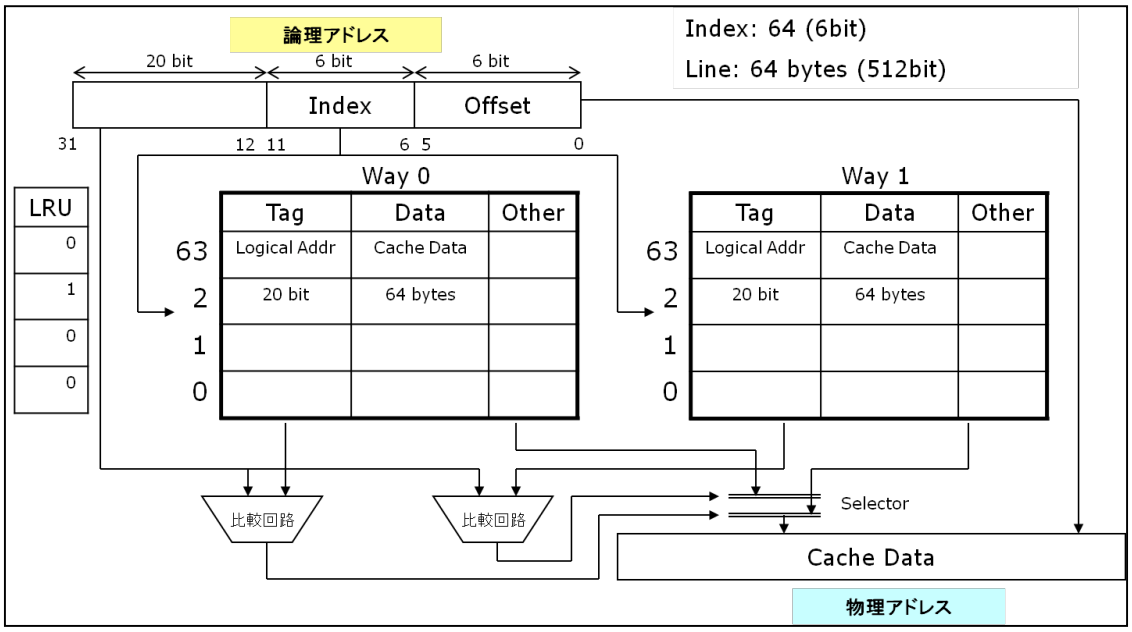


図 27：キャッシュ回路

3.8.4 動的回路変更の必要性

バスに接続されている回路はアドレスバスを使って接続先の回路を指定し、また、アドレスバスの値を参照して自分の回路あての通信かを判断し回路間のネットワークを機能させている。クローズドなバス内での回路同士接続では問題ないが、本研究が提案するネットワークを経由したバス接続が行なう場合、リモート回路を指定する手法が必要になる。

本研究ではブリッジ内に論理アドレスに対してホストアドレスと物理アドレスを対応させた TLB (Translation Lookaside Buffer)を設けることにより、リモート回路と接続する手法を提案する。

3.8.5 既存 TLB 回路における問題と本研究での提案

既存の TLB を本ブリッジ内で利用するのに考えられる問題点を下記にあげる。

・ページサイズが固定

既存の TLB はメモリの仮想化を行うことが前提のためページサイズが固定となっている。本ブリッジが対象とする回路は様々なサイズや用途のメモリや I/O デバイスを扱うのには適さない。

・I/O デバイス未対応

アドレスの範囲がメモリか I/O か識別できない。

・ネットワーク未対応

ローカルメモリにしか対応できず、ネットワーク上の他のコンピュータのメモリ空間を指定することが出来ない。

図 28 に TLB の使用例を示す。

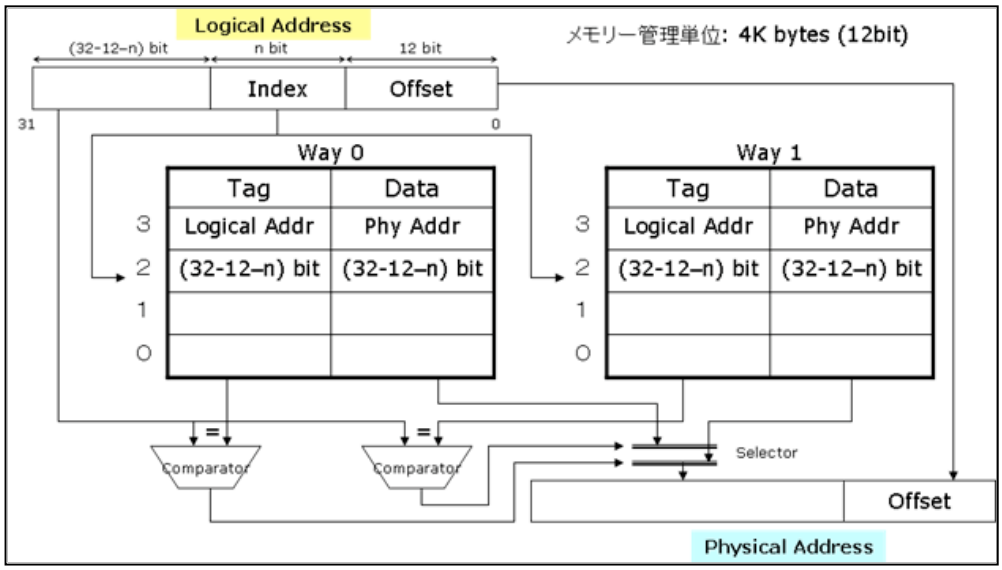


図 28：一般的な TLB（2-way set associative）

以上の問題を解決するために下記の特徴を持つ TLB の構造を図 29 に示す。フルアソシアティブ型の TLB を改良したものである。

・可変セグメンテーション

論理アドレスの開始アドレスを TLB 上の Tag フィールド、終了アドレスを Tag2 フィールドに記述することにより 1～4G バイトまでの範囲指定が可能。1 エントリーあたり 2 つのコンパレータ回路が必要になる。

・メモリ、I/O の識別

TLB 上の Type フィールドによりメモリのタイプを識別する。メモリであればキャッシュを有効にできる。

・ネットワーク対応

TLB 上の Host ID フィールドにより、該当物理アドレスの回路がローカル上に存在するか、ネットワーク上のリモートにあるかを管理する。

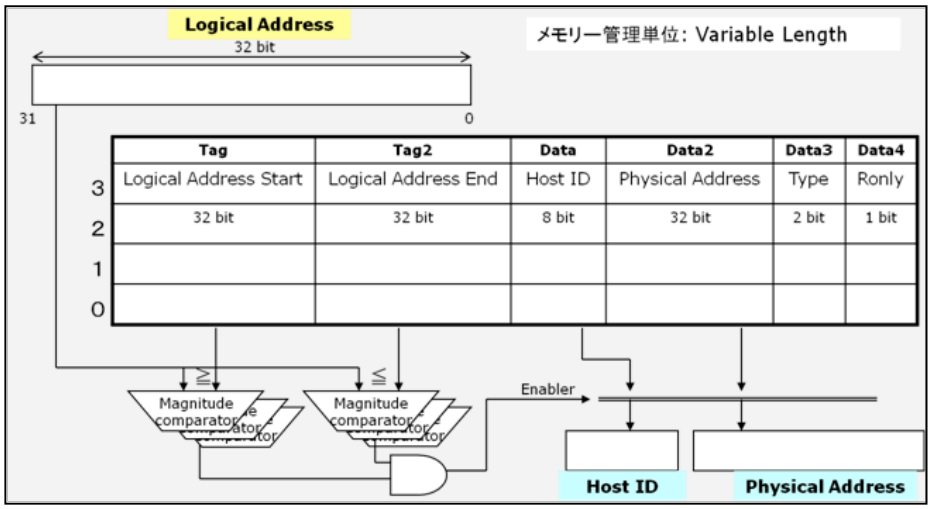


図 29：変形フルアソシアティブ TLB

3.9 本章のまとめ

本章では、ネットワークによるグローバル接続が可能なブリッジの設計について述べた。

複数バスが存在するブリッジを経由してもローカル回路同士の接続遅延が発生しにくい仕組みを設けた。また、専用 LAN コントローラを設計することによるハードウェアによるネットワーク高速通信の手法と、TLB の改良によるリモート回路との接続手法を提示した。

第4章　実装

本章では 3 章で述べた設計に基づいて、評価で用いるシステムの実装を行なう。いくつかのサブシステム毎に解説をする。

4.1　実装の概要

実装の対象は本機構を持つブリッジ回路と評価で用いる CPU、周辺回路とする。
全体図の概要を図 30 に示す。

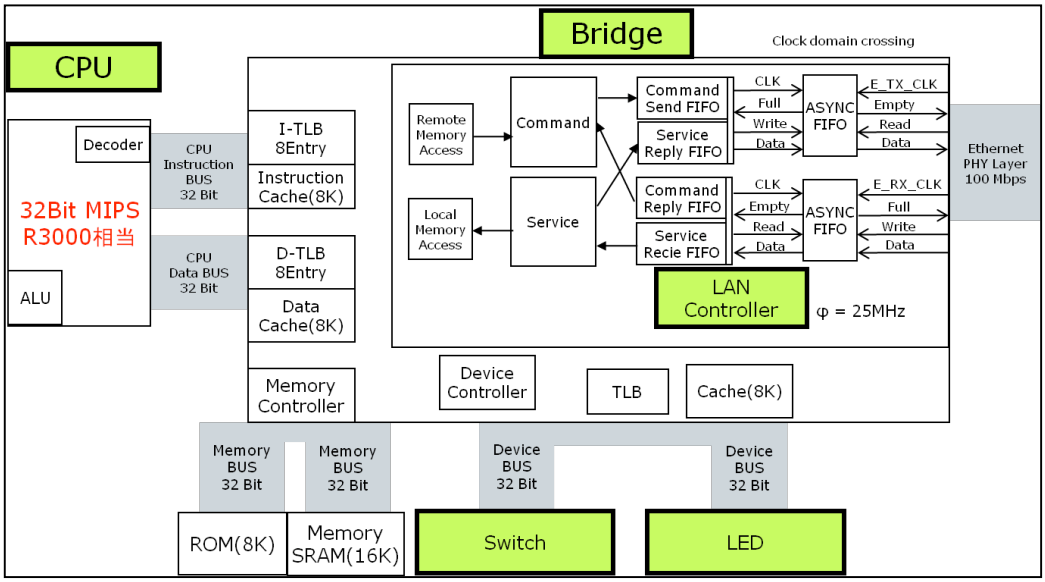


図 30：実装概要

4.2　実装環境

本節では、実装で利用するハードウェア環境および、設計・開発で用いたソフトウェアツールについて述べる。

4.2.1 実装ハードウェア

回路の設計は Verilog HDL[11][16]で記述し、Veirlog シミュレータ上で動作を確認した FPGA ボード上に実装する。

FPGA ボードの仕様を表 6 に示す。

表 6：FPGA ボード仕様

FPGA ボード名	Xilinx Spartan 3AN Starter Kit
FPGA	Spartan 3AN 75 万ゲート
ネットワークインターフェイス	RJ-45 (100Base-TX)
LAN PHY チップ	SMSC 8700
基準クロック	50MHz
画面出力	VGA 15 ピン
出力装置	8 LED, LCD, Audio
入力装置	PS/2 キーボード

4.2.2 設計・開発ツール

回路の設計ツールおよび、回路の検証プログラミング言語を表 7 に示す。

表 7：設計・開発ツール

回路記述言語	Verilog HDL
FPGA デザインツール	Xilinx ISE 10.1
Verilog HDL シミュレータ	VeritakWin
プログラム記述言語	MIPS[9]アセンブリ言語
MIPS アセンブラー	GNU as

4.3 バスの仕様

本ブリッジのバスは独自仕様である。バスマスタデバイスには専用バスを割り当てることにより、バスアービトレーション回路が不要になる。

実装するバスの仕様を表 8 に示す。

表 8：実装バスの仕様

	バスの用途	Clock	Width	同期	ADDR,DATA 以外の信号
バス 1	CPU インストラクションバス	12.5MHz	32bit	同期	Read
バス 2	CPU データバス	12.5MHz	32bit	同期	Read,Write
バス 3	メモリバス	50MHz	32bit	非同期	Read,Write
バス 4	デバイスバス	25MHz	32bit	同期	Read,Write, Ready,

4.4 バスアービトレーション

デバイスごとにバスを用意し占有させるためバスアービトレーション回路は実装しない。

4.5 メモリアービトレーション

メモリへのアクセスは、CPU インストラクション、CPU データ、デバイス、LAN からの要求の 4 種類がある。それぞれのメモリアクセスのタイミングをずらすことにより競合を回避する。図 31 にメモリアクセスのクロックタイミングを示す。

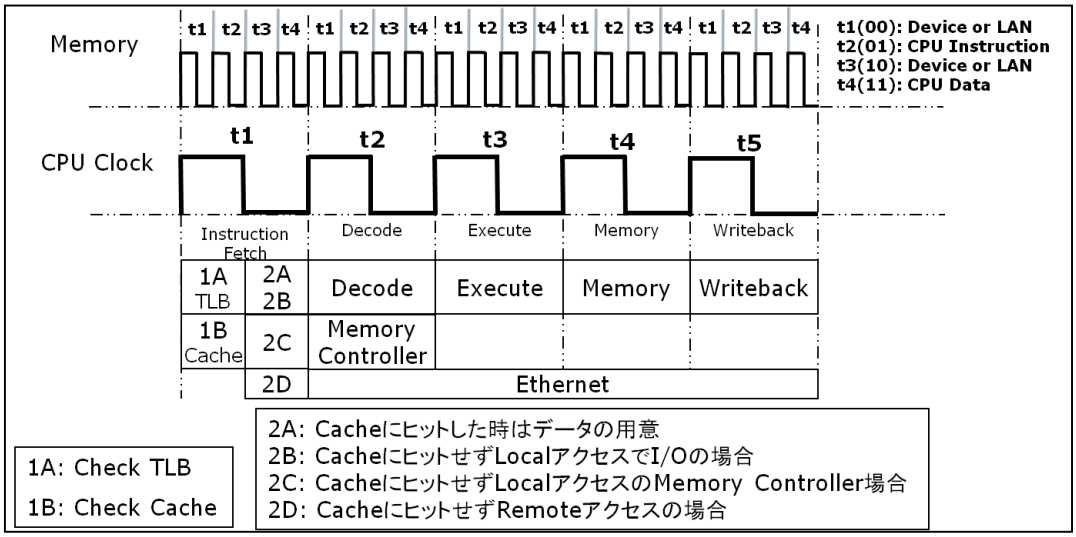


図 31：クロックタイミング

4.6 TLB 回路

TLB回路はバス上の論理アドレスに対する物理アドレスの対応を格納するTLBレジスタとTLBのヒット判定回路から成り立つ。

4.6.1 TLB レジスタ

本レジスタには論理アドレスに対応する物理アドレスやホストIDへ変換する為のテーブルおよび、メモリの属性に関する情報が記録される。表 9 に TLB レジスタについての詳細を示す。TLB レジスタの数は 2 つ以上あり、本実装においては 8 つある。

TLB レジスタはブリッジ内のレジスタメモリでしかないため、TLB への設定と参照を行うためには任意のアドレス空間上にマッピングするようにデコーダ回路を作成する。これにより CPU または外部ネットワークからメモリを通じて読み書きすることができる。

表 9：TLB レジスタ

番号	名前	ビット幅	内容
1	TLBI_Logical_Address_Start	32	論理アドレス開始
2	TLBI_Logical_Address_End	32	論理アドレス終了
3	TLBI_Machine_Address	8	ホストID(0:Local)
4	TLBI_Physical_Address	32	物理アドレス
5	TLBI_Location	4	0:Memory 1:LAN 2:CPU 3:DEVICE
6	TLBI_Cache_Enable	1	1:キャッシュが有効
7	TLBI_Read_Only	1	1:読み込みのみ可能

4.6.2 TLB 回路

本回路は、バスより論理アドレスへの要求があった場合、ヒットする TLB レジスタ番号を調べる。Verilog HDL による回路記述例を図 32 に示す。

```
wire    [`TLB_ENTRY-1:0]    TLBI_Hit;
assign  TLBI_Hit=
TLBI_Logical_Address_Start[1]<=INST_ADDR&INST_ADDR<=TLBI_Logical_Address_End[1] ? 1 : 0 |
TLBI_Logical_Address_Start[1]<=INST_ADDR&INST_ADDR<=TLBI_Logical_Address_End[2] ? 2 : 0 |
TLBI_Logical_Address_Start[1]<=INST_ADDR&INST_ADDR<=TLBI_Logical_Address_End[3] ? 3 : 0 |
TLBI_Logical_Address_Start[2]<=INST_ADDR&INST_ADDR<=TLBI_Logical_Address_End[4] ? 4 : 0 |
```

図 32：TLB 回路

4.6.3 TLB によるローカル回路とリモート回路へのアクセス

本節ではブリッジがローカル回路とリモート回路へアクセスするときの CPU、ブリッジ、メモリのタイミングについて解説する。図 33 は CPU がローカル回路へアクセスしているときのタイミングを示している。

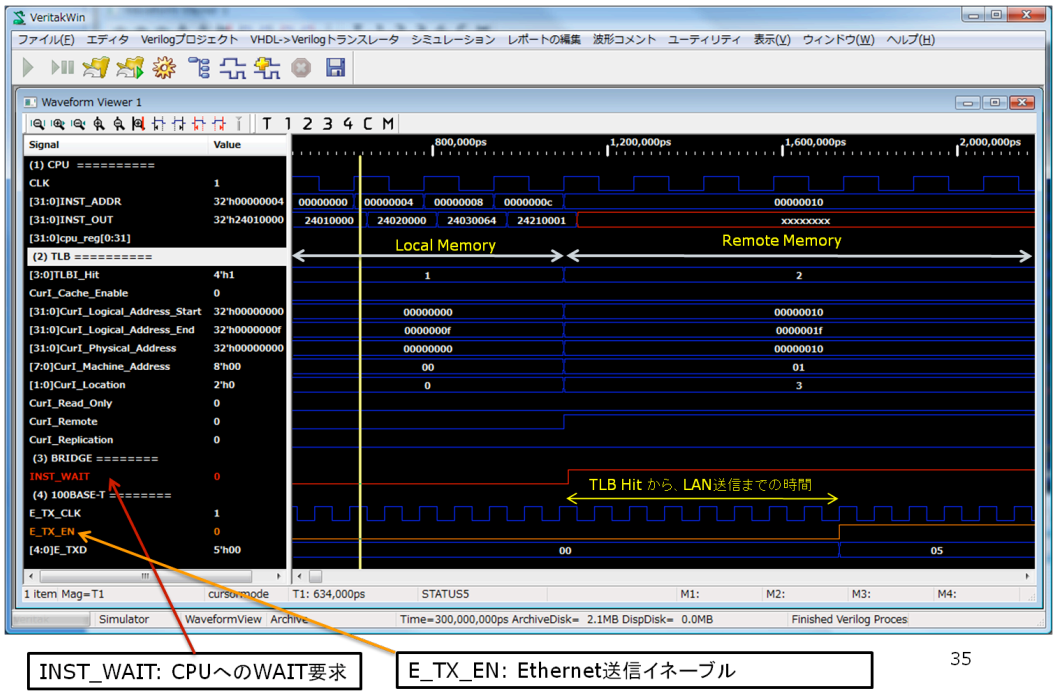
INST_ADDR が CPU のプログラムカウンタであり、メモリへの読み込み要求アドレスとなる。INST_ADDR が設定されると組み合わせ回路により TLBI_Hit に該当する論理アドレスを持つ TLB の番号が設定される。同時に CurI_Remote に該当回路がリモートに存在するかを示す。本図ではローカル回路をさすので 0 になっている。

図 33：ローカル回路へのアクセス

次に CPU がリモートメモリ回路へアクセスするときのタイミングを図 34 で示す。この例ではメモリアドレスの 0000F 番地までがローカル回路で、00010 番地以降がリモート回路上の 00010 番地へ接続されるように TLB レジスタの設定が行われている。

INST_ADDR が 00010 番地になると TLBI_Hit の値が 1 から 2 に変わり、INST_ADDR がリモート回路に接続すべきことを示す CurI_Remote が 1 になる。それとほぼ同時に CPU に対して INST_WAIT (~READY)信号を使って WAIT 要求を発生する。

間もなく LAN コントローラにより 100Base-T 回線上に要求パケットが送信され、ブリッジはリモート回路からの返信データを受け取り CPU のインストラクションバスに対しての WAIT 要求をネゲートする。



INST_WAIT: CPUへのWAIT要求

E_TX_EN: Ethernet送信イネーブル

35

図 34：リモート回路へのアクセス

4.7　キャッシュ回路レジスタ

本実装では、キャッシュサイズが少ないため、レジスタを用いてキャッシュを構成する。キャッシュレジスタの定義を図 35 に示す。

キャッシュ回路の構成を図 27 で示した。キャッシュは 1 ライン 64 バイト単位で管理しており、32bit の論理アドレスのうち下位 6 ビットは、1 ライン上のオフセットアドレスを示す。論理アドレスの第 6 ビット目から第 11 ビット目がインデックスをあらわし、キャッシュの何ライン目かをあらわす。該当するラインの CACHEI_Tag0 の内容と論理アドレスの第 12 ビット目から第 31 ビット目を比較し、同じであればヒットとなり該当キャッシュラインの該当オフセット位置からデータを参照する。

書き込みキャッシュに関しては 1 バイトのライトバックで実装をおこなう。

```
  `define        CACHE_ENTRY          64    ;Cache Line Size は 64 Bytes
(512bit)
  assign         CacheI_Tag       = inst_addr[ 31:12 ];
  assign         CacheI_Index     = inst_addr[ 11:6 ];
  assign         CacheI_Offset    = inst_addr[ 5:0 ];
  reg            CACHEI_LRU                        [ 0:`CACHE_ENTRY-1 ];
                 0:Way0 が新しい  1:Way1 が新しい
  reg    [7:0]   CACHEI_Machine_Address            [ 0:`CACHE_ENTRY-1 ];
  reg    [19:0]  CACHEI_Tag?                       [ 0:`CACHE_ENTRY-1 ];
  reg    [511:0] CACHEI_Data?                      [ 0:`CACHE_ENTRY-1 ];
  reg            CACHEI_Invalid?                   [ 0:`CACHE_ENTRY-1 ];
                 0:Valid 1:Invalid
  reg            CACHEI_Modified?                  [ 0:`CACHE_ENTRY-1 ];
                 0:メインメモリと一致  1:Dirty(メインメモリと不一致)
  reg            CACHEI_Owner?                     [ 0:`CACHE_ENTRY-1 ];
                 0:占有  1:Multi のオーナー（書き換え時に他への告知が必要)
```

図 35：CACHE レジスタ

4.8　LAN コントローラ回路

LAN コントローラ回路の概要は図 24 で示したように、2 つの非同期 FIFO と複数の同期 FIFO を組み合わせることにより複数の送受信のチャンネルを同時に受け付けられるようになっている。図 36 に LAN の受信処理の回路を示す。100BASE-T よりの信号を受信すると、RAM などにバッファリングせずに FIFO を経由しパイプライン処理される。

```
if ( E_RX_DV ) begin
        // Receive Data is valid
        if ( rx_status == 0 ) begin
                // プリアンブル部検索中の場合
                rx_status       <= E_RXD[3:0] == 4'b1101 ? 1 : 0;
                fifo_r_wr       <= E_RXD[3:0] == 4'b1101 ? 1 : 0;
                fifo_r_din[8:0] <= 9'b100000001;
                rx_nibble       <= 0;
        end else if ( ~E_RXD[4] ) begin
                // データ部分検索中の場合
                if ( rx_nibble == 0 ) begin
                        fifo_r_wr       <= 0;
                        fifo_r_din[3:0] <= E_RXD[3:0];
                        rx_nibble       <= 1;       // 次は上位 4bit
                end else begin
                        fifo_r_wr       <= 1;       // 次は下位 4bit
                        fifo_r_din[8:4] <= {1'b0,E_RXD[3:0]};
                        rx_nibble       <= 0;
                end
        end else begin
                // RX_ER が発生した場合
                rx_status       <= 0;
```

図 36：LAN 受信処理

4.8.1 ネットワークコマンド

本バスブリッジ間で通信されるネットワークコマンドに関して解説する。

表 10 にコマンドとコードの対応表を示す。メモリの読み込み要求は Read Memory コマンドを用い、メモリの書き込み要求には Write Memory コマンドを送信する。受信したブリッジは Completion コマンドを送信することで応答を行う。

表 10：ネットワークコマンド

番号	コマンド	コード
1	Read Memory Request with cache	10H
2	Read Memory Request without cache	11H
3	Write Memory Request with cache	18H
4	Write Memory Request without cache	19H
5	Cache Line n is Invalid (Reserved)	28H
6	Completion with Data	Req+80H
7	Completion without Data	Req+C0H

4.9 CPU

本節では評価で用いる 32bit RISC CPU[6]の実装について述べる。

4.9.1 CPU 概要

本 CPU は 32bit アーキテクチャであり内部構造は図 37 に示すように、以下の 5 ステージのパイプライン処理によっておこなわれている。

1) 命令の読み込み
2) レジスタからの読み込み
3) ALU を用いたレジスタ間演算
4) メモリへの読み書き
5) レジスタへの書き込み

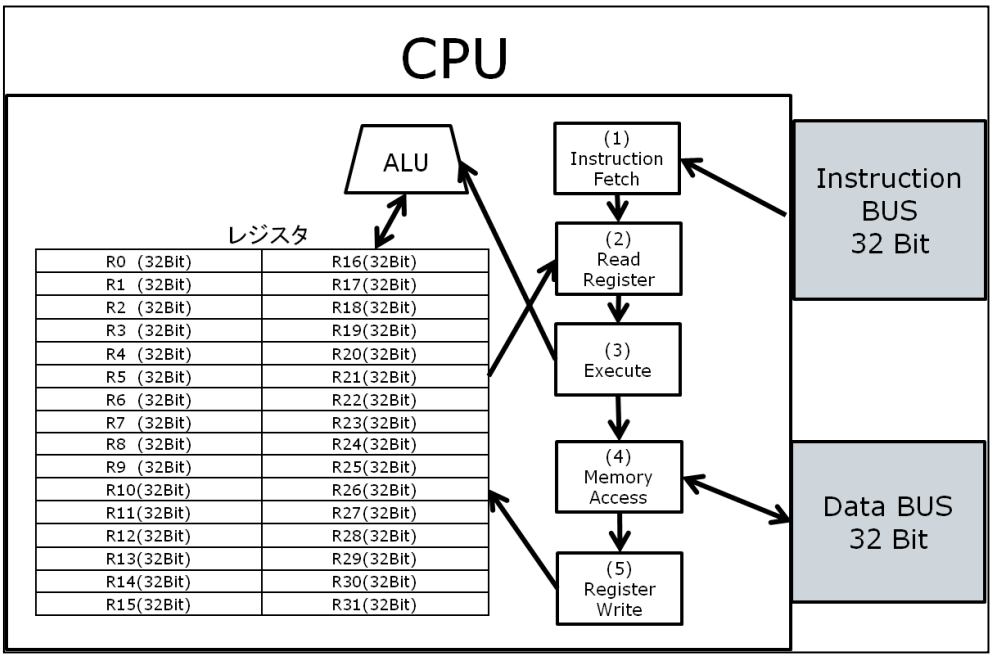


図 37：CPU 内部構成

上記のステージは 1 番目から 5 番目までパイプライン化処理されレジスタによって順次受け渡されていく。常にすべてのステージが稼働しているため、命令用読み込み用のインストラクションバスと、データアクセス用のデータバスをもつハーバードアーキテクチャである。このときの様子を図 38 に示す。

一番最初の状態では IF しか動作していないが、次のサイクルでは RR と IF が同時に動作し、そのつぎのサイクルでは IF と RR と EX が同時に実行され、最終的には IF,RR,EX,MA,RW の 5 つのすべてのステージが常にフル稼働することにより、1 サイクルでひとつの命令を実行することができる。本実装ではパイプライン処理により高速演算がおこなえる CPU で評価を行う。

4.9.2 実行ステージとパイプライン

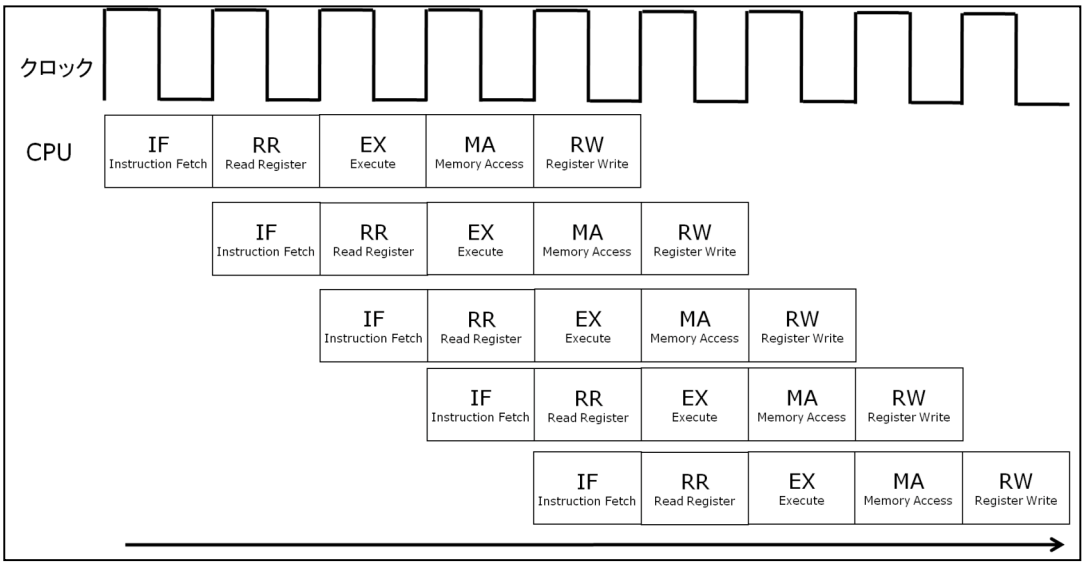


図 38：CPU の 5 ステージとパイプライン

4.10 本章のまとめ

本章では設計に基づき実装を行ったバスブリッジシステムの概要ならびに以下の各部の実装について述べた。

・TLB 回路
・キャッシュレジスタ
・LAN コントローラ
・32 ビット RISC CPU

第5章　評価

本章では本機構が実現した機能、動作に関する評価について述べる。

5.1　評価概要

前章により本研究の目的であるアプリケーションの要求に応じハードウェア割り当てをおこなうことができる機能を実装した。

本章では2.8節で本研究の必要機能要件として示した5つの項目に関して定性評価をおこない関連技術との比較をする。

また、3.8.3 節で述べた高遅延環境におけるキャッシュメカニズムの有効性を評価するための定量評価を行う。

定性評価はCPUとメモリとI/Oデバイスを備える8ビットコンピュータシステムを用い、定量評価では性能の評価を行うため、現状のコンピュータアーキテクチャに近い 32 ビット CPU を用いる。

5.2　関連技術との比較

本節では、設計に基づき実装した機能を評価し、既存関連技術との比較を行う。

実現した機能を下記にあげる

1. コンピュータバスを経由して透過的にリモートにある回路と接続できること。
2. ローカル回路とリモート回路でも透過的に接続できること。
3. ソフトウェアの要求に応じて回路構成を変更できること。
4. 同じメモリ空間を持つ２つ以上のコンピュータバスを接続し、互いの資源を利用することができること。

5.2.1 評価概要

本バスブリッジの有効性を確認するために、最初に単独で動作する 8 ビットコンピュータシステムを用意する。本コンピュータの構成を図 39 に示す。CPU は Z80 でコンピュー

タバス上にデバイス、メモリ、キーボード、ビデオコントローラが接続されている。

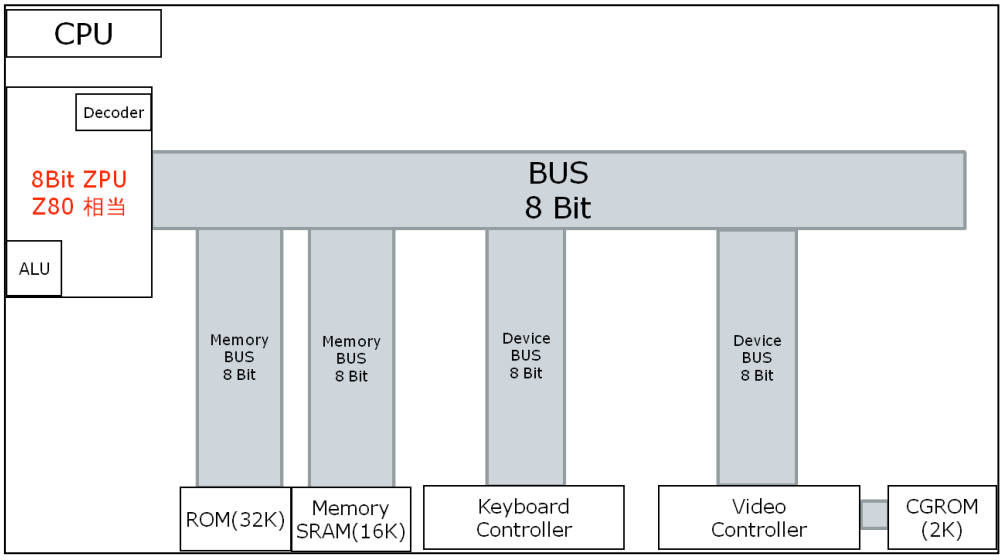


図 39：8 ビットコンピュータシステム概要

次にコンピュータの各デバイス回路は一切変更せずに、バスの部分で回路を分断し、変わりに本機構のバスブリッジを組み込む。このときの内部構成を図 40 に示す。

本バスブリッジ機構はローカル回路もリモート回路も透過的にアクセスできるので、本ブリッジに置き換えた場合でも単独で動作できることを確認する。

次に、同じ構成の 8 ビットコンピュータシステムを用意しネットワークで接続し、ローカルとリモート回路の切り替えが正しくおこなわれることを確認する。

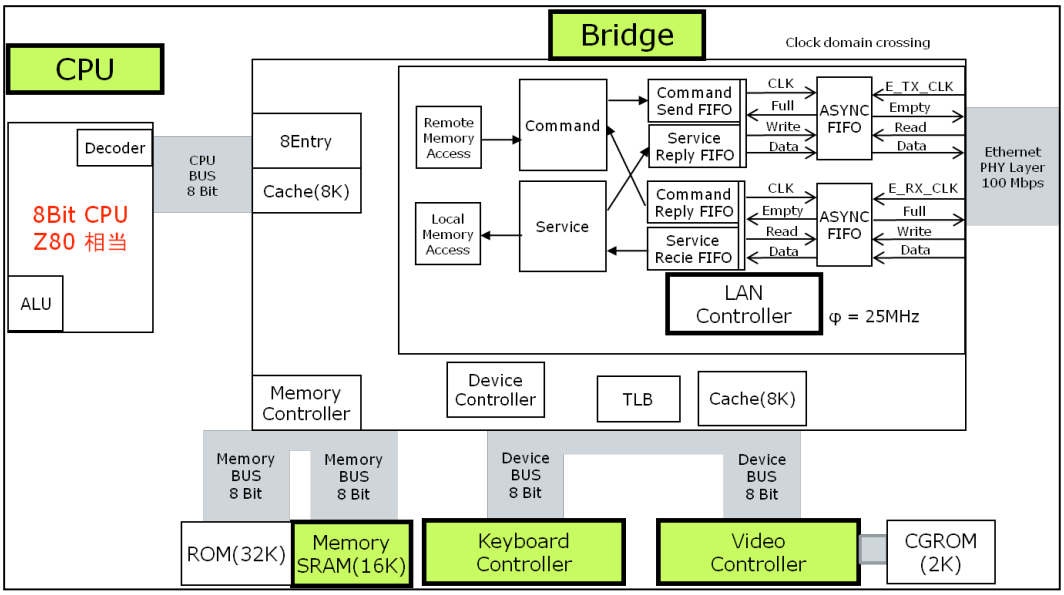


図 40：8 ビットコンピュータシステム バスブリッジ組込版概要

本 8 ビットコンピュータシステムの評価環境を表 11 に示す。

表 11：評価環境

実装基板	Xilinx Spartan 3E Starter Kit
FPGA	Xilinx Spartan 3E (50 万ゲート)
CPU	FPGA 上に実装 Z-80(4MHz)
I/O	下記 I/O 回路を FPGA に実装 Keyboard Video Controller(80×25 テキスト) DMA Controller Beep
ROM	BASIC ROM(32Kbytes)
RAM/VRAM	16Kbytes (ノーウェイト)
ブリッジ	本バスブリッジ FPGA 上に実装
LAN	100Base-T バスブリッジ専用

5.2.2 評価方法

本ブリッジを組み込んだ 8 ビットコンピュータシステムを 2 台用意し、ネットワークで接続する。すべての回路がローカル回路と接続された状態であることを確認したのちに、メモリ上にある TLB 領域にアクセスして、対象とする回路をローカルからリモートへ切り替える。

1. I/O 回路のリモートへの切り替え

コンピュータのキーボード回路が割り当てられている I/O 空間領域$00~$09 をリモート回路上の同じアドレス空間に割り当てる。このときに、キーボードの操作がローカルからリモート側の回路へ切り替わることを確認する。このときにアクティブになっている回路を図 41 に示す。

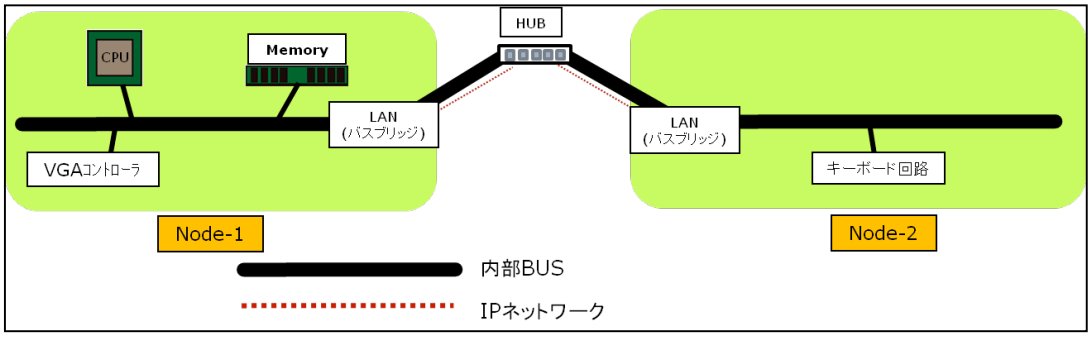


図 41：ネットワーク経由での I/O 回路接続

2. メモリ回路のリモートへの切り替え

コンピュータのビデオメモリが割り当てられているメモリ空間領域$F300~$FEB7 をリモート回路上の同じアドレス空間に割り当てる。このときに、画面表示がリモートメモリの内容へ切り替わることを確認する。このときにアクティブになっている回路を図 42 に示す。

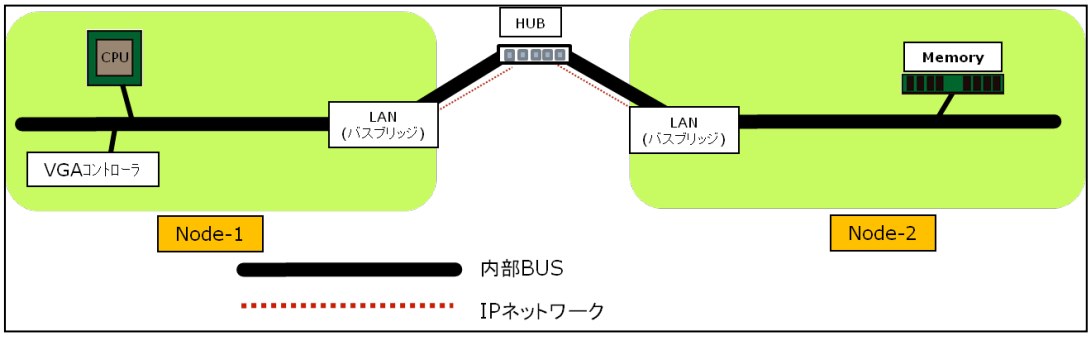


図 42：ネットワーク経由でのメモリ回路接続

3. メモリ回路の共有と CPU の追加実行

コンピュータのビデオメモリの半分に相当する空間領域$F300~$F8DB をリモート回路上の同じアドレス空間に割り当てる。このときに、画面上部半分の表示がリモートメモリの内容へ切り替わり、下半分がローカルメモリの表示であることを確認する。また、画面上半分は 2 台のコンピュータで共有しているため、お互いに画面の上半分が共有できていることを確認する。

また、Node-1 のプログラムメモリを Node-2 の CPU で共有することにより同じプログラムを 2 台の CPU で実行をする。

このときにアクティブになっている回路を図 43 に示す。

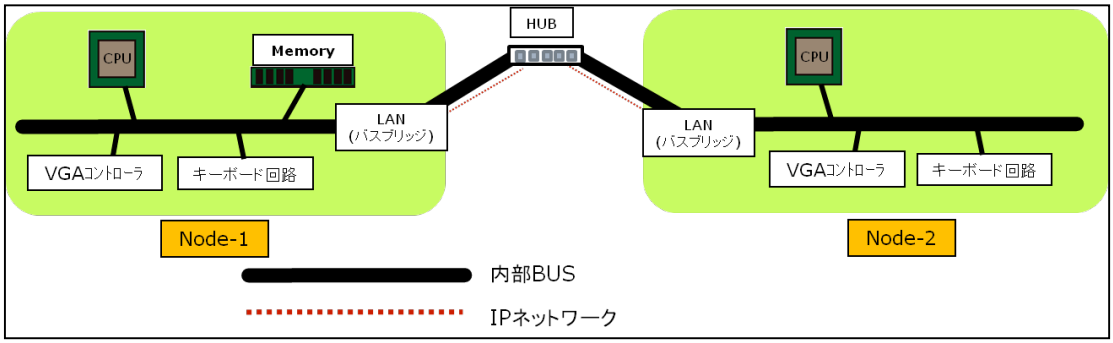


図 43：ネットワーク経由でのメモリ回路共有

5.2.3 評価結果

前節の評価結果を表 12 に示す。この結果をもとに本研究の必要機能要件への対応状況を検討する。

表 12：評価結果

	1.リモートへのI/O切替	2.リモートメモリへの切替	3.メモリ回路の共有とCPUの追加実行
本バスブリッジ	○	○	○

1. リモートへの I/O 切換えをおこなうことによって、I/O 増設ができる。
2. リモートメモリへの切替をおこなうことにより、メモリ増設ができる。
3. メモリ回路の共有と CPU の追加実行ができることにより、CPU の増設ができることと、アドレス空間が同じふたつのコンピュータ上で共有ができることをあらわす。

また IP プロトコルに対応している。

以上の結果を本研究が 2.8 節であげた必要機能要件への対応させたものを表 13 に示す。

表 4 であげた既存技術と必要機能要件の比較表と比べた結果、本手法が最も有効であることが確認できた。

表 13：本研究の必要機能要件への対応状況

接続レベル／手法	CPU増設	I/O増設	メモリ増設	IP対応	共有可否
本バスブリッジ	○	○	○	○	○

5.3　遅延環境におけるキャッシュメカニズムの有効性の評価

本節では、ネットワーク経由で CPU やメモリを接続した場合に、接続回線や利用プロトコルによってどのような特性があらわれるかを調査し、3.8.3 節で述べた高遅延対策がどのように有効に働くかを定量的に評価する。

5.3.1 評価概要

通信で用いられているネットワークは回線性能や品質も様々で、ハードウェアで用いられるコンピュータバスとして扱うには遅延が大きく、性能の低下が懸念される。

回線やその他の条件に応じた本実装の特性を調べるために、表 14 にあげる環境での性能にかかわる定量評価をおこなう。

遠距離接続においての評価をおこなうために、dummynet[33]を用いて遅延環境をシミュレーションする。遅延時間は東京と関西のラウンドトリップタイムにあたる 16ms と日本とヨーロッパのラウンドトリップにあたる 200ms の 2 種類を用いて計測する。

表 14：評価条件

項目	条件
ネットワーク接続方法	クロスケーブル直接
	リピータ HUB
	スイッチ HUB
	dummynet (16ms RTT) (東京と関西との接続を想定)
	dummynet (200ms RTT) (日本とヨーロッパとの接続を想定)
使用プロトコル	IPv4 UDP
	IPv6 UDP
	Ethernet
キャッシュ	キャッシュ機構なし
	キャッシュ機構あり

定量評価は図 44 に示すように様々なネットワークで接続された 2 台の Node を用いて行う。

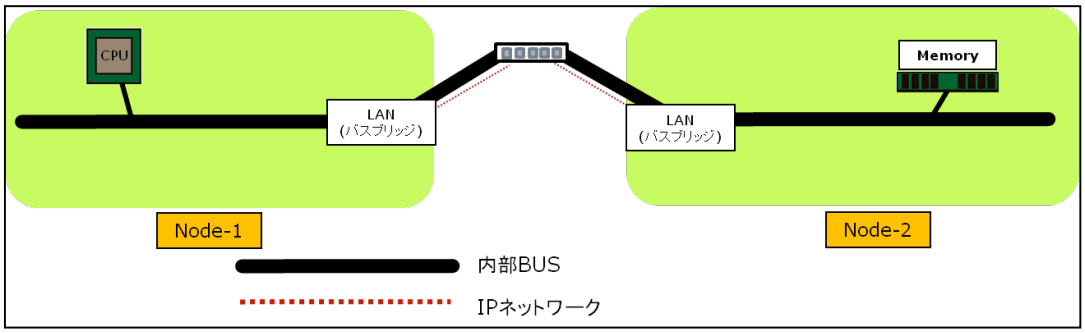


図 44：評価の環境

評価で用いる 32 ビットコンピュータシステムの環境を表 15 に示す。

表 15：評価環境

実装基板	Xilinx Spartan 3E Starter Kit
FPGA	Xilinx Spartan 3E (50 万ゲート)
CPU	FPGA 上に実装 32bit RISC (12.5MHz)
I/O	下記 I/O 回路を FPGA に実装 LED SWITCH
ROM	8Kbytes (ノーウェイト)
RAM	16Kbytes (ノーウェイト)
ブリッジ	本バスブリッジ FPGA 上に実装
LAN	100Base-T バスブリッジ専用

5.3.2 測定方法・環境

測定はオシロスコープを使い、Node-1 の CPU 上で行う。

表 16 に測定環境を示す。

表 16：測定環境

オシロスコープ	Tektronix TDS2014 4 チャンネル 100MHz (1GS/s)

5.3.3 評価方法

評価方法は、表 15 の計測条件を変更しながら下記の 3 項目について計測をおこなう。

計測 1. 1 サイクルあたりの時間

リモートまたはローカル回路間での要求と応答という 1 サイクルを実行するのに所要する時間を計測する。この時間を周波数に直したものが実質的な動作クロックにあたる。

計測 2. 整数演算 1 の実行時間

CPU 上で 1 万回の整数演算にかかる所要時間を計測することで、処理能力を計測する。

キャッシュ機構の効果についても同時に計測する。

計測 3. 整数演算 2 の実行時間

100 万回の整数演算を行う。

CPU に実行時間が長いプログラムを実行させることによって、遠距離接続環境においてのキャッシュメモリの有効性を評価する。

5.3.4 評価計測結果

評価方法にもとづいた計測結果を下記に記す。

計測 1. 1 サイクルあたりの時間

表 17 に、接続条件ごとに測定した 1 サイクルあたりの所要時間を示す。以下、サイクル時間と表記する。

ローカル回路と直接接続した場合のサイクル時間は 0.0805μs であり、周波数に換算すると 12.49MHz となり、CPU の動作クロックと同じである。

クロス TP ケーブルによる接続は HUB を経由せず Node 間をダイレクトに接続するため、遅延が少なくサイクル時間が短い。

スイッチング HUB はパケットを一度バッファ上に記憶してから送信をする。このため遅延が大きくなることが表からもわかる。また、スイッチング HUB は 1 パケット以上の遅延が生じるため、プロトコルヘッダが短い Ethernet においてもそのメリットを生かすことが十分にできない。クロス TP ケーブルでは Ethernet が 7.9μs で IPv6 が 15.6μs と 2 倍近いサイクル時間であったが、スイッチング HUB を利用すると 26.4μs と 34.8μs になり差が少なくなっている。

ローカル回路に直接接続された場合の周波数はおよそ 12490KHz、一番サイクル時間が短いクロス TP ケーブル接続で Ethernet を使用した場合で 126.6KHz であることから、ローカル接続と比べておよそ 100 倍程度速度が低下してしまう。

表 17：プロトコルと回線による 1 サイクルあたりの所要時間

		クロスTPケーブル		リピータ HUB		スイッチングHUB	
	バス種類	時間(μs)／cycle	周波数(Hz)	時間(μs)／cycle	周波数(Hz)	時間(μs)／cycle	周波数(Hz)
1	Ethernet	7.9	126,600	11.0	90,910	26.4	37,880
2	IPv4	12.4	80,650	13.2	75,760	28.8	34,720
3	IPv6	15.6	64,100	16.3	60,970	34.8	28,740
4	ローカル						

		dummynet 遅延16ms		dummynet 遅延200ms		ローカル	
	バス種類	時間(μs)／cycle	周波数(Hz)	時間(μs)／cycle	周波数(Hz)	時間(μs)／cycle	周波数(Hz)
1	Ethernet						
2	IPv4	16,002	62.49	200,013	4.99		
3	IPv6						
4	ローカル					0.0805	12,490,000

計測 2. 整数演算 1 の実行時間

表 18 に 1 万回の整数演算を行うプログラムの実行時間を示す。

メモリをローカルと接続した場合の実行時間は 0.97ms である。

Ethernet でクロス TP ケーブル接続をした場合は 95ms ほどかかり、およそ 100 倍程度の速度低下となる。これは計測 1 で述べたローカル回路に対する速度低下の数値とほぼ同じある。

次にキャッシュを有効にした場合、実行時間はローカルと比べても数パーセントから十数パーセント程度の増にしかならない。キャッシュ無効時と有効時の速度の比較を図 45 に示す。プログラムにおいてはキャッシュ機構が有効になるといえる。

表 18：整数演算 1 の実行時間（1 万ループ）

単位：ms

	バス種類	キャッシュなし		キャッシュあり		ローカル
		クロスTPケーブル	スイッチHUB	クロスTPケーブル	スイッチHUB	
1	Ethernet	95	296	1.01	1.07	
2	IPv4	148	348	1.03	1.09	
3	IPv6	188	424	1.04	1.12	
4	ローカル					0.97

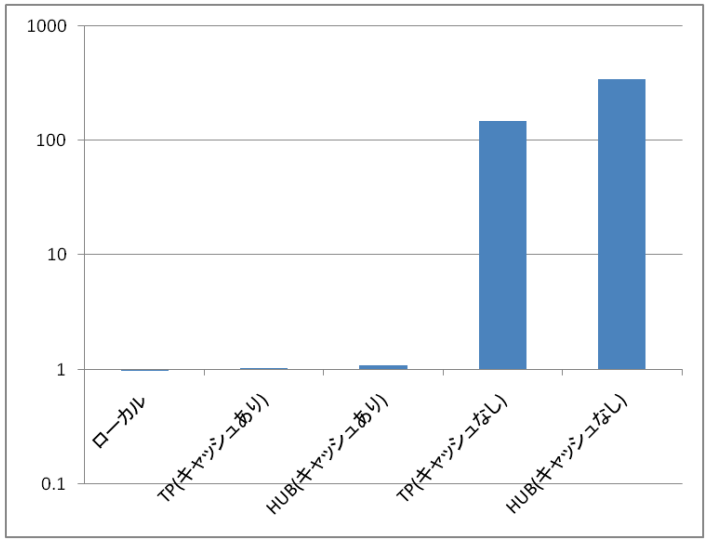


図 45：LAN 環境でのキャッシュ効果(IPv4)

計測 3.　整数演算 2 の実行時間

表 19 に 100 万回ループの整数演算の実行時間について示す。

ローカル回路上で 970ms かかるものを、中程度と高い程度のネットワーク環境上のリモート回路上で実行したものである。

遅延 16ms は東京と関西間を想定し、遅延 200ms は日本とヨーロッパ間の接続を想定している。特に遅延 200ms 環境では、表 17 より周波数が 5Hz となり、CPU の速度の 12.5MHz と比べて相当な速度低下が考えられる。

しかし図 46 で示されるように、本手法で設計、実装したキャッシュを利用することにより、ローカル回路に対して数パーセントから数十パーセント程度の実行時間増ですむことが確認できた。

この時の実行プログラムを付録 A に示す。

表 19：整数演算 2 の実行時間（100 万ループ）

単位:ms

	バス種類	dummynet		ローカル
		遅延16ms	遅延200ms	
1	IPv4(キャッシュあり)	1020	1570	
2	ローカル			970

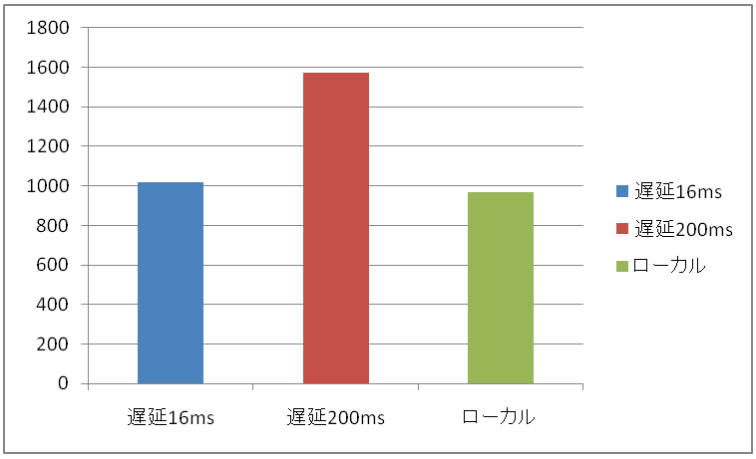


図 46：高遅延環境でのキャッシュの効果(IPv4)

5.3.5 考察

計測 1 と 2 により、キャッシュが無効の場合、回路を接続するネットワークの遅延速度がサイクル時間となり、周波数同様に CPU の性能にリニアに影響してしまうことが確認できた。

またキャッシュを利用することにより、遅延による性能低下を大幅におさえられることがわかった。

特に遅延が 200ms ある環境でも、実行時間が長いプログラムでかつキャッシュが有効に働けば CPU とメモリのように密とおもわれる接続でも利用できると考えられる。

キャッシュの効用はメモリへの読み取りがキャッシュの 1 ライン単位となり、本実装では 64 バイト／ラインでメモリの読み込みが行われ、キャッシュされる。

付録 A に示す今回の整数演算プログラムではサイズが$C0（192 バイト）あるので、本キャッシュ機構の実装では 3 ラインの読み込みを行えば読み取りに関するミスキャッシュをなくすことができると考えられる。

5.4 本章のまとめ

本章では IP 化したバスに対応していない 8 ビットコンピュータシステムに対し本バスブリッジを使うことによりバスを IP 化し、ネットワーク経由でリモートの CPU、I/O、メモリ回路と切り替えた。また、同じアドレス空間を持つ 2 台のコンピュータバスを本 TLB 機構によりアドレスのコンフリクト問題を回避し共有接続した。

以上の結果から本手法は本研究の必要機能要件を満たすことを確認した。

CPU とメモリ間の接続において、キャッシュが有効でキャッシュヒット率が高い場合、

遅延時間が 200ms と大きくても実行時間が長ければ十分実用に耐えうる速度になり、高遅延環境における本キャッシュの有効性が認められた。

第6章　結論

本章では本研究を総括し，得られた成果や未解決の問題点および今後の展望について述べる。

6.1　本研究のまとめ

本研究では、アプリケーションソフトウェアによってハードウェア構成を動的に変更することを提案し、そのために必要な要件として下記の 5 点をあげたが、これらの条件を満たす既存技術はなかった。

1) CPU 増設ができる
2) I/O 増設ができる
3) メモリ増設ができる
4) IP ネットワーク対応
5) 共有／占有の可否

本研究では､これらの要件をすべて満たす方法として IP レベルのネットワークを CPU、メモリ、I/O デバイスのバス間で相互接続する方法を検討し、コンピュータバスと IP 間でプロトコル変換を行えるバスブリッジを設計、実装、評価をした。

評価は 8 ビットパソコンシステムを用い、それぞれ CPU、メモリ、I/O 装置の回路を IP ネットワーク経由で接続することによりすべての要件を満たすことができた。

またネットワーク接続に対応したアドレス変換機構を備える TLB を実装することにより、重なるメモリ空間を持つ 2 つのコンピュータシステム上で回路を共有することができた。

遅延環境における本キャッシュ機構が有効であることを評価するために、32bit の MIPS ベースのコンピュータを使い、200ms 以内の遅延ネットワークで接続されているメモリで負荷の高い計算をおこなった。

その結果、遅延が大きい環境でもキャッシュが働けばローカルと比べてもそれほど劣らない性能で演算ができることが確認できた。

6.2 未解決の問題点

本実装は、UDP プロトコルに対応しているがパケットロスト時の再送処理が行われず、ネットワークの状態によって不安定になる。

6.3 今後の展望

本研究の IP バスブリッジによって、CPU、メモリ、I/O が IP で接続できるようになった。

今後の展望として、本バスブリッジで動作するワークステーションを設計し、汎用 OS である Linux を動作させる。本回路用に OS を適用することにより、アプリケーションの必要に応じて動的に CPU を増やし計算能力をあげられるコンピュータシステムが実現可能となる。

謝辞

本論文の執筆にあたり、多くの方々にお世話になりました。この場をお借りしてお礼を述べさせていただきます。

主査である慶應義塾大学環境情報学部教授 村井純博士に感謝致します。副査である慶應義塾大学環境情報学部教授 中村修博士と慶應義塾大学環境情報学部専任講師 Rodney Van Meter 博士に感謝致します。特に Rodney Van Meter 氏には、夜遅くや早朝にもかかわらず貴重なアドバイスを頂き有難うございました。

ご指導いただきました環境情報学部専任講師 重近範行博士、政策・メディア研究科講師 吉藤英明博士、トヨタ ITC の湧川隆次博士に感謝いたします。

また様々なことについてアドバイスいただきました環境情報学部准教授 植原啓介博士、環境情報学部准教授 楠本博之博士、政策・メディア研究科特別研究准教授 西田佳史博士、政策・メディア研究科特別研究准教授 朝枝仁博士、環境情報学部准教授 三次仁博士、メディアデザイン研究科準教授 杉浦一徳博士、政策・メディア研究科特別研究講師 川喜田佑介博士、政策・メディア研究科特別研究助教 鈴木茂哉、政策・メディア研究科特別研究講師 三川荘子博士に感謝いたします。

修論作成に協力をしていただいた、奥村祐介氏、水谷正慶氏、空閑洋平氏、岡田耕司氏、六田佳祐氏、波多野敏明氏、飯塚裕貴氏ありがとうございました。

また、いつも研究室の素敵な生活に欠かせない堀場勝広氏、久松剛氏、松園和久氏、海崎良氏、石原知洋氏、田崎創氏、三島和宏氏、片岡広太郎氏、工藤紀篤氏、永山翔太氏、本多倫夫氏、佐藤龍氏、江村桂吾氏、金井瑛氏、中里恵氏、黒宮佑介氏、勝利友香氏をはじめみなさま方に感謝いたします。

修論を一緒に執筆した中村友一氏、遠峰隆史氏のおかげで楽しく修論をすすめることができました。

また、研究に関してアドバイスをいただいた、慶應義塾大学理工学部情報工学科教授 天野英晴博士や慶應義塾大学理工学研究科 訪問研究員 松谷宏紀博士に感謝いたします。

３月の WIDE 合宿の屋台村で声をかけていただいたみなさまがたにも本当に感謝です。

毎朝デルタとシャワー室の掃除をしていただいているおばさま方にも感謝しています。

元気に走ると２輪のハングオンを思い出させてくれる Segway にも感謝しています。

そして両親の協力があったからこそできました。長年にわたってありがとうございます。

以上を持って、謝辞といたします。

参考文献

[1] Gordon E. Moore,"Cramming more components onto integrated circuits",Electronics, Vol.38,April 1965.

[2] John von Neumann,"First Draft of Report on the EDVAC", June 1945.

[3] ヘネシー&パターソン,”コンピュータアーキテクチャ 第 4 版”,翔泳社(2008)

[4] 標準バス・インターフェイス研究会,”標準バス・インターフェイス活用入門”, オーム社(1991)

[5] 天野英晴,“並列コンピュータ”, 昭晃堂（1996）

[6] 中森章, “マイクロプロセッサ・アーキテクチャ入門”, CQ 出版社(2006)

[7] インターフェイス編集部, “Ethernet のしくみとハードウェア設計技法”, CQ 出版(2006)

[8] インターフェイス編集部, “改訂新版 PCI デバイス設計入門”, CQ 出版(2005)

[9] Gerry Kane, “mips RISC Architecture R2000/R3000”

[10] 蒲地輝尚,”はじめて読む 486”,アスキー出版局(1994)

[11] 深山正幸,“HDL による VLSI 設計”, 共立出版（1999）

[12] Curt Schimmel, “Symmetric Multiprocessing and Caching for Kernel Programmers”, Addison-Wesley Publishers

[13] Interface 特集,”マルチタスク／マルチコア時代の並列処理技術”, CQ 出版（2007）

[14] 大川善邦,”PCI バスによる I/O 制御”,オーム社(1999)

[15] 荒井信隆,”PCI Express 入門講座”,電波新聞社(2007)

[16] 枝均,”Verilog-HDL による論理合成の基礎”,テクノプレス(2002)

[17] 立川純,”FPGA を実装した PCI カードによる分散共有メモリ型並列計算機の構築”, システム LSI 設計技術 94-21, pp.157-164, 2000.

[18] 日本電気株式会社,”ExpEther”, http://www.nec.co.jp/press/ja/0612/0605.html, Dec. 2006.

[19] J. Satran and K. Meth and C. sapuntzakis and M. Chadalapaka and E. Zeidner,"Internet Small Computer Systems Interface", April 2004, RFC3720.

[20] Fran Berman and Geoffrey C. Fox and Anthony J. G. Hey,”Grid Computing”, John Wiley & Sons Inc, 2003.

[21] 石川裕,”Linux で並列処理をしよう”,共立出版,2007.
[22] Tom Clark,”IP SAN”,ソフトバンクパブリッシング,2003.
[23] 松本尚, 平木敬, “100BASE-TX によるメモリベース通信の性能評価”, コンピュータシステムシンポジウム論文集, 情報処理学会, pp.101--108 (November 1997).
[24] 国沢亮太、松本尚、平木敬,”アドレス変換ハードウェアで支援されたメモリベース通信の性能評価”, 電子情報通信学会技術研究報告, 社団法人電子情報通信学会, Vol.98, No.233(19980804) pp. 61-66.
[25] 国沢亮太、松本尚、平木敬,”アドレス変換機能を持つネットワークインターフェイス”, 全国大会講演論文集, 社団法人情報処理学会, Vol.第 56 回平成 10 年前期, No.1(19980317) pp. 117-118.
[26] 菅原豊、平木敬,”遅延予測に基づいた命令フェッチ機構”, 電子情報通信学会技術研究報告, 社団法人電子情報通信学会,Vol.100, No.20(20000421) pp. 7-14
[27] Martin W. Sachs and Avraham Leff and Denise Sevigny,”LAN and I/O Con-ver-gence:A Survey of the Issues”, IEEE Computer, pp.24-32,Dec. 1994.
[28] Rodney Van Meter, Greg Finn ,Steve Hotz, Bruce Parham,”Netstation Project”, http://www.isi.edu/division7/netstation/,1994.
[29] Garth A. Gibson and Rodney Van Meter, “Network Attached Storage Architecture”, Communications of the ACM Vol.43,pp.37-45,Nov. 2000.
[30] Rodney Van Meter,” A Brief Survey of Current Work on Network Attached Peri-pherals”, ACM Operating Systems Review,pp.63-70,Jam. 1996.
[31] Rodney Van Meter and Greg Finn and Steve Hotz,” VISA: Netstation's Virtual In-ternet SCSI Adapter”, http://www.isi.edu/netstation/
[32] S. Bailey and T. Talpey,” The Architecture of Direct Data Placement (DDP) and Remote Direct Memory Access (RDMA) on Internet Protocols”, December 2005, RFC 4296
[33] Luigi Rizzo, “dummynet”, http://info.iet.unipi.it/~luigi/ip_dummynet/

付録 A

参考資料および評価で用いられたソースリストについて記載する。

定量評価用　ソースリスト

```
 1                  .align 4
 2                  .set noreorder
 3                  .set noat
 4                  .text
 5                  .globl _start
 6                  _start:
 8 0000 240A0000            addiu $10,$0,0
 9 0004 240B0000            addiu $11,$0,0
10 0008 24012710            addiu $1,$0,10000
11
12 000c 24020064            addiu $2,$0,100
13 0010 24030000            addiu $3,$0,0
14                  loop1:
15 0014 24040000            addiu $4,$0,0
16 0018 00000000            nop
17 001c 00000000            nop
18                  loop2:
19 0020 256B0001            addiu $11,$11,1
20 0024 24840001            addiu $4,$4,1
21 0028 00000000            nop
22 002c 00000000            nop
23 0030 00000000            nop
24 0034 014B5020            add $10,$10,$11
25 0038 00000000            nop
26 003c 00000000            nop
27 0040 1482FFF7            bne $4,$2,loop2
28 0044 00000000            nop
29 0048 00000000            nop
30 004c 00000000            nop
31 0050 00000000            nop
32 0054 24630001            addiu $3,$3,1
33 0058 00000000            nop
34 005c 00000000            nop
35 0060 00000000            nop
36 0064 1461FFEB            bne $3,$1,loop1
37 0068 00000000            nop
38 006c 00000000            nop
39 0070 00000000            nop
40 0074 00000000            nop
41 0078 240500BC            addiu $5,$0,lvar
42 007c 00000000            nop
43 0080 00000000            nop
44 0084 00000000            nop
45 0088 00000000            nop
46 008c ACAA0000            sw $10,0($5)
47 0090 00000000            nop
48 0094 00000000            nop
49 0098 00000000            nop
50 009c 00000000            nop
51 00a0 00000000            nop
52 00a4 00000000            nop
53                  loop3:
54 00a8 0800002A            j loop3
55 00ac 00000000            nop
56 00b0 00000000            nop
57 00b4 00000000            nop
58 00b8 00000000            nop
59
60 00bc 00000000            lvar:.long 0x00000000
```