修士論文

分散システム上で共有メモリ環境を提供するネットワークインタフェースカード

指導教官　　富田 眞治 教授

京都大学大学院情報学研究科
修士課程通信情報システム専攻

鳥崎 唯之

平成 14 年 2 月 8 日

分散システム上で共有メモリ環境を提供するネットワークインタフェースカード

鳥崎 唯之

内容梗概

ネットワークで接続された分散システムにおいて並列処理を行う、所謂分散並列システムは、スケーラビリティ、コストパフォーマンスの点において、従来の並列マシンよりも優れている。しかし、現在の分散システムでは、通信機能が付加的であるため、通信コストが高い。このため、従来の並列マシンと同等の処理能力を得るためには、その通信コストがネックとなってくる。

そこで、我々は、分散システムの長所を活かしつつ、通信コストを抑えるため、分散システム上で共有メモリを実現するネットワークインタフェースカードの開発を行った。本稿では、我々の開発したネットワークインタフェースカードと、そのネットワークインタフェースカードを用いて実装した通信機構について述べる。

開発したネットワークインタフェースカードは、FPGA と、高速ネットワークへの入出力、シンクロナス SRAM 等によって構成されている。我々は、低遅延の通信を実現するために、通信機構と MBus インタフェース、ネットワークインタフェース等の各種インタフェースを設計し、それらを FPGA 上に実装した。これにより、ネットワークインタフェースカード上のシンクロナス SRAM を分散共有メモリとして実現するシステムコールを介さない通信が可能となった。

このネットワークインタフェースカードは、システムバスに直接接続され、仮想記憶管理のための MMU(Memory Management Unit) によるメモリ保護機構を利用することで、通信資源が通常のメモリアクセスと同様に保護される。また、物理アドレスをもとに、リモートメモリへのアクセスの必要性を判断し、自律的に通信を起動する。これらの仕組みにより、ユーザレベルの通信を可能とし、システムコールとソフトウェアオーバヘッドを削減することができる。

開発したネットワークインタフェースカードで、測定を行った結果、リモートノードへのリード、ライトにそれぞれ 4.12μs、3.70μs を要した。これは、細粒度の並列分散処理を行うために必要なμs 単位でのリモートアクセスを実現しており、このネットワークインタフェースカードによって、分散システムにおいて共有メモリを実現し、並列分散処理を行うことが可能であることが分かった。

The Network Interface Card that Supports Shared Memory Environment on the Distributed System

Yuishi Torisaki

Abstract

The distributed parallel processing system, which executes parallel programs on a distributed system interconnected with a commodity local area network, is superior to a traditional parallel system with respect to both the scalability and the cost performance. However, in return for the universality of its communication system, the cost of communication is too expensive to run parallel programs efficiently.

Accordingly, we developed a network interface card(NIC) that provides a mean for the low latency fine grain communication through a shared memory environment on the distributed system. With this card,we can reduce the cost of communication without loosing benefits of the distributed system like scalability and autonomy. In this thesis, we describe the result of our development of our network Interface Card.

This NIC consists of a Field Programmable Gate Array(FPGA), a interface with high-speed network, a synchronous SRAM for cache and so on. To realize such a fast communication system, we designed the communication mechanism, a system bus interface, a network interface and others and implemented them onto the FPGA. In this implementation we use SSRAM on each NIC as a part of the distributed shared memory and a cache for the remote data.

Being connected with the system bus of the node,this NIC enables user level communication without system calls because the NIC itself distinguishes remote memory accesses from local memory accesses,and executes communication with remote nodes if necessary.

As a result of the performance measurement of this NIC,we found that the costs of Read and Write to a remote node's memory are 4.12μs and 3.70μs respectively. This result shows that this communication system with our NIC meets the requirement of the communication cost in fine-grain parallel processing.

分散システム上で共有メモリ環境を提供するネットワークインタフェースカード

目次

第 1 章　はじめに　1

第 2 章　Computer Colony　3

2.1　コンピュータ・コロニーの背景と目標 3

2.1.1　背景 3

2.1.2　コンピュータ・コロニーの目標 3

2.2　コンピュータ・コロニーのシステムモデル 4

2.2.1　コンピュータ・コロニーのシステム 4

2.2.2　システム構成 5

2.3　コンピュータ・コロニーの通信機構 5

2.3.1　コンピュータ・コロニーで求められる通信機構 5

2.3.2　コンピュータ・コロニーにおける分散共有メモリ 6

2.3.3　分散 OS Colonia の共有メモリ管理 7

2.3.4　Colony の共有メモリ空間 8

2.4　プロトタイプ・ハードウェア 9

2.5　関連研究 9

2.5.1　メッセージベースの通信機構 10

2.5.2　分散共有メモリベースの通信機構 10

第 3 章　ネットワークインタフェースカードの開発　13

3.1　ネットワークインタフェースカード 13

3.1.1　ネットワークインタフェースカードの概要 13

3.1.2　ネットワークインタフェースボードの構成 13

3.2　通信ボードコア 14

3.2.1　通信ボードコアの概要 15

3.2.2　通信ボードコアの仕様 16

3.3　通信ボードコアの実装 19

3.3.1　実装に当たって 19

3.3.2　通信ボードコアの構成 21

3.3.3 通信ボードコアの動作 21
3.3.4 MBus インタフェース部の実装 24
3.3.5 送受信部の実装 26
3.3.6 シンクロナス SRAM インタフェース部の実装 31

第 4 章 ネットワークインタフェースカードの基本性能評価 33

4.1 評価環境 33
4.2 ローカルメモリアクセス 33
4.2.1 システムコールを用いたメモリアクセス 34
4.2.2 システムコールを用いないメモリアクセス 34
4.2.3 ボード上での処理にかかる時間 35
4.3 リモートメモリアクセスに要する時間の測定 35
4.3.1 仮想リモートメモリアクセス 35
4.3.2 リモートメモリアクセス 37

第 5 章 低信頼ネットワーク環境のためのプロトコルと拡張 38

5.1 通信エラーの対策 38
5.1.1 32bit 版 FC Interface モジュール 38
5.1.2 二回送信版 FC Interface モジュール 40
5.2 システムコール削減に関する考察 42
5.3 通信処理時間に関する考察 43

第 6 章 まとめと今後の展望 45

謝辞 47

参考文献 48

付録 A-1

第 1 章　はじめに

近年のネットワークの高速化と、マイクロプロセッサの単体性能の劇的な向上及び低価格化により、分散システムの形態をとりながら、高い計算能力を持つ計算機システム、クラスタシステムへの注目が高まっている。クラスタシステムは、従来のスーパーコンピューティング分野の流れと分散コンピューティング分野の流れが融合してできた計算機システムである。これまで、スーパーコンピューティング分野の研究が、主にピーク性能の追求を目的としてきたのに対して、分散コンピューティング分野の研究は、能率や快適性を目的としてきた。クラスタシステムの将来的な目標は、これらの二つの流れの長所をあわせ持つシステムを実現することである。

しかし、スーパーコンピューティング分野からのアプローチとしての種々のクラスタシステムにおいては、クラスタリングによる分散化のメリットは、価格性能比の良さだけにとどまっている。また、パフォーマンスを重視する考え方から、その利用形態も、従来のスーパーコンピュータのそれと大差ないものであり、物理的形態こそ分散システムではあるが、それ以外の性質はまだ集中システムのそれに近いものとなっている。このため分散コンピューティングの利点である快適性には程遠かった。

また、分散コンピューティング分野からのアプローチとしてのクラスタシステムにおいては、主にソフトウェアオーバーヘッドに起因する通信コストの問題により、ネットワークのバンド幅を有効に利用できず、ネットワークで結合された計算機資源の有効利用を実現するまでには至っていない。また、こういったクラスタシステムにおいては、プロセスレベルのマルチユーザ環境はある程度実現されているものの、システム全体にわたるスケジューリングが不十分であることも、計算機資源の有効利用を実現できない一因である。

そこで、我々の研究室では、既存の分散システムに対し、ハードウェア、ソフトウェア両面から新規開発を行うことにより、従来のクラスタシステムにおける通信コストとマルチユーザ環境対応の問題を解決する「コンピュータ・コロニー」(Computer Colony) の開発を行っている。

コンピュータ・コロニーにおいては、LAN 程度の分散環境において細粒度の並列処理を行うことにより、ネットワークで接続された計算機資源を有効利用し、高い性能とマルチユーザにとって快適な環境を実現することを目指す。こ

のために、ハードウェアにおいては、分散環境で、細粒度の並列処理を可能にする高速、かつ低遅延の通信を行う通信機構を設計し、その機構を実現するネットワークインタフェースカードの開発を行っている。また、ソフトウェアからのアプローチとして、適切なスケジューリングとプロセス移送により、全てのユーザに計算資源を公平に分配すると共に、計算機資源を有効活用する分散オペレーティングシステムを開発してきた。

本稿では、このうち通信コストの問題を解決し、分散環境において比較的細粒度の並列処理を可能にする通信機構を実現するネットワークインタフェースカードの開発とそのネットワークインタフェースカードに実装した通信機構について述べる。

開発したネットワークインタフェースカードは、システムバスに直接接続され、仮想管理記憶のための MMU(Memory Management Unit) によるメモリ保護機構を用いることにより、通信資源をユーザに対して安全に開放することが出来る。また、現在行われているメモリアクセスが、ローカルのメモリへ対するものであるか、リモートへ対するものであるかを判断し、必要に応じて、リモートノードへのアクセスを行う。これにより、通信を行う際に、オペレーティングシステムの関与しないユーザレベル通信を行うことが可能となり通信コストを削減することが出来ると共に、インタフェースカード上に搭載されているシンクロナス SRAM を、共有メモリとして提供し、細粒度の分散並列処理を可能とする。

本稿の構成は以下の通りである。まず、第二章において我々の研究室で開発中のクラスタシステム「コンピュータ・コロニー」について概要を示し、ネットワークインタフェースカードに求められる機能について述べる。次に、第三章において、開発したネットワークインタフェースカードについて詳細を示し、第四章において、そのネットワークインタフェースカードを用いた測定評価について述べる。続いて第五章において、今回の測定評価より得られた結果について考察を行い、第六章においてまとめを行う。

第 2 章　Computer Colony

本章では、我々の研究室で開発しているクラスタシステム「コンピュータ・コロニー」[1][2] について、背景とその目標、システムモデルについて述べ、コンピュータ・コロニーに求められる通信機構について述べる。さらに、コンピュータ・コロニーの実現のために、我々が開発しているプロトタイプ・ハードウェアについて述べ、そのネットワークインタフェースカードについて構成を述べる。

2.1　コンピュータ・コロニーの背景と目標

2.1.1　背景

第 1 章でも述べたように、マイクロプロセッサの高性能化と低価格化、ネットワークの普及により、分散システムにおいて並列処理を行うクラスタシステムが注目されてきた。分散システムにおいては、低価格化と同時に、そのスケーラビリティも利点の一つと挙げられている。

しかし、分散システムから派生したクラスタシステムは、分散システムのもつこういった長所を有する反面、通信コストが高く、細粒度の並列処理を行うには通信のレイテンシがボトルネックとなり、高性能化、高速化を阻害する原因となっていた。

また、計算機資源の有効活用と公平な分配とを行うためには、適切なスケジューリングと負荷分散が必要になるが、現在の分散システムではこういった機能は十分に実現されていなかった。

2.1.2　コンピュータ・コロニーの目標

2.1.1で述べた問題点を解決し、分散システムの長所の維持と高性能化とを両立するために、我々の研究室は、コンピュータ・コロニーの提案を行っている。コンピュータ・コロニーとは、「複数の独立したコンピュータからなり、全体として一つのコンピュータのように動作するシステム」を意味する。

コンピュータ・コロニーは、現在の分散システムを拡張したシステムである。その目標の一つは、ネットワークによって分散した計算資源を有機的に結合し、現在の分散システムでは実現できない高性能を実現することである。また、コンピュータ・コロニーでは、高性能を実現しつつ、従来の分散システムのような利用形態を実現し、快適性を維持することも目指す。これらの目標をまとめると、以下のように示すことができる。

- スケーラビリティ
  計算機資源の追加によってリニアな性能向上が得られる。
- Single System Image[3]
  分散システム全体を、一つの仮想的なコンピュータとして、ユーザに提供する。これにより、ユーザは、コンピュータ・コロニー内に存在する全ての計算資源を、その位置や能力を意識すること無く利用することが可能となる。
- マルチ・ユーザ環境
  バッチ処理やシングル・ユーザ環境ではなく、複数のユーザが同時にシステムを利用できるマルチ・ユーザ環境を実現する。
- 公平な資源分配（Fair Share）
  マルチ・ユーザ環境においては、計算資源の分配に公平性が必要となる。これにより、全てのユーザにとって快適な環境を実現する。
- コンピュータ資源の有効活用
  動的負荷分散により、コンピュータ・コロニー内に存在する全ての計算機資源を有効に活用できるようにする。

2.2 コンピュータ・コロニーのシステムモデル

2.2.1 コンピュータ・コロニーのシステム

前述の目標を実現するため、コンピュータ・コロニーは、ハードウェア及びソフトウェア（オペレーティングシステム）の両面から開発が進められてきた。なお、コンピュータ・コロニーのために開発された分散 OS を Colonia と呼ぶ。

それぞれに必要とされる機能は、下記のようになる。

ハードウェア　MPP(Massively Parallel Processor) と比肩する並列処理性能を実現するためには、ノード間の通信性能を向上させる必要がある。そこで、ハードウェアにおいて、ネットワークで接続されたノード間での効果的な並列処理が行えるような通信機構を実現する。

分散 OS Colonia　前述で挙げたコンピュータ・コロニーの目標の多くは、主に OS に依存する部分が大きい。これらは、システムを利用する全てのユーザが快適、公平にシステムを利用できるよう資源管理を行うことにより実現される。マルチユーザ向けのスケジューリングを行う [4] と共に、プロセス移送を用いた負荷分散 [5] を行い、その目標を実現する。

これらの機能を実現するハードウェアと、分散 OS Colonia を開発することにより、コンピュータ・コロニーの実現を目指す。

2.2.2 システム構成

コンピュータ・コロニーでは、プロセッサ、メモリ、ネットワークインタフェースカードを基本単位とするモジュールカードによるハードウェア構成を採用する。コンピュータ・コロニーを構成するノードは、このようなカードを 1 枚から複数枚装着した本体と、モニタ、キーボードなどの周辺機器を備えたものとなる。このノードを、ネットワークで結合することにより、全体で分散システムを形成する。図 1に、システム構成の例を示す。このようなモジュールカード型の構成をとることにより、将来の拡張が容易であると考えられる。

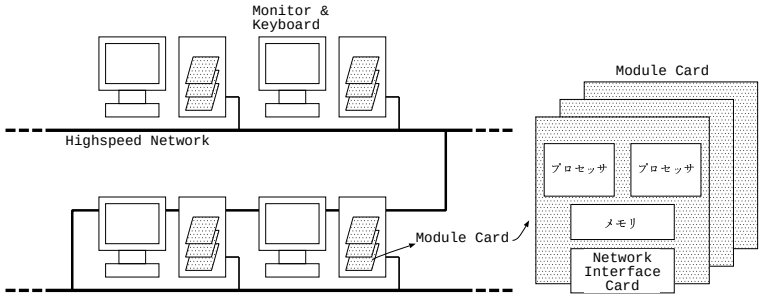


図 1: コンピュータ・コロニーの構成例

2.3 コンピュータ・コロニーの通信機構

先に述べたコンピュータ・コロニーの目標を実現するためには、専用の通信機構の開発が必要となる。本節においては、その通信機構について述べる。

2.3.1 コンピュータ・コロニーで求められる通信機構

コンピュータ・コロニーの目標を実現するためには、高速かつ低コストな通信機構を実現することが必要となる。これにより、LAN 規模の分散システムにおいて比較的細粒度の通信を実用可能なレベルで実現することが可能となり、分散している計算資源の有効利用と、高性能化をはかることができる。

このような通信機構を実現するための要件としては、

- システムコールを介さない通信
  通信資源を保護するために起こるシステムコールと、それに伴う OS の介入を経ること無く通信を実現するために、通信処理の開始と通信資源のプ

ロテクションをネットワークインタフェースカードで行う。これにより、通信レイテンシを削減し、細粒度並列処理を可能とする。

- プロトコル処理によるオーバヘッドの削減
  汎用のプロトコルを使用すると膨大なプロトコル処理によるオーバヘッドが発生するため、通信プロトコルを簡略化しこのオーバヘッドを削減する。
- リモートメモリのキャッシング
  リモートノードにあるメモリの内容を、自ノードのメモリにキャッシングし、それ以降のメモリアクセスを高速化する。メインメモリのキャッシュタグをネットワークインタフェースカード上に備え、システムバスに接続されることによりコヒーレンス処理が可能となる。
- デッドロック対策
  デッドロック防止機構を、ネットワークインタフェースカード上に備える。

といったものが考えられる。

2.3.2 コンピュータ・コロニーにおける分散共有メモリ

2.3.1で述べた要件を満たすために、コンピュータ・コロニーでは、2.2.2で提案したモジュールのシステムバスに、ネットワークインタフェースカードを接続し、共有メモリベースの通信を行う。

そこで、コンピュータ・コロニーは仮想共有メモリ [6] を使用している。これは、ネットワーク上に一つの仮想空間を構成し、物理的に分散したメインメモリをその仮想空間にマッピングすることによって共有メモリを実現するものである。

ノード間でのデータの共有単位としては、ページ単位とキャッシュブロック単位が考えられる。Colony で採用する通信機構 [7] では、リモートメモリアクセスが発生した場合、アクセスした側のメモリ領域の確保はページ単位で行うが、実際のデータ転送はブロック単位で行う。この方式により、ディレクトリの容量を抑えつつ、false-sharing による無駄な通信の発生を無くすことができる。

ユーザはシステムの提供する仮想空間にアクセスを行う。そのアクセスは、MMU(Memory Management Unit:メモリ管理ユニット) によって物理アドレスへと変換され、システムバスに流れる。

このとき、ネットワークインタフェースカードが、システムバスを監視し、そのアクセスに対して、リモートアクセスを行う必要があるかどうかを判断する。リモートアクセスの必要が無いと判断されれば、ネットワークインタフェース

カードは、何も行わず、一般のローカルアクセスと同様に、ローカルノード上のメモリへのアクセスが行われる。また、リモートアクセスが必要であると判断されれば、ネットワークインタフェースカードがそのアクセス先のリモートノードを判断し、リモートノードへの通信を行う。

これにより、システムコールを介さないリモートアクセスを行うことが可能となる。また、ネットワークインタフェースカードが、システムバスに直接接続されていることにより、キャッシュコヒーレンス制御をハードウェアによって高速に行うことが可能となる。

2.3.3 分散 OS Colonia の共有メモリ管理

コンピュータ・コロニー上で動作する分散 OS Colonia は、前項で述べた共有メモリシステムを用い、共有メモリの管理を行う。そこで、分散 OS Colonia における共有メモリ管理について述べる。

分散 OS Colonia は、プログラミング・モデルとして、ミッション・ユニットモデルを提供する。これは、一般的な共有メモリ上の並列アクティビティモデルであるタスク-スレッドモデルを分散環境向けに拡張したモデルである。システムは、ノードの負荷状況に応じて、ユニットの投入、移送を行い [8]、負荷を動的に分散させる。移送に伴うユニットの位置の変更は、Global Directory Service(GDS)[9] が管理する。

タスク-スレッド・モデルでは，同一タスクに属するスレッドは全アドレス空間を共有するが，ミッション-ユニット・モデルでは，アドレス空間を部分的に共有する.

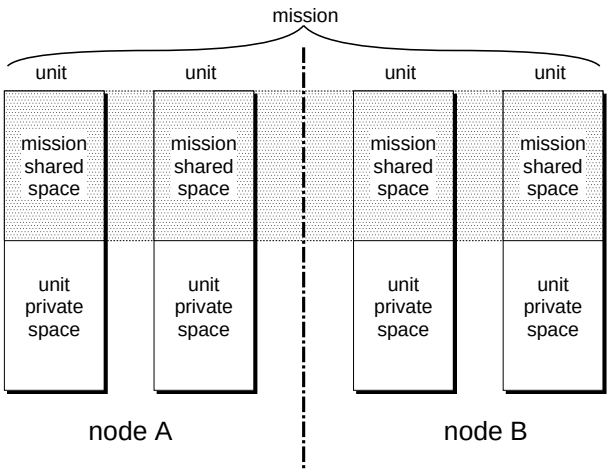


図 2: アドレス空間の構成

図 2のように，ユニットのアドレス空間を二つの領域に分ける．一つは，ユニット固有領域 (unit private space) と呼び，スタックなどのユニットに固有なデータを配置する．もう一方の領域は，ミッション共有領域 (mission shared space) と呼び，同一ミッションに属する全てのユニットで共有する領域である．ミッション共有領域には，ユニットが共通に使用するデータ，およびコードなどを配置する．

アドレス空間を二つの領域に分けたのは，管理コストを軽減するためである．

2.3.4 Colony の共有メモリ空間

2.3.3で述べたように、Colonia では、メモリ空間を共有領域と固有領域に区別して管理している。このうち、通信機構に関連してくるのは、共有領域である。この共有領域は図 3のように区別して管理される。

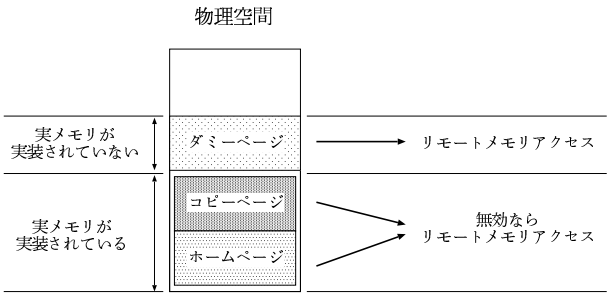


図 3: 物理メモリ空間

図 3中における各ページは、次のような意味を持つ。

- ホームページ
  自ノード上に物理メモリが実装されている物理アドレスに位置し、自ノードのメモリ上にデータの実体が存在する。アクセスの際には、リモートノードにキャッシングされている場合、有効か無効かが判断され、無効であればリモートメモリアクセスが行われる。
- コピーページ
  自ノード上に物理メモリが実装されている物理アドレスに位置し、リモートノードにあるホームページの内容をキャッシングしている。ホームページ同様、有効か無効かが判断され、無効であればリモートメモリアクセスが行われる。
- ダミーページ

自ノード上に物理メモリが実装されていない物理アドレスに位置し、リモートメモリ上にデータの実体が存在する。このため、リモートメモリアクセスが行われる。

初めてアクセスが行われた際、リモートノードに存在するページは、ダミーページにマッピングされる。その後、ダミーページへのメモリアクセスが起こると、リモートメモリアクセスが行われ、場合によってはその内容をコピーページとして自ノードにキャッシングする。

2.4 プロトタイプ・ハードウェア

ここまで述べてきた機能を実現するために、我々の研究室では、コンピュータ・コロニーのプロトタイプ・ハードウェア [10] の開発を進めてきた。プロトタイプ・ハードウェアは、高速通信機構の実験・評価を行うためのプロトタイプである。

プロトタイプ・ハードウェアでは、ノードとして米 Sun Mycro Systems 社の SPARCstation20 を用いている。SPARCstation20 は、システムバスとして MBus を持っており、そのスロットに専用のネットワークインタフェースカードを接続することにより、システムバスを監視し、通信コストを削減した共有メモリベースの通信を行うことができる。

プロトタイプ・ハードウェアにおいて、SPARCstation20 のプロセッサである SuperSPARC と、メインメモリ、そしてそのシステムバスにネットワークインタフェースカードを接続することで、コンピュータ・コロニーで想定しているモジュール構成をとる。このモジュールを一つのノードとして見なし、それらのノードを、高速な光通信インタフェースである FibreChannel[11] を介して結合し、互いに通信を行い、実験、評価を行う。

プロトタイプ・ハードウェアの概要を図 4 に示す。

2.5 関連研究

ここまで述べたように、我々は分散共有メモリベースの通信機構を Colony において採用している。ここでは、今までに提案されている、分散環境における種々の通信機構について述べ、それにより、Colony における通信機構の利点を明らかにする。

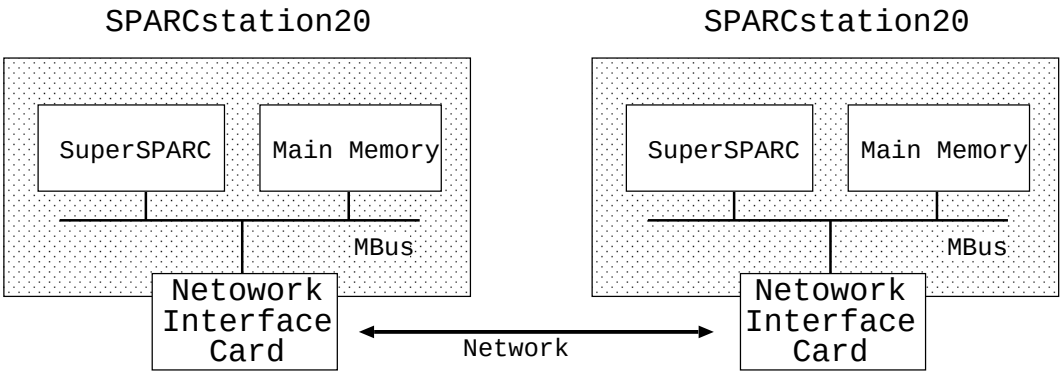


図 4: プロトタイプ・ハードウェア

2.5.1 メッセージベースの通信機構

並列処理を目指すメッセージベースの通信機構では、通信のオーバヘッドを減少させる通信ライブラリ、ファームウェアの開発が行われている。そのような通信ライブラリの例としては、myrinet[12] ベースのものが多い。

メッセージベースの通信機構では、通信はプログラム上で明示的に指定される。ホストプロセッサによって合成されたパケット識別子とデータは、システムコールや Programmed I/O(PIO)、Direct Memory Access(DAM) によって、通信ハードウェア上の送信バッファに転送されて通信が行われる。

メッセージベースの通信では、通信レイテンシの他に、通信前のパケット識別子構成の処理や DMA の起動コストが大きい。さらに、プロトコル処理のためのソフトウェアオーバヘッドも存在してしまう。また、プログラム中で通信を明示する必要があるため、並列化が困難な場合がある。

よってメッセージベースの通信機構では、比較的粒度の荒い並列処理は可能であるものの、我々の目指す細粒度の並列処理には向いていないということが言える。

2.5.2 分散共有メモリベースの通信機構

分散共有メモリベースの通信機構に関しても、種々の研究がなされている。ここでは、比較的 Colony と似たアプローチを取っている DIMMnet-1[13] について述べる。

DIMMnet-1 コンピュータ・コロニーのネットワークインタフェースカードの特長の一つとして、システムバスに接続することにより、システムコールの介在しない通信及びコヒーレンス制御を可能とし、通信レイテンシを削減していることが挙げられる。

同様の考えに基づいてPCクラスタ用に開発されたネットワークインタフェースに、DIMMスロット搭載型ネットワークインタフェースDIMMnet-1がある。

DIMMnet-1の他にも、高性能なPCクラスタ用ネットワークインタフェースとしてPCI-SCI[14]、MEMORY CHANNEL2[15]などの開発が行われているが、これらはいずれもPCIバスに接続して用いられるものである。

しかし、PCIバスはバンド幅に制限があり、さらに、PCIバスから主記憶へのDMAアクセスとCPUからのアクセスが競合し、主記憶のバンド幅が足りなくなるといった問題も生じる。また、システムコール等によるオーバヘッドは避けられない。

PCIバスに接続するネットワークインタフェースのこういった欠点を解決するため、PCIバスでなく、メモリスロットに挿入するネットワークインタフェースが研究されている。このようなネットワークインタフェースとして、MEMOnet[16]が提唱されており、その一つがDIMMnet-1である。

DIMMnet-1は、PC100またはPC133仕様のDIMMスロットに挿入される。低遅延のFETバス・スイッチにより2バンクのSO-DIMMを切り替えつつ、RHiNET-2互換の光リンク・インタフェースとデータの送受信を行う。メモリバス側のインタフェースは、「プロセッサ搭載メモリモジュール(PEMM)動作仕様標準」[17]に準拠する。

DIMMnet-1では、CPUとメモリスロット間のバンドを用いるため、十分なバンド幅を確保することができ、また、ユーザプロセスが、システムコールの介在なしにDIMMnet-1用ネットワークインタフェースカード上のメモリにアクセスすることが可能になる。これにより、通信レイテンシを削減することができる。

しかし、DIMMnet-1には、キャッシュ制御に問題点がある。Intel P6系MPUでは、命令からキャッシュ全体のフラッシュはできるが、ライン単位や属性毎の無効化ができない。さらに、PCIからのDMA転送ではデータに対応するキャッシュラインを無効化することができるが、メモリスロット側からはそれができない。これらの理由から、DIMMnet-1では、受信データをアンキャッシャブル(Uncachable)な領域に配置することになる。

Colonyにおけるネットワークインタフェースカードと比較した場合、どちらもメインメモリの存在するシステムバス上に接続し、通信レイテンシの削減を図るという共通点を持っている。

しかし、Colonyにおけるネットワークインタフェースカードが、MBusのサポートしているキャッシュ制御機能を用いてキャッシュ制御をハードウェアで行うことができるのに対して、DIMMnet-1では、キャッシュ制御に前述の問題点が存在する。これは、PCをノードとするために、避けられない問題であるとも言えるが、この点において、Colonyにおけるネットワークインタフェースカードが、DIMMnet-1よりも優れていると言うことができる。

第 3 章　ネットワークインタフェースカードの開発

3.1　ネットワークインタフェースカード

3.1.1　ネットワークインタフェースカードの概要

第 2 章で述べたコンピュータ・コロニーにおける通信機構を実現するために、我々は、専用のネットワークインタフェースカードを考案した。このネットワークインタフェースカードの目指す機能は、次のようなものである。

- キャッシングによる通信最適化のサポート
  通信の絶対量を減らし、高速なメモリアクセスを実現するために、メインメモリをリモートメモリの大容量キャッシュに利用できる機構を実現する。
- ユーザレベル通信の実現
  従来の分散システムで必要だった通信の際のシステムコールをなくすユーザレベル通信を実現する。このために、システムバスに接続したネットワークインタフェースカードを利用し、従来システムコールで行なわれていた、通信開始処理と通信資源のプロテクションを行なう。
- 軽量通信プロトコルの適用
  通信プロトコルを簡略化し、ネットワークインタフェースカード上で信頼性保証処理を行う。このことにより、プロトコル処理全体に関わるオーバヘッドを削減する。

我々は、このような機能を備えたネットワークインタフェースカードの開発研究、評価のために、プロトタイプ・ハードウェアにおけるネットワークインタフェースカードの開発を進めてきた。

3.1.2　ネットワークインタフェースボードの構成

我々の開発したネットワークインタフェースボードは、下図（図 5）のような構成をとっている。このネットワークインタフェースボードは、プロトタイプ・ハードウェアのシステムバスである MBus に接続される。

- FPGA
  再構成可能な Field Programmable Gate Array(FPGA) を搭載している。これにより、様々なプロトコル・通信機構を実装・評価することが可能である。
- プロトコルプロセッサ
  SPARC アーキテクチャ[18] に準拠した SPARClite をプロトコルプロセッサとして搭載している。プロトコルプロセッサは、リモートメモリアクセ

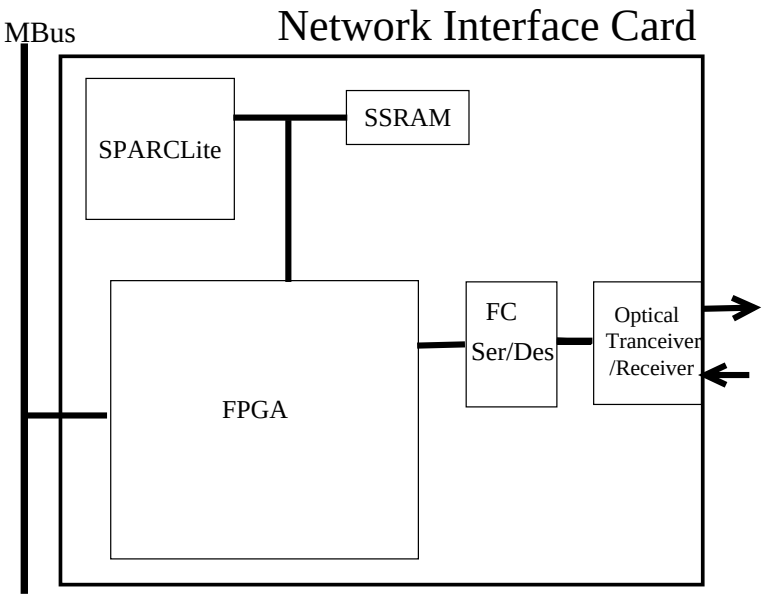


図 5: ネットワークインタフェースカード

ス時のアドレス変換に必要なテーブル類の管理などに用いる。

- シンクロナス SRAM（SSRAM）
  プロトコルプロセッサ用のキャッシュ、あるいは通信用のバッファ等に用いられるシンクロナス SRAM。
- Fibre Channel 用 Chipset
  高速ネットワークインタフェースとして、Fibre Channel を採用した。パラレル/シリアル変換を行なう Fibre Channel Ser/Des チップと光電変換を行なう Optical Transceiver/Receiver を備えている。
- その他　クロックジェネレータなど

我々は、このような構成をとるネットワークインタフェースカードを考案し、開発した。このネットワークインタフェースカードは、その上に搭載されている FPGA に回路を実装することにより、様々な機能を実現することが可能となる。

そこで我々は、まず、通信機能を司る通信ボードコアを設計し、実装することにした。

3.2　通信ボードコア

通信機能の実装及び、開発したネットワークインタフェースカードの検証・評価を行うため、通信ボードコアと呼ぶ、主に通信機能を司る機能を FPGA 上

に実装した。ここでは、実装した通信機構と、通信ボードコアの機能について述べる。

3.2.1 通信ボードコアの概要

通信ボードコアの開発は、大きく二段階に別けて考えることができる。それは、

- 通信の実現（ネットワークを介したデータ送受信の実装）
- 分散共有メモリ環境の実現

の二つである。

通信の実現に関しては、複雑なプロトコルを用いることはせず、独自の簡易なプロトコルを開発することにした。これは、Colonyでは、プロトコル処理のオーバヘッド削減のため独自のプロトコルを用いることを想定しており、今後のプロトコル開発に応じて容易に拡張出来るようにするためである。

このために、我々の開発した通信ボードコアでは、ネットワークインタフェースボードを構成している以下の部品を、次のように用いる。

- Fibre Channel用Chipset
  Fibre Channel用Chipsetは、本来、Fibre Channel用プロトコルを用いた通信を行うものである。しかし、本研究では独自のプロトコルを用いている。このため、Fibre Channel用チップセットは、光ギガビットネットワークのインタフェースとして用いられることになる。これは、本ネットワークカード上のFibre Channel用Chipsetが、OSIで言うところの極めて下位の階層のみ(物理層の一部)をサポートしているものであることから可能となっている。
- SSRAM
  ネットワークインタフェースカード上のSSRAMを、分散共有メモリとして用いる。

このように素子を用い、通信ボードコアを実装することにより、通信機能と、SSRAMを分散共有メモリとして使用する機能を実現する。

このために必要なネットワークインタフェースカードの機能は、次のようなものになる。

- ローカルノード上のSSRAMへのRead/Write
- リモートノード上のSSRAMへのRead/Write
- MBus上のメモリアクセスの監視とアクセス先の判別

これらのRead/Writeは、あらかじめ設定されたノードIDとそのIDごとに割り当てられたアドレス空間により、ユーザはリモート/ローカルを気にすること無く使用することができる(図6)。つまり、ユーザにとっては、共有メモリへのアクセスは、ネットワークインタフェースカードに割り当てられたメモリ空間へのアクセスとして認識され、リモート/ローカルに関わらず同一の方法でアクセスすることが可能となる。これにより、SSRAMを共有メモリとして使用することができ、細粒度での分散並列処理を可能とする。

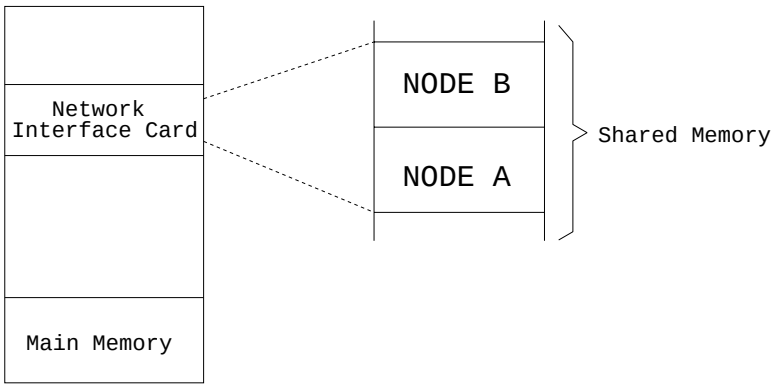


図6: メモリ空間

3.2.2 通信ボードコアの仕様

3.2.1で述べたように、ネットワークインタフェースカードに実装することにより同カード上のSSRAMを共有メモリとし細粒度での分散並列処理を可能とする通信ボードコアの開発を行った。

通信ボードコアの3つの主要な機能は、下記のようになる。

- MBus上のメモリアクセスの監視とアクセス先の判別
  ネットワークインタフェースカードは、SPARCstation20のシステムバスであるMBusに接続される。このため、MBusとのインタフェースが必要である。また、ユーザからローカルアクセス/リモートアクセスの区別を隠すために、行われている処理のアクセス先がリモートメモリであるかローカルメモリであるかをアドレスに応じて判別する必要がある。
- SSRAMへのRead/Write
  ローカルのネットワークインタフェース上にあるSSRAMに対して、Read/Writeを行える必要がある。

- Fibre Channel Chipset を用いたデータの送受信
  前述のように、Fibre Channel Chipset を、ギガビットネットワークのインタフェースとして用い、その上に独自のプロトコルを載せる。ここで用いられるプロトコルは、プロトコル処理のオーバヘッドを削減するため、なるべく軽量なものが望ましい。

これらの機能を実現するためには、予め、アドレス空間、デッドロック対策、エラー制御の 3 点を決めておく必要がある。

以下、これらについて詳細を述べる。

- アドレス空間
  アドレス空間としては、図 6で表されるアドレス空間を採用している。各ノード上のメインメモリは、そのノード固有のメモリ領域となる。一方、ネットワークインタフェースカードに割り当てられたメモリ領域は、各ノード間での共有メモリとなる。

  これは、ネットワークインタフェースカードに割り当てられたメモリ空間をノード ID ごとに分割し、その ID に応じたノード上の SSRAM を、そのメモリ空間に割り当てることにより実現している。つまり、共有メモリ領域へのアクセスが行われた場合、そのアクセス先のアドレスが割り当てられたノード ID を持つノード上の SSRAM に対して、アクセスが行われることになる。
- エラー制御
  汎用ネットワークのビットエラー率は、10^{-12}から 10^{-15}程度であることが期待される [11][19]。エラー処理をソフトウェアで行う場合、非常に大きなソフトウェアオーバヘッドが生じるため、ハードウェアで対応することが望ましい。これは、本研究では、細粒度の並列処理を目指しているため、通信の都度ソフトウェアオーバヘッドが生じてしまうと、それによるレイテンシが大きな問題となるからである。

  そこで、パリティチェックによるエラーの検出を行い、それに伴う Acknowledgement(Ack.) もしくは No-acknowledgement(Nack.) によってハードウェアによる再送処理を行うこととした。

  また、パケットの喪失に備えて、タイムアウトを設定し、タイムアウトが発生した場合も、ハードウェアで再送処理を行うこととする。この他、各パケットの先頭に一定の値をヘッダとしてつけることにより、パケットを

受信した際、そのヘッダをチェックすることにより誤ったデータを受信することによる受信処理の開始を回避している。

- デッドロック対策

  前述のエラー制御の一貫として、Ack. と Nack. の導入を述べた。本通信ボードコアでは、Ack. もしくは Nack. もしくは Reply が返ってくるまで待ち状態に入り、次の要求を出さないことにしている。

  しかし、このような待ち状態に入った時でも、他ノードからの要求に答えられるようにしておかなければ、デッドロックが生じる可能性がある。そこで、待ち状態に入っている時でも、他ノードからの要求に応答できるように、データパスを別に設けると共に、送信/受信モジュールの占有を最低限のものとする。同時に、一つのノードが一度に行うことの出来る要求処理を一つに限定することにより、デッドロック対策を行っている。

  また、エラー制御の一貫として導入したタイムアウトは、デッドロック対策の一つとしても働く。

通信プロトコル　これまでの仕様を元に、通信プロトコルの決定を行った。ここから、本通信ボードコアで採用している通信プロトコルについて述べる。

- トランザクション

  本プロトコルで行われるトランザクションは、次の 5 つである。

  - 要求系
    - Write
    - Read
  - 応答系
    - Acknowledgement(Ack.)
    - No-acknowledgement(Nack.)
    - Reply

  エラー制御、及びデッドロック対策として、全ての要求系のトランザクションに対しては、応答系のトランザクションが付随することになる。つまり、Write トランザクションには Ack. もしくは Nack. が Read トランザクションには、Reply もしくは Nack. が、通信先ノードから返信されてくる。

- パケット

  一回のトランザクションに当たり、ヘッダ、トランザクションの種類、アドレス、データ、パリティデータを送信する。

パリティには、(ヘッダ、アドレス、トランザクションの種類) の合計とデータの各ビットごとの偶数パリティを用いることとする。
ここで言うアドレスは、通信ボードコアによって抽出された、送信先リモートノードの SSRAM のメモリアドレスを指している。なお、回路の単純化のため、Read や Ack. 等、本来データを必要としないトランザクションの場合にもデータとして空データを送信することにより各トランザクションで送られるパケットが同様の形をするようにしている。

- エラー処理
  受信したパケットに対するパリティチェックの結果、エラーが検出された場合、受信したパケットが要求系のトランザクションのものであれば、Nack. の送信によってエラーが伝えられる。要求系のトランザクションを開始したノードは、Nack. を受け取った場合は再送処理を行う。
  また、エラーの発見されたパケットが応答系のトランザクションのものであれば、そのパケットを破棄することにより、後述のタイムアウトを引き起こし、再送処理を行うこととする。なお、パリティエラーが、トランザクションの処理を示すビットに対して生じている場合も、パケットが破棄され、タイムアウトを引き起こす。
- タイムアウトの設定
  前述のように、エラー対策の一貫として、要求系のトランザクションには、Ack. などの応答系のトランザクションが必ず伴われることとした。この際、通信エラー等の何らかの原因で、Ack. や Reply が得られなかった場合、タイムアウトが発生し、データの再送が行われることとする。これは、デッドロック対策も兼ねている。

3.3 通信ボードコアの実装

3.3.1 実装に当たって

ここまで述べた仕様に基づき、通信ボードコアの実装を行った。実装に当たっては、考慮すべき点がいくつかある。ここでは、これらの点について述べる。

通信パケットの構成 3.2.2で述べたように、本実装で用いる通信プロトコルでは一回のトランザクションに当たり、ヘッダ、トランザクションの種類、アドレス、データ、パリティデータを送信する。これには、下図 (図 7) のようなパケットを使用することとした。このように、一回のトランザクションに当たり、

ヘッダに12bit、トランザクションの種類(Read,Write,Ack./Reply)に2bit、アドレスに18bit、データに32bit、パリティデータに32bitの計96bitを用いる

図7: パケットフォーマット

共有メモリとして使用するSSRAMのアドレスが18bit幅であることから、通信パケットのアドレスは18bitとなる。また、SSRAMのデータ幅が32bitであることから、データに32bitを用いることとした。

トランザクションの種類には、2bitを用いている。"01"がWriteトランザクション,"10"がReadトランザクション,"11"が応答系トランザクションを示す。Nack.の区別は、引き続くデータの中身にそれぞれを区別する特定の値を入れることによって行う。また、Ack.とReplyの区別は、それらを受信したノード(要求系のトランザクションを開始したノード)が、Ack.とReplyとを同時に待つことはないこととした(3.2.2)ため、その時待っているトランザクションとしてそれらを処理することになる。

パリティには、(ヘッダ、アドレス、トランザクションの種類)の合計32bitとデータの32bitの各ビットごとの偶数パリティを用いることとする。

インタフェース　ネットワークインタフェースカードを構成している素子や使用している規格によって、それらと通信ボードコアとの間のインタフェースが定まる。

- MBus
  MBusでは、64bit単位でのアクセスが規定されているため、64bit幅のデータ送受信が行える必要がある。また、MBusのアドレス空間は36bit幅であることから、その36bitアドレスから、リモート／ローカルアクセスの判断、及び、SSRAMのアドレスの抽出を行うことになる。
- SSRAM
  我々の開発したネットワークインタフェースボードには、1wordが16bit幅、アドレスが18bit幅のSSRAM(容量500Kbyte)が二つ搭載されている。この二つのSSRAMに対して同じアドレスでアクセスすることにより、

1word が 32bit 幅のメモリ（容量 1Mbyte）として用いる。

- FC Ser/Des
  FC Ser/Des チップは、8bit 単位でデータを送受信するため、96bit のパケットを 8bit 単位に分割して送受信する必要がある。

動作クロックの違い　MBus 及び SSRAM が 50MHz のクロックで動作するのに対して、FC Ser/Des は 106MHz で動作している。このため、この違いを吸収する必要がある。

モジュール構成　今回開発した通信ボードコアは、今後の研究において適宜、改良及び機能の追加などの拡張がなされていくものと考えられる。このため、いくつかのモジュールに切り分けることにより、今後の研究開発を容易なものとしている。

3.3.2　通信ボードコアの構成

前節で述べた機能を実現するために、以下のような構成をとる通信ボードコアを FPGA に実装した。

- MBus インタフェース部
  - MBus Interface Module、MBus Data CTRL Module
- 送受信部
  - Fibre Channel Data CTRL Module、Send Data Module、Receive Data Module 、Fibre Channel Interface Module
- シンクロナス SRAM インタフェース部
  - SSRAM Data CTRL Module、SSRAM Interface Module

概略図を図 8に示す。

3.3.3　通信ボードコアの動作

実装した通信ボードコアは、次のように動作する。

- 要求系トランザクションを開始するノードの動作
  - MBus Interface モジュールが、アクセスを感知し、そのアクセス先を予め設定されたノード ID ごとのメモリ空間と照らし合わせることにより判断し、MBus DATA CTRL モジュールに送信する。
  - ローカルアクセスの場合
    * MBus DATA CTRL モジュールは、SSRAM DATA CTRL モジュールに CTRL 信号とデータを送信する。その後、Read であれば待ち状態となり、SSRAM DATA CTRL モジュールから結果が戻っ

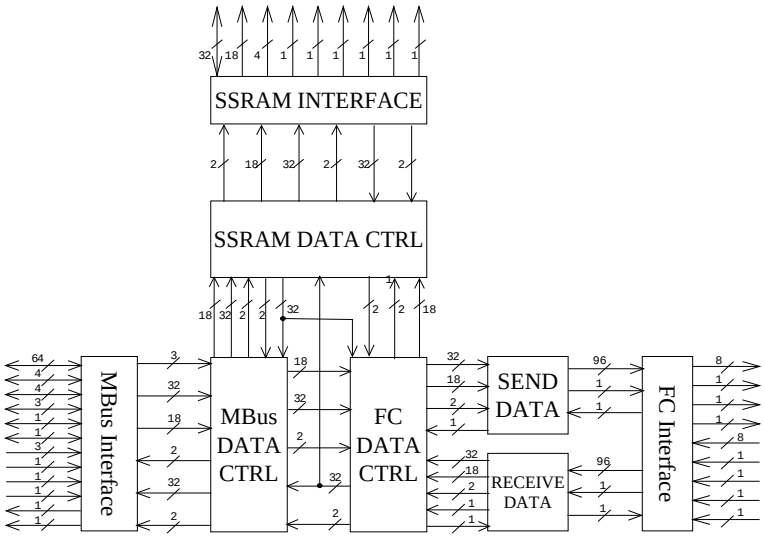


図 8: 通信ボード コアの構成

てくるのを待つ。Write であれば、処理の終了を MBus Interface モジュールに知らせる。

- SSRAM DATA CTRL モジュールは、SSRAM Interface モジュールを用いて SSRAM に対する Read/Write を行う。
- Read であった場合は、SSRAM DATA CTRL モジュールから、MBus DATA CTRL モジュールに結果が返される。
- MBus Interface モジュールは、MBus DATA CTRL モジュールから結果を受け取り、それを MBus に流す

– リモート アクセスの場合

- MBus DATA CTRL モジュールは、FC DATA CTRL モジュールに CTRL 信号と必要な DATA を送信する。その後、Ack. もしくは Reply を待つ待ち状態に入る。
- FC DATA CTRL モジュールは、SEND DATA モジュールに対して、CTRL 信号とデータ信号を送る。
- SEND DATA モジュールは、FC DATA CTRL モジュールから受け取ったデータを元にパケットを作成し、それを FC Interface モジュールに送る。
- FC Interface モジュールは、SEND DATA モジュールから受け取ったパケットを 8bit 単位に分割し、FC Ser/Des チップに送信する。

    * 要求系のトランザクションの結果、Reply、もしくは Ack.,Nack. のいずれかの応答系トランザクションが行われる。
    * FC Interface モジュールは、FC Ser/Des チップから受信したデータをパケットに整え、RECEIVE DATA モジュールに送信する。
    * RECEIVE DATA モジュールは、受信したパケットのパリティチェックとヘッダチェックを行い、エラーが検出されなければそれを FC DATA CTRL モジュールへと送信する。また、パリティエラーが発見されれば、エラー信号を FC DATA CTRL モジュールへ送信し、ヘッダエラーが発見されれば、そのパケットを破棄する。
    * FC DATA CTRL モジュールは、Ack. もしくは Reply の受信を MBus DATA CTRL に知らせ、データを渡す。
    * MBus DATA CTRL モジュールは、トランザクションの結果を MBus Interface モジュールに渡す。
    * MBus Interface モジュールは、結果を MBus に流し処理を終了する。
- 要求系トランザクションを受信するノードの動作
  - FC Interface モジュールは、FC Ser/Des チップから受信したデータをパケットに整え、RECEIVE DATA モジュールに送信する。
  - RECEIVE DATA モジュールは、受信したパケットのパリティチェックとヘッダチェックを行い、エラーが検出されなければそれを FC DATA CTRL モジュールへと送信する。また、パリティエラーが発見されれば、エラー信号を FC DATA CTRL モジュールへ送信し、ヘッダエラーが発見されれば、そのパケットを破棄する。
  - FC DATA CTRL モジュールは、SSRAM DATA CTRL モジュールに CTRL 信号とデータを送信する。また、トランザクションの種類が Write トランザクションであれば、この時点で Ack. を送信するように SEND DATA モジュールに命令する。また、RECEIVE DATA モジュールでパリティエラーが検出された場合は、Nack. を送信するように SEND DATA モジュールに命令する。Read トランザクションの場合は、結果が SSRAM DATA CTRL モジュールから戻ってくるまで待ち状態になる。

- SSRAM DATA CTRL モジュールは、SSRAM Interface モジュールを用いて SSRAM に対する Read/Write を行う。
- Read トランザクションであった場合、SSRAM DATA CTRL モジュールはその結果を FC DATA CTRL モジュールに送る。
- FC DATA CTRL モジュールは、SSRAM DATA CTRL モジュールから返された結果を元に、SEND DATA モジュールに対して Reply を送るように命令する。
- SEND DATA モジュールは、パケットを生成し、FC Interface モジュールに送信する。

デッドロックについて　MBus DATA CTRL モジュールが Ack. 等の待ち状態に入った際でも、FC DATA CTRL モジュールが動作しているため、リモートノードからの要求に答えることが可能となり、デッドロックを回避している。他にも待ち状態に入るモジュールが存在するが、主に自ノード上の SSRAM への READ の結果待ちであり、数クロックで完了することが見込まれるため、デッドロックは生じない。

タイムアウトについて　MBus DATA CTRL モジュールにおいて待ち状態に入り、既定時間内に Ack. や Reply を受け取れなかった場合、タイムアウトが発生し再送処理が行われる。また、FC Interface モジュールの受信部において、パケットが 96bit 分揃わなかった場合にも、タイムアウトが発生し、そのパケットが破棄される。

3.3.4　MBus インタフェース部の実装

MBus インタフェース部は、MBus とのインタフェースであると同時に、MBus のアクセスを監視し、共有メモリへの Read/Write であればそれに応じた処理を開始する機能を持つ。

MBus インタフェース部は、MBus Interface モジュールと MBus DATA CTRL モジュールの二つのモジュールからなる。

MBus Interface Module　MBus Interface は、その名の通り、MBus とのインタフェースを司るモジュールである。また、MBus のメモリアクセスを監視して、それが他ノードもしくは自ノード上のシンクロナス SRAM へのアクセスであった場合、実際に行う処理及びアクセス先を指定する機能ももち、MBus DATA CTRL モジュールに CTRL 信号 (Local Read,Local Write,Remote Read,Remote Write) とデータ (DATA,ADDRESS) を送る。

メインとなるステートマシーンは、6 つのステートを持つ（図 9）。ステートマシンの詳細については付録に記す。

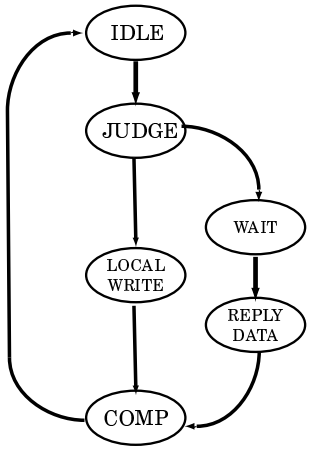


図 9: MBus Interface Module のメインステートマシン

MBus Interface モジュール内には、コマンド用のレジスタ、アドレス用のレジスタ、データ用のレジスタが備えられており、そのレジスタを介して実際の処理が行われる。また、ノード ID を保持するためのレジスタも備え、そのノード ID を元に、アクセス先の決定が行われる。

一連の処理は全て、この MBus Interface モジュールから始まる。

MBus DATA CTRL Module　MBus DATA CTRL モジュールは、MBus Interface モジュールからの CTRL 信号と DATA 信号に基づき、SSRAM DATA CTRL モジュールもしくは FC DATA CTRL モジュールへ CTRL 信号 (Read or Write) とデータ (Data,ADDRESS) を送るモジュールである。

データ用、アドレス用のレジスタを備えており、MBus Interface からデータをそれらのレジスタに取り込んだ後、MBus Interface からの CTRL 信号に従い、SSRAM DATA CTRL もしくは FC DATA CTRL のどちらかへ CTRL 信号を送り、ローカルメモリへの Write の場合を除いて、Reply もしくは Ack. を待つことになる。

メインとなるステートマシーンは、8 つのステートを持つ（図 10）ステートマシンの詳細については付録に記す。

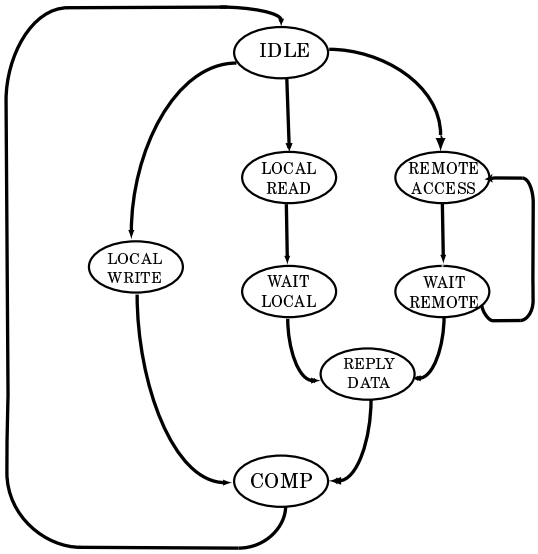


図 10: MBus DATA CTRL モジュールのメインステート

3.3.5 送受信部の実装

送受信部は、Fibre Channel チップセットを用いて、3.2.2で述べた通信プロトコルに従った送受信を行う機能をもつ。パケットの作成や、パリティの付加を行うと共に、リモートノードからのアクセスに対してパケットを受信し、そのトランザクションに応じたモジュールへとデータを受け渡す機能をもつ。

送受信部は、FC DATA CTRL モジュール、SEND DATA モジュール、RECEIVE DATA モジュール、FC Interface モジュールの 4 つのモジュールからなる。

FC DATA CTRL Module FC DATA CTRL モジュール は、MBus DATA CTRL モジュール もしくは RECEIVE DATA モジュール からの CTRL 信号を受けて処理を開始するモジュールである。

MBus DATA CTRL モジュール からの信号に従い動作を開始した場合は、リモートノード上の SSRAM への Read もしくは Write を行う場合であり、SSEND DATA モジュールへ、CTRL 信号と ADDRESS、DATA を渡しリモートノードへ送信を行うよう命令する。

一方、RECEIVE DATA CTRL モジュールからの信号に従い動作を開始した場合は、リモートノードからの SSRAM への Read/ Write を行う場合、もしくは、以前に自ノードが行ったリモートノードへの Read/Write に対する Reply、Nack. もしくは Ack. が返ってきた場合である。

リモートノードからのローカルノードへのReadに対しては、SSRAM DATA CTRL モジュールへCTRL信号とADDRESSを送り、結果が返されるのを待つ。結果が返ってきたら、それを Reply としてリモートノードへ送るようにSEND DATA モジュールへ渡すことになる。

リモートノードからのWriteに対しては、まずリモートノードへとAck.を返すようSEND DATA モジュールへ命令を送る。その後、SSRAM DATA CTRL モジュールへCTRL信号、ADDRESS、書き込みデータを送り、IDLE状態へと戻る。

リモートノードからのReply もしくはAck.であった場合は、送られてきたデータをM Bus DATA CTRL へと渡す。

なお、RECEIVE DATA モジュール においてパリティエラーが検出された場合は、リモートノードからのRead/Writeであれば Nack. を返すよう Send Data モジュールに命令し、SSRAM へのアクセスは行われない。リモートノードからの Reply もしくは Ack. であれば、パリティエラーが生じていた場合、そのデータを放棄することにより、タイムアウト処理を引き起こす。

FC DATA CTRL モジュールのメインとなるステートマシンは、下図 (図11)の11ステートからなる。ステートマシンの詳細については付録に記す。

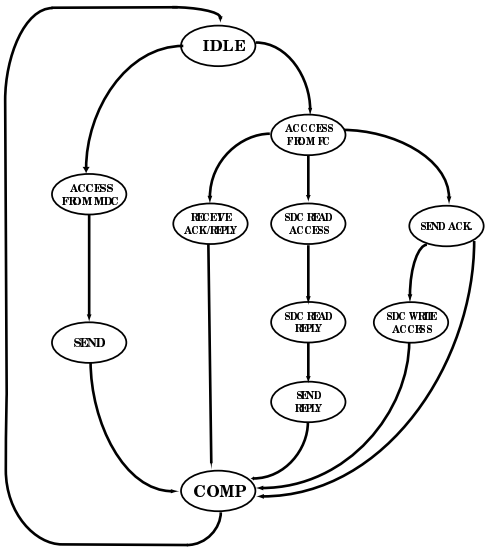


図11: FC DATA CTRL モジュールのメインステート

SEND DATA Module SEND DATA モジュールは、FC DATA CTRL モジュールから受信したデータを元に、実際に送るパケットを作成するモジュールである。

一回の送信当たり、送信するデータの種類(Read,Write,Ack)に2bit、アドレスに18bit、データに32bit、パリティに32bitの計96bitを用いる(図7)。

FC DATA CTRL モジュールから受け取ったデータからパリティデータを作成し、単純化のためReadの場合にはデータとして空データ32bitを挿入する。これにより、トランザクションの種類に関わらず同様に送信が行われるようにしている。

メインとなるステートマシンは5ステートを持つ(図12)。ステートマシンの詳細については付録に記す。

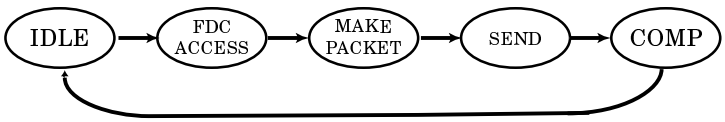


図12: SEND DATA モジュールのメインステート

RECEIVE DATA Module RECEIVE DATA モジュールは、FC Interface モジュールが受信したパケットを受け取り、FC DATA Control モジュールへと引き渡すモジュールである。

受け取ったデータは、32bitごとに順に、ヘッダとデータの種類とアドレス、データ(Readの場合は空)、パリティデータとしてレジスタに格納され、ヘッダチェック及びパリティチェックが行われる。この際、ヘッダチェックでヘッダが規定のヘッダと異なっていた場合、そのパケットは破棄される。パリティチェックで、エラーが見つからなかった場合、それらのデータをFC DATA CTRL モジュールへと引き渡す。また、パリティチェックの結果、エラーがあった場合は、エラー信号をFC DATA CTRL モジュールに送る。ただし、データの種類を表す2bitにパリティエラーが生じていた場合はそのパケットは破棄し、FC DATA CTRL モジュールにはエラー信号は送信されない。

メインとなるステートマシンは6ステートを持つ(図13)ステートマシンの詳細については付録に記す。

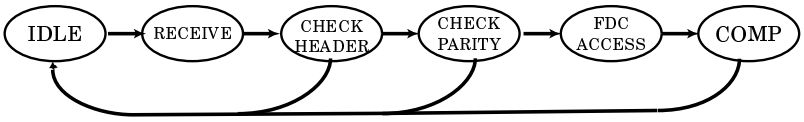


図 13: RECEIVE DATA モジュールのメインステート

FC Interface Module FC Interface モジュールの機能は、Fibre Channel Ser/Des チップへの送信と Fibre Channel Ser/Des チップからの受信である。

Fibre Channel Ser/Des チップは、8bit 単位での送受信を行うため、96bit であるパケットを、8bit 単位に分割して送信する。また、受信の際は、8bit 単位で受信したデータを 96bit のパケットに結合する必要がある。

このため、上位モジュールとなる SEND DATA モジュールから、CTRL 信号を受けると、次に送られてくるパケットを受け取り、それを順に 8bit ずつ Fibre Channel Ser/Des チップへと送信する。また、Fibre Channel Ser/Des チップから、8bit ずつデータを受信し、96bit が揃った時点で受信を示す CTRL 信号を RECEIVE DATA モジュールに送り、受信したパケットを渡す。

FC Interface は、受信部と送信部の二つからなる。上位モジュールである SEND DATA モジュール、RECEIVE DATA モジュールが 50MHz で動作、Fibre Channel Ser/Des chip が 106MHz で動作しているため、内部に 50MHz 動作部と 106MHz 動作部とを持つことになる。

- 送信部

  SEND DATA モジュール からデータ送信信号が入ると、パケットを受け取り、96bit レジスタに格納する。それを 32bit レジスタへ 3 回に分けて順に送信し、次に、32bit レジスタから 16bit レジスタへ 2 回に別けて順に送信、それをさらに 8bit レジスタに順に送信し、最終的に CTRL 信号と共に FC Ser/Des Chip へ送信する。

  動作周波数が内部で変わるため、50MHz のレジスタから、106MHz のレジスタへデータを移す部分のステートマシンが重要となる。その部分は 50MHz 動作部、106MHz 動作部それぞれ 4 ステートを持つステートマシーンを使用している。そのうちの一つは空ステートなのだが、ステートを現すレジスタの各ステート間の変化を 1bit ずつとするために導入されたものである。

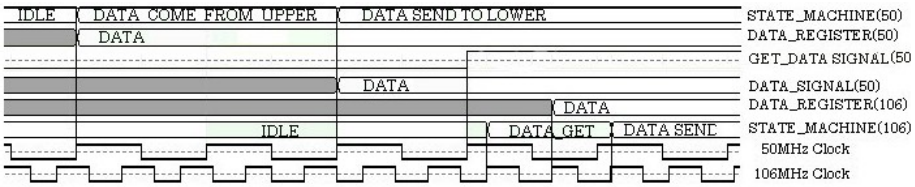


図 14: タイミング図

図 14は、50MHz 動作部と 106MHz 動作部のインタフェースのシミュレーション結果を示したものである。図中の DATA は、渡されるべきデータを示しており、IDLE、DATA_COME_FROM_UPPER などはステートマシンの状態名である。また、図中右に示している各信号の名前の ( ) 内の値は、その信号の動作クロックを示している。

50MHz のステートマシンは、上位層からデータが来るとそれを受け取って DATA_COMING_ FROM_ UPPER ステートに遷移する。次に、DATA_SEND _TO_LOWER ステートに遷移し、それと同時にデータ線 (DATA_SIGNAL) に信号を送信する。次に、GET_DATA_SIGNAL が送られ、それを受けて 106MHz 動作側のステートマシンが DATA_GET ステートに遷移する。DATA_GET ステートにおいて、DATA_SIGNAL から DATA を受け取り、内部レジスタ (DATA_REGISTER(106)) に格納する。このようにすることにより、DATA_SIGNAL 信号があげられてからそれを DATA_REGISTER(106) に取り込むまでに最低でも 50MHz クロックの 1 クロック分以上の間隔をおき、信号を確定している。

また、FC Interface モジュールには、高速動作に対応するためのパイプラインレジスタを多数挿入している。これは、Fibre Channel Ser/Des チップへの出力のタイミングが間に合うようにするためのものである。

- 受信部

  FC Ser/Des Chip からの信号を監視し、有効なデータが送られてきている場合はそれを受信する。受信は 8bit 単位で行われ、その後 CTRL 信号から生成された VALID データが付加される。

  8bit で受信されたデータを結合し、順に 17bit レジスタ、33bit レジスタ、97bit レジスタへと格納していく。17bit、33bit、97bit レジスタのそれぞれ 1bit は VALID データであり、それぞれのレジスタへ格納する際は、VALID を判断して、有効なデータが来たときのみ格納され、それぞれ 16bit、32bit、

96bit のデータを受信したところでそれに付随する VALID データがセットされる。

今回作成した FC Interface モジュールでは、96bit のパケットを用いているため、96bit 揃った時点で、RECEIVE DATA モジュールへパケットの到着を知らせる CTRL 信号を送り、その後、受信したデータを引き渡す。仮に何らかのエラーで 96bit 揃わない場合は、一定の期間を置いた後、タイムアウトが発生しそれまでに受信していたデータは破棄される。これは、96bit を連続して送信するように送信部を設計しているため、受信部においてデータは 96bit 連続して受信することが前提となっているからである。仮に 96bit 連続して受信しなかった場合は、何らかの通信エラーが生じたと判断され、そのデータが破棄されることになる。

一度パケット (96bit) を受信すると、RECEIVE DATA モジュールがそのパケットを受け取るまで、あるいは、そのパケットを破棄するための信号が来るまでは、その後の受信データは破棄される。

送信部同様、106MHz 動作部と 50MHz 動作部とが存在するため、その間のステートマシンには厳密なハンドシェイクが必要となる。これは、基本的には送信部で用いたものと同様のものを用いている。また、送信部と同様で、106MHz での動作を可能とするため、多数のパイプラインレジスタが挿入されている。

3.3.6 シンクロナス SRAM インタフェース部の実装

SSRAM DATA CTRL Module SSRAM DATA CTRL モジュールは、MBus DATA CTRL モジュール、もしくは、FC DATA CTRL モジュールからの CTRL 信号に従い処理を開始し、SSRAM Interface に、SSRAM の READ/WRITE を行なうよう信号を送信する。Read の場合は、SSRAM へのアクセスを終えた SSRAM Interface モジュールより値を受け取りリクエスト元 (MBus Data CTRL モジュール 又は FC Data CTRL モジュール) へとその値を返す。

SSRAM_DATA_CTRL モジュールのメインとなるステートマシンは、下図 (図 15) の全 8 ステートからなる。ステートの詳細については、付録に記す。

SSRAM Interface Module ネットワークインタフェースカード上に存在する SSRAM に Read/Write を行うためのモジュールが SSRAM Interface である。

カード上に搭載されている SSRAM は、アドレスが 18bit、1 ワードが 18bit(うち 16bit を使用) である。この SSRAM 二個を、同じ命令 (Read,Write) で同じ

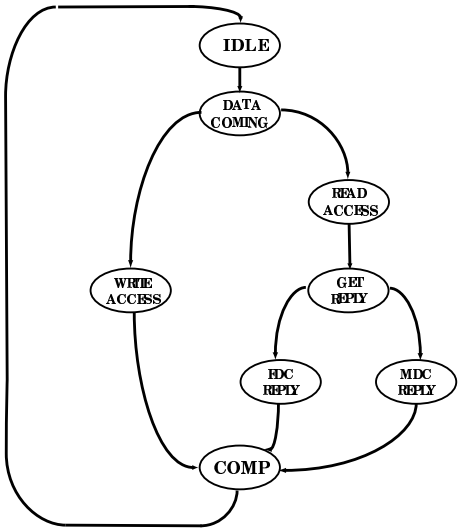


図15: Main State Machine of SSRAM DATA CTRL Module

アドレスにアクセスすることにより、アドレスが18bit、1ワードが32bitのメモリとして使用している。SSRAMには、1アドレスからの読み出し、書き込みが可能となっている。

第 4 章　ネットワークインタフェースカードの基本性能評価

前章において、我々の開発したネットワークインタフェースボードの目的及び機能と、通信ボードコアの設計と実装、さらにその通信ボードコアを搭載したネットワークインタフェースカードの機能について述べた。

本章においては、前章で述べたネットワークインタフェースボードを用いて通信を行い、いくつかの測定・評価を行った。本章ではこの測定・評価について述べる。

4.1　評価環境

評価環境には、前述のプロトタイプハードウェアを用いる。これは、SparcStation20 の MBus スロットに、我々の開発したネットワークインタフェースボードを挿入したものである。このネットワークインタフェースボードは、3.3V、5V の外部電源を必要とする。

このネットワークインタフェースボード上の FPGA に、前章で通信ボードコアを実装し、プロトタイプハードウェア一台、及び二台による評価を行った。

評価に当たっては、SparcStation20 上に、VineLinux をインストールし、ネットワークインタフェースボードにアクセスするためのシステムコールを追加した。これは、システムコールを介さずにアクセスする際に、短縮される処理時間を評価するためである。

4.2　ローカルメモリアクセス

第 3 章で述べたように、本研究で開発した通信ボードコアを実装したネットワークインタフェースカードは、共有メモリ環境を提供している。この共有メモリにアクセスする際は、ローカルノード上の SSRAM へのアクセスかリモートノード上の SSRAM へのアクセスが行われることになる。

そこでまず我々は、ローカルノード上の SSRAM へのアクセス時間を測定することにした。この測定に当たって、システムコールを用いたメモリアクセスと、システムコールを用いないメモリアクセスとの二通りの測定を行っている。これにより、システムコールを用いないことによる通信レイテンシの削減率を推定することができる。

4.2.1 システムコールを用いたメモリアクセス

まず、システムコールを用いてローカルのネットワークインタフェースボード上のSSRAMアクセスを行った際に要する時間を測定した。測定にあたっては、空ループ10^6回にかかる時間をあらかじめ測定しておき、システムコールによるローカルメモリWrite10^6回と、メモリRead10^6回を行うことにした。これらの値を比較することにより、システムコールによるオーバヘッドが生じる際のメモリアクセスに要する時間を計算することが出来る。

結果、

- 空ループ10^6回に0.28 s
- ローカルSSRAM Write 10^6回に3.64 s
- ローカルSSRAM Read 10^6回に4.56 s

を要した。

これより、システムコールを介した場合、ループにかかる時間を引いた、ローカルSSRAMへのメモリアクセス10^6回に要する時間は、Write、Readそれぞれ3.36s、4.28sであることが分かる。これより、1回に要する時間は、それぞれ3.36μs、4.28μsとなる。

4.2.2 システムコールを用いないメモリアクセス

第3章で述べたように、本研究で開発したネットワークインタフェースボードは、MBusに直接接続されており、メモリ空間を割り当てておくことにより、システムコールを介さずにSSRAMへのメモリアクセスを行うことが可能である。そこで、システムコールを介さずに、ローカルSSRAMへのRead、Writeを行い、その時間を測定した。

この結果、

- ローカルSSRAM Write 10^6回に0.72 s
- ローカルSSRAM Read 10^6回に1.24 s

を要した

前節の通り、空ループ10^6回に要する時間が0.28sであるから、その時間を引いた、メモリアクセス10^6回に要する時間は、Read、Writeそれぞれ0.44s、0.96sとなる。これより、1回のWrite、Readに要する時間は、それぞれ0.44μs、0.96μsとなる。

4.2.3 ボード上での処理にかかる時間

FPGA 内に、カウンタを追加し、MBus 上のメモリアクセスを感知してからローカルメモリアクセスの処理を終了するまでにかかる時間を、カウンタを用いて計測した。

この結果、

- ローカル SSRAM Write 1 回に 5 クロック (0.10μs)
- ローカル SSRAM Read 1 回に 27 クロック (0.54μs)

を要した。

この結果は、設計した回路の状態遷移から期待されるクロック数と一致している。

Read の方が圧倒的に時間を要しているのは、ローカル SSRAM Write の場合は処理の結果を待つ必要が無いためである。

以上の結果より、システムコールを用いずにメモリアクセスを行うことにより、システムコールを用いてメモリアクセスを行う場合と比べて、Read、Write の場合でそれぞれ、763%、446%の速度の向上を見ることができた。

4.3 リモート メモリアクセスに要する時間の測定

4.3.1 仮想リモート メモリアクセス

一台のプロトタイプ・ハードウェアにおいて、ネットワークインタフェースカードの送信部と受信部を直接 Fibre Channel ケーブルで繋ぎ、ループを作成することにより、一台のプロトタイプ・ハードウェアでリモートアクセスに要する時間を計測することが可能となる。このようにして計測した値は、理論的には、実際に二台で通信を行った値と一致するはずである。

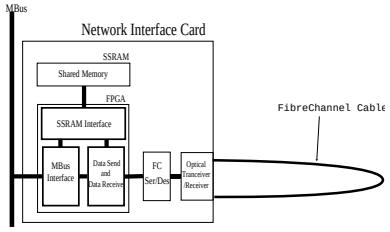


図16: 仮想リモート メモリアクセス

測定は、FPGA にカウンタを付加し、MBus からのメモリアクセスを感知して

から、リモートメモリアクセスを行い、Ack. もしくは Reply が返信され、MBus に結果が返されるまでの時間をカウンタによって計測することによって行った。

この計測の結果、

- リモート メモリへの Read 1 回に平均 133.5 クロック (2.67μs)
- リモート メモリへの Write 1 回に平均 114.3 クロック (2.28μs)

を要した。

ここでローカルメモリアクセスの場合と異なり Read と Write の処理時間の差が比較的少ないのは、リモートメモリアクセスの場合、現在は、処理の成功を示す Ack. が戻ってくるまで待ち状態となっているためである。Read と Write の処理時間の差は、FC DATA CTRL モジュールによる Ack. と Reply の発行するタイミングの差より生じている。

一回の処理に要する最短時間は、Read で 131 クロック、Write で 112 クロックであり、MBus DATA CTRL モジュールでタイムアウトが発生して再送が行われる再送率は約 0.01 であった。

ここで、再送率とは、1 回の要求トランザクションを終了するまでに、タイムアウトもしくはパリティチェックエラーによるデータ再送が行われる確率である。

$$\text{再送率} = \frac{\text{タイムアウトの発生した回数} + \text{パリティエラーが検出された回数}}{\text{トランザクション処理の回数}} \times 100$$

となる。再送率は、次に示す通信エラー発生率は直接計測することが出来ないため、通信処理時間の計測結果が妥当なものであるかを判別するために導入した。

通信エラー発生率の計算　処理にかかる平均の時間を T、エラーが発生しない場合に処理に要する時間を t、通信エラー発生率を p とする。このとき、T は次式で表すことができる。

$$T = t + 2tp + 3tp^2 + 4tp^3 + \bullet\bullet\bullet$$

これより、

$$T = \frac{\Sigma p^n}{1-p} t = \frac{1}{(1-p)^2} t$$

と求められる。T 及び t に、測定より得られた結果 (T=133.5、t=131) を代入し、これよりエラー発生率 p を求めると、約 0.01 となる。

これらの結果は、良好な結果が得られたといって良いであろう。

4.3.2 リモートメモリアクセス

次に、二台のプロトタイプ・ハードウェアを用いて、リモートノードへのメモリアクセス（Read/Write）に要する時間の測定を行った。

測定方法は、仮想リモートメモリアクセスの計測の際と同様に、MBus からのメモリアクセスを感知してから、リモートメモリアクセスを行い、Ack. もしくは Reply が返信され、MBus に結果が返されるまでの時間をカウンタによって計測するというものである。

計測の結果、

- リモートメモリへの Read 1 回に平均 229.7 クロック (4.59μs)
- リモートメモリへの Write 1 回に平均 194.1 クロック (3.89μs)

を要した。

これは、理論上一致するはずである仮想リモートメモリアクセスに要した Read131 クロック Write112 クロックと大きく異なっている。

一回の処理に要する最短時間は、Read で 131 クロック、Write で 112 クロックと仮想リモートメモリアクセスの結果と一致しているため、この処理時間の差は、通信エラーの発生に伴う再送処理によるものと考えられる。

この結果から計算される再送率は 74.0% となる。そこで、実際に再送率の測定を行った。

再送率の測定　MBus DATA CTRL モジュールにおいて、データの再送を行う回数を、その回数を数えるカウンタを付加することにより測定した。この結果、再送率は 72.4% となり、先ほど計算して得られた再送率とほぼ一致することが分かった。

エラー発生率の導出　前項と同様にエラー発生率の計算を行うと、エラー発生率は、24.5% となった。

以上の結果より、仮想リモートアクセスの際にはほとんど生じなかった通信エラーが、二台でリモートアクセスを行った際には生じていることが分かった。本来、仮想リモートアクセスと二台でのリモートアクセスは、理論上は同じものとなるはずである。これは、二台で通信を行う際に特有のエラーが発生しているものと考えられ、そのための対策が必要となる。このエラー対策については次章で述べる。

第 5 章　低信頼ネットワーク環境のためのプロトコルと拡張

第4章において、我々の開発したネットワークインタフェースカードに、通信ボードコアを実装した場合の測定結果を示した。本章では、この結果をもとに考察を行い、信頼性の低いネットワーク環境にも対応可能とするプロトコル拡張を行う。

5.1　通信エラーの対策

4.3.1で述べたように、Fibre Channel ケーブルでのループを使用した 1 台のプロトタイプ・ハードウェアによる仮想リモートアクセスの際にはほとんど通信エラーが見られなかったのだが、実際に二台のプロトタイプ・ハードウェアによるリモートアクセスを行うと、通信エラーが頻発し、その結果再送処理が複数回に渡って行われ、通信効率を著しく下げていた。

論理的には、仮想リモートアクセスも、二台でのリモートアクセスも同様のものである。仮想リモートアクセスの際には、通信エラーがほとんど起きていないことから、通信ボードコアに欠陥は無いものと考えられる。そこで考えられる通信エラーの原因としては、ボード特性、電源ノイズ、Fibre Channel Chipset 間での同期が取れていない、などが考えられる。これらの原因そのものを、現在のネットワークインタフェースカードで解決することは非常に困難であるため、通信ボードコアを変更することにより、通信エラー率の低減と通信速度の向上を試みた。

5.1.1　32bit 版 FC Interface モジュール

下図 (図 17) のような通信テスト用回路を設計し、通信テストを行った。これは、より単純な送受信テストを行うことにより、通信エラーの原因の解明を試みたものである。

MBus Interface モジュールを拡張し、MBus Interface モジュール内の送信レジスタに送信データを、コマンドレジスタに送信命令を書くことにより、FC Interface モジュールがある一定間隔で送信し、FC Interface モジュールが受信したデータを MBus Interface モジュール内の受信レジスタに書き込むものである。

この結果、原因の解明はならなかったものの、Fibre Channel Ser/Des チップ

に連続してデータを送信するのでなく、比較的細かい単位でデータを送信した方が、二台で通信を行った際のエラー率が低いことが分かった。

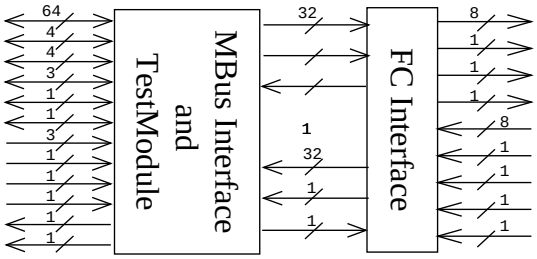


図 17: 通信テスト用回路

そこで、FC Interface モジュールを作り替え、これまでは 1 パケット分 96bit を連続して Fibre Channel Ser/Des チップに送信していた（図 18）ものを、パケットを 3 つに分割し、32bit ずつ、3 回に別けて送信する (図 19) バージョンを作成した。前者のバージョンの FC Interface モジュールを 96bit 連続送信版 FC Interface モジュール、後者のそれを 32bit 送信版 FC Interface モジュールと呼ぶことにする。

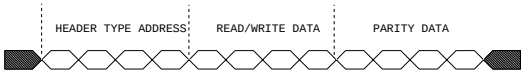


図 18: 96bit 連続送信

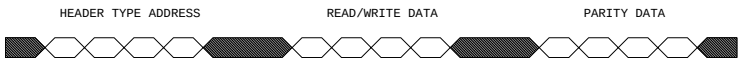


図 19: 32bit 送信

仮想リモートアクセスの測定　この 32bit 送信版 FC Interface モジュールを用いて第 4 章と同様に、まず、一台のプロトタイプ・ハードウェアによる仮想リモートアクセスに要する時間をカウンタを用いて計測した。

その結果、

- リモートメモリへの Read 1 回に 150 クロック (3.00μs)
- リモートメモリへの Write 1 回に 131 クロック (2.62μs)

を要した。

当然のことながら、96bit 連続送信版 FC Interface を用いた場合に比べて、処理に要する時間が増加している。増加率は Read、Write それぞれ 14.5% 17%であった。

リモート メモリアクセスの測定　次に、二台のプロトタイプ・ハードウェア間でのリモート メモリアクセスに要する時間をカウンタを用いて計測した。

その結果、

- リモート メモリへの Read 1 回に平均 226 クロック (4.52μs)
- リモート メモリへの Write 1 回に平均 194 クロック (3.88μs)

を要した。

これは、第 4 章で得られた 96bit 連続送信版 FC Interface モジュールを用いた場合の結果とほぼ同じ結果となっている。この結果からエラー発生率を求めたところ、18.6%となった。このことから、エラー発生率は、96bit 連続送信版と比べて低下しているものの、32bit 送信ずつの送信に替えたことによる一回の処理にかかる時間の増加が、エラー発生率の減少による通信速度の改善効果を打ち消してしまっていることが分かる。

5.1.2　二回送信版 FC Interface モジュール

次に、前項の 32bit 送信版 FC Interface モジュールを改良し、32bit に分割したパケットを、二回ずつ送るバージョンを作成した。

前述のように、ある処理を行う際に送られるパケットは、ヘッダーと ADDRESS とタイプ、WRITE/READ データ、パリティデータのそれぞれ 32bit、計 96bit である (図 7)。32bit 送信版 FC Interface モジュールで、ヘッダーと ADDRESS とタイプ、WRITE/READ データ、パリティデータの順で 1 度ずつ送っていたものを、ヘッダーと ADDRESS とタイプ、ヘッダーと ADDRESS とタイプ、WRITE/READ データ、WRITE/READ データ、パリティデータ、パリティデータの順で 2 回ずつ送るように FC Interface モジュールを変更した (図 20)。このバージョンの FC Interface モジュールを二回送信版 FC Interface モジュールと呼ぶことにする。

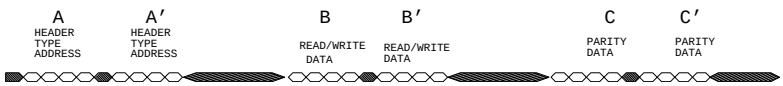


図 20: 二回送信

受信側においては、ある 32bit データを受信した場合、その後一定時間 T の間、次のデータは無視される。例えば、図中 A のデータを正常に受信した場合、その後、T が経過するまでに受信される A' のデータは無視されることになる。また、A のデータの送受信に失敗した場合も A' のデータの送受信に成功していれば、処理に影響は出ない。ただし、A' のデータを受信してから、B のデータを受信するまでの間隔を T 以上開けておく必要がある。B、C のデータに関しても同様である。

これにより、受信側の特別な変更を必要とせず、処理の向上が期待できる。

仮想リモートアクセスの測定　この二回送信版 FC Interface モジュールを用いて前章と同様に、まず、一台のプロトタイプ・ハードウェアによる仮想リモートアクセスに要する時間をカウンタを用いて計測した。

その結果、

- リモートメモリへの Read 1 回に 187 クロック (3.74μs)
- リモートメモリへの Write 1 回に 168 クロック (3.36μs)

を要した。

当然のことながら、先の 96bit 連続送信版、あるいは 32bit 送信版の FC Interface モジュールを用いた場合に比べて、処理に要する時間が増加している。増加率は、96bit 連続送信版と比べて、Read、Write それぞれ 42.7%、50%、32bit 一回送信版と比べてそれぞれ、24.7%、29.5% となっている。

リモートメモリアクセスの測定　次に、二台のプロトタイプ・ハードウェア間でのリモートメモリアクセスに要する時間をカウンタを用いて計測した。

その結果、

- リモートメモリへの Read 1 回に平均 206 クロック (4.12μs)
- リモートメモリへの Write 1 回に平均 185 クロック (3.70μs)

を要した。

この結果より、96bit 連続送信版の FC Interface モジュールを用いた場合と比べて、Read、Write それぞれ、10%、5% の処理速度の向上を見ることができる。

この結果より、エラー発生率を計算すると、4.7% と、かなり低くなっている。これより、通信エラーの低減による再送率の低下が、一回の処理に要する時間の増加を補って、全体として処理速度の向上を実現していることが分かる

5.2 システムコール削減に関する考察

第2章及び第3章で述べたように、我々の開発したネットワークインタフェースカードは、システムバスであるMBusに直接接続される。その理由の一つは、仮想管理記憶のためのMMUによるメモリ保護機構を利用することで、通信機構をユーザに対して安全に解放することが可能となるためであった。また、このネットネットワークインタフェースカードは、メモリアクセスに際して、リモートアクセスが必要であるかどうかを判断し、必要であれば自動的に通信を開始する。これらの機能により、通信時にOSが関与せず、従来通信の際に必要であったシステムコールを削減し、通信レイテンシを短縮することができる。ここでは、第4章で行った測定を元に、システムコールの削減の効果について考察を行う。

4.2で示したように、システムコールを用いてネットワークインタフェース上のメモリへのReadを行った場合、4.28μsを要した。対して、システムコールを介さずにローカルメモリへのReadを行った場合、0.96μsを要している。このため、システムコールによるオーバヘッドは、およそ3.32μsであることが分かる。

この際に使用されているシステムコールは、あるアドレスへのreadを行うものであり、通信処理を行う際に用いられるシステムコールと比べれば、極めて軽量なシステムコールであると言える。つまり、システムコールの介在する通信を行う場合、少なくとも3.32μsのシステムコールオーバヘッドが生じると考えることが出来る。

一方、5.1.2で示したように、通信ボードコアにおいてMBus Interfaceモジュールがメモリアクセスを感知してから、リモートノードへのReadトランザクションを終了するまでには4.12μsを要している。

これより、仮にシステムコールを介した通信を行った場合、最低でも7.44μsを要することになり、そのうちシステムコールの占める割合は実に45%に及ぶ。このことから、処理時間のうちシステムコールの占める割合が非常に大きく、システムコールを削減することにより、通信の高速化が測れることを示している。

システムコールの削減が、処理の高速化に極めて有効であることから我々がネットワークインタフェースカードを開発するに当たって採用した「システムコールを介さないユーザレベル通信」を用いる通信機構の有効性を示せたもの

と考えられる。

5.3 通信処理時間に関する考察

5.1.2で述べたように、二回送信版の FC Interface モジュールを用いた場合、リモートノードに対する Read、Write はそれぞれ 206 クロック、185 クロックで処理されることが分かった。

我々は、ネットワークインタフェースカードの開発に当たり通信処理時間の見積もりを行っている。見積もりの結果、リモートノードに対する Read を処理するのに要する時間は、234 クロック [7] であった。

本研究で開発した通信ボードコアは、32bit の Read/Write しか行っていない。Colony では、32Byte 程度の Read/Write の実装が必要であると考えられる。このデータサイズによる処理時間の増加は、主に、FC Interface モジュールで発生すると思われる。これは、FC Interface モジュールでは、FC Ser/Des チップの制約上、8bit 単位までパケットを分割する必要があり、他のモジュールの様にデータ幅を広げることによっては、データサイズの増大に対応できないからである。ただし、8bit に分割する部分は、106MHz クロックで動作しているため、比較的処理時間の増加を抑えることができる。

現在、32bit を 8bit×4 に分割しているものが、32Byte データの場合は、8bit×32 に分割することになり、最低で 28 クロック (106MHz) の増加となる。さらに、二回送信版であればこの倍の時間がかかる。これは、50MHz ではほぼ 28 クロックにあたり、この時間が送信・受信の計 4 回に渡って増加するため、112 クロックの増加となる。これから、32Byte 単位での Read/Write に要する時間は、318 クロック/297 クロック程度となると予想される。

これは、見積もりで出している通信時間を大幅に越えてしまっている。これは、見積もりの段階では、エラーに対する考慮がなされておらず、二回送信を行う処理時間などが含められていないためである。

第 3 章で述べた 96bit 連続送信版の FC Interface モジュールを用いた場合、エラーを考慮しなければ、データを 32Byte とした場合、最低 56 クロックの増加となり、187 クロック程度での 32Byte の Read/Write が可能となると推定され、見積もりで得た 234 クロック以内での Read/Write が可能であることが分かる。

このように、現在は、エラー対策として二回送信を行っているため、データ

サイズが大きくなった場合の処理時間の増加率が大きくなっている。しかし処理速度のさらなる向上には、さらなる通信エラーの解明と対策が必要になってくるものと思われる。

しかし、エラー対策を行っている場合でも、本稿で示されたリモートノードへの Read/Write 処理に要する時間は、細粒度並列処理に必要なμs 単位の処理を実現しており、このネットワークインタフェースカードを用いることにより、細粒度の分散並列処理が行えることが分かる。

また、共有メモリにおける通信はメモリアクセスであるため、通信処理自体に参照の局所性が介在する。我々の開発したネットワークインタフェースカードは、比較的大容量のキャッシュメモリを実現することが可能であり、通信処理に関して積極的に局所性を利用することにより、通信処理のさらなる高速化に有効であると考えられる。

第 6 章　まとめと今後の展望

本稿では、分散並列システムにおいて分散共有メモリ環境を実現するためのネットワークインタフェースカードの開発について述べた。

我々の開発したネットワークインタフェースカードは、従来システムコールで行なわれていた、通信開始処理と通信資源のプロテクションをシステムコールを介さずに行うことを可能とする。これにより、ユーザレベル通信を可能とし、また、キャッシング機能を備えることにより、細粒度の分散並列処理を行える通信機構を実現することを目指している。

我々は、通信機能の実装及び、開発したネットワークインタフェースカードの検証・評価を行うため、通信ボードコアと呼ぶ、主に通信機能を司る機能をネットワークインタフェースカードに実装した。

通信ボードコアの主な機能は、MBus 上のメモリアクセスの監視とアクセス先の判別、SSRAM への Read/Write、FibreChannel Chipset を用いたデータの送受信の 3 つである。これらの機能を実現するために、ネットワークインタフェースカード上の FPGA に、MBus Interface、MBus DATA CTRL、SSRAM Interface、SSRAM DATA CTRL、FC Interface、FC DATA CTRL、SEND DATA、RECEIVE DATA の計 8 モジュールを設計・実装した。

この通信ボードコアを実装したネットワークインタフェースカードを用いることにより、ローカルノード上及びリモートノード上の SSRAM への Read/Write が可能となる。さらに、この通信ボードコアは、システムバス上のメモリアクセスを監視し、ノード ID ごとに割り当てられたメモリ空間と比較することによりアクセス先がリモートであるかローカルであるかを判断する。さらに、その結果リモートアクセスが必要であれば自動的に通信を行う。これらの機能により、ネットワークインタフェースカード上の SSRAM を分散共有メモリとして用いることが可能となる。

開発した通信ボードコアを実装したネットワークインタフェースカードを用いて測定と評価を行った。この結果、一台のノードを用いた仮想リモートアクセスにおいてはほとんど通信エラーが見られなかったものの、二台のノードにおけるリモートアクセスでは、通信エラーが頻発し、処理速度の低下を招いていることが分かった。仮想リモートアクセスとリモートアクセスは、論理上は同じものであるため、エラーの原因は通信ボードコア以外に存在するものと思われ、

その解消を行うことは非常に困難である。そこで、送受信を行う FC Interface モジュールを改良することにより処理速度の向上を目指し、約10%の処理速度の向上が得られた。

この結果、リモートノードに対する Read、Write はそれぞれ 206 クロック、185 クロックで処理されることが分かった。このことから、細粒度並列処理に必要なμs 単位の処理を実現していることがいえ、このネットワークインタフェースカードを用いることにより、細粒度の分散並列処理が行えることが示せた。処理時間はキャッシュを実装することにより、さらに短縮することができるものと考えられる。

また、システムコールを介さないメモリアクセスとシステムコールを介したメモリアクセスについて計測した結果、システムコールを介さない場合、システムコールを介した場合に比べて 763%、446%の速度の向上を見ることができた。これにより、システムコールを介さないユーザレベル通信の有効性が示され、我々の開発しているネットワークインタフェースカード及びそれを用いた通信機構が、妥当なものであることを示せた。

本稿で述べた通信ボードコアの実装においては、ネットワークインタフェースカードに搭載されているプロトコルプロセッサを使用していない。今後の研究では、プロトコルプロセッサを使用し、通信ボードコアを拡張することにより、ネットワークインタフェースカード上の SSRAM を、キャッシュとして用いる高速通信を可能としたいと考えている。

謝辞

本研究の機会を与えて頂いた、本研究室の富田眞治教授に深甚な謝意を表します。

また、本研究に関して適宜御指導、御鞭撻を賜った森眞一郎助教授、中島康彦助教授、北村俊明助教授、五島正裕助手に深く感謝致します。

さらに、共同研究者である生雲公啓氏をはじめとして、日頃様々な角度から助力してくださった京都大学大学院情報学研究科通信情報システム専攻富田研究室の諸兄及びその他の学友に心より感謝致します。

参考文献

[1] 青木秀貴, 他: 共有メモリベースのシームレスな並列計算機環境を実現するオペレーティングシステムの構想, 情報研報, Vol. 97.

[2] 山添博史, 他: 共有アプリケーションを志向した分散システムコンピュータコロニーの構想, 情報研報, Vol. 97.

[3] Kai Hwang, Hai Jin, E. C. C. W. A. X.: Designing SSI Clusters with Hierarchikal Checkpointing and single I/O Space, *IEEE Concurrency*, pp. 60–69 (1999).

[4] Moal, D. L., 他: 分散システムにおける Fair Share プライオリティスケジューラ, 情報研報, Vol. 99.

[5] Itzkovitz, A.Schuster, a. L.: Thread Migration and its Applications in Distributed Shared Memory Systems, *TechnionIIT*.

[6] Kai Li, P. H.: Memory Coherence in Shared Virtual Memory Systems, *the ACM Transactions on Computer Systems*, Vol. 7, pp. 265–272 (1989).

[7] 伊達新哉, 他: コンピュータコロニーを実現する高速通信機構, 信学技法 (1999).

[8] 増田峰義, 鳥崎唯之, 他: 分散 OS Colonia における並列アクティビティの高速移送, 信学技法, Vol. 00.

[9] 増田峰義, 他: 分散 OS Colonia における共有メモリを利用した大域的ネーム・サービス, 信学技法, Vol. 99.

[10] 伊達新哉, 他: 並列応用を志向した分散システムのプロトタイプハードウェア, 特別研究報告書, 京都大学工学部情報学科.

[11] of ANSI, X. T. G.: Fibre ChannelPhysical and Signaling Interface, *FC-PH* (1994).

[12] Corp., M.: http://www.myri.com, Vol. 5.

[13] 田邊昇, 他: DIMM スロット搭載型ネットワークインタフェース DIMMnet-1 の通信性能評価, 情報処理学会研究報告, Vol. 2001.

[14] Corp., D.: PCI-SCI Adapter Card D320/D321 Functional Overvier Part, *http://www.dolphinics.com/* (1999).

[15] Fillo and Gillett: Architecture and Implementation of MEMORY CHANNEL2, *Digital Technical Journal*, Vol. 9 (1997).

[16] 田邊昇, 他: 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 MEM-Onet, 情報処理学会計算機アーキテクチャ研究会.

[17] 日本電子機械工業会: 日本電子機械工業会規格：プロセッサ搭載メモリモジュール (PEMM) 動作仕様標準, *EIAJ ED-5514*.

[18] Catanzaro, B.: SPARC マルチプロセッサアーキテクチャ.

[19] N.J.Boden, 他: Myrinet - A Gigabit-per-second Local-Area Network, *IEEE MICRO*, Vol. 15, pp. 29–36 (1995).

[20] Caplener, H. D. and Janku, J. A.: Improved Modeling of Computer Hardware Systems, *Computer Design*, Vol. 12, pp. 59–64 (1973).

付録

第三章において挙げた各モジュールのメインとなっているステートマシンについて詳細を付録として記す。

MBus Interface Module　メインとなるステートマシーンは、6 つのステートを持つ（図 A.1）。

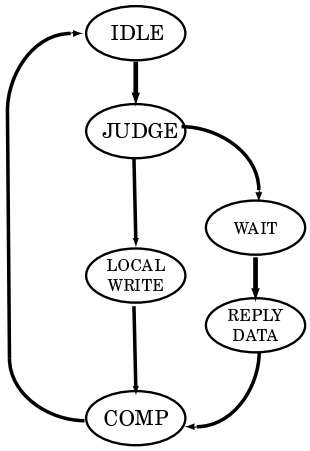


図 A.1: Main State Machine of MBus Interface Module

- IDLE
  IDLE ステート。通常このステートにあり、MBus からの信号を監視している。MBus からのアクセスがあった場合、JUDGE ステートに移る。
- JUDGE
  MBus からの信号を判定し、ローカル SSRAM への Read/Write、リモート SSRAM への Read/Write の 4 つのうちいずれの処理を実行するかを決定する。決定後、ローカルの SSRAM WRITE であった場合は、LOCAL_WRITE ステートへ、その他の場合は、WAIT ステートへ遷移する。
- LOCAL_WRITE
  ローカルノード上の SSRAM への WRITE を行うステートである。MBus DATA CTRL モジュールに、MBus からの信号より生成したアドレスと、データ、さらに、ローカルリードであることを示す CTRL 信号を送る。返答を必要としないため、その後速やかに COMP ステートへと遷移する。
- WAIT

ローカルノード上の SSRAM への READ、もしくはリモートノード上の SSRAM への READ/WRITE を行うステートである。これらの処理はいずれも何らかのリプライを必要とするため、必要な CTRL 信号と DATA 信号を MDC に送信した後、待ち状態となり、MBus DATA CTRL モジュールからのリプライを待つ。MBus DATA CTRL モジュールから Reply/Ack. データの到着を示す CTRL 信号が来れば、そのデータを受け取るために REPLY_DATA ステートに遷移する。

- REPLY_DATA
  ローカルノード上もしくはリモートノード上の SSRAM への READ であった場合は、その READ の結果を、リモートノード上 SSRAM への WRITE であった場合は、Acknowledgement(以後、Ack.) を受け取る。データを受け取ったら、COMP ステートへと遷移する。
- COMP
  終了処理を行い、各種信号をリセットし、MBus を解放する。その後、IDLE ステートに戻る。

MBus DATA CTRL Module　MBus Data CTRL モジュールのメインとなるステートマシーンは、8 つのステートを持つ（図 A.2）

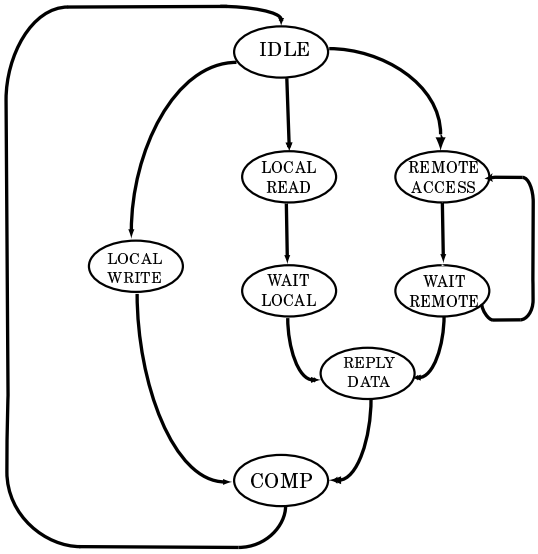


図 A.2: Main State Machine of MBus DATA CTRL Module

- IDLE

IDLE ステート。通常この状態にあり、MBus Interface モジュールからの Ctrl 信号により処理を開始する。Ctrl 信号の内容により、LOCAL_WRITE、LOCAL_READ、REMOTE_ACCESS の三つのステートのいずれかに遷移する。

- LOCAL_WRITE
  ローカルの SSRAM への WRITE を行なうステート。SSRAM DATA CTRL モジュールへ、CTRL 信号と DATA 信号、ADDRESS 信号を送信する。返信を待つ必要が無いため、SSRAM DATA CTRL モジュールが信号を受け取れば、COMP ステートへと遷移する。
- LOCAL_READ
  ローカルの SSRAM への READ を行なうステート。SSRAM DATA CTRL モジュールへ、CTRL 信号と ADDRESS 信号を送信する。その後、WAIT_LOCAL ステートへ遷移する。
- WAIT_LOCAL
  ローカルの SSRAM への READ を行なった際に、READ した結果のデータを待つステート。SSRAM DATA CTRL モジュールから、データが準備できたことを示す CTRL 信号が来るのを待ち、そのデータを受け取る。その後、REPLY_DATA ステートへ遷移する。
- REMOTE_ACCESS
  リモートの SSRAM に対する READ もしくは WRITE を行なうステート。READ か WRITE かを示す CTRL 信号、ADDRESS 信号、それに WRITE の場合は DATA 信号を FC DATA CTRL モジュールへ送信し、WAIT_REMOTE ステートへ遷移する。
- WAIT_REMOTE
  リモートの SSRAM に READ/WRITE を行なった際、その返信を待つステート。READ の場合はその結果が、WRITE の場合は、Ack. が返って来るのを待つ。これらの到着を示す、FC DATA CTRL モジュールからの CTRL 信号を待ち、その信号を受け取って、REPLY_DATA ステートへ遷移する。
  また、一定の期間内に、Ack. や Reply が返ってこなければ、タイムアウトが発生し、REMOTE_ACCESS ステートに戻る。また、Nack. を受信した場合も REMOTE_ACCESS ステートに戻り、データの再送が行われる。

- REPLY_DATA。
  WAIT_LOCAL ステートもしくはWAIT_REMOTE ステートで得られた結果を MBus Interface モジュールへ送信するステート。結果を返すことを示すCTRL 信号と、DATA 信号を MBus Interface モジュールへ送信し、MBus InterfaceL モジュールが受信を示す CTRL 信号を返して来たら COMP ステートへと遷移する。
- COMP
  終了処理を行なうステート。

FC DATA CTRL Module　FC DATA CTRL モジュールのメインとなるステートマシンは、下図（図 A.311 ステートからなる。

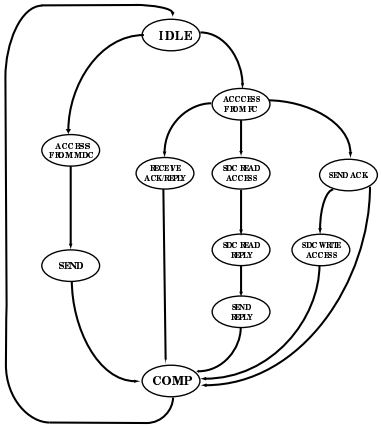


図 A.3: Main State Machine of FC DATA CTRL Module

- IDLE
  IDLE ステート。通常はこのステートにあり、MBus DATA CTR モジュール、もしくは、FC DATA CTRL モジュールからの CTRL 信号により処理を開始する。前者の場合は、ACCESS_FROM_MDC ステートへ、後者の場合は、ACCESS_FROM_FC ステートへ遷移する。
- ACCESS_FROM_MDC
  リモートノードへアクセスする際に処理を行うステート。MBus DATA CTRL モジュールから、処理の種類 (READ/WRITE) と ADDRESS 信号、DATA 信号 (WRITE 時）を受け取り、その後、SEND ステートへ遷移する。
- SEND

SEND_DATA モジュールへ送信命令を示す CTRL 信号と、データを送信する。SE ん D_DATA モジュールがそれらの信号を受け取ったのを確認した後、COMP ステートに遷移する。

- ACCESS_FROM_FC
  RECEIVE_DATA モジュールからデータを受け取り、それらのデータが、Ack. もしくはリプライなのか、SSRAM への READ なのか、あるいは SSRAM への WRITE なのかを判断する。そして、それぞれ、RECEIVE_ACK./REPLY ステート、SDC_READ_ACCESS ステート、SEND_ACK. ステートへ遷移する。
- RECEIVE_ACK./REPLY
  RECEIVE_DATA モジュールから受け取ったデータが、自ノードが始めた通信に対する Ack. もしくはリプライだった場合に処理をするステート。データを、Ack. もしくはリプライとして MBus DATA CTRL モジュールへ CTRL 信号と共に送信する。MBus DATA CTRL モジュールが受信したのを確認した後、COMP ステートへ遷移する。
- SDC_READ_ACCESS
  RECEIVE_DATA モジュールから受け取ったデータが、リモートノードからの READ 命令であった場合に処理をするステート。SSRAM_DATA_CTRL モジュールに、READ 命令を示す CTRL 信号と ADDRESS 信号を送信し、SSRAM_DATA_CTRL モジュールが受信したのを確認した後、待ち状態に入る。SSRAM_DATA_CTRL が、SSRAM READ の結果が来たことを示す CTRL 信号を送って来たら、待ち状態から SDC_RED_REPLY ステートに遷移する。
- SDC_READ_REPLY
  SSRAM READ の結果を SSRAM DATA CTRL モジュールから受け取るステート。データを受信後、受信完了を CTRL 信号によって SSRAM DATA CTRL モジュールへ知らせ、SEND_REPLY ステートに遷移する。
- SEND_REPLY
  SSRAM DATA CTRL モジュールから受け取った SSRAM READ の結果を送信するよう SEND DATA モジュールに命令するステート。リプライであることを示す CTRL 信号と、DATA 信号を SEND DATA モジュールに送信し、SEND DATA モジュールが受信したのを確認した後、COMP ス

テートに遷移する。

- SEND_ACK.
  RECEIVE_DATA モジュールから受け取ったデータが、リモートノードからの WRITE 命令であった場合に Ack. を送信するよう SEND DATA モジュールへ命令するステート。また、RECEIVE_DATA モジュールで、通信エラーが検出された場合には、このステートで、エラーを示す Ack. を送信するよう SEND DATA モジュールに信号を送る。。
  Ack. 送信後、SSRAM WRITE の処理であった場合には、SDC_WRITE_ACCESS ステートに、通信エラーによる Ack. 送信であった場合には、COMP ステートに遷移する。
- SDC_WRITE_ACCESS
  SSRAM DATA CTRL モジュールに、CTRL 信号、DATA 信号、ADDRESS 信号を送信するステート。SSRAM DATA CTRL モジュールが、受信したのを確認後、COMP ステートに遷移する。
- COMP
  終了処理を行なうステート。

SEND DATA Module SEND DATA モジュールは、FC DATA CTRL モジュールから受信したデータを元に、実際に送るパケットを作成するモジュールである。

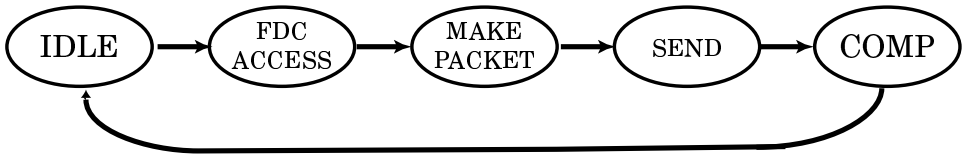


図 A.4: Main State of SEND DATA Module

- IDLE
  IDLE ステート。通常はこのステートにあり、FC DATA CTRL モジュールからの CTRL 信号が入ると、FDC_ACCESS ステートに遷移する。
- FDC_ACCESS
  FC DATA CTRL モジュールから、必要なデータを受け取るステート。受信完了後、MAKE_PACKET ステートに遷移する。
- MAKE_PACKET

パリティデータの生成とヘッダの付与を行うステート。タイプとADDRESS、READ/WRITEデータから、各ビットごとの偶数パリティを生成し、SENDステートへ遷移する。

- SEND
  生成したパケットをFC Interfaceモジュールに送信するステート。FC Interfaceモジュールに、CTRL信号を送り、FC Interfaceモジュールが送信可能状態であればデータを送信する。全て送り終えた後、COMPステートへと遷移する。
- COMP
  終了処理を行うステート。

RECEIVE DATA Module　RECEIVE DATAモジュールは、FC Interfaceモジュールが受信したパケットを受け取り、FC Data Controlモジュールへと引き渡すモジュールである。

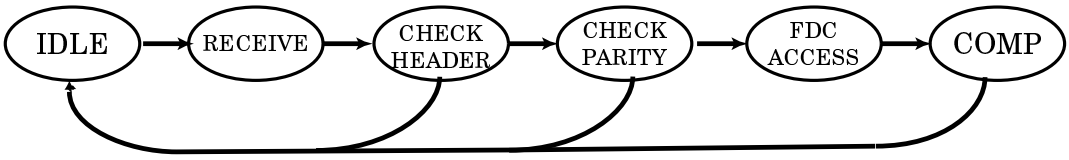


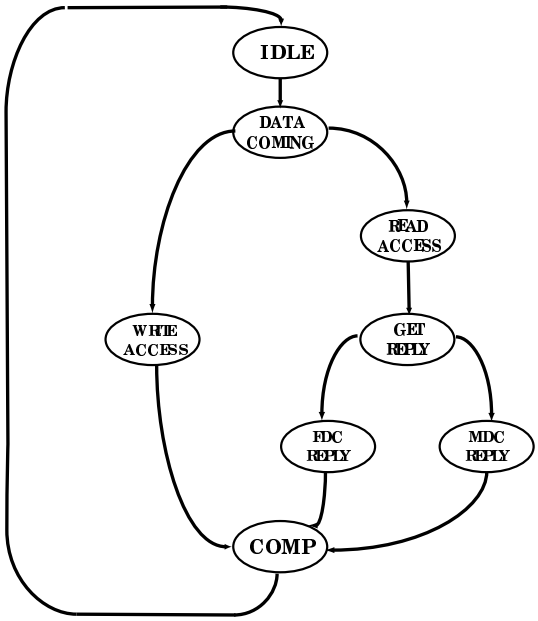
図 A.5: Main State of RECEIVE DATA Module

- IDLE
  IDLEステート。FC Interfaceから、データ受信を示すCTRL信号が来たら、RECEIVEステートへと遷移する。
- RECEIVE
  パケットをFC Interfaceから受信するステート。受信完了後、CHECK_HEADERステートに遷移する。
- CHECK_HEADER
  ヘッダーチェックを行うステート。規定のヘッダーと受信パケットのヘッダーが等しいかどうかをチェックし、等しければCHECK_PARITYステートへ、異なっていればパケットを破棄し、IDLEステートへ遷移する。
- CECHK_PARITY
  パリティチェックを行うステート。チェックの結果、もしエラーが検出されれば、エラー信号をFC DATA CTRLモジュールに送信する。いずれにし

ろ、チェック終了後、FDC_ACCESS ステートに遷移する。

- FDC_ACCESS
  データ受信を示すCTRL 信号をFC DATA CTRL モジュールに送信し、それに引き続き受信したデータを送るステート。FC DATA CTRL モジュールが受信したのを確認した後、COMP ステートへ遷移する。
- COMP
  終了処理を行うステート。その後、IDLE ステートに遷移する。

SSRAM DATA CTRL Module　SSRAM_DATA_CTRL モジュールは、下図（図A.6）の全8 ステートからなる。



図A.6: Main State Machine of SSRAM DATA CTRL Module

- IDLE
  IDLE ステート。通常はこのステートにあり、MBus Data CTRL モジュールもしくはFC Data CTRL モジュールから、CTRL 信号が入れば、DATA_COMING ステートに遷移する。
- DATA_COMING
  MBus DATA CTRL モジュール、もしくは、FC DATA CTRL モジュールからデータを受け取る。その後、行なう処理がREAD であればREAD_ACCESS に、WRITE であれば、WRITE_ACCESS に遷移する。

- WRITE_ACCESS
  SSRAM Interface モジュールへ CTRL 信号、DATA 信号、ADDRESS 信号を送信し、SSRAM へ WRITE を行なうよう命令する。その後、COMP ステートへ遷移する
- READ_ACCESS
  SSRAM Interface モジュールへ CTRL 信号、ADDRESS 信号を送信し、SSRAM へ READ を行なうよう命令する。返信を待つ必要があるため、返信が来るまで待ち状態になり、SSRAM Interface から返信が来たら GET_REPLY ステートに遷移する。
- GET_REPLY
  SSRAM Interface から、READ の結果を受け取るステート。受信後、一連の処理の開始が FC DATA CTRL モジュールからの信号であれば FDC_REPLY ステートに、MBus DATA CTRL モジュールからの信号であれば MDC_REPLY ステートにそれぞれ遷移する。
- FDC_REPLY
  FC DATA CTRL に、SSRAM READ の結果を返すステート。FC DATA CTRL がデータを受け取ったら、COMP ステートに遷移する。
- MDC_REPLY
  MBus DATA CTRL に、SSRAM READ の結果を返すステート。MBus DATA CTRL がデータを受け取ったら、COMP ステートに遷移する。
- COMP
  終了処理を行なうステート。完了信号を送信し、IDLE ステートへ遷移する。