筑波大学技術報告
25: 42-49, 2005

Maestro2 ネットワークインターフェースの開発

小野雅晃[1]
筑波大学システム情報工学等支援室（装置開発班）
〒305-8573 茨城県つくば市天王台 1-1-1

概要

クラスタ型コンピュータ Maestro2 用のネットワークインターフェース（NI）を開発したので報告する。NI はパーソナルコンピュータ(PC)の 64 ビット幅の PCI スロットに挿して使用するイーサーカードのような通信用のカードである。NI の PCI インターフェースの最大スループットは 64 ビットデータバス幅、66MHz 動作であるため 533Mbyte/sec で、PCI としては最大のスループットである。インターフェースは LVDS を使用し、物理的な最大スループットは片方向 3.2Gbps(Giga Bit Per Second)である。

1．はじめに

システム情報工学研究科、コンピュータサイエンス専攻の和田研究室では、クラスタ型のコンピュータに使用するために、高速、高機能のアドイン・カード及びスイッチから構成される通信システムを開発している。

5 年ほど前には、200Mbps の IEEE1394(i-LINK)を使用した第一世代の通信システム Maestro1 を開発した。

今回は、Maestro2 として第 2 世代の通信システムを開発した。使用した通信インターフェースは 3.2Gbps の LVDS である。通信システムは前述したように通信カードとスイッチで構成されるが、今回はそのうちのネットワーク・インターフェース(NI)について発表する。

2．NI の構成

図 1 に NI の写真、図 2 に NI のブロック図を示す。

NI は、300MHz 動作の PowerPC プロセッサを搭載したインテリジェントな PCI カードである。NI には CPLD(Complex Programmable Logic Device) FPGA(Field Programmable Gate Array)、PowerPC プロセッサ、FLASH ROM(Flash Read Only Memory)、SDRAM(Synchronous Dynamic Random Access Memory)、LVDS(Low Voltage Differential Signalling)インターフェース用 IC が搭載されている。CPLD は主に FPGA のコンフィギュレーションの制御を行う。コンフィギュレーションの方式はスレーブセレクトマップで、コンフィギュレーション･データを FLASH ROM から 8 ビットずつ FPGA に転送できるため、迅速に FPGA をコンフィギュレーションすることが出来る。また、コンフィギュレーション・データが間違っていて、FPGA が立ち上がらない場合には、ウオッチドック機能により、正常なコンフィギュレーション・データを FPGA にロードして、エラーの回復を図ることも出来るように設計されている。（現在この機能はまだ実装されていない）CPLD は Xilinx 社の XC95144XL を使用している。

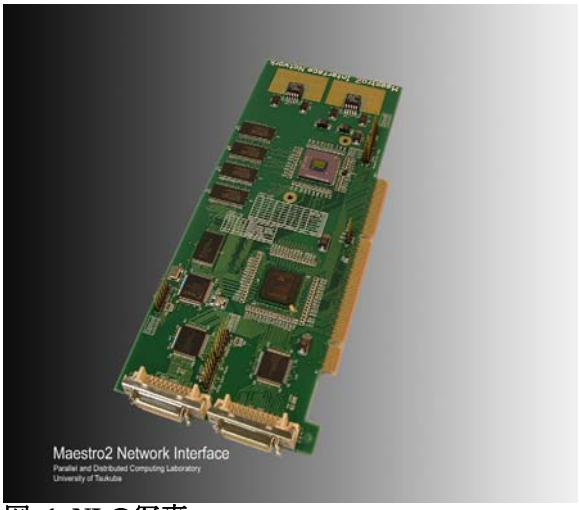

図 1 NI の写真

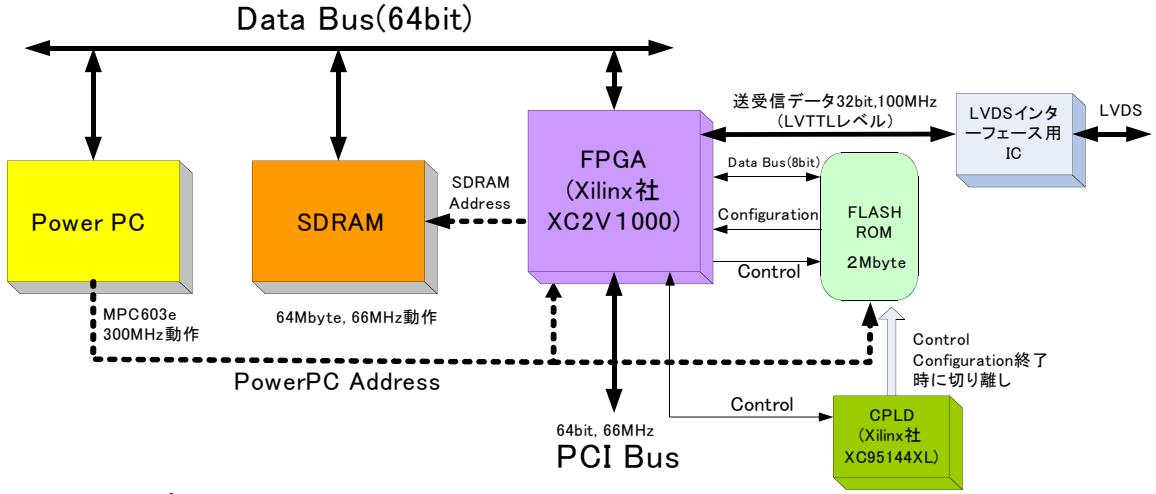

図 2 NI のブロック図

[1] E-mail: ono@sie.tsukuba.ac.jp; Tel: 029-853-5195

FLASH ROM は主に、FPGA のコンフィギュレーション・データを格納する目的で使用される汎用の FLASH ROM である。電源 ON 時には、CPLD の制御下に入り、FPGA にコンフィギュレーション･データを送る。それが終了すると、CPLD の制御から切り離され、FPGA の制御下に入る。

FPGA は、各デバイスの制御回路、DMA コントローラー、制御用レジスタが実装されている NI の心臓部である。NI 上の PowerPC プロセッサやホスト PC からのリソースの要求に応えて、データ転送の仲立ちをする。FPGA には Xilinx 社の Virtex2 シリーズのうちから 100 万ゲート相当の XC2V1000 を使用した。

SDRAM は PowerPC プロセッサと FPGA にバスが直結されている。データバス幅は 64 ビット、動作周波数 66MHz で動作している。

LVDS インターフェース用 IC は、ナショナル・セミコンダクタ社の DS90CR481 と DS90CR484 を使用している。これらの IC は、LVTTL48 ビットの入出力を LVDS データ 8 ビット＋クロック 1 ビットに変換する。FPGA からの LVTTL 信号は 32 ビットしかないので、48 ビットの内 32 ビットを使用して、後の 16 ビットは使用していない。これは主に FPGA のピン数が足りないためである。FPGA から LVDS 用 IC に出力する周波数は 100MHz、バンド幅は 32 ビットの 100MHz なので、3.2Gpbs となる。LVDS インターフェースで他の PC に挿入された NI や独自仕様のスイッチと接続される。LVDS インターフェースで接続されるのは、NI や独自仕様のスイッチに限られ、汎用のボードやスイッチなどとは接続できない。

NI は PCI バスでホスト PC と接続される。この PCI バスは 64 ビット幅、動作周波数は 64MHz である。NI へのスレーブ・アクセスは連続的にデータ転送(バースト転送)をすることが出来ないので、全て単一転送となる。マスタ・アクセスはバースト転送をすることが出来る。その際の最大転送レートは、533Mbyte/sec(約 4.26Gbps)となる。

尚、NI の製作で私が担当した箇所は、CPLD 設計と FPGA 設計の 2/3 ほどである。NI の回路設計は山際氏[2]と共同で担当した。NI のファームウェア、ホスト PC のソフトウェアは、山際氏と青木氏[3]が担当した。

３．CPLD の設計

CPLD は電源 ON 時に、FPGA のコンフィギュレーションの制御を行う。CPLD はコンフィギュレーション・データを EEPROM に格納しているため、電源 ON ですぐに動作できる。しかし、FPGA は内部の回路情報を RAM に格納しているため、電源を OFF すると内部の回路情報はクリアされてしまう。そのため、電源 ON 時には、毎回 ROM から内部回路の情報（コンフィギュレーション･データ）を読み込まなければならない。この ROM には、FPGA コンフィギュレーション専用の ROM を選択するのが普通であるが、本 NI では、あえて汎用 FLASH ROM を使用することとした。

それは次のようなメリットがあるからである。

1. PCI バス経由でホスト PC から、コンフィギュレーション･データを書き込むことが出来る。
2. ホスト PC から書き込んだコンフィギュレーション･データが不正でも、回復させることが可能である。

電源 ON の FPGA コンフィギュレーション時には、FPGA はモード選択ピンの設定によって、スレーブセレクトマップモードが選択される。スレーブセレクトマップモードは FPGA の外からクロックと 8 ビット幅のコンフィギュレーション用のデータを FPGA に入れる方式である。データが 8 ビット幅と広いので、高速にコンフィギュレーションすることが出来る。

CPLD は FLASH ROM にアドレスを供給して、FPGA のコンフィギュレーション用のデータを出力させる。FPGA とのハンドシェークを行って、FPGA にデータが入力できたら、FLASH ROM のアドレスを＋１して、FLASH ROM に次のデータを出力させる。このように、CPLD は FPGA のコンフィギュレーションを行う。

FPGA コンフィギュレーションが終了すると、CPLD は FLASH ROM の制御信号線を全てハイ･インピーダンス状態にして、FLASH ROM を切り離す。FLASH ROM の制御信号線は FPGA にも接続されているので、それから後は FPGA が FLASH ROM を使用することができる。FPGA は PCI コンフィギュレーション・アクセスに対して、FLASH ROM の領域をメモリ領域にアドレス･マップするので、ホスト PC から読み書きすることが出来るようになる。さらに、NI のローカル・プロセッサの PowerPC からも書き込むことが出来るようになる。FPGA コンフィギュレーション専用の ROM でコンフィギュレーションする場合には、NI ボード上の JTAG 端子にコンフィギュレーション用ツールを接続しなければならない。しかし、本 NI で採用した方式を使用すれば、ホスト PC から直接コンフィギュレーション･データを書き換えることが出来る。また、その他のデータも FLASH ROM に入れておくことが出来る。現在、イーサネットの MAC アドレスなどを入れて、利用している。

FPGA に書き込むコンフィギュレーション･データが間違っていると、NI が立ち上がらなくなる。こうなると、遠隔操作でリカバリすることが出来なくなるため、ホスト PC が遠い所にある場合はその場所まで行かなければならない。そのような場合のために、FLASH ROM に 2 つのコンフィギュレーション･データを格納しておく。新しく書き込んだデータが間違っていて、NI が立ち上がらない場合に、古く確実に立ち上がったデータを書き直すことで、NI を確実に立ち上げるようにすることも出来る。現在は、この機能は CPLD に実装していない。

４．FPGA の設計

FPGA 内部モジュールの構成を図 3 に示す。矢印の線は、モジュール間のデータや制御信号のつながりを示す。

FPGA は、MLX32、PCI、PCI Target Controller、PowerPC Interface、Internal Registers、Network Buffer Link Interface(NBL Interface)、SDRAM Controller、DMA Controller、System Arbiter、FLASH ROM Controller の各モジュールから構成される。

MLX32 は、Maestro2 独自の通信プロトコルを実装する。OSI 階層のトランスポート層でデータの完全性を保障し、バースト･データ転送を続けて行う機能を持っている。このモジュールは山際氏が設計した。

PCI モジュールは 66MHz、64 ビット幅の PCI バスのインターフェース回路である。PCI スレーブ･アクセス(ホ

[2] Instituto de Engenharia de Sistemas e Computadores (INESC-ID), Portugal

[3] 筑波大学　システム情報工学研究科

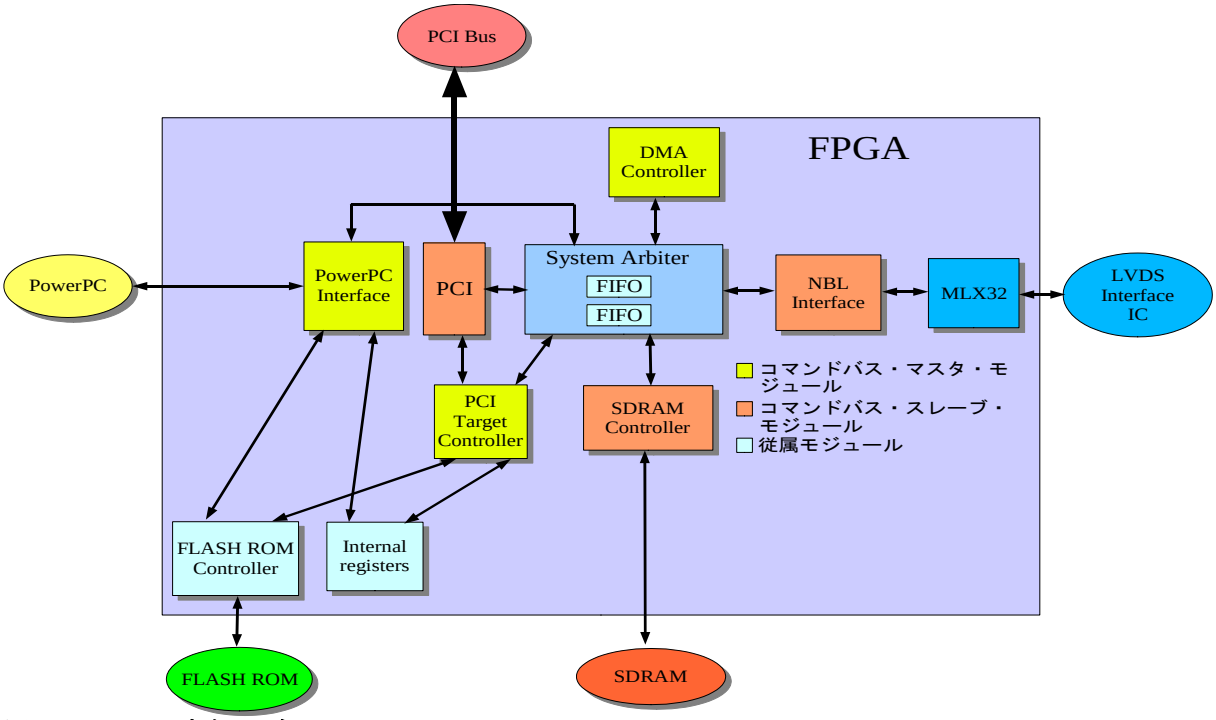

図 3: FPGA の内部モジュール

ストPCからNIへのデータ・アクセス)は、FPGA回路内でマスタとなる必要があるので、PCI Target Controllerに渡され、スレーブ・アクセスを行う。その他、PCIモジュールはNIからホストPCへのデータ・アクセスを実行する。PowerPC Interfaceは、PowerPC603eとのバス・インターフェース回路である。バス幅は64ビット、バスの動作周波数は66MHzとなっている。Internal Registersは、FPGAの内部の状態などを保持しておく回路である。NBL Interfaceは、MLX32とのインターフェース用の回路である。SDRAM Controllerは、66MHz動作のSDRAMを制御する回路である。DMA Controllerは、DMAコントローラーが2チャネル実装されていて、各モジュール間のDMA転送を制御する。詳細はセクション5で記述する。

FPGA回路内には、自分からその他のリソースを要求できるマスタとなれるモジュール(コマンドバス・マスタ・モジュール、以下マスタ・モジュールとする)が3つある。それは、PCI Target Controller、PowerPC Controller、DMA Controllerである。マスタ・モジュールの要求で、データ転送を行うのが、コマンドバス・スレーブ・モジュール(以下スレーブ・モジュール)である。DMA Controllerは、マスタ・モジュールではあるが、スレーブ・モジュールを2つ使用し、それらの間でデータ転送を行わせる特殊なマスタ・モジュールである。スレーブ・モジュールの一方をソース(データを読み出し、出力するモジュール)、もう一方をディスティネーション(データを受け取り、書き込むモジュール)として、データ転送を行う。DMA Controller自体はデータ・ストリームを送ったり、受けたりはしないで、制御だけを受け持つ。

マスタ・モジュールがどのようにスレーブ・モジュールとデータ転送をするかを説明する。図4にFPGA内部コマンドバスを示す。

1. マスタ・モジュールが、System Arbiterにリソースを要求する。(図4のreq)

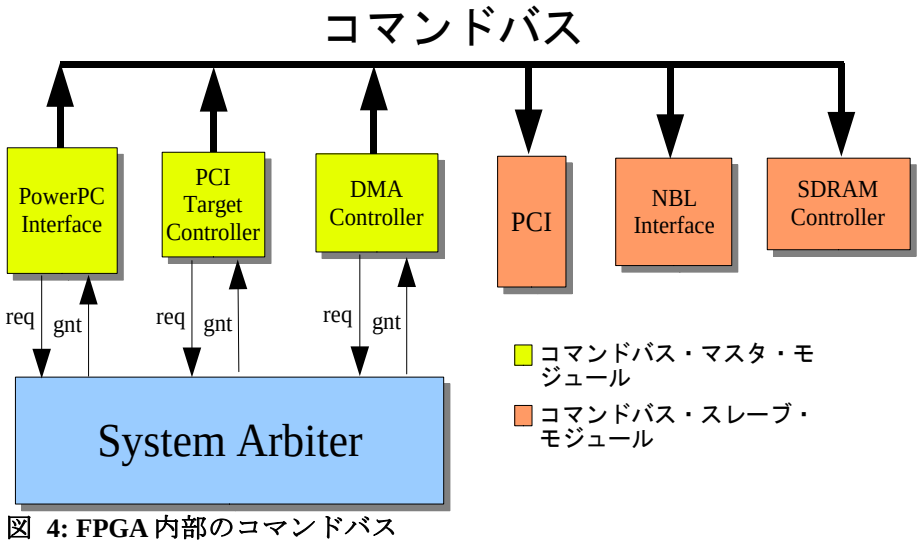

図 4: FPGA内部のコマンドバス

2. リソースが未使用で確保することが出来たら、System Arbiter はマスタ・モジュールに使用を許可する。(図 4 の gnt)
3. System Arbiter は、マスタ・モジュールとスレーブ・モジュール間のデータバスと制御バスを接続する。(セクション 4.2 で詳述する)
4. マスタ・モジュールは、そのリソースのコントローラーにコマンドを送りコネクションを確立し、使用を開始する。(図 4 のコマンドバス)

従属モジュールは、 Internal Registers と FLASH ROM Controller の 2 つで、PowerPC Controller と PCI Target Controller からだけアクセスされる。

4.1 コマンドバス

コマンドバス(図 4)はグローバル・コマンドバス、ローカル・コマンドバスとコマンド・イネーブルに分けることが出来る。コマンドバスでコマンドを送るのは、前述したとおり、マスタ・モジュールである。また、コマンドを受けるのはスレーブ・モジュールである。

グローバル・コマンドバスには、各モジュールに共通の項目が設定される。例えば、モジュールの ID、ソースまたはディスティネーション、バースト長などである。

ローカル・コマンドバスは、そのスレーブ・モジュールに特有の機能が設定される。例えば、SDRAM Controller ならば、アドレスと、バースト転送の最中にリフレッシュをするかどうかが設定される。

コマンド・イネーブルは、コマンドバスのコマンドが有効かどうかを表す。1 の時に有効となる。

Maestro1 では、中央制御回路で全ての制御を行っていたが、この方法は制御線が多くなると、制御手順を頭で考えるのが困難になる。それで、Maestro2 では、各モジュールがコマンドによって、制御される方式にした。こうすることで、各モジュールごとにシミュレーションで動作を確認するのにも都合が良い。

4.2 FPGA 内部モジュール間のデータバスと制御バス

FPGA 内部モジュール間のデータバスは 64 ビット幅のバスを 2 チャネル実装してある。現在は、チャネルごとにデータを流す方向が決まっている。これは、主にデータ・セレクタを節約して、ゲート規模を節約するためである。データバスは System Arbiter に実装してあり、データバスの内部には、FIFO やバイト・レーン変換をするための回路が実装してある。バイト・レーン変換機能は、8 バイトの境界の途中から DMA 転送する場合に、バイト・レーンの入替を行う回路である。(セクション 5.8 で詳しく述べる)図 5 に FPGA 内のデータの流れを示す。図 5 で DMA 転送はバースト転送可能な CH0、CH1 の FIFO を通るデータパスを使用する。

制御バスは、データバスに対応して 2 チャネル実装されている。制御バスは次に示す 3 種類の信号がある。

1. レディ
2. データ転送終了
3. バイト・イネーブル

レディは、”データ出力や入力する準備が出来た”ということを表す信号である。各チャネルごとにソース・レディとディスティネーション・レディがある。

データ転送終了はコマンドで指定されたバースト長のデータ転送が終了したことを表す。これもレディと同様に各チャネルごとにソースとディスティネーションの信号がある。

バイト・イネーブルは、どのバイト・レーンが有効化を示す信号である。データバスは 64 ビット、8 バイト幅なので、バイト・イネーブルは 8 ビット幅である。ソースから出力され、ディスティネーションでデバイスに書き込む時にどのバイト・レーンが有効であるかを示す。

5. FPGA 内部モジュール

FPGA 内部モジュールを順に詳しく説明する。

5.1 MLX32

MLX32 は、セクション 4.2 で記述したデータバスからデータをケーブルが接続されている先の NI や SB に送り出すためにカプセル化する。データは 32 バイト単位の 1 つ又は複数のパケットにまとめられ、ヘッダを付けられて、送り出される。ハードウェアによるエラー訂正機能も実装されていて、受信側でエラーが検出されると、送信側は同じパケットを再送信し、正常に受信できるまで再送する。MLX32 は OSI 参照モデルで言うとデー

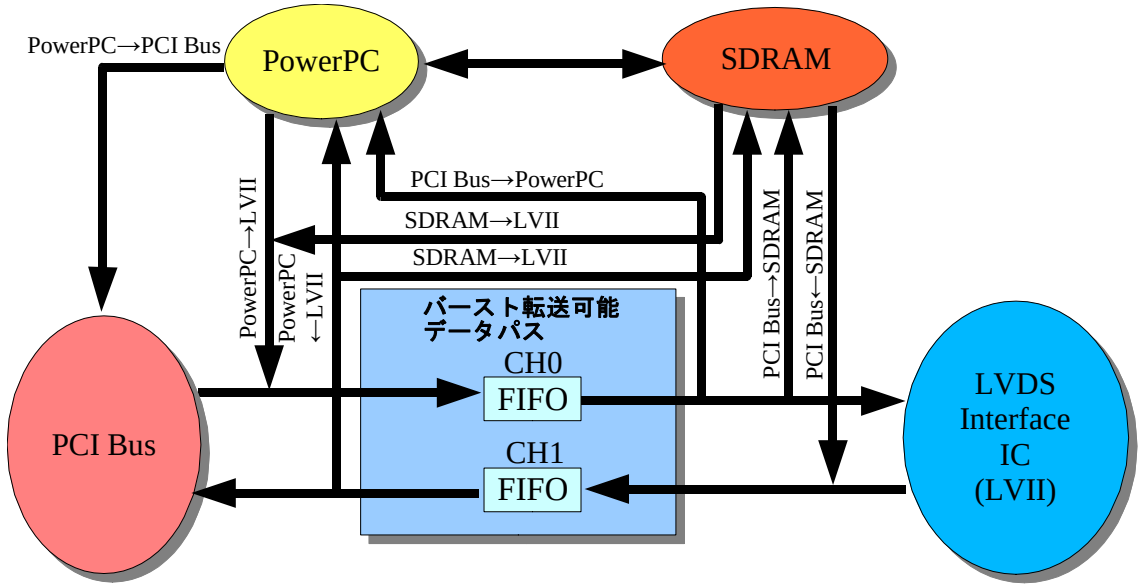

図 5: FPGA 内のデータパス

タリンク層に相当する。MLX32 の入力は 64 ビット 66MHz で、出力は 32 ビット 100MHz であるので、出力側の最大ビットレートは 3.2Gbps である。MLX32 は山際氏が作製した。

5.2 PCI

PCI は 64 ビット、66MHz の PCI バスとのインターフェースを行う。PCI は、マスタ、スレーブ、コンフィギュレーションの各モジュールに分けられる。

マスタは自分から主体的にデータ転送をするモジュールで、PCI の機能で一番使われる主要なモジュールである。ホスト PC メモリからの読み出し、書き込み双方向とも独立にデータ転送することが出来て、66MHz、64 ビットでバースト転送することが出来る。

PCI 仕様 2.2 では、64 ビット幅の PCI デバイスは、32 ビットのターゲットにアクセスする場合に、64 ビットアクセスとしてデータ転送を始めていても、32 ビットアクセスに変更しなければならないという規定がある。しかし、32 ビットアクセスへの変更は、回路の複雑化、動作スピードの低下、およびスループットの低下を招くため、本マスタでは全て 64 ビット幅でアクセスする。このように、性能向上の観点から、あえて PCI 規格を逸脱した実装になっている部分が数ヶ所ある。

スレーブは、ホスト PC からの PCI データ・アクセスを受け取って、NI のリソースへ書き込んだり、リソースからホスト PC へ読み出したりする。スレーブは 64 ビットと 32 ビットの両方のアクセスに対応する。ペンティアム・プロセッサはデータバスが 64 ビット幅であるが、基本的には、32 ビット・プロセッサであり、キャッシュが禁止されているメモリ領域には、32 ビット・アクセスが行われる。よって、32 ビット・アクセスに対応しないと、データ転送が出来ない。その代わりといっては何だが、スレーブはバースト転送には対応していない。連続的にデータが来ても 1 つのデータ転送で PCI スレーブ・アクセスを切断する。ペンティアム・プロセッサでは、キャッシュ禁止メモリ領域への読み書き(NI への読み書きに相当する)は、単一の PCI スレーブ・アクセスとなる。しかし、サーバーワークスのチップセットでは、連続したアドレスへの読み書きではバースト転送となる場合もあるようだ。

コンフィギュレーションは、主にコンフィギュレーション・アクセスの処理とコンフィギュレーション・レジスタの実装を行っている。

ホスト PC の電源 ON 時には、アドレスマップがホスト PC の BIOS によって、各 PCI デバイスに設定される。このように、PCI のコンフィギュレーション・アクセスは、初期設定をホスト PC から各 PCI デバイスに設定する作業である。コンフィギュレーションの中には、コンフィギュレーション・アクセスでアクセスするレジスタ(コンフィギュレーション・レジスタ)を実装してある。コンフィギュレーション・レジスタには、ホスト PC から PCI スレーブ・アクセスでアクセスする時に必要なベース・アドレスを設定するレジスタなどがある。

5.3 PCI Target Controller

PCI Target Controller(PCITC)は、PCI のスレーブに直結されていて、PCI スレーブ・アクセスの FPGA 内部処理を行う。PCI スレーブ・アクセスがあると、PCITC は起動し、FPGA 内部バス経由で使用するデバイス用のコントローラ(NBL Controller、SDRAM Controller)にアクセスする場合には、前述したようにマスタ・デバイスとして FPGA の内部バスを要求する。Internal Registers と FLASH ROM Controller へアクセスする場合には、PowerPC Interface と共用の専用バスがあるので、PowerPC Interface とバスの調停を行うだけで、それらを使用することが出来る。

なお、NBL Controller、SDRAM Controller にアクセスする場合には、PowerPC の PCI マスタ・アクセスとの競合を避けるために、リソースをロックしなければならない。その機能は、Internal Registers に実装してある。

5.4 PowerPC Interface

PowerPC Interface は、PowerPC プロセッサ(MPC603e,300MHz,バス動作周波数 66MHz)とのインターフェースを実装する。インターフェースの基本構造は Maestro1 と同一なので、大部分を流用した。ただし、使用言語を AHDL から VHDL に変更したので、変換を行った。

PowerPC のメモリ・アクセス手順はアドレス転送とデータ転送に分けられる。PowerPC Interface はアドレス転送を処理して、デバイスへのアクセス要求を受け取る。ここから、PowerPC のデータ転送が始まる。もし、要求が LVDS インターフェス用 IC か PCI バスへのアクセスの場合には、PowerPC Interface は、内部バスを使用するので、System Arbiter にデバイスを要求する。その後、デバイスが使用可能になった時に、許可を受け取る。次に、PowerPC Interface は、そのデバイスの管理を受け持つコントローラーとデータをやり取りする。データのやり取りが終了したら、PowerPC のデータ転送が終了し、完了を送出して終了する。

PowerPC の要求が Internal Registers か FLASH ROM へのアクセスの場合は、専用のアービタで PCI スレーブとの調停をおこなった後に、データ転送を行う。この場合のデータ転送は、全て単一転送となる。

5.5 Internal Registers

Internal Registers は、FPGA 内部の制御レジスタや状態レジスタ、PCI バス用のコンフィギュレーション・レジスタなどが実装されている。Internal Registers は、PowePC と PCI スレーブ・アクセスから専用バスで書き込み、読み出しができるが、バースト転送は出来ずに、単一転送のみとなる。

アドレス・オフセットが 00h(16 進数)から 3Fh までは、PCI バスのコンフィギュレーション領域となっている。デバイス ID、ベンダ ID、ベースアドレス・レジスタなどの、各種レジスタが実装されている。

アドレス・オフセットが 40h から FFh までは、FPGA 内部の 22 個の制御、状態レジスタが実装されている。DMA 設定用レジスタもここにある。面白い機能としては、FPGA を初期化する設定ビットもあり、ここを設定すると、もう一度 FLASH ROM からコンフィギュレーション・データを FPGA に読み込んで、FPGA が完全に初期化される。

アドレス・オフセットが 100h から FFFh までは、内部 Block RAM 領域である。FPGA 内部の Block RAM を SRAM(Static Random Access Memory) としてマップしてある。Internal Registers としてのアクセスなので、単一転送のみであるが、ホスト PC からアクセスする時に、SDRAM のようにリソースの競合を防ぐためのリソース・

ロックが必要ないため、結果としてSDRAMよりも高速にアクセスが出来る。

5.6 NBL Controller

NBL Controllerは、MLX32とのインターフェースを行う。NBL Controllerは、入出力共に3エントリのFIFOを持っていて、MLX32と内部データバスのデータのプロトコル変換を行う。特にMLX32からのデータ読み出しの場合は、イネーブルを入れてから、1クロック後にデータが出てくるので、内部バスのプロトコルと合わない。よって、NBL Controllerでイネーブルと同時にデータが出力されるように変換を行う。

NBL Controllerは、MLX32のバッファに読み出すデータがあるかどうか、また、MLX32のバッファにデータが書き込めるかどうかを常時監視している。その結果、データ転送が行えない場合には、イネーブルを落として、FPGA内の伝送相手のデバイスを停止する。

5.7 SDRAM Controller

SDRAM Controllerは、66MHz動作、64ビット幅、64MbyteのSDRAMを制御する。SDRAMとPowerPCのデータバスは共通となっていて、PowerPCとのやり取りでは、PowerPCとSDRAM間で直接データをやり取りし、FPGA内とのデータのやり取りはない。(図2参照)

SDRAMの初期化時には、モードレジスタにバースト長やアクセス方法などを設定する。それにより、バースト長が決定されるが、本SDRAM Controllerではバースト長を1に設定してある。それは、PowerPCの非キャッシュメモリ領域のアクセスの時に、単一転送とすることが出来るからである。バースト転送時の動作は以下で説明する。

SDRAMは、行アドレスと列アドレスの2つのアドレスを入力する。始めに行アドレスを入れておいて、その後、READ又はWRITEコマンドと共に列アドレスを入力する。ここで転送を終了させれば、バースト転送長は１にセットされているため単一転送となる。バースト転送時にはREAD又はWRITEコマンドと列アドレスを連続的に入れることで、バースト転送を実現している。バースト転送は列アドレスが9ビットなので、512個までしか行えない。それ以上は、一度バースト転送を中断して、もう一度、行アドレスを入れなおさなければならない。

SDRAMは、ダイナミック・メモリなので、そのままではデータが消えてしまう。データを消さないためには、15.6usecごとにリフレッシュを行って、データの再書き込みを行う必要がある。SDRAMとPCIバス間のバースト・データ転送の際にこの点が問題となった。定期的にリフレッシュを行う必要がSDRAMにはあるが、PCIバスは、どの位の時間でデータ転送が終了するかは、ホストPCの都合による。つまり、バースト転送中にリフレッシュを必要とするタイミングになってしまうかもしれない。そこで、その時には、PCIバスの方を待たせておき、SDRAMの方はいったんデータ転送を打ち切って、リフレッシュを行うこととした。リフレッシュ後に、SDRAM Controllerは残りのアドレスを出力し、続きを行う。この辺が一番苦労した箇所である。リフレッシュによる中断そして再開が15.6usecごとにあるために、なかなかバグを取りきれなかった。バグが出る条件の特定に時間がかかってしまった。

5.8 DMA Controller

DMA ControllerはDMAの制御を受け持っている。DMA Controllerは制御だけを受け持ち、実際のデータはSystem Arbiter内にあるFIFOを経由する。図5に示した”バースト転送可能データパス”がそれである。そのFIFOはバイト・レーン変換機能を持っている。

DMA Controllerは、2つのスレーブ・デバイスにコマンドを送って、相互間でデータ転送を行わせる。最初に、ソース・デバイス、ディスティネーション・デバイス、バースト長などをInternal Registers内のDMA設定レジスタに書き込むことによって、DMA転送は、起動する。

DMA転送のチャネルは前述してきたように2チャネルあるので、設定用のレジスタも2チャネル分用意されている。各チャネルは、転送方向が決まっている。チャネル0はPCIから読み出してLVDSへ出力、PCIから読み出してSDRAMへ書き込み、SDRAMから読み出してLVDSへ出力の3種類のDMA転送をサポートする。(図5参照)

チャネル1はLVDS入力からPCIへ出力、LVDS入力からSDRAMへ書き込み、SDRAMから読み出してPCIへ出力の3種類である。(図5参照)

このように、転送可能デバイスを分けることによって、データ入力用マルチプレクサの回路規模が小さくなるため、FPGAのリソースを節約することが出来る。

DMA転送のバースト長の最大値は、64Mbyte-1である。これは、主にSDRAMの容量が64Mbyteであるためである。設定用のレジスタに書かれたDMA転送は、DMA Controllerで、内部的に最大256バーストの転送単位に分けられて実行される。特に、SDRAMは列アドレスが9ビットで512のアドレスであるため、512の境界では、DMA転送を終了して、再起動しなくてはならないためでもある。

その他に、DMA転送は便利な機能を持っている。それはバイト・レーンの変換機能である。

ペンティアム・プロセッサ、PowerPCプロセッサやNIの内部バスなどは64ビット幅のデータバスを持っている。通常のCPU(Central Processing Unit)は、8ビットCPUからの慣習で8ビット単位のデータの塊(バイト)を1単位として、アドレスを付加している。64ビットのデータバス幅を持つCPUで、バイト単位にアクセスすると

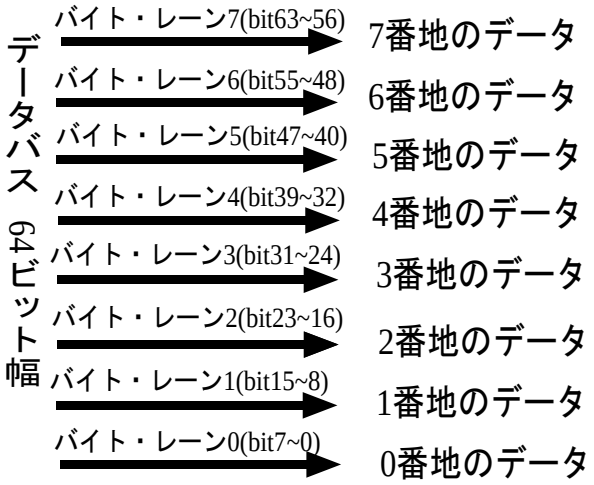

図 6: x86プロセッサのバイト・レーン

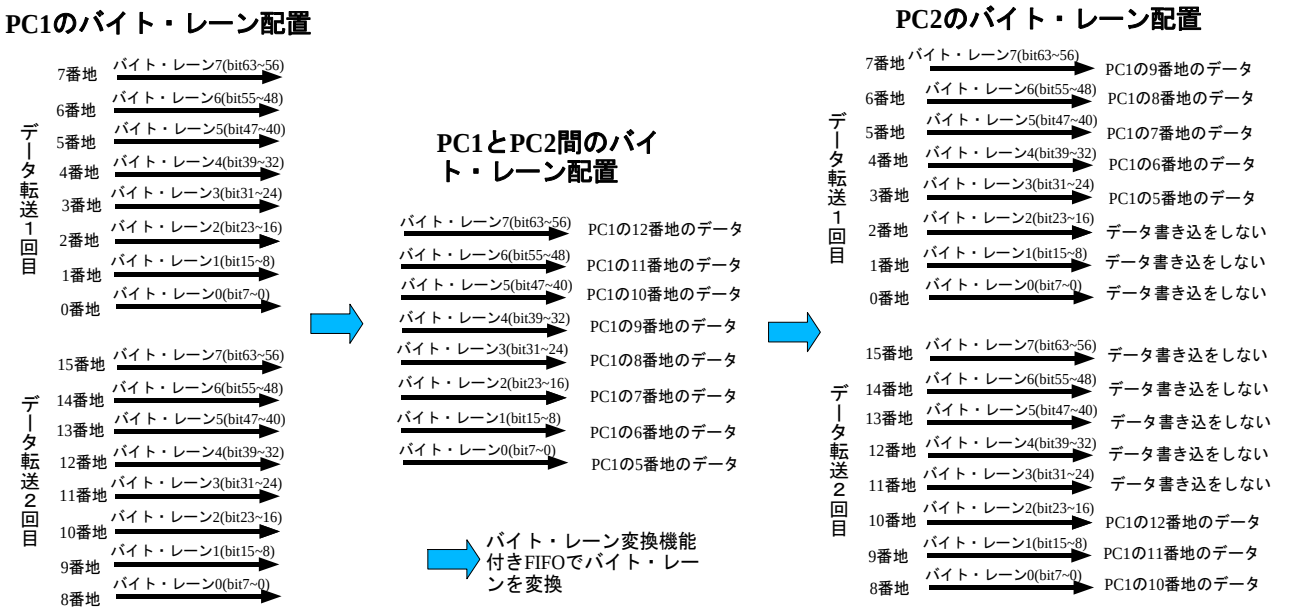

図 7: バイト・レーン変換機能を使用したデータ転送

64 ビット幅のうちの 8 ビットを使用することになる。例えば 0 番地から 7 番地までのバイト・アクセスでは、64 ビット幅のうち、違った場所のデータバスを使用することになる。これをバイト・レーンと呼ぶ。バイト・レーンを図 6 に示す。

以上のことを踏まえて、バイト単位でデータを読んで、それを違った番地に書き込むことを考えてみよう。まず、8 バイトの境界から、8 バイト単位でアクセスしていれば、問題はない。しかし、そうでないと、単純なデータ転送では問題の生ずる場合がある。例えば 5 番地のデータをその他の PC に送ろうとした時に、送り先の PC では 3 番地にデータを入れたいとしよう。その場合には、5 番地のバイト・レーンと 3 番地のバイト・レーンが異なるため、単純に 64 ビット幅のデータを転送していたので、正しくデータを転送することが出来ない。

図 6 で説明すると、5 番地はバイト・レーン 5 だが、3 番地はバイト・レーン 3 になり、データを 64 ビット幅でそのまま転送すると、自分の PC の 5 番地からのデータを相手方の PC の 3 番地からに転送することは無理である。その問題を解決するために、DMA 転送時にバイト・レーン変換機能を使用する。バイト・レーン変換機能は前述した System Arbiter 内にある FIFO に実装されている。これを BLTFIFO と呼ぶことにする。

バイト・レーン変換機能を説明するために、自分の PC(PC1 とする)の 5 番地から 12 番地のデータを他の PC(PC2 とする)の 3 番地から 10 番地に送るとしよう。まずは、PC1 で 0 番地から 15 番地までのデータを DMA で読み出す。次に BLTFIFO は 0～4 番地、13～15 番地までのデータは捨てて、5～12 番地までのデータをバイト・レーン 0 から詰めなおし(この転送を Packed 転送と呼ぶことにする)、詰めなおしたデータを PC2 に送る。PC2 の BLTFIFO は、3 番地のバイト・レーン(バイト・レーン 3)にもう一度、詰めなおして、PC2 のメモリに DMA で書き込む。その際に、データは 8 バイト分だけなので、余計な分(0～ 2 番地、11～15 番地)は、書き込まない(この転送を Unpacked 転送と呼ぶことにする)。その様子を図 7 に示す。

BLTFIFO はバイト・レーン変換機能と FIFO から構成されている。バイト・レーン変換機能は図 8 の DFF0~7、MUX1、TR0~6、MUX2-0~7 から構成される。

前述の Packed 転送の場合は、まずは必要な 5～7 番地のデータを MUX1 でバイト・レーンを変換して

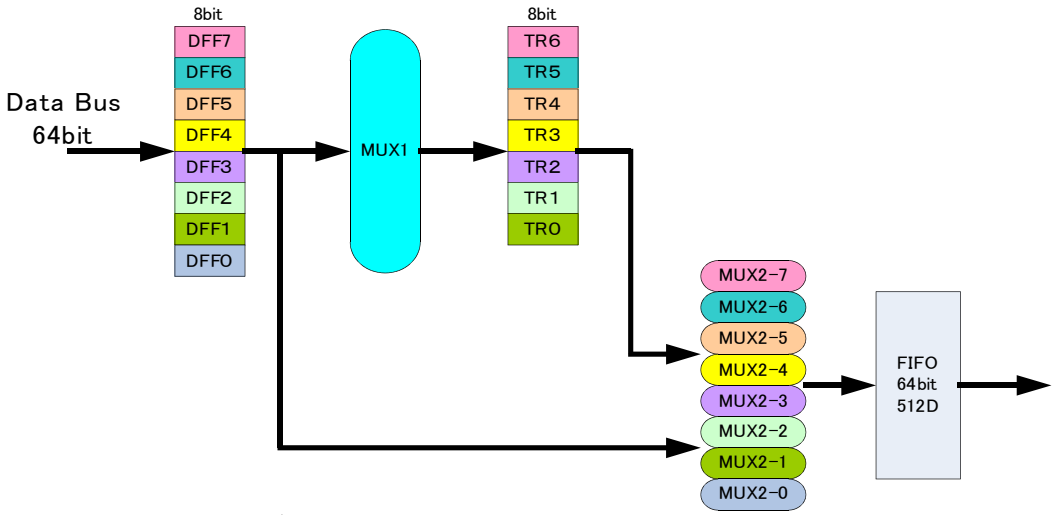

図 8: BLTFIFO の構成

TR0~2 にラッチする。次のクロックでDFF から出力されている 8～15 番地のデータから 8～12 番地のデータを MUX2-3~7 で選択して FIFO に入力する。MUX2-0~2 は TR0~2 のラッチされたデータ(5~7 番地のデータ)を選択して FIFO に入力する。こうして、FIFO には 5～12 番地の 64 ビット幅 1 個に詰め替えられたデータが入力され、他の PC に転送される。

Unpacked 転送の場合は、PC1 の 5～12 番地のデータのうち 5～9 番地のデータだけが(図 8 の DFF0~4)は MUX2-0~4 を通って FIFO に入力される。PC1 の 10～12 番地のデータ(図 8 の DFF5~7)は TR0~2 にラッチされる。次のクロック(つまり FIFO の次のエントリ)で、TR0~2 にラッチされたデータは、MUX2-0~2 を通って FIFO に入力される。入力されなかったデータは不定となる。

６．まとめ

本報告では、クラスタ向けのネットワークインターフェース(NI)の開発について述べた。本 NI は、PC メモリ上のデータを 0 コピーで転送する機能により、高速にデータを他の PC へ伝送できる。NI のピンポン転送時の実効最大スループットは 2.4Gpbs(Giga Bit Per Second)である。

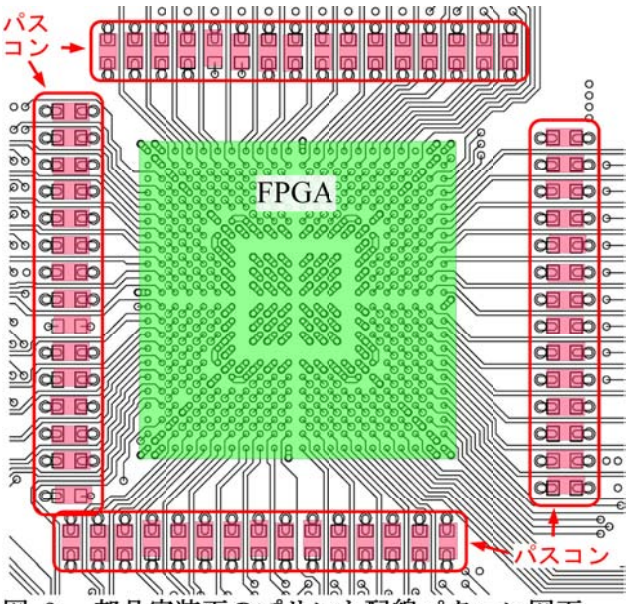

図 9: 部品実装面のプリント配線パターン図面

NI 制作上の反省点としては、FPGA の電源用バイパス・コンデンサ(パスコン)の位置が適切でなかったことが挙げられる。パスコンは当初あまり重要だとは考えてなかったため、パターン設計会社に実装位置を決定してもらった。図 9 に部品実装面のプリント配線パターン図面の一部を示す。図 9 の緑色の四角形が FPGA、赤色の長方形がパスコンである。パスコンは FPGA の周りに実装されているのがわかる。このパスコンの位置では、FPGA の電源からの距離が長くなり、電源のインピーダンスが低くならないという欠点があることがわかった。その結果として、100MHz 動作の LVDS インターフェース用 IC へのデータパスにグラント・バウンズによるデータの誤りが生じてしまった。現在は、IO の出力電流値を 12mA から 4mA に変更したところ、データの誤りはなくなった。本来ならば、パスコンの大きさを小さくして、裏面の FPGA の直下に実装し、FPGA の電源からの距離をより小さくしなければならない。

７．謝辞

Maestro2 システムの開発補助をさせていただいたシステム情報工学研究科コンピュータ・サイエンス専攻の和田耕一教授に深く感謝いたします。また、共同制作者である INESC-ID の山際伸一氏とシステム情報工学研究科の青木圭一氏に深く感謝いたします。

参考文献

[1] 小野雅晃,Xilinx 社製 FPGA を搭載した PCI ボードのシミュレーション,平成 15 年度高エネルギー加速器研究機構技術研究会報告集ポスターセッション(2004)

[2] Shinichi Yamagiwa, Keiichi Aoki, Masaaki Ono, Tetsuya Sakurai, Koichi Wada, and Luis Miguel Campos. Maestro2: A new challenge for high performance cluster network. In The 6th World Multiconference on Systemics, Cybernetics and Informatics, volume XI, Computer Science II, pp.382-387, 2002.

[3] Keiichi Aoki, Shinichi Yamagiwa, Masaaki Ono, Koichi Wada, Luis Miguel Campos. An architecture of high performance cluster network : Maestro2. In 2003 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing(PacRim03), 2003.

[4] Shinichi Yamagiwa, Kevin Ferreira, Luis Miguel Campos, Keiichi Aoki, Masaaki Ono, Koichi Wada, Munehiro Fukuda, Loenel Sousa. On the Performance of Maestro2 High Performance Network Equipment, Using New Improvement Techniques. In 23rd IEEE International Performance Computing and Communications Conference(IPCCC 2004), 2004.

[5] Keiichi Aoki, Shinichi Yamagiwa, Kevin Ferreira, Luis Miguel Campos, Masaaki Ono, Koichi Wada, Leonel Sousa. Maestro2: High Speed Network Technology for High Performance Computing. In 2004 IEEE International Conference on Communications (ICC 2004), 2004.