博 士 論 文

メモリスロット装着型ネットワークインタフェースにおける低遅延通信機構に関する研究

2007年度

慶應義塾大学大学院理工学研究科

辻 聡

論文要旨

近年, 汎用のパーソナルコンピュータ (PC) を多数, 相互に接続した PC クラスタが高性能計算機の主流となっている. PC クラスタにおいて, PC 間のインターコネクトには Gigabit Ethernet のような汎用的なネットワークのほか, Myrinet や QsNET, InfiniBand といった専用のネットワークが用いられる.

こういったインターコネクトのネットワークインタフェースは PC の汎用 I/O バスに装着されるが, 長い間 32bit/33MHz の PCI バスが PC における汎用 I/O バスのデファクトスタンダードであった. 32bit/33MHz PCI バスの最大スループットは 132MByte/s と, インターコネクトのスループットが数百 MByte/s に達していたことを考えると非常に低い値であった. サーバやワークステーションには 64bit 幅の PCI バスや PCI バスの上位規格である PCI-X バスが搭載されており, I/O バスの性能は高かった. しかし, PC に比べると高コストであるため, これらを用いて PC クラスタを構築すると PC クラスタの利点の 1 つであるコストパフォーマンスの高さが損なわれてしまう.

一方, PC に搭載されているメモリバスはホストプロセッサの性能向上に追随する必要があることから, 汎用 I/O バスよりもスループット, アクセスレイテンシの面で性能が高く, また, 性能向上率も良いという利点がある. このことから, メモリスロットに装着するネットワークインタフェースである MEMOnet が 1999 年に提案された. 本研究では DIMM スロットに装着する MEMOnet である DIMMnet を用いて, 低レイテンシかつ高スループットな通信を実現することを目的とする.

本研究で用いた DIMMnet は第二世代目の DIMMnet-2 である. DIMMnet-2 は PC の DDR-SDRAM スロットに装着する. 本研究ではコントローラに Xilinx 社の FPGA である Virtex-II Pro を搭載した試作基板を対象に, ネットワークインタフェースコントローラの設計, 及び実装を行い, 基本通信性能の評価を行った. 試作基板には FPGA のほかに 2 枚の DDR SO-DIMM や IEEE 10GBASE-CX4 コネクタが搭載されており, このコネクタと FPGA 内蔵の高速シリアルトランシーバを用いることで, InfiniBand (4X：10Gbps) のネットワークに接続することを可能にしている.

本実装では通信処理やメモリアクセスなどの DIMMnet-2 で実行されるすべての処理をハードワイヤードで実現し, ソフトウェアによる処理を介在させないことで通信性能の向上を達成している. DIMMnet-2 に搭載した低遅延通信機構である BOTF (Block On-The-Fly) の評価の結果, 片方向の最小の通信遅延が 0.632μs であった. また, BOTF は PIO (Programmed I/O) による通信ながら, BOTF を連続して実行することで, 片方向で 631.11MByte/s, 双方向で 1163.70MByte/s という高いスループットを達成した. これらの値は冒頭で述べた PC クラスタ専用のインターコネクトに匹敵する性能である. これらの評価を通し, メモリスロットを利用することで低コストな汎用 PC においても高い通信性能を得られることを示した.

さらに, MPI (Message Passing Interface) に代表されるメッセージ通信を支援するためのメッセージ “受信” 機構である IPUSH (Indirect PUSH) や LHS (Limited-length Head Separation) の提案, 及び実装を行った. これらの受信機構の評価を行い, メッセージ通信時における受信バッファの利用の効率化やメッセージタグの比較のオーバヘッドが削減可能であることを示した. これらは汎用 I/O バスに装着する一般的なインターコネクトに対しても適用可能であり, メッセージ通信を用いるシステムにおいて性能向上が期待できる.

Abstract

Recently, a PC (Personal Computer) cluster, which is consisting of many PCs connected each other with networks, has been a mainstream of high performance computing in most of enterprises and laboratories. For networks in such PC clusters, Myrinet, QsNET and InfiniBand as dedicated networks are used as well as Gigabit Ethernet. The network interfaces for such networks are attached into general I/O buses on PCs. However, the de-fact standard I/O bus; PCI bus running at 33 MHz with 32-bits data width supported only 132 MByte/s throughput, and it was far less than that of the networks. On servers and high-end workstations, high performance I/O buses; PCI bus with 64-bits data width and PCI-X bus were equipped. However, their expensive cost often spoiled the high degree of performance per cost, which is the fundamental benefit of PC clusters.

In contrast, the performance of memory system is higher than that of general I/O buses in order to adapt to the improvement trend of host processors. MEMOnet, a network interface attached into memory slot, was proposed in 1999 to make the best use of this performance. One of the forms of MEMOnet attached into DIMM slot is called DIMMnet.

This research is about designing a network controller logic for DIMMnet-2 which is the second generation of DIMMnet attached into the DDR-SDRAM memory slot, and supports the low latency and high throughput communication. The DIMMnet-2 prototype board used for implementation of the network controller has an FPGA (Virtex-II Pro) with a high speed serial transceiver, two DDR SO-DIMMs and an IEEE 10GBASE-CX4 connector. DIMMnet-2 is connected to InfiniBand network (4X: 10Gbps) using the transceiver and the connector.

All primitive operations on DIMMnet-2 have been implemented with hard-wired logic in the FPGA to improve the communication performance. The performance at micro-benchmark level has been evaluated on the controller. The results indicate that the lowest unidirectional latency of BOTF (Block On-The-Fly) communication mechanism is 0.632 μs. Although the BOTF is for short messages using PIO (Programmed I/O), the throughput is reached at 631.11 MByte/s with unidirectional communication and 1163.70 MByte/s with bidirectional by issuing multiple BOTF requests continuously. They are even equal to those of other recent high performance networks. Thus, it is shown that the general PCs are able to get high communication performance by utilizing the memory slot.

Moreover, the message-receiving mechanisms, IPUSH (Indirect PUSH) and LHS (Limited-length Head Separation), are proposed and have been implemented. These mechanisms support the processing of message passing like MPI (Message Passing Interface). In the result of the evaluation, the efficiency of memory usage and the improvement of the overhead of comparing message tags are showed. These mechanisms are able to be applied to other networks attached into general I/O buses, and the performance improvement is expected on the parallel distributed computing systems using message passing.

目 次

表 目 次

図 目 次

第1章　緒論

近年, ベクトル型スーパーコンピュータに代わり, スカラ型プロセッサを多数用いた高性能なシステムが企業や研究機関で計算資源の主流となっている. 中でも, 汎用の PC (Personal Computer) を多数, 相互に接続した PC クラスタシステムの躍進は目覚しく, このことは世界中のスーパーコンピュータの性能をランキングした Top500[1] にランクインしているシステムの割合の, ここ数年の推移を見ると明らかである.

PC クラスタが多く用いられている背景には, PC 市場の発展による量産効果と搭載される CPU の著しい性能向上から高性能な PC が安価に入手可能となり, 低コストに高性能なシステムを構築可能になったということが挙げられる.

こういった PC クラスタの PC 間の接続に用いられるインターコネクトには Gigabit Ethernet のような汎用的なネットワークのほか, Myrinet[2], QsNET (Quadrics NETwork)[3], InfiniBand[4] といった PC クラスタ専用のインターコネクトが存在する.

PC クラスタ専用のインターコネクトは, SAN (System Area Network) と呼ばれ, RDMA (Remote Direct Memory Access) 転送のサポートや, 低レイテンシでデータの転送が可能なネットワークスイッチ, ハードウェアによる Collective 通信のサポートなどにより, 汎用的なネットワークよりも高い性能を達成している. これらインターコネクトのネットワークインタフェースは, 通常 64bit/133MHz の PCI-X バスや PCI-Express[5] に装着され, 近年では 10Gbps クラスのネットワークを構築することが可能になっている.

ネットワークインタフェースが装着される I/O バスは PCI-Express や HyperTransport[6] の登場により, PCI バスに比べて性能が飛躍的に向上した. 特に, PCI-Express は 2005 年頃には AGP (Accelerated Graphics Port) バスにとって代わり, グラフィックスデバイス向けに汎用 PC に搭載され, 現在では PCI バスに代わる汎用 I/O バスとしての地位を築きつつある. しかし, これらの I/O バスの登場以前は, 汎用 PC においては 32bit/33MHz の PCI バスが主流であり, PCI バスの最初のバージョンである PCI 1.0[7] が策定されてから 10 年以上が経過していた.

図 1.1 に汎用 PC における I/O バスとメモリの進化を時系列で示す. 図 1.1 は Intel 製の PC 向けチップセットでサポートしているメモリや I/O を元にしたものであり, それ以外のチップセットベンダの製品やサーバ向けの製品は対象外としている. メモリの規格もすべてを記載しておらず(注 1), 代表的な値のみをプロットしている. また, PCI-Express は双方向のスループットを示している. この図から, PCI-Express 登場以前は 32bit/33MHz PCI バスとメモリバスとの性能差が拡大し続けていたことが分かる. 特に, チップセットが i875/i865 の世代になると, Dual Channel でのメモリアクセスをサポートするようになり, 性能差は一段と拡大した. 2004 年になると DDR2-SDRAM, 及び PCI-Express をサポートした i925/i915 チップセットが市場に登場したが, 当初 16 レーン (x16)(注 2)の

(注 1)PC-1600 DDR-SDRAM や PC-600/700 RDRAM など

(注 2)PCI-Express は片方向 2 本のシリアル差動信号方式で伝送を行う. 従って, 双方向で 4 本の信号線を用いる. この 4 本の信号線を基本単位 (1 レーン (x1)) とする. 1 レーン当たり, 片方向 2.5Gbps (双方向 5.0Gbps) の伝送速度を持つ (8B10B エンコーディングにより実効速度は片方向 2.0Gbps, 双方向 4.0Gbps). 16 レーンは 1 レーンの信号線を 16 組束ねたものである.

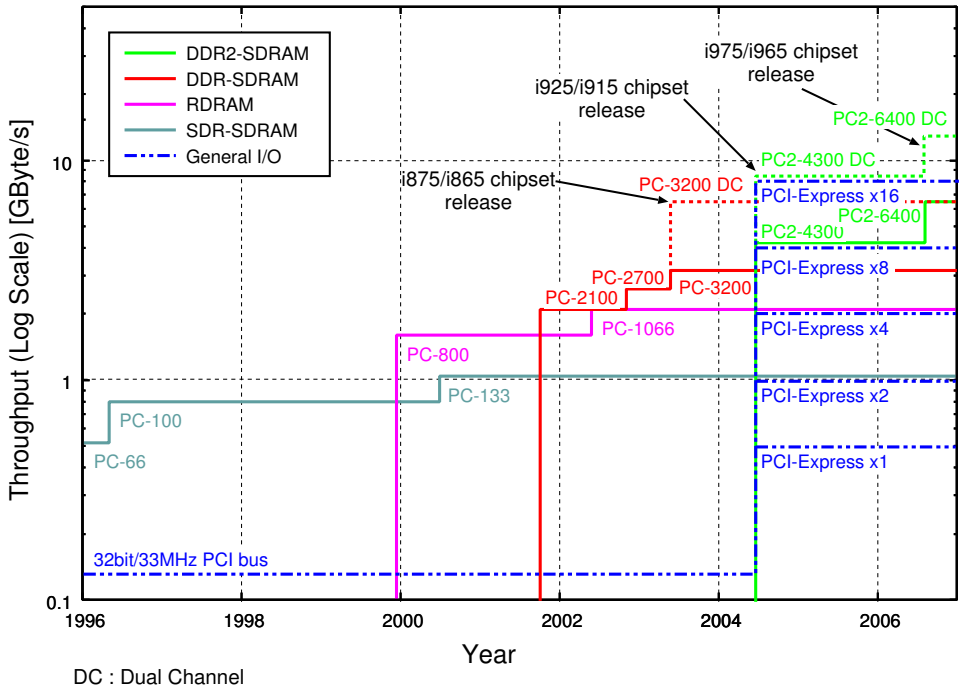


図 1.1 メモリバスと I/O バスの進化

PCI-Express はグラフィックスデバイス専用という位置付けであり, グラフィックスデバイス以外のデバイスを接続すると 1 レーン (x1) のモードで動作するという代物であった [8][9]. i975/i965 からグラフィックデバイス以外のデバイス向けに 4 レーン (x4) や 8 レーン (x8) のポートがマザーボードに搭載されるようになってきた.

このように, PC における汎用 I/O バスの進化の速度は遅く, プロセッサやメモリの性能が向上するにつれて, 汎用 PC を用いて PC クラスタを構築すると PCI バスがボトルネックになるという問題が存在した. 一部では PCI バスを拡張した 64bit/66MHz PCI バスや, 上位規格の PCI-X バスが採用され, 32bit/33MHz PCI バスの 4 倍以上のスループットを示していたが, これらは主にサーバやワークステーションにのみ搭載されていた. そのため, 高性能な PC クラスタの構築には汎用 PC ではなく, これらの高速な I/O バスを持つシステムが用いられたが, これらは一般的に PC より高コストであり, PC クラスタの利点の一つであるコストパフォーマンスの高さを損なう結果となった.

そこで, MEMOnet[10] と呼ばれるネットワークインタフェースのクラスが 1999 年に提案された. MEMOnet は “主記憶を搭載するメモリスロットに接続するネットワークインタフェース” と定義されており, 32bit/33MHz の PCI バスしか搭載されていない低コストな汎用 PC 上でメモリバスのスループットに近い通信性能を実現することを目的としている.

メモリバスは汎用 I/O バス (32bit/33MHz PCI バス) よりもスループットが高く, アクセスレイテンシも低いため, 通信性能が向上すると期待できる. また, 性能向上の速度も汎用 I/O バスより高く, ボトルネックになりにくいという利点がある. さらに, メモリスロットはほぼすべての汎用 PC に搭載されているため, サーバやワークステーションを用いずとも, 上記の利点による恩恵を受けることができ, PC クラスタの構築コストを抑えることが可能となる.

本論文の研究対象である DIMMnet-2 は, DIMM (Dual Inline Memory Module) スロットに装着する MEMOnet として定義される DIMMnet の実装例である.

1.1 DIMMnet-2 プロジェクト

DIMMnet-2 プロジェクトは, 総務省の戦略的情報通信研究開発推進制度 (SCOPE) のプロジェクトの一環として, 東京農工大学中條研究室 (東京農工大), 株式会社 東芝 研究・開発センター, 慶應義塾大学天野研究室 (慶大), 和歌山大学國枝研究室(注 3)によって, 2002 年度に立ち上げられた. 総務省のプロジェクトは 2006 年度で終了したものの, それ以後も上記の研究機関によって研究は続けられている.

DIMMnet-2 は 184pin DDR-SDRAM スロットに装着する. 基板設計は慶大と東京農工大, 株式会社 日立情報通信エンジニアリング (日立 JTE) (注 4) によって行われた [11][12][13]. 2003 年度に設計, 及び製造した基板は, 2004 年度より稼働を開始し, 慶大と東京農工大が中心となり, ネットワークインタフェースコントローラの設計, 実装といったハードウェア部分の開発を行った [14][15]. 一方, 和歌山大学では分散共有メモリシステムの開発が行われた.

2005 年度以降は DIMMnet-2 の基本通信性能の評価やメッセージ通信支援機構の実装 [16][17] などが行われ, これを利用した MPI (Message Passing Interface) が実装された [18]. しかしながら, 通信性能の評価を通じて, ホストから DIMMnet-2 に対してバースト転送でデータを読み書きすると, 意図しないデータが読み書きされるという現象が明らかになり, アプリケーションレベルでの評価を行うのが難しい状況となった. メモリバスへの供給クロックを落とすなどの対策がとられたが, 完全な解決には至っていない.

本プロジェクトの期間に, PC におけるメモリバスが DDR-SDRAM バスから DDR2-SDRAM バスに移行したことを受け, 2006 年度より DDR2-SDRAM スロットに装着する DIMMnet-3 の開発が開始された. 2007 年 7 月の時点で, 基板の設計, 及び製造が完了している.

1.2 DIMMnet-2 プロジェクトにおける筆者の貢献

筆者は 2004 年度より DIMMnet-2 の開発に加わった. 当時, DIMMnet-2 試作基板が完成したばかりの時期であった. 筆者は DIMMnet-2 のプロジェクトにおいて, 慶大側で中心的な立場にあり, ネットワークインタフェースコントローラの開発を主導した.

DIMMnet-2 では通信遅延を削減するために, ネットワークインタフェースコントローラにおける処理をすべてハードワイヤードロジックで実装する方針とした. そのため, プリミティブと呼ばれる, ネットワークインタフェースコントローラで実行される基本命令や, その動作など, ネットワークインタフェースコントローラのアーキテクチャの検討, 及び決定を行った. 2004 年に実装したネットワークインタフェースコントローラには, 後に機能拡張を行ったが, 基本的な構成は変更していない. これは, アーキテクチャ検討の際に, 機能ごとにモジュール化し, 機能追加の際には最小限の変更で済むような構成を採ったことが功を奏したと言える.

2005 年度以降は基本通信性能の評価とネットワークインタフェースコントローラの高機能化を行った. 基本通信性能の評価では, PIO (Programmed Input/Output) 通信機構である BOTF (Block On-The-Fly) やネットワークインタフェース上のメモリ間転送のスループットとレイテンシを測定した. 評価の結果, BOTF の通信性能が Myrinet などの PC クラスタ向けインターコネクトにおける RDMA に匹敵する通信性能を持つことを示した.

ネットワークインタフェースコントローラの高機能化においては, メッセージ通信を支援するための通信機構である IPUSH と LHS の設計, 及び実装を行った. MPI などのメッセージ通信をこれ

(注 3)後, 立命館大学國枝研究室
(注 4)当時, 日立インフォメーションテクノロジー (日立 IT)

らの通信機構を用いて実装することで低レイテンシな通信を実現可能であることを示した.

また, DIMMnet-2 のデバッグツールとして, DIMMnet-2 を対話的に操作できる dsh (DIMMnet Shell) を開発し, 実機を用いた動作確認のための環境を整備した.

1.3 本論文の構成

図 1.2 に本論文の各章の関係を示す. 2 章で DIMMnet-2 に関連のある研究として, メモリスロットを機能拡張に用いる種々のシステム, PC クラスタ向けインターコネクト, 及びメッセージ通信を支援する通信機構について述べる. 3 章では第一世代の DIMMnet である DIMMnet-1 と, 研究対象である DIMMnet-2 の概要を述べる. 4 章と 5 章では DIMMnet-2 ネットワークインタフェースコントローラの設計と実装についてそれぞれ述べる. 6 章では 5 章で実装したネットワークインタフェースコントローラの基本通信性能を示し, 7 章ではメッセージ通信支援機構の評価を示す. そして, 8 章で本研究をまとめる.

付録として, 付録 A と付録 B ではネットワークインタフェースコントローラ内部のレジスタのビットフィールドを, 付録 C では dsh のマニュアルを掲載する.

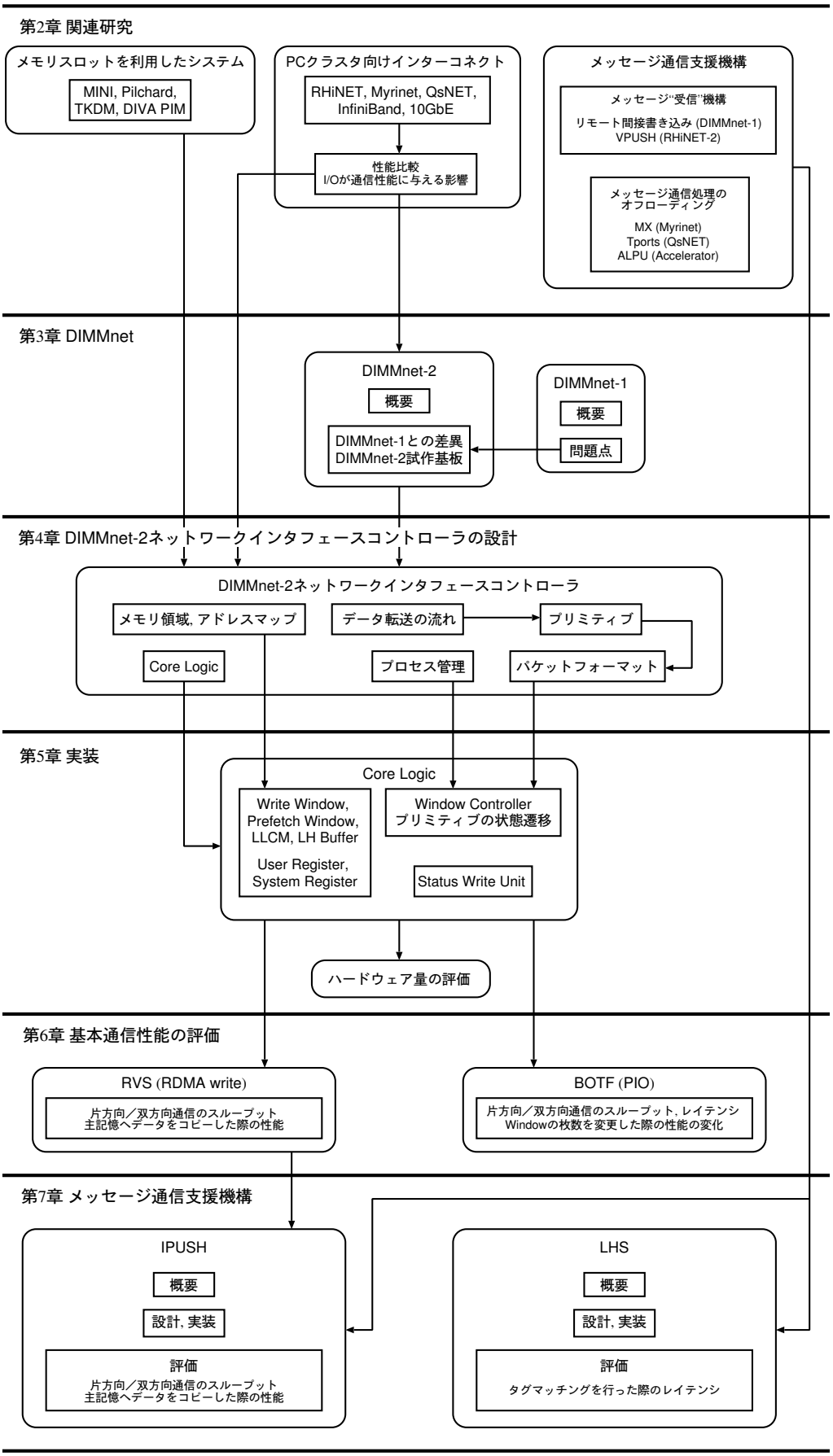

第2章 関連研究
メモリスロットを利用したシステム
MINI, Pilchard, TKDM, DIVA PIM
PCクラスタ向けインターコネクト
RHiNET, Myrinet, QsNET, InfiniBand, 10GbE
性能比較
I/Oが通信性能に与える影響
メッセージ通信支援機構
メッセージ"受信"機構
リモート間接書き込み (DIMMnet-1)
VPUSH (RHiNET-2)
メッセージ通信処理のオフローディング
MX (Myrinet)
Tports (QsNET)
ALPU (Accelerator)
第3章 DIMMnet
DIMMnet-2
概要
DIMMnet-1との差異
DIMMnet-2試作基板
DIMMnet-1
概要
問題点
第4章 DIMMnet-2ネットワークインタフェースコントローラの設計
DIMMnet-2ネットワークインタフェースコントローラ
メモリ領域, アドレスマップ
データ転送の流れ
プリミティブ
Core Logic
プロセス管理
パケットフォーマット
第5章 実装
Core Logic
Write Window, Prefetch Window, LLCM, LH Buffer
User Register, System Register
Window Controller
プリミティブの状態遷移
Status Write Unit
ハードウェア量の評価
第6章 基本通信性能の評価
RVS (RDMA write)
片方向／双方向通信のスループット
主記憶へデータをコピーした際の性能
BOTF (PIO)
片方向／双方向通信のスループット, レイテンシ
Windowの枚数を変更した際の性能の変化
第7章 メッセージ通信支援機構
IPUSH
概要
設計, 実装
評価
片方向／双方向通信のスループット
主記憶へデータをコピーした際の性能
LHS
概要
設計, 実装
評価
タグマッチングを行った際のレイテンシ

図 1.2　各章の関係

第2章　関連研究

本章では DIMMnet に関連する事例についてまとめる. まず, DIMMnet と同様に, メモリスロットを機能拡張に用いる様々なシステムについて述べる. 続いて, PC クラスタ向けインターコネクトについて触れる. ここでは DIMMnet に関係が深いインターコネクトである RHiNET, 及び商用のインターコネクトとして代表的な Myrinet, QsNET, InfiniBand についてまとめる. 最後に並列処理に用いられているメッセージ通信を支援する通信機構について述べる.

本研究ではこれらの事例の問題点や通信性能に影響を与える要件をもとにして, DIMMnet-2 ネットワークインタフェースコントローラの設計, 及び実装を行っていく.

2.1　メモリスロットを機能拡張に用いるシステム

本節ではメモリスロットを汎用 I/O バスのように機能拡張のために利用するシステムについて述べる. これらのシステムがホストプロセッサから, どのように利用可能であるのかを示し, その問題点をまとめる.

2.1.1　MINI

MINI (Memory-Integrated Network Interface) は Minnich らによって提案された, SIMM extender を介して 72pin SIMM バスに接続するネットワークインタフェースである [19].

MINI は並列分散処理向けに,

- 単一の ATM (Asyncronous Transfer Mode) のセルを 1μs のレイテンシで転送
- 1Gbps のスループット
- ゼロコピー通信 (2.2 節) を利用した TCP/IP, 及び NFS (Network File System) の実現

といったことを目標に開発された. MINI は単一の ATM のセルを 400ns でネットワークに送出することが可能であり, ホストの処理を含めた RTT (Round Trip Time) は 3.9μs であった.

図 2.1 に MINI のブロック図を示す. Host Interface Logic 上にメモリや制御ロジックなどが搭載されており, これらはホストプロセッサから直接アクセス可能である.

2.1.2　Pilchard

Pilchard は Leong らによって開発された, FPGA (Field Programmable Gate Array) を搭載した Reconfigurable Computing 用のシステムである [20][21]. Pilchard は PC の 168pin SDR-SDRAM バスに接続する. 対応するバスクロックは PC100, 及び PC133 である.

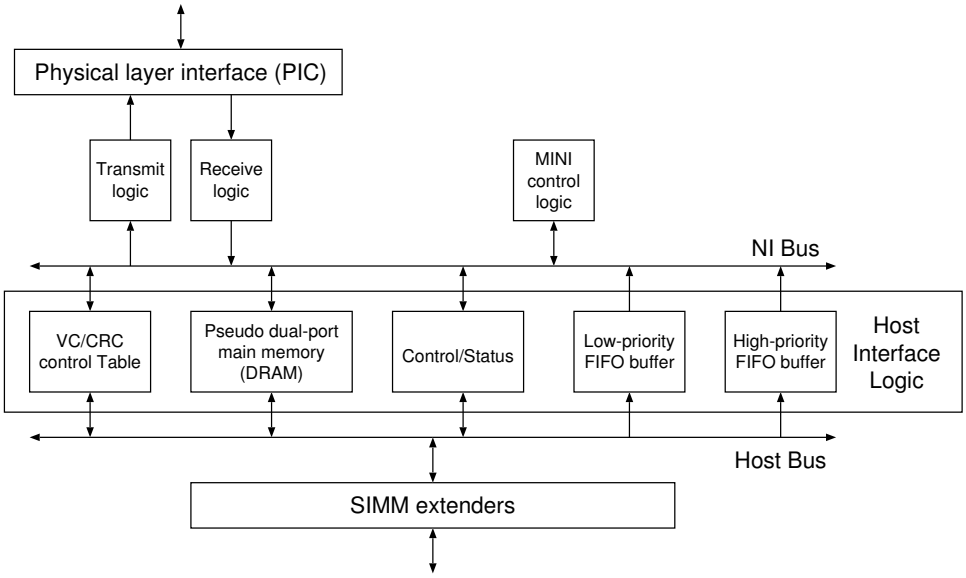


図 2.1 MINI Architecture

CPU や FPGA の性能が向上するにつれて, 汎用 I/O バスである PCI バスとの性能差が拡大する. そのため, PCI バス接続型の Reconfigurable Computing システムでは PCI バスがボトルネックとなり, 高い性能が得られないことを背景として Pilchard は開発された.

PC においては, チップセットが SPD (Serial Presence Detect) インタフェースを持つようになり, BIOS (Basic Input/Output System) がメモリの情報をメモリモジュール上の SPD ROM から読み出すことでメモリの検出や設定を行うようになった. しかし, Pilchard は SPD ROM を持っていないため, PC の起動時には Pilchard は検出されない. そこで, PC 起動後にチップセットのレジスタを書き換えることで Pilchard を利用可能にしている.

図 2.2 に Pilchard のブロック図を示す.

ユーザプログラムは UNIX の mmap システムコールを用いて, Pilchard のインタフェースを自プロセスのアドレス空間にマップする. このようにすることで, 一度マッピングすれば, システムコールを介することなく, Pilchard を扱うことが可能になる.

Pilchard では DIMM インタフェースの領域を read 用と write 用に分けている. これは, ホストから Pilchard へのアクセスを最適化するためである. Pentium Pro 以降の Intel 製プロセッサでは MTRR (Memory Type Range Register) [注 1)]を利用して, ページ単位でメモリアクセスの挙動を制御可能である. そのため, write 用の領域を Write Combining 属性, read 用の領域を Uncachable 属性に設定することで, 書き込み時に Pilchard に対して高いスループットを得ることができる.

Write Combining 属性に設定した場合の Pilchard への書き込み時のスループットは PC100 のメモリバスを使用した際に 409.64MByte/s まで達しており, 32bit/33MHz の PCI バスの理論最大スループットの 4 倍近い性能を示している. 一方, read 領域が Uncachable 属性に設定されていることによって, 読み出し時のスループットは 52.8MByte/s 程度にとどまっている.

[注 1)]Pentium Pro 以降の IA32 アーキテクチャのプロセッサで利用可能な, プロセッサからメモリへのアクセス方式を制御するためのレジスタ

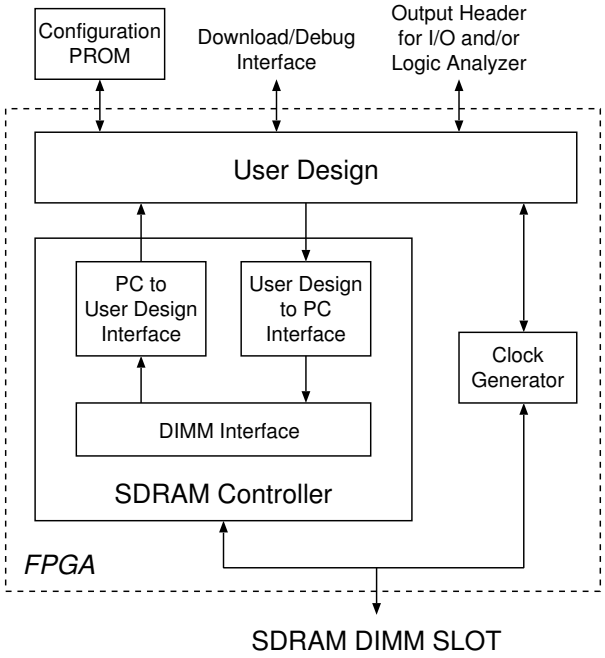


図 2.2 Pilchard Architecture

2.1.3 TKDM

TKDM はスイス連邦工科大学で開発された, PC の SDR-SDRAM スロットに装着するストリーミング処理向けのアクセラレータである [22]. 基板上に FPGA を搭載しており, これを用いてアプリケーションのアクセラレーションを行う. TKDM は Pilchard と同様に, 汎用 I/O バスがボトルネックとなっていることを背景として開発された. 図 2.3 に TKDM のブロック図を示す.

TKDM にはホストのメモリバスと接続する AL-FPGA (Abstraction Layer FPGA) と, ユーザの設計した論理を搭載する T-FPGA (Target FPGA) という 2 個の FPGA が搭載される. これにより, T-FPGA にホストのメモリバスのインタフェースを搭載する必要がなくなるため, ユーザが利用可能なリソースを多く確保することができるという利点がある. さらに, ボード上に 4 枚の SDRAM が搭載されており, Bus Switch を切り替えることで AL-FPGA と T-FPGA が同時に SDRAM を利用することが可能になっている. Bus Switch の切り替えは AL-FPGA から行い, ホストからは AL-FPGA を介して, SDRAM にアクセス可能である.

TKDM は Pilchard 同様, PC の起動時には BIOS から検出されないため, PC の起動後にチップセットの設定を変更することで, TKDM を利用可能にしている. また, この変更と同時に TKDM のホストからアクセス可能な領域を Uncachable 属性に設定している. このことにより, TKDM に対するアクセスのスループットは書き込み時に 128MByte/s, 読み出し時に 53MByte/s 程度にとどまっている.

2.1.4 DIVA PIM

DIVA PIM (Data IntensiVe Architecture Processing-In-Memory) はプロセッサとメモリの速度のギャップを埋めるための手法である PIM ベースのシステムであり, USC/ISI で研究が行われてい

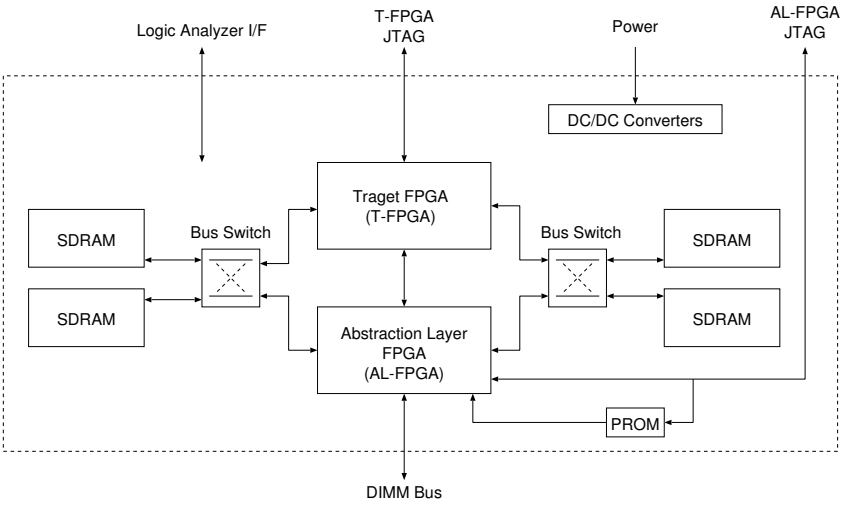


図 2.3 TKDM Architecture

る. DIVA PIM はストリーミングデータを扱うマルチメディア系のアプリケーションなど, スループットが求められるアプリケーションを対象としたシステムである [23][24]. 図 2.4 に DIVA システムのアーキテクチャを示す.

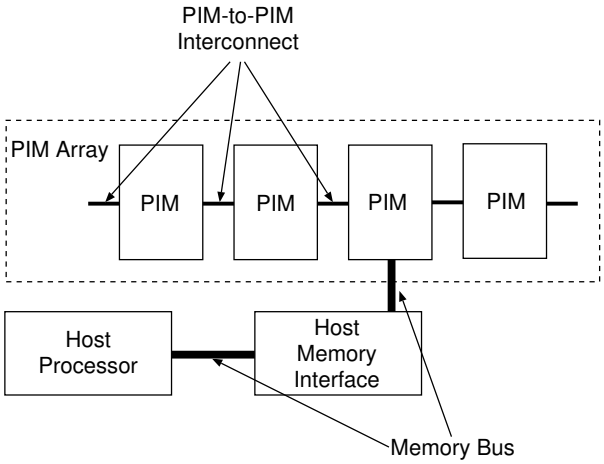


図 2.4 DIVA PIM Architecture

ホストはメモリバスに接続された PIM によって相互に接続される. PIM を co-processor として用いるために, DDR-SDRAM スロット装着型の DIVA PIM チップを搭載した基板が開発されている.

2.1.5 まとめ

本節で紹介したメモリスロット装着型のシステムは, いずれも汎用 I/O バスではスループットが不足するという問題を解決するために開発された.

しかし, Pilchard と TKDM はホストからアクセス可能な領域のすべて, または一部を Uncachable

属性にする必要があるため, 汎用 I/O バスに対して大幅な性能向上は見られなかった. 仮に, Pilchard の read 領域をホストの CPU にキャッシュされる Write Back 属性に設定した場合, 読み出し時のスループットは向上する. しかし, 一度 Pilchard の read 領域をキャッシュに格納すると, Pilchard 側から当該の read 領域を書き換えても CPU はキャッシュからデータを読み出すため, 新しい値が反映されないという問題がある. そのため, Pilchard からデータを読み出すたびに CPU のキャッシュをフラッシュすることが必要となる. しかし, Pilchard は PentiumIII のシステムに搭載されており, PentiumIII のキャッシュフラッシュ命令を実行するとキャッシュの全領域がフラッシュされる. そのため, 結果としてシステムの性能低下を招くことになる.

Intel 製の x86 命令セットのプロセッサでは, Pentium4 からキャッシュライン単位でキャッシュをフラッシュする CLFLUSH 命令が追加された. DIMMnet-2 では, この CLFLUSH 命令とメモリへのプリフェッチ命令である PREFETCHNTA 命令を利用することで, ホストから DIMMnet-2 に対する読み出し時に高いスループットを得ることができるようにしている.

また, MINI のように, 基板上のメモリに対してホストから直接アクセス可能な形態を採用すると, メモリバスが高速になるに従って, システムを構築することが難しくなる. MINI の場合は SIMM extender にネットワークインタフェースを装着するが, この場合, メモリバスからの物理的な距離が大きくなる. 近年の SDRAM の場合, RAS (Row Address Strobe) 信号を出してから一定クロック後に CAS (Column Address Strobe) 信号を出し, それから一定クロック後にデータの読み書きを行うということが仕様で定められているため, この制約を満たせなくなることが予想される. このことから, DIMMnet-2 では, ホストプロセッサから基板上のメモリに対して, コントローラ内部のバッファやレジスタを介して, 間接的にアクセスする構造を採用した. これにより, 基板に搭載するメモリの物理的な配置や容量 (枚数) の柔軟性が高まるという利点がある.

2.2 並列分散処理環境用インターコネクト

本節では PC クラスタなどの並列分散処理環境向けに開発されたインターコネクトについて述べる. 各インターコネクトの特徴をまとめ, その性能比較を行うことで, システムを構成する要素のうち, 特に通信レイテンシに影響が大きい事柄を明らかにする.

一般に, 高性能な並列分散処理環境用インターコネクトではユーザレベル通信とゼロコピー通信を実現することにより, 汎用のネットワークである Fast Ethernet や Gigabit Ethernet より高スループット, 低レイテンシな通信を行っている.

ユーザレベル通信 ユーザレベル通信とは, ユーザプロセスが通信を起動する際に OS のシステムコールを用いずに, ユーザ権限でデバイスに直接要求を出して通信を起動する方式である. ユーザレベル通信を用いることで, システムコールの発行に伴う OS のオーバヘッドを除外し, 通信が開始されるまでのレイテンシを大幅に低減することが可能となる.

ゼロコピー通信 一般に, Ethernet などのネットワークを用いた通信では, 送信データは一度, OS 上の送信バッファにコピーされ, ネットワークインタフェースは送信バッファからデータを送信する. 受信側においても, ネットワークから到着したデータは一度, OS 上の受信バッファに蓄えられ, その後, 受信バッファからユーザプロセスの領域にコピーされる. このメモリ間のデータコピーは PIO で行われるため, 転送速度は極めて低く, 通信オーバヘッドを大きくする要因の一つとなる.

ゼロコピー通信は, 通信からメモリ間のデータコピーを排除した通信方式である. 予め転送デー

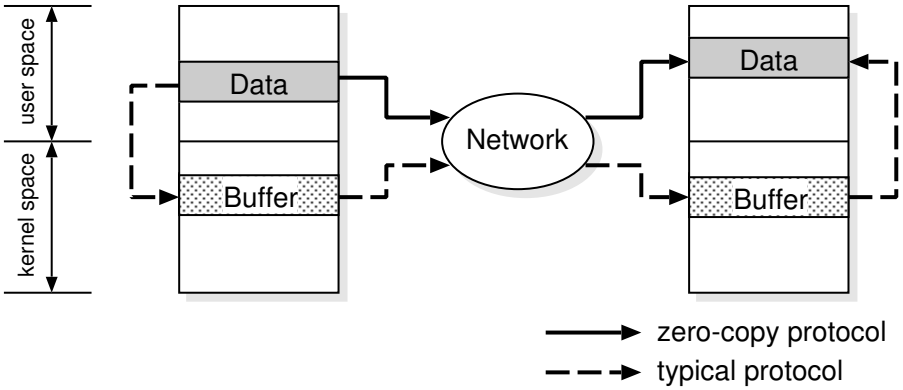


図 2.5　ゼロコピー通信

タが置かれた領域とデータを受信する領域を, 各々ネットワークインタフェースに登録しておき, 通信が発行されるとネットワークインタフェースがメモリとネットワークの間で DMA 転送を行うことでメモリ間のデータコピーを排除する.

一般的な通信とゼロコピー通信におけるデータの流れを図 2.5 に示す. 図 2.5 中の実線はゼロコピー通信のデータの流れを, 点線は一般的な送受信バッファを用いた通信のデータの流れを示している.

2.2.1 RHiNET

RHiNET[25][26] は本研究室と新情報処理開発機構が共同で開発した, LASN (Local Area System Netowrk)[27] というネットワーククラスのためのインターコネクトである.

一般の PC クラスタに用いられる SAN の場合, インターコネクトのトポロジやリンク長に制限があるため, システムを構成するノード (PC) は 1 箇所に集中して設置される. これに対し, LASN ではオフィスの 1 フロアなど, ある程度の広さを持った空間に分散配置された PC を相互に結合し, 並列分散処理環境を構築する.

このようなシステムを提案した背景として, 近年の PC の価格対性能比の著しい向上から, 高速な PC が数十〜数百台規模でオフィスなどに導入されるようになったことが挙げられる. このような PC は 1 台 1 台がスーパーコンピュータの 1PE (Processor Element) 相当の性能を持つに至っており, また, オフィスのような環境では事務処理などの比較的負荷の軽い処理が主な用途であるため, 各 PC は計算資源に余裕があるものと考えられる. そこで, これらの PC を相互結合し, 余剰計算資源を利用して分散並列処理を行うことによって, 計算資源の有効利用を図るのが LASN の目的である.

LASN には LAN (Local Area Network) のような, トポロジやリンク長に対する柔軟性と SAN のような高い信頼性や通信性能が要求される.

図 2.6 に LASN を用いてフロア内の PC を接続した際の概観を示す.

LASN に要求される高スループットで低レイテンシな通信を実現するには, ハードウェアそのもののデータ転送能力を向上させることもさることながら, 通信へのソフトウェアの介在を極力排除することが効果的である. RHiNET では, 次の機能をハードウェアで提供することにより, ソフトウェアによるオーバヘッドを排除している.

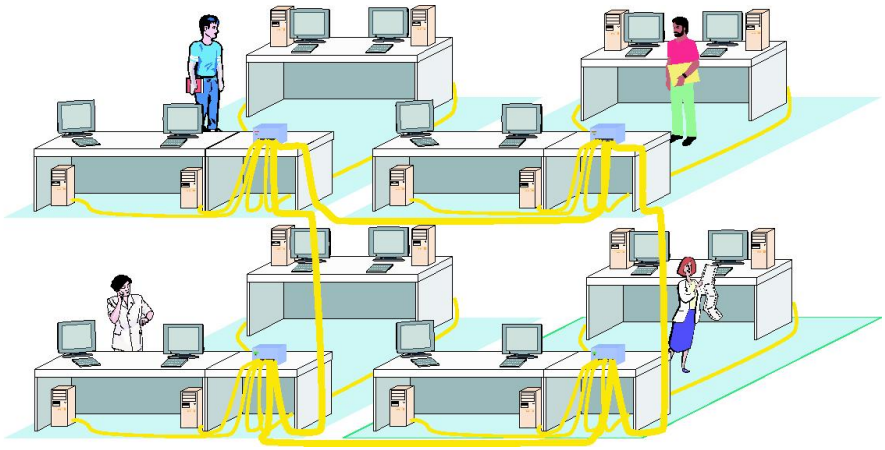


図 2.6　LASN によるフロア内 PC 接続時の概観

- パケットの順序保証と, 通信エラーの回避を行うことで, 上位レイヤによる通信保証処理のオーバヘッドを排除
- 通信におけるプロテクション機構を設けた上でユーザレベル通信を用いることで, 通信起動時の OS の介在によるオーバヘッドを排除
- RDMA を用いたゼロコピー通信を行うことで, ユーザメモリと送信バッファとの間のメモリ間コピーによるオーバヘッドを排除

LASN では, これらを任意のトポロジで実現する. さらに, フロアレベルで分散配置された PC を相互接続すべく 100m〜1km 程度のリンク長をサポートする必要がある. RHiNET では, 専用のネットワークスイッチと専用のネットワークインタフェースを開発し, 伝送媒体に光ファイバを用いることで, これらを実現している.

RHiNET のネットワークインタフェースは RHiNET/NI と呼ばれる. RHiNET/NI はホスト上のソフトウェアオーバヘッドを除外するために, ユーザレベル通信とゼロコピー通信をハードウェアで提供する. また, 高い通信性能を実現するために, 基本的な通信機能をハードウェアで提供している. ユーザに対してハードウェアで提供する通信処理をプリミティブと呼び, ネットワークインタフェースに対してプリミティブの要求を発行することで通信処理を起動する.

RHiNET のネットワークスイッチは RHiNET/SW と呼ばれる. RHiNET/SW では, ネットワーク上でのパケットの順序性を保証し, かつデッドロックによるパケットの破棄を行わないよう, 縮約構造化チャネル法 [28] を採用し, ソフトウェアによる通信保証処理を不要としている. また, スイッチを経由することによる遅延の加算を低く抑えるために, Asynchronous Wormhole Routing[29] を採用している.

2.2.1.1　RHiNET-1

RHiNET の最初の実装である RHiNET-1 は, ネットワークインタフェース RHiNET-1/NI, ネットワークスイッチ RHiNET-1/SW, 及び 1.33Gbps の転送容量を持つ光リンクで構成される.

RHiNET-1/NI は, PC で標準的に利用されている 32bit/33MHz の PCI バスに接続するネットワー

クインタフェースであり, 光インタコネクションモジュール, CPLD (Complex Programmable Logic Device) を用いたコントローラ部, アドレス変換テーブル保存用のメモリなどで構成される.

RHiNET-1/SW は 0.35μm プロセスの CMOS エンベデッドアレイによる 1 チップの ASIC スイッチ LSI と大容量の外部 SRAM で構成される. 外部 SRAM はパケットバッファとして用い, チップ内部のメモリをキャッシュとして利用する仮想チャネルキャッシュ方式 [28] を採用している. また, チップ内に 8×8 のクロスバを内蔵し, 光インタコネクションモジュールを 8 組接続可能な構造となっている. フロー制御は Stop-and-Go 方式で行っている.

2.2.1.2 RHiNET-2

RHiNET-2 は, 多様な形態のネットワークスイッチ, ネットワークインタフェース, 及び伝送媒体をサポートしたインターコネクトである. RHiNET-2 は RHiNET-1 と同様にネットワークスイッチ, ネットワークインタフェース, 光リンクで構成される.

RHiNET-2/NI は, 64bit/66MHz の PCI バスに装着するネットワークインタフェースであり, コントローラ部に Martini[30][31][32][33][34] と呼ばれる専用 ASIC を搭載している. さらに, RHiNET-2/SW に接続可能な光インタコネクションモジュールやノート PC 用の SDR SO-DIMM を備える. Martini には RDMA read/write(注 2)のプリミティブしか実装されておらず, PUSH/PULL 以外の通信処理や例外処理はオンチッププロセッサやホストでソフトウェア処理するという方針で設計されている. この Martini は DIMMnet-1 のネットワークインタフェースコントローラとしても使用された. Martini のブロック図を図 2.7 に示す. Martini は RHiNET-2 と DIMMnet-1 という, ホストとのインタフェースが異なるインターコネクトをサポートするために PCI インタフェース部と DIMM インタフェース部を持つ. ハードワイヤードロジック部には RDMA 転送を実現するための DMA 制御部や, ハードウェアサポートされていない通信の処理や例外処理用のオンチッププロセッサが搭載されている. Switch Interface は RHiNET-2 用のスイッチのほか, RHiNET-3 用のスイッチなど, 様々なスイッチとの接続をサポートする.

Martini には, RDMA による転送のほか, PIO による通信機構をサポートし, BOTF (Block On-The-Fly)[35] と AOTF (Atomic On-The-Fly)[36] という少量のデータ転送用の低遅延な通信機構を提供している.

RHiNET-2/SW は 0.18μm プロセスの CMOS エンベッデッドアレイで構成される 1 チップスイッチである. チップ内部に大容量の SRAM を持ち, 外部メモリを必要とせず, さらに, 1Gbps に加え 8Gbps の光リンクに対応している. RHiNET-2 には光インタコネクションモジュールを搭載せずに, 電気信号 (LVDS) を用いる実装も存在する.

2.2.1.3 RHiNET のソフトウェア

RHiNET は, 基本通信処理へのソフトウェアの介在を極力減らすことで, 高スループットで低レイテンシな通信を実現するインターコネクトである. しかし, ハードウェアで提供されていない通信機能や例外処理, ホスト上でのデバイス管理などにおいてはソフトウェアが必要となる. RHiNET-2 システムのソフトウェアレイヤは, 個々のノード上に構築される. RHiNET-2 システムのソフトウェアレイヤを図 2.8 に示す.

階層の底辺にはハードウェアである RHiNET/NI が位置し, その上にオンチッププロセッサで実

(注 2)RDMA write を PUSH, RDMA read を PULL と呼ぶ

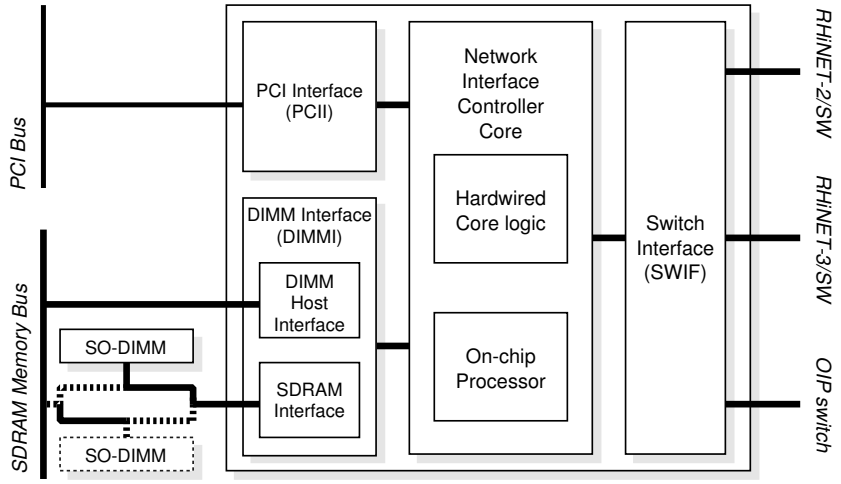


図 2.7 Martini のブロック図

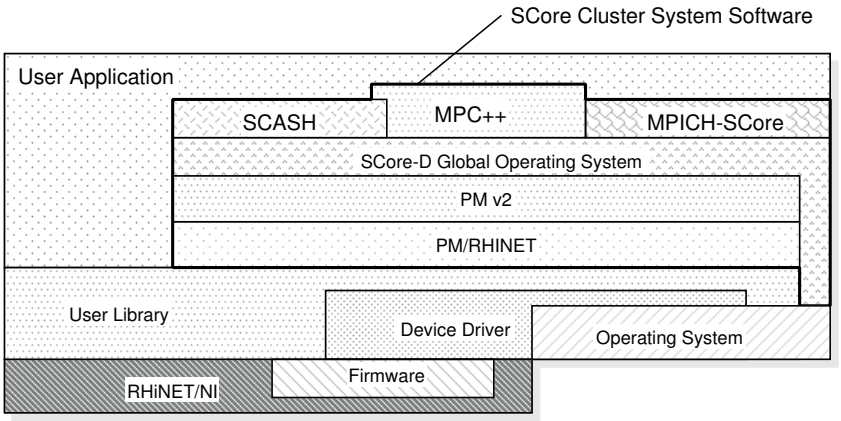


図 2.8 RHiNET のソフトウェアレイヤ

行されるファームウェアが位置する. それより上位はホスト PC 上で実行されるソフトウェアレイヤであり, カーネルレベルで実行されるデバイスドライバとユーザレベルで実行されるユーザライブラリが位置している.

ユーザライブラリはデバイスドライバや OS を介することなく RHiNET/NI に直接アクセスし, プリミティブやそのほかの処理を要求することが可能である. メモリ管理機能などの, 通信と直接関わらない処理はユーザライブラリやデバイスドライバから OS の機能を利用して実現する.

ユーザはユーザライブラリを直接用いて並列アプリケーションを記述することができるが, PM/RHiNET[37] と呼ばれる, SCore システム [38] の通信ライブラリを用いることで, SCore システムを RHiNET 上に実現し, その上で並列アプリケーションを記述することも可能である.

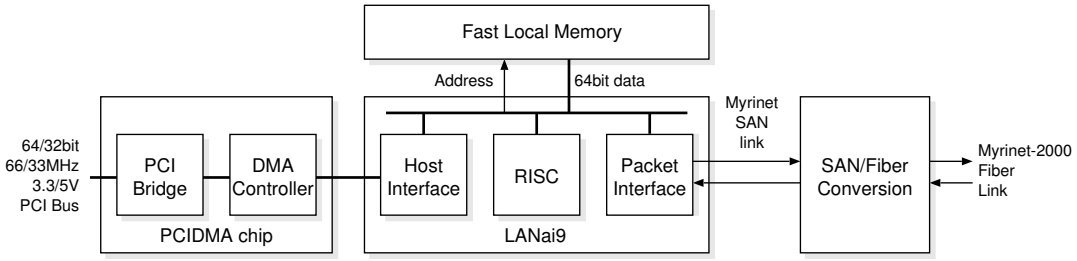


図 2.9 Myrinet-2000 ネットワークインタフェースのブロック図

2.2.2 Myrinet

Myrinet[2] は細粒度並列処理に対応した並列計算機である Caltech Mosaic C[39], 及び Mosaic に用いられた USC/ISI ATOMIC LAN [40] の研究成果から生まれたインターコネクトである. 専用の RISC プロセッサを搭載したネットワークインタフェースと, 専用のネットワークスイッチ, 及びそれらを接続するリンクにより構成されている. Myrinet は ANSI で規格化されており, リンクとルーティングの規格は公開されている. 現在, 米 Myricom 社 [41] によって開発, 販売が行われている.

Myrinet のネットワークインタフェースは LANai(注 3) と呼ばれるネットワークインタフェースコントローラと大容量の SRAM を搭載する. LANai は内部に 32bit の RISC プロセッサを持ち, ネットワークインタフェース上でのプロトコル処理は RISC プロセッサ上で実行される MCP (Myrinet Control Program) と呼ばれるファームウェアによって実現される. SRAM は通信バッファなどに用いられる. また, LANai 外部の専用コントローラ(注 4) によって, ホスト PC 上の物理メモリやネットワークとの間での DMA 転送が提供されている. 通信の信頼性と順序性は MCP によって保証される.

LANai は仕様が公開されているため, ユーザが MCP を開発することも可能であり, PM[42][43][44] や BIP[45] などの, MCP を独自に開発することで高性能な通信を実現しようとする研究が数多く見られる. 図 2.9 に第 3 世代の Myrinet である Myrinet-2000 用のネットワークインタフェースの構成を示す. 図の中央には, LANai9[46] と呼ばれるコントローラが位置している.

Myrinet のスイッチは, カットスルー方式でパケットのスイッチングを行うクロスバスイッチであり, Stop-and-Go 方式のパケット転送を行う. 8×8 や 16×16 のクロスバスイッチをバックプレーンを介して多段接続し, Fat-Tree や Clos 網と呼ばれるトポロジの結合網を構築してノード間を接続する. 16×16 のクロスバスイッチを組み合せて Fat-Tree を構築し, 128 ノードの接続に対応した Myrinet の結合網を図 2.10 に示す. このような結合網上でノード側でソースルーティングによる経路選択を行うことで, トラフィックの分散や経路の冗長化を実現する.

Myrinet は信頼性の高いリンクを用いており, エラー発生率は低いが, さらに CRC (Cyclic Redundancy Check) を用いたエラー検出を提供している.

2.2.2.1 第 1 世代および第 2 世代の Myrinet

1994 年に登場した最初の Myrinet は, Sun Microsystems 社のワークステーションをホストとしてサポートしており, ネットワークインタフェースは SPARC 向けのバスである SBus を介してホ

(注 3)最近のものは “Lanai” と表記が変更されている.
(注 4)最近の Myrinet のネットワークインタフェースではこれらは LANai に統合されている.

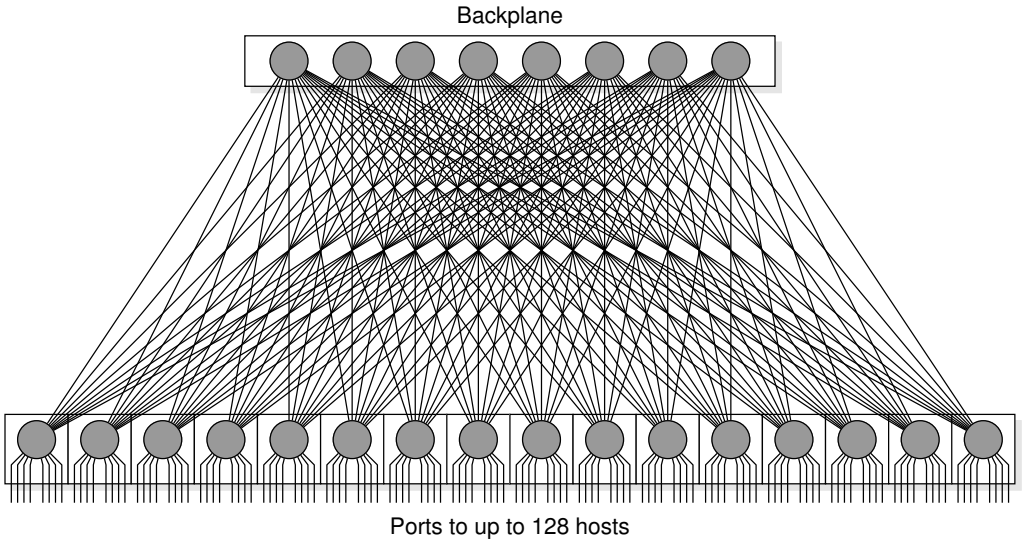


図 2.10　16×16 のクロスバスイッチを多段結合して Fat-Tree を構築した Myrinet の結合網

ストと接続可能であった. リンク速度は 0.64G+0.64Gbps であった.

1990 年代後半に登場した第 2 世代の Myrinet である Myrinet-1280 はリンク速度が 1.28G+1.28Gbps に強化された. ネットワークインタフェースには 33MHz 動作の LANai (LANai4[47]) と最大 1MByte の SRAM が搭載され, SBus に加えて新たに 32bit/33MHz の PCI バスへの対応が行われた. LANai は改良を加えられ, LANai5[48] からは外部へのバスが 64bit に拡張された. また, LANai7[49] は 66MHz で動作し, 64bit/66MHz PCI バス対応のネットワークインタフェースも登場した.

この頃の Myrinet のリンク媒体には銅線が用いられており. SAN モードと呼ばれる接続方式では最大 3m, LAN モードと呼ばれる接続でも最大 10m と, Ethernet などの LAN と比べてリンク長に厳しい制限が存在していた.

2.2.2.2 Myrinet-2000

2000 年頃に登場した第 3 世代の Myrinet は Myrinet-2000 と呼ばれ, リンク速度が 2.0G+2.0Gbps に向上した. 当初はネットワークインタフェースとして 64bit/66MHz PCI バスに対応したもの (最大 200MHz 動作の LANai (LANai9[46]) を搭載) が提供されていたが, 後に 64bit/133MHz の PCI-X バスに接続可能なもの (最大 333MHz 動作の Lanai (LanaiX[50]) を搭載) が登場している. PCI-X バスの転送能力と比較してリンク速度は低いが, この問題を回避するためにネットワークインタフェース上のポート数を 2 ポートに増やして, ノード間のデータ転送速度の強化を図ったデュアルポート形式のネットワークインタフェースも提供されている.

ネットワークスイッチには XBar16 や XBar32 と呼ばれるスイッチチップが用いられ, 最大 32 ポートのクロスバスイッチが提供されている.

リンクの媒体は光ファイバ (50/125 マルチモードファイバ) が標準となっており, 最大で 200m まで延長可能となっている. 光ファイバ以外にも, 銅線によるシリアル接続である HSSDC (High Speed Serial Data Connector, 最大 10m), 及び内部に全二重のリンクを 2 つ備える Myrinet-SAN ケーブル (最大 3m) と呼ばれるマイクロリボンケーブルのそれぞれに対応するネットワークインタフェース

が提供されている.

2.2.2.3 Myri-10G

Myri-10G は 10GbE (10Gigabit Ethernet) との相互運用が可能な Myricom 社による最新のインターコネクトである. IEEE 802.3ak や IEEE 802.3ae といった 10GbE と同じ物理層の規格を採用しており, スイッチやネットワークインタフェースのリンクの媒体には 10GBASE-CX4 規格の銅線や 10GBASE-R 系列の光ファイバの利用が可能である. Myri-10G のリンク速度は 10G+10Gbps である. それに対してネットワークインタフェースは PCI Express x8 を介した接続に対応しているため, ホストとの間は 16G+16Gbps の全二重接続となり, リンクに対して十分なデータ転送性能を提供可能な構成となっている.

Myri-10G のネットワークインタフェースは, 300MHz 以上のクロックで動作する Lanai (Lanai Z8E) を搭載し, 従来の Myrinet と同様にソフトウェアを用いてプロトコル処理を行う. Lanai 上のファームウェアを用いて, データリンクレベルで Myrinet と 10GbE の両方のプロトコルに対応することができるため, Myri-10G のネットワークインタフェースは 10GbE のネットワークインタフェースとしても利用可能である.

2.2.2.4 Myrinet のソフトウェア

Myrinet 向けのソフトウェア環境については, Myricom 社により低レベル通信ライブラリである GM が提供されており, GM を利用した MPI や TCP/IP の実装が用意されている. また, MX (Myrinet eXpress) と呼ばれる, より低遅延な通信を提供する通信ライブラリが用意されている [51].

2.2.3 Quadrics Network

QsNET (Quadrics NETwork)[3][52][53] は, Compaq と Los Alamos 国立研究所の共同プロジェクトである 30 TeraOps ASCI Q-machine と呼ばれる SMP クラスタ用に開発されたインターコネクトである. 現在は Quadrics 社 [54] によって開発・販売が行われている. QsNET はプログラマブルなネットワークインタフェースと高スループットかつ低レイテンシなネットワークスイッチで構成される.

QsNET のネットワークインタフェースには Elan と呼ばれるプログラマブルな ASIC が搭載される. Elan は内部に DMA 要求やパケット処理などの通信処理を専門に行うプロセッサと, MPI などの上位の通信プロトコルを実装するのに用いるプロセッサを内蔵する. また, Elan は MMU を内蔵しており, Elan 内部の処理で用いられる仮想アドレスを，ホスト上のメモリやネットワークインタフェース上の外部メモリの物理アドレスに高速に変換できるようになっている. さらに, ネットワークインタフェース上のメモリをユーザのメモリ空間にマップすることも可能であり, これにより, Elan からホスト上のメモリとネットワークインタフェース上のメモリに対して透過的にアクセスすることが可能になる. そのほか，Elan はプロセッサ用のキャッシュやリンク接続用のロジックなどを内蔵している.

QsNET のスイッチには Elite と呼ばれる ASIC が搭載されている. Elite は 4-array Fat-Tree トポロジを基本とするフルクロスバスイッチである. Elite はソースルーティングに対応し, パケットの先頭についたタグの並びに応じてルーティングを行う. タグは任意の出力ポートの集合を指すこと

ができ, これを利用したハードウェアマルチキャストがサポートされている. パケットは Wormhole 方式で転送され, 送信元ノードと宛先ノードの間の経路は, 宛先ノードがパケットを受け取ってから送信元ノードに確認応答 (Ack) を返すまで維持される.

さらに, 複数レイヤで構成される通信ライブラリを提供しており, これらハードウェアとソフトウェアを組み合わせて, 安全で効率的にアクセス可能なグローバル仮想アドレス空間を構築する.

QsNET はリンクレベルでエラー検出と再送を行うことで, 高い信頼性を実現している.

2.2.3.1 QsNET のプロトタイプ

QsNET は Meiko Scientific 社によって開発された MPP (Massively Parallel Processing) である Meiko CS-2[55] に用いられたインターコネクトが元になっている. CS-2 は SPARC (SuperSPARC または hyperSPARC) とメモリ, 通信用 Co-processor を 1PE とするシステムであり, この通信用の Co-processor は Elan と名付けられた. Elan は SPARC 向けのバスである MBus 経由でアクセスされ, Elan のインターコネクト側のリンク速度は 0.4+0.4Gbps であった.

インターコネクトには Fat-Tree のトポロジが用いられ, ネットワークスイッチに用いられた 8×8 のクロスバスイッチは Elite と呼ばれた. Elite はマルチキャストに対応していた.

2.2.3.2 QsNET

QsNET はネットワークインタフェースコントローラに Elan3 を搭載した 64bit/66MHz PCI バスに装着するネットワークインタフェースと, Elite3 を搭載したネットワークスイッチから構成される.

Elan3 は 100MHz で動作する通信処理専用プロセッサ (マイクロコードプロセッサ) と, 上位プロトコル処理用プロセッサ (スレッドプロセッサ) を持つ. どちらも 32bit プロセッサであり, スレッドプロセッサは 8KByte の 4-way セットアソシアティブキャッシュを持つほか, 通信処理やスケジューリングを支援する拡張命令を備えている. また, Elan3 は 32bit の仮想アドレスを 28bit のローカル SDRAM アドレス, または 48bit のホスト物理アドレスへ変換する MMU, 送受信で各々2 つの仮想チャネルを持つ Link ロジックなどを搭載している. Elan3 のブロック図を図 2.11 に示す.

Elite3 は 2 つの仮想チャネルを備えた 8 つの双方向リンクを持つ 16×8 のフルクロスバスイッチ (入力ポートはチャネルごとに独立) であり, 3.2G+3.2Gbps のリンク速度に対応する. また, CRC によるパケットエラー検出やリカバリにより, 信頼性の高い通信を実現する.

2.2.3.3 QsNET II

QsNET II[56][57] はネットワークインタフェースコントローラに Elan4 を搭載した 64bit/133MHz PCI-X バスに装着するネットワークインタフェースと, Elite4 を搭載したネットワークスイッチから構成される.

Elan4 は Elan3 と比べて, 内部プロセッサの 64bit 化, 並びに 64bit の仮想アドレススペースのサポート, 短パケット処理専用ユニット STEN (Small Transaction ENgine) の搭載といった変更が加えられている. また, Elan4 は Elan3 同様, スレッドプロセッサ (図 2.12 の 64bit RISC processor) を持つ一方, マイクロコードプロセッサを持たず, 複数のプロセッサを搭載することにより, これに相当する機能を実現している. 図 2.12 に Elan4 のブロック図を示す.

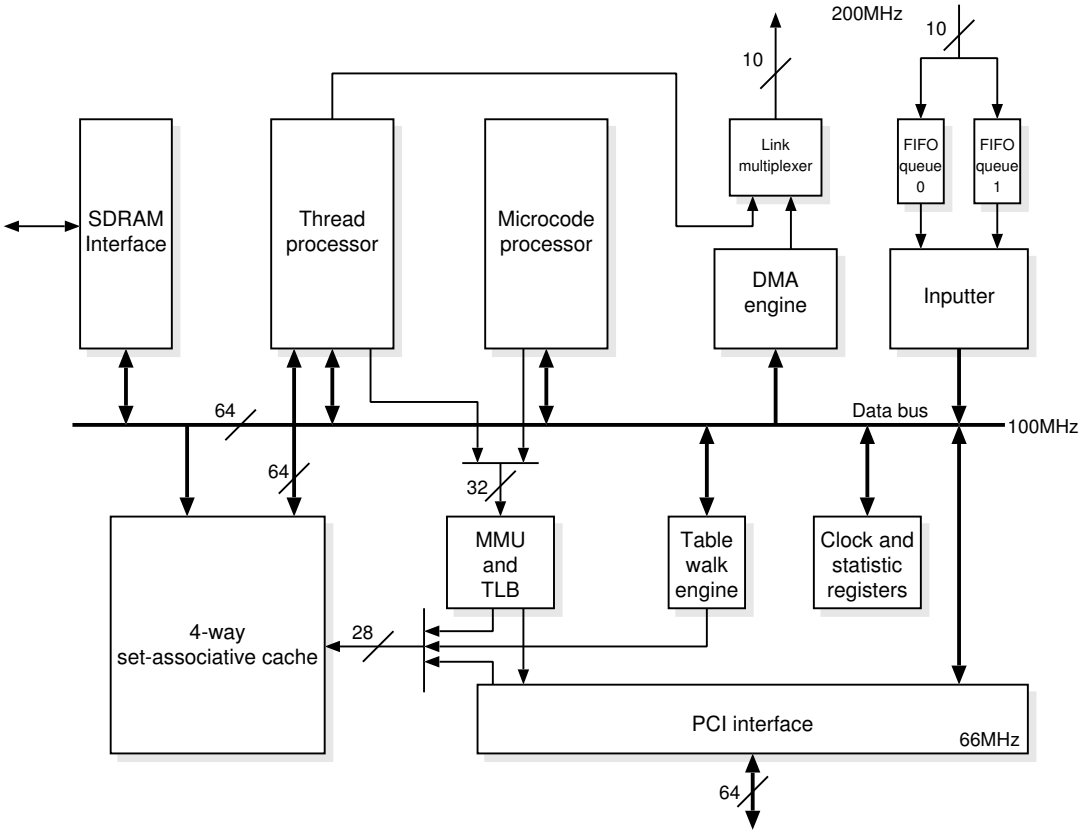


図 2.11 Elan3 のブロック図

Elite4 は 13G+13Gbps (実効データ転送速度は 10.6G+10.6Gbps) のリンク速度, 及び最大 100m のリンク長に対応する.

2.2.3.4 QsNET のソフトウェア

QsNET では, Quadrics 社によって Elanlib[58] と呼ばれる独自のソフトウェアライブラリが提供されており, その上で動作する MPI-2 などの標準的な並列プログラミング環境が用意されている. また, Quadrics 社よりデバイスドライバや qsnetlibs と呼ばれるユーザライブラリなどがオープンソースで提供されているため, ユーザが Elan 上のプログラムを独自に開発することも可能となっている.

2.2.3.5 QsTenG

QsTenG は Quadrics 社が提供する 10GbE のネットワーク環境である [59]. 並列分散処理のみならず, データセンターなどでの利用を視野に入れている. 提供されているのはネットワークスイッチのみであり, ネットワークインタフェースやケーブルは汎用の 10GbE の製品を使用する. ネットワークスイッチにはカットスルー方式を採用しており, スイッチ通過遅延が 200ns と低く抑えられている.

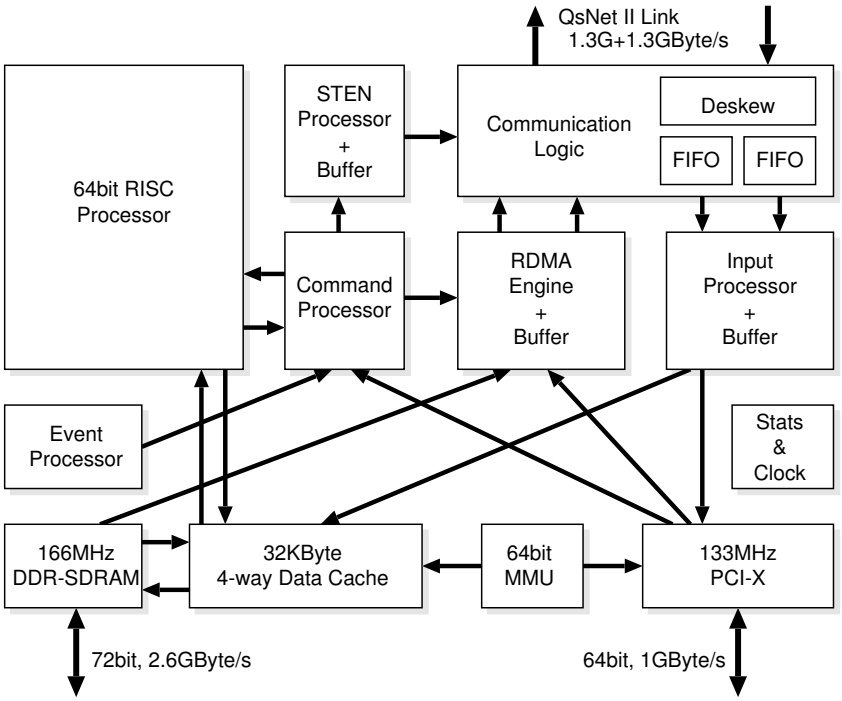


図 2.12 Elan4 のブロック図

2.2.4 InfiniBand

InfiniBand[4] は, IBTA (InfiniBand Trade Association)[60] によって策定されたインターコネクトアーキテクチャである. InfiniBand はサーバ I/O やサーバ間の通信に RAS (Reliability, Availability, Scalability) を提供することを目的に開発されており, 従来の I/O に用いられていたバスアーキテクチャの性能面の問題を解決するために, スイッチベースアーキテクチャとなっている. 最初の規格として, InfiniBand Architecture Specification Release 1.0[61] が 2000 年に策定されている.

当初, InfiniBand は PCI バスなどの汎用 I/O バスに取って代わる内部インターコネクトと, 外部との接続に用いるクラスタリングという, 異なる分野を統合するインターコネクトとして開発が始められた. しかし, Intel によって PCI バスの後継となるチップ間のインターコネクト規格として PCI-Express が提案され, 汎用 PC において普及したため, 内部インターコネクトとしての必要性は薄れ, 現在はクラスタリングを主なターゲットとしたネットワーク規格となっている.

このように, InfiniBand は規格が標準化されていることから様々なベンダから製品が供給されているため, Myrinet や QsNET よりも量産効果による価格の低下が期待できる. このことから, DIMMnet-2 においてはネットワークインタフェース以外のネットワークコンポーネントに InfiniBand を採用している.

2.2.4.1 InfiniBand のプロトコル

InfiniBand では, 物理層, データリンク層, ネットワーク層, 及びトランスポート層の各通信層においてプロトコルが定められている (図 2.13).

- 物理層 (Physical Layer)

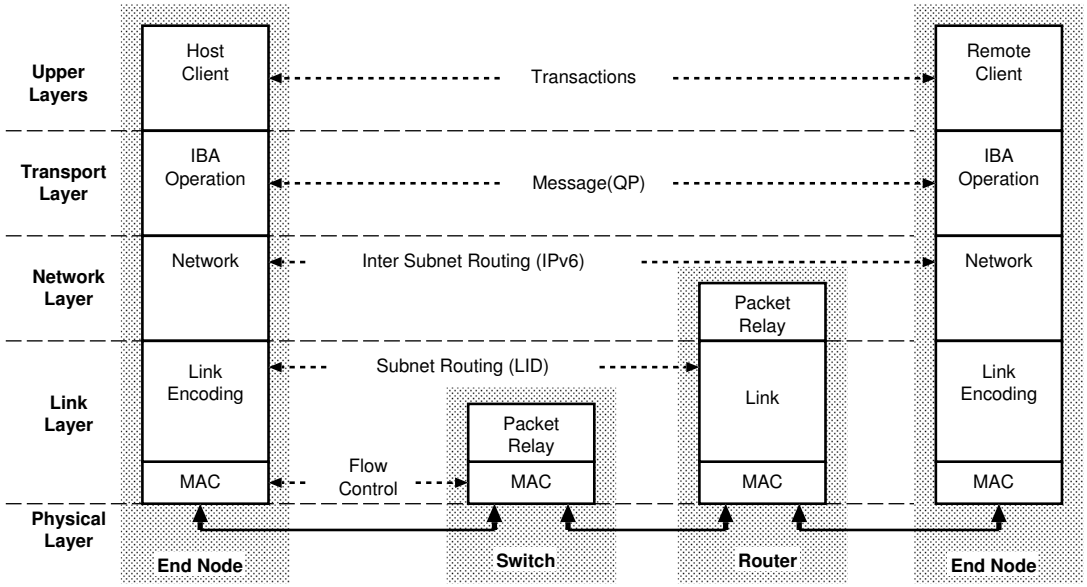


図 2.13　InfiniBand のプロトコル階層

InfiniBand の物理層では, 片方向 2 本のシリアル差動信号方式で伝送を行う双方向の 4 本の信号線を基本構成 (1X) とする. 1X は片方向 2.5Gbps のデータレートを実現するが, 8B10B エンコーディングを用いているため, 実効データ転送レートは片方向当たり 2Gbps となる. 物理層のメディアとしては, プリント基板, 銅線, 及びファイバケーブルが使用可能である.

この 1X の信号線の対を 4 本束ねた 4X や 12 本束ねた 12X が規格として定義されており, それぞれ, 10Gbps, 30Gbps のデータ転送速度を提供する. 2004 年に策定された InfiniBand Architecture Specification Release 1.2[62] では DDR と QDR の動作モードが新たに追加され, 最大で 120Gbps のデータ転送速度が規定されている.

- データリンク層 (Link Layer)

  InfiniBand のデータリンクレベルのパケットには, 通常のデータ転送に用いるデータパケットと, リンクの管理などに用いる管理パケットが存在する. データパケットのサイズは最大 4KByte である.

  InfiniBand において, スイッチで構成されるネットワークをサブネットワークと呼ぶ. サブネットワーク内のパケットスイッチングはリンク層で処理される. 16bit の LID (Local ID) が Subnet Manager (後述) によって各デバイスに割り付けられ, LID を用いてスイッチングする.

  各物理リンクは, VL (Virtual Lane) と呼ばれる論理的な通信路を持っており, 15 本のデータ用 VL (VL0〜VL14) と 1 本の管理用 VL (VL15) が存在する. 数字が大きいほど優先度が高く設定されている. QoS (Quarity of Service) を保証するために, SL (Service Level) が定義されており, スイッチやルータは, SL と VL の対応を保持することで, 適切な QoS を提供することが可能となる.

  VL ごとに受信側が送信側に受信可能なデータ量をクレジットという形で通知することでフロー制御を実現する, クレジットベースのフローコントロールが Point-to-Point で用いられる.

  データリンク層の CRC には, VCRC (Variant CRC) と ICRC (Invariant CRC) の 2 つの CRC

が存在する. VCRC がホップ間でリンクレベルのデータの整合性を保証し, ICRC が end-to-end のデータ整合性を保証する. Ethernet などの CRC が一つしかないプロトコルでは, デバイスで発生したエラーも含めて CRC が再計算されてしまうのに対し, InfiniBand では ICRC により常にエラーを検出できる.

スイッチではルーティングテーブルを用いたルーティングを行う. スイッチにおいてマルチキャストを行うことも可能である.

- ネットワーク層 (Network Layer)

  ネットワーク層はサブネットワーク間のルーティングを行う. サブネットワーク間の通信には InfiniBand ルータが用いられ, IPv6 (Internet Protocol version 6) で表記された送信元と送信先の情報で通信を行う.

- トランスポート層 (Transport Layer)

  トランスポート層では, Send Queue と Receive Queue から構成される QP (Queue Pair) という送受信用のキューを用いた通信が提供される. QP 間でパケットシーケンス番号を用いた Ack/Nack プロトコルにより, ノード間のパケット到達保証を行う.

  QP を用いた通信では, コネクション型の通信とデータグラム型の通信が規定されており, それぞれに対して, パケットの到達保証と順序保証が行われる信頼性のあるモード (RC：Reliable Connection, RD：Reliable Datagram) と不正なパケットの破棄のみが行われ, 再送処理が行われない信頼性のないモード (UC：Unreliable Connection, UD：Unreliable Datagram) が利用可能である.

  また, QP を用いないデータグラム型の通信モード (Raw Datagram) も提供されている.

2.2.4.2 InfiniBand の構成要素

InfiniBand は CA (Channel Adapter) と呼ばれるホストインタフェース, スイッチ, ルータの各種ハードウェアとサブネットワークの管理を行う Subnet Manager から構成される.

CA はノード間接続に用いられる HCA (Host Channel Adapter) と, ストレージなどの I/O デバイスとの接続に用いられる TCA (Target Channel Adapter) の 2 種類が存在する. スイッチは LID を用いてサブネットワーク内のパケットの転送を行い, ルータはサブネットワーク間のパケットのルーティングを行う.

CA を搭載したノードとスイッチから構成されるサブネットワーク内には Subnet Manager が少なくとも 1 つ存在する必要がある. Subnet Manager はスイッチやルータの管理, リンクの up/down 時の再設定を行う. そのため, すべての CA, 及びスイッチには Subnet Manager と通信するための SMA (Subnet Manager Agent) が実装されている必要がある. Subnet Manager はサブネットワーク内のどのデバイス上にも存在することが可能である.

2.2.4.3 高性能通信へのアプローチ

InfiniBand は様々なベンダが製品をリリースしており, 代表的なベンダとして, Mellanox 社 [63] が挙げられる. Mellanox 社の HCA には InfiniHost という高性能なネットワークインタフェースコントローラが搭載されており, 通信処理をネットワークインタフェースコントローラ上で行うことでホストプロセッサの負荷を低減している.

これに対し, QLogic 社 [64] の InfiniPath[65] (注 5)ではネットワークインタフェースコントローラは最低限のデータ送受信の機能のみを提供し, それ以外の複雑な処理はすべてホストプロセッサで実行することで, 通信性能を向上させるアプローチをとっている. これは,

1. ネットワークインタフェースコントローラはホストプロセッサに比べて動作周波数が低く, 複雑な処理をネットワークインタフェースコントローラで実行した場合に通信性能が低下する可能性があること
2. 今後, ホストプロセッサがマルチコア化し, ホストプロセッサの演算能力がインターコネクトの性能に比べて大幅に高くなると予想されること

が背景として挙げられる.

MPI レベルの最小通信レイテンシを比較すると, Mellanox 社の HCA が約 4.0μs なのに対し, InfiniPath の通信性能は 1.29μs と通信レイテンシが低く抑えられている [66][67]. しかし, InfiniPath のアプローチでは, ホストプロセッサのキャッシュが通信処理で使用され, アプリケーションの性能に影響を与える可能性があるため, 一概に InfiniPath のアプローチが良いとは言い切れない.

2.2.4.4 InfiniBand のソフトウェア

InfiniBand の仕様では VIA (Virtual Interface Architecture)[68] を拡張した verb と呼ばれる, CA の提供すべき機能のみが定められている. 具体的な API の定義は各ベンダに委ねられており, Mellanox 社による VAPI[69] などが存在する. VAPI はメッセージ通信と RMA (Remote Memory Access) 型の通信の両方をサポートし, InfiniBand で規定されているトランスポートサービスのうち RC と UD を提供している. また, オハイオ州立大学によって MVAPICH[70] と呼ばれる, VAPI 上で動作する MPI が実装されている.

2.2.5 10Gigabit Ethernet

近年, 並列分散処理環境用インターコネクトにおいて, Myri-10G や InfiniBand のように物理的に 10GbE と互換性を持ち, Ethernet としても用いることが可能なインターコネクトが登場している. また, 10GbE のスループットが PC クラスタ向けインターコネクトに匹敵しており, 量産効果により導入コストが下がっていくことが確実であることから, 今後, PC クラスタへの導入が進むことが期待される.

10GbE のネットワークインタフェースは既に多くのベンダからリリースされており, Intel 社をはじめ, Chelsio Communications 社 [71], Neterion 社 [72], NetEffect 社 [73], NetXen 社 [74], Tehuti Networks 社 [75], LeWiz Communications 社 [76] などがベンダとして挙げられる.

これらのネットワークインタフェースはいずれも TCP の処理の一部, またはすべてをネットワークインタフェースコントローラ上で実行でき, TCP/IP 使用時のホストオーバヘッドの低減が図られている. これは, ネットワークの速度が 10Gbps クラスになると, ネットワークインタフェースコントローラのサポートがない場合, 通信処理だけでホストプロセッサの処理能力の大半が消費されてしまうためである. また, これらのネットワークインタフェースのうち, Chelsio Communications 社, NetEffect 社, NetXen 社の製品では, iSCSI のアクセラレーションや, iWARP 準拠の RDMA の機

(注 5)もともと, InfiniPath は PathScale 社が開発, 販売していたが, 2006 年に QLogic 社に買収された

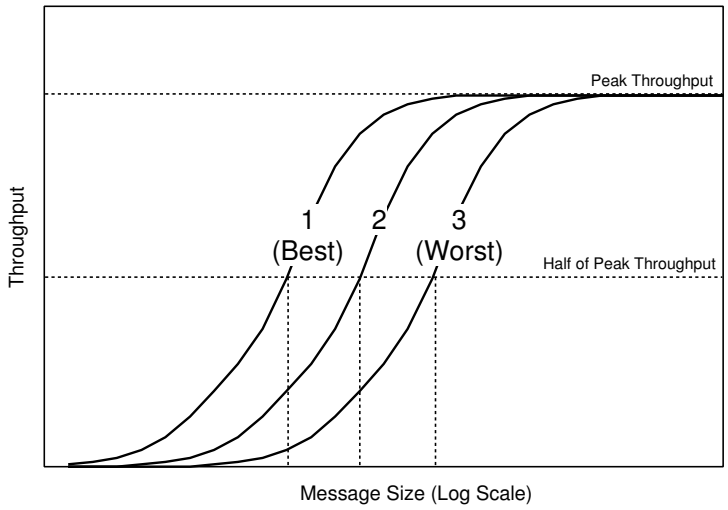


図 2.14　スループットの上昇曲線

能を提供している. iWARP は RDMA Consortium[77] によって策定された, TCP/IP 上での RDMA の利用 (RDMA over TCP/IP) に関する規格である.

これらの機能により, 10GbE の性能はスループットの面においては Myri-10G や InfiniBand に匹敵する性能を達成している. また, これまで Ethernet のネットワークスイッチは Store and Forward 方式のものが大多数であり, HPC 用途で用いるにはレイテンシが大きかったが, QsTenG のようにカットスルー方式のネットワークスイッチがベンダより提供されるようになってきたため, レイテンシも低く抑えることが可能になっていくと考えられる.

2.2.6　まとめ

本節においては, 並列分散処理環境向けに開発されたインターコネクトについてまとめた.

最後にこれらのインターコネクトの性能の比較を表 2.2 に示す. 表 2.2 内に示した通信性能は RHiNET-2, Myrinet-2000, 及び InfiniPath を除き, 参考とした文献の片方向のスループットのグラフから読み取ったものである. また, 括弧で囲まれているものは, MPI レベルの通信性能である. half of peak throughput は片方向通信時に最大スループットの半分のスループットが得られるデータサイズを示しており, インターコネクトのレイテンシの小ささを示す指標の 1 つである [66].

一般に, インターコネクトのスループットは, 図 2.14 のように, あるデータサイズから急激にスループットが上昇し, ある一定値に落ち着くというグラフを描く. これは, 転送するデータサイズが小さい間は通信のセットアップに要するレイテンシが通信処理全体のレイテンシに大きな影響を与えるためである. 通信のセットアップに要するレイテンシは転送するデータサイズに依存せず一定である場合が多く, 転送するデータサイズが大きくなるに従って, このレイテンシの影響は小さくなる. よって, このレイテンシが小さいほど, スループットが上昇し始めるのが早くなる. 図 2.14 では, 1〜3 のグラフはすべて最大スループットは同じであるが, 1 のグラフが最も低レイテンシであると言える.

half of peak throughput の値が大きいと, スループットの最大性能が得られるようになる転送データサイズが大きいということになり, そのようなサイズのメッセージを転送しないアプリケーション

表 2.1　転送されるメッセージサイズと個数

Application	<2K	2K - 16K	16K - 1M	>1M
IS	14	11	0	11
CG	16113	0	11856	0
MG	1607	630	3702	0
LU	100021	0	1008	0
FT	24	0	0	22
SP	9	0	9636	0
BT	9	0	4836	0
Sweep3D-50	19236	0	0	0
Sweep3D-150	28836	28800	0	0

においては, インターコネクトの性能の恩恵を受けられない. 表 2.1 は NAS Parallel Benchmarks[78] の Class B, 及び ASCI Sweep3D ベンチマーク [79] を 8 ノード (16CPU) の PC クラスタで動かした際のメッセージサイズごとのメッセージの転送回数を示したものである [80]. これらのベンチマークのうち, CG, MG, LU, 及び Sweep3D においてはメッセージサイズが 2KByte 未満のメッセージが多数転送されており, このようなアプリケーションにおいては, 最大スループットが高いがレイテンシが大きいインターコネクトよりも, 多少スループットが劣っても低レイテンシなインターコネクトの方が良い結果を得る可能性が高いと言える.

表 2.2 に示したインターコネクトのうち, PCI, 及び PCI-X に接続されながらも, 専用の低遅延通信機構を持つ RHiNET-2, 及び QsNET II のレイテンシが PCI-Express に接続されるインターコネクトよりも低い値を示している. コントローラの動作周波数が 300MHz を超える Myrinet-2000 よりも 66MHz の RHiNET-2 の方がレイテンシが小さいことから, 通信処理をハードワイヤードで実現する方がレイテンシを小さく抑えられると言える. 基本通信処理がハードワイヤードで実現されている割には InfiniBand (Mellanox 製) のレイテンシが大きいが, これは InfiniBand のプロトコル処理がほかのインターコネクトに比べて複雑であることが原因と考えられる. また, PCI バスや PCI-Express よりも低レイテンシな I/O バスに接続される InfiniPath が最も低いレイテンシを示していることから, I/O バスのレイテンシが性能に与える影響は大きいと思われる.

一方, スループットは接続される I/O バスの最大スループットに近い値が出ており, また, I/O バスの世代が新しくなるにつれ, それに従った性能が得られていると言える. さらに, 片方向の I/O バスである PCI バスや PCI-X バスから双方向の I/O バスである PCI-Express や HyperTransport に置き換わったことが大きな影響を与えている.

2.3　メッセージ通信を支援する通信機構

近年, PC クラスタ向けインターコネクトにおいて, ネットワークインタフェースに搭載されているコントローラを利用して, MPI に代表されるメッセージ通信を支援する機構を搭載する研究が行われている. これは, PC クラスタにおいては並列アプリケーションを記述する際に, メッセージ通信ライブラリが多く用いられていることを背景としている. メッセージ通信では, メッセージの受信処理時に受信関数で指定したパラメータに一致するメッセージを探索するなどの処理が行われ

る. そのため, 基本通信性能が高いインターコネクトであっても, インターコネクト側でメッセージ通信を支援する機能がない場合, メッセージ通信レベルの通信レイテンシやホストプロセッサのオーバヘッドが高くなる. その結果, このようなアプリケーションにおいては基本通信性能から期待されるほどの性能が得られないことになる.

本節では受信側のネットワークインタフェースコントローラがメッセージの受信先アドレスを指定する方式のメッセージ受信機構, 及びネットワークインタフェースコントローラが MPI のメッセージ受信処理をホストプロセッサに代わって実行する手法について述べる. また, それらの手法が抱えている問題点を明らかにする.

2.3.1 受信側がメッセージの受信先アドレスを指定する受信機構

PC クラスタに代表される分散メモリ型の並列システムにおいては, 各ノードで動作するプロセス間の通信にメッセージ通信が用いられる. メッセージ通信において各プロセスはメッセージを明示的に送信, 及び受信することで並列処理を実現する. メッセージの送受信において, RDMA をサポートしないインターコネクトの場合, 一度カーネルのバッファにメッセージをコピーするなどの操作により, メッセージの送受信を実現する. しかしながら, カーネルを介した場合, 受信のオーバヘッド, レイテンシは RDMA より増大する.

一方, ネットワークインタフェースが RDMA をサポートするインターコネクトでは, 受信先のアドレスを事前に通知し合うなどの手段によって送信側が受信先のアドレスを知る必要があるものの, 受信側が明示的な受信処理を行う必要がなく, 受信側ホストのオーバヘッドが小さいという利点がある. そのため, RDMA を用いてメッセージ通信を実現することで高い性能を得ることができる. しかしながら, 複数プロセスから単一の受信領域に対して RDMA を行う場合, 各プロセスは独立に RDMA でメッセージを送信してしまうため, 受信メッセージが書き潰されてしまうことが起こり得る. これを防ぐために, 通信ごとに送信側と受信側で転送サイズやメッセージ転送領域を通知し合うか, 受信側で送信プロセスごとに受信領域を分けるといった対策をとる必要がある. 前者の場合, 通信レイテンシの増大を招くことになり, 後者の場合, 通信プロセス数が増大した際のスケーラビリティに乏しいという問題が存在する上に, メッセージを到着した順番に読み出すことが難しくなる.

このような背景から DIMMnet-1 (3.1 節) におけるリモート間接書き込み [87][88] や, RHiNET-2 における VPUSH[89] のような “受信側のネットワークインタフェースコントローラがメッセージの受信先アドレスを指定する” というメッセージ受信の手法が提案された. これらは共に, ネットワークインタフェースコントローラである Martini に搭載されたオンチッププロセッサを用いたソフトウェア処理とハードワイヤードで実装された通信機構が協調処理を行うことによって実現されている.

DIMMnet-1 のリモート間接書き込みでは, 送信側で受信側のメッセージ受信先アドレスが格納された領域を指すポインタ値をセットし, 受信側にメッセージを送信する. 受信側では, そのポインタ値が指す領域に格納されたメッセージ受信先のアドレスを取得し, そのアドレスに従ってメッセージを格納する. 受信先のアドレスの取得, フロー制御, ACK パケットの処理などをすべてオンチッププロセッサで実行していたが, オンチッププロセッサが例外処理用の簡素なものであり, 性能が高くなかったため, RDMA と比較すると大幅に性能が低下していた.

RHiNET-2 で実装された VPUSH では, 受信したメッセージのアドレス値の変更, 及び受信領域のアドレス管理のみをオンチッププロセッサで行い, パケットのヘッダ解析や主記憶への DMA は

ハードワイヤード部で行う. VPUSH は RDMA の最大スループットの 38%程度の性能を示したが, やはりオンチッププロセッサの性能から, RDMA と比べてスループットの低下, 受信側のレイテンシの増加が見られていた [30][89]. VPUSH ではパケット長によらず, オンチッププロセッサの処理時間が 6.6μs で一定である. この値はデータサイズが 2048Byte のパケット受信の処理時間の約 60%に相当する. 文献 [89] によると, データサイズが 2048Byte 時の VPUSH のスループットが 180.3MByte/s であり, 仮にこのソフトウェアオーバヘッドが 0 になった場合は処理時間が約 40%程度になることから, スループットは約 450MByte/s に到達すると予測される. 実際にはソフトウェアの処理と同等の機能を持つハードウェアによるオーバヘッドが加わり, 若干低い値になると考えられるが, この値は RHiNET-2 の最大スループットに匹敵する値である.

2.3.2 ネットワークインタフェースコントローラによる MPI のメッセージ受信処理の高速化

MPI ではメッセージ受信の際に用いるメッセージキューとして posted receive queue と unexpected message queue の 2 つのキューを用いて, メッセージの受信処理を行う. posted receive queue は受信要求がメッセージの到着前に発行された際に, その要求をバッファリングするためのものであり, unexpected message queue はメッセージの到着が対応する受信要求より先であった場合に, 受信データをバッファリングするためのものである.

メッセージを受信した際の処理の流れを以下に示す.

1. posted receive queue を探索し, そのメッセージに対応する受信要求が存在するかどうかを調べる.
2. 対応する受信要求が存在する場合には, その要求に対するメッセージとして受信側のプロセスに渡され, posted receive queue から受信要求が削除される.
3. 対応する受信要求が存在しない場合には, 受信したメッセージを unexpected message queue に格納する.

受信要求が発行された場合には,

1. unexpected message queue を探索し, 対応する受信メッセージが既に受信されているかどうかを調べる.
2. 既に受信されている場合はそのメッセージが受信側のプロセスに渡される.
3. 受信されていない場合は, その要求が posted receive queue に格納される.

という流れで処理が行われる.

このように, MPI においてはメッセージの受信の際にキューの探索処理が行われるが, アプリケーションによってはキューの探索範囲が大きくなり, その結果, メッセージの受信処理に要するレイテンシが増加するという問題がある [90].

そのため, これらのキューの探索処理をネットワークインタフェースコントローラで行うことでホスト側の負荷の軽減, 及び探索処理の高速化を図る研究が数多く行われている. Myrinet の MX や QsNET の Tports[54] ではマッチングのための API が用意されている. また, MX を利用した MPICH[91] である MPICH-MX が Myricom 社より提供されている. これらはプログラマブルなネットワークインタフェースコントローラを利用し, ソフトウェアで実行される.

それに対し, 文献 [92] では, ALPU (Associative List Processing Unit) という posted receive queue と unexpected message queue の探索処理を行うハードウェアを提案している (図 2.15). ALPU は複数の Cell Block と呼ばれるロジック, 結果を出力するためのマルチプレクサ, 及び制御ロジックから構成される. Cell Block は 2^n 個の Cell から構成される (図 2.16). Cell は単一のマッチング情報の比較を行うユニットである. そのため, ALPU 全体で Cell の個数分のマッチング情報を保持することができる. Cell の個数以内のマッチングにおいては大幅なレイテンシの削減が可能であるが, Cell の個数以上のマッチング情報からマッチングを行う場合には性能が改善しないという問題がある.

文献 [92] では Virtex-II Pro XC2VP100[93] を対象デバイスとしてハードウェア量を評価している. posted receive queue と unexpected message queue に対してそれぞれ Cell の個数が 256 個の ALPU を設けた場合に, 総スライス数の約 62.0%を占めており, ハードウェア量が極めて大きいと言える. さらに, ハードウェア量は Cell の個数にほぼ比例するため, スケーラビリティに乏しい.

2.3.3 まとめ

リモート間接書き込み, VPUSH はいずれもネットワークインタフェースコントローラの性能が低かったために, 高い性能が得られなかった. しかし, VPUSH ではソフトウェア処理を極力少なくすることでリモート間接書き込みよりも高い性能を示していたことから, ハードワイヤードで通信処理を実現するアプローチは性能面で有効であると言える.

Tports や MX におけるマッチング処理のネットワークインタフェースコントローラへのオフローディングはホストプロセッサの負荷を軽減するという点では有効であるが, ホストプロセッサの性能に比べてネットワークインタフェースコントローラの性能は低いため, 今後, マルチコア化などによりホストプロセッサとの性能差が広がると, ネットワークインタフェースコントローラがシステムのボトルネックになりうる. また, ALPU のように, マッチング処理をハードワイヤードで実現した場合, ソフトウェアで実現するよりも高い性能が得られると考えられるが, ネットワークインタフェースコントローラのハードウェア量が増大するため, 性能とのトレードオフを考える必要がある. さらに, これらの手法はマッチング処理時のキューの探索範囲を削減する手法ではないため, 文献 [90] で挙げられている問題を本質的には解決していない.

7 章では, メッセージ通信を支援する通信機構である IPUSH と LHS について述べる. これらの通信機構は “ホストプロセッサが効率良く処理できるようにするためのメッセージ受信機構” であり, ホストプロセッサとハードワイヤードによる通信機構の協調処理を行うというアプローチを採っている. どちらの機構もキューの探索範囲を削減することができるため, 本節で述べた手法の問題点を解決可能である.

表 2.2　各インターコネクトの比較

インターコネクト	RHiNET-2[34]	Myrinet-2000[81]	QsNET II[82]	InfiniBand[83]
ネットワークインタフェース	RHiNET-2/NI	M3F2-PCIXE	QM500	InfiniHost
コントローラ	Martini	LANai-2XP	Elan4	MT23108
ベンダ	original	Myricom	Quadrics	Mellanox
基本通信処理	Hardware	Firmware	Hardware	Hardware
コアクロック	66MHz	333MHz	N/A	N/A
ホストインタフェース	PCI 64bit/66MHz	PCI-X 64bit/133MHz	PCI-X 64bit/133MHz	PCI-X 64bit/133MHz
リンク性能 (実効データ転送速度)	600MByte/s×2	250MByte/s×2×2	1.06GByte/s×2	1GByte/s×2 (4X)
パケット転送方式	VCT	VCT	Wormhole	VCT
トポロジ	Any	Any	Fat Tree	Any
ホストプロセッサ	Pentium III 933MHz×2	Opteron 1.8GHz×2	Itanium2 N/A	Xeon 3.4GHz×2
チップセット	ServerWorks ServerSet III LE	AMD AMD-8131	Intel E8870	N/A
ファームウェア		MX-2G 1.0		
接続方式	back-to-back	1 Myrinet switch (0.5μs)	N/A	InfiniScale
レイテンシ	1.74μs	2.6μs	1.6μs	4.8μs
スループット (片方向) half of peak throughput	444MByte/s 1KByte	495MByte/s 300Byte	911MByte/s 4K - 8KByte	781MByte/s 1K - 2KByte
スループット (双方向)	470MByte/s	912MByte/s	900MByte/s	946MByte/s

インターコネクト	InfiniBand[83]	InfiniBand [67]	Myri-10G[84]	10GbE[85][86]
ネットワークインタフェース	InfiniHost III Ex	InfiniPath HTX	10G-PCIE-8A-C	T110
コントローラ	MT25208	N/A	Lanai Z8E	N/A
ベンダ	Mellanox	QLogic	Myricom	Chelsio
基本通信処理	Hardware	Hardware	Firmware	Hardware
コアクロック	N/A	N/A	300MHz〜	N/A
ホストインタフェース	PCI-Express 8x	HyperTransport	PCI Express 8x	PCI-X 64bit/133MHz
リンク性能 (実効データ転送速度)	1GByte/s×2 (4X)	2GByte/s	1.25GByte/s×2	1.25GByte/s
パケット転送方式	VCT	VCT	VCT, SAF	SAF
トポロジ	Any	Any	Any	Any
ホストプロセッサ	Xeon 3.4GHz×2	N/A	Dual-Core Xeon 3.2GHz×2	Opteron 2.2GHz with TOE
チップセット	N/A	N/A	Intel 5000P	N/A
ファームウェア			MX-10G 1.2.0h	
接続方式	InfiniScale	N/A	back-to-back	back-to-back
レイテンシ	3.8μs	(1.29μs)	(2.2μs)	8.9μs
スループット (片方向) half of peak throughput	972MByte/s 1K - 2KByte	(954MByte/s) (88Byte)	(1215MByte/s) (4K - 16KByte)	950MByte/s 256 - 512Byte
スループット (双方向)	1932MByte/s	(1884MByte/s)	(2355MByte/s)	N/A

VCT：Virtual Cut Through

SAF：Store And Forward

TOE：TCP Offload Engine

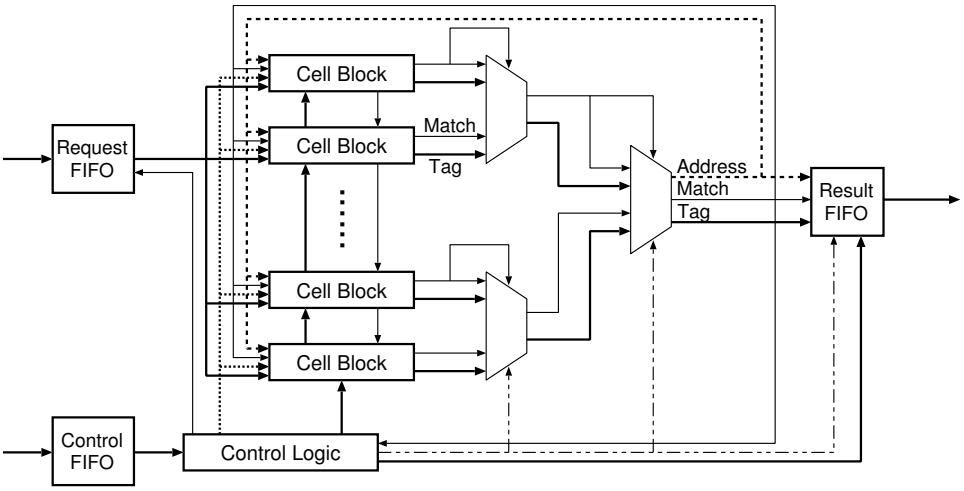


図 2.15　ALPU のブロック図

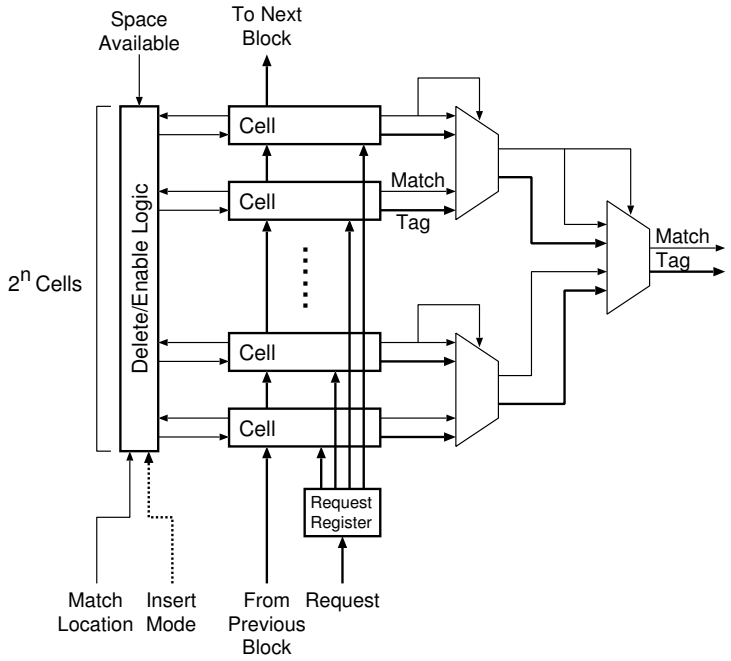


図 2.16　Cell Block のブロック図

第3章　DIMMnet

本章では, DIMM スロット装着型ネットワークインタフェースである DIMMnet について述べる. まず, 最初のプロトタイプである DIMMnet-1 について触れ, DIMMnet-1 を運用した結果, 明らかになった問題点について述べる. そして, DIMMnet-2 の概要と, DIMMnet-1 の問題点の解決方法, 及び試作基板について述べる. また, メモリスロットにネットワークインタフェースを装着することによって発生する問題点について, DIMMnet-2 の場合に解決可能であるかどうか考察する. 最後に, 本研究の目的について述べる.

3.1　DIMMnet-1

DIMMnet-1[35] は, MEMOnet の有効性を実証するために, DIMMnet のプロトタイプとして東京農工大学中條研究室と新情報処理開発機構によって共同開発された, PC100 または PC133 仕様の DIMM スロットに装着するネットワークインタフェースである. ネットワークインタフェースコントローラやネットワークスイッチには RHiNET-2 で用いられた Martini や RHiNET-2/SW を使用していた.

DIMMnet-1 の主な仕様を表 3.1 に示す. また, その基本構造を図 3.1 に, 基板の概観を図 3.2 に示す.

ネットワークインタフェース上にはネットワークインタフェースコントローラである Martini と, 低遅延 FET (Field Effect Transistor) バススイッチを介してアクセス可能な 2 枚の SO-DIMM が搭載されている. ホストプロセッサからのデータは図 3.1 に示される Data Bus を通り, FET スイッチを介して SO-DIMM に転送される. 一方, Martini に対する要求は Control Bus を通り, Martini に直接転送される.

2 枚の SO-DIMM はバンク構成になっている. Control Bus 経由で Martini にアクセスすることで FET スイッチを操作し, 2 枚の SO-DIMM を切り替えて (バンク切り替え) ホストプロセッサとネットワーク間で共有可能な 2 ポートメモリ構造を実現している. 図 3.1 は FET SW2 と FET SW3 が “ON” の状態であり, SO-DIMM1 がホストプロセッサ側, SO-DIMM2 がネットワーク側と接続され

表 3.1　DIMMnet-1 の主な仕様

ホストとのインタフェース	PC100,PC133 DIMM, 及び PEMM[94]
コントローラ	Martini
搭載メモリ	PC133, SO-DIMM ×2
搭載可能 SO-DIMM 容量	64MByte〜1GByte
通信リンクスループット	各方向 8Gbps (全二重)
リンク性能	800MByte/s (PC100), 1024MByte/s (PC133)

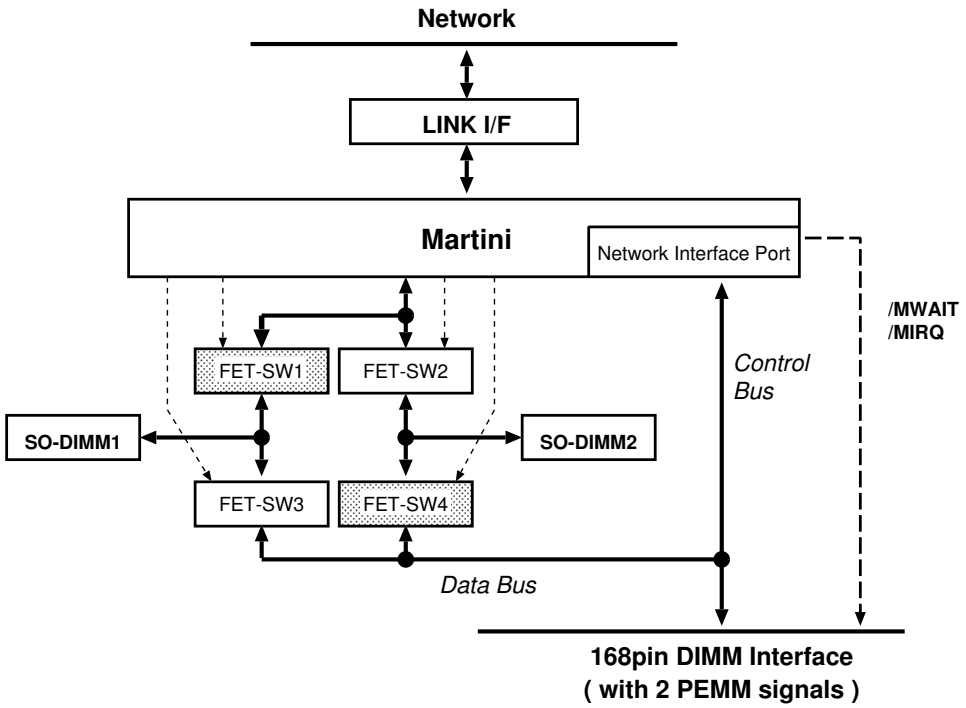


図 3.1 DIMMnet-1 の基本構造

ている様子を示したものである.

メモリバス側のインタフェースは, 日本電子機械工業会規格の “プロセッサ搭載メモリ・モジュール (PEMM) 動作仕様標準 [94]” にも対応しており, PEMM で追加された 2 つの信号 (メモリへのアクセスを待たせる信号と割り込み信号) も有する.

3.1.1 DIMMnet-1 の問題点

DIMMnet-1 の運用の結果, 次のような問題点が明らかとなった.

- ネットワークインタフェースコントローラである Martini が基本的な RDMA read/write の機能しか提供していなかったため, 複雑な処理はコントローラ内部のオンチッププロセッサを用いてファームウェアで実現する必要があった. しかしながら, オンチッププロセッサの性能が低かったため, 複雑な処理を行うと通信性能が低下してしまっていた.
- DIMMnet-1 は PC133 の規格の SDR-SDRAM バスに対応可能なように設計されたが, 実際には PC100 の場合にのみ安定して動作した. ネットワークインタフェース上の SO-DIMM に対するアクセスを FET スイッチを介して行う構造になっていたことが原因と考えられる. FET スイッチにより, 本来メモリスロットに想定されていない容量性負荷が加わってしまい, PC133 の規格では安定動作しなかったものと推測される. そのため, DIMMnet-1 の構造ではより高速な DDR-SDRAM バスなどには対応できないと言える.

DIMMnet-2 ではこれらの問題を解決するために, ネットワークインタフェースコントローラで行う処理をすべてハードワイヤードで実現し, ネットワークインタフェース上の SO-DIMM に対し

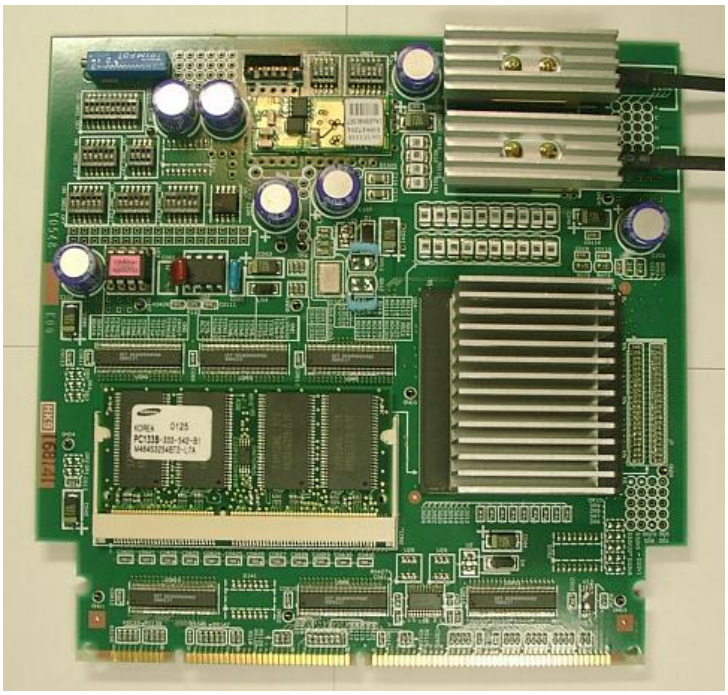


図 3.2　DIMMnet-1

てはホストプロセッサから直接アクセスできない構造を採用する. 次節でこれらの詳細について述べる.

3.2 DIMMnet-2

DIMMnet-2 プロジェクトは DIMMnet-1 の経験を活かして次世代のネットワークインタフェースを開発し, より低コストで高性能な PC クラスタを実現することを目標として 2002 年度に立ち上げられた [95].

メモリバスに接続し, ネットワークインタフェース上にネットワークインタフェースコントローラのほかに SO-DIMM を搭載する点では同様であるが, DIMMnet-2 では様々な改良が施されている. DIMMnet-1 と DIMMnet-2 の相違点を以下にまとめる.

- DDR-SDRAM バスへ接続

  DIMMnet-2 のプロジェクトが開始された 2002 年度には PC における主記憶の主流が DDR-SDARM に移り変わっていたことから, DIMMnet-2 は DDR-SDRAM バスに接続する. また, ネットワークインタフェース上に搭載されるメモリも DDR SO-DIMM となっている.

- ホストからネットワークインタフェース上の SO-DIMM への間接アクセス方式の採用

  DIMMnet-1 における FET スイッチの問題から, DIMMnet-2 ではネットワークインタフェース上の SO-DIMM に対して直接アクセスせず, ネットワークインタフェースコントローラ内部のバッファやレジスタを介してアクセスする間接アクセス方式を採用した. このようにすることで, SDR-SDRAM バスよりも高速な DDR-SDRAM バスに対応可能であり, さらには,

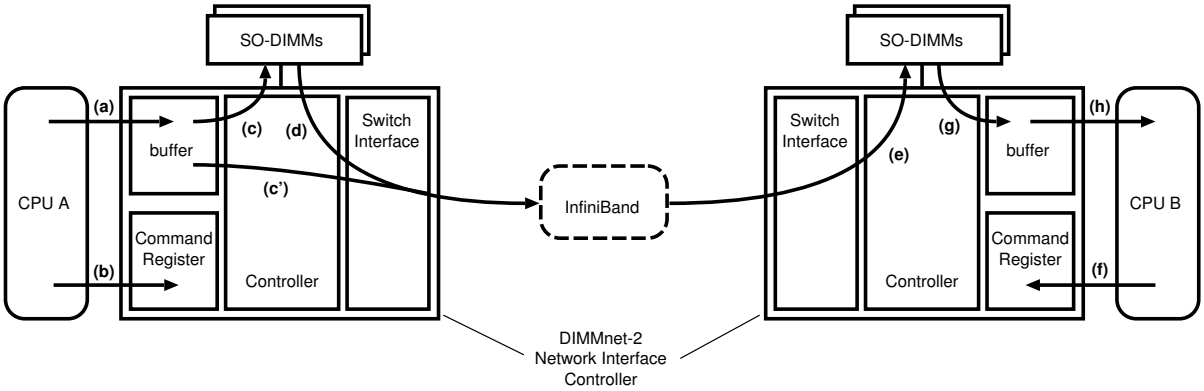


図 3.3　間接アクセス方式

より高速な DDR2-SDARM バスにも対応可能であると考えられる.

また, ローカルのネットワークインタフェース上の SO-DIMM とリモートの SO-DIMM の双方に対するアクセスがコントローラ内部のバッファやレジスタを介して行われるようになるため, 双方に対して, 統一した方式でアクセスすることが可能となる.

図 3.3 に間接アクセス方式の概略を示す. ネットワークインタフェース上の SO-DIMM に対して書き込みを行う際には,

(a) コントローラ内部のバッファにデータを書き込む.

(b) そのデータを SO-DIMM に転送する要求を要求発行レジスタ (Command Register) に発行する.

という手順を踏むことで実行される. リモートの SO-DIMM にデータを転送する際には, そのような要求を発行することでバッファからネットワークにデータが送出される (c'). バッファからではなく, ローカルの SO-DIMM からリモートの SO-DIMM にデータを転送することも可能である (d→e). 要求発行レジスタに SO-DIMM のデータを読み出す要求を発行すると (f), SO-DIMM からコントローラ内部のバッファにデータが転送される (g). このデータをホストから読み出すことで (h), SO-DIMM 内部のデータを参照することができる.

- ネットワークインタフェース上の SO-DIMM への不連続アクセスのハードウェアサポート

  コントローラ内部のバッファとネットワークインタフェース上の SO-DIMM 間の転送機能を拡張し, ベクトル型スーパーコンピュータで行われているような不連続領域に対するアクセスをハードウェアでサポートする [96][97][98].

  このような機能を追加することによって, ホストプロセッサのキャッシュのヒット率やメモリバスの利用効率を向上させることが可能になり, DIMMnet-2 をネットワークインタフェースとしてではなく, 高機能なメモリモジュールとして用いることでアプリケーションの性能を向上させることが可能となる (図 3.4).

  図 3.4 は, ホストプロセッサがネットワークインタフェース上の SO-DIMM 領域のうち, 斜線部分のデータを必要としている場合のプリフェッチの動作を示したものである. 本来, このように不連続な領域のデータを必要とする場合, ホストプロセッサは最大で, 必要とするデータの個数回だけメモリアクセスを実行する必要がある. しかし, 近年の PC におけるプロ

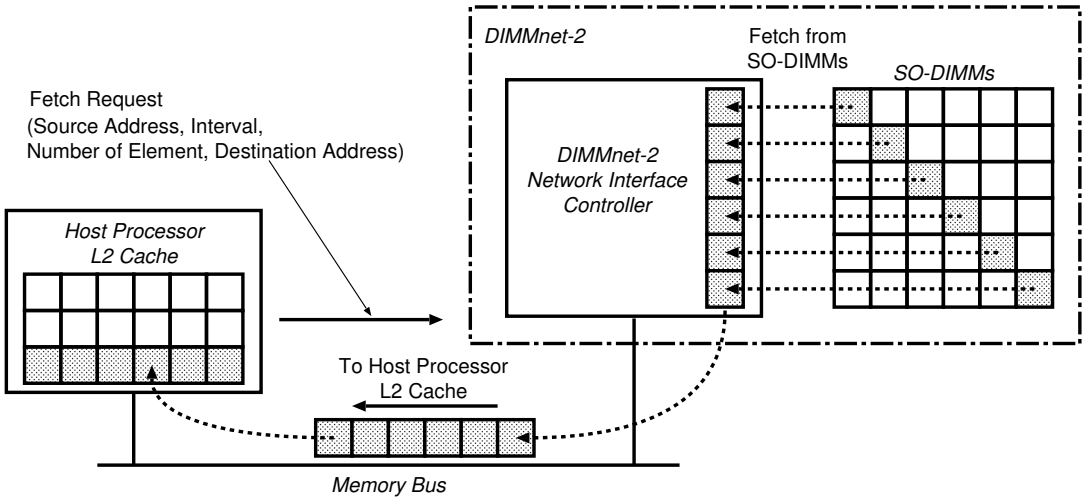


図 3.4　不連続アクセス機構

セッサでは L2 キャッシュのキャッシュラインサイズ単位でデータの読み出しを行うため, メモリバスを流れるデータや L2 キャッシュに格納されるデータのうち, ホストプロセッサが必要とするデータが含まれる割合が低下し, メモリバスの利用効率や L2 キャッシュのヒット率が低下する.

それに対して DIMMnet-2 は, ホストプロセッサからはデータ間の間隔 (インターバル) や要素数を指定したアクセス要求を 1 回発行するだけで, DIMMnet-2 のコントローラが必要なデータを SO-DIMM から読み出し, ホストプロセッサから直接アクセス可能な位置にある, コントローラ内部のバッファに格納する. ホストプロセッサはこのバッファからデータを読み出す. これにより, メモリバスの利用効率や L2 キャッシュのヒット率が改善される.

- 商用コンポーネントの採用

  ネットワークスイッチやケーブルといった, ネットワークインタフェース以外のコンポーネントには標準的な規格である InfiniBand の製品を採用し, 独自規格の製品を用いていた DIMMnet-1 と比較して汎用性の向上とコストの低減を図っている.

3.2.1　DIMMnet-2 試作基板

DIMMnet-2 はネットワークインタフェースコントローラに FPGA を搭載した試作基板を用いて研究・開発が行われている [11][12]. 内部論理の変更が可能な FPGA を用いることで, 様々な機能の実装や検証を容易にしている. 図 3.5 に試作基板の構成図を, 図 3.6 に概観を示す.

- Xilinx Virtex-II Pro XC2VP70-7FF1517C

  ネットワークインタフェースのコアとなるコントローラ部は Xilinx 社 [99] の Virtex-II Pro XC2VP70-7FF1517C[93] 上に実装される. このチップは 8,272 個の Configuration Logic Block (CLB) と 5,904Kbit の内部 RAM (BRAM：Block RAM), 964 本のユーザ I/O ピンを持っており, さらに PowerPC を 2 個, RocketIO トランシーバ [100] を 16 個備えた, Virtex-II Pro の中

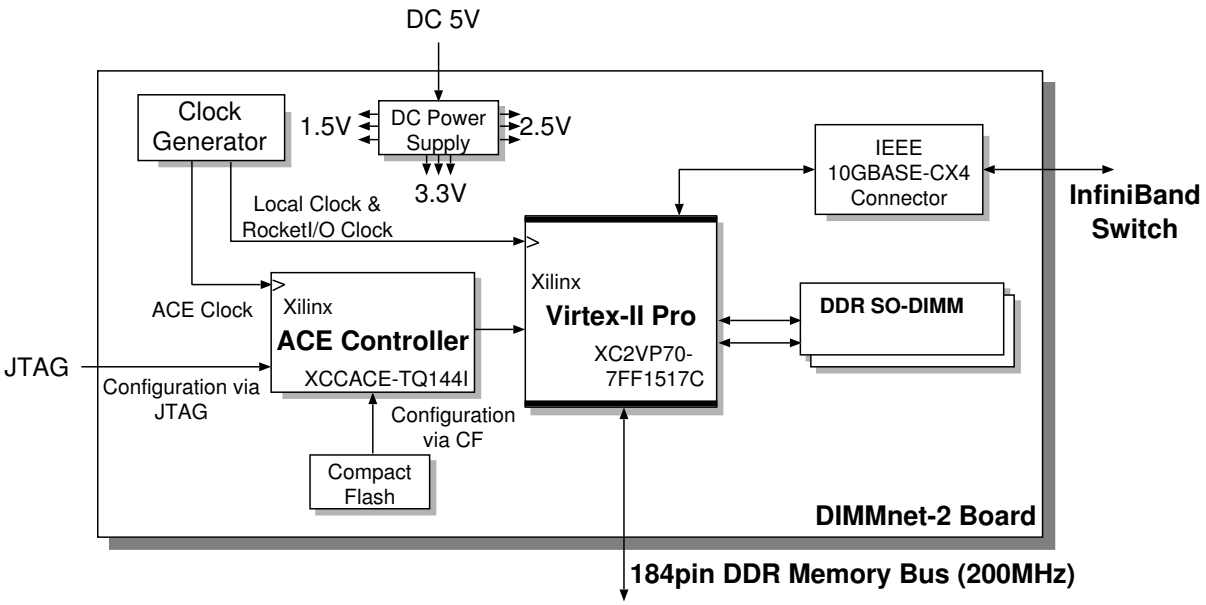


図 3.5 DIMMnet-2 試作基板の構成図

でも大規模でハイエンドな FPGA である.

RocketIO トランシーバは InfiniBand, Fibre Channel や Gigabit Ethernet に対応する高速シリアル I/O インタフェースであり, これを利用して InfiniBand (4X：10Gbps) に接続する.

- Xilinx System ACE Controller

  FPGA のコンフィギュレーションには Xilinx 社より提供されている, 大規模 FPGA のコンフィギュレーションソリューションである System ACE Compact Flash[101] を用いている. Virtex-II Pro のような大規模な FPGA には大容量のコンフィギュレーションメモリが必要となる. そのため, コンフィギュレーションメモリに PROM を用いると部品点数が増加し, 基板面積を圧迫してしまう. また JTAG 経由のコンフィギュレーションは複数のノードから構成される PC クラスタを構築した場合に使い勝手が悪いため, コンパクトフラッシュからのコンフィギュレーションを採用した.

- DDR SO-DIMM × 2

  ネットワークインタフェース上にはノート PC 用の汎用メモリである 200pin DDR SO-DIMM が 2 枚搭載される. これらは通信用のバッファに使用されるほか, ホスト PC のデータ記憶領域として用いられる. SO-DIMM を 2 枚搭載し, それぞれの SO-DIMM に対して FPGA から独立にアクセスすることで, Dual Channel 動作を実現でき, SO-DIMM アクセス時のスループットが向上する.

  DIMMnet-2 は DDR-SDRAM メモリバスに接続されるため, 最大で PC-3200 の規格に対応したメモリバスに装着される. この場合, ホストから DIMMnet-2 のコントローラ内部へのバッファに対しては PC-3200 の速度 (3.2GByte/s) でアクセスされることになる. もし, 転送先である SO-DIMM が SDR SO-DIMM であった場合, PC133 の SO-DIMM を 2 枚使用して Dual Channel 動作させた際の最大転送スループットは 133 [MHz] × 64 [bit] × 2 (Dual Channel) = 2GByte/s となり, ホストからのデータ転送速度に比して, SO-DIMM への転送速度が低く,

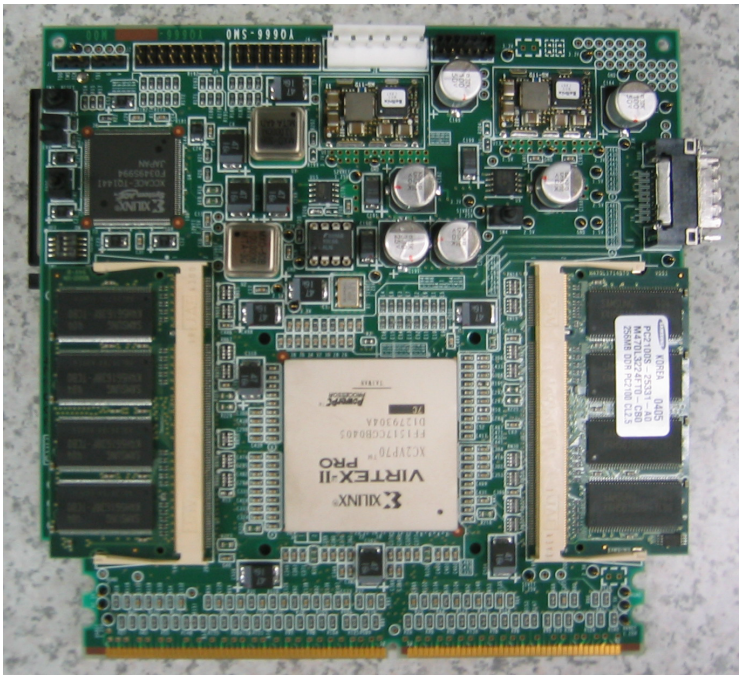


図 3.6　DIMMnet-2 試作基板

SO-DIMM へのアクセスがボトルネックとなる. このことから, DDR SO-DIMM を採用した.

試作基板では 256MByte の SO-DIMM を 2 枚, 計 512MByte 分のメモリを搭載しているが, より容量の大きい SO-DIMM を搭載することで PC のメモリスロット 1 本当たりに搭載可能な最大容量のメモリよりも大規模なメモリ空間を提供することが可能である.

- IEEE 10GBASE-CX4 Connector

  InfiniBand スイッチとの接続のために IEEE 10GBASE-CX4 のコネクタを搭載している. 従って, FPGA 内部の論理を変更することで, InfiniBand のみならず, 10GbE にも接続することが可能である.

- DDR Host Interface

  PC の DDR-SDRAM スロットに装着可能とするために, 184pin DDR-SDRAM インタフェースを備える.

試作基板は PC のメモリバスに装着するため, PC 起動時に BIOS によりアクセスされる. その際に正常に PC を起動させるには, FPGA が実際のメモリのように振る舞い, BIOS のアクセスに応答しなければならない. 従って, 試作基板を運用する場合は, 電源を投入してから BIOS によるメモリバスへのアクセスが始まるまでにコンフィギュレーションが完了している必要がある. マザーボードによってはコンフィギュレーションが間に合わない場合があり, その場合はコンフィギュレーションが完了した後に PC をリセットする必要がある. PC リセット時にはメモリバスへの電源供給は絶たれないため, FPGA のコンフィギュレーションは保持されたままとなる.

試作基板は FPGA を用いているため, 高い動作周波数での稼働が困難である. そのため, ホスト側のインタフェースを 100MHz で動作させることで PC-1600 の規格でのみ, 動作可能としている.

3.3 メモリスロットにネットワークインタフェースを装着することによる問題点

メモリスロットにネットワークインタフェースを装着することで汎用 I/O バスに装着した場合よりも低レイテンシ, 高スループットな通信を実現できると考えられるが, メモリスロットを使用することによるデメリットも存在する.

3.3.1 PC に搭載可能な主記憶の最大容量の問題

一般的な PC に搭載されているメモリスロットは, 2〜4 本程度であり, ネットワークインタフェースをメモリスロットに装着すると, その分だけ主記憶として使用できるメモリが減ってしまうことになる.

しかし, DIMMnet-2 はネットワークインタフェース上に大容量の SO-DIMM を搭載しており, DIMMnet-2 同士をネットワークで相互接続することで大規模なメモリ空間を提供することが可能である. そのため, 科学技術計算などのスーパーコンピュータを必要とするアプリケーションにおいては, アプリケーションで使用するデータのすべて, または一部をネットワークインタフェース上に展開されるメモリ空間に配置し, 演算に必要なデータをその都度ホスト側に読み出すことで, メモリ容量の欠点を補えると考えられる.

また, 一般的な PC に搭載可能な最大メモリ容量は高々4〜8GByte 程度であり, PC を用いて大規模なアプリケーションを動かすには, PC クラスタのように複数の PC を使用することで広いメモリ空間を実現する必要があるため, DIMMnet-2 によってメモリスロットを消費することに対するデメリットは大きな問題とならないと思われる. さらに, 疎行列を含む連立方程式など, メモリに対して不連続なアクセスを伴うアプリケーションにおいては不連続アクセスをサポートしている DIMMnet-2 を用いることで高い効果を期待できる [96][97][102][103][104].

3.3.2 Dual Channel 動作への対応の問題

DDR-SDRAM が PC の主記憶に採用されるようになってきた頃から, 主記憶へのバスを 2 系統搭載し, Dual Channel 動作が可能なチップセットが PC に搭載されてきた. Dual Channel はそれぞれのメモリバス (チャネル) に同時にアクセスを行うことでメモリバスのスループットを 2 倍にする技術である. Dual Channel 動作には, それぞれのチャネルに同一の速度, 容量を持ったメモリモジュールを装着することが求められる.

しかし, DIMMnet-2 は通常のメモリモジュールとは異なるため, Dual Channel 動作でのメモリアクセスができなくなり, メモリバスのスループットが低下するという問題がある. Dual Channel 動作を可能にするための対応として,

- DIMMnet-2 をそれぞれのチャネルに搭載する.
- DIMMnet-2 が装着されているチャネルと対になるメモリスロット(注 1)に DIMMnet と同じ SPD 情報を持つダミーモジュールを装着する (図 3.7).

ということが考えられる. しかしながら, DIMMnet-2 を 2 枚搭載し, Dual Channel 動作を実現した場合, 1 台のホストに LID が 2 つ割り振られてしまう, 2 枚の DIMMnet-2 へ同時にアクセスしてし

(注 1)図 3.7 中の channel A-1 と B-1, A-2 と B-2 のように, 各チャネルの同じブランチ同士のこと

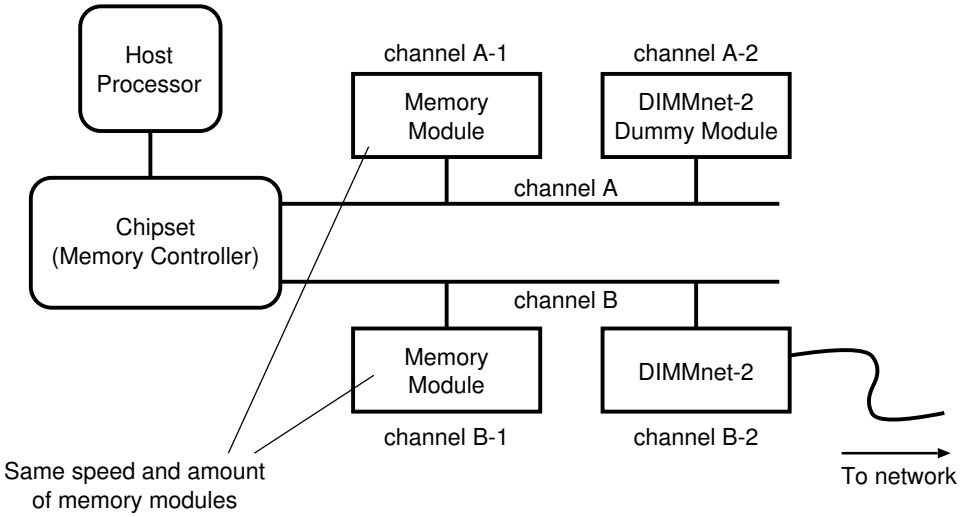


図 3.7 Dual Channel 動作への対応

まうなど, ホスト側からの扱いが複雑になるという欠点が存在する.

3.4 本研究の目的

本研究では DIMMnet-2 ネットワークインタフェースコントローラの設計, 及び実装を行い, 汎用 PC を用いた場合でも DIMMnet-2 と組み合わせることで, サーバやワークステーションと PC クラスタ向けインターコネクトを組み合わせたシステムに匹敵する通信性能を達成することを目的とする.

通信レイテンシに関してはメモリバスを利用することで, 汎用 I/O バスを用いたインターコネクトよりも低くすることは容易であると考えられる. しかし, 主記憶上に存在するデータを転送する際に高いスループットを得るのは難しいと推測される. 主記憶と DIMMnet-2 が共にメモリバスに存在するため, 主記憶のデータを DMA で DIMMnet-2 に転送することはできず, また逆に, DIMMnet-2 上のデータを主記憶に DMA で転送することもできない. そのため, 主記憶と DIMMnet-2 との間のデータ転送は必然的に PIO になるため, 一般に高いスループットは期待できない. 本研究では PIO 通信でも高いスループットを達成可能とするために, 複数の PIO 通信処理をオーバラップさせる手法を提案する. この手法の有効性は 6 章の評価を通して示す.

また, 2.3 節で述べたメッセージ通信支援機構の問題点を解決した通信機構を 7 章で提案し, DIMMnet-2 ネットワークインタフェースコントローラへ実装する. そして, DIMMnet-2 を用いた評価から, 通信性能がどの程度改善されるのかを示す.

第4章　DIMMnet-2ネットワークインタフェースコントローラの設計

本章では, DIMMnet-2 ネットワークインタフェースコントローラの設計について述べる. 本章で述べる内容は DIMMnet-2 試作基板を対象としており, ロジックはすべて FPGA に実装する.

本章以降, アドレスやビットフィールドの表記で様々な形式の数値表現を用いる. ビット幅が重要となる値に関しては HDL (Hardware Description Language) 的な表現を用い, “[ビット幅]’[基数プリフィックス][数値]” の形式で表す. 基数プリフィックスは 16 進数の場合に h, 2 進数の場合に b となる. 例えば, 10 進数の 256 を 16bit の 16 進数で表現すると, 16’h0100 となる. また, ビット幅が重要とならず, 数値のみに意味がある場合はスクリプト言語的な表現を用い, “[基数プリフィックス][数値]” の形式で表す. 基数プリフィックスは 16 進数の場合に 0x, 2 進数の場合に 0b となる. 例えば, 10 進数の 256 を 16 進数で表現すると, 0x100 となる. これらの数値は場合により, 4 桁ずつアンダースコア (_) で区切る.

さらに, DIMMnet-2 内部のレジスタをほかの語句と区別するために Sans Serif フォントを用いて表記する. 例えば, DIMMnet-2 の内部にはネットワークの MTU 値を設定するための MTU という名称のレジスタが存在するが, MTU と表記されていればレジスタを指し, MTU と表記されていれば一般的な Maximum Transmission Unit のことを指す.

4.1　DIMMnet-2ネットワークインタフェースコントローラの概要

DIMMnet-2 ネットワークインタフェースコントローラは汎用プロセッサを搭載せず, すべての処理をハードワイヤードで実現する. 一般にネットワークインターフェースコントローラはホストプロセッサに比べ, 大幅に低い動作周波数で動作するため, 汎用プロセッサ上のファームウェアで通信処理を実現するとレイテンシが大きくなると考えられるためである.

図 4.1 は, DIMMnet-2 ネットワークインタフェースコントローラ部のブロック図である.

コントローラ部は大きく分けて, 4 つのブロックから構成される.

- DDR Host Interface

  ホストプロセッサとのトランザクションを処理するブロックである. ホスト側は DDR メモリのプロトコルに従って動作する. つまり, データの転送はクロックの立ち上がりと立ち下がりの両エッジに対してそれぞれ 64bit ずつ行われる. これに対し, コントローラ内部はクロックの立ち上がりのみの片エッジで動作する. そこで, DDR Host Interface は, この 64bit 幅 両エッジのデータを 128bit 幅 片エッジデータへの変換を行う. また, 128bit 幅 片エッジデータから 64bit 幅 両エッジデータへの変換も行う.

- DDR SO-DIMM Interface

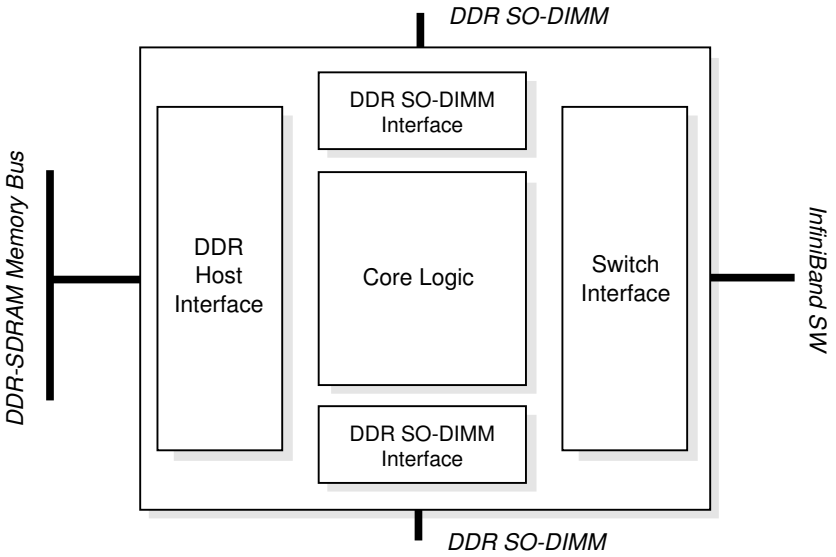


図 4.1　コントローラ部のブロック図

ネットワークインタフェース上の DDR SO-DIMM へのアクセスを制御するブロックである. Core Logic (後述) からの片エッジのデータ転送を DDR SO-DIMM への両エッジ転送に変換する. また, この逆の変換も行う.

- Switch Interface (SWIF)

  InfiniBand ネットワークとのインタフェースである. Core Logic で生成した DIMMnet-2 独自形式のパケットを InfiniBand パケットでカプセル化し, ネットワークにパケットを送出する. また, ネットワークから受信した InfiniBand パケットを DIMMnet-2 独自形式のパケットにアンカプセル化する. End-to-End の再送機構を持ち, SWIF 内部の InfiniBand のレイヤで再送制御を行うことで, DIMMnet-2 のレイヤはネットワークのエラーに関与せずに済む構造になっている.

- Core Logic

  ネットワークインタフェースの制御部である. 送信パケットの生成や受信パケットの解析といった通信処理や, SO-DIMM Interface へのアクセス要求の制御を行う.

InfiniBand のネットワークを流れる信号は 1X 当たり 2.5Gbps であり, Virtex-II Pro に内蔵される RocketIO は入力された信号を 20 倍の周波数で出力する [100]. そのため, SWIF は 125MHz で動作させる必要がある. SWIF 以外のブロックはホスト側の周波数に合わせるために 100MHz で動作させる.

DDR Host Interface, 及び DDR SO-DIMM Interface は日立 JTE が開発を担当し, それを元に本研究で DIMMnet-2 向けに変更を加えた.

SWIF は東京農工大が開発を担当した. DIMMnet-2 では SWIF を Core Logic から分離し, DIMMnet-2 独自形式のパケットを InfiniBand パケットでカプセル化して, ネットワークに送出する. このような形式ではカプセル化/アンカプセル化の分だけレイテンシの増大を招くため, 本来ならば DIMMnet-2 独自形式のパケットを用いずに, InfiniBand パケットを Core Logic で直接生成した方がレイテンシ

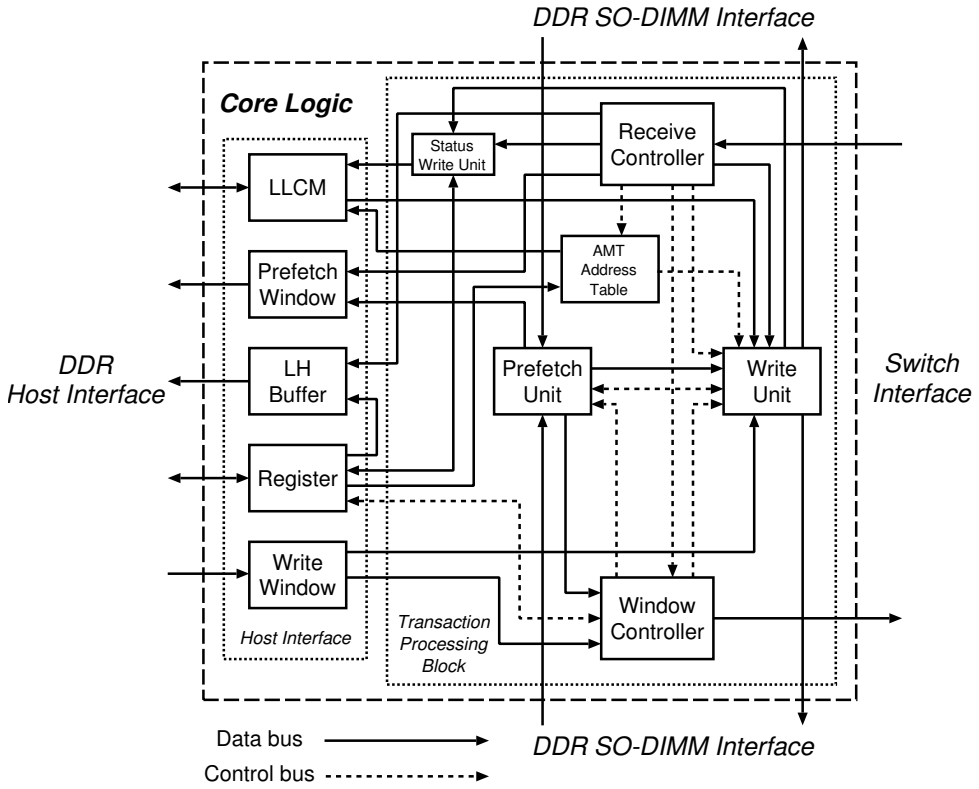


図 4.2　Core Logic 部のブロック図

の削減という点では有利である. しかしながら, SWIF のようにインターコネクトに依存した部分を分離することで, この部分の変更のみで別のインターコネクトへ対応することが可能となり, 汎用性, 柔軟性という点で優れる.

本研究では Core Logic の開発を担当した. PIO 通信を用いた高スループットな通信を実現するためには, ネットワークインタフェースコントローラ内の各モジュールを独立して動作可能にし, 各処理をオーバラップ可能にする必要がある. また, ホストプロセッサからのアクセス時に高いスループットが得られるようにホストインタフェース部を設計する.

次節以降では Core Logic の設計について述べる. メッセージ通信支援機構の設計については 7 章で述べる.

4.2　Core Logic の設計

図 4.2 に Core Logic のブロック図を示す. Core Logic は大きく分けて, ホストから直接アクセスされるホストインタフェース部とパケットの送受信や SO-DIMM へのアクセスを行う要求処理部から構成される.

表 4.1 ホストインタフェース部の各モジュールのアクセス属性と MTRR の設定

Module	Attribute	Host	Controller
LLCM	Uncachable	Read/Write	Read/Write
Prefetch Window	Write Back	Read Only	Write Only
LH Buffer	Write Back	Read Only	Write Only
Register	Uncachable	Read/Write	Read/Write
Write Window	Write Combining	Write Only	Read Only

4.2.1 Core Logic ホストインタフェース部

ホストインタフェース部は, ホストプロセッサがデータの読み書きを行うバッファや, 要求発行などを行うレジスタ群から構成される. 図 4.2 で示されるモジュールのうち, 以下のものがホストインタフェース部に含まれる.

- LLCM (Low Latency Common Memory)

  ホストプロセッサと要求処理部から読み書き可能なメモリ領域であり, パケット受信ステータスやサイズの小さなメッセージを格納するなど, 様々な用途で用いることが可能である.

- Prefetch Window

  ホストプロセッサからは読み出しのみ, 要求処理部からは書き込みのみが可能なメモリ領域であり, ローカルまたはリモートに位置するネットワークインタフェース上の SO-DIMM から読み出したデータを格納するバッファである.

- LH Buffer

  ホストプロセッサからは読み出しのみ, 要求処理部からは書き込みのみが可能なメモリ領域であり, メッセージ通信支援機構である LHS (Limited-length Head Separation) 機構 (7 章) で使用するバッファである.

- Register

  ホストプロセッサからの要求発行や DIMMnet-2 の資源管理, 動作設定のためのレジスタ群である.

- Write Window

  ホストプロセッサからは書き込みのみ, 要求処理部からは読み出しのみが可能なメモリ領域であり, ローカルまたはリモートのネットワークインタフェース上の SO-DIMM に書き込むデータを格納するバッファである.

これらのバッファやレジスタは, ホストプロセッサからは主記憶として認識され, 通常のメモリモジュールにアクセスするのと同一の作法でアクセスされる. また, 各領域を物理的に別のメモリ領域として確保することで, Pilchard[20] のように各領域に対して異なる MTRR の設定を行うことができるため, ホストプロセッサからの読み出し, 書き込みの双方のアクセスを最適化することが可能となる. 各バッファ, レジスタのアクセス属性と MTRR の設定を表 4.1 に示す.

LLCM LLCM はホストプロセッサのキャッシュにキャッシュされない Uncachable 属性とした. LLCM はパケット受信ステータスなどの書き込みによって要求処理部側からデータが変更される. そのため, LLCM をホストプロセッサにキャッシュされる領域に指定すると, 要求処理部からの変更をホストプロセッサから検出できなくなる. Uncachable 属性に指定すると, ホストプロセッサから LLCM へのアクセスの際に, データがバースト転送されなくなり, スループットが著しく低下するという問題があるが, LLCM で扱う個々のデータは少量のデータであるため, 性能への影響は小さいと考えられる.

Prefetch Window, LH Buffer Prefetch Window は LLCM とは異なり, ホストプロセッサにキャッシュされる Write Back 属性である. Prefetch Window も LLCM と同様, 要求処理部によってデータの変更が行われる領域である. しかし, Prefetch Window には SO-DIMM から読み出したデータが格納されるため, ホストプロセッサはある程度サイズの大きいデータを Prefetch Window から読み出す. そのため, Uncachable 属性に指定するとスループットが低下し, 性能に大きな影響を与える. また, Write Combining 属性もホストプロセッサにキャッシュされない属性であるが, この属性は書き込み時のスループットのみが高く, 読み出し時のスループットは Uncachable 属性と変わらない [95].

これらの理由より, Prefetch Window は Write Back 属性とする. Write Back 属性の場合, 読み出したデータはホストプロセッサのキャッシュに格納されるため, キャッシュにヒットする限り高い読み出しスループットを得ることができる. そこで, Prefetch Window からデータを読み出す際には, 以下の手順を踏むことで高いスループットを達成する.

1. 読み出しを行うアプリケーション側でホストプロセッサのキャッシュをライン単位で無効化する命令を用いて Prefetch Window の当該のラインを無効化する.

2. 要求処理部が Prefetch Window にデータを書き込んだ後にプリフェッチ命令を用いて, Prefetch Window のデータをホストプロセッサのキャッシュにプリフェッチする.

このようにすることで, 読み出し時のスループットを改善することができる [97]. Pentium4 以降の x86 系プロセッサの場合, L2 キャッシュをフラッシュする命令として CLFLUSH が, プリフェッチ命令として PREFETCHNTA などが定義されている [105][106]. しかしながら, 十分な性能を得るには, プリフェッチ命令発行後から実際にプリフェッチしたデータを使用するまでにある程度の間隔(注 1)が必要である.

LH Buffer も Prefetch Window と同様の理由から Write Back 属性である.

Register Register はホストプロセッサからの要求発行やネットワークインタフェースの動作設定などを行うための領域である. そのため, ホストプロセッサにキャッシュされる領域に指定すると, ネットワークインタフェースの制御がホストプロセッサ側からできなくなるため, Uncachable 属性としている. Register の用途の性質上, ホストプロセッサからは 64bit または 128bit 程度のサイズのデータしか読み書きを行わないため, LLCM と同様, Uncachable 領域に指定しても性能への影響は小さいと考えられる.

Write Window Write Window はホストプロセッサのキャッシュを使用せず, その上, メモリに対して高いスループットが得られる Write Combining 属性とする. Write Window を Write Back 属性

(注 1)プリフェッチが完了するのに十分な時間

のようにキャッシュされる領域に設定すると, ホストプロセッサから書き込みを行った際にキャッシュに対してのみ書き込まれ, Write Window にデータが書き込まれない. そのため, Write Window はキャッシュされない属性とする必要がある. しかしながら, Uncachable 属性に設定すると, ホストプロセッサからメモリに対してのデータ転送において, バースト転送されなくなるため, スループットが著しく低下する. これらの理由により, Write Window は Write Combining 属性とした.

4.2.2 Core Logic 要求処理部

DIMMnet-2 の Core Logic は 4.2.1 節で示したホストインタフェース部のほかに, 以下の要求処理部から構成される.

- Window Controller

  ホストプロセッサや Receive Controller (後述) からの要求の解釈を行い, それに従って, Write Unit (後述) や Prefetch Unit (後述) の制御やパケットの送信処理を行う. パケット送信処理には, パケットヘッダ生成のほか, 要求された転送サイズが MTU を超える際のパケット分割, Register への処理の完了通知が含まれる.

- Receive Controller

  ネットワーク側から受信したパケットをパケットヘッダに従って処理する. パケットは Prefetch Window または SO-DIMM に受信可能である. Prefetch Window に受信する場合は Receive Controller がデータを書き込むが, SO-DIMM に受信する場合は Write Unit を制御して受信処理を行う. また, リモートプロセスから RDMA read 要求が発行された際には Window Controller に処理の要求を出す.

- Prefetch Unit

  SO-DIMM からのデータ読み出しの制御を行う. アドレスが連続した領域へのアクセスのほか, 等間隔に配置されたデータや, アドレスのリストに基づいたデータに対するアクセスといった不連続領域に対するアクセスの制御も行う.

- Write Unit

  SO-DIMM へのデータ書き込みの制御を行う. Prefetch Unit と同様に, 連続領域に対するアクセスのほか, 不連続領域に対するアクセスも制御する.

- Status Write Unit

  パケットの受信完了後に, Receive Controller からの要求に従って, パケット受信ステータスを LLCM に書き込む. また, ステータスを書き込む領域が full または empty であるかどうかの通知を Register に対して行う. さらに, メッセージ通信支援機構である IPUSH を使用する際に LLCM 上に確保される AMT (Address Management Table) の更新を行う (7 章).

- AMT Address Table :

  IPUSH 使用時に AMT にアクセスする際のアドレスを格納したテーブルである. 詳細は 7 章で述べる.

表 4.2　各バッファ間のデータ転送

	Source	Destination
(1)	Write Window (Local)	SO-DIMM (Local)
(2)		SO-DIMM (Remote)
(3)		Prefetch Window (Remote)
(4)		LLCM (Remote)
(5)	SO-DIMM (Local)	SO-DIMM (Local)
(6)		SO-DIMM (Remote)
(7)		Prefetch Window (Local)
(8)		Prefetch Window (Remote)
(9)	SO-DIMM (Remote)	SO-DIMM (Local)
(10)		Prefetch Window (Local)

4.3　DIMMnet-2 におけるデータ転送

DIMMnet-2 ではデータ転送はホストインタフェースと SO-DIMM 間, または SO-DIMM 同士で行われる. データ転送時のソースとディスティネーションの関係を表 4.2 にまとめる. また, 表 4.2 に対応した図を図 4.3 に示す. 表 4.2 の Local は図 4.3 の CPU A 側, Remote は CPU B 側を示す.

(1)〜(4) の処理は CPU A 側の Write Window のデータを転送する処理である. (1) は CPU A 側の SO-DIMM への書き込みである. (2)〜(4) はホスト上のデータをネットワークに送出するという処理であり, Martini (2.2.1 節) に搭載されていた BOTF 通信機構と同等の通信機構を用いて実現される. 要求処理部からの Write Window に対するアクセスは SO-DIMM へのアクセスよりも低レイテンシで行えるため, SO-DIMM のデータを転送するよりも低レイテンシな通信を実現することが可能である.

(5)〜(8) の処理は CPU A 側の SO-DIMM のデータを転送する処理である. (5) は CPU A 側で行われる SO-DIMM 間のデータコピーである. (6), (8) は CPU B 側とのデータ転送であり, これが DIMMnet-2 における RDMA write に相当する. (7) は CPU A 側の SO-DIMM からのデータ読み出しである.

(9), (10) は CPU B 側の SO-DIMM のデータを CPU A 側に転送する処理であり, これが DIMMnet-2 における RDMA read に相当する.

DIMMnet-2 ではこれらの処理を実現するための命令を基本命令 (プリミティブ) としてハードワイヤードで実現する (4.6 節).

4.4　DIMMnet-2 におけるプロセスの識別

本節では 4.3 節で述べたデータ転送処理を行う際のプロセス, 及びプロセスグループについて述べる. 本節で述べるプロセスとは, 並列処理を実行する際に各ノードで起動されるプロセスを指す.

DIMMnet-2 ではユーザレベル通信を実現する. 従って, DIMMnet-2 を利用するプロセス間で DIMMnet-2 に対する不正な書き込みを防ぐために, 各プロセスに異なるインタフェースを提供する必要がある. しかし, 同時に利用するプロセス数が増加すれば, その分だけ Write Window や Prefetch

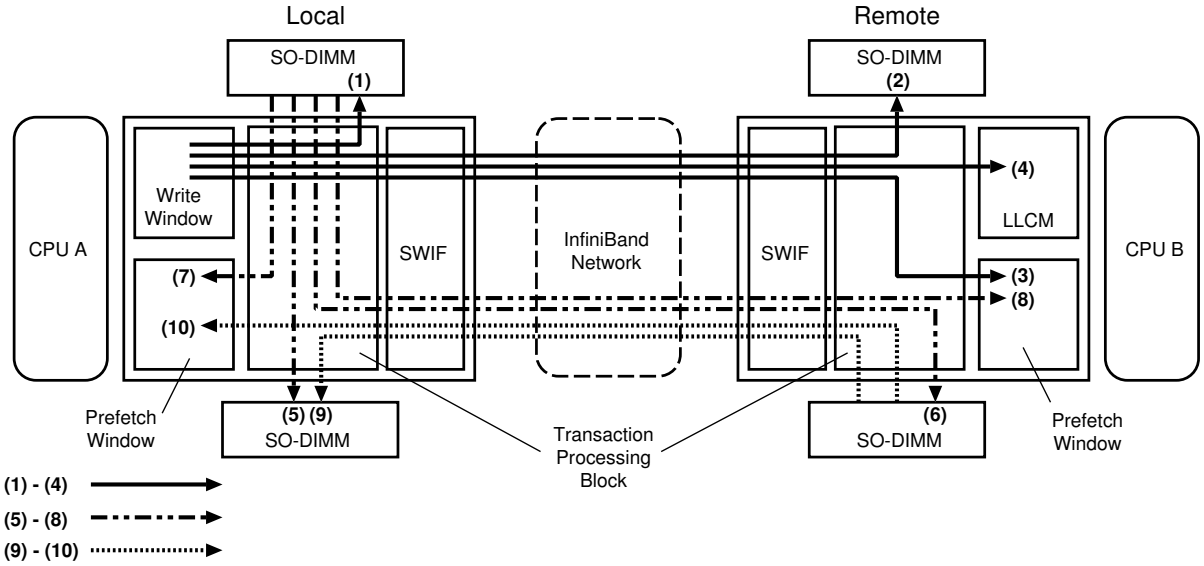


図 4.3　各バッファ間のデータ転送

Window のサイズが増加する. DIMMnet-2 試作基板のネットワークインタフェースコントローラは FPGA を用いて実装されるため, 内部の RAM 領域には限りがある. さらに, PC-1600 DDR-SDRAM バスの規格で動作させるためには 100MHz で動作させる必要があることから, FPGA 内部の RAM 領域のうち, ホスト側のインタフェースに近い位置のバッファを利用しなければならないという制約が存在する. これらの理由から, あまり多くのプロセスが同時に利用可能とするのは困難と考えられ, 同時に DIMMnet-2 を利用するのは 2 プロセスまでという制限を設けた.

各プロセスは DIMMnet-2 を搭載したノード群から構成されるシステム内において, LID, WID (Window ID), PGID (Process Group ID), 及び PID (Process ID) の 4 つの ID で区別される.

LID　LID は InfiniBand の規格で定められた, 各ネットワークインタフェースに与えられる識別子である. LID は InfiniBand の規格により 16bit と定められており, 各ノードに 1 枚の DIMMnet-2 を搭載する場合, 各ノードを一意に識別できる識別子としてみなすことができる.

DIMMnet-2 試作基板では LID の 16bit のうち, 12bit を使用し, 単一の InfiniBand のサブネット内で 3072 ノードまでのシステムをサポートする. 12bit あれば, 本来ならば 4096 ノードまで識別可能であるが, InfiniBand の規格でいくつかの LID が予約されており, ネットワークインタフェースに割り振ることができない LID が存在する.

WID　WID は各ノード内で動作するプロセスに与えられた識別子である. DIMMnet-2 試作基板では最大で 2 プロセスが DIMMnet-2 を使用するため, 1bit の識別子となっている. LID はネットワークインタフェースごとに 1 つ与えられるため, 各ノード内で動作するプロセスを識別することは LID だけではできない. そこで, WID によって各ノード内で動作するプロセスを識別することでネットワーク上のプロセスを一意に識別する.

PGID　PGID は各並列プロセス群に一意に与えられる識別子である. DIMMnet-2 試作基板では PGID は 0〜255 までの 256 個の値をとる. つまり, 同一のシステム内において, 256 個の並列プロ

セス群が同時にシステムを使用可能となっている.

DIMMnet-2 試作基板のサポートする最大ノード数が 3072 台であり, 各ノードで同時に DIMMnet-2 を使用可能なプロセスが 2 プロセスであることから, 単一のシステムで識別可能なプロセス数は 3072×2=6144 プロセスである. この場合, すべてのプロセスが単体で独立したプロセスグループを形成したとすると, 最大で 6144 個のプロセスグループが同時に存在しうる. そのため, これらを識別するには, PGID は 13bit 必要であるが, PC クラスタ上で各プロセスが完全に独立に動作する場合は稀であると考えられるため, 最大プロセスグループ数を 256 個, PGID を 8bit とした.

プロセス間の通信は, 同一の PGID を持つプロセス同士でのみ行うことが可能である. 並列プロセスを起動させる最初の時点で, 各プロセスに対して DIMMnet-2 上の資源を割り当てるのと同時に, 特権プロセスが PID と PGID を組にしてネットワークインタフェースコントローラ上のレジスタに設定する. このレジスタ上の PGID は特権プロセス以外のプロセスからは操作ができないようにホスト側で制御する.

パケットを構築する際, ネットワークインタフェースコントローラ側でコントローラ上のレジスタに設定された PGID をパケットヘッダに付加することで, 異なる PGID のプロセス宛のパケットを生成できないようにする. これにより, プロセスグループ間での通信が防止されるため, PGID の異なるプロセスグループに属するプロセスのメモリ領域が干渉されるのを防ぐことが可能となる.

PID PID は各プロセスグループ内のプロセスを識別するための識別子である. PID は LID, 及び WID と対応づけられており, 各プロセスは PID をもとに PID-LID/WID 対応テーブル (後述) を引くことで通信処理時に通信相手の LID と WID を得る. PID のビット幅はホスト側のテーブルの実装に依存する.

4.4.1 プロセス識別子の管理

本節では WID, LID, PGID の管理手法について述べる.

PID-LID/WID 対応テーブル PID-LID/WID 対応テーブルは各プロセスグループ内のプロセスとそのプロセスの LID, 及び WID を対応づけるテーブルである. 通信を行う際に, 送信先の LID と WID を得るために使用する. このテーブルは図 4.4 のような構造を持ち, ホストの主記憶上に確保される. テーブルのエントリ数はそのプロセスグループに属するプロセス数と同じになる.

DIMMnet-2 を利用するプロセスはこのようなテーブルを個々に保持する. このテーブルは並列プロセス起動時に特権プロセス同士が何らかの制御用ネットワーク(注 2)を用いて通信し合うことにより生成する. DIMMnet-2 を利用するプロセスは自分の属するプロセスグループのテーブルを, 並列プロセス起動時に特権プロセスのメモリ領域から自らのメモリ領域にコピーする.

WID-PGID 対応テーブル WID-PGID 対応テーブルは, WID と PGID を対応付けるテーブルであり, 同一ノード上で DIMMnet-2 を利用するプロセスの WID と PGID を管理する. このテーブルは, 特権プロセスにより DIMMnet-2 のネットワークインタフェースコントローラ内部に保持される. このテーブルは図 4.5 のような構造となる.

ユーザプロセスから通信要求が発行された際に, 要求が書き込まれたアドレスに対応する WID でこのテーブルを引き, PGID を得る. 得られた PGID をパケットヘッダに付加することでメモリ

(注 2)NFS によるファイル共有のための Ethernet など

PGID=X

PID	LID	WID
0	3	0
1	8	1
…	…	…
N	M	0

図 4.4 PID-LID/WID table

WID	PGID
0	3
1	8

図 4.5 WID-PGID table

表 4.3 ホストインタフェース部のメモリ領域のアドレス割り当て

Module	Address
Write Window	0x00000 〜
Prefetch Window	0x10000 〜
LLCM	0x20000 〜
LH Buffer	0x30000 〜
User Register	0x40000 〜
System Register	0x50000 〜

保護を実現する.

これらのテーブルを使用して, パケットがネットワークに送出されるまでの流れを図 4.6 に示す.

4.5 Core Logic ホストインタフェース部のメモリ領域

DIMMnet-2 では, ホストインタフェース部のメモリ領域をユーザプロセスのプロセス空間にマップし, ユーザプロセスからはユーザレベルで主記憶にアクセスするのと同様の操作でこれらのメモリ領域にアクセスする. ホストインタフェース部の Register は特権プロセスのみがアクセスする System Register とユーザプロセスが使用する User Register に分けられ, ユーザプロセスは System Register 以外のメモリ領域をすべて利用可能である.

ホストインタフェース部のメモリ領域の, 物理アドレスの割り当てを表 4.3 に示す. また, 表 4.3 に対応する図を図 4.7 に示す. なお, 表 4.3 に示されるアドレスは, DIMMnet-2 用に確保した物理アドレス空間のベースアドレスからのオフセット値である.

MTRR による設定はメモリのページ単位で行うため, DIMMnet-2 のように各領域で設定が異なる場合, 各領域を別のページに割り当てることが必要となる. x86 系 CPU の場合, メモリのページ管理は 4KByte 単位で行われるため, 各領域間のアドレスを 4KByte 以上離している.

4.5.1 Write Window

Write Window は 512Byte の領域を 1 個の Window とし, 各プロセスに 4 個 (2KByte) 割り当てる. 図 4.8 に Write Window の物理アドレス空間への割り当てを示す.

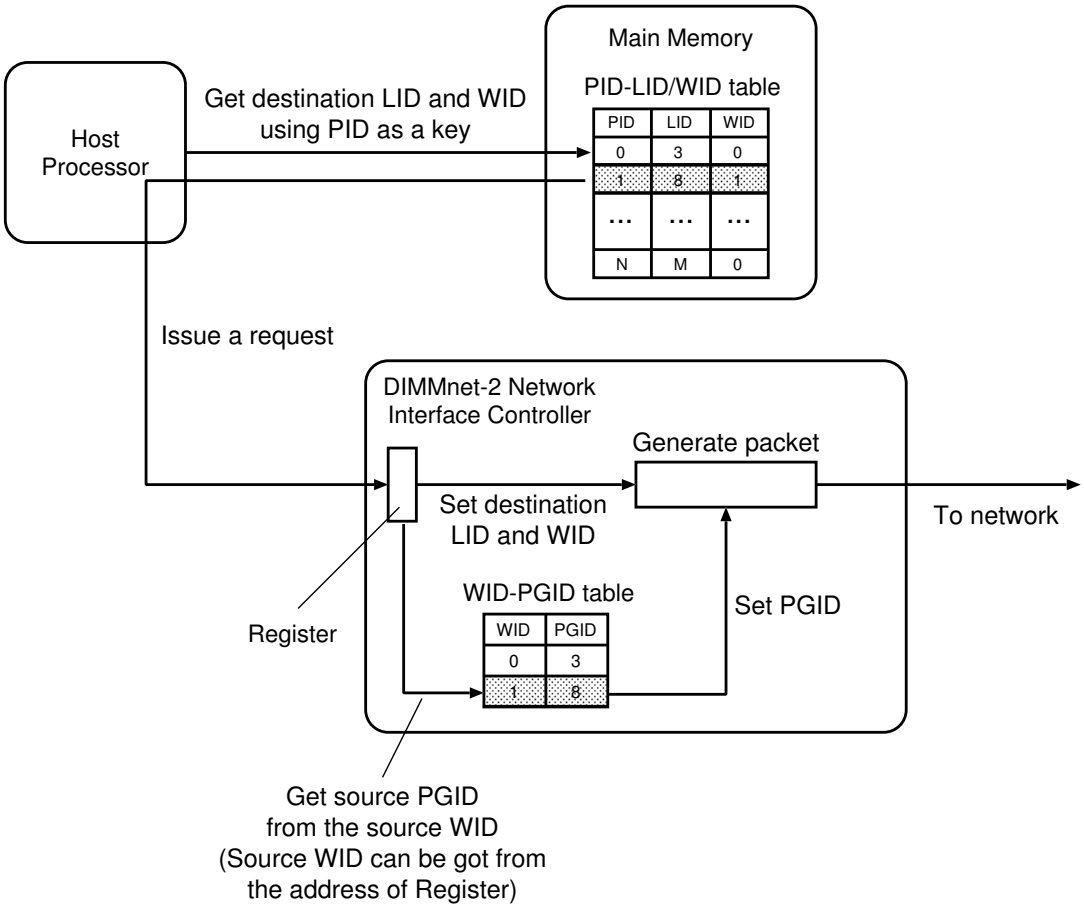


図 4.6　パケットの送出処理時の各 ID の取得

4.5.2 Prefetch Window

Prefetch Window も Write Window と同様に 512Byte の領域を 1 個の Window とし, 各プロセスに 4 個 (2KByte) 割り当てる. 図 4.9 に Prefetch Window のアドレス空間への割り当てを示す.

4.5.3 LLCM

LLCM は各プロセスに 32KByte ずつ割り当てる. LLCM は Write Window や Prefetch Window に比べ, 容量が多く確保されている. これは, パケット受信ステータスの受信バッファや, IPUSH のアドレス管理テーブルなど, 様々なデータを格納するためである. 図 4.10 に LLCM のアドレス空間への割り当てを示す.

4.5.4 LH Buffer

LH Buffer は各プロセスに 4KByte ずつ割り当てる. 図 4.11 に LLCM のアドレス空間への割り当てを示す.

LH Buffer はリングバッファとして用いられ, 要求処理部より LH Buffer に書き込みが行われる

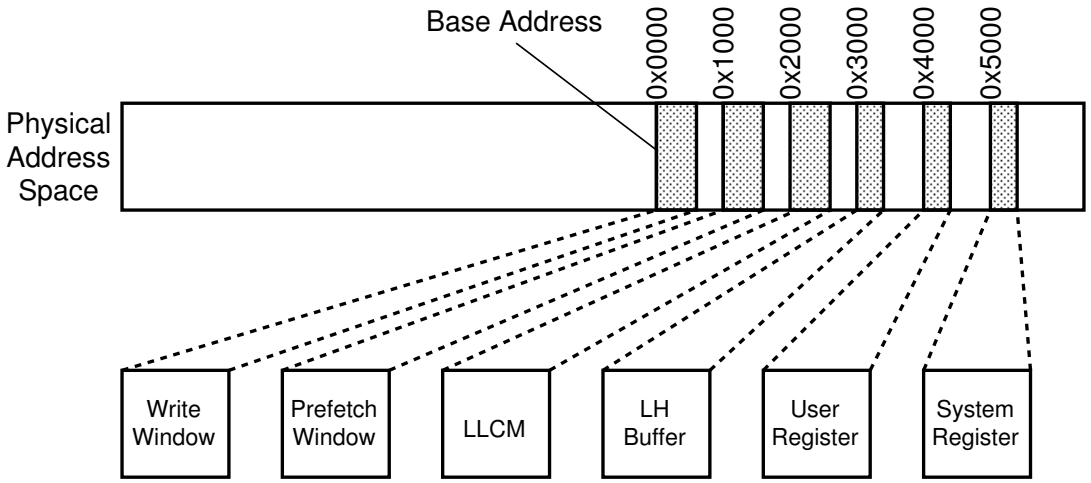


図 4.7　ホストインタフェース部のメモリ領域のアドレスマップ

とリングバッファの Tail ポインタが更新される. ホストプロセッサは LH Buffer から値を読み出すと, User Register を介して Head ポインタを更新する. LH Buffer は 64Byte を 1 エントリとし, エントリ単位で使用する. エントリの総数は 64 個とし, 63 エントリを使用した段階で LH Buffer が full であると検出されるように設計した. そのため, 最大で 63 個のデータを格納することができる.

4.5.5 User Register

User Register 内に設けられているレジスタの用途は大きく分けて, 次のようになる.

- ホストプロセッサからの要求発行
- ネットワークインタフェースコントローラの状態の通知
- パケット受信ステータス用のバッファなど, LLCM を使用して構築されるバッファのアドレス管理

図 4.12 に User Register のアドレス空間への割り当てを示す.

4.5.5.1 要求発行レジスタ

図 4.12 に示されるレジスタのうち, Command0 Lower, Command0 Upper, Command Ex, Command1 Lower, Command1 Upper はホストプロセッサからの要求発行用レジスタである. 図 4.13 にホストから発行する要求のフィールドフォーマットを示す.

DIMMnet-2 で用いられる要求は通常, 128bit から構成される. この 128bit の上位 64bit を Upper 側のレジスタに書き込み, 下位 64bit を Lower 側のレジスタに書き込む. Lower 側のレジスタに値を書き込むと, それに対応した Upper 側のレジスタ(注 3)に書き込まれた値と共に要求処理部に要求が発行される. そのため, ホストから要求を発行する際には, Upper 側のレジスタに書き込んだ後に, Lower 側のレジスタに書き込む必要がある. このように, 要求を書き込むレジスタと発行する

(注 3)Command0 Lower ならば Command0 Upper, Command1 Lower ならば Command1 Upper

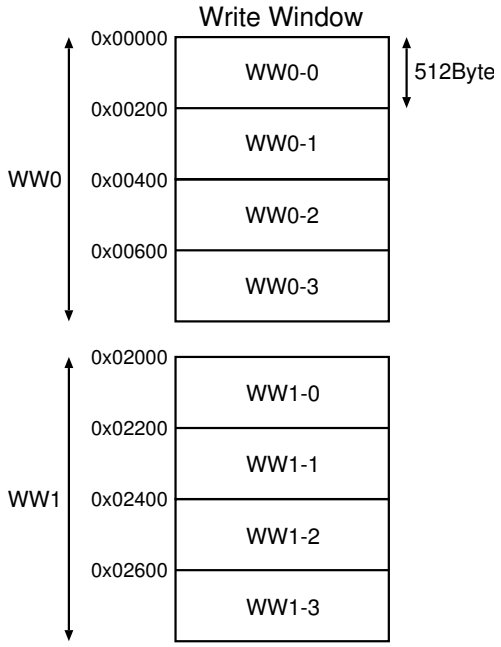


図 4.8 Write Window のアドレスマップ

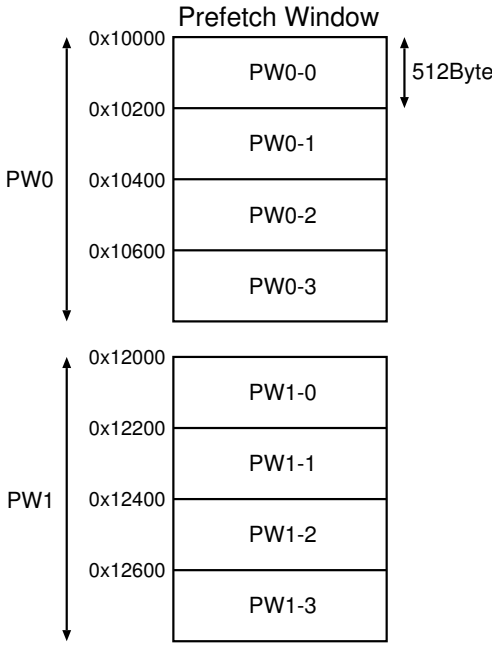


図 4.9 Prefetch Window のアドレスマップ

(キックする) レジスタを同一のレジスタにすることで, キックするための専用の領域を用意する必要がなくなり, また, 要求の書き込みとキックを同時にできるため, 要求発行時のレイテンシの削減が可能となる. 通信処理要求を発行する場合, Command0 のレジスタを用いて要求を発行するとデータの受信先が SO-DIMM になり, Command1 を用いると Prefetch Window になる.

Command Ex は Command Ex Flag (後述) を有効にすることで使うことができるレジスタである. リモートホストとの SO-DIMM 間でのデータ転送の際に, ローカル側とリモート側でそれぞれ不連続アクセスを行う際に用いる. Command Ex を用いない場合, ローカル側では連続領域から読み出したデータの転送しかできない. Command Ex はローカル側の SO-DIMM へのアクセスパラメータを指定するために用いる. Command Ex の利用により, MPI における派生データ型通信の Pack/Unpack 処理を高速化することができる [18][107][108].

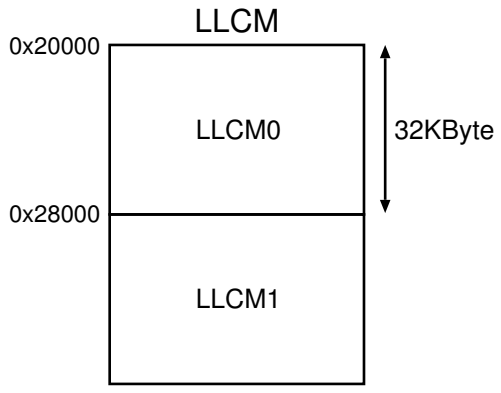


図 4.10 LLCM のアドレスマップ

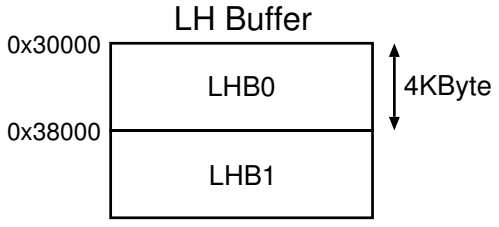


図 4.11 LH Buffer のアドレスマップ

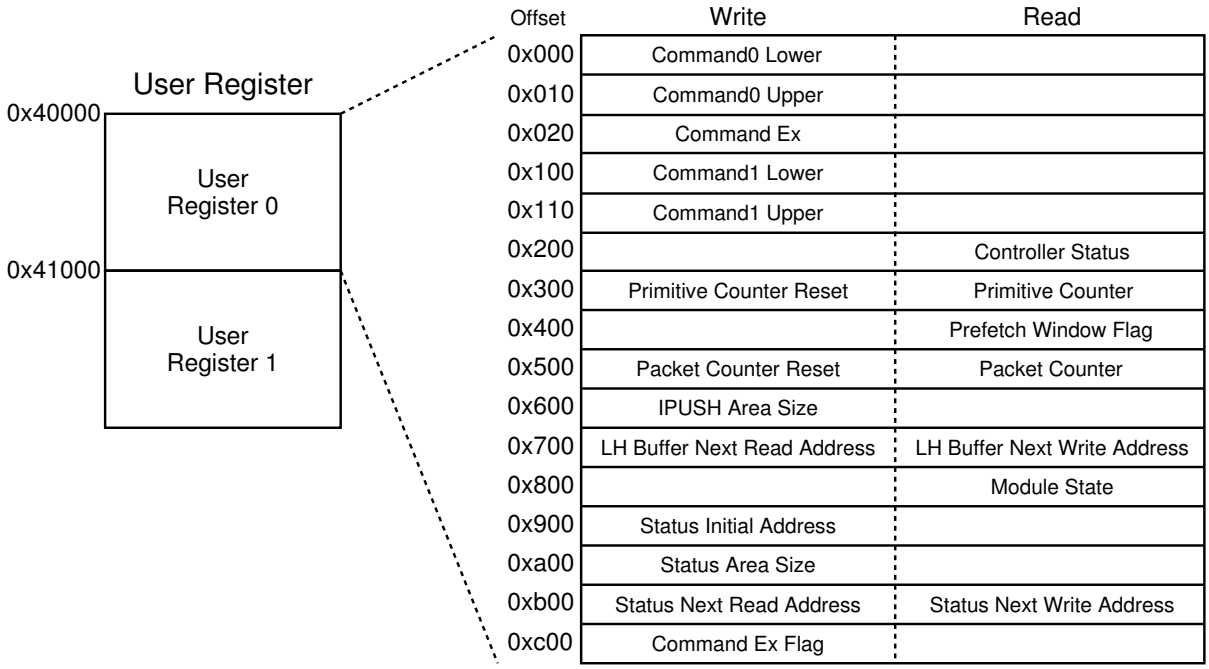


図 4.12　User Register のアドレスマップ

図 4.13 に示される各パラメータの意味を表 4.4 に示す.

不連続アクセス時にはネットワークインタフェース上の SO-DIMM に対して, 特定サイズのデータを複数回に渡って読み書きする. そのため, ホストプロセッサから DTYPE と Iteration で, 要素当たりのデータサイズと要素数を指定する. 表 4.5 に DTYPE と要素当たりのデータサイズの関係を示す(注 4).

CLID (Compressed LID) とは, InfiniBand における LID を 16bit から 12bit に圧縮したものである. InfiniBand において, ノードは表 4.6 のように 16bit の LID で識別される. しかし, DIMMnet-2 試作基板でサポートするシステムの最大ノード数は 3072 ノードであるため, 0x0001〜0xBFFF までの 49151 ノードを識別できる必要はなく, 未使用の LID が多数存在することになる. そこで, 16bit の

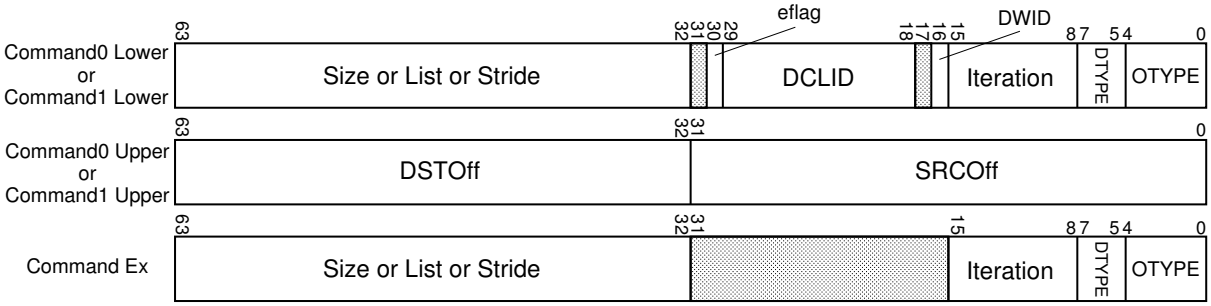


図 4.13　要求のフィールドフォーマット

(注 4)表 4.5 では 1Byte から DTYPE が定義されているが, SO-DIMM に対しては 4Byte 以上のデータサイズを単位とするアクセスを行うように設計している

表 4.4　要求発行時のパラメータ

パラメータ名	ビット幅	意味
OTYPE (Operation TYPE)	5	要求の種別 (4.6 節)
DTYPE (Data TYPE)	3	不連続アクセス時の要素当たりのデータサイズ (表 4.5)
Iteration	8	不連続アクセス時のデータの要素数
DWID (Destination WID)	1	リモートノードとの通信の際に指定するリモートプロセスの WID
DCLID (Destination CLID)	12	リモートノードとの通信の際に指定するリモートプロセスの CLID (後述)
eflag	1	パケット受信ステータスを受信側で生成するかどうかを指定するフラグ (0：生成しない, 1：生成する)
Size or List or Stride	32	OTYPE によって以下のいずれかの値を示す
Size		連続アクセス (4.6 節) 時の転送サイズ
List		不連続アクセス (リストアクセス (4.6 節)) 時に使用する, リストが格納されている SO-DIMM 領域の先頭アドレス
Stride		不連続アクセス (ストライドアクセス (4.6 節)) 時に使用するストライド間隔
SRCOff (SouRCe Offset)	32	SO-DIMM または Write Window のベースアドレスからのデータ読み出し元オフセット
DSTOff (DeSTination Offset)	32	SO-DIMM または Prefetch Window のベースアドレスからのデータ書き込み先オフセット

LID の 9〜12bit 目を除いた 12bit の CLID にし, LID のビット幅を削減する. このようにすることで, ホストから要求を発行する際に必要なビット数を削減することができる.

この CLID をネットワークインタフェースコントローラ側で 16bit に展開し, パケットヘッダ (4.7 節) に使用する. CLID に対応しない LID はネットワーク内で割り当てられないように Subnet Manager によって制御される.

要求発行時に User Register に書き込むパラメータ以外にも, Window Controller がパケットヘッダを生成する際にローカル側の WID (SWID：Source WID) や PGID が必要となる. しかし, SWID は書き込まれたレジスタのアドレスから得られ, PGID は SWID から WID-PGID 対応テーブル (4.4.1 節) を引くことで得ることができる. そのため, 要求発行時に明示的にホストプロセッサから指定する必要はない.

OTYPE に BOTF を指定した場合のみ要求のフィールドフォーマットが異なり, 図 4.14 のようになる. BOTF は Write Window に書き込まれたパケットイメージ(注 5)を, そのままパケットとしてネットワークに送出するプリミティブである. BOTF では, パケットヘッダを構築するのに必要な情報の大部分を Write Window にパケットイメージとして書き込んでいるため, User Register に書き込む情報は, ほかのプリミティブよりも少なくなる. 図 4.14 に示される各フィールドの詳細を表

(注 5)パケットヘッダとデータ本体から構成

表 4.5　DTYPE とデータ 1 要素のサイズの関係

DTYPE	バイト数
3'b000	1
3'b001	2
3'b010	4
3'b011	8
3'b100	16
3'b101	32
3'b110	64
3'b111	128

表 4.6　LID と CLID の対応

LID	CLID	用途
16'h0000	12'h000	予約済み (使用不可)
16'h0001〜16'hBFFF	12'h001 (node#1)〜12'hBFF (node#3072)	ユニキャスト
16'hC000〜16'hFFFE	12'hC00〜12'hFFE	マルチキャスト
16'hFFFF	12'hFFF	初期化時に使用

4.7 に示す. BOTF の場合, 要求処理部は必ず Window の先頭から読み出しを開始するため, Window の番号を指定するだけで実行可能になっている.

4.5.5.2　要求発行レジスタ以外の User Register

要求発行レジスタ以外の User Register はネットワークインタフェースコントローラ内部の状態を示すステータスレジスタとバッファのアドレス管理用レジスタである. これらのレジスタの用途を表 4.8 に示す. 表 4.8 の R/W の項目はホストプロセッサ側から可能な操作を示す. また, これらのレジスタのビットフィールドについては付録 A に示す.

Controller Status　Controller Status はネットワークインタフェースコントローラ内部の様々な状態をホストプロセッサから検出可能にするためのレジスタである. メモリバスには割り込み線が存在しないため, DIMMnet-2 においてはプリミティブの完了や実行時のエラーの検出は, ホストプロ

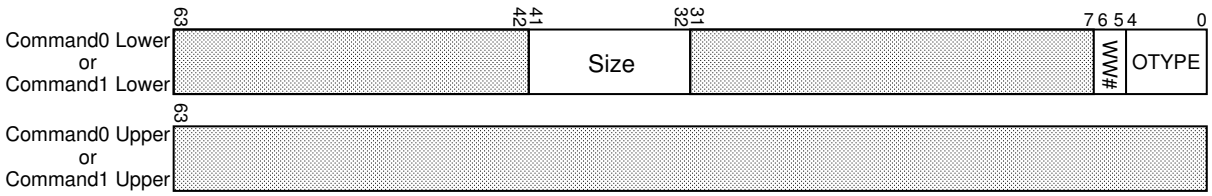


図 4.14　BOTF 時の要求発行のフィールドフォーマット

表 4.7　BOTF 時の要求発行パラメータ

パラメータ名	ビット幅	意味
OTYPE	5	要求の種別 (BOTF 要求であることを指定)
WW# (Write Window #)	2	どの Window のデータを転送するかを指定
SIZE	10	WW#で指定した Window から転送するデータサイズ

セッサ側から能動的に Controller Status を読み出す必要がある. このような手法をとると, Controller Status のポーリングが必要になり, ホストプロセッサのオーバーヘッドが増加するが, コンテキストスイッチが発生しないため, 割り込みの場合に比べてプリミティブを低いレイテンシで処理することができる.

Primitive Counter, Primitive Counter Reset　Primitive Counter は各プリミティブの実行回数を系列 (4.6 節) ごとに計測するカウンタであり, プロセスごとに独立に実行回数を計測する. 従って, WID=0 のプロセスが実行したプリミティブは WID=0 のカウンタに反映される. プリミティブの実行が完了すると, 対応するカウンタがインクリメントされる. この領域に対して書き込み操作を行うと, 書き込んだデータの内容に関係なくカウンタがリセットされる.

Prefetch Window Flag　Prefetch Window Flag は Prefetch Window を 128Byte の単位 (ライン) で区切り, SO-DIMM からのデータ読み出し時に, そのラインに対する読み出しが完了したかどうかを示すフラグである. 従って, Window 1 個当たり 4bit のフラグとなる. 128Byte ごとに 1bit のフラグを設けることで, SO-DIMM から Prefetch Window へのデータの書き込みとホストプロセッサの Prefetch Window からのデータの読み出しがオーバラップ可能となる. このフラグの操作を次に示す.

1. ホストから SO-DIMM のデータ読み出し要求が発行されると, 読み出し先に指定された Window のフラグがリセット (4'b0000) される
2. 要求処理部が Window への書き込み開始位置と読み出しサイズから, 書き込みの行われないラインを検出し, それに対応するフラグを 1 にする
3. 要求処理部が読み出しを完了させたラインから順次フラグを 1 にする
4. 読み出し先となった Window のフラグが 4'b1111 になると読み出し完了となる

Packet Counter, Packet Counter Reset　Packet Counter は受信したパケット数を計測するカウンタであり, Primitive Counter と同様にプロセスごとに独立にパケット数をカウントする. この領域に対して書き込み操作を行うと, 書き込んだデータの内容に関係なくカウンタがリセットされる.

Module State　Module State は Core Logic 内部の各モジュールのステートをホストプロセッサから検知可能にするためのレジスタである. Window Controller や Receive Controller は, それぞれが独立したステートマシンとして実装されており, ネットワークインタフェースコントローラがエラーにより停止した場合に, ホストプロセッサ側からどの状態で停止したのかを検出することが可能となる.

表 4.8　要求発行レジスタ以外の User Register の用途

オフセット	レジスタ名	R/W	用途
0x200	Controller Status	Read	ネットワークインタフェースコントローラ内部の状態を示す
0x300	Primitive Counter	Read	実行したプリミティブの回数を計測するカウンタ
	Primitive Counter Reset	Write	Primitive Counter のリセット
0x400	Prefetch Window Flag	Read	Prefetch Window へのデータ読み出し完了フラグ
0x500	Packet Counter	Read	受信したパケット数を計測するカウンタ
	Packet Counter Reset	Write	Packet Counter のリセット
0x600	IPUSH Area Size	Write	IPUSH で使用する SO-DIMM 領域のサイズ (7 章)
0x700	LH Buffer Next Write Address	Read	LH Buffer の Tail ポインタ (7 章)
	LH Buffer Next Read Address	Write	LH Buffer の Head ポインタ (7 章)
0x800	Module State	Read	Core Logic 内部の各モジュールのステート
0x900	Status Initial Address	Write	パケット受信ステータス用バッファの先頭アドレス
0xa00	Status Area Size	Write	パケット受信ステータス用バッファのサイズ
0xb00	Status Next Write Address	Read	パケット受信ステータス用バッファの Tail ポインタ
	Status Next Read Address	Write	パケット受信ステータス用バッファの Head ポインタ
0xc00	Command Ex Flag	Write	Command Ex 有効化フラグ

Status Initial Address, Status Area Size, Status Next Read Address, Status Next Write Address
Status Initial Address〜Status Next Write Address までのレジスタはパケット受信ステータスの受信領域を管理するためのレジスタである. DIMMnet-2 では, パケットを受信した際に “どのプロセスから”,“どの領域に”,“何バイトのデータを” 受信したのかを示すパケット受信ステータスを LLCM に書き込む. 受信ステータスは 1 つ当たり 128bit であり, 図 4.13 の Command Ex を除いたフォーマットで, 受信処理で行った操作の情報が書き込まれる. ただし, 図 4.13 に示される情報のうちのいくつかは, 表 4.9 のように書き換えられる.

LLCM はパケット受信ステータス以外に IPUSH 機構のアドレス管理テーブルのために使用される. そのため, LLCM の全領域を受信ステータス用に使用することはできず, パケット受信ステータスを格納する領域をユーザプロセスから指定する必要がある. パケット受信ステータスを格納する領域は Status Initial Address で指定した領域から, Status Area Size で指定したサイズのリングバッファとして確保され, Status Next Write Address と Status Next Read Address をリングバッファの Tail ポインタと Head ポインタとして用いて管理する.

表 4.9　パケット受信ステータスで書き換えられる情報

from	to
DWID	送信元の WID
DCLID	送信元の CLID
SRCOff	データが書き込まれた最初のアドレス
DSTOff	特に意味を持たない情報

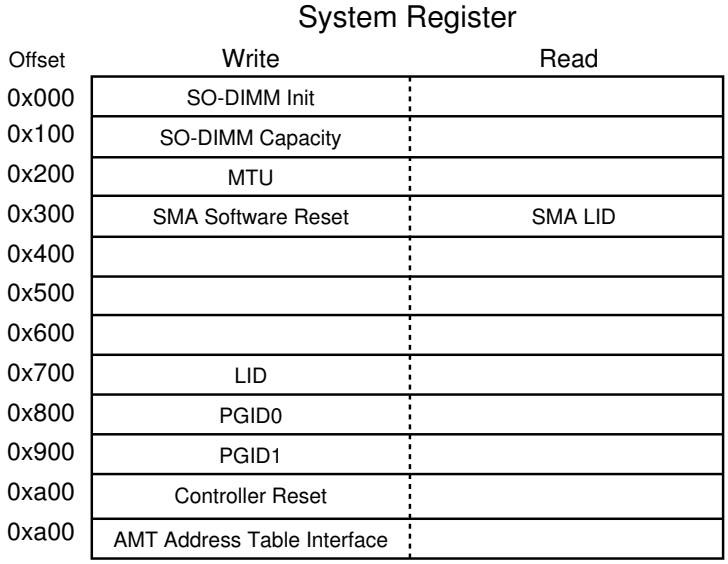


図 4.15　System Register

Command Ex Flag　Command Ex Flag は Command Ex を利用可能にするためのレジスタである. このレジスタに 1 をセットすると, Command Ex が有効になり, プリミティブが 192bit で構成されるようになる.

4.5.6 System Register

System Register は特権プロセスのみ読み書き可能な, DIMMnet-2 の動作設定を行うためのレジスタである. System Register の詳細を図 4.15 に示す. また, これらのレジスタの用途を表 4.12 に示す. 表 4.12 の R/W の項目は表 4.8 と同様である. これらのレジスタのビットフィールドについては付録 B に示す.

SO-DIMM Init　SO-DIMM Init は DIMMnet-2 に搭載されている SO-DIMM を Write Window や Prefetch Window を介してアクセス可能にするためのレジスタである. この領域に 1 を書き込むと SO-DIMM が有効化される.

SO-DIMM Capacity　SO-DIMM Capacity は SO-DIMM 1 枚当たりの容量を Core Logic に通知するためのレジスタである. Core Logic は搭載する SO-DIMM の容量に応じたアクセス制御を行う.

表 4.10 SO-DIMM Capacity

設定値	SO-DIMM Capacity[MByte]
6'b000001	256 (default 値)
6'b000010	512
6'b000100	1024
6'b001000	2048
6'b010000	4096
6'b100000	8192
上記以外	default 値

表 4.11 MTU

設定値	MTU[Byte]
5'b00001	256
5'b00010	512
5'b10100	1024
5'b01000	2048 (default 値)
5'b10000	4096
上記以外	default 値

DIMMnet-2 では 1 枚当たりの容量が 8GByte のメモリモジュールまでならば搭載可能としている. これは, "マザーボードのメモリスロット当たりの搭載可能容量を上回るメモリ領域を DIMMnet-2 で提供する" ことを目的としているためである.

表 4.10 に SO-DIMM Capacity に書き込む値と設定される SO-DIMM の容量の関係を示す. また, 図 4.16 に SO-DIMM Capacity に 6'b000010 (512MByte) を設定した場合の, 各プロセスに割り当てられる SO-DIMM 領域を示す. 点線の矢印はアドレスの増加方向を示す.

DIMMnet-2 では SO-DIMM のアドレスを 8Byte 単位で交互に振り, 領域を図 4.16 のように各プロセスに均等に割り当てる. 各プロセスが 2 枚の SO-DIMM に同時にアクセス可能な構造にすることで Dual Channel 動作でのアクセスを実現し, SO-DIMM に対するスループットを向上させる. DDR-SDRAM へのアクセスでは, 1 回のアクセスでバースト長 ×64 [bit] のデータを読み書きする. 連続領域へのアクセスの場合, そのバースト長に応じた高いスループットが得られるが, 読み出しアドレスが連続していない不連続アクセスの場合でも, 1 回のアクセスにつきバースト長に応じたデータが出力されてしまい, 必要のないデータを読み出してしまうことになる. 従って, 不連続アクセス時には高い実効スループットを得ることができない. そこで, DIMMnet-2 では SO-DIMM のバースト長を最小 (=2) にし, 各 SO-DIMM に交互にアドレスを割り振ることで, 不連続アクセス時にも高いスループットが得られるようにしている.

MTU MTU は DIMMnet-2 が使用するネットワークスイッチの MTU 値をセットするためのレジスタである. MTU 値は InfiniBand パケットのデータ部のサイズであり, この値までのサイズのデータに InfiniBand のヘッダとトレイラが付加され, ネットワークにパケットが送出される. MTU に書き込む値と設定される MTU 値の関係を表 4.11 に示す. これらの MTU 値は, 本研究で使用している Voltaire 社の InfiniBand スイッチルータ ISR6000[109] で指定可能な MTU 値を元にしている.

SMA LID, SMA Software Reset SMA LID は SMA から取得した LID をホストプロセッサが読み出すためのレジスタである. LID だけでなく, DIMMnet-2 が通信可能な状態にあるかどうかを検出するための情報も提供する. SMA Software Reset は SMA をホストプロセッサからリセットするのに使用する. このレジスタに対して書き込みを行うと, 書き込んだデータの内容に関係なく SMA がリセットされる.

LID LID は自ノードの LID を Core Logic に通知するためのレジスタである.

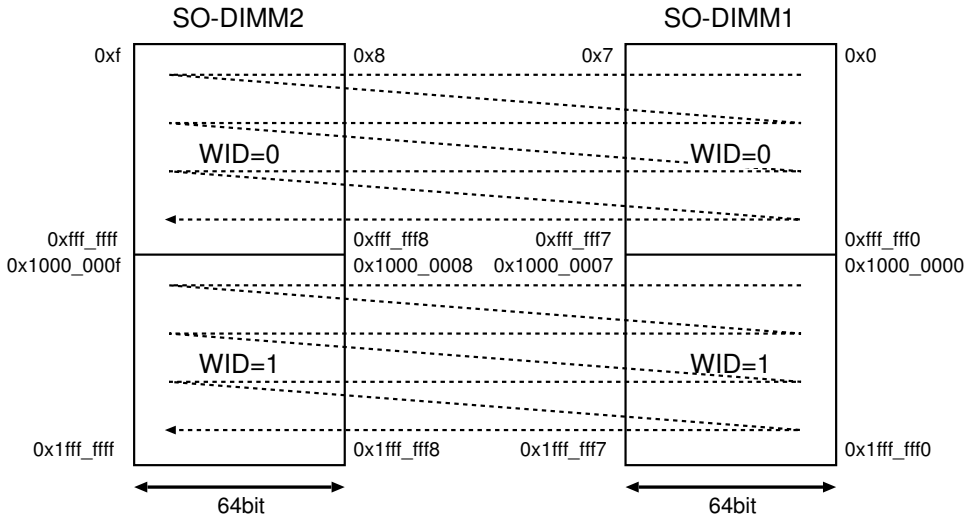


図 4.16 SO-DIMM Address (512MByte/module)

PGID0, PGID1 PGID0, PGID1 は DIMMnet-2 を利用しているプロセスが属するプロセスグループの PGID を Core Logic に通知するためのレジスタである. ホストから PGID0 に書き込むと, Core Logic が WID=0 のプロセスの PGID として設定する. また, PGID1 に書き込みを行うと, WID=1 のプロセスの PGID が設定される.

Controller Reset Controller Reset はネットワークインタフェースコントローラをリセットするためのレジスタである. Controller Reset にアクセスすると, ネットワークインタフェースコントローラにリセットがかかる.

AMT Address Table Interface AMT Address Table Interface は AMT Address Table に書き込みを行うためのレジスタである. 詳細は 7 章で述べる.

4.6 プリミティブ

本節では DIMMnet-2 で定義されているプリミティブについて述べる. 4.3 節で述べたデータ転送処理は本節で示すプリミティブ単体, またはプリミティブの組み合わせで実現される. プリミティブはデータを読み出す VL (Vector Load) 系列とデータを書き込む VS (Vector Store) 系列に分けられる. また, それぞれについてローカル側, リモート側に作用するプリミティブが存在する. プリミティブの総数は 17 個であり, それぞれに 5bit のコードが割り当てられる. DIMMnet-2 で定義されているプリミティブを表 4.13 に示す.

4.6.1 NOP (No OPeration)

その名の通り, 何も処理を行わないプリミティブである. ユーザプロセスから明示的に発行されるものではなく, コントローラのリセット時などにコントローラが予期せぬ挙動を示すのを防ぐために, コントローラの内部処理で用いられるプリミティブである.

表 4.12　System Register の用途

オフセット	レジスタ名	R/W	用途
0x000	SO-DIMM Init	Write	SO-DIMM を利用可能にするための領域
0x100	SO-DIMM Capacity	Write	SO-DIMM の容量の設定
0x200	MTU	Write	MTU 値の設定
0x300	SMA LID	Read	Subnet Manager Agent から得た LID 値
	SMA Software Reset	Write	Subnet Manager Agent のリセット
0x700	LID	Write	LID の設定
0x800	PGID0	Write	WID=0 のプロセスの PGID の設定
0x900	PGID1	Write	WID=1 のプロセスの PGID の設定
0xa00	Controller Reset	Write	ネットワークインタフェースコントローラのリセット
0xb00	AMT Address Table Interface	Write	AMT Address Table (7 章) へ値を設定するための領域

4.6.2　BOTF (Block On-The-Fly)

Write Window のデータをそのままパケットとしてネットワークに送出するプリミティブである.

ホストからは BOTF 要求を User Register に発行する前に, Write Window にパケットヘッダ (4.7 節) とデータ本体から構成されるパケットイメージを書き込んでおく必要がある. この際に, パケットヘッダに含まれるプリミティブの種別を示すフィールドを任意のプリミティブに書き換えることで, RVL や RVS など, 様々な処理をリモート側で行わせることが可能である.

パケットヘッダには PGID の情報も含まれるが, PGID フィールドにホストから書き込まれた値を使用すると, ユーザプロセスが不正な PGID を指定することでほかのプロセスグループに属するプロセスのメモリ領域に干渉することが可能となってしまう. そこで, メモリ保護のために Write Window のデータを要求処理部が読み出した際に, パケットヘッダの PGID のフィールドのみ, 事前に特権プロセスによって設定されている PGID の値に書き換える.

BOTF では要求発行レジスタで指定されたサイズだけ Write Window のデータを転送する. そのため, パケットイメージのパケットヘッダ部に指定されているパケットサイズの値が BOTF 要求発行時のサイズと異なっていると, 受信側でエラーを起こす原因となってしまう. そこで, パケットヘッダ上のパケットサイズを示すフィールドも要求処理部で書き換える. ただし, BOTF の転送サイズは Window 1 個当たりのサイズに制限され, 最大 512Byte となるため, 512Byte より大きな BOTF 要求だった場合には, 512Byte が指定されたものとして処理される.

4.6.3　VL 系列 (Vector Load Family)

VL 系列のプリミティブには VL, VLS (Vector Load Stride), VLI (Vector Load Index) が定義されている.

VL (連続ロード)　SRCOff で指定された SO-DIMM の位置から, Size で指定された大きさのデータを DSTOff で指定された Prefetch Window の対応する位置へ格納するプリミティブである. 図 4.17

表 4.13　プリミティブ一覧

プリミティブ		動作	ビット
NOP		何もしない	5'b00000
BOTF		Write Window からの任意のパケットの転送	5'b00001
VL 系列	VL	連続ロード	5'b00100
(Local)	VLS	ストライドロード	5'b00101
	VLI	リストロード	5'b00110
VS 系列	VS	連続ストア	5'b01000
(Local)	VSS	ストライドストア	5'b01001
	VSI	リストストア	5'b01010
VL 系列	RVL	リモート連続ロード	5'b10000
(Remote)	RVLS	リモートストライドロード	5'b10001
	RVLI	リモートリストロード	5'b10010
VS 系列	RVS	リモート連続ストア	5'b10100
(Remote)	RVSS	リモートストライドストア	5'b10101
	RVSI	リモートリストストア	5'b10110
IPUSH 系列	IPUSH without LHS	リモート間接連続ストア	5'b11000
(Remote)	IPUSH with LHSv1	リモート間接連続ストア (LHSv1 使用)	5'b11001
	IPUSH with LHSv2	リモート間接連続ストア (LHSv2 使用)	5'b11010

に連続ロードの動作を示す.

Prefetch Window の Window 1 個当たりの容量は 512Byte であるため, DSTOff+(転送サイズ) が Window の末尾を超える場合は, Prefetch Window に書き込むことのできるサイズまで転送を行う.

VLS (ストライドロード)　SRCOff で指定された SO-DIMM の位置から, Stride で指定された間隔で DTYPE で指定されたサイズのデータを Iteration 回読み出し, DSTOff で指定された Prefetch Window の対応する位置に書き込むプリミティブである. 図 4.18 にストライドロードの動作を示す.

総転送サイズは DTYPE×Iteration [Byte] となるが, DSTOff+(総転送サイズ) が対象の Window の領域を超えた場合は, Prefetch Window に書き込むことのできるサイズまで転送を行う.

VLI (リストロード)　List で指定される SO-DIMM の領域に格納された Iteration 個のオフセット値 (index 配列：1 要素当たり 32bit) に SRCOff を足した位置のデータ (サイズ：DTYPE) を DSTOff で示された Prefetch Window の対応する位置に書き込むプリミティブである. 図 4.19 にリストロードの動作を示す.

総転送サイズは DTYPE×Iteration [Byte] となる. DSTOff+(総転送サイズ) が対象の Window の領域を超えた場合は, Prefetch Window に書き込むことのできるサイズまで転送を行う.

4.6.4　VS 系列 (Vector Store Faimily)

VS 系列のプリミティブには VS, VSS (Vector Store Stride), VSI (Vector Store Index) が定義されている.

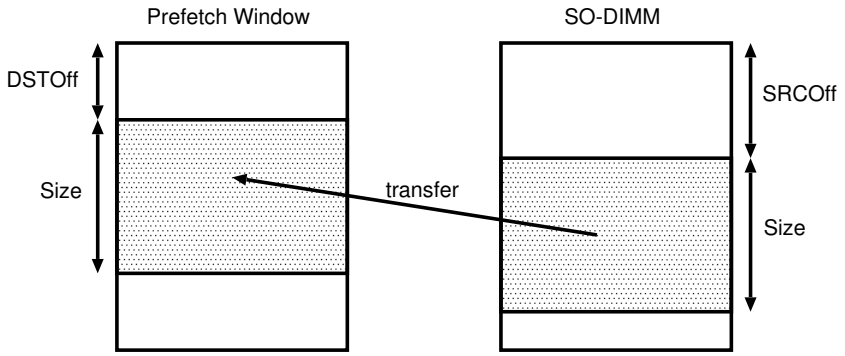


図 4.17　連続ロード

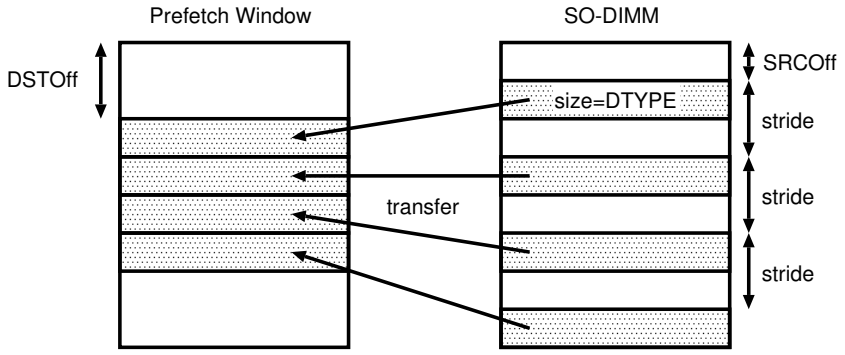


図 4.18　ストライドロード (Iteration=4)

VS (連続ストア)　SRCOff で指定された Write Window の対応する位置から Size で指定された大きさのデータを読み出し, DSTOff で指定された SO-DIMM の位置へ格納するプリミティブである. 図 4.20 に連続ストアの動作を示す.

Write Window の Window 1 個当たりの容量は 512Byte であるため, SRCOff+(転送サイズ) が Window の領域を超えた場合は, コントローラ側でこの値を超えないような値に Size 値を書き換えて処理を行う. また, DSTOff+(転送サイズ) が自 SO-DIMM 領域を超える場合は, 自 SO-DIMM 領域に書き込み可能なサイズまでデータを書き込む. これらの場合は, SO-DIMM への書き込み処理のエラーを示すフラグが変化する (付録 A).

VSS (ストライドストア)　SRCOff で指定された Write Window の対応する位置から Iteration 個の DTYPE サイズのデータを読み出し, DSTOff で指定された SO-DIMM の位置から, Stride で指定された間隔で格納するプリミティブである. 図 4.21 にストライドストアの動作を示す.

総転送サイズは DTYPE×Iteration [Byte] となる. SRCOff+(総転送サイズ) が対象の Window の領域を超えた場合は, コントローラ側で超えないような値に Iteration 値を書き換えて処理を行う. また, DSTOff+(総転送サイズ) が自 SO-DIMM 領域を超える場合は, 自 SO-DIMM 領域に書き込み可能なサイズまでデータを書き込む. これらの場合は, SO-DIMM への書き込み処理のエラーを示すフラグが変化する (付録 A).

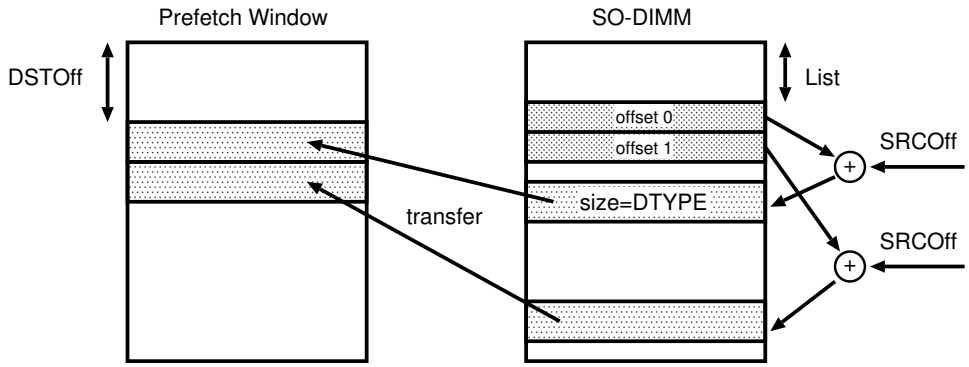


図 4.19　リストロード (Iteration=2)

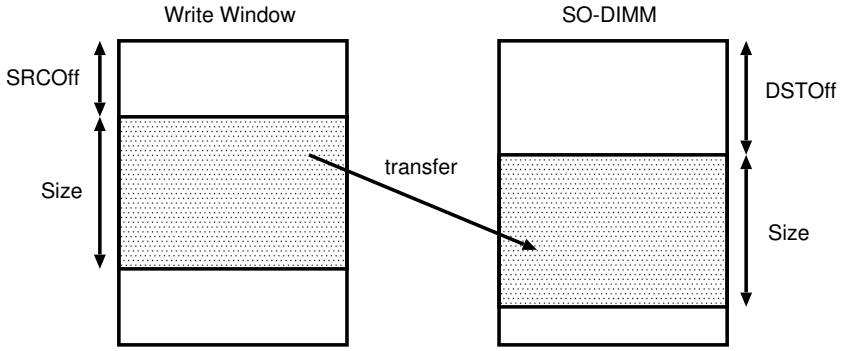


図 4.20　連続ストア

VSI (リストストア)　List で指定された SO-DIMM の位置に格納された Iteration 個のオフセット値 (index 配列：1 要素当り 32bit) に DSTOff を足した位置に, SRCOff で示された Write Window の対応する位置に連続して格納された Iteration 個のデータ (サイズ：DTYPE) を格納するプリミティブである. 図 4.22 にリストストアの動作を示す.

総転送サイズは DTYPE×Iteration [Byte] となる. SRCOff+(総転送サイズ) が対象の Window の領域を超えた場合は, コントローラ側で超えないような値に Iteration 値を書き換えて処理を行う. また, DSTOff+(オフセット値)+(DTYPE) が自 SO-DIMM 領域を超える場合は, 自 SO-DIMM 領域に書き込み可能なサイズまでデータを書き込む. これらの場合は, SO-DIMM への書き込み処理のエラーを示すフラグが変化する (付録 A).

4.6.5　RVL 系列 (Remote Vector Load Family)

RVL 系列のプリミティブには RVL, RVLS (Remote Vector Load Stride), RVLI (Remote Vector Load Index) が定義されている. これらのプリミティブは, RDMA read に相当し, リモートノードでそれぞれ VL, VLS, VLI の処理を行う. 読み出されたデータは連続ストア (RVS) パケットとしてローカル側に送出される.

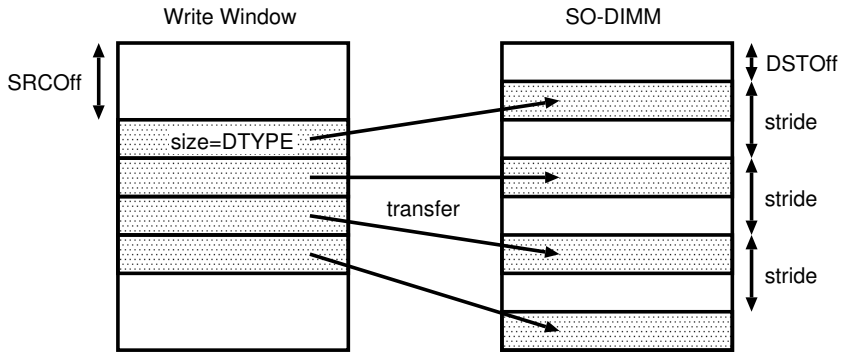


図 4.21　ストライドストア (Iteration=4)

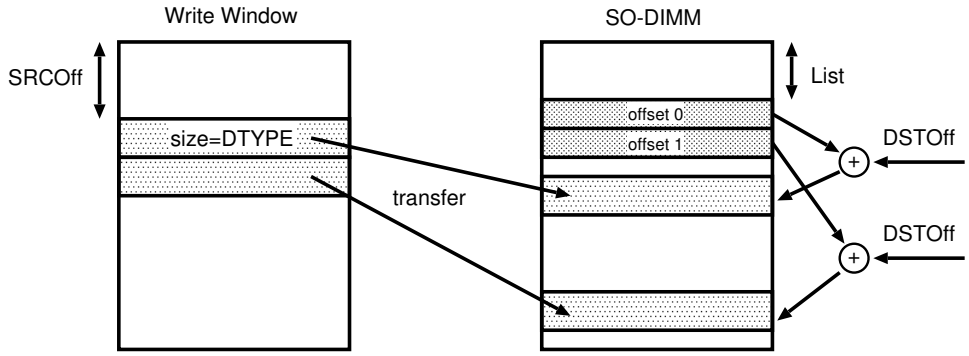


図 4.22　リストストア (Iteration=2)

4.6.6　RVS 系列 (Remote Vector Store Family)

RVS 系列のプリミティブには RVS, RVSS (Remote Vector Store Stride), RVSI (Remote Vector Store Index) が定義されている. これらのプリミティブは, RDMA write に相当し, リモートノードでそれぞれ VS, VSS, VSI が実行される.

4.6.7　IPUSH 系列 (Indirect PUSH Family)

IPUSH 系列のプリミティブは RDMA write に似た処理を行うが, 通常の RDMA write ではデータの格納先アドレスを送信側が指定するのに対し, IPUSH ではデータの格納先アドレスを受信側が指定する. 詳細は 7 章で述べる. 指定する命令コードによって, LHS 機構を利用するかしないかを指定することができる.

4.6.8　SO-DIMM 間コピー

単一の DIMMnet-2 上の SO-DIMM 間で行われるデータコピーは, VL 系プリミティブと VS 系プリミティブの 2 つのプリミティブを連続して実行することで実現される. DSTOff を 0xffff_ffff とした VL 系プリミティブと SRCOff を 0xffff_ffff とした VS 系プリミティブを順に発行することで,

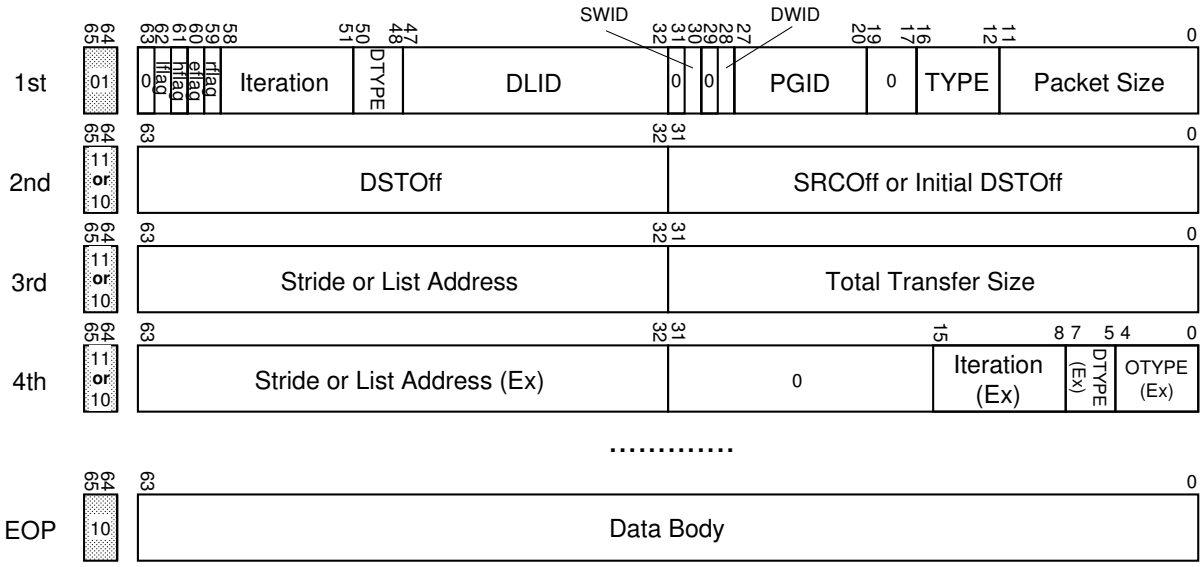


図 4.23　パケットフォーマット

Prefetch Unit と Write Unit がコントローラ内に設けられたバッファを介して SO-DIMM 間のデータ転送を行う. この際, VL 系プリミティブと VS 系プリミティブはリストアクセスを行う VLI, VSI を除いて自由に組み合わせることができる. 例えば, VLS と VS を組み合わせれば, 行列の転置を実現でき, これは FFT などの行列転置を伴うアプリケーションに有効である.

SO-DIMM 間コピー用命令の発行順序が守られているかどうかはネットワークインタフェースコントローラ側では保証せず, DIMMnet-2 を使用するユーザプロセスが保証するものとしている.

4.6.9　Command Ex を利用した拡張プリミティブ

RVS, RVL, IPUSH 系列の各プリミティブでは, ローカル側のデータの書き込み, 及び読み出しは必ず連続ストア, 連続ロードになる. そのため, 不連続領域に格納されたデータをリモートノードに転送する場合は, 転送するデータの要素数だけ RVS を実行するか, VL 系列のプリミティブで Prefetch Window に読み出したデータを Write Window に書き込み, BOTF で転送する必要がある. どちらの方法もホスト側のオーバヘッドが大きく, 高いスループットが期待できない. そこで, ローカル側の SO-DIMM へのアクセスパラメータを指定するレジスタである Command Ex を用いて, ローカル側で不連続領域から読み出したデータをネットワークに送出可能とする.

4.7　パケットフォーマット

DIMMnet-2 のパケットフォーマットを図 4.23 に示す.

図 4.23 の各ラインは 66bit であり, 2bit のライン識別子と 64bit のデータから構成される. 上位 2bit は, そのラインがパケットのどの部分であるかを示し, SWIF や Receive Controller でのラインの識別に用いられる. 表 4.14 に各ラインの上位 2bit と, それの持つ意味を示す.

1st DW (Double Word) から 3rd DW まではパケットヘッダであり, 4th DW から実データ部となる. また, 拡張プリミティブが有効な場合に RVL 系列のプリミティブを実行した場合のみ 4th DW

表 4.14　ライン識別子

識別子	意味
01	パケットの先頭ライン
10	パケットの最終ライン
11	上記以外のライン

までがパケットヘッダとなり, 5th DW から実データ部となる. 各フィールドの詳細を表 4.15 に示す.

表 4.15　パケットヘッダのパラメータ

ライン	フィールド名	ビット幅	内容
1st DW	Packet Size	12	パケットヘッダを含むパケットサイズを Byte 単位で示す.
	TYPE	5	プリミティブの種別を示す (表 4.13).
	PGID	8	送信側のプロセスの PGID を示す.
	DWID	1	受信側のプロセスの WID を示す.
	SWID	1	送信側のプロセスの WID を示す.
	DLID	16	受信側のプロセスの LID を示す.
	DTYPE	3	不連続アクセスを伴うプリミティブの際のデータ 1 要素当たりのサイズを示す (表 4.5).
	Iteration	8	不連続アクセスを伴うプリミティブの際のデータの要素数を示す.
	rflag	1	データの受信先を示す (0：SO-DIMM, 1：Prefetch Window).
	eflag	1	ある 1 つのプリミティブにおける最後のパケットであることを示す. 例えば, MTU が 2048Byte の場合に 9KByte のデータを転送すると, パケットが 5 個送信されることになるが, 最後のパケットのみ eflag が 1 になる. 受信側は eflag が 1 のパケットを受信したときのみパケット受信ステータスを LLCM に書き込む.
	hflag	1	パケットヘッダのライン数を示す (0：2 ライン, 1：3 ラインまたは 4 ライン).
	lflag	1	RVS パケット, 及び IPUSH パケットの受信先を指定するフラグ. lflag が 1 の RVS パケットの場合, LLCM に受信される. また, lflag が 1 の IPUSH パケットの場合, LHS 機能を利用した受信が行われる (7 章).
2nd DW	SRCOff or Initial DSTOff	32	RVL 系パケットの場合はリモートノードでのデータ読み出し元アドレスを示す (SRCOff). RVS 系パケットの場合はプリミティブ発行時に指定された DSTOff を示す (Initial DSTOff). Initial DSTOff の場合は, データがある 1 つのプリミティブでパケットが複数転送される場合, すべてのパケットで同じ値を持つ.
	DSTOff	32	RVL 系パケットの場合はプリミティブを発行した側 (ローカル側) のデータの受信先を示す. RVS 系パケットの場合はリモート側のデータの受信先を示す.
3rd DW	Total Transfer Size	32	そのプリミティブによって転送されるデータ (ヘッダを除く) の総サイズを示す.
	Stride of List Address	32	プリミティブ発行時に指定されたストライド間隔, またはリストが格納されたアドレスを示す.
4th DW	Ex パラメータ		拡張プリミティブ時に指定されたパラメータを示す.

第5章　実装

4 章で示した Core Logic において, 本研究で実装を行ったモジュールはホストインタフェース部, Window Controller, Status Write Unit, 及び AMT Address Table である. 本章では, AMT Address Table 以外のモジュールの実装について述べる. AMT Address Table については 7 章で述べる. また, Receive Controller, Write Unit, 及び Prefetch Unit は文献 [18][104] に詳しい.

本章では実装したモジュールの状態遷移図を記載している. 状態遷移図の説明に際し, 状態名をほかの語句と区別するために本文中では Sans Serif フォントを用いて表記する. また, 状態遷移図中に記載する動作に関しては状態遷移に影響を与える動作のみを示すこととする(注 1).

5.1　Write Window, Prefetch Window, LLCM, LH Buffer

Write Window, Prefetch Window, LLCM, 及び LH Buffer は, いずれもホストプロセッサと Core Logic の要求処理部からアクセスされるメモリ領域であり, 基本構造を変えることなく実装可能である. これらのメモリ領域は Virtex-II Pro に内蔵されている Block RAM を利用して実装する.

ホスト側とは DDR-SDRAM バスで接続されるため, 2×64bit/clock の速度でデータの転送が行われる. そこで, 各メモリ領域を Even と Odd の 2 つの領域に分割し, それぞれの領域を 64bit 幅のメモリとして実装することで DDR-SDRAM バスのデータ転送に対応する. 図 5.1 にこれらのメモリ領域の構造を示す. Even 側にはメモリバスのクロックの立ち上がりの入力 (データ, アドレス, 及び Write Enable) を接続し, Odd 側は立ち下がりの入力を接続する.

ホストプロセッサからは, 通常のメモリモジュールにアクセスするのと同様の方法でこれらのメモリ領域のアドレスを指定し, 読み書きを行う. アドレスの割り当ては表 4.3 で示した通りである.

5.1.1　Write Window

Write Window はホストプロセッサからは書き込みのみが行われ, 要求処理部からは読み出しのみが行われるバッファである. Write Window の構造を図 5.2 に示す.

要求処理部側からは Window Controller と Write Unit がアクセスする. しかし, Write Unit は Window Controller からの要求がない限り Write Window に対してアクセスしない. そのため, Window Controller で Read Enable を制御することで, Write Unit と Window Controller が同時に Write Window に対してアクセスすることがないようにしている.

5.1.2　Prefetch Window

Prefetch Window はホストプロセッサからは読み出しのみが行われ, 要求処理部からは書き込みのみが行われるバッファである. Prefetch Window の構造を図 5.3 に示す.

(注 1)カウンタのデクリメントやフラグの変化など

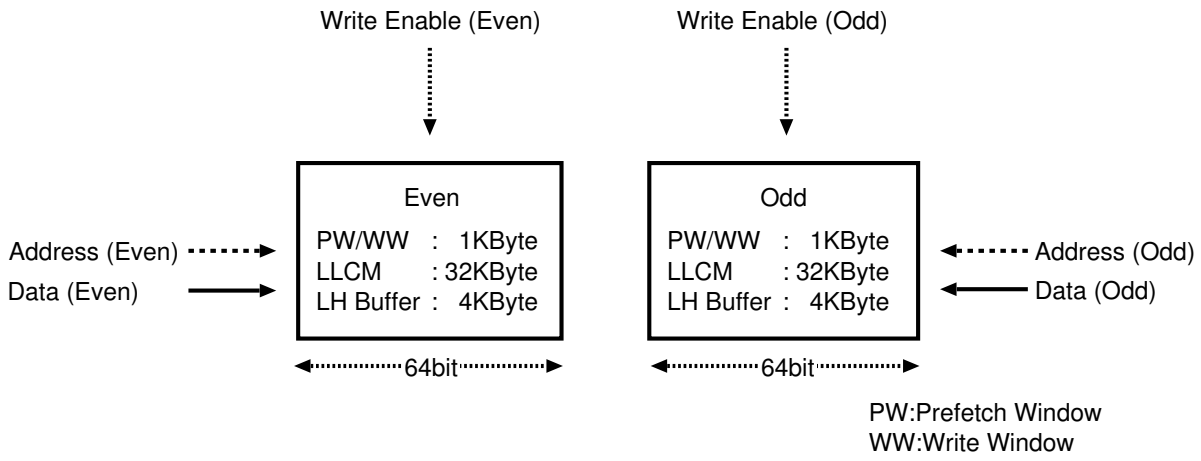


図 5.1 Write Window, Prefetch Window, LLCM の構造

Prefetch Window に対して書き込みを行うモジュールは Receive Controller と Prefetch Unit である. これらのモジュールからのアクセス制御は要求処理部内に設けたアービタを用いて行うため, Prefetch Window に対して同時にアクセスするモジュールは常に 1 つとなる.

5.1.3 LLCM

LLCM はホストプロセッサからも要求処理部からも読み書き可能なバッファである. LLCM の構造を図 5.4 に示す.

LLCM には, ホストプロセッサからは 128bit 単位で読み書き可能であるが, 要求処理部からは 64bit 単位での読み書きのみが可能である. これは, LLCM にはパケット受信ステータスなど, 少量のデータの読み書きが行われるため, 高いスループットを必要としないためである. 64bit 単位での読み書きに限定することにより, 128bit 単位での読み書きに比べて要求処理部側のデータ線の幅を半分にすることができ, FPGA の配線資源が削減されるという利点がある.

ホスト側からは通常のメモリモジュールにアクセスするのと同様の作法でアクセスされるため, 64bit 単位でのアクセスのみ可能とすると, ホスト側の制御が複雑になる. そのため, ホスト側からは 64bit×2=128bit 単位でのアクセスとなっている.

5.1.4 LH Buffer

LH Buffer はホストプロセッサからは読み出しのみが行われ, 要求処理部からは書き込みのみが行われるバッファである. LH Buffer の構造を図 5.5 に示す.

LH Buffer はリングバッファとして用いるため, 内部でアドレス管理を行う論理が組み込まれている. Receive Controller からは順番にバッファのエントリを使用していく. 一方, ホストプロセッサからの読み出しの際には, バッファのどこからでも読み出せるような構造にしている. LH Buffer はメッセージ通信を用いた際に, メッセージのエンベロープ部を格納するためのバッファである (7 章). ホスト側はメッセージの受信関数を呼んだ際に LH Buffer 内のエンベロープを探索し, 受信関数で指定したメッセージが既に受信されているかどうかを調べる. ホスト側は一致するエンベロー

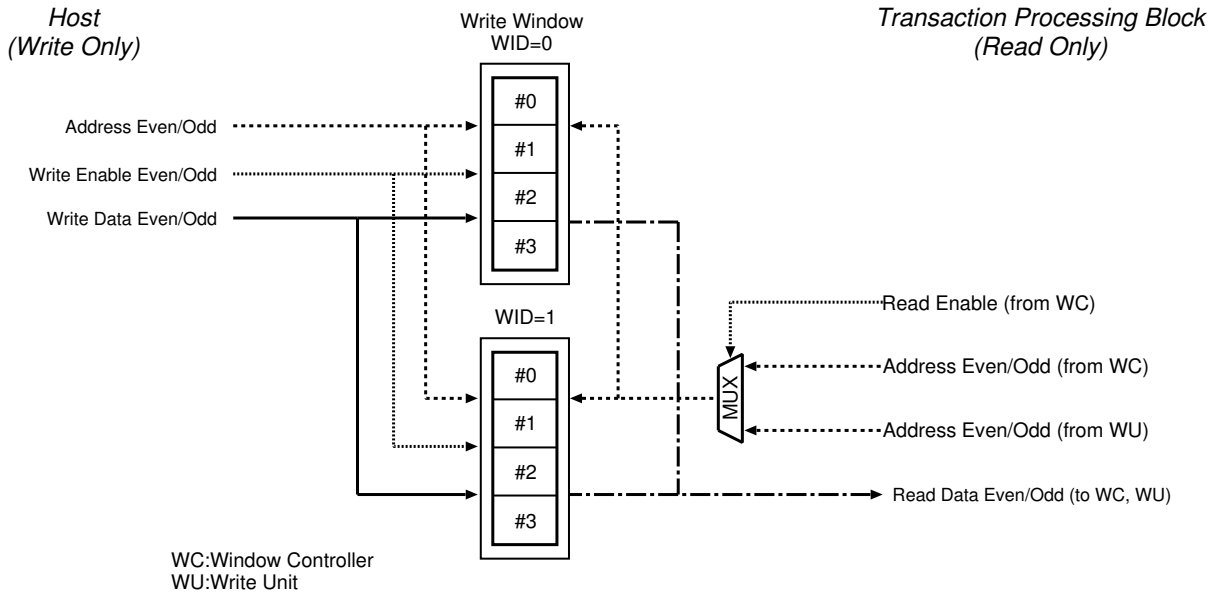


図 5.2 Write Window の構造

プをバッファから取り出すが, このエンベロープが常にバッファの先頭にあるとは限らないため, ホスト側からはバッファのどこからでも読み出せるようになっている.

Receive Controller から書き込みが行われると, 内部で Tail ポインタを更新し, User Register に出力する. ホストプロセッサは User Register 経由で Head ポインタの更新を行う. Tail ポインタと Head ポインタの差からバッファに空きがあるかどうかを判定し, 空きがない場合は Receive Controller に full 信号を出力する.

5.2 Register

4.5.5 節で述べた通り, Register は数多くの設定・制御系レジスタ, 要求発行レジスタ, 及びステータスレジスタから構成される. これらのうち, 設定・制御系レジスタと要求発行レジスタはホストから書き込みを行い, 書き込まれた値を要求処理部の各モジュールに出力するためのレジスタである. また, ステータスレジスタは要求処理部から書き込みが行われ, ホストからは読み出しが行われるレジスタである. ホストからこれらのレジスタへは Address, Data, 及び Write Enable 信号が入力となる. Address をデコードして得られた結果を用いて, 読み書きを行うレジスタを指定する (図 5.6).

5.2.1 設定・制御系レジスタ

設定・制御系レジスタは SO-DIMM の容量や, ネットワークスイッチの MTU 設定など, Core Logic の動作を決定するのに用いるレジスタである. 設定値の出力先は User Register の設定・制御系レジスタの場合, Receive Controller, 及び Status Write Unit であり, System Register の設定・制御系レジスタの場合, Window Controller となる.

設定・制御系レジスタに値がホストから書き込まれると, 次のクロックで, どのレジスタが書き

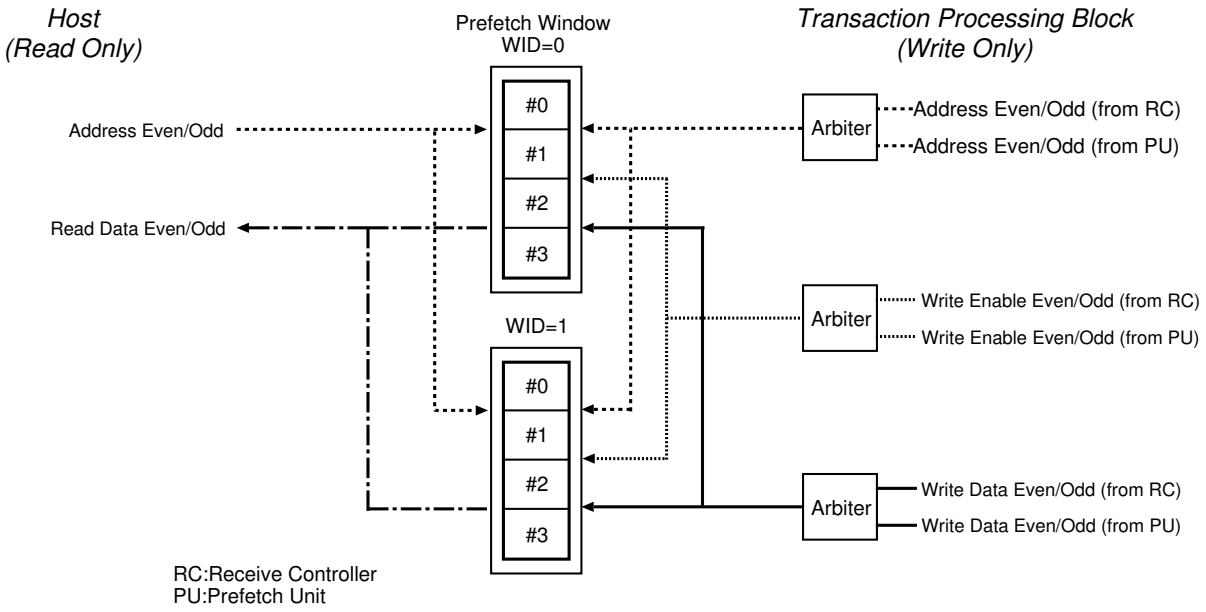


図 5.3 Prefetch Window の構造

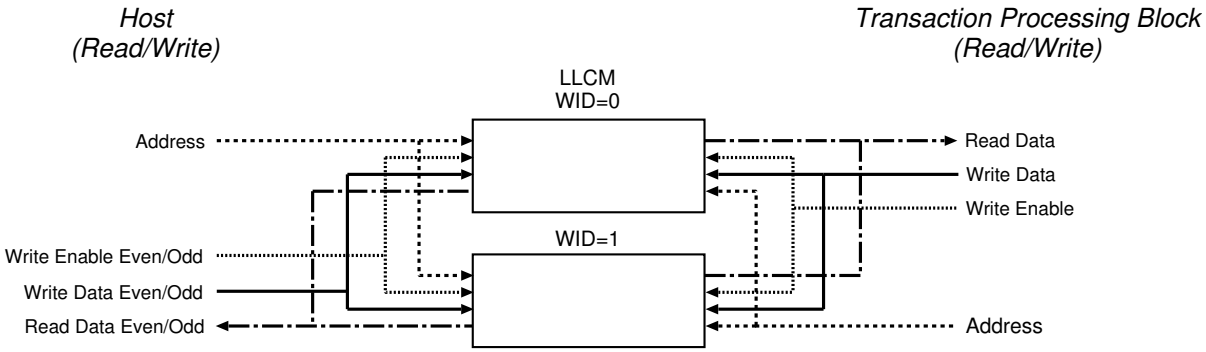


図 5.4 LLCM の構造

換えられたのかを示すビット (有効ビット) と, どのプロセスの値が更新されたのかを示す WID が付加されて出力される. 受信側のモジュールは有効ビットをチェックすることで, 必要なデータのみを読み込む. 有効ビットの種類と, それに対応するビット, 及び出力先を表 5.1 に示す.

5.2.2 要求発行レジスタ

要求発行レジスタはホストプロセッサから要求を発行する際に使用するレジスタである. 発行された要求は Window Controller 内部の Request FIFO (5.3 節) に格納される. ホストプロセッサが要求を発行し, それが Window Controller に転送されるまでの流れは以下のようになる.

1. ホストから Command0 Upper (Command1 Upper) にプリミティブの上位 64bit を書き込む.
2. 拡張プリミティブが有効になっている場合は, ホストから Command Ex に拡張プリミティブ

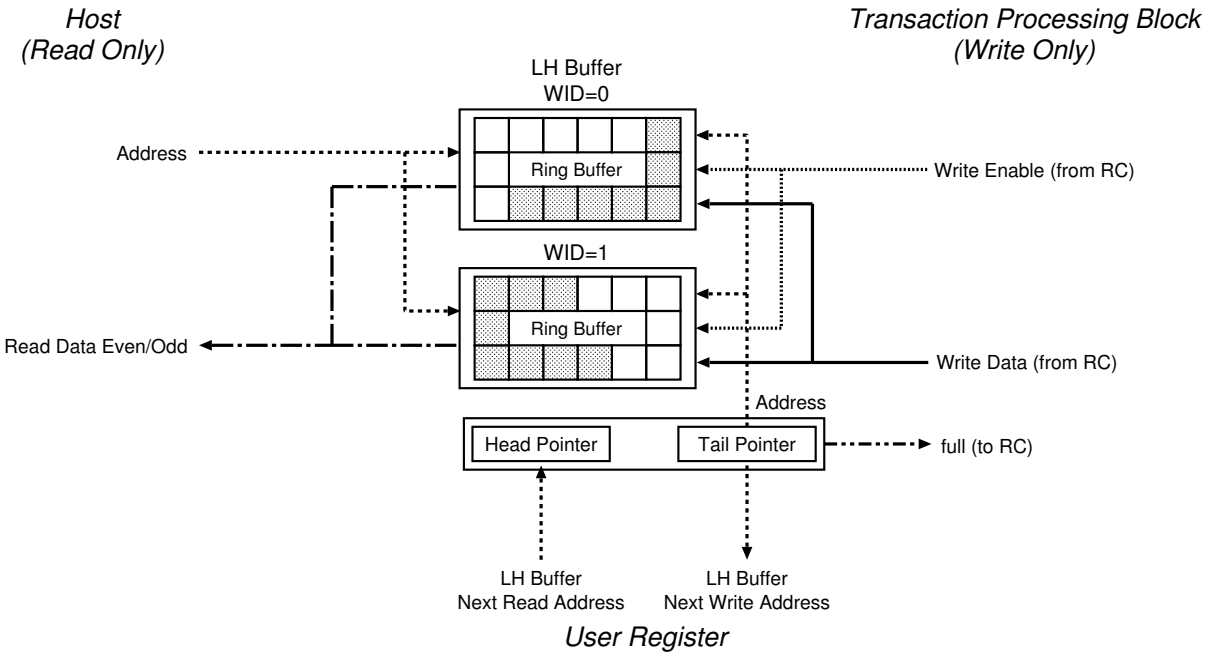


図 5.5　LH Buffer の構造

のパラメータを書き込む.

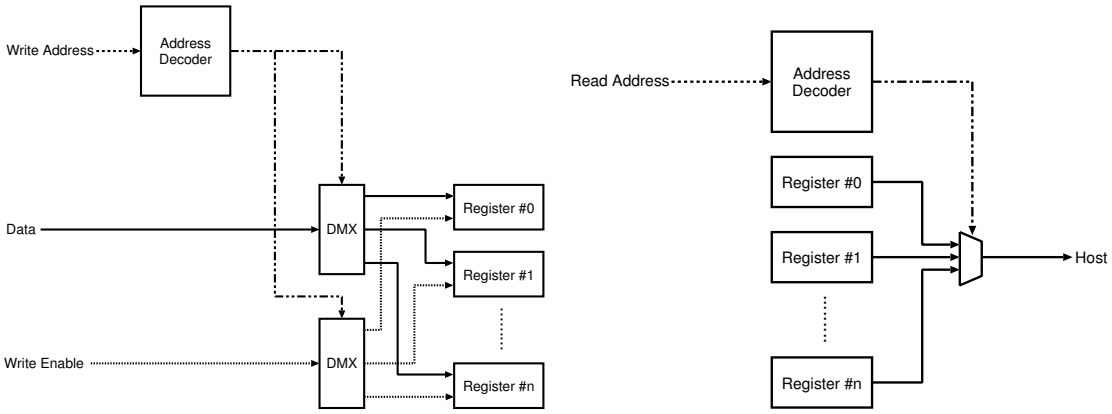
3. ホストから Command0 Lower (Command1 Lower) にプリミティブの下位 64bit を書き込む. Command0 Lower (Command1 Lower) に値が書き込まれると, 要求を Window Controller に転送可能であることを示すフラグが立つ.

4. Window Controller の Request FIFO に空きがあればプリミティブが転送され, フラグを下げる. Request FIFO に空きがなければ, 空きができるまで待機する.

このフラグはホストプロセッサからも読み出すことが可能であり, User Register 内部の Controller Status (付録 A) の 0bit 目にマップされている. また, Controller Status の 2bit 目には Request FIFO が almost full であるかどうかを示すフラグがあり, ユーザプロセスはこのフラグをポーリングすることで次のプリミティブを発行可能かどうか判断する.

要求発行レジスタから Window Controller 内部の Request FIFO にプリミティブを転送する際, 128bit 目にパケットの受信先を示すフラグ (SO-DIMM：0, Prefetch Window：1), 129bit 目に WID が付加され, 130bit の要求に拡張される (図 5.7). 要求されたプリミティブが通信要求でない場合は, 128bit 目のフラグは Core Logic 内部での処理には影響を与えない. Command Ex の内容は図 5.7 の 130bit 幅のデータとは別に Request FIFO に格納される. Request FIFO は全プロセスで共通であるため, WID を付加することで Window Controller は, どのプロセスからの要求であるかを判定する.

プリミティブが BOTF の場合はプリミティブの下位 64bit のみを使用するため, Command0 Upper (Command1 Upper) への書き込みを行う必要がない. 結果として, BOTF の要求発行時のレイテンシは, ほかのプリミティブよりも小さくなる.



(a) ホストからレジスタ部への入力　　(b) レジスタ部からホストへの出力

図 5.6　ホスト – レジスタ間の入出力

表 5.1　有効ビット

プロセス	種類	ビット	出力先
ユーザプロセス	IPUSH Area Size	3'b010	Receive Controller
	Status Initial Address	3'b101	Status Write Unit
	Status Area Size	3'b110	
	Status Next Read Address	3'b111	
特権プロセス	SO-DIMM Capacity	3'b001	Window Controller
	MTU	3'b010	
	LID	3'b011	
	PGID (WID=0)	3'b100	
	PGID (WID=1)	3'b101	
共通	INVALID	3'b000	ALL

5.3 Window Controller

Window Controller はホストからの要求を受け, それに従ってパケットの生成と SWIF への転送, Write Unit, 及び Prefetch Unit の制御を行う. また, Receive Controller からの RVL 系通信要求も受け付ける.

Window Controller は Request Acceptor, Request Executor, Request FIFO, 及び Configuration Register から構成される. Window Controller の構成を図 5.8 に示す.

Request Acceptor は, 要求発行レジスタから転送されたプリミティブや Receive Controller からの RVL 系プリミティブの要求を受け付け, ソースアドレス, デスティネーションアドレス, 転送サイズなどのデコードを行う. Request Executor は Request Acceptor がデコードした要求を処理する.

Configuration Register は, パケット生成に必要となる MTU や PGID といった情報を設定レジスタから受け取り, 表 4.10 などに示した値に従ったデコード, 及び値の保持を行う.

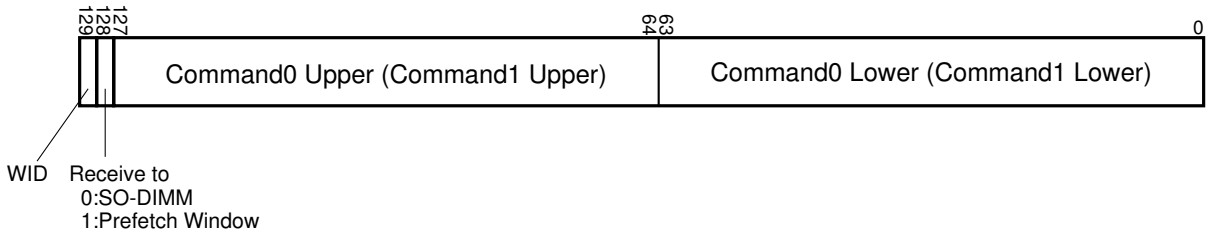


図 5.7　User Register から転送するプリミティブのフィールドフォーマット

SWIF との間には FIFO が存在しており, Window Controller は図 4.23 に示した 66bit 幅のデータを毎クロック FIFO に転送する. この FIFO では Core Logic の動作周波数である 100MHz と SWIF の動作周波数である 125MHz の乗せ換えを行う. Core Logic は 100MHz の周波数で 66bit ずつ転送するが, 実データ部が 64bit であるため, DIMMnet-2 試作基板の最大実効スループットは片方向で 800MByte/s となる. また, Window Controller から SWIF への転送と SWIF から Receive Controller への転送は並行して実行できるため, 双方向の最大実効スループットは 1.6GByte/s となる.

5.3.1 Request Acceptor

Request Acceptor は要求発行レジスタから発行されたプリミティブをデコードし, Request Executor に未処理の要求が存在することを通知する. Request Acceptor は図 5.9 に示す状態遷移を行う.

1. NOP：ホスト (要求発行レジスタ), または Receive Controller からプリミティブが発行されるのを待つ.
   - プリミティブが発行されると, プリミティブが書き込まれた方の Request FIFO に対して Read Enable を High にし, REQ に遷移する.
   - プリミティブが発行されていない場合は NOP へ遷移する.
2. REQ：Request FIFO からプリミティブが読み出されるのを待つ.
   - Read Enable を Low にし, REQ IN に遷移する.
3. REQ IN：プリミティブをデコードし, Request Executer に対して Request Enable を High にして出力し, 未処理のプリミティブが存在することを通知する.
   - WAIT に遷移する.
4. WAIT：Request Executer からプリミティブ読み出しの完了が通知されるまで待つ.
   - Request Executer から完了通知がなされたら NOP へ遷移する.
   - 完了通知がなされていない場合は WAIT へ遷移する.

Request Executor があるプリミティブを処理している間に次のプリミティブが発行されると, Request Acceptor は Request FIFO からのプリミティブの読み出し, 及びデコードを行う. そして, デコード後のプリミティブを Request Executor から読み出されるまで WAIT で待ち続ける.

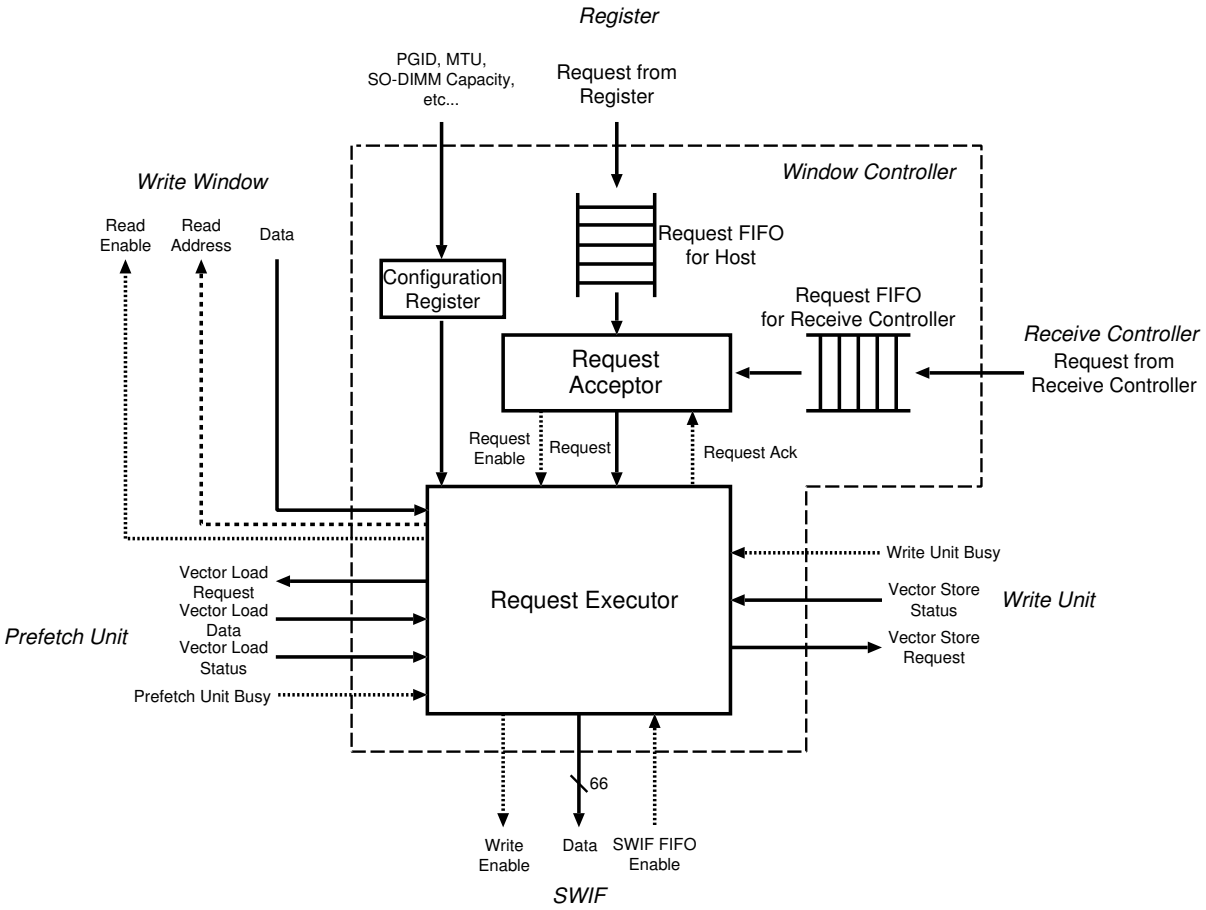


図 5.8 Window Controller の構成

このように, プリミティブの受け付け, 及びデコードを行うモジュールをプリミティブを処理するモジュールと分離することにより, 連続してプリミティブが発行された場合に, プリミティブ間のインターバルを削減することが可能となる.

5.3.2 Request Executor

Request Executor は Request Acceptor がデコードしたプリミティブを処理するモジュールである. Request Executor は, 各プリミティブの系列ごとに独立したモジュールと, それらを制御するコントローラから構成される (図 5.10). このような構成にすることで, あるプリミティブ系列に新しいプリミティブを追加する, またはプリミティブの系列を増やすといった場合に, ほかのモジュールへ影響を与えることなく拡張することが可能になる.

Request Executor は Request Acceptor から未処理のプリミティブ要求が存在することを通知されると, プリミティブの種別に応じて, プリミティブを処理するモジュールを起動する. 存在しないプリミティブの種別だった場合(注 2)は, モジュールを起動せずに次の要求を待つ.

(注 2)表 4.13 に示した 5bit のコードの上位 2bit で識別

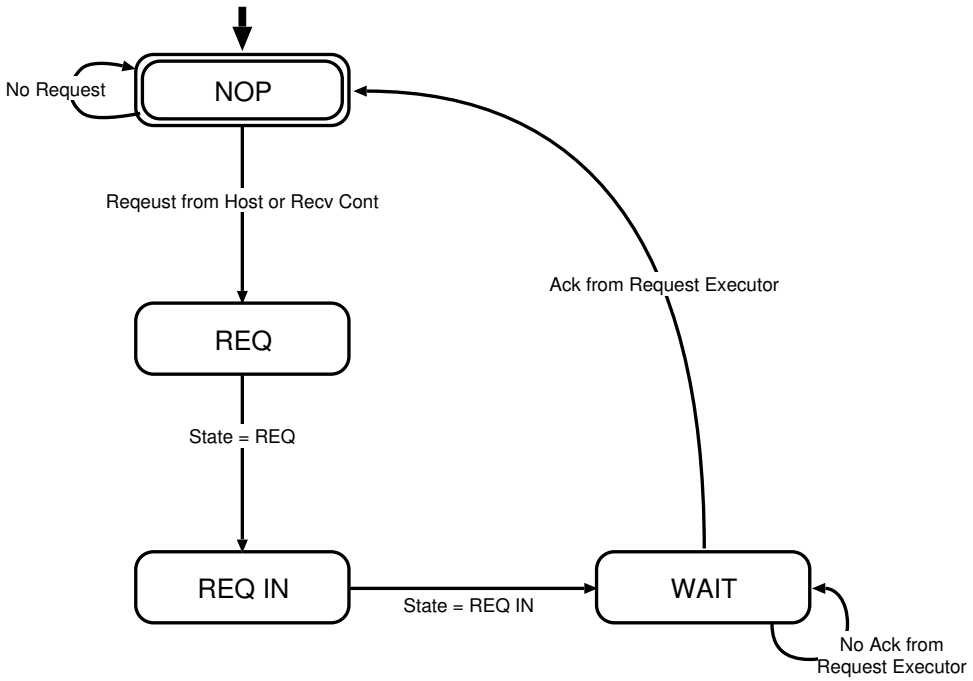


図 5.9　Request Acceptor の状態遷移図

5.3.3　BOTF 処理時の状態遷移

Request Acceptor から読み出した要求が BOTF であった場合, 図 5.11 に示す状態遷移を行う. BOTF はプリミティブで指定された転送サイズから転送するライン数(注 3)を求め, そのライン数だけ Write Window へアクセスする. そのため, このライン数をカウンタとし, 特定のステートを繰り返すようになっている.

1. PRIM：転送するデータサイズから Write Window にアクセスする回数を求め, その値をカウンタ (Transfer Counter) にセットする.
   - 転送するデータサイズのチェックを行い, 転送サイズが 0 だった場合は, 無効な要求として処理を終了し, PRIM に遷移する.
   - 転送サイズが 0 でなく, SWIF FIFO に空きがない場合は WAIT に遷移する.
   - 転送サイズが 0 でなく, さらに SWIF FIFO に空きがある場合は, Write Window に読み出し要求を出し, BOTF1 に遷移する.
2. WAIT：SWIF FIFO に空きができるまで待つ.
   - SWIF FIFO に空きができると, Write Window に読み出し要求を出し, BOTF1 に遷移する.
   - SWIF FIFO に空きがない場合は WAIT に遷移する.
3. BOTF1：Write Window からのデータを待つ.

(注 3)1 ライン 64bit

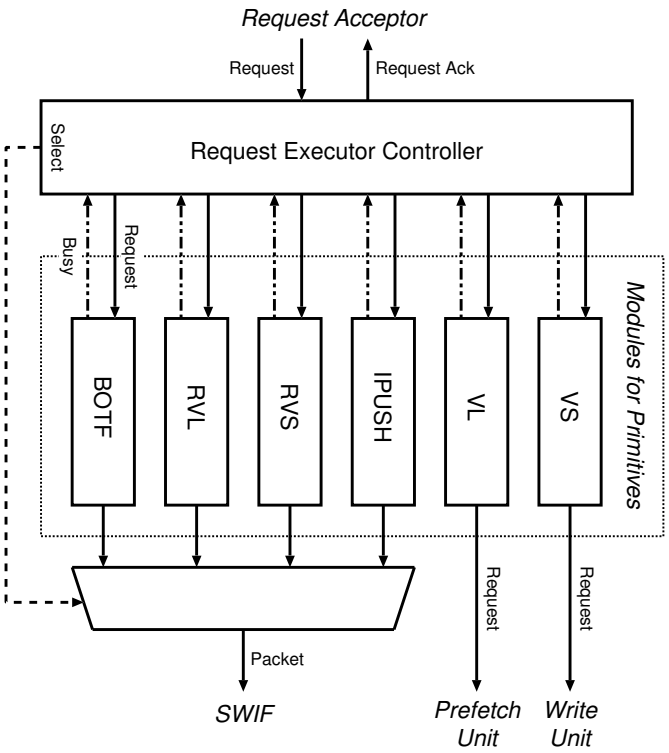


図 5.10 Request Executor

- Transfer Counter をデクリメントし, BOTF2 に遷移する.

4. BOTF2：Write Window から読み出したデータを Window Controller 内部の転送用レジスタに格納する. このデータはヘッダの最初のラインであり, PGID のフィールドを設定レジスタの内容に変更する.
   - Transfer Counter≠1 である場合はカウンタをデクリメントし, BOTF3 に遷移する.
   - Transfer Counter=1 である場合は BOTF END1 に遷移する.
5. BOTF3：前の状態で転送用レジスタに格納したデータに 2bit のライン識別子を付加して SWIF に転送する. Write Window から新たに読み出したデータを転送用レジスタに格納する.
   - Transfer Counter≠1 である場合はカウンタをデクリメントし, BOTF4 に遷移する.
   - Transfer Counter=1 である場合は BOTF END1 に遷移する.
6. BOTF4：BOTF3 で転送用レジスタに格納したデータに 2bit のライン識別子を付加して SWIF に転送する. Write Window から新たに読み出したデータをデータ転送用レジスタに格納する.
   - Transfer Counter≠1 である場合はカウンタをデクリメントし, BOTF3 に遷移する.
   - Transfer Counter=1 である場合は BOTF END1 に遷移する.
7. BOTF END1：SWIF に最後のラインを送信する.
   - BOTF END2 に遷移する.

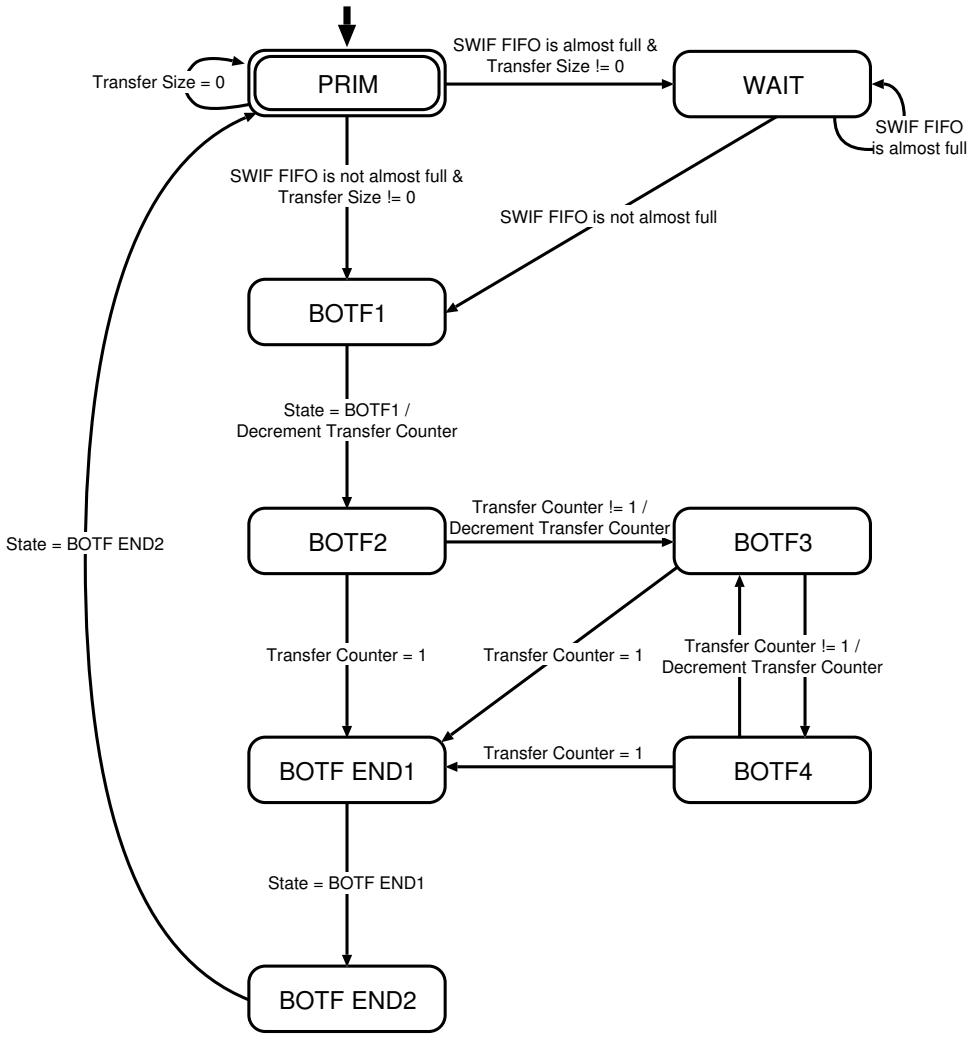


図 5.11　BOTF の状態遷移図

8. BOTF END2：内部のレジスタをリセットする.
    - PRIM に遷移する.

5.3.4　VL 系プリミティブ処理時の状態遷移

Request Acceptor から読み出した要求が VL 系プリミティブであった場合, 図 5.12 に示す状態遷移を行う.

VL 系プリミティブでは Prefetch Unit が処理の主体となるため, Window Controlller は Prefetch Unit にデータ読み出し要求を発行した後は, その処理が完了するまで待つだけである.

1. PRIM：Prefetch Unit へのアクセス要求を発行.
    - Prefetch Unit が not busy の場合, Prefetch Unit にデータ読み出し要求を出し, VL1 に遷移する.

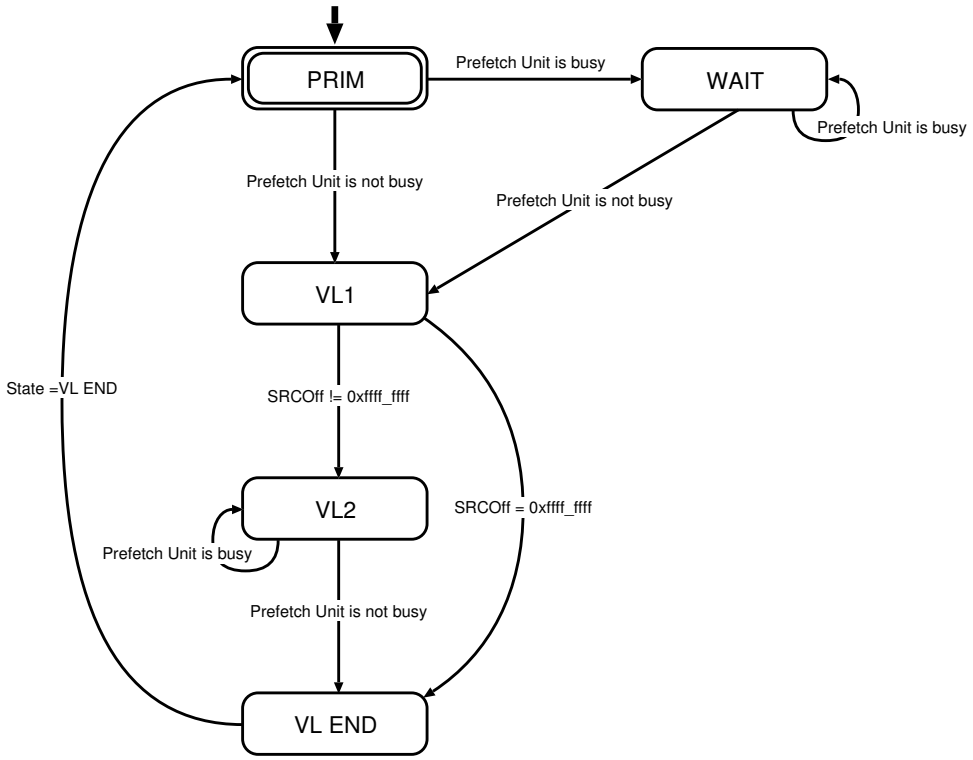


図 5.12　VL 系プリミティブの状態遷移図

- Prefetch Unit が busy の場合, WAIT に遷移する.

2. WAIT：Prefetch Unit が not busy になるまで待つ.
   - Prefetch Unit が not busy になると, Prefetch Unit にデータ読み出し要求を出し, VL1 に遷移する.
   - Prefetch Unit が busy の場合, WAIT に遷移する.
3. VL1：SO-DIMM 間コピー要求かどうかの判定を行う. SO-DIMM 間コピーの場合, VL 系プリミティブに続く VS 系プリミティブと組み合わせて処理を行う. そのため, 次の VS 系プリミティブを受け付け, Write Unit を起動させる必要があるため, SO-DIMM 間コピーの場合は Prefetch Unit の処理の完了を待たずに終了する.
   - プリミティブの SRCOff が 0xffff_ffff であった場合, SO-DIMM 間コピーであると判断し, VL END に遷移する.
   - SRCOff が 0xffff_ffff でない場合は通常の VL 系プリミティブであると判断し, VL2 に遷移する.
4. VL2：Prefetch Unit が not busy になるまで待つ. Prefetch Unit が not busy になると, 要求が完了したと判断する.
   - Prefech Unit が not busy になると, VL END に遷移する.
   - Prefech Unit が busy の場合, VL2 に遷移する.

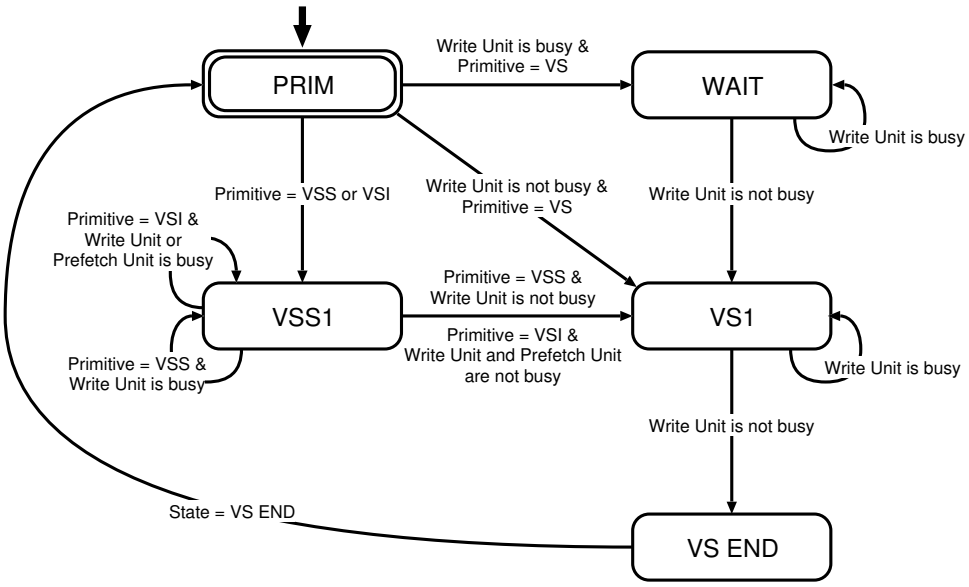


図 5.13　VS 系プリミティブの状態遷移図

5. VL END：内部のレジスタをリセットする.

   - PRIM に遷移する.

5.3.5　VS 系プリミティブ処理時の状態遷移

Request Acceptor から読み出した要求が VS 系プリミティブであった場合, 図 5.13 に示す状態遷移を行う.

VS 系プリミティブでは Write Unit が処理の主体となるため, Window Controlller は Write Unit にデータ書き込み要求を発行した後は, その処理が完了するまで待つだけである. ただし, VSS など不連続アクセスを伴う場合は転送サイズの計算などを行うステートに遷移した後に Write Unit に要求を発行する.

1. PRIM：要求されたプリミティブが VS の場合, ソースアドレスと転送サイズから, データ転送領域が Write Window の領域を超えていないかどうかを検出し, 超えている場合は超えないようにサイズを修正する. サイズの修正が発生した場合, そのことを示すフラグを立てる.
   要求されたプリミティブが VSS または VSI の場合, DTYPE と Iteration から総転送サイズを, SRCOff から Write Window から読み出すことのできるデータサイズを求める.

   - 要求されたプリミティブが VS であり, Write Unit が not busy の場合, Write Unit にデータ書き込み要求を出力し, VS1 に遷移する.
   - 要求されたプリミティブが VS であり, Write Unit が busy の場合, WAIT に遷移する.
   - 要求されたプリミティブが VSS または VSI であれば, VSS1 に遷移する.

2. WAIT：Write Unit が not busy になるまで待つ.

- Write Unit が not busy になると, Write Unit にデータ書き込み要求を出し, VS1 に遷移する.
- Write Unit が busy の場合, WAIT に遷移する.

3. VSS1：PRIM で求めた総転送サイズと Write Window から読み出すことのできるデータサイズから, データ転送領域が Write Window の領域を超えていないかどうかを検出し, 超えている場合は超えないように Iteration 数を修正する. Iteration 数の修正が発生した場合, そのことを示すフラグを立てる.
   - プリミティブが VSS であり, Write Unit が not busy の場合, Write Unit に書き込み要求を出し, VS1 に遷移する.
   - プリミティブが VSS であり, Write Unit が busy の場合, VSS1 に遷移する.
   - プリミティブが VSI であり, Write Unit と Prefetch Unit が not busy の場合, Write Unit に書き込み要求を, Prefetch Unit にリストの読み出し要求を出し, VS1 に遷移する.
   - プリミティブが VSI であり, Write Unit または Prefetch Unit が busy の場合, VSS1 に遷移する.
4. VS1：Write Unit が not busy になるまで待つ. Write Unit が not busy になると, 要求が完了したと判断し, Write Unit からのステータス (図 5.8 の Vector Store Status) を User Register に出力する.
   - Write Unit が not busy になると, VS END に遷移する.
   - Write Unit が busy の場合, VS1 に遷移する.
5. VS END：内部のレジスタをリセットする.
   - PRIM に遷移する.

5.3.6 RVL 系プリミティブ処理時の状態遷移

Request Acceptor から読み出した要求が RVL 系プリミティブであり, それがホストから発行されたものであった場合, 図 5.14 に示す状態遷移を行う.

RVL 系プリミティブでは, 発行した側のノードはリモートノードに転送サイズやアドレスの情報を送信するだけであるため, 図 4.23 に示される 1st DW〜4th DW をリモートノードに転送して終了する.

1. PRIM：SWIF FIFO の空きをチェック.
   - SWIF FIFO に空きがある場合はパケットの 1st DW を SWIF に転送し, RVL1 に遷移する.
   - SWIF FIFO に空きがない場合, WAIT に遷移する.
2. WAIT：SWIF FIFO に空きができるまで待つ.
   - SWIF FIFO に空きができると, パケットの 1st DW を SWIF に転送し, RVL1 に遷移する.
   - SWIF FIFO に空きがない場合, WAIT に遷移する.

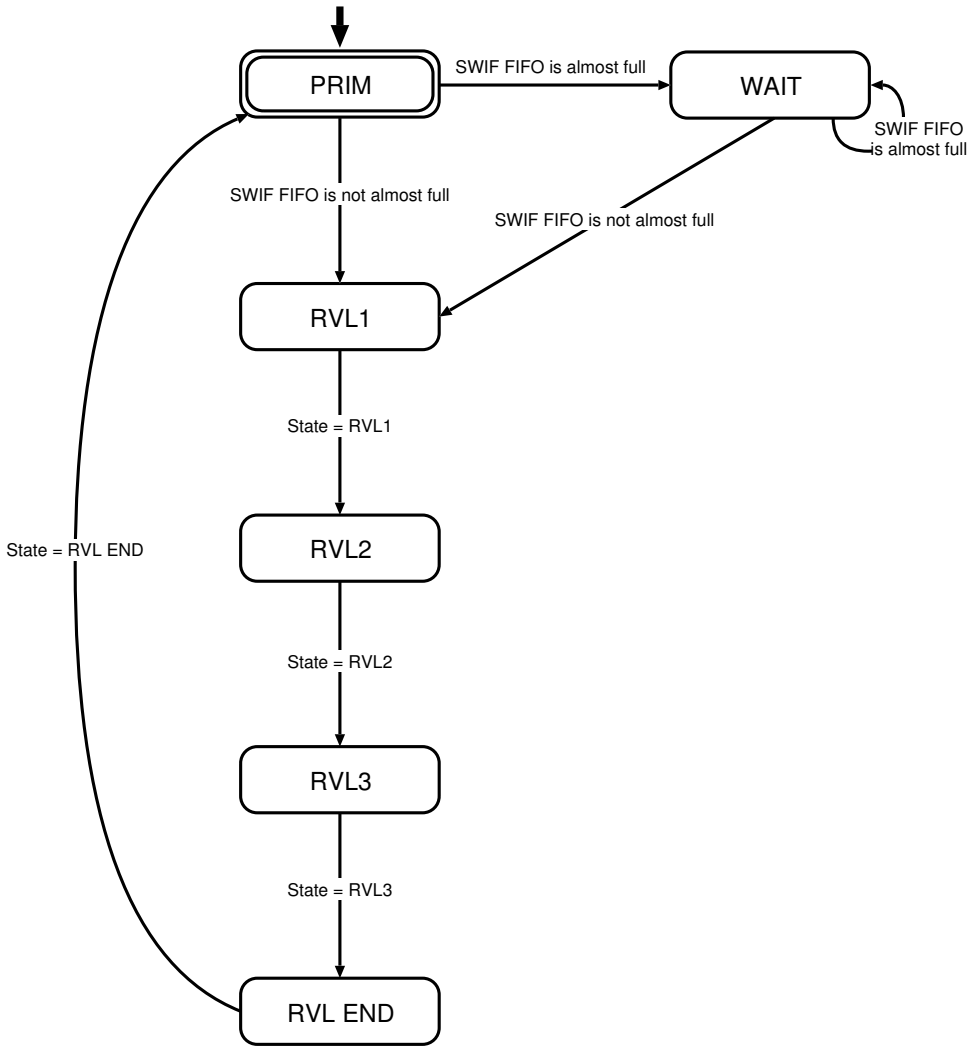


図 5.14　RVL 系プリミティブの状態遷移図

3. RVL1：パケットの 2nd DW を SWIF に転送する.
    - RVL2 に遷移する.
4. RVL2：パケットの 3rd DW を SWIF に転送する.
    - RVL3 に遷移する.
5. RVL3：パケットの 4th DW を SWIF に転送する. この 4th DW は拡張プリミティブが有効でない場合は意味を持たないラインである.
    - RVL END に遷移する.
6. RVL END：内部のレジスタをリセットする.
    - PRIM に遷移する.

Receive Controller から渡される RVL 系のプリミティブは RVS 系プリミティブとして処理される. Receive Controller からの RVL 系プリミティブの場合, その RVL 系プリミティブの発行元はリモートノードである. そのため, RVL 系プリミティブの要求を Receive Controller から受けた側のノードは, リモートノードに RVS 系プリミティブのパケットを送信する. この処理の際に, Window Controller 内部でプリミティブの書き換えが行われ, RVS 系プリミティブの状態遷移 (5.3.7 節) に従って処理が行われる.

5.3.7 RVS 系プリミティブ処理時の状態遷移

Request Acceptor から読み出した要求が RVS 系プリミティブであった場合, 図 5.15 に示す状態遷移を行う.

RVS 系プリミティブは指定されたプリミティブや転送サイズにより, ヘッダのライン数や生成するパケット数が異なるため, ほかのプリミティブに比べて複雑な状態遷移を行う.

1. PRIM：要求されたプリミティブが RVS の場合は, 転送するサイズ (Total Transfer Size) から, そのパケットで転送するライン数をカウンタ (Data Counter) にセットする. また, 1 パケットで転送が終了する場合は最後のパケットであることを示すフラグ (Last Packet Flag) を立てる.
   要求されたプリミティブが RVSS または RVSI の場合, DTYPE と Iteration から総転送サイズを求める.

   - 要求されたプリミティブが RVS であり, Prefetch Unit が not busy の場合, Prefetch Unit にデータ読み出し要求を出し, RVS1 に遷移する.
   - 要求されたプリミティブが RVSS または RVSI であり, Prefetch Unit が not busy の場合, Prefetch Unit にデータ読み出し要求を出し, RVSS1 に遷移する.
   - Prefetch Unit が busy の場合, WAIT に遷移する.

2. WAIT：Prefetch Unit が not busy になるまで待つ.

   - Prefetch Unit が not busy になり, 要求されたプリミティブが RVS の場合, Prefech Unit にデータ読み出し要求を出し, RVS1 に遷移する.
   - Prefetch Unit が not busy になり, 要求されたプリミティブが RVSS または RVSI の場合, RVSS1 に遷移する.
   - Prefetch Unit が busy の場合, WAIT に遷移する.

3. RVS1：パケットヘッダの 1st DW と 2nd DW を生成する. このパケットで転送するサイズを Total Transfer Size から引き, 残りの転送サイズを計算する.

   - SWIF FIFO に空きがあり, パケットヘッダが 3 ラインの場合は RVS2 に遷移する.
   - SWIF FIFO に空きがあり, パケットヘッダが 2 ラインの場合は RVS3 に遷移する.
   - SWIF FIFO に空きがない場合は, RVS1 に遷移する.

4. RVS2：パケットヘッダの 3rd DW を生成する.

   - Prefetch Unit がデータの読み出しを完了したことを検出すると, RVS4 に遷移する.

- Prefetch Unit のデータの読み出しが未完了である場合は, RVS2 に遷移する.

5. RVS3：Prefetch Unit の処理が完了するまで待つ.
   - Prefetch Unit がデータの読み出しを完了したことを検出すると, RVS4 に遷移する.
   - Prefetch Unit のデータの読み出しが未完了である場合は, RVS3 に遷移する.
6. RVS4：すべてのパケットヘッダを SWIF に転送する. 従って, このステートはパケットヘッダのライン数に応じて 2 クロックまたは 3 クロック要する.
   - 未送出のパケットヘッダがある場合, RVS4 に遷移する.
   - パケットヘッダをすべて送出すると, RVS5 に遷移する.
7. RVS5：Data Counter をデクリメントし, Data Counter が 0 になるまで Prefetch Unit から得たデータを SWIF へ転送する.
   - Data Counter が 0 でない場合, RVS5 に遷移する.
   - プリミティブが RVS の場合で, Data Counter が 0, かつ Last Packet Flag が立っていないならば, 次のパケットを送信するために RVS1 に遷移する. また, 同時に残りの転送サイズから, 次のパケットが最後のパケットかどうかを判定し, 最後のパケットの場合は Last Packet Flag を立てる.
   - プリミティブが RVSS または RVSI の場合で, Data Counter が 0, かつ Last Packet Flag が立っていないならば, RVSS1 に遷移する.
   - Data Counter が 0, かつ Last Packet Flag が立っているならば RVS END に遷移する.
8. RVSS1：求めた総転送サイズから, 1 パケットの転送で完了するかどうか判定する.
   - 1 パケットで終了する場合は Last Packet Flag を立て, RVS1 に遷移する. 現パケットで転送するライン数をカウンタにセットする.
   - 複数パケットを転送する場合は, このパケットで転送するデータの要素数を計算し, RVSS2 に遷移する.
9. RVSS2：RVSS1 で求めたデータの要素数と DTYPE からこのパケットで転送するデータサイズを計算し, 転送するライン数をカウンタにセットする.
   - RVS1 へ遷移する.
10. RVS END：内部のレジスタをリセットする.
    - PRIM に遷移する.

5.3.8 IPUSH 系プリミティブ処理時の状態遷移

IPUSH 系プリミティブの送信側の処理は RVS 系プリミティブと同じであるため, 基本的な状態遷移は同じである. ただし, LHS 機構を同時に使用する場合 (IPUSH with LHSv2) は最初に Write Window のデータを転送し, それに続けて SO-DIMM のデータを転送するため, Prefetch Unit にデータ読み出し要求を出す前に, Write Window のデータを読み出す処理が加わる. IPUSH, 及び LHS については 7 章にて述べる.

5.4 Status Write Unit

Status Write Unit は, Receive Controller からのパケット受信ステータスの書き込み要求に応じて, LLCM へパケット受信ステータスを書き込む処理を行う. 図 5.16 に Status Write Unit と周辺モジュールの構造を示す.

Status Write Unit は Request Separator からパケット受信ステータス書き込み要求を受けることにより, パケット受信ステータスの書き込みを開始する. Request Separator は Receive Controller からの要求を識別し, Request Write Enable と Status Write Enable を制御することで, Window Controller への要求なのかパケット受信ステータスの書き込みなのかを周辺モジュールに通知する. 転送されるデータは要求もパケット受信ステータスも図 4.13 に従ったものである. Request Separator では OTYPE のフィールドを参照し, RVL 系命令であれば Request, それ以外のものであればパケット受信ステータスであると識別する.

Status Write Unit は, Request Separator からパケット受信ステータスの書き込み要求を受けると, LLCM にパケット受信ステータスを書き込み, User Register の Status Next Write Address を更新する. また, Status Next Write Address と Status Next Read Address の値から, LLCM のパケット受信ステータスの領域の状態をチェックする. パケット受信ステータスの領域が一杯になると busy 信号を出し, Receive Controller にこれ以上の要求を出さないように通知する.

Status Write Unit は 3 状態からなるステートマシンである. 図 5.17 に Status Write Unit の状態遷移図を示す.

ステータスの書き込みは 2 回に分けて行われる構造となっており, WRITE1 で上位 64bit を, WRITE2 で下位 64bit を書き込む. 従って, ステータス書き込み要求を受信してから, ステータス書き込みが完了するまでに 3 クロック要する. Receive Controller がステータス書き込み要求を出す間隔はこれより長いため(注 4), Status Write Unit によるステータス書き込みがパケット受信処理のボトルネックとなることはない.

5.5 ハードウェア量

本研究で実装を行った DIMMnet-2 ネットワークインタフェースコントローラのハードウェア量を Xilinx 社の合成ツールである ISE を用いて測定する.

合成結果を表 5.2 に示す. 各モジュールの合成結果は論理合成 (Synthesize) までの結果であり, 配置配線 (Place and Route) は行っていない. また, Prefetch Window へのアービタなどの周辺モジュールは個々のモジュールの合成結果にには含まれていない. ALL は Core Logic のみならず, SWIF などのモジュールをすべて含めて論理合成, 及び配置配線を行った結果である.

表 5.2 を見ると, Core Logic を構成するモジュールは Write Unit を除いて 100MHz を超えており, 目標とする動作周波数に達している. しかしながら, PC-2100 DDR-SDRAM バスのバスクロックである 133MHz に到達していないモジュールは Write Unit 以外にも存在することから, より高速なメモリバスに対応するには, 論理の見直し, もしくはホストインタフェース部との間にクロックの乗せ換えを行う FIFO を挿入するといった変更が必要になると言える. なお, Write Window, Prefetch Window は内部にレジスタが存在しないため, 動作周波数は見積もることができない.

SWIF などのすべてのロジックを含めて論理合成した場合, 動作周波数は 100MHz に達していないが, パスの遅延やロジックの配置に制約を設けて配置配線を行うことで PC-1600 DDR-SDRAM

(注 4) 最短で 8 クロック

表 5.2　DIMMnet-2 ネットワークインタフェースコントローラのハードウェア量

Module	Slices	Block RAM	clock [MHz]
Write Window	13	4	N/A
Prefetch Window	2	4	N/A
LLCM	196	32	382.190
LH Buffer	305	8	306.227
Register	869	0	291.424
Window Controller	2481	12	139.228
Receive Controller	756	0	175.180
Prefetch Unit	2234	8	112.771
Write Unit	3331	0	98.184
Status Write Unit	341	0	195.090
AMT Address Table	57	6	306.654
ALL (Synthesize)	17761/33088 (53%)	150/328 (45%)	93.595
ALL (Place and Route)	19406/33088 (58%)	150/328 (45%)	100.120 (Core Logic) 125.597 (SWIF)

合成ツール：Xilinx ISE 8.2i SP3
対象デバイス：XC2VP70-7FF1517C

の規格で動作させることが可能であると示された.

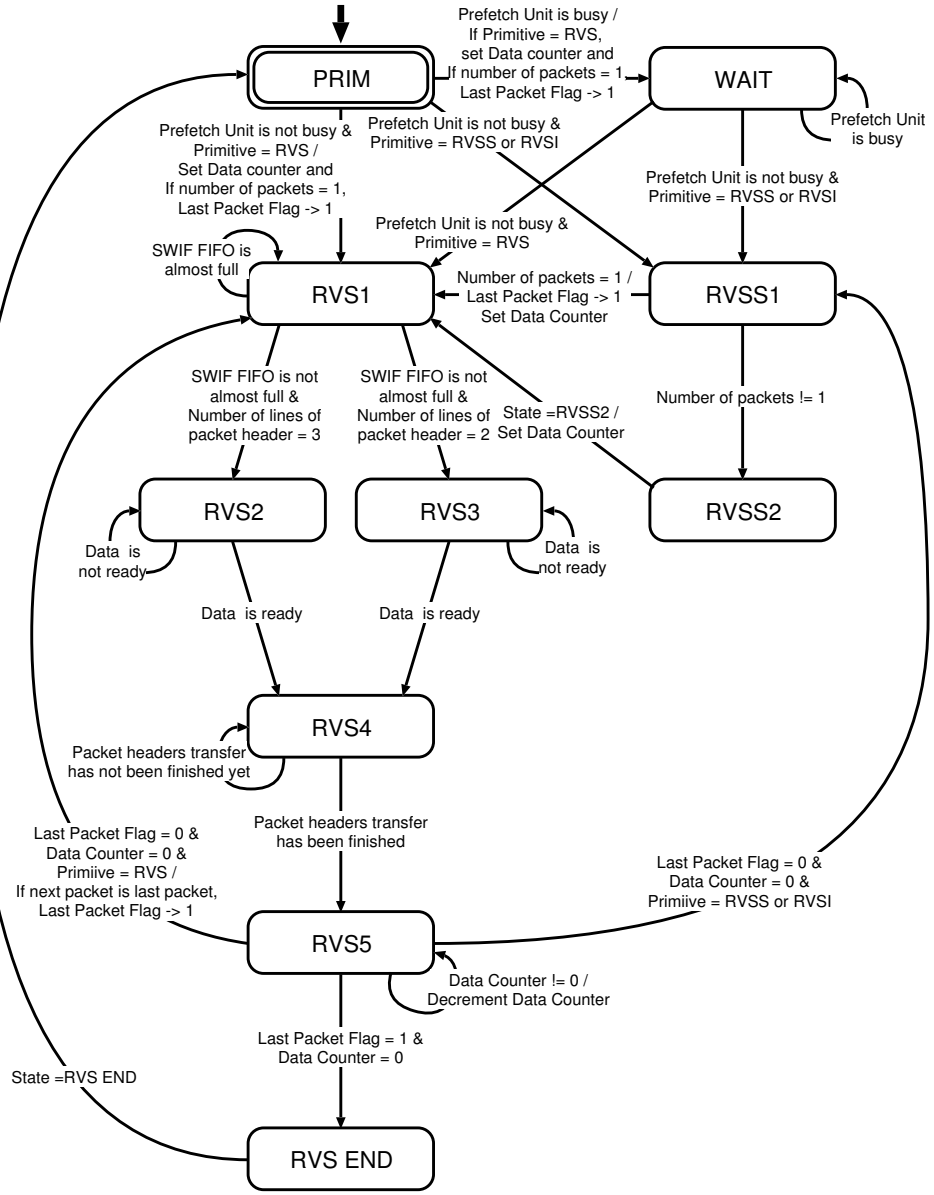


図 5.15　RVS 系プリミティブの状態遷移図

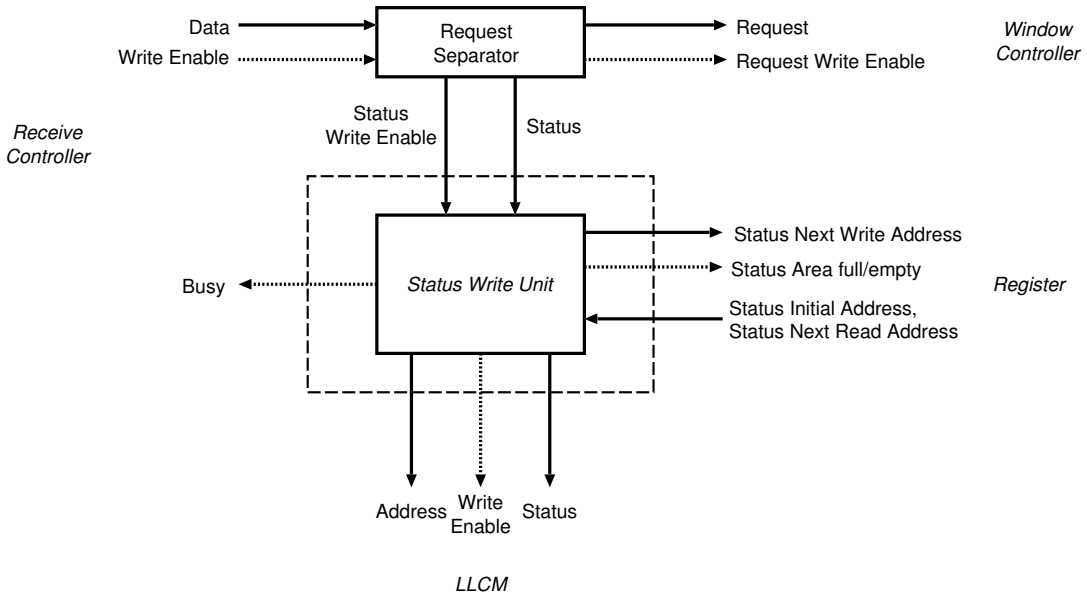


図 5.16　Status Write Unit と周辺モジュールの構造

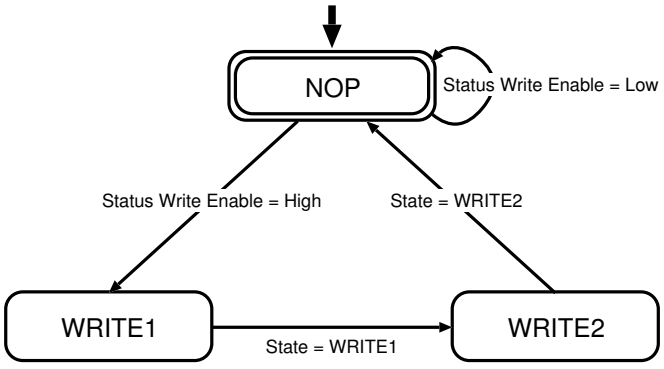


図 5.17　Status Write Unit の状態遷移図

第6章　基本通信性能の評価

本章では, 5 章で実装を行った Core Logic の基本通信性能の評価を示す. 本評価では BOTF を用いたホスト - SO-DIMM 間転送, 及び RVS を用いた SO-DIMM 間転送におけるスループットと通信遅延を測定した.

6.1　評価環境

評価環境を表 6.1 に, 概観を図 6.1 に示す. 評価環境は表 6.1 に示す PC をノードとして, これを 2 台, InfiniBand スイッチと 2m のケーブルを介して接続している.

6.2　BOTF

本節では BOTF の評価について述べる. BOTF の評価では片方向, 及び双方向のスループットとレイテンシを測定した. また, BOTF でリモートの SO-DIMM に対して書き込みを完了するまでのスループットとレイテンシを測定した.

BOTF はパケットヘッダと転送するデータから構成されるパケットイメージを Write Window に書き込み, そのパケットイメージをそのままパケットとしてネットワークに送出するという, PIO による通信処理である. パケットイメージのフォーマットは, 図 4.23 から 2bit のライン識別子を除いたものと同一である. 1 ラインを 64bit とし, パケットヘッダ部はリモート側での処理に応じて 2〜4 ライン, それ以降のラインがデータという構成である. Window 1 個当たりのサイズは 512Byte であるため, 1 回の BOTF で転送可能な最大データサイズは, ヘッダが 2 ラインの場合で 496Byte となる. 本評価ではパケットヘッダ部は 2 ラインとしている.

Write Window は内部に 4 個の Window を持っていることから, 複数の Window を利用すること

表 6.1　評価環境

Node	CPU	Pentium4 2.6GHz (2625.918MHz) L2=512KByte
	Chipset	VIA VT8751A
	Memory	PC-1600 DDR-SDRAM 512MByte DIMMnet-2 ×1
	OS	RedHat8.0 (Kernel 2.4.27)
Network	InfiniBand Switch	Voltaire ISR6000
	Switch Module	SW-6IB4C
	cable	2m
Compiler	gcc	3.3.5 (compile option: -Wall)

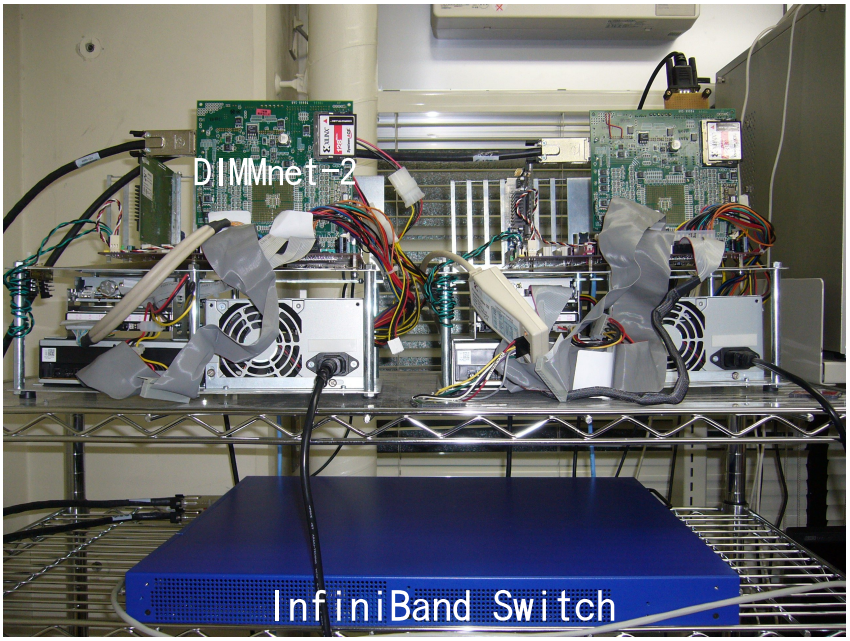


図 6.1　評価環境の概観

で複数の BOTF をオーバラップさせることが可能である. 図 6.2 は, Write Window の Window #0 から Window Controller がパケットイメージを読み出し, 同時にホストプロセッサが Window #2 にパケットイメージを書き込んでいる状況を表したものである. 本評価では BOTF で使用する Window の個数を変更して評価を行った.

6.2.1　片方向通信性能

片方向の通信性能の測定においては, 送信側のノードが BOTF でデータを転送し続ける. BOTF 要求の発行が可能になり次第, 次の BOTF 要求を発行する. データの転送先にはリモートノードの SO-DIMM を指定した. 転送データサイズを 8Byte から 1984Byte まで増加させて測定を行った. 1984Byte というサイズは BOTF を 4 回実行した場合の最大転送データサイズと同じサイズである. 使用する Window の枚数は 1, 2, 4 個とした.

評価結果のスループットを図 6.3 に, レイテンシを図 6.4 に示す.

転送データサイズが 1984Byte までの範囲での最大スループットは Window を 1 個使用した場合で 297.44MByte/s, 2 個で 463.55MByte/s, 4 個で 520.31MByte/s となった. Window を複数個使用することで BOTF の処理がオーバラップされていることが分かる. Window を 1 個使用した場合, データサイズが約 500Byte ごとにスループットの低下が見られている. 1 回の BOTF で転送可能な最大データサイズが 496Byte であり, また, 1 個の Window を要求処理部とホストで使用することによる競合が起きている. これにより, ホスト側は 496Byte ごとに要求処理部の処理が完了するまで待つことになるため, スループットの低下が起こる. さらに, ホスト側の処理と要求処理部の処理がオーバラップされないため, データサイズが大きくなっても, データサイズが 496Byte の時点のスループット以上の値は得られなかった.

使用する Window の個数を 1 個から 2 個に変更した場合のスループットの増加は大きいが, 2 個

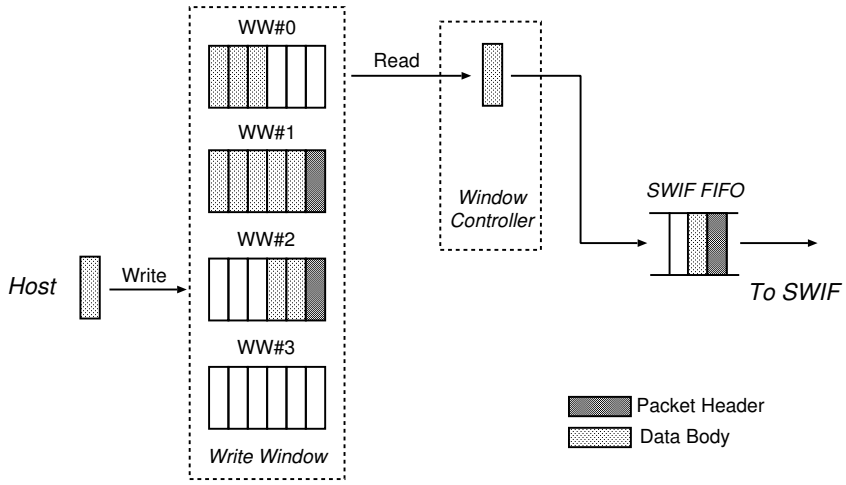


図 6.2　BOTF のオーバラップ

表 6.2　ヘッダ 16Byte, データ 496Byte 転送時のホスト側と Core Logic 側のレイテンシ

処理内容	レイテンシ [μs]
送信側でパケットイメージを書き込み, BOTF 要求を発行	0.407
送信側の Core Logic がパケットイメージを SWIF に転送	0.660

から 4 個に変更した際のスループットの増加は小さい. これはホストプロセッサ側と Core Logic 側の処理の速度がほぼ同じであることによると考えられる. ヘッダ 16Byte, データ 496Byte のパケットイメージを BOTF で送信した場合の送信側のレイテンシの内訳を調査した結果を表 6.2 に示す. 表 6.2 の Core Logic のレイテンシは RTL シミュレーションにより得た値である.

ホストプロセッサは, 次の BOTF のパケットイメージを Window に書き込む前に, その Window が Core Logic 側によって使用されていないことを確認する必要がある. Core Logic の処理がホストプロセッサに比べて大幅に遅い場合, ホストプロセッサは Core Logic 側の処理が完了するまで待つ時間が長い. このような状況では Window の個数が多いほど, その個数分の BOTF までは Core Logic の状態に関係なく, ホストプロセッサは BOTF の処理を進めることができるため, スループットの増加量は大きくなる. しかし, 実際にはホストプロセッサの処理と Core Logic 側の処理に大きな差がないため, Window の個数を 2 個から 4 個に増やしてもホストプロセッサ側が待つ時間が短く, スループットはあまり大きく増加しなかった. このことは, より高速なホストプロセッサを使用する, DIMMnet をより高速なメモリバスに装着する, ネットワークインタフェースコントローラにより高速な FPGA を搭載するなど, 双方の性能が異なってくると, 最適な Window の個数が変わってくることを示している. しかし, Window の個数を増やすと, その分だけネットワークインタフェースコントローラのハードウェア量が増えるため, ハードウェア量の増加量と性能向上の割合のトレードオフを考慮すべきである.

最小のレイテンシはヘッダ 16Byte とデータ 8Byte の計 24Byte 転送時で 0.632μs であった. RTL シミュレーションで Core Logic の処理時間を測定した結果, 0.100μs であったため, ホスト側の処理に要する時間は 0.532μs ということになる. ホスト側の処理は大きく分けて, 次のようになる.

(1) パケットイメージの書き込み

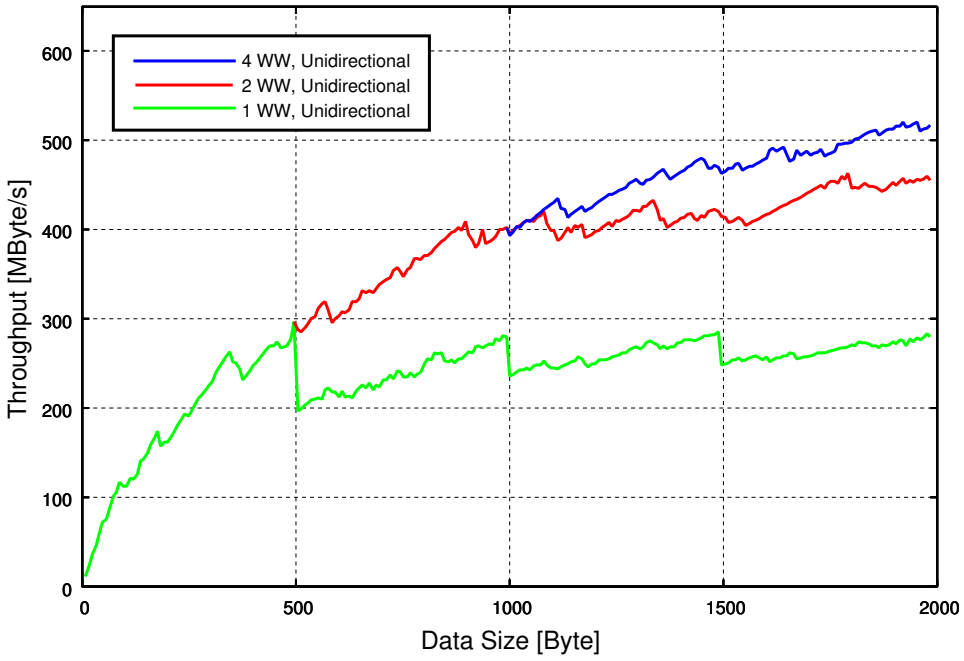


図 6.3　BOTF スループット (片方向)

(2) BOTF 要求の発行

(3) BOTF の送信処理完了検知

このうち, (3) のレイテンシは PC-1600 DDR-SDRAM メモリモジュールに対するアクセスレイテンシを測定することで求めることができる. アクセスレイテンシを測定した結果, Uncachable 領域に対するアクセスは約 0.189μs であった. そのため, (3) 以外の処理には 0.343μs 程度要することが分かる. プログラム上でこれらの処理が完了するまでのレイテンシを測定すると, (1) の処理に 0.076μs, (2) の処理に 0.060μs 要するという結果が得られた. これらの値は実際のレイテンシと大きな差があるが, どちらの値も実際にメモリモジュールにデータが書き込まれるまでを測定しているわけではなく, ホストプロセッサがチップセットにメモリへの書き込み要求を発行した時点までの測定になってしまっていることが原因と考えられる.

6.2.2　双方向通信性能

双方向の通信性能の測定では, 双方のノードが BOTF でデータを転送し続け, 各々のノードで BOTF 要求の発行が可能になり次第, 次の BOTF 要求を発行する. 転送データサイズや使用する Window の個数は片方向通信の評価と同じである. 評価結果のスループットを図 6.5 に, レイテンシを図 6.6 に示す.

転送データサイズが 1984Byte までの範囲での最大スループットは, Window を 1 個使用した場合で 556.08MByte/s, 2 個で 908.20MByte/s, 4 個で 1037.18MByte/s であった. 双方向のスループットは片方向の約 2 倍を達成しているため, Core Logic が送信処理と受信処理を並行して処理していることが分かる. 最小のレイテンシは 8Byte のデータ転送時で 0.740μs であった. BOTF は PIO 通

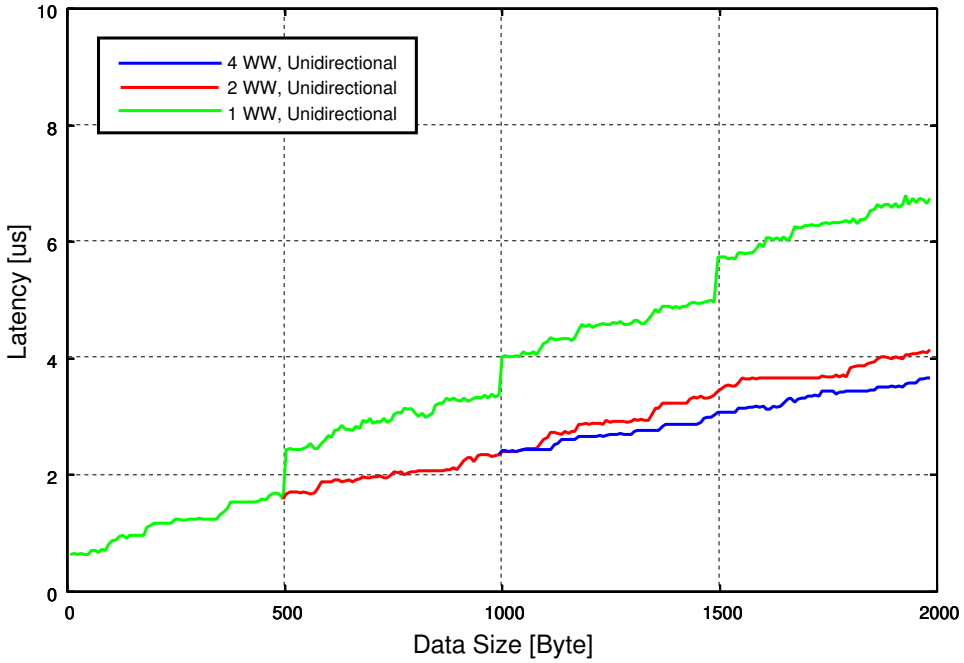


図 6.4　BOTF レイテンシ (片方向)

信機構であるにもかかわらず, 双方向で 1GByte/s 以上のスループットを実現しており, 表 2.2 に示されるインターコネクトの RDMA による通信性能に迫る性能を示している.

6.2.3　受信処理を含めた BOTF の通信性能

受信処理を含めた評価では, 送信側は BOTF でパケットを送信した後, 受信側から同じサイズのパケットを受信したことを検出するまでポーリングを続ける. 受信側は送信側からパケットを受信したことを検出すると, 同じサイズのパケットを BOTF で送信側に送出する. この評価によって, BOTF で送信したパケットが受信側の受信領域に受信され, 受信側のホストがそれを検出するまでのスループットとレイテンシを測定することができる. 転送データサイズや使用する Window の個数は片方向通信の評価と同じである. ただし, 今回は Window を 4 個使用したときのみ, データの受信先を Prefetch Window と SO-DIMM の双方を指定した.

評価結果のスループットを図 6.7 に, レイテンシを図 6.8 に示す. レイテンシは測定値を 2 で割り, RTT/2 として示している.

転送データサイズが 1984Byte までの範囲での最大スループットは Window を 1 個使用した場合で 209.71MByte/s, 2 個で 296.52MByte/s, 4 個で SO-DIMM に受信した場合で 298.95MByte/s, Prefetch Window に受信した場合で 323.22MByte/s となった. 最小のレイテンシは SO-DIMM に受信した場合で約 1.82μs, Prefetch Window で約 1.76μs であった. 表 2.2 に示されるインターコネクトと比較すると, RHiNET-2 と同等のレイテンシであるが, RHiNET-2 はスイッチを用いずに back-to-back で接続しているのに対し, DIMMnet-2 は InfiniBand スイッチを介している. InfiniBand スイッチのレイテンシは文献 [110] によると, 約 0.385μs である. そのため, 仮に back-to-back で接続したとすると, SO-DIMM に受信した場合で約 1.44μs, Prefetch Window に受信した場合で約 1.38μs となる.

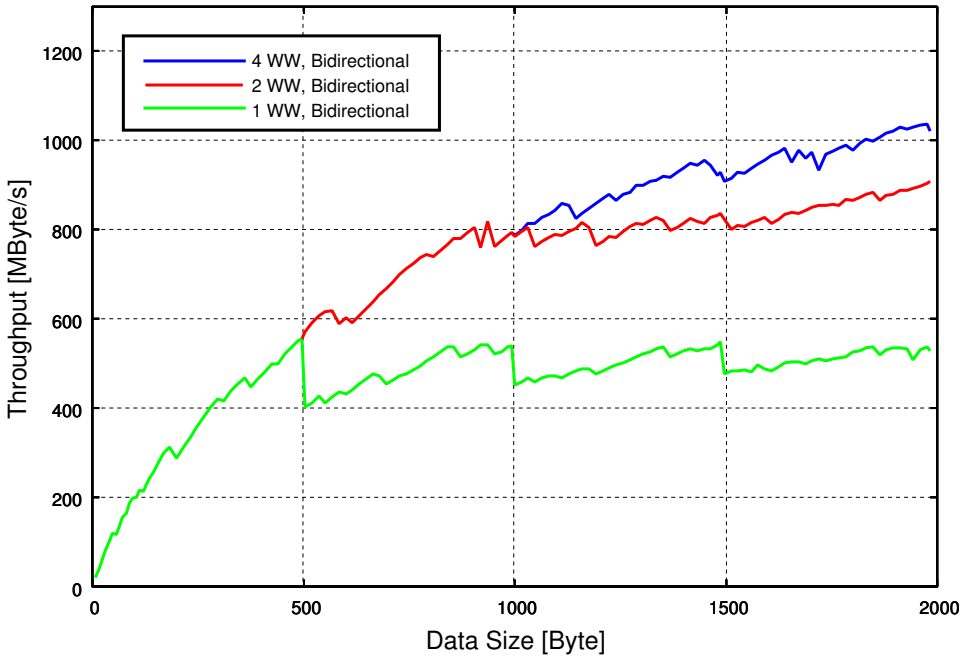


図 6.5　BOTF スループット (双方向)

DIMMnet-2 はネットワークインターフェースコントローラの動作周波数が 100MHz と, 比較的低いにもかかわらず, ほかのインターコネクトに匹敵するレイテンシを達成していると言える.

6.2.4　BOTF の最大スループット

本節で行った BOTF の各評価項目について, 転送データサイズを 2K〜64KByte まで変化させ, それぞれの通信時における最大スループットを測定した. 使用した Window の個数は 2, 及び 4 個である.

片方向通信, 及び双方向通信の結果を図 6.9 に, 受信処理を含んだ通信の結果を図 6.10 に示す.

片方向通信の最大スループットは Window を 2 個使用した場合で 521.89MByte/s, 4 個使用した場合で 631.11MByte/s となった. また, 双方向通信の場合は Window が 2 個の場合で 1049.84MByte/s, 4 個の場合で 1163.70MByte/s となった. いずれもデータサイズを増加させた際のスループットの変化は小さいため, 64KByte までにスループットが飽和していると言える. また, 約 2KByte までのデータ転送時に得られたスループットの値と最大スループットの値に大きな差は見られず, BOTF がデータサイズが小さい段階で, 最大性能に近いスループットを達成していると言え, BOTF のレイテンシの小ささが示されている.

受信処理を含めた場合の BOTF の最大スループットは Window を 2 個使用した場合で 517.22MByte/s, 4 個使用した場合で 564.91MByte/s となった. 受信処理を含めた場合は, 片方向, 双方向通信時の結果と異なり, データサイズが 2KByte までの評価に比べてスループットが大幅に増加している. これは, データサイズが増加するに従って送信される BOTF パケットの個数が増加し, 送信側の送信処理と受信側の受信処理がオーバラップしたことによると考えられる. 約 2KByte のデータ転送では BOTF によるパケットは高々4 個しか送信されないため, 通信に要する時間全体の, パケット間でのオーバラップ可能な時間の割合が低いが, パケット数が増加するに従って, この割合は高くな

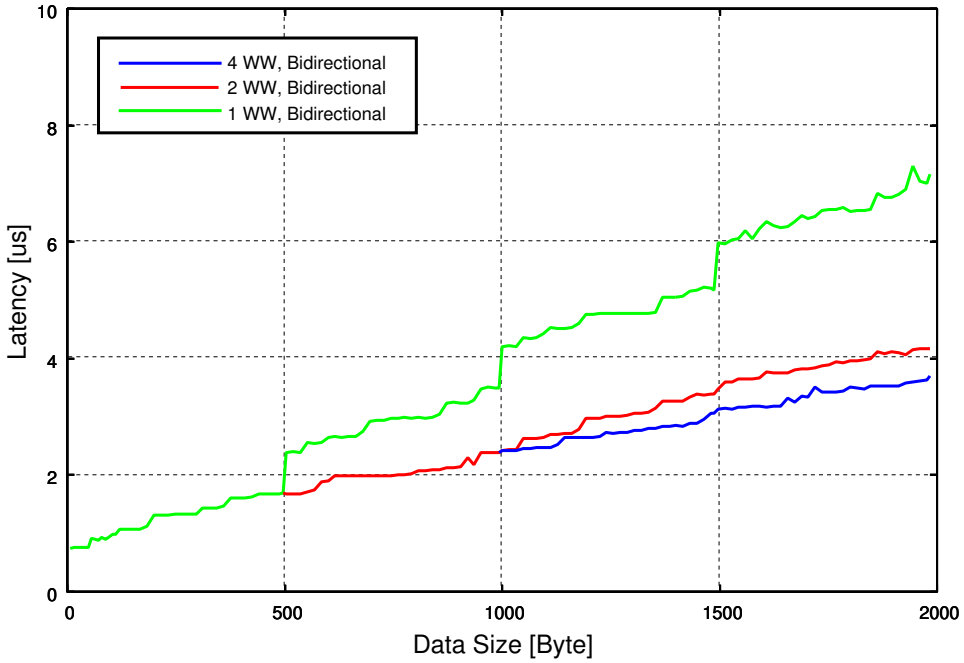


図 6.6　BOTF レイテンシ (双方向)

る. これにより, 送受信双方の処理がオーバラップされ, 最終的には片方向通信時の最大スループットに近い値が得られていると考えられる.

しかし, BOTF は PIO 通信であり, 通信処理中はホストプロセッサの負荷が高く, また, 転送データサイズが大きくなるに従ってホストプロセッサのキャッシュが転送データで占められ, ホスト側の処理に大きな影響を与えるという欠点がある. そのため, 転送データサイズが大きい場合には, 次節で評価を行う SO-DIMM 間転送を行うべきである.

6.3　SO-DIMM 間転送

本節では SO-DIMM 間でパケットの送受信を行った場合の評価について述べる. SO-DIMM 間通信の評価では片方向, 及び双方向のスループット, さらに, リモートの SO-DIMM に対して書き込みを完了するまでのスループットとレイテンシを測定した.

6.3.1　片方向, 及び双方向の通信性能

片方向の通信性能の測定においては, 送信側のノードが RVS でデータを送信し, 送信側の処理が完了するまでのスループットとレイテンシを測定した. また, 双方向の通信性能の測定においては, 双方のノードが RVS でデータを送信し続け, 各々のノードで RVS 要求の発行が可能になり次第, 次の RVS 要求を発行する. 測定結果を図 6.11 に示す.

片方向の最大スループットは 727.20MByte/s であり, DIMMnet-2 の理論最大スループットである 800MByte/s の 90.9%を達成している. half of peak throughput は 564Byte であり, 表 2.2 に示されるインターコネクトの中では低レイテンシであると言える.

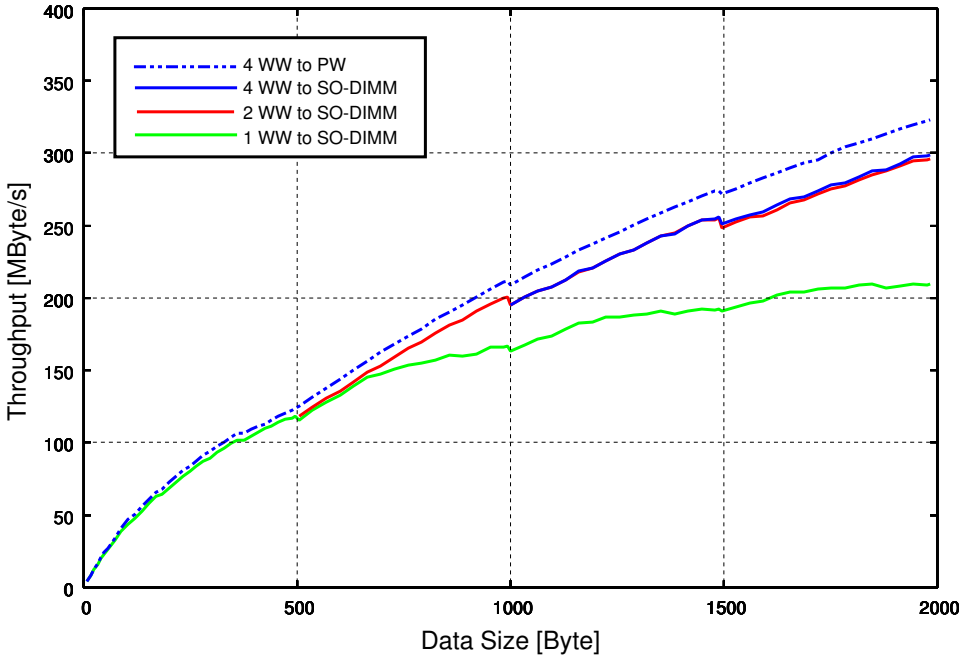


図 6.7　受信処理を含めた BOTF のスループット

双方向の最大スループットは 953.50MByte/s であった. BOTF の評価では片方向通信時の倍のスループットを示したが, SO-DIMM 間転送ではそのような結果は得られなかった. また, 最大スループットの値も Window を複数個使用した場合の BOTF に比べて低くなっている. これは, BOTF がデータの転送元と転送先がそれぞれ Write Window と SO-DIMM なのに対して, SO-DIMM 間通信では転送元と転送先が共に SO-DIMM であり, 送信処理の Prefetch Window による SO-DIMM へのアクセスと受信処理の Write Window による SO-DIMM へのアクセスが競合していることによる. しかしながら, BOTF が PIO 通信であり, 通信処理の間はホストプロセッサがビジーになるのに対し, SO-DIMM 間通信はホストプロセッサは通信要求を発行するだけであるため, ホストプロセッサの負荷は低い. また, BOTF の場合は転送データサイズが増加するに従ってホストプロセッサのキャッシュが転送データで汚染されてしまうが, SO-DIMM 間通信の場合はそのようなことは起こらないという利点がある.

SO-DIMM 間通信の最小レイテンシは, 片方向通信の場合でデータサイズが 64Byte のときに 0.81μs であり, 双方向通信の場合でデータサイズが 8〜32Byte のときに 1.09μs であった. BOTF ほどではないものの, 低いレイテンシを示していると言える.

6.3.2　受信処理を含めた SO-DIMM 間転送の通信性能

受信処理を含めた評価では, 送信側から RVS でパケットを送信し, 送信側が受信側からパケットを受信したことを検出するまでポーリングを続ける. 受信側は送信側からパケットを受信したことを検出すると, 同じサイズのパケットを RVS で送信側に送出する. この際に, 受信領域を SO-DIMM にした場合と, SO-DIMM に受信したデータをホストの主記憶にコピーした場合で測定を行った. 主記憶のコピーの際には VL で SO-DIMM から Prefetch Window に受信したデータを読み出し, それを主記憶にコピーする. この主記憶へのコピーの際に, コピー先の領域を受信したデータサイズ

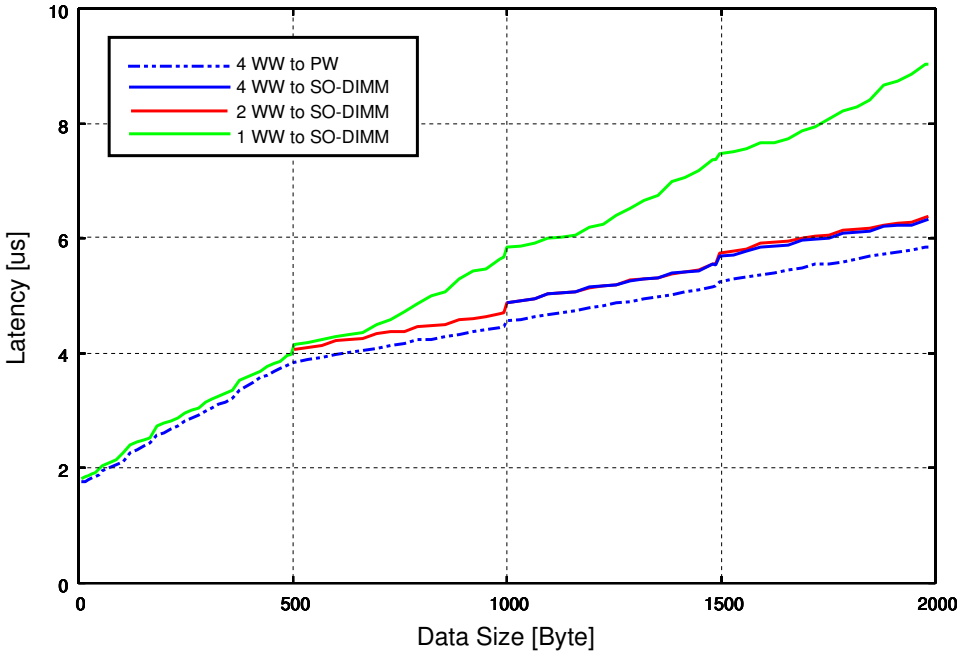


図 6.8　受信処理を含めた BOTF のレイテンシ

だけ確保した場合と, コピー先の領域を 64Byte に固定した場合で測定した. 測定結果を図 6.12 に示す.

SO-DIMM 間のデータ転送の最大スループットは 650.43MByte/s であった. それに対し, 主記憶へのコピーまで含めたスループットは 226.36MByte/s であり, SO-DIMM 間のデータ転送に比べて大幅にスループットが低下している. さらに, 転送サイズが 128KByte を超えたところから, さらにスループットの低下が見られ, 163MByte/s 程度までスループット落ちている. しかし, 主記憶のコピー先の領域を固定すると, 最大スループットが 236.37MByte/s であり, また, 転送サイズが大きくなってもスループットの低下は見られていない. コピー先の領域は Write Back 領域であることから, 受信領域が大きくなるに従ってホストプロセッサのキャッシュを圧迫し, キャッシュヒット率が低下したのが原因と考えられる. また, Prefetch Window の総サイズは 2KByte であるため, SO-DIMM から読み出すサイズが大きくなると, キャッシュフラッシュ命令とプリフェッチ命令を繰り返すことになり, プリフェッチが間に合わないものと考えられる. SO-DIMM からデータを読み出し, それを使用した演算の間に, 次の演算で使用するデータを SO-DIMM から読み出すというのが理想的な状況であるが, 基本通信性能の評価のように, 通信以外の処理が行われないベンチマークでは高い性能が得られないと考えられる.

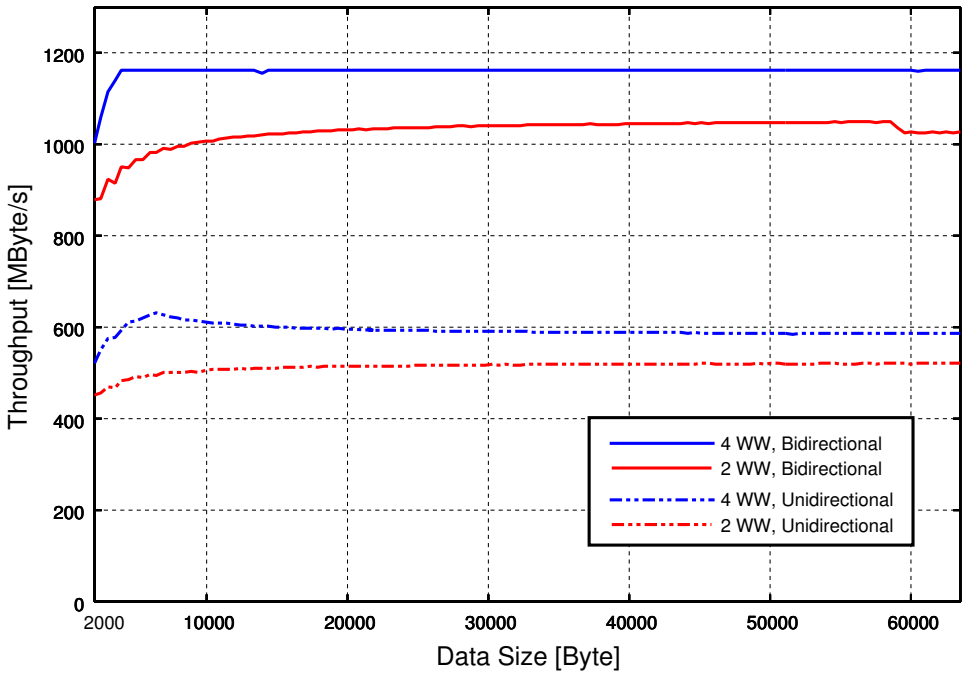


図 6.9　BOTF スループット (データサイズ：2KByte 以上)

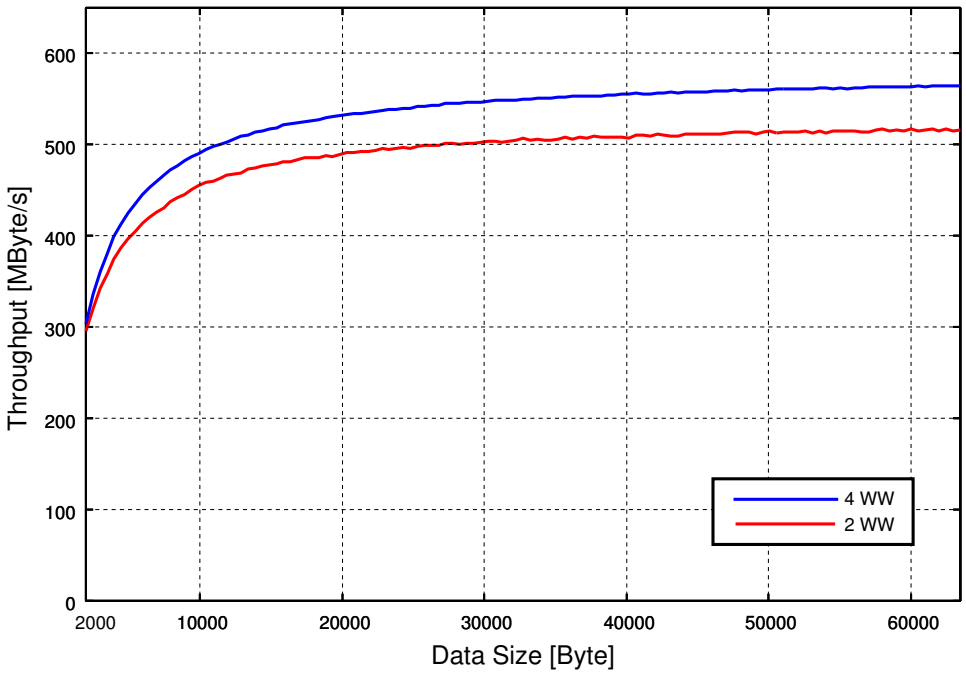


図 6.10　受信処理を含めた BOTF のスループット (データサイズ：2KByte 以上)

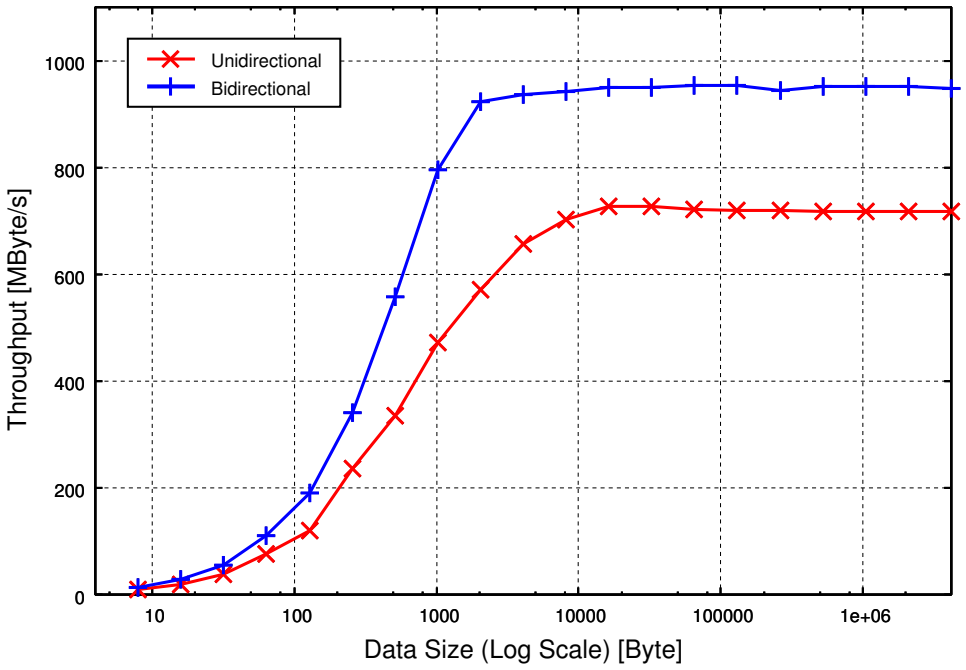


図 6.11　SO-DIMM 間通信 スループット

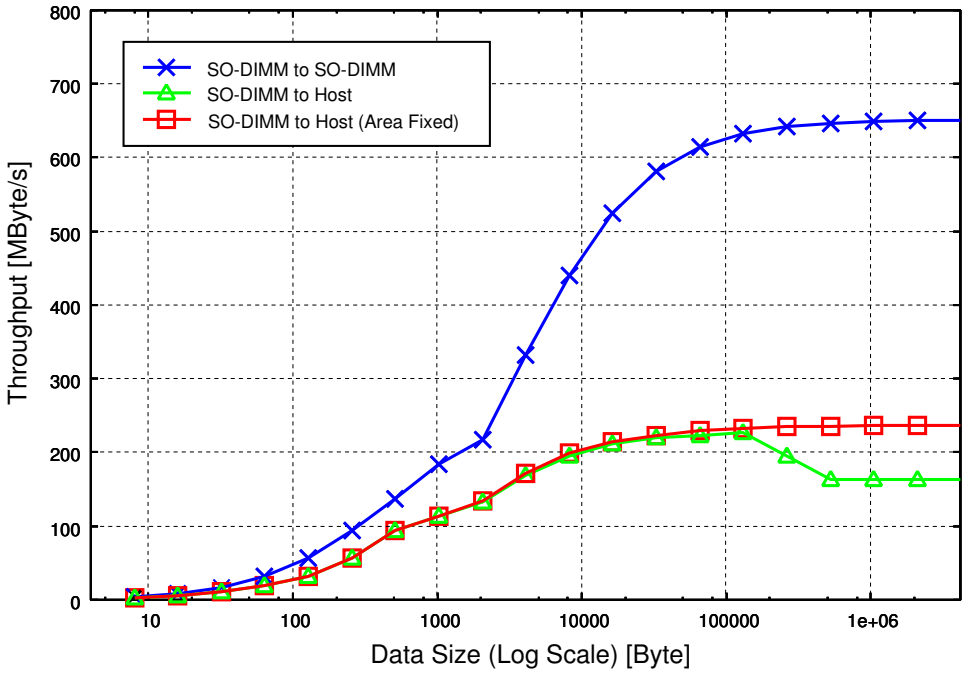


図 6.12　受信処理を含めた SO-DIMM 間通信

第7章　メッセージ通信支援機構

本章ではメッセージ通信支援機構である IPUSH と LHS について述べる. どちらもメッセージ通信時に発生する処理を実行するのではなく, ホストプロセッサで処理を効率良く行えるようにするためのメッセージ “受信” 機構である. これらの機構は DIMMnet-2 に依存したものではないが, ネットワークインタフェースコントローラに FPGA を用いており, コントローラの構造の変更が容易であることから DIMMnet-2 試作基板を実装対象とした. これらの機構は, DIMMnet-2 ではプリミティブとしてホスト側から使用することができる.

7.1　IPUSH (Indirect PUSH)

RDMA を用いて実現されたメッセージ通信における問題点は 2.3.1 節で述べた. 本研究で提案する IPUSH 機構は, この問題点を解決することを目的とする.

IPUSH 機構は受信側のネットワークインタフェースコントローラがメッセージの受信先アドレスを指定し, メッセージの受信, 及び受信領域のアドレス管理を行う. さらに受信したメッセージ単位で, またはパケット単位でメッセージ受信ステータスを単一のバッファに格納することにより, メッセージの到着順序の検知手法を提供する. このメッセージ受信ステータスには送信側のプロセス識別子やデータサイズといった, 受信したメッセージに関する情報が含まれており, 受信ステータスをメッセージ単位で格納するかパケット単位で格納するかは送信側がメッセージの送信要求を発行する際に指定可能としている.

また, IPUSH 機構では受信領域を複数の送信プロセスで共有することが可能な構造となっている. これにより, メッセージの受信頻度の高いプロセスには個別のメッセージ受信領域を確保し, 頻度の低いプロセスにはそれらのプロセス全体でメッセージ受信領域を共有することで, メッセージ受信のための領域のサイズを削減することが可能となる. 以降, 本章では IPUSH に対し, 通常の RDMA write(注 1)を PUSH と呼称する.

7.1.1　先行研究との差異

IPUSH 機構と同様の機構として 2.3.1 節ではリモート間接書き込み, 及び VPUSH について述べた. これらと IPUSH 機構の差としては次のことが挙げられる.

- リモート間接書き込みと VPUSH では, 送信プロセスごとにメッセージ受信バッファを分けた際のメッセージの到着順序の検知手法を提供していない.
- 複数プロセスでメッセージ受信バッファを共有した際に, 単一のメッセージが複数パケットに分かれた場合, プロセス間でインタリーブして受信される可能性がある. しかし, これらの機構では, このような場合のメッセージ読み出しに対応していない.

(注 1)DIMMnet-2 の場合は RVS

表 7.1　MPI レベルのレイテンシの内訳 (単位：μs)

	QsNET II QM500	Myrinet-2000 M3F2-PCIXE
Host	0.8	0.3
Network Interface	0.8	2.8
Network Switch	0.1	0.4
Total	1.7	3.5

IPUSH 機構ではメッセージ受信ステータスをメッセージ単位またはパケット単位に単一のバッファに格納することで, メッセージまたはパケットの受信順序を検知する手段をホスト側に提供する. 複数プロセスで受信バッファを共有する場合には, パケット単位でメッセージ受信ステータスを格納するようにすることで, 受信メッセージがプロセス間でインタリーブされた場合にもメッセージを正常に読み出すことができる.

また, IPUSH 機構は受信バッファを送信プロセスごとに独立して確保することが可能である. このバッファを unexpected message queue として用いることで, 各キューに特定のプロセスからのメッセージのみが格納されるようになり, キューの探索範囲の削減が可能となる. この点が ALPU と異なる.

7.1.2　IPUSH 機構の設計

2.3.1 節で触れたリモート間接書き込みと VPUSH において, 従来の RDMA に比べて大幅な性能低下を起こしていたことから, メッセージの受信処理をネットワークインタフェースコントローラで高速に行う場合にはネットワークインタフェースコントローラに高い性能が求められると言える. この場合, 高い性能を持つ受信機構の実現方法として, Myrinet のようにプログラマブルな高性能ネットワークインタフェースコントローラ上でソフトウェアで実行するか, ハードワイヤードで専用の受信処理部をネットワークインタフェースコントローラ上に搭載する 2 通りの手法が挙げられる.

ソフトウェアを用いた処理は柔軟性を持つ一方, 一般に, 同等の機能をハードワイヤードで実装した場合に比べ, 処理に要するレイテンシが大きくなる. このことは文献 [66] から読み取れる. 文献 [66] によると, Myrinet-2000[41] の MPI レベルでの最小の送信レイテンシは約 3.5μs であり, 対して, QsNET II では 1.7μs である. このレイテンシの内訳を表 7.1 に示す.

表 7.1 から, レイテンシの差が主にネットワークインタフェースによるものであることが分かる. Myrinet のネットワークインタフェースコントローラ LANai-XP が RISC プロセッサであり, この上でソフトウェアを用いて通信処理を行っているのに対し, QsNET II の Elan4 では RISC プロセッサを搭載しているものの, 基本的な通信処理はハードウェアで実装されていることが影響していると考えられる.

そのため, IPUSH 機構はアドレス管理, メッセージの受信処理などをすべてネットワークインタフェースコントローラ上にハードワイヤードロジックで搭載する. これにより, 送信側は RDMA でメッセージを送信し, 受信側は IPUSH 機構で受信することで, 低レイテンシなメッセージ通信を実現することが可能となる.

メッセージの受信領域はプロセスごとに個別の領域をリングバッファとして確保可能とする. し

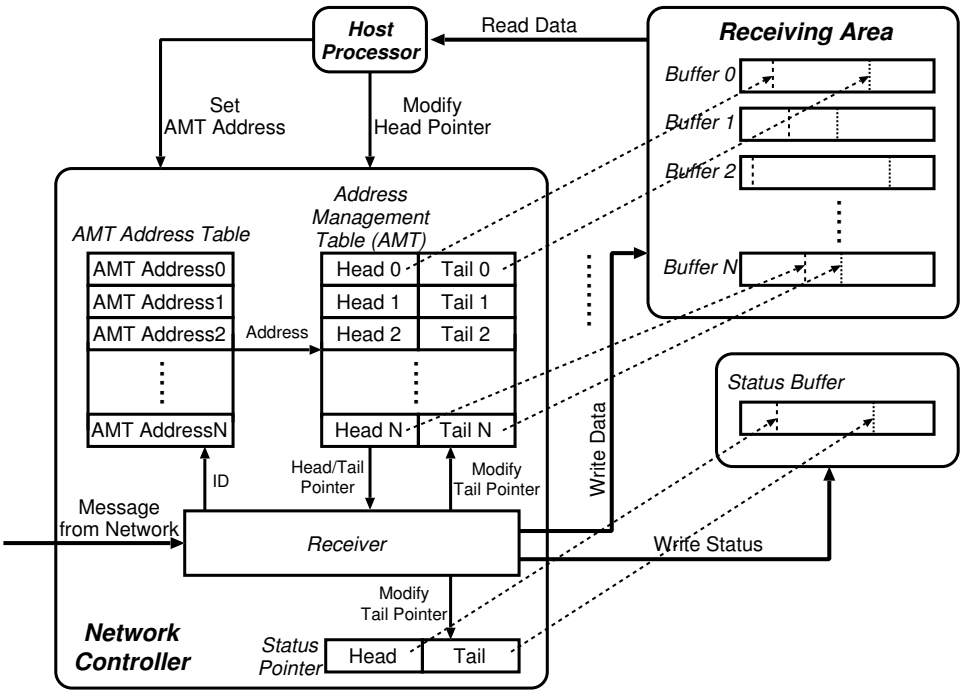


図 7.1 IPUSH 機構

かしながら, 全プロセスに対して異なるメッセージ受信領域(注 2)を確保すると受信領域を確保する領域(注 3)が圧迫されるため, スケーラビリティに乏しくなる. そこで IPUSH 機構では複数の送信プロセスでバッファを共有可能とすることで, この問題を解決する.

7.1.3 IPUSH 機構の概観

IPUSH 機構の概観を図 7.1 に示す. IPUSH 機構の機能を満たすために求められるのは以下の事柄である.

- 送信プロセスごとに独立したメッセージ受信領域の制御
- メッセージまたはパケット受信時の受信ステータスの制御

送信プロセスごとに独立したメッセージ受信領域を確保し, アドレス管理を行うために, IPUSH 機構ではネットワークインタフェースコントローラのメッセージ受信処理部に表 7.2 に示す 2 つのテーブルを追加する.

一般に並列システムにおいては, 各ノードで実行される並列処理のプロセスに固有の ID が振られている. IPUSH 機構ではこの ID を利用して, 表 7.2 に示したテーブルにアクセスする. その手順を以下に示す.

1. ネットワークインタフェースコントローラ内で受信処理を行う Receiver は, メッセージ受信時に送信プロセスの ID を使用して AMT Address Table にアクセスする.

(注 2)ページアウトされない領域 (ピンダウン領域)
(注 3)ホストの主記憶やネットワークインタフェース上のメモリ

表 7.2　IPUSH 機構に追加するテーブル

テーブル名	機能
Address Management Table (AMT)	各プロセスの受信領域の Head ポインタと Tail ポインタを管理するテーブル
AMT Address Table	AMT にアクセスするためのアドレスを管理するテーブル

2. AMT Address Table から出力された値を AMT へのアドレスとし, その ID に対応したプロセスの Head ポインタと Tail ポインタを得る.

3. Head ポインタと Tail ポインタから, 受信したメッセージのサイズ以上に受信領域に空きがあるかチェックする.

4. 受信領域に空きがある場合は受信処理を継続する.

受信領域に空きがない場合は以下のいずれかの手法を取ることでメッセージの受信がなされることを保証する必要がある.

- 受信領域が空くまで処理を停止し, 受信側のホストに受信領域が一杯であることを通知する. その後, ホストがメッセージの読み出しを行い, 受信領域に空きを作ることでメッセージの受信処理を継続する.
- メッセージを破棄し, 送信元にメッセージの再送を要求する.

IPUSH 機構を既存のインターコネクトに搭載する場合は, そのインターコネクトで採用されている (パケットが受信できない場合の) エラー処理をそのまま利用することで, この処理を実現することができる. 例えば, DIMMnet-1 や RHiNET-2 のネットワークインタフェースコントローラである Martini では, メッセージが受信できない場合には NACK が送信元に返信される.

受信可能な場合はこのポインタに従って, メッセージを受信領域に書き込む. 書き込み完了後は AMT の Tail ポインタを更新し, 処理を完了する. このテーブルはホスト側からも参照可能であり, 受信したメッセージを別のバッファにコピーした場合などに Head ポインタの更新を行う.

AMT Address Table をネットワークインタフェースコントローラ内部に搭載することで, 送信側がネットワークインタフェースコントローラに対してメッセージ送信要求を発行する際に, 受信先のアドレスを指定する必要がなくなり, 要求発行に要するレイテンシが削減されると考えられる. また, AMT Address Table を用いることで, 受信側においてもメッセージの読み出し後の Head ポインタの更新処理などをネットワークインタフェースコントローラ上で行うことが容易となる. しかしながら, AMT Address Table のエントリ数はそのインターコネクトがサポートする最大プロセス数が大きくなるに従って増加するため, このような場合にはハードウェアコストが増大する. そこで, 実装するハードウェアコストによっては AMT Address Table をネットワークインタフェースコントローラ内部に搭載せずに IPUSH 機構を実現するという選択もありうる. この場合は, 送信側で受信側の AMT のアドレスを指定してメッセージを送信する必要がある (図 7.2).

AMT Address Table のエントリは各プロセス ID に個別に対応している. 図 7.1 では AMT Address0 がプロセス ID=0, AMT Address1 がプロセス ID=1 ･･･ というように対応する. 両テーブルの初期値は並列プロセス起動時にホストから書き込む. その際に AMT Address Table の各エントリに書

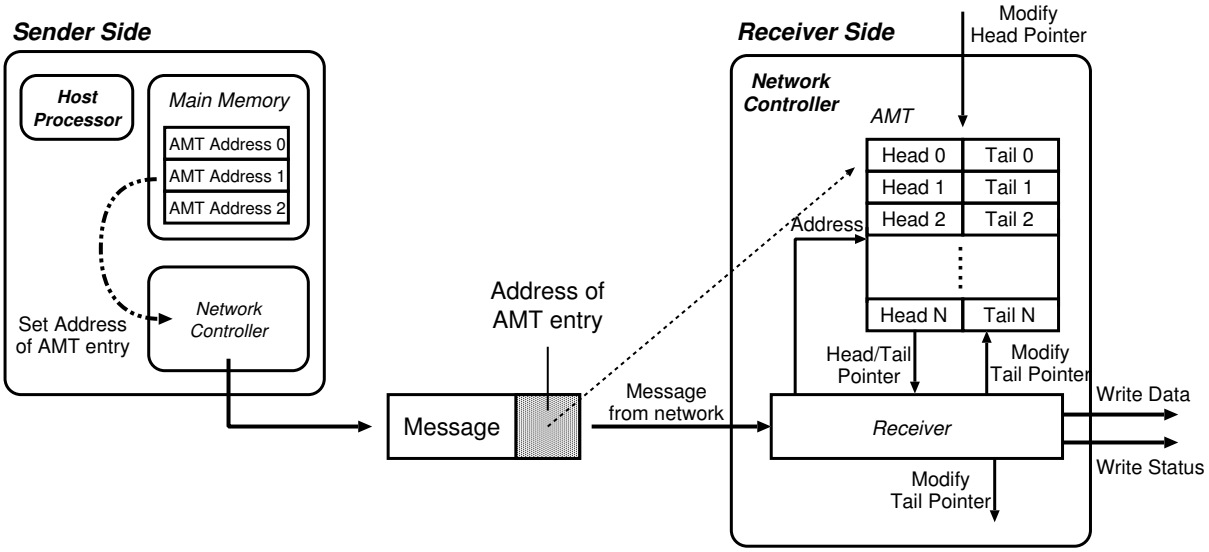


図 7.2 AMT Address Table を用いない IPUSH 機構

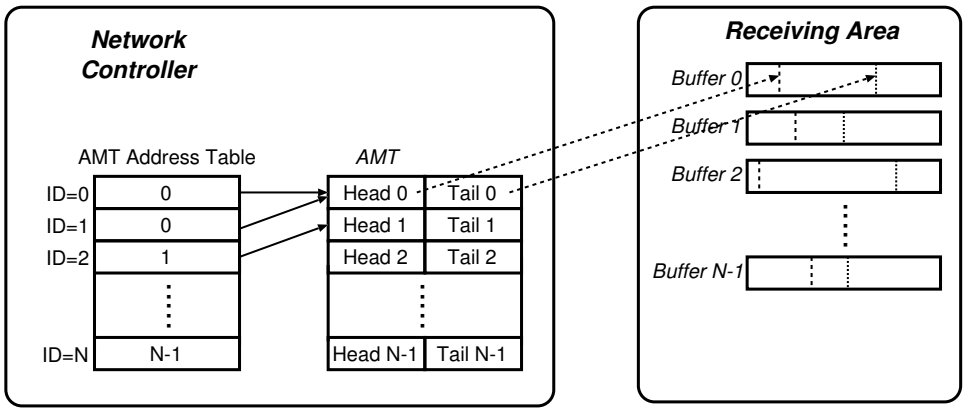


図 7.3 AMT Address Table の設定例

き込む値を同じ値にすると, AMT から読み出されるポインタは複数のプロセスで同一のものになる. このようにすることで, 例えばプロセス ID=0〜9 で単一の受信バッファを, プロセス ID=10〜19 で単一の受信バッファを使用するというような設定が可能である. 図 7.3 は, ID=0 と 1 のプロセスが同じ AMT のエントリを参照するように設定した例である. このような設定を行うと, ID=0 と 1 のプロセスから受信されたメッセージは同一の受信バッファに格納されるようになる.

上記のテーブルに加え, 受信したメッセージを受信した順番に読み出すための手段を提供することも必要である. そこで, パケット単位, またはメッセージ単位に受信ステータスを格納する領域をリングバッファとして主記憶に確保する. このリングバッファの Head ポインタと Tail ポインタはネットワークインタフェースコントローラ内に保存される. ネットワークインタフェースコントローラはステータス領域に送信元 ID, メッセージサイズなどの情報を書き込み, Tail ポインタを更新する. ホストはこのリングバッファの Tail ポインタが更新されていることを検知することで, メッセージが受信されていることを検出し, ステータスを読み出すことによってメッセージの

読み出しに必要な情報の取得を行う. ホストからはステータスを読み出した後に受信ステータスの Head ポインタを変更する.

複数プロセスで共通のバッファを使用するように AMT Address Table を設定した際に, メッセージが複数パケットに分割されて受信された場合, 受信バッファを共有しているプロセス間でメッセージがインタリーブされて受信される可能性がある. そのため, IPUSH 機構ではメッセージ単位だけでなく, パケット単位でも受信ステータスを格納できるようにしている. この場合, もし, 受信ステータスがメッセージ単位で格納されていたとすると, インタリーブされて受信されたパケットのどこからどこまでが同一プロセスからのメッセージを構成するのかを識別することができなくなり, メッセージを正常に受信することができなくなる.

PM[44] では, 複数の通信相手からのメッセージが単一の受信バッファに到着順に格納される. そのため, IPUSH 機構のような受信機構をハードウェアで持たない場合, ネットワークインタフェースコントローラ上のプロセッサを利用したソフトウェア処理を行うことによって受信先アドレスを制御するか, PM/RHiNET の実装 [37] のように, 送信元のプロセスごとに独立したバッファを確保する必要がある. 前者の場合はオーバヘッドが大きく, 後者の場合はスケーラビリティに乏しい上に, メッセージを到着順に読み出すことができなくなる. そのため, 受信ステータスの管理機構を持つ IPUSH 機構は PM を実装する際にも有利となると考えられる.

プロセスごとに独立したバッファを確保している場合には, 受信メッセージのインタリーブは起きないため, メッセージ単位で受信ステータスを格納しても問題は起こらず, また, 格納される受信ステータスの数が削減されるために, バッファを探索するレイテンシを抑えることが可能となる. さらに, 受信したメッセージのマッチングの際にも, 各送信プロセスごとにバッファが独立しているために, 探索するバッファの範囲を削減することができ, 文献 [90] で述べられている, アプリケーションによっては unexpected message queue や posted receive queue の探索範囲が大きくなるという問題も解決可能である.

パケット単位で受信ステータスを格納するか, メッセージ単位で格納するかはパケットヘッダに 1bit のフラグを用意し, 送信側がメッセージ送信要求時にそのフラグを指定することで容易に選択可能となる.

7.1.4 メッセージ受信領域の削減

一般に, バリア同期などの集団通信を除いて, あるプロセスがすべてのプロセスと通信を行うアプリケーションは稀であり, 一部のプロセスとのみ通信を行うアプリケーションも存在する. 例えば, NAS Parallel Benchmarks の IS, FT では集団通信を主に行うが, CG, MG では一部のプロセスとしか通信を行わない. 従って, AMT Address Table の設定により, 通信頻度の高いプロセスに対しては個別の受信バッファを確保し, それ以外のプロセスに対しては共通の受信バッファを 1 つ確保し, そこに IPUSH 機構を用いて受信することで, 受信領域全体のサイズを抑えることが可能となる. このように IPUSH 機構はアプリケーションに応じた設定をすることで受信領域の使用量を柔軟に変えることができる.

実際に AMT Address Table の設定によってどの程度受信領域を削減できるかを, NAS Parallel Benchmarks のトレースデータを元に検討する. アプリケーションには Class A の MG, CG, IS, LU, SP, BT を選択し, プロセス数は 64 プロセスとする. IPUSH 機構を用いない場合には各プロセスに個別の受信バッファを確保するものとし, IPUSH 機構を用いる場合には通信を行うプロセスに対しては個別の受信バッファを確保し, それ以外のプロセスに対しては共通の受信バッファを 1 つ確

表 7.3　IPUSH 機構における受信領域削減効果

アプリケーション	通信プロセス数	総受信領域サイズ	比率
MG	9	(9+1)×N	23.4%
CG	4	(4+1)×N	7.8%
IS	1	(1+1)×N	1.6%
LU	4	(4+1)×N	7.8%
SP	6	(6+1)×N	10.9%
BT	6	(6+1)×N	10.9%

保することとする. 従って, IPUSH 機構を用いない場合には 64×N Byte の受信領域を, IPUSH 機構を用いる場合には (通信を行うプロセスの数+1)×N Byte の受信領域を確保することになる. N はリングバッファ1 つ当たりの容量である.

検討結果を表 7.3 に示す. 表 7.3 の各項目はそれぞれアプリケーション, 通信相手となるプロセスが最も多いプロセスの通信プロセス数, 必要となる受信領域の総サイズ, PUSH の受信領域使用サイズを 100%としたときの IPUSH の使用領域の比率を示す.

表 7.3 から, 各アプリケーションにおいて大幅に使用領域を削減可能であると言える. 従って, AMT Address Table をアプリケーションに適した設定にすることにより, 受信領域の使用量を抑えることが可能と考えられる. MG は一部のプロセスが 9 プロセスと通信を行うが, 多くのプロセスは 6 プロセスと通信を行うため, このようなプロセスでは表 7.3 に示された値よりも受信領域の使用量はさらに少なくなる. IS では各プロセスの通信を行うプロセス数が 1 となっているが, IS はプログラムの実行時間測定範囲内において MPI_Alltoall 関数や MPI_Reduce 関数による集団通信しか行わないため, 通信プロセス数はこれらの関数の実装に依存する. しかしながら, 仮にすべてのプロセスと通信を行うような MPI の実装でも, 複数のプロセスで 1 個のリングバッファを利用するような設定を行うことで受信領域を削減することは可能である.

一般のアプリケーションにおいても, 海洋や大気のシミュレーションのように問題を格子状に区切り, 各格子点をプロセスに割り当て, 隣接する格子点を割り当てられたプロセスとのみステップ毎に通信を行うようなアプリケーションの場合は通信相手が静的に定まるために, 本手法を用いてバッファの使用量を抑えることが可能と考えられる. また, 同一のアプリケーションをパラメータを変更して複数回に渡り実行する場合には, 初回実行時にトレースデータを取得し, その結果を次回以降の AMT Address Table の設定に反映することで本手法を適用することが可能と思われる.

トレースデータの取得が難しい場合には, 複数ノードで同一のバッファを共有することで, ある程度はバッファの使用量を抑えることが可能となると考えられるが, この場合はバッファ使用量の削減効果は限定される.

7.1.5　IPUSH 機構の DIMMnet-2 への実装

DIMMnet-2 ネットワークインタフェースコントローラの詳細は 5 章で述べた. IPUSH 機構は受信処理を行うブロックである Receive Controller, Write Unit, LLCM に対して変更を加えることで実現する. AMT はホストからもコントローラからも値を更新するため, LLCM 上に設ける. 一方で, AMT Address Table は一度ホストから設定が完了すれば, そのプロセスの起動中は書き換えられる頻度は低いと考えられるため, コントローラ内部に設け, ホストからは System Register を介し

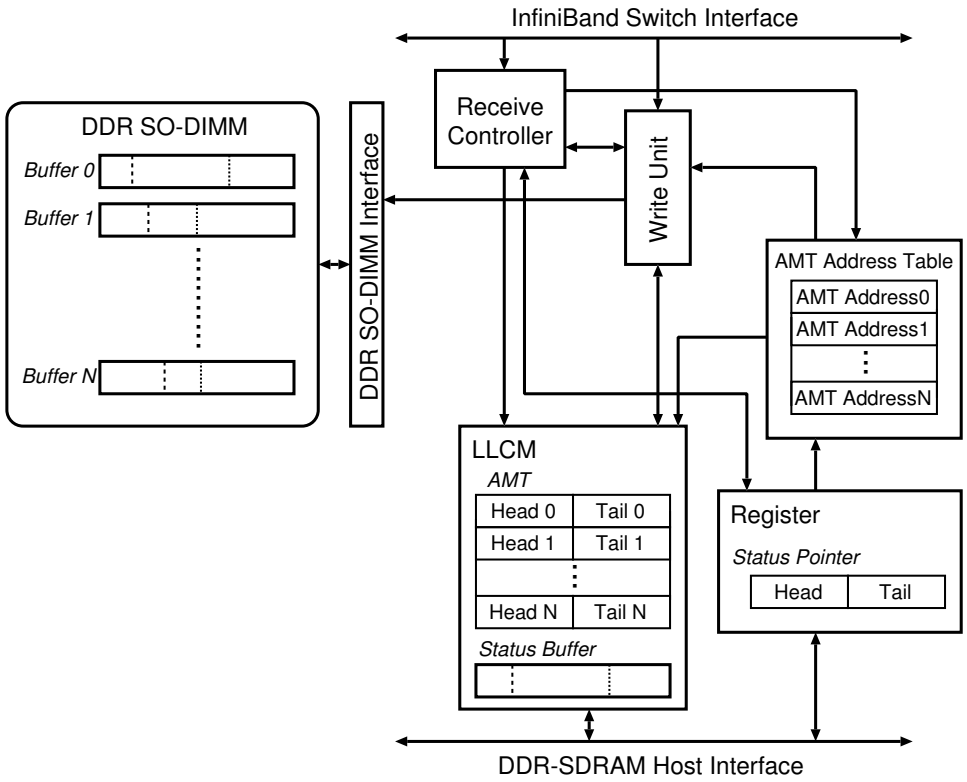


図 7.4 IPUSH 機構の実装

て設定する. DIMMnet-2 ネットワークインタフェースコントローラはパケット受信ステータスを扱う機能を持っており, これをそのまま IPUSH 機構のメッセージ受信ステータスに使用する. 受信ステータスの格納先は LLCM 上に確保し, Head ポインタと Tail ポインタは User Register 上に確保する. これらの初期値設定は並列プロセス起動時に行う. DIMMnet-2 ではメッセージの主な受信先は SO-DIMM であるため, 各プロセスに対するメッセージ受信用のリングバッファは SO-DIMM 上に確保する. これらのブロック間の接続図を図 7.4 に示す.

IPUSH 機構によりパケットを受信する際の処理の流れを以下に示す.

1. SWIF がパケットを受信したことを Receive Controller に通知する
2. 通知を受けて, Receive Controller がパケットのヘッダ部分を読み出す
3. AMT Address Table に対し, 送信元のプロセス ID でアクセスする.
4. パケットヘッダを解析し, Write Unit に転送データサイズなどを通知する
5. - AMT Address Table の出力が LLCM と Write Unit に転送される
   - LLCM は AMT Address Table の出力に対応した AMT のエントリを Write Unit に転送する
6. - Write Unit が AMT からの出力を元に SO-DIMM に対してデータを書き込む
   - Head ポインタと Tail ポインタから受信領域に空きがないと検知された場合は, 空きができるまで待機状態となる

表 7.4　PUSH パケット (24Byte) の受信処理の詳細

処理内容	クロック数
(RC) SWIF にパケットが到着したことを検出	0
(RC) SWIF へパケット (ヘッダ部) の読み出し要求	+1
(RC) 1st Header 受信	+1
(RC) 2nd Header 受信	+1
(RC) 1st Header を解析し, Write Unit を起動	+1
(WU) SO-DIMM I/F に渡すサイズ, アドレスの計算	+2
(WU) SWIF へパケット (データ部) の読み出し要求	+1
(WU) SWIF からデータ部の受信	+1
(WU) SO-DIMM へのデータ書き込み権要求	+1
(WU) SO-DIMM へのデータ書き込み権取得	+1
(WU) SO-DIMM へのデータの書き込み	+6
(RC) パケット受信ステータスの書き込み	+4
合計	20

7. SO-DIMM への書き込み終了後, Write Unit は AMT に対して Tail ポインタの更新を行う
8. Receive Controller が LLCM に対してパケット受信ステータスを書き込み, Status Pointer の Tail ポインタを更新する

これに対し, PUSH によるパケットを SO-DIMM に受信する処理の流れは以下の通りである.

1. SWIF がパケットを受信したことを Receive Controller に通知する
2. 通知を受けて, Receive Controller がパケットのヘッダ部分を受け取り, 解析する
3. Receive Controller がパケットのデータ部の受信先, サイズなどを Write Unit に通知する
4. Write Unit が SWIF からパケットのデータ部を受け取り, SO-DIMM に書き込む
5. SO-DIMM への書き込み終了後, Receive Controller が LLCM にパケット受信ステータスを書き込む

表 7.4, 表 7.5 に PUSH と IPUSH のそれぞれでメッセージを受信した際の処理の詳細を示す. これらの表中の (RC), (WU) はそれぞれ, Receive Controller, Write Unit の処理であることを示す.

表 7.4 に示されるように, DIMMnet-2 において, PUSH パケットを受信するのに要するクロック数は 20 クロックであり, DIMMnet-2 のネットワークインタフェースコントローラの Core Logic は 100MHz で動作するため, 0.2μs の処理時間となる. 一方, 表 7.5 に示されるように, IPUSH パケットの処理に要するクロック数は 24 クロックとなり, PUSH よりも 4 クロックの増加となっている. これは 0.24μs の処理時間であり, PUSH に比べて 0.04μs の処理時間増加となる.

この通信では, 受信側のホストは都合のよいときにパケット受信ステータスの領域を読み出すことでパケットの受信を検知する. しかしながら, 実際に受信側の都合の良いときのみにメッセージの読み出しを行った場合, メッセージの受信領域が足りない状況が起こり得る. 特に PCI-X バスなどに装着される一般的なネットワークインタフェースにおいて, メッセージの受信領域を主記憶上

表 7.5　IPUSH パケット (24Byte) の受信処理の詳細

処理内容	クロック数
(RC) SWIF にパケットが到着したことを検出	0
(RC) SWIF へパケット (ヘッダ部) の読み出し要求	+1
(RC) 1st Header 受信 & AMT Address Table にアクセス	+1
(RC) 2nd Header 受信	+1
(RC) 1st Header を解析し, Write Unit を起動	+1
(WU) LLCM から Head と Tail を取得	+1
(WU) 受信領域の始端を計算	+1
(WU) 受信領域の終端を計算	+1
(WU) 受信領域に空きがあるかどうか判定	+1
(WU) SO-DIMM I/F に渡すサイズ, アドレスの計算	+2
(WU) SWIF へパケット (データ部) の読み出し要求	+1
(WU) SWIF からデータ部の受信	+1
(WU) SO-DIMM へのデータ書き込み権要求	+1
(WU) SO-DIMM へのデータ書き込み権取得	+1
(WU) SO-DIMM へのデータの書き込み	+6
(RC) パケット受信ステータスの書き込み	+4
合計	24

に確保する場合, 受信領域はページアウトされないピンダウン領域であるため, 受信領域を大きく確保すると主記憶を圧迫することになる. そのため, 圧迫を防ぐために受信領域を小さく確保することになるが, この場合, 受信領域の不足はより高い確率で起こり得る上に, ホストからメッセージの受信検知のためのポーリングを頻繁に行う必要が出てくるため, ホストのオーバヘッドが増加する. 従って, オーバヘッドを抑えつつ, IPUSH 機構によるメッセージの送受信を実現するには, 受信領域内の各リングバッファのサイズを大きめに確保し, AMT Address Table の設定により複数のプロセスで 1 つのリングバッファを共有するといった設定を行う必要があると考えられる.

一方, DIMMnet-2 においては, ほかの PC クラスタ向けネットワークインタフェースと異なり, メッセージの受信領域がネットワークインタフェース上の SO-DIMM であるため, この問題は起こりにくくなる. この SO-DIMM 領域はホストのアドレス空間上にマップされていないため, ホストのチップセットなどに制限されずに大容量の SO-DIMM を搭載することが可能である. 仮に 1GByte の SO-DIMM をこの試作基板上に搭載した場合, SO-DIMM の総容量は 2GByte である. 一方で, 汎用 I/O バスに接続される PC クラスタ向けネットワークインタフェースに搭載されているメモリ容量は, QsNET II のネットワークインタフェースである QM500 PCI-X Network Adapter で 64MByte[54], Myrinet/PCI-X E Card で 2MByte または 4MByte[41], Mellanox 社の InfiniHost III Ex HCA Cards シリーズで最大 128MByte[63] である. DIMMnet-2 のアーキテクチャはネットワークインタフェース上のメモリ領域に受信バッファを確保することで受信バッファサイズを大きくできるという点で IPUSH 機構によるメッセージの受信に適していると言える.

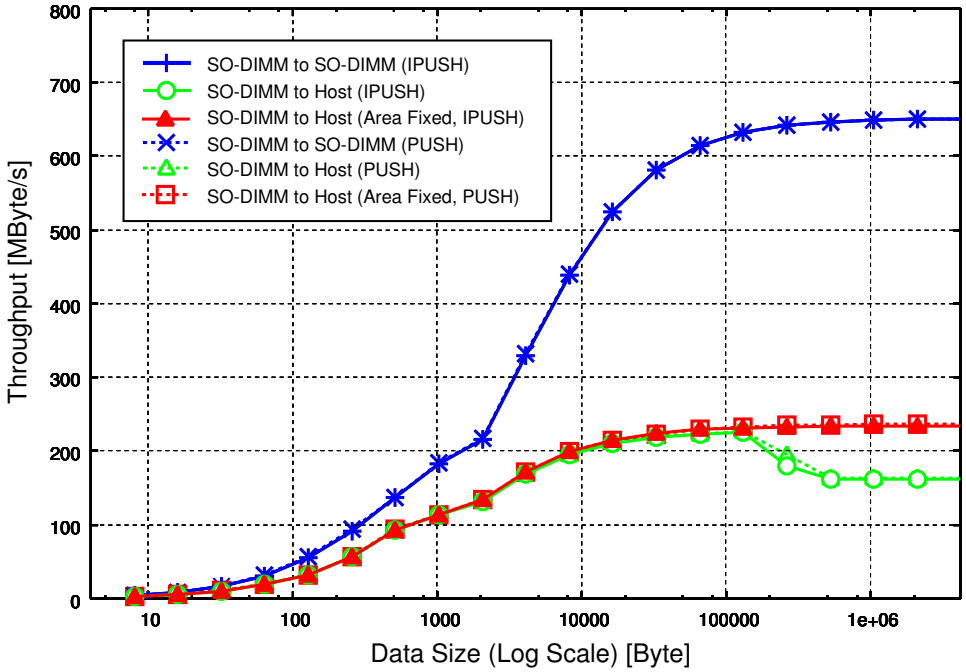


図 7.5　PUSH と IPUSH のスループットの比較

7.1.6　IPUSH 機構の評価

本節では DIMMnet-2 に実装した IPUSH 機構の評価を示す. 本評価では IPUSH 機構でパケットを受信した際に, PUSH に比べ, どの程度性能低下が起こるかを調べる. 評価環境は表 6.1 に示したものと同一である.

IPUSH 機構を用いた場合と用いない場合で, 受信したメッセージの受信処理が完了するまでのレイテンシとスループットを比較する. 測定方法は 6.3.2 節の手法と同一である. PUSH と IPUSH では送信側の処理はどちらも全く同じであり, 受信処理のみ異なるため, それぞれの機構の通信性能を比較することが可能となる.

測定結果を図 7.5 に示す. PUSH の結果は図 6.12 と同一である. SO-DIMM 間のメッセージ通信処理の最大スループットは 650.56MByte/s であり, PUSH のと同等の性能が得られている. また, 図 7.5 のスループットの上昇率が PUSH とほぼ同じであることから, IPUSH 機構のオーバヘッドは小さいと言える. ヘッダ 16Byte, データ 8Byte の計 24Byte のメッセージを送受信した際の RTT/2 は PUSH で 1.75μs, IPUSH で 1.79μs であり, ちょうど IPUSH 機構の付加によって増加するレイテンシと同じだけ増加している.

受信したデータの主記憶へのコピーを含めた際の最大スループットは IPUSH で 225.85MByte/s であり, データサイズを大きくすると, 最終的に 161MByte/s 程度に落ち着く. コピー先の領域を 64Byte に固定するとスループットの低下は見られておらず, 234MByte/s 程度の性能が得られており, PUSH と同じ挙動を示している. 主記憶へデータをコピーした際の最小の RTT/2 は PUSH で 2.80μs, IPUSH で 2.84μs であった.

これらの評価結果より, IPUSH 機構は PUSH, つまり通常の RDMA と変わらない性能を実現していることが示された.

7.2 LHS (Limited-length Head Separation)

LHS 機構は “受信したメッセージの先頭部分と残りの後続部分を別の受信バッファに格納する” 受信機構である. 一般にメッセージ通信で用いられるメッセージには, 実データ部にメッセージの識別子 (エンベロープ) が付加されており, メッセージの受信側はこの識別子の情報を元に, メッセージのマッチングを行い, 一致したメッセージを読み出す. 例えば, MPI においてはエンベロープとしてコミュニケータ (Context), ランク (Rank), タグ (Tag) を含む.

2.3.2 節で MPI におけるメッセージの受信処理について述べたが, 受信処理ではエンベロープの情報をもとに受信関数に対応するメッセージがキューに格納されているかどうかを探索する. そのため, 受信したメッセージサイズが大きいとキューの探索範囲が広がってしまい, メッセージの受信処理に要するオーバヘッドが大きくなるという問題がある.

LHS 機構を MPI に適用すると, エンベロープとデータ部を別の領域に受信できるため, キューの探索時にはエンベロープのみが格納されたバッファを探索することになり, 探索の効率が改善される. また, メッセージサイズに探索処理が依存しなくなるという利点がある.

本研究ではこの LHS 機構を IPUSH 機構に追加するという形で DIMMnet-2 ネットワークインタフェースコントローラ上に実装を行った.

7.2.1 LHS 機構の設計と実装

DIMMnet-2 では LHS 機構はメッセージの先頭部分を 64Byte までとし, 64Byte を 1 エントリとするバッファを設ける. 図 4.2 中の LH Buffer が LHS 用のバッファであり, 各プロセスに 64 エントリ分の領域を確保している. 従って, 各プロセスに 4KByte, LH Buffer 全体で 8KByte の領域となる. この LH Buffer をリングバッファとして用い, ホストプロセッサは Head ポインタと Tail ポインタを User Register を介して参照可能である.

エンベロープを含めた総メッセージサイズが 64Byte を超える場合, 後続のメッセージは SO-DIMM に IPUSH 機構を利用して格納され, LH Buffer にはエンベロープ以外に SO-DIMM へ格納されたデータに対するポインタも格納される. 図 7.6 に LHS を使用する際にユーザプロセスが送信するメッセージのフォーマットを, 図 7.7 に LH Buffer に格納されるデータのフォーマットを示す. これらの図では DIMMnet-2 向けに開発された MPI[18] を例にし, Context を 16bit, Rank を 14bit, Tag を 32bit としている.

図 7.6 に示したフォーマットでユーザプロセスが用意したデータに, DIMMnet-2 ヘッダ (図 4.23) が付加されてネットワークに送出される. LHS 機構を利用してメッセージを受信するには要求発行時にプリミティブとして表 4.13 の IPUSH with LHSv1 もしくは IPUSH with LHSv2 を指定する. これらのプリミティブを指定すると, DIMMnet-2 ヘッダ内の OTYPE が IPUSH, lflag が 1 のパケットが送出される.

LHSv1 の場合, エンベロープと送信するデータを SO-DIMM の連続した領域に書き込んでおき, そのデータを送信する. しかし, 既に SO-DIMM に格納されているデータに対してエンベロープを追加する際に, SO-DIMM の領域が空いていない場合があるため, エンベロープを付加するのが難しいという問題がある. そこで, LHSv2 では Write Window のデータと SO-DIMM のデータを合わせて送信できるようにし, Write Window にエンベロープを書き込んでおくことで SO-DIMM 内の任意のデータで LHS 機構を利用することできる. LHSv2 のこの機能は “512Byte までの Write Window 内のデータと SO-DIMM のデータを合わせてネットワークに送出する” というものである. そのため, エンベロープ以外のデータを付加することができ, また, エンベロープのサイズやフォー

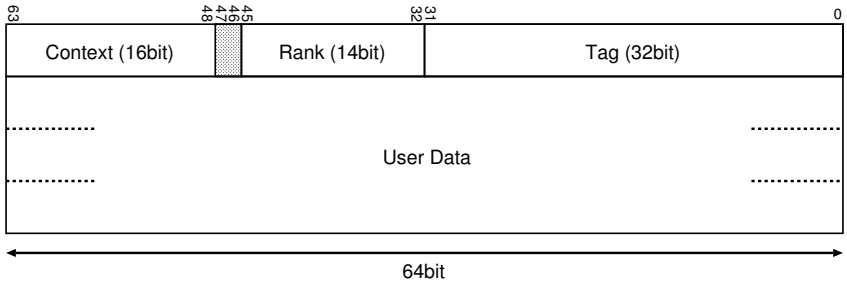


図 7.6　LHS を利用する際のメッセージフォーマット

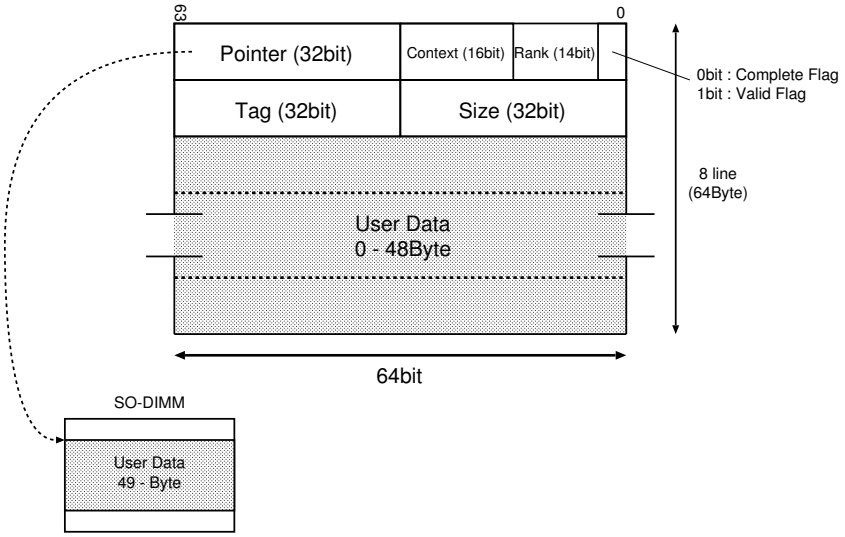


図 7.7　LH Buffer に格納されたメッセージのフォーマット

マットが異なるメッセージ通信にも対応することが可能となる.

LHSv2 の場合, Write Window と SO-DIMM の 2 箇所からデータを読み出すという形式上, ホストから発行するプリミティブのフォーマットが異なる. LHSv2 のプリミティブのフォーマットを図 7.8 に示す. LHSv2 ではプリミティブフォーマットの DSTOff の領域の下位 10bit が Write Window から転送するサイズ, 続く 12bit が Write Window のアドレスを示す. また, Lower 側の Size には SO-DIMM から読み出すサイズに Write Window から読み出しすサイズを加算した値を指定する.

LHSv1 または LHSv2 によって送出されたメッセージを受信したリモートノードは, そのメッセージを LH Buffer と SO-DIMM に分けて書き込む. 受信したメッセージのうち, エンベロープとそれ以外のデータの先頭 48Byte までが LH Buffer のエントリに書き込まれる. LH Buffer に書き込まれる Tag, Context, Rank の情報は送信側のプロセスが SO-DIMM, または Write Window に書き込んだ値と同じである. Size は受信したメッセージのサイズから, マッチング用データサイズを引いた値となる. Complete Flag と Valid Flag は受信検出のために用いられる. 48Byte を超えた部分のデータは, SO-DIMM 上の IPUSH 用のリングバッファに書き込まれる. LH Buffer に格納されたデータの Pointer は, SO-DIMM に書き込まれたデータの先頭アドレスを示す.

LH Buffer のエントリに書き込みが行われたことは User Register をポーリングすることで検出できるが, Valid Flag と Complete Flag の値の反転をポーリングすることによって検出することも

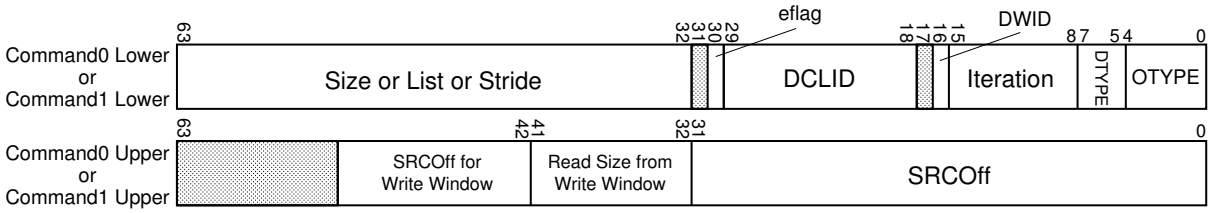


図 7.8 IPUSH with LHSv2 のプリミティブフォーマット

できる. コントローラリセット時, これらのフラグはすべて 0 にリセットされる. 以下に LH パケット受信時のフラグの挙動を示す.

1. パケットを受信すると, Valid Flag が反転する

   Valid Flag が反転した時点で, Context, Rank, Tag, Size, Pointer (48Byte を超える場合) には, 適切なデータが書き込まれている. Size の値が 48Byte を超える場合, Valid Flag の反転を検出した段階で Size や Pointer の値を用いて, VL のプリミティブを発行して読み出しを開始することができる. このプリミティブは, Core Logic 内部で受信処理が完了するまで実行されない.

2. エントリへのデータ書き込みが完了すると, Complete Flag が反転する

7.2.2 LHS 機構の評価

本節では LHS 機構の評価を示す. 評価環境は表 6.1 に示したものと同一である. LHS 機構は IPUSH 機構と一緒に用いることを前提に実装を行ったため, 本評価では IPUSH で通信を行った際に, LHS を用いる場合と用いない場合のレイテンシの比較を行う.

測定項目は次の 4 パターンである. それぞれのノードが送信側と受信側を交互に実行し, RTT/2 を測定した. 本評価ではエンベロープは図 7.6 に従い, Rank：14bit, Context：16bit, Tag：32bit とした. 送信側はパケットを BOTF で送出し, パケットヘッダを 16Byte, データ部は 8〜120Byte の範囲で変化させた. IPUSH Only と IPUSH + LHSv1 は送信するデータが Write Window にある場合を想定し, VL + IPUSH + LHSv1 と IPUSH + LHSv2 は SO-DIMM に存在するものとしている.

- IPUSH Only
  - 送信側
    1. Write Window に OTYEP=IPUSH のパケットヘッダ, エンベロープ, データ部を書き込む
    2. BOTF 要求を発行
  - 受信側
    1. User Register 内の Status Next Write Address をポーリングし, パケットの受信を検知
    2. パケット受信ステータスを読み出し, 受信データサイズ, 受信先のアドレスを取得
    3. 当該の SO-DIMM 領域からエンベロープを読み出し, Rank, Context, Tag を比較

4. SO-DIMM からデータ本体を読み出し, 主記憶にコピー

- IPUSH + LHSv1
  - 送信側
    1. Write Window に OTYPE=IPUSH with LHSv1 のパケットヘッダ, エンベロープ, データ部を書き込む
    2. BOTF 要求を発行
  - 受信側
    1. LH Buffer の Complete Flag と Valid Flag とポーリングし, パケットの受信を検知
    2. LH Buffer からエンベロープを読み出し, Rank, Context, Tag を比較
    3. LH Buffer からデータを読み出し, 主記憶にコピー
    4. データサイズが LH Buffer の 1 エントリに収まらない場合 (データ部のサイズが 48Byte 以上) は SO-DIMM からデータを読み出し, 主記憶にコピー
- VL + IPUSH + LHSv1
  - 送信側
    1. VL で SO-DIMM から Prefetch Window に転送するデータを読み出す
    2. 読み出している間に, Write Window に OTYPE=IPUSH with LHSv1 のパケットヘッダを書き込む
    3. Prefetch Window Flag をポーリングし, VL の完了を検知
    4. VL で読み出したデータをエンベロープと一緒に Write Window に書き込む
    5. BOTF 要求を発行
  - 受信側
    * IPUSH + LHSv1 の評価と同じ
- IPUSH + LHSv2
  - 送信側
    1. Write Window にエンベロープを書き込む
    2. IPUSH with LHSv2 要求を発行
  - 受信側
    * IPUSH + LHSv1 の評価と同じ

評価結果を図 7.9 に示す. 図 7.9 の横軸はエンベロープのサイズ (8Byte) を加えたサイズである. また, VL + IPUSH + LHSv1 は IPUSH + LHSv2 との比較のために測定したものであり, SO-DIMM にエンベロープが書き込めない場合に LHS 機構を利用する際の処理を想定したものである.

まず, IPUSH Only と IPUSH + LHSv1 の結果であるが, パケットがすべて LH Buffer の 1 エントリに収まるデータサイズまでは約 3μs のレイテンシの差があり, LHS 機構の効果は明らかである. LHS の場合, LH Buffer が Write Back 属性であり, 評価に用いた Pentium4 の L2 キャッシュのキャッシュラインが 64Byte であることから, エンベロープをポーリングした時点で 1 エントリ分のデー

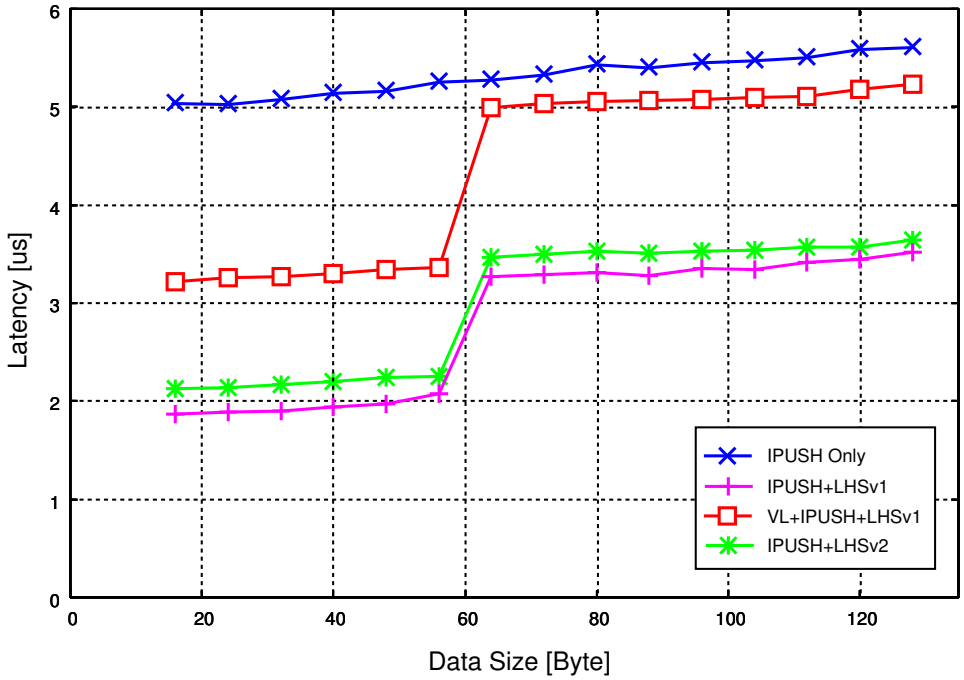


図 7.9　LHS 機構によるレイテンシの変化

タがホストプロセッサのキャッシュに格納される. そのため, ポーリングした結果, まだデータを受信していない場合はキャッシュをフラッシュする必要があるが, 既にデータを受信していた場合は LH Buffer からのデータの読み出しなどの操作はすべて L2 キャッシュに対して行われることになり, レイテンシは低く抑えられる. また, 1 エントリに収まらないデータサイズにおいても, 約 2μs の差があり, エンベロープを LH Buffer から読み出すことによるレイテンシの削減が見られている.

次に IPUSH with LHSv2 の評価であるが, LHSv2 を利用することにより, SO-DIMM にエンベロープが格納できない場合に LHSv1 に比べて約 1〜1.5μs のレイテンシの削減が実現されている.

7.3　まとめ

本章ではメッセージ “受信” 機構である IPUSH と LHS の設計, 及び実装について述べた.

IPUSH 機構の基本通信性能の評価を行った結果, 通常の RDMA write である PUSH と同等の通信性能を示した. これは, IPUSH 機構をハードワイヤードロジックで実現したことによるものである.

LHS 機構では, MPI におけるマッチング処理を想定した評価を行い, その結果, メッセージ通信時におけるレイテンシの削減が確認された. これらの機構を利用して MPI などのメッセージ通信ライブラリを実装することで, メッセージ通信を使用する並列システムでの性能向上が期待できる.

第8章　結論

8.1　本研究のまとめ

本研究では, DDR-SDRAM スロット装着型ネットワークインタフェースである DIMMnet-2 のネットワークインタフェースコントローラの設計, 及び実装を行い, その基本通信性能の評価を行った. また, メッセージ通信を支援するためのメッセージ “受信” 機構である IPUSH と LHS を実装し, その評価を行った.

DIMMnet-2 プロジェクトは汎用の PC を用いたとしても高性能な PC クラスタを構築できるようにすることを目的として開始された. DIMMnet-2 は DIMMnet-1 はもちろんのこと, RHiNET-2 とも非常に関係の深いインターコネクトであるが, すべての処理をハードワイヤードで実装する, ネットワークインタフェース上のメモリに対して間接アクセス方式を採用するなど, 全く異なるアプローチでネットワークインタフェースコントローラの設計を行った. また, ネットワークインタフェース以外のコンポーネントに商用の高性能インターコネクトであり, 標準規格である InfiniBand を採用し, 汎用性という点でも向上させている.

実機を用いた評価の結果, 低遅延通信機構である BOTF においては, ホストプロセッサの処理とネットワークインタフェースコントローラの処理をオーバラップ可能にしたことによって, PIO 通信ながら双方向で 1163.70MByte/s と, 1GByte/s 以上のスループットを達成した. レイテンシの面でも, RTT/2 の値が 1.73μs と, 汎用 PC を用いているにも関わらず, サーバやワークステーションに PC クラスタ向けインターコネクトを接続した場合に匹敵する低い通信レイテンシを達成した.

ネットワークインタフェースコントローラが FPGA であることを利用して機能の拡張を行い, メッセージ通信支援機構である IPUSH と LHS を DIMMnet-2 試作基板に搭載した. これらの機構は, 先行研究のように通信処理をネットワークインタフェースコントローラとホストプロセッサのどちらか一方に任せてしまうのではなく, 双方が協力してメッセージ通信の高速化を行うというアプローチである. そのため, ホストプロセッサの高性能化による恩恵と, ネットワークインタフェースコントローラへのオフローディングによるホストプロセッサの負荷の低減という, 両者の利点を活かすことができる. IPUSH と LHS の評価の結果, メッセージ受信時の処理のレイテンシを削減することや, 受信バッファの削減などが実現可能であることが示された.

しかしながら, 受信したデータをホスト上の主記憶に読み出すと, データサイズが大きくなるに従って, ホストプロセッサのキャッシュにデータが収まらなくなり, ネットワークのスループットに比べてホスト側が低くなるという問題が明らかになった. Prefetch Window のサイズが SO-DIMM のサイズに比べて小さいため, サイズの大きいデータを SO-DIMM から読み出す際に, キャッシュフラッシュとそれに伴うプリフェッチを繰り返す必要がある. このプリフェッチが間に合わなくなることが性能低下の原因である. この問題は Prefetch Window のサイズを大きくしたり, ホスト上で動作するソフトウェア側でキャッシュフラッシュとプリフェッチをプログラム中の適切な位置に挿入できれば, ある程度は解決可能であると考えられる. しかし, DIMMnet の本質的な問題として,

主記憶に DMA 転送できないため, ホスト側のスループットの改善には限界があると言える.

8.2 DIMMnet を取り巻く現状

DIMMnet-2 プロジェクトの間に, 汎用 PC における標準的なメモリバスは DDR2-SDRAM バスに移り変わっていった. このような背景から, また, DIMMnet-2 の基板上の問題を解決するためにも, 第三世代の DIMMnet であり, DDR2-SDRAM バスに接続される DIMMnet-3 の開発が 2006 年度より開始された. しかし, PC における I/O バスも PCI バスから PCI-Express へと移り変わってきており, また, AMD のプロセッサを使用したシステムでは内部で HyperTransport が採用されていることから, 汎用 PC においても高スループット, かつ低レイテンシな通信環境を構築することが可能になってきた. このことから, 2007 年の時点ではメモリスロットを利用するという利点を通信性能のみに求めることはできなくなってきたと言える.

しかし, 近い将来ネットワークの速度が 10Gbps クラスから 100Gbps クラスへと進化した際に, PCI-Express ではスループットが不足することが予想される. PCI-Express 2.0 では PCI-Express の 1 レーン当たりのスループットが現行の 2.5Gbps から 5Gbps に拡張され, 2007 年には製品化される見込みである. この場合の 16 レーンの実効スループットは片方向で 64Gbps (注 1)となるが, 2009 年には 100Gbps の Ethernet が登場すると言われており [111], PCI-Express 2.0 では性能が不足する. PCI-Express は PCI バスに比べて性能向上率が高く, 既に PCI-Express 3.0 の規格が立ち上げられている. PCI-Express 3.0 の 1 レーン当たりのスループットは 8Gbps となるため, 16 レーン用いることで 100Gbps のネットワークに対して十分なスループットを持つ. 製品化は 2010 年とされているが, 汎用 I/O バスの場合, メモリバスほど市場からの性能要求が厳しくないこともあり, PCI バス時代の 64bit PCI バスや, PCI-X バスのように, 汎用 PC には搭載されない可能性がある.

それに対して, メモリバスはチャネル数の増加やベースクロックの向上によりスループットがさらに向上している. Dual-Core Xeon 用のチップセットである i5000X/i5000P では Dual Channel をさらに拡張し, Quad Channel での動作をサポートしている [112][113]. PC2-5300 FB-DIMM を Quad Channel で動作させた場合, スループットは 21.2GByte/s(注 2)となり, 既に 100Gbps のネットワークをサポートするのに十分な性能を持っている. このチップセットはサーバ向けの製品であるが, ホストプロセッサのマルチコア化により急速にメモリバスの性能向上が求められていることから, Quad Channel の技術は汎用 PC においても採用される可能性はあると思われる.

DIMMnet でこういった複数のメモリチャネルを利用するためにはメモリバスに複数枚の DIMMnet を接続する必要がある. そこで, 次世代の DIMMnet である DIMMnet-3 では Dual Channel に対応するために, 親基板と子基板の 2 種類の基板に分割されている. 図 8.1 に DIMMnet-3 の概観を示す. 子基板はメモリスロットに装着する基板であり, ホストとのインタフェースの機能のみを持つ. この子基板は 2 枚搭載する. 親基板は PCI バスに装着され, 通信処理はすべて親基板で処理される. PCI バスは電源供給と基板支持のために用いられ, PCI バス経由でのデータの転送は行われない. 子基板とは専用のケーブルで接続され, 親基板 1 枚に対し子基板 2 枚を接続する.

このような構成をとることで, ホスト側は Dual Channel 動作で DIMMnet を扱うことができ, InfiniBand ネットワーク側からは LID が親基板に 1 個だけ割り当てられるので, 制御が容易になる. 基板を分割したことにより, それぞれの基板間でのデータ転送によるオーバヘッドから, DIMMnet-2 よりもレイテンシは増加すると考えられるが, Dual Channel での動作が可能になるという点でス

(注 1)8B10B エンコーディングにより, 1 レーン当たりの実効スループットは 4Gbps
(注 2)PC2-5300 のスループットを 5.3GByte/s として計算

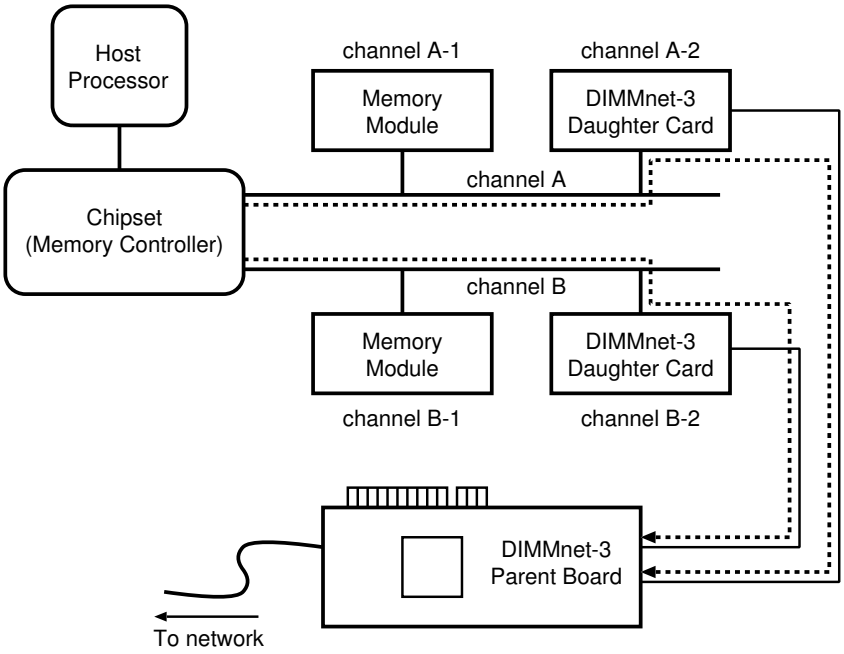


図 8.1　DIMMnet-3 概観

ループットの向上が見込める. この方式を発展させることで, Quad Channel への対応も容易と思われる.

8.3　おわりに

本研究では, DIMMnet-2 試作基板を用いて基本通信性能の評価を行ったが, ネットワークインタフェースそのものの問題により, これ以上のアプリケーションレベル, システムレベルの評価が困難となってしまった. そのため, 本研究で扱った通信機構や, ネットワークインタフェース上のメモリに対する不連続アクセス機構を利用したアプリケーションの評価を行うことができなくなってしまったのが残念である.

とはいえ, 本研究で実装を行った, BOTF の連続発行による高性能化手法や IPUSH 機構, LHS 機構といった通信機構は DIMMnet 以外の, 汎用 I/O バスに装着される一般的なインターコネクトにも適用可能である. 本研究の成果が, 今後のインターコネクトの発展に寄与できれば幸いである.

謝辞

本研究の機会を与えてくださり, 手厚くご指導くださった慶應義塾大学 理工学部 天野 英晴 教授に心より感謝致します.

博士論文の執筆に際し, 査読の労を執って頂いた慶應義塾大学 理工学部 山本 喜一 教授, 寺岡 文男 教授, 西 宏章 准教授に心より感謝致します.

様々な場面で多大なご助言をくださった慶應義塾大学 理工学部 山崎 信行 准教授に心より感謝致します.

DIMMnet-2 プロジェクトに誘って頂き, また, 本研究に対して様々なご助言を頂きました東京農工大学 中條 拓伯 准教授, 株式会社 東芝 研究・開発センター 田邊 昇 氏に心より感謝致します.

基板製作, メモリインタフェースの設計, 及び実装にご協力頂き, また, これらの動作に関して相談に乗って頂いた, 株式会社 日立情報通信エンジニアリング 今城 英樹 氏, 上嶋 利明 氏, 岩田 英伸 氏に心より感謝致します.

DIMMnet-2 ネットワークインタフェースコントローラの設計, 及び InfiniBand との接続にご協力頂きました東京農工大学 中條研究室 濱田 芳博 氏, 株式会社 ルネサステクノロジ 荒木 健志 氏に心より感謝致します.

予算管理でお世話になりました, 慶應義塾大学 矢上研究支援センター 片桐 素美 氏に心より感謝致します. 至らぬ点が多々あり, 本当にご迷惑をお掛けしました.

本研究を共に行った DIMMneters の三氏,
日本電気株式会社 宮部 保雄 氏, ソニー株式会社 宮代 具隆 氏, 日本電気株式会社 伊沢 徹 氏に心より感謝致します. コントローラのバグではご迷惑をお掛けしました.

本研究に惜しみないご協力を頂きました天野研究室 PDARCH グループの皆様,
株式会社 東芝 セミコンダクター 渡邊 幸之介 氏, 慶應義塾理工学インフォメーションテクノロジーセンター 大塚 智宏 助教, ソニー株式会社 伊豆 直之 氏に心より感謝致します. 公私に渡り, お世話になりました.

日頃より暖かいご支援を頂きました天野研究室の皆様に心より感謝致します. RHiNET の研究から始まって, 足掛け 6 年間, 本当にお世話になりました.

最後に, 6 年に渡る研究生活を支えてくれた家族に心より感謝致します. 本当に有難うございました.

2007 年 8 月
辻 聡

参考文献

[1] TOP500 Supercomputer sites. http://www.top500.org/.

[2] Nanette J.Boden, Denny Cohen, Robert E.Felderman, Alan E.Kulawik, Charies L.Seitz, Jakov N.Seizovic, and Wen-King Su. Myrinet - A gigabit per second local area network. *IEEE Micro*, Vol. 15, No. 1, pp. 29–36, Feb. 1995.

[3] Fabrizio Petrini, Wu-chun Fang, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. The Quadrics Network: High-Performance Clustering Technology. *IEEE Micro*, Vol. 22, No. 1, pp. 46–57, Jan./Feb. 2002.

[4] Mellanox Technologies, Inc. Introduction to InfiniBand. http://www.mellanox.com/, 2003.

[5] Intel White Paper. Creating a Third Generation I/O Interconnect.

[6] HyperTransport Consortium. http://www.hypertransport.org/.

[7] PCI-SIG. http://www.pcisig.com/home.

[8] Intel, Corp. *Intel® 915G/915GV/910GL/915P/i15PL/910GL Express Chipset Datasheet*, Feb. 2005.

[9] Intel, Corp. *Intel® 925X/925XE Express Chipset Datasheet*, Nov. 2004.

[10] Noboru Tanabe, Junji Yamamoto, Hiroaki Nishi, Tomohiro Kudoh, Yoshihiro Hamada, Hironori Nakajo, and Hideharu Amano. MEMOnet: Network interface plugged into a memory slot. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'00)*, pp. 17–26, 2000.

[11] 伊豆直之. DIMMnet-2 ネットワークインタフェースの設計と実装. 修士論文, 慶應義塾大学大学院理工学研究科, 2003.

[12] 北村 聡, 伊豆 直之, 田邊 昇, 濱田 芳博, 中條 拓伯, 渡邊 幸之介, 大塚 智宏, 天野 英晴. DIMMnet-2 ネットワークインタフェースボードの試作. 情報処理学会研究報告, 第 2004-ARC-159 巻, pp. 151–156, Jul. 2004.

[13] 北村 聡, 伊豆 直之, 伊沢 徹, 宮代 具隆, 宮部 保雄, 渡邊 幸之介, 大塚 智宏, 濱田 芳博, 田邊 昇, 中條 拓伯, 天野 英晴. FPGA を用いたメモリスロット装着型ネットワークインタフェースの設計. 第 12 回 FPGA/PLD Design Conference ユーザ・プレゼンテーション, pp. 13–20, Jan. 2005.

[14] 北村聡. DIMMnet-2 ネットワークインタフェースコントローラの設計と実装. 修士論文, 慶應義塾大学大学院理工学研究科, 2004.

[15] 北村 聡, 濱田 芳博, 宮部 保雄, 伊澤 徹, 宮代 具隆, 田邊 昇, 中條拓伯, 天野 英晴. DIMMnet-2 ネットワークインタフェースコントローラの設計と実装. 情報処理学会論文誌コンピューティングシステム, Vol. 46, No. SIG12 (ACS11), pp. 13–26, Aug. 2005.

[16] 北村 聡, 宮部 保雄, 中條 拓伯, 田邊 昇, 天野 英晴. メッセージパッシングモデルを支援するパケット受信機構の実装. 情報処理学会研究報告, 第 2005-ARC-165 巻, pp. 39–44, Nov. 2005.

[17] 北村 聡, 宮部 保雄, 中條 拓伯, 田邊 昇, 天野 英晴. メッセージパッシングモデルを支援するパケット受信機構の DIMMnet-2 への実装と評価. 情報処理学会論文誌コンピューティングシステム, Vol. 47, No. SIG12 (ACS15), pp. 59–73, Sep. 2006.

[18] 宮部保雄. ハードウェアによる MPI 派生データ型通信の支援. 修士論文, 慶應義塾大学大学院理工学研究科, 2006.

[19] Ron Minnich, Dan Burns, and Frank Hady. The Memory-Integrated Network Interface. *IEEE Micro*, Vol. 15, No. 1, pp. 11–20, Feb. 1995.

[20] P.H.W.Leong, M.P.Leong, O.Y.H.Cheung, T.Tung, C.M.Kwok, M.Y.Wong, and K.H.Lee. Pilchard - A Reconfigurable Computing Platform with Memory Slot Interface. In *Proceedings of the 9th Annual IEEE Symposium on Field Programmable Custom Computing Machines (FCCM'01)*, pp. 170–179, 2001.

[21] Dennis Ka Yau Tong, Pui Sze Lo, Kin Hong Lee, and Philip H.W.Leong. A System Level Implementation of Rijndael on a Memory-slot based FPGA Card. In *Proceedings of the 2002 IEEE International Conference on Field-Programmable Technology (FPT'02)*, pp. 102–109, Dec. 2002.

[22] Christian Plessl and Marco Platzner. TKDM - A Reconfigurable Co-processor in a PC's Memory Slot. In *Proceedings of the 2003 IEEE International Conference on Field-Programmable Technology (FPT'03)*, pp. 252–259, Dec. 2003.

[23] Jeff Draper, Jacqueline Chame, Mary Hall, Craig Steele, Tim Barrett, Jeff LaCoss, John Granacki, Jaewook Shin, Chun Chen, Chang Woo Kang, Ihn Kim, and Gokhan Daglikoca. The Architecture of the DIVA Processing-In-Memory Chip. In *Proceedings of the 16th ACM International Conference on Supercomputing (ICS'02)*, pp. 14–25, Jun. 2002.

[24] Sumit D.Mediratta, Craig Steele, Jeff Sondeen, and Jeffrey Draper. An Area-efficient and Protected Network Interface for Processing-In-Memory Systems. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'05)*, pp. 2951–2954, May. 2005.

[25] Tomohiro Kudoh, Shinji Nishimura, Junji Yamamoto, Hiroaki Nishi, Osamu Tatebe, and Hideharu Amano. RHiNET: A network for high performance parallel processing using locally distributed computers. In *Proceedings of the IEEE International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'99)*, pp. 69–73, Nov. 1999.

[26] 渡邊幸之介. RHiNET-2 システムの実装と評価. 修士論文, 慶應義塾大学大学院理工学研究科, 2002.

[27] 西 宏章, 多昌 廣治, 西村 信治, 山本 淳二, 工藤 知宏, 天野 英晴. LASN 用 8Gbps/port 8×8 One-chip スイッチ：RHiNET-2/SW. 並列処理シンポジウム JSPP'00 論文集, pp. 173–180, May. 2000.

[28] 西 宏章, 多昌 廣治, 工藤 知宏, 天野 英晴. 仮想チャネルキャッシュを持つネットワークルータの構成と性能. 並列処理シンポジウム JSPP'99 論文集, pp. 71–78, Jun. 1999.

[29] 天野 英晴. 並列コンピュータ, 情報系教科書シリーズ, 第 18 巻. 昭晃堂, 1996.

[30] 山本 淳二, 渡邊 幸之介, 土屋 潤一郎, 原田 浩, 今城 英樹, 寺川 博昭, 西 宏章, 田邊 昇, 上嶋 利明, 工藤 知宏, 天野 英晴. 高性能計算をサポートするネットワークインタフェース用コントローラチップ Martini. 情報処理学会論文誌ハイパフォーマンスコンピューティングシステム, Vol. 43, No. SIG06 (HPS5), pp. 122–133, Sep. 2002.

[31] 山本 淳二, 田邊 昇, 西 宏章, 土屋 潤一郎, 渡辺 幸之介, 今城 英樹, 上島 利明, 金野 英俊, 寺川 博昭, 慶光院 利映, 工藤 知宏, 天野英晴. 高速性と柔軟性を併せ持つネットワークインタフェース用チップ：Martini. 情報処理学会研究報告, 第 2000-ARC-140 巻, pp. 19–24, Nov. 2000.

[32] 山本 淳二, 渡邊 幸之介, 土屋 潤一郎, 今城 英樹, 寺川 博昭, 西 宏章, 田邊 昇, 工藤 知宏, 天野 英晴. RHiNET の概要と Martini の設計／実装. 情報処理学会研究報告, 第 2001-ARC-144 巻, pp. 37–42, Jul. 2001.

[33] Konosuke Watanabe, Junji Yamamoto, Jun-ichiro Tsuchiya, Noboru Tanabe, Hiroaki Nishi, Tomohiro Kudoh, and Hideharu Amano. Preliminary Evaluation of Martini: a Novel Network Interface Controller Chip for Cluster-based Parallel Processing. In *Proceedings of the 20th IASTED International Multi-Conference on Applied Informatics (AI'02)*, pp. 390–395, Feb. 2002.

[34] 渡邊幸之介. 高性能並列分散処理環境向けネットワークインタフェースコントローラ Martini の実装と評価. PhD thesis, 慶應義塾大学大学院理工学研究科, 2006.

[35] 田邊 昇, 山本 淳二, 濱田 芳博, 中條 拓伯, 工藤 知宏, 天野 英晴. DIMM スロット搭載型ネットワークインタフェース DIMMnet-1 とその高バンド幅通信機構 BOTF. 情報処理学会論文誌, Vol. 43, No. 4, pp. 866–878, Apr. 2002.

[36] 田邊 昇, 濱田 芳博, 山本 淳二, 今城 英樹, 中條 拓伯, 工藤 知宏, 天野 英晴. DIMM スロット搭載型ネットワークインタフェース DIMMnet-1 とその低遅延通信機構 AOTF. 情報処理学会論文誌ハイパフォーマンスコンピューティングシステム, Vol. 44, No. SIG01 (HPS6), pp. 10–23, Jan. 2003.

[37] 大塚 智宏, 渡邊 幸之介, 北村 聡, 原田 浩, 山本 淳二, 西 宏章, 工藤知宏, 天野 英晴. 分散並列処理用ネットワーク RHiNET-2 の性能評価. 先進的計算基盤システムシンポジウム SACSIS'03 論文集, pp. 45–52, May. 2003.

[38] Yutaka Ishikawa, Hiroshi Tezuka, Atsushi Hori, Shinji Sumimoto, Toshiyuki Takahashi, Francis O'Carroll, and Hiroshi Harada. RWC PC Cluster II and SCore Cluster System Software – High Performance Linux Cluster. In *Proceedings of the 5th Annual Linux Expo*, pp. 55–62, May. 1999.

[39] Charles L.Seitz, Nanette J.Boden, Jakov Seizovic, and Wen-King Su. The Design of the Caltech Mosaic C Multicomputer. In *Proceedings of the 1993 Symposium on Research on Integrated Systems*, pp. 1–22, 1993.

[40] Danny Cohen, Gregory Finn, Robert Felderman, and Annette DeSchon. ATOMIC: A High-Speed Local Communication Architecture. *Journal of High Speed Networks*, Jan. 1994.

[41] Myricom, Inc. `http://www.myri.com/`.

[42] Hiroshi Tezuka, Atsushi Hori, and Yutaka Ishikawa. Design and Implementation of PM: A Communication Library for Workstation Cluster. 並列処理シンポジウム JSPP'96 論文集, pp. 41–48, Jun. 1996.

[43] Hiroshi Tezuka, Atsushi Hori, and Yutaka Ishikawa. PM: A High-Performance Communication Library for Multi-user Parallel Environments. In *RWC Technical Report*, Nov. 1996.

[44] Toshiyuki Takahashi, Shinji Sumimoto, Atsushi Hori, Hiroshi Harada, and Yutaka Ishikawa. PM2: High Performance Communication Middleware for Heterogeneous Network Environment. In *Proceedings of the 2000 ACM/IEEE International Conference on Supercomputing (Supercomputing'00)*, p. Article No.16, Nov. 2000.

[45] Loic Prylli and Bernard Tourancheau. BIP: a new protocol designed for high performance networking on Myrinet. In *Proceedings of the International Workshop on Personal Computer based Networks of Workstations (IPPS/SPDP'98)*, pp. 475–485, 1998.

[46] Myricom, Inc. *LANai 9*, Jun. 2000.

[47] Myricom, Inc. *LANai 4*, Feb. 1999.

[48] Myricom, Inc. *LANai 5*, Feb, 1999.

[49] Myricom, Inc. *LANai 7*, Jun. 1999.

[50] Myricom, Inc. *Lanai X*, rev. 1.1 edition, Jul. 2003.

[51] Inc. Myricom. *Myrinet Express(MX): A High-Performace, Low-Level, Message-Passing Interface for Myrinet Version 1.1*, Jan. 2006.

[52] Fabrizio Petrini, Salvador Coll, Eitan Frachtenberg, and Adolfy Hoisie. Hardware- and Software-Based Collective Communication on the Quadrics Network. In *Proceedings of the IEEE International Symposium on Network Computing and Applications (NCA'01)*, pp. 24–35, 2001.

[53] Fabrizio Petrini, Eitan Frachtenberg, Adolfy Hoisie, and Salvador Coll. Performance Evaluation of the Quadrics Interconnection Network. *Journal of Cluster Computing*, Vol. 6, No. 2, pp. 125–142, Apr. 2003.

[54] Quadrics, Ltd. `http://www.quadrics.com/`.

[55] Jon Beecroft, Mark Homewood, and Moray McLaren. Meiko CS-2 interconnect Elan-Elite design. *Parallel Computing*, Vol. 20, No. 10–11, pp. 1627–1638, Nov. 1994.

[56] Jon Beecroft, David Addison, Fabrizio Petrini, and Moray McLaren. QsNet^{II}: An Interconnect for Supercomputing Applications. http://www.quadrics.com/, Feb. 2004.

[57] David Addison, Jon Beecroft, David Hewson, Moray McLaren, Duncan Roweth, and Daniel Kidger. QsNet^{II}: Performance Evaluation. http://www.quadrics.com/, May. 2004.

[58] Quadrics, Ltd. *Elan Programming Manual*, Apr. 2005.

[59] Christophe Lemuet. Quadrics QsTenG Ethernet Switch Performance Report. http://www.quadrics.com/, Nov. 2006.

[60] InfiniBand Trade Association. http://www.infinibandta.org/.

[61] InfiniBand Trade Association. *InfiniBand Architecture Specification, Release 1.0*, Oct. 2000.

[62] InfiniBand Trade Association. *InfiniBand Architecture Specification Release 1.2*, Oct. 2004.

[63] Mellanox Technologies, Inc. http://www.mellanox.com/.

[64] QLogic, Corp. http://www.qlogic.com/.

[65] Lloyd Dickman. AN INTRODUCION TO QLOGIC INFINIPATH. PathScale, Inc.

[66] Lloyd Dickman. BEYOND HERO NUMBERS: FACTORS AFFECTING INTERCONNECT PERFORMANCE. PathScale, Inc., Jun. 2005.

[67] QLogic, Corp. INFINIPATH^{TM} INTERCONNECT PERFORMANCE. http://www.pathscale.com/infinipath-perf.html.

[68] Dave Dunning, Greg Regnier, Gary McAlpine, Don Cameron, Bill Shubert, Frank Berry, Anne Marie Merritt, Ed Gronke, and Chris Dodd. The Virtual Interface Architecture. *IEEE Micro*, Vol. 18, No. 2, pp. 66–76, Mar.–Apr. 1998.

[69] Mellanox Technologies, Inc. *Mellanox IB-Verbs API (VAPI)*, 2001.

[70] MVAPICH. http://mvapich.cse.ohio-state.edu/.

[71] Chelsio Communications. http://www.chelsio.com/.

[72] Neterion, Inc. http://www.chelsio.com/.

[73] NetEffect. http://www.neteffect.com/.

[74] NetXen. http://www.netxen.com/.

[75] Tehuti Networks. http://www.tehutinetworks.net/.

[76] LeWiz Communications. http://www.lewiz.com/.

[77] RDMA Consortium. Architectural Specifications for RDMA over TCP/IP. http://www.rdmaconsortium.org/.

[78] NASA. NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/.

[79] Lawrence Livermore National Laboratory. SWEEP3D Benchmark. http://www.llnl.gov/asci_benchmarks/asci/limited/sweep3d/.

[80] Jiuxing Liu, Balasubramanian Chandrasekaran, Jiesheng Wu, Weihang Jiang, Sushmitha Kini, Weikuan Yu, Darius Buntinas, Peter Wyckoff, and Dhabaleswar K.Panda. Performance Comparison of MPI Implementations over InfiniBand, Myrinet and Quadrics. In *Proceedings of the 2003 ACM/IEEE International Conference on Supercomputing (SC'03)*, p. 58, Nov. 2003.

[81] Myricom, Inc. Myrinet Peformance. http://www.myri.com/scs/performance/Myrinet-2000.

[82] Jon Beecroft, David Addison, David Hewson, Moray McLaren, Duncan Roweth, Fabrizio Petrini, and Jarek Nieplocha. QsNet^{II}: Defining High-Performance Network Design. *IEEE Micro*, Vol. 25, No. 4, pp. 34–47, Jul.–Aug. 2005.

[83] Jiuxing Liu, Amith Mamidala, Abhinav Vishnu, and Dhabaleswar K.Panda. Evaluating InfiniBand Performance with PCI Express. *IEEE Micro*, Vol. 25, No. 1, pp. 20–29, Jan.–Feb. 2005.

[84] Myricom, Inc. Myri-10G Peformance. http://www.myri.com/scs/performance/MX-10G.

[85] Wu-chun Feng, Pavan Balaji, Chris Baron, Laxmi N.Bhuyan, and Dhabaleswar K.Panda. Performance Characterization of a 10-Gigabit Ethernet TOE. In *Proceedings of the 13th IEEE International Symposium on High Performance Interconnects (Hot Interconnects 13)*, pp. 58–63, Aug. 2005.

[86] Pavan Balaji, Wu-chun Feng, and Dhabaleswar K.Panda. Bridging the Ethernet-Ethernot Performance Gap. *IEEE Micro*, Vol. 26, No. 3, pp. 24–40, May.–Jun. 2006.

[87] 田邊 昇, 山本 淳二, 工藤 知宏. メモリスロットに搭載されるネットワークインタフェース MEMnet. 情報処理学会研究報告, 第 1999-ARC-134 巻, pp. 73–78, Aug. 1999.

[88] 田邊 昇, 濱田 芳博, 三橋 彰浩, 山本 淳二, 今城 英樹, 中條 拓伯, 工藤 知宏, 天野 英晴. DIMMnet-1 における Martini オンチッププロセッサによる通信の性能評価. 情報処理学会研究報告, 第 2002-ARC-150 巻, pp. 53–58, Nov. 2002.

[89] 渡邊 幸之介, 大塚 智宏, 天野 英晴. ネットワークインタフェース用コントローラチップ Martini における乗っ取り機構の実装と評価. 情報処理学会論文誌コンピューティングシステム, Vol. 45, No. SIG11(ACS7), pp. 393–407, Oct. 2004.

[90] Ron Brightwell and Keith D.Underwood. An Analysis of NIC Resource Usage for Offloading MPI. In *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS'04)*, Apr. 2004.

[91] MPICH Home Page. http://www-unix.mcs.anl.gov/mpi/mpich/.

[92] Keith D.Underwood, K.Scott Hemmert, Arun Rodrigues, Richard Murphy, and Ron Brightwell. A Hardware Acceleration Unit for MPI Queue Processing. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, Apr. 2005.

[93] Xilinx, Inc. Virtex-II Pro および Virtex-II Pro X FPGA FPGA ユーザーガイド, 第 v4.0 版, Mar. 2005.

[94] 社団法人 電子情報技術産業協会. プロセッサ搭載メモリ・モジュール (PEMM) 動作仕様標準 ED-5514, Jul. 1998.

[95] 田邊 昇, 濱田 芳博, 三橋 彰浩, 中條 拓伯, 天野 英晴. メモリスロット装着型ネットワークインタフェース DIMMnet-2 の構想. 情報処理学会研究報告, 第 2002-ARC-152 巻, pp. 61–66, Mar. 2003.

[96] 田邊 昇, 土肥 康孝, 中條 拓伯, 天野 英晴. プリフェッチ機能を有するメモリモジュール. 情報処理学会研究報告, 第 2003-ARC-154 巻, pp. 139–144, Aug. 2003.

[97] 田邊 昇, 中武 正繁, 箱崎 博孝, 土肥 康孝, 中條 拓伯, 天野 英晴. プリフェッチ機能付きメモリモジュールによる不連続アクセスの連続化. 情報処理学会研究報告, 第 2003-ARC-157 巻, pp. 139–144, Mar. 2004.

[98] 田邊 昇, 安藤 宏, 箱崎 博孝, 土肥 康孝, 中條 拓伯, 天野 英晴. プリフェッチ機能を有するメモリモジュールによる PC 上での間接参照の高速化. 情報処理学会論文誌コンピューティングシステム, Vol. 46, No. SIG12 (ACS11), pp. 1–12, Aug. 2005.

[99] Xilinx, Inc. `http://www.xilinx.co.jp/`.

[100] Xilinx, Inc. *RocketIO^{TM} Transceiver User Guide*, v3.0 edition, Feb. 2007.

[101] Xilinx, Inc. *System ACE CompactFlash Solution*, v1.5 edition, Apr. 2002.

[102] 田邊 昇, 箱崎 博孝, 安藤 宏, 土肥 康孝, 中條 拓伯, 宮代 具隆, 北村聡, 天野 英晴. メモリモジュール上での等間隔アクセス連続化の効果. 情報処理学会研究報告, 第 2004-ARC-162 巻, pp. 139–144, Mar. 2005.

[103] 宮代 具隆, 宮部 保雄, 北村 聡, 田邊 昇, 中條 拓伯, 天野 英晴. DIMMnet-2 を用いた間接メモリアクセスの高速化. 情報処理学会研究報告, 第 2006-ARC-170 巻, pp. 85–90, Nov. 2006.

[104] 宮代具隆. DIMMnet-2 におけるベクトルアクセス機構の実装と評価. 修士論文, 慶應義塾大学大学院理工学研究科, 2006.

[105] Intel, Corp. IA-32 インテル®アーキテクチャソフトウェア・デベロッパーズマニュアル, 中巻 A：命令セット・リファレンス A-M, 2004.

[106] Intel, Corp. IA-32 インテル®アーキテクチャソフトウェア・デベロッパーズマニュアル, 中巻 B：命令セット・リファレンス N-Z, 2004.

[107] 宮部 保雄, 宮代 具隆, 北村 聡, 田邊 昇, 中條 拓伯, 天野 英晴. ハードウェアによる MPI 派生データ型通信の支援. 情報処理学会研究報告, 第 2006-ARC-170 巻, pp. 91–96, Nov. 2006.

[108] 宮部 保雄, 宮代 具隆, 北村 聡, 田邊 昇, 中條 拓伯, 天野 英晴. MPI 派生データ型通信支援機構の DIMMnet-2 への実装と評価. 先進的計算基板システムシンポジウム SACSIS'07 論文集, pp. 211–218, May. 2007.

[109] Voltaire. http://www.voltaire.com/.

[110] 濱田 芳博, 荒木 健志, 西 宏章, 田邊 昇, 天野 英晴, 中條 拓伯. bDais: DIMMnet-1/InfiniBand 間ルータの評価. 情報処理学会研究報告, 第 2004-ARC-159 巻, pp. 145–150, Sep. 2004.

[111] 西村信治. イーサネットの最新動向 -100Gb イーサネット他-. 先進的計算基盤システムシンポジウム SACSIS'07 チュートリアル・企業展示資料, May. 2007.

[112] Intel, Corp. *Intel® 5000X Chipset Memory Controller Hub (MCH) Datasheet*, Sep. 2006.

[113] Intel, Corp. *Intel® 5000P/5000V/5000Z Chipset Memory Controller Hub (MCH) Datasheet*, Sep. 2006.

論文目録

本研究に関する論文

公刊論文

1. 北村 聡, 濱田 芳博, 宮部 保雄, 伊澤 徹, 宮代 具隆, 田邊 昇, 中條 拓伯, 天野 英晴. DIMMnet-2 ネットワークインタフェースコントローラの設計と実装. 情報処理学会論文誌コンピューティングシステム Vol.46, No.SIG12 (ACS11), pp.13–26, Sep. 2005.

2. 北村 聡, 宮部 保雄, 中條 拓伯, 田邊 昇, 天野 英晴. メッセージパッシングモデルを支援するパケット受信機構の DIMMnet-2 への実装と評価. 情報処理学会論文誌コンピューティングシステム Vol.47, No.SIG12 (ACS15), pp.59–73, Sep. 2006.

3. 宮部 保雄, 北村 聡, 田邊 昇, 中條 拓伯, 天野 英晴. MPI 派生データ型通信支援機構の DIMMnet-2 への実装と評価. 情報処理学会論文誌コンピューティングシステム (掲載予定).

国際会議

1. Noboru Tanabe, Akira Kitamura, Tomotaka Miyashiro, Yasuo Miyabe, Tohru Izawa, Yoshihiro Hamada, Hironori Nakajo and Hideharu Amano. Preliminary Evaluation of a FPGA-based-Prototype of DIMMnet-2 Network Interface. In *Proceedings of the IEEE International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'05)*, Jan. 2005.

2. Yoshihiro Hamada, Hiroaki Nishi, Akira Kitamura, Noboru Tanabe, Hideharu Amano and Hironori Nakajo. A Packet Forwarding Layer for DIMMnet and its Hardware Implementation. In *Proceedings of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'05)*, pp.461–467, Jun. 2005.

3. Akira Kitamura, Yoshihiro Hamada, Yasuo Miyabe, Tetsu Izawa, Tomotaka Miyashiro, Konosuke Watanabe, Tomohiro Otsuka, Noboru Tanabe, Hironori Nakajo and Hideharu Amano. Evaluation of Network Interface Controller on DIMMnet-2 Prototype Board. In *Proceedings of the 6th IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'05)*, pp.778–780, Dec. 2005.

4. Noboru Tanabe, Akira Kitamura, Tomotaka Miyashiro, Yasuo Miyabe, Takeshi Araki, Zhengzhe Luo, Hironori Nakajo and Hideharu Amano. Hardware Support for MPI in DIMMnet-2 Network Interface. In *Proceedings of the IEEE International Workshop on Innovative Architecture for Future Generation High Performance Processors and Systems (IWIA'06)*, pp.73–82, Jan. 2006.

5. Tetsu Izawa, Konosuke Watanabe, Akira Kitamura, Yasuo Miyabe, Tomotaka Miyashiro and Hideharu Amano. Cooperative Simulation Environment of Hardware Plugged into a DIMM slot. In *Proceedings of the 13th International Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI'06)*, pp.79–84, Apr. 2006.

6. Tomotaka Miyashiro, Akira Kitamura, Masato Yoshimi, Noboru Tanabe, Hironori Nakajyo and Hideharu Amano. DIMMnet-2: A Reconfigurable Board Connected into a Memory Slot. In *Proceedings of the 16th IEEE International Conference on Field Programmable Logic and Applications (FPL'06)*, Aug. 2006.

7. Akira Kitamura, Yasuo Miyabe, Tomotaka Miyashiro, Noboru Tanabe, Hironori Nakajo and Hideharu Amano. Performance Evaluation on Low-Latency Communication Mechanism of DIMMnet-2. In *Proceedings of the 25th IASTED International Multi-Conference on Applied Informatics (AI'07)*, pp57–62, Feb. 2007.

8. Yasuo Miyabe, Akira Kitamura, Yoshihiro Hamada, Tomotaka Miyashiro, Tetsu Izawa, Noburu Tanabe, Hironori Nakajo and Hideharu Amano. Implementation and Evaluation of the Mechanisms for Low Latency Communication on DIMMnet-2. In *Proceedings of the 6th International Symposium on High Performance Computing (ISHPC-VI)* (掲載予定).

研究会ほか

1. 北村 聡, 伊豆直之, 田邊 昇, 濱田 芳博, 中條 拓伯, 渡邊 幸之介, 大塚 智宏, 天野 英晴. DIMMnet - 2 ネットワークインタフェースボードの試作. 情報処理学会研究報告, 第 2004-ARC-159 巻, pp.151–156, Sep. 2004.

2. 田邊 昇, 箱崎 博孝, 安藤 宏, 土肥 康孝, 中條 拓伯, 宮代 具隆, 北村 聡, 天野 英晴. メモリモジュール上での等間隔アクセス連続化の効果. 情報処理学会研究報告, 第 2004-ARC-162 巻, pp.139–144, Mar. 2005.

3. 北村 聡, 濱田 芳博, 宮部 保雄, 伊澤 徹, 宮代 具隆, 田邊 昇, 中條 拓伯, 天野 英晴. DIMMnet-2 ネットワークインタフェースコントローラの設計と実装. 先進的計算基盤システムシンポジウム SACSIS'05 論文集, pp.293–300, May. 2005.

4. 伊澤 徹, 渡邊 幸之介, 北村 聡, 宮部 保雄, 宮代 具隆, 天野 英晴. メモリスロット装着型ハードウェアの評価検証環境の構築. 情報処理学会研究報告, 第 2005-ARC-163 巻, pp.1–6, May. 2005.

5. 宮部 保雄, 北村 聡, 濱田 芳博, 宮代 具隆, 伊澤 徹, 田邊 昇, 中條 拓伯, 天野 英晴. DIMMnet-2 低遅延通信機構の実装と評価. 情報処理学会研究報告, 第 2005-ARC-163 巻, pp.7–12, May. 2005.

6. 宮代 具隆, 宮部 保雄, 伊澤 徹, 北村 聡, 箱崎 博孝, 田邊 昇, 中條 拓伯, 天野 英晴. DIMMnet-2 ネットワークインタフェースにおけるプリフェッチ機構の実装と評価. 情報処理学会研究報告, 第 2005-ARC-163 巻, pp.13–18, May. 2005.

7. 濱田 芳博, 北村 聡, 西 宏章, 田邊 昇, 天野 英晴, 中條 拓伯. DIMMnet 通信インタフェース用パケット伝送レイヤ. 情報処理学会研究報告, 第 2005-MPS-055 巻, pp.33–36, Jun. 2005.
8. 田邊 昇, 羅 微哲, 濱田 芳博, 中條 拓伯, 北村 聡, 宮代 具隆, 宮部 保雄, 天野 英晴. DIMM スロット装着型デバイス DIMMnet-2 の改良方針. 情報処理学会研究報告, 第 2005-ARC-164 巻, pp.127–132, Aug. 2005.
9. 北村 聡, 宮部 保雄, 中條 拓伯, 田邊 昇, 天野 英晴. メッセージパッシングモデルを支援するパケット受信機構の実装. 情報処理学会研究報告, 第 2005-ARC-165 巻, pp.39–44, Nov. 2005.
10. 北村 聡, 宮部 保雄, 中條 拓伯, 田邊 昇, 天野 英晴. メッセージパッシングモデルを支援するパケット受信機構の DIMMnet-2 への実装と評価. 先進的計算基盤システムシンポジウム SACSIS'06 論文集, pp.359–366, May. 2006.
11. 田邊 昇, 北村 聡, 宮部 保雄, 宮代 具隆, 天野 英晴, 羅 微哲, 中條 拓伯. DIMMnet-3 ネットワークインターフェースにおける MPI 支援機能. 情報処理学会研究報告, 第 2006-ARC-169 巻, pp.103–108, Sep. 2006.
12. 宮代 具隆, 宮部 保雄, 北村 聡, 田邊 昇, 中條 拓伯, 天野 英晴. DIMMnet-2 を用いた間接メモリアクセスの高速化. 情報処理学会研究報告, 第 2006-ARC-170 巻, pp.85–90, Nov. 2006.
13. 宮部 保雄, 宮代 具隆, 北村 聡, 田邊 昇, 中條 拓伯, 天野 英晴. ハードウェアによる MPI 派生データ型通信の支援. 情報処理学会研究報告, 第 2006-ARC-170 巻, pp.91–96, Nov. 2006.
14. 田邊 昇, 北村 聡, 宮部 保雄, 宮代 具隆, 天野 英晴, 羅 徴哲, 中條 拓伯. 主記憶以外に大容量メモリを有するメモリ/ネットワークアーキテクチャ. 情報処理学会研究報告, 第 2007-ARC-172 巻, pp.157–162, Mar. 2007.
15. 宮部 保雄, 宮代 具隆, 北村 聡, 田邊 昇, 中條 拓伯, 天野 英晴. MPI 派生データ型通信支援機構の DIMMnet-2 への実装と評価. 先進的計算基板システムシンポジウム SACSIS'07 論文集, pp.211–218, May. 2007.

その他

1. 北村 聡, 伊豆 直之, 伊澤 徹, 宮代 具隆, 宮部 保雄, 渡邊 幸之介, 大塚 智宏, 濱田 芳博, 田邊 昇, 中條 拓伯, 天野 英晴. FPGA を用いたメモリスロット装着型ネットワークインタフェースの設計. 第 12 回 FPGA/PLD Design Conference ユーザ・プレゼンテーション 論文集, pp.13-20, Jan. 2005. 優秀論文賞.

その他の論文

国際会議

1. Akira Kitamura, Konosuke Watanabe, Tomohiro Otsuka, Hideharu Amano. The evaluation of dynamic load balancing algorithm on RHiNET-2. In *Proceedings of the 15th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'03)*, pp.262–267, Nov. 2003.

研究会ほか

1. 北村 聡, 天野 英晴, 渡邊 幸之介, 大塚 智宏. PC クラスタ用ネットワーク RHiNET - 2 上における動的負荷分散アルゴリズムの評価. 情報処理学会研究報告, 第 2002-ARC-152 巻, pp.73–78, Mar. 2003.

2. 大塚 智宏, 渡邊 幸之介, 北村 聡, 原田 浩, 山本 淳二, 西 宏章, 工藤 知宏, 天野 英晴. 分散並列処理用ネットワーク RHiNET-2 の性能評価. 先進的計算基盤システムシンポジウム SACSIS'03 論文集, pp.45–52, May. 2003.

3. 大塚 智宏, 渡邊 幸之介, 北村 聡, 鯉渕 道紘, 山本 淳二, 西 宏章, 工藤 知宏, 天野 英晴. RHiNET プロジェクトの最終報告. 情報処理学会研究報告, 第 2004-ARC-158 巻, pp.31–36, May. 2004.

付 録A　要求発行レジスタ以外のUser Registerのビットフィールド

本章では, 要求発行レジスタ以外の User Register のビットフィールドについて述べる. 各レジスタのビットフィールドを図 A.1 に示す.

Controller Status　ネットワークインタフェースコントローラの内部状態を示すステータスは 20bit から構成される. 各ビットの詳細を表 A.1 に示す.

Primitive Counter　各プリミティブの系列ごとに 10bit のカウンタが用意されている (表 A.2). この領域へ書き込みを行うと, カウンタがリセットされる.

Prefetch Window Flag　Prefetch Window はプロセスごとに, 1 個当たり 512Byte の Window が 4 個設けられることから, フラグは Window 当たり 4bit 必要となり, Prefetch Window 全体で 16bit のフラグとなる (表 A.3).

Packet Counter　64bit のカウンタである. この領域へ書き込みを行うと, カウンタがリセットされる.

IPUSH Area Size　IPUSH 系プリミティブのパケット受信用に確保する SO-DIMM 領域のサイズ指定に使用する領域である. ホストプロセッサから書き込んだ値の下位 16bit が有効な値として設定される.

LH Buffer Next Write Address, LH Buffer Next Read Address　LH Buffer Next Read Address はホストプロセッサが次に LH Buffer を読出す位置を示すアドレス値 (Head ポインタ) が格納されるレジスタである. ホストから書き込んだ値の下位 12bit が有効な値として設定される.

LH Buffer Next Write Address は LH Buffer に次に書き込まれる位置を示すアドレス値 (Tail ポインタ) が格納されるレジスタである. LH Buffer にデータが書き込まれるたびに要求処理部によって更新される. ホストプロセッサから読み出した際には, 下位 12bit が有効な値となる.

Module State　Module State の詳細を表 A.4 に示す.

Status Initial Address　ホストプロセッサから書き込んだ値の下位 15bit が有効な値として Status Initial Address に設定される.

表 A.1　コントローラステータス

ビット	意味
0	要求発行完了フラグ (0：未発行の要求なし, 1：未発行の要求あり)
1	SO-DIMM 初期化完了フラグ (0：初期化未完了, 1：初期化完了)
2	Request FIFO Almost Full フラグ (0：Request FIFO に余裕あり, 1：Request FIFO に余裕なし)
3	Status Area Full フラグ (0：Status Area Not Full, 1：Status Area Full)
4	Status Area Empty フラグ (0：Status Area Not Empty, 1：Status Area Empty)
5〜7	SO-DIMM Write Status (Write Window #0)
5	SO-DIMM への書き込み完了フラグ (0：未完了, 1：完了)
6	SO-DIMM への書き込みの際に (ソースアドレス)+(書き込みサイズ) が Write Window の末尾を超えたかどうかを示すフラグ (0：超えていない, 1：超えた)
7	SO-DIMM への書き込みの際に (デスティネーションアドレス)+ (書き込みサイズ) が自 SO-DIMM 領域の末尾を超えたかどうかを示すフラグ (0：超えていない, 1：超えた)
8〜10	SO-DIMM Write Status (Write Window #1)
11〜13	SO-DIMM Write Status (Write Window #2)
14〜16	SO-DIMM Write Status (Write Window #3)
17	Receive Controller Status (0：データ受信処理時に受信領域を超えていない, 1：超えた)
18〜19	Receive Controller の状態
2’b00	正常な状態
2’b01	EOP を受信する前に次のパケットの 1st DW を受信
2’b10	hflag とヘッダのライン数が合わない
2’b11	不正な RVL 系パケットを受信 (データ部が付加されているなど)

Status Area Size　パケット受信ステータス用に確保するバッファのサイズ指定に使用するレジスタである. ホストプロセッサから書き込んだ値の下位 7bit が有効な値として, Status Area Size に設定される. この 7bit の値と, それによって設定されるバッファサイズの関係を表 A.5 に示す.

Status Next Write Address, Status Next Read Address　Status Next Read Address はホストプロセッサが次にパケット受信ステータスを読み出す位置を示すアドレス (Head ポインタ) を格納するレジスタである. ホストから書き込んだ値の下位 15bit が有効な値として設定される.

Status Next Write Address はパケット受信ステータスが次に書き込まれる位置を示すアドレス (Tail ポインタ) を格納するレジスタである. パケット受信ステータスが書き込まれるたびに要求処理部によって更新される. ホストプロセッサから読み出した際には, 読み出した値の下位 15bit が有効な値となる.

Command Ex Flag　この領域に 1 を書き込むと Command Ex が有効になる.

表 A.2 Primitive Counter

ビット	内容
0〜9	BOTF 用カウンタ
10〜19	VL 系プリミティブ用カウンタ
20〜29	VS 系プリミティブ用カウンタ
30〜39	RVL 系プリミティブ用カウンタ
40〜49	RVS 系プリミティブ用カウンタ
50〜59	IPUSH 系プリミティブ用カウンタ

表 A.3 Prefetch Window Flag

ビット	Prefetch Window
0〜3	PW#0
4〜7	PW#1
8〜11	PW#2
12〜15	PW#3

表 A.4 Module State の詳細

ビット	意味
0〜16	Window Controller 内部の各モジュールのステート
17〜36	Receive Controller 内部の各モジュールのステート
37〜50	Write Unit 内部の各モジュールのステート
51〜57	Prefetch Unit 内部の各モジュールのステート

表 A.5 Status Area Size

設定値	Status Area Size[Byte]
7’b0000000	default 値
7’b0000001	64
7’b0000010	128
7’b0000100	256
7’b0001000	512
7’b0010000	1024(default 値)
7’b0100000	2048
7’b1000000	4096
上記以外	default 値

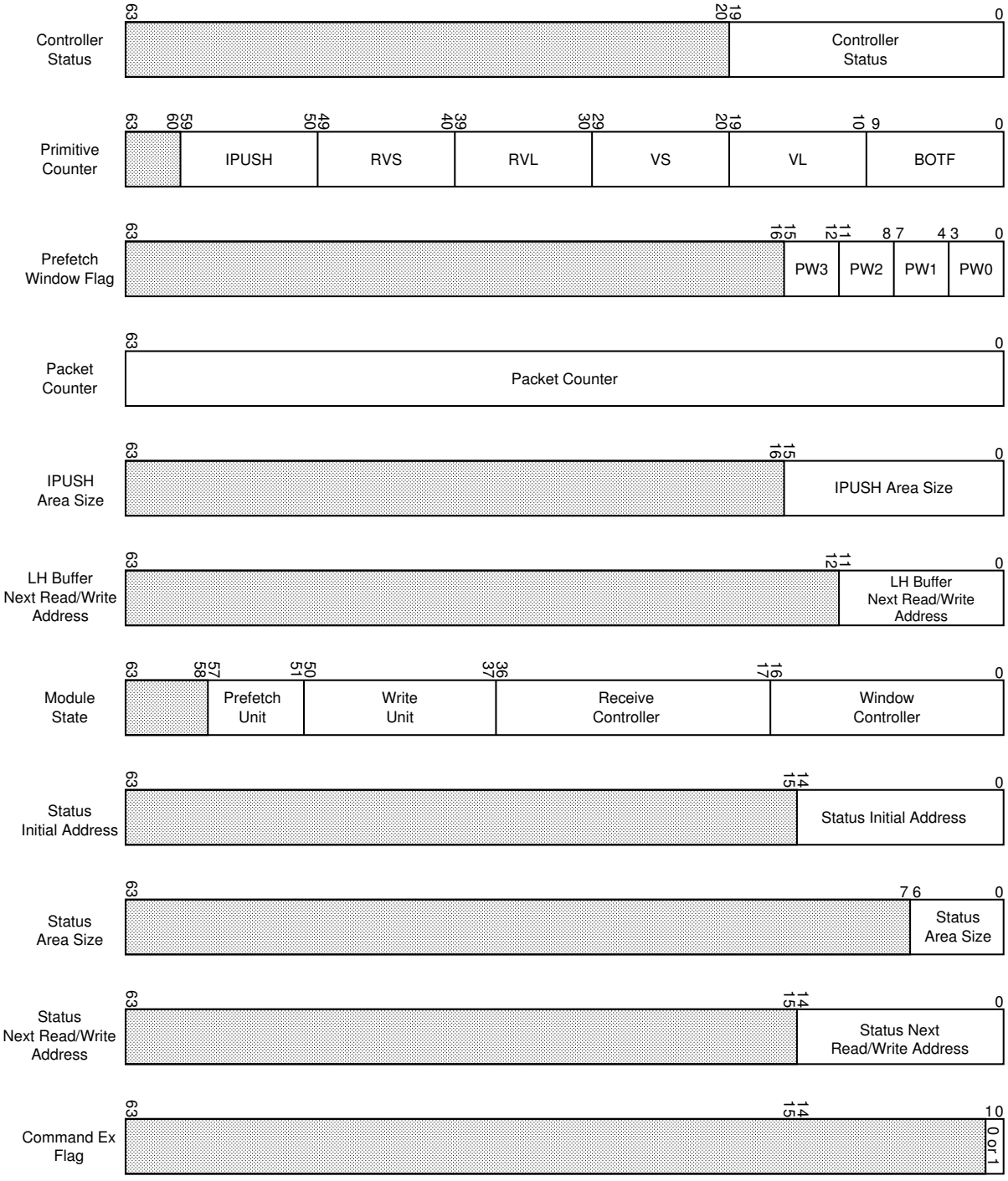


図 A.1　ユーザレジスタのビットフィールド

付 録B　System Registerのビットフィールド

本章ではSystem Register内の各レジスタのビットフィールドについて述べる. 各レジスタのビットフィールドを図B.1に示す.

SO-DIMM Init　1bit目に1を書き込むことでSO-DIMMに対するアクセスが可能になる. システム起動中に1度行えば, システムの電源を落とすまで再度有効化する必要はない.

SO-DIMM Capacity　SO-DIMM Capacityに書き込んだ値の下位6bitを有効な値として, SO-DIMM 1枚当たりの容量に設定する.

MTU　ホストからMTUに書き込んだ値の下位5bitを有効な値として, MTU値に設定する.

SMA Software Reset　SMAをリセットするのに用いられる. 書き込み操作によってリセットがかかるため, 書き込むデータの内容は意味を持たない.

SMA LID　読み出した値の下位16bitがLIDで, 上位2bitはSWIFの状態を示す. 上位2bitが2'b11であれば通信が可能な状態であることを示す.

LID　LIDに書き込んだ値の下位16bitが有効な値としてLIDに設定される.

PGID0, PGID1　PGID0, PGID1に書き込んだ値の下位8bitがそれぞれのプロセスのPGIDとして設定される.

Controller Reset　読み出し, 書き込みに関わらず, Controller Resetにアクセスすると, ネットワークインタフェースコントローラにリセットがかかる.

AMT Address Table Interface　全30bitから構成され, 下位12bitがAMT Address Tableにセットする値, 残りの18bitで送信側プロセスのLIDとWIDを指定する.

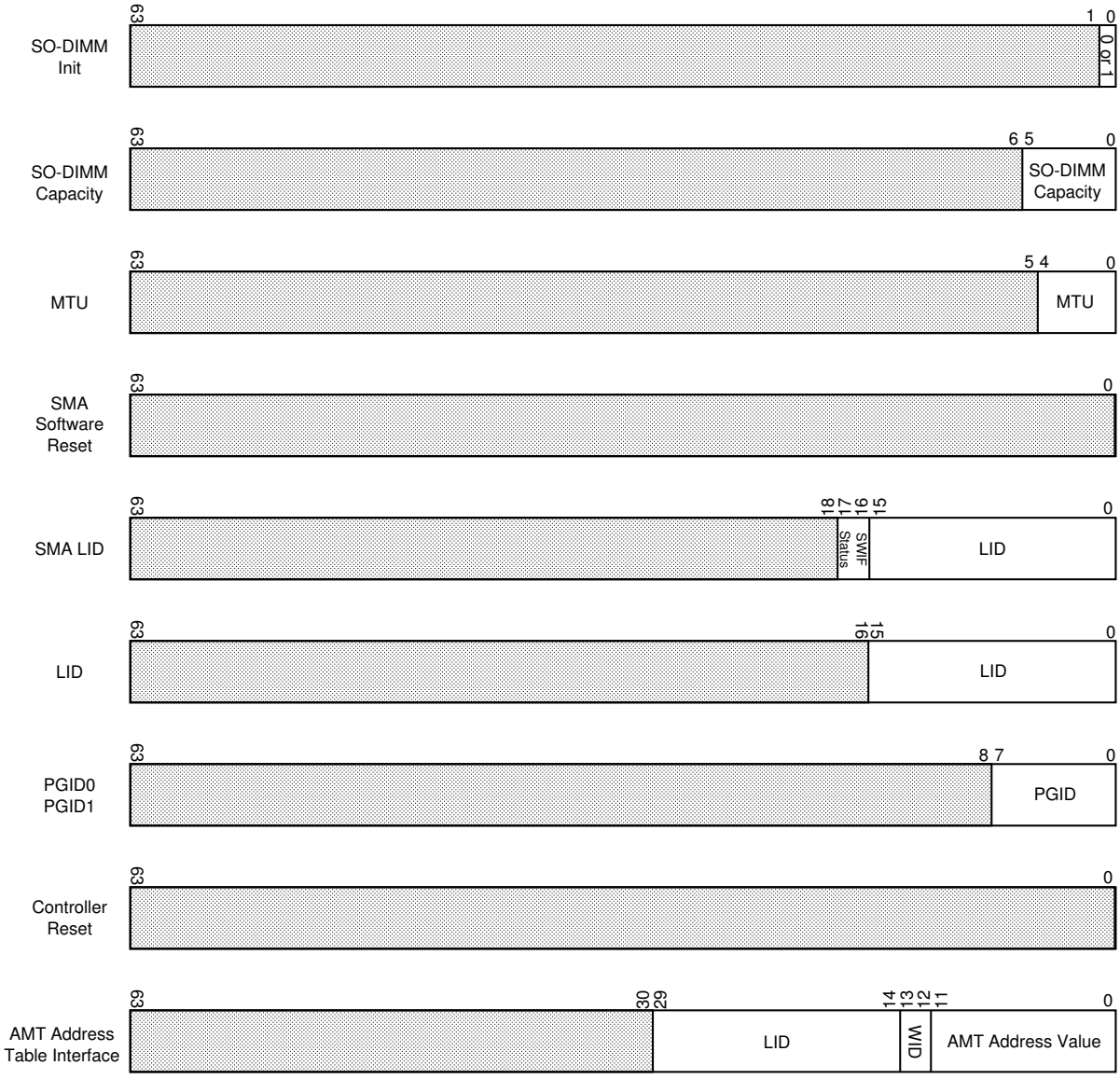


図 B.1　システムレジスタのビットフィールド

付 録C　DIMMnet Shell マニュアル

本章は, DIMMnet-2 ネットワークインタフェースコントローラの開発に際して, 実機を用いた際に使用するデバッグツールである dsh (DIMMnet Shell) のマニュアルである.

C.1　概要

dsh は対話的に DIMMnet-2 をホスト側から操作することができ, ネットワークインタフェースコントローラの内部状態の読み出しやプリミティブの発行が可能である. DIMMnet-2 を装着したノードが複数ある場合に, それぞれのノードで dsh を起動させれば通信処理も実行することができる.

C.2　ファイル構成

dsh は図 C.1 に示されるファイルから構成される.

utils ディレクトリ以下で make を実行すると, dnet2_func.o, dsh.o, dsh_func.o が生成され, 最終的に dsh の実行ファイルが出力される.

C.3　dsh の実行

dsh は図 C.2 に示される書式で実行する. なお, 実行の際には root 権限が必要である.

すると, “dsh >” というプロンプトが表示され, dsh に対して, 様々なコマンドを発行することができる. dsh で利用可能なコマンドを表 C.1 にまとめる.

C.3.1　evpbuf_read

evpbuf_read の書式と実行例を図 C.3 に示す(注 1).

evpbuf_read は LH Buffer の最初から Offset だけ離れたアドレスから Size バイト読み出し, 出力する.

Offset や Size の値は “0x” を先頭に付けると, 16 進数として扱われる. また, “0” を先頭に付けると, 8 進数として扱われる. ただし, 0x を付けずに 55ff などと指定すると, 10 進数として解釈可能な 55 の部分だけが有効な値となり, ff は全く考慮されない(注 2).

(注 1)LH Buffer は LHS 機構の開発当初, Envelope Buffer という名称であったことから, コマンド名に evpbuf という名前が使用されている

(注 2)引数の解釈に strtoul 関数を使用しているため

```
header2：DIMMnet-2 を操作する上で必要となる基本的な変数や関数の定義ファイルが
  |        置かれているディレクトリ
  +-header.h
  +-dnet2.h
  +-dnet2_func.h

utils   ：dsh 本体が置かれているディレクトリ
  |
  +-dsh.c
  +-dsh.h
  +-dsh_func.c
  +-dnet2_func.c
  +-Makefile

doc     ：本マニュアルが置かれているディレクトリ
  |
  +-dsh.manual.tex
```

図 C.1　dsh を構成するファイル

C.3.2 h (or help)

h (or help) の書式と実行例を図 C.4 に示す. 図 C.4 には evpbuf_read のヘルプしか表示されていないが, 実際には全コマンドのヘルプが表示される.

C.3.3 llcm_read

llcm_read の書式と実行例を C.5 に示す.

llcm_read は LLCM の最初から Offset だけ離れたアドレスから Size バイト読み出し, 出力する. Offset の書式は evpbuf_read と同様である.

C.3.4 llcm_write

llcm_write の書式と実行例を図 C.6 に示す.

llcm_write は LLCM の最初から Offset だけ離れた番地から Iteration 回, Value (32bit) で指定した値を書き込む. その際に, Value の値を Increment で指定した値だけインクリメントする. 図 C.6 の例 1 では LLCM の最初から値を 10 回書き込む. 書き込まれる値は 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 となる. 引数に指定する値の書式は evpbuf_read の Offset と同様である. また, パラメータの順番を入れ換えても動作する.

```
[書式]
dsh [総プロセス数] [自プロセスの PID] [自プロセスの WID]

[例 (総プロセス数=2, 自プロセスの PID=1, 自プロセスの WID=0 の場合)]
# ./dsh 2 1 0
```

図 C.2　dsh の書式と実行例

表 C.1　dsh で利用可能なコマンド

Command	Discription
evpbuf_read	LH Buffer の読み出し
h (or help)	ヘルプの表示
llcm_read	LLCM の読み出し
llcm_write	LLCM への書き込み
prim	DIMMnet-2 への要求発行
pw_read	Prefetch Window の読み出し
q (or quit)	dsh の終了
rllcm_write	リモートの LLCM への書き込み
sreg_read	System Register の読み出し
sreg_write	System Register への書き込み
ureg_read	User Register の読み出し
ureg_write	User Register への書き込み
v (or version)	dsh のバージョンの表示
ww_write	Write Window への書き込み

C.3.5　prim

prim の書式と実行例を図 C.7 に示す.

PRIM で指定されたプリミティブを dt 以降の引数で指定したパラメータで実行する. PRIM に指定できる値は “NOP’, “VL”, “VLS”, “VLI”, “VS”, “VSS”, “VSI”, “RVL”, “RVLS”, “RVLI”, “RVS”, “RVSS”, “RVSI”, “IPUSH”, “LHSv1”, “LHSv2” である.

dt 以降のパラメータは表 4.5 などで定められている値をそのまま使用可能である. 例えば, dt に 2 を指定した場合, Dtype は 4 としてプリミティブが発行される.

llcm_write 同様, dt 以降のパラメータの順番は特に決まっておらず, また, “0x” を数値の先頭に用いれば 16 進数表記も使用可能である. しかし, VL 実行時の dt や it のように, そのプリミティブで使用しないパラメータであっても, 何らかの値を指定しておく必要がある.

実行すると 16 進数表記でどのような要求が発行されたかを表示する. プリミティブの完了を待たないため, プリミティブの完了は ureg_read (後述) でしかるべきレジスタを読み出して確認する

```
[書式]
evpbuf_read [Offset] [Size(Byte)]

[例 (LH Buffer の最初から 64Byte 読み出す)]
dsh > evpbuf_read 0 64
#  ADDR     7 6 5 4 3 2 1 0  F E D C B A 9 8
#4216c000: 0000000000000000 0000000000000000
#4216c010: 0000000000000000 0000000000000000
#4216c020: 0000000000000000 0000000000000000
#4216c030: 0000000000000000 0000000000000000
```

図 C.3 evpbuf_read の書式と実行例

必要がある.

C.3.6 pw_read

pw_read の書式と実行例を図 C.8 に示す.

pw_read は PW#で指定した番号の Window の最初の番地から Size バイト読み出し, 出力する. SO-DIMM から読み出す機能は持っていないため, あらかじめ prim で SO-DIMM から Prefetch Window にデータを読み出しておく必要がある. Size の書式は evpbuf_read の Offset と同様である.

C.3.7 q (or quit)

q または quit と入力すると, dsh を終了する.

C.3.8 rllcm_write

rllcm_write の書式と実行例を図 C.9 に示す. rllcm_write は Dclid で指定した LID のプロセスの LLCM に対して, LLCM の最初から Offset だけずれた番地に Value (64bit) で指定した値を書き込む. データサイズは 8Byte 限定である.

C.3.9 sreg_read

sreg_read の書式を図 C.10 に示す.

sreg_read は System Register の Option で指定した領域を読み出し, 表示する. System Register から読み出せる領域は SMA LID のみであるため, 指定可能なオプションは “sma_lid” のみである.

```
[書式]
h
 または
help

[例]
dsh > h
evpbuf_read  : LH Buffer Read
             - evpbuf_read [Offset] [Size(Byte)]
```

図 C.4　help の書式と実行例

```
[書式]
llcm_read [Offset] [Size(Byte)]

[例 (LLCM の最初から 64Byte 読み出す)]
dsh > llcm_read 0 64
#  ADDR     7 6 5 4 3 2 1 0  F E D C B A 9 8
#4215c000: 0000000000000000 0000000000000000
#4215c010: 4000000040000000 0000000000000000
#4215c020: 0000000000000000 0000000000000000
#4215c030: 0000000000000000 0000000000000000
```

図 C.5　llcm_read の書式と実行例

C.3.10　sreg_write

sreg_write の書式と実行例を図 C.11 に示す.

sreg_write は System Register の Option で指定した領域に対して, Value で指定した値を書き込む. Option に指定可能な値を表 C.2 に示す.

ctl_reset のように, “書き込む値に関係なく, 書き込みという処理に対して動作する” 領域であっても, Value に何らかの値を指定する必要がある. また, 指定するパラメータにはこれまでのコマンドと同様に 16 進数を使用することもできる.

C.3.11　ureg_read

ureg_read の書式と実行例を図 C.12 に示す.

```
[書式]
llcm_write from=[Offset] it=[Iteration] val=[Value] inc=[Increment]

[例 1 (LLCM の最初から順番に 0 から 9 までの数値を書き込む)]
dsh > llcm_write from=0 it=10 val=0 inc=1

[例 2 (LLCM の最初から順番に 0xaaaa0000〜0xaaaa0009 までの値を書き込む) ]
dsh > llcm_read 0 64
#  ADDR     7 6 5 4 3 2 1 0  F E D C B A 9 8
#4215c000: 0000000000000000 0000000000000000
#4215c010: 4000000040000000 0000000000000000
#4215c020: 0000000000000000 0000000000000000
#4215c030: 0000000000000000 0000000000000000

dsh > llcm_write from=0 it=10 val=0xaaaa0000 inc=1
dsh > llcm_read 0 64
#  ADDR     7 6 5 4 3 2 1 0  F E D C B A 9 8
#4215c000: aaaa0001aaaa0000 aaaa0003aaaa0002
#4215c010: aaaa0005aaaa0004 aaaa0007aaaa0006
#4215c020: aaaa0009aaaa0008 0000000000000000
#4215c030: 0000000000000000 0000000000000000
```

図 C.6　llcm_write の書式と実行例

ureg_read は User Register の Option で指定した領域を読み出し, 表示する. Option に指定可能な値を表 C.3 に示す.

module_state 以外の場合は図 C.12 のように 64bit の値が読み出されるだけであるが, module_state のみ図 C.13 に示す通り, 表示形式が異なる. Option に module_state を指定した場合は, 64bit の値とともに, 各モジュールのステートも表示する. ステートの名称は Core Logic の Verilog 記述で用いられている名称に準じている.

C.3.12　ureg_write

ureg_write の書式を図 C.14 に示す.

ureg_write は User Register の Option で指定した領域に対して, Value で指定した値を書き込む. Option に指定可能な値を表 C.4 に示す.

prim_counter_reset のように, “書き込む値に関係なく, 書き込みという処理に対して動作する” 領域であっても, Value に何らかの値を指定する必要がある.

```
[書式]
prim PRIM dt=[Dtype] it=[Iteration] dw=[Dwid] dl=[Dclid] ef=[Eflag]
          size=[Size(Byte)] src=[Src Addr] dst=[Dst Addr]

[例 (Write Window の 0x400 の位置から SO-DIMM の 0x600 番地へ 0x200Byte を VS で書き
込む)]
dsh > prim VS dt=0 it=0 dw=0 dl=0 ef=0 size=0x200 src=0x400 dst=0x600
cmd:60000000400_20000000008
```

図 C.7　prim の書式と実行例

```
[書式]
pw_read [PW#(0-3)] [Size(Byte)]

[例 (Prefech Window の 0 番目の Window から 64Byte 読み出す)]
dsh > pw_read 0 64
#  ADDR     7 6 5 4 3 2 1 0  F E D C B A 9 8
#4214c000: 0000000000000000 0000000000000000
#4214c010: 0000000000000000 0000000000000000
#4214c020: 0000000000000000 0000000000000000
#4214c030: 0000000000000000 0000000000000000
```

図 C.8　pw_read の書式と実行例

C.3.13　v (or version)

v または version と入力すると, 図 C.15 のように dsh のバージョンを表示する.

C.3.14　ww_write

ww_write の書式と実行例を図 C.16 に示す.

ww_write は WW#で指定した番号の Window の最初から Offset だけ離れた番地から Iteration で指定された回数だけ Value (32bit) で指定された値を書き込む. その際に, Value の値を Increment で指定した値だけインクリメントしながら書き込む. 使用方法は llcm_write とほぼ同じである.

```
[書式]
rllcm_write [Dclid] [Offset] [Value]

[例 (CLID が 3 のノードの LLCM のオフセットが 200 の位置に 8 を書き込む)]
dsh > rllcm_write 3 200 8
```

図 C.9　rllcm_write の書式と実行例

```
[書式]
sreg_read [Option]

[例 (System Register の SMA LID を読み出す)]
dsh > sreg_read sma_lid
####### Get LID from sma #######
#  ADDR     7 6 5 4 3 2 1 0  F E D C B A 9 8
#4218c300: 0000000000000000
```

図 C.10　sreg_read の書式と実行例

```
[書式]
sreg_write [Option] [Value]
ただし, [Option] が‘‘amt’’ の場合は
sreg_write amt addr=[LLCM Address] wid=[Wid] lid=[Lid]

[例 (コントローラをリセットする)]
dsh > sreg_write clt_reset 1
```

図 C.11　sreg_write の書式と実行例

表 C.2 sreg_write の [Option] で指定可能な値

Option	Discription
amt	Address Management Table への値の設定
sdimm_init	SO-DIMM の有効化
sdimm_capacity	SO-DIMM の容量の設定
mtu	MTU の設定
sma_reset	SMA のリセット
lid	自ノードの LID の設定
pgid0	WID=0 のプロセスの PGID の設定
pgid1	WID=1 のプロセスの PGID の設定
ctl_reset	コントローラのリセット

```
[書式]
ureg_read [Option]

[例 (コントローラステータスを読み出す)]
dsh > ureg_read ctl_status
####### Controller Status #######
#  ADDR     7 6 5 4 3 2 1 0  F E D C B A 9 8
#4217c200: 0000000000000832
```

図 C.12 ureg_read の書式と実行例

表 C.3 ureg_read の [Option] で指定可能な値

Option	Discription
ctl_status	コントローラステータスの読み出し
prim_counter	Primitive Counter の読み出し
pw_flag	Prefetch Flag の読み出し
pkt_counter	Packet Counter の読み出し
evpbuf_nwa	LH Buffer の Tail ポインタの読み出し
module_state	コントローラ内部の各モジュールのステートの読み出し
status_nwa	パケット受信ステータスの Head ポインタの読み出し

```
dsh > ureg_read module_state
####### Module Status #######
#  ADDR     7 6 5 4 3 2 1 0  F E D C B A 9 8
#4217c800: 0000422200020001
Window Controller
   Accepter : P_RPATH_NOP
   Executer : P_BOTF_WAIT
Receive Controller
  IDLE
PWrite
  IDLE
Write Unit rctl_stat
  IDLE
Write Unit wctl_stat
  IDLE
Write Unit wu_stride
  IDLE
Write Unit wu_ipush
  M_IDLE
Prefetch Unit
  ST_R_IDLE
Prefetch Unit pu_ctl
  ST_IDLE
Prefetch Unit pu_write
  ST_FIFO_IDLE, ST_PW_IDLE
```

図 C.13　ureg_read で module_state を指定した場合の表示

表 C.4　ureg_write の [Option] で指定可能な値

Option	Discription
prim_counter_reset	Primitive Counter のリセット
pkt_counter_reset	Packet Counter のリセット
ivs_area_size	IPUSH の受信領域のサイズの設定
evpbuf_nra	LH Buffer の Head ポインタの設定
status_initial_addr	パケット受信ステータスの初期アドレスの設定
status_area_size	パケット受信ステータスの領域のサイズの設定
status_nra	パケット受信ステータスの Tail ポインタの設定

```
[書式]
ureg_write [Option] [Value]

[例 (Primitive Counter のリセット)]
dsh > ureg_read prim_counter
####### Primitive Counter #######
#  ADDR     7 6 5 4 3 2 1 0  F E D C B A 9 8
#4217c300: 0000000000100001
BOTF  : 1
VL    : 0
VS    : 1
RVL   : 0
RVS   : 0
IPUSH : 0

dsh > ureg_write prim_counter_reset 1
####### Primitive Counter Reset #######

dsh > ureg_read prim_counter
####### Primitive Counter #######
#  ADDR     7 6 5 4 3 2 1 0  F E D C B A 9 8
#4217c300: 0000000000000000
BOTF  : 0
VL    : 0
VS    : 0
RVL   : 0
RVS   : 0
IPUSH : 0
```

図 C.14　ureg_write の書式と実行例

```
[書式]
v
または
version

[例]
dsh > v
DIMMnet-2 Shell version 2.1a released by Yomi
```

図 C.15　version の実行例

```
[書式]
ww_write ww=[WW#(0-3)] from=[Offset] it=[Iteration] val=[Value]
         inc=[Increment]

[例 (Write Window の 0 番目の Window に 0xaaaa0000〜0xaaaa0009 を書き込む)]
dsh > ww_write ww=0 from=0 it=10 val=0xaaaa0000 inc=1
```

図 C.16　ww_write の書式と実行例