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

渡邊 幸之介† 大塚 智宏† 天野 英晴†

乗っ取り機構は新しい形態のハードウェア/ソフトウェア協調処理である．乗っ取り機構では，オンチッププロセッサがハードウェアのステートや内部レジスタを任意に書き換えることで，ハードウェア処理の一部をソフトウェア処理に置き換えることや，逆にソフトウェア処理の一部をハードウェアモジュールの機能を用いて高速化することが可能となる．我々は，この乗っ取り機構を RHiNET のネットワークインタフェース用コントローラチップである Martini に実装し，その有効性について評価を行った．評価の結果，乗っ取り機構を実装することで数%程度回路規模が増大するものの，効率的な例外処理が可能となるうえ，ソフトウェア通信処理の大幅な高速化が実現できることが分かった．

Taking Over Mechanism on the Network Interface Controller Chip Martini

Konosuke Watanabe,† Tomohiro Otsuka† and Hideharu Amano†

"Taking over mechanism" is a novel framework for a hardware/software cooperation. In this mechanism, an on-chip processor partly emulates a certain operation of hardwired logic, or uses a hardware module as an accelerator during software operation by stopping a state machine and accessing to registers of the module. We implemented this mechanism on Martini: a network interface controller chip of RHiNET. Evaluation results show that the taking over mechanism makes exception handling efficient, and moreover, it greatly accelerates software communication processing with a few percent of hardware increase.

1. はじめに

近年，汎用的なプロセッサ，メモリ，専用ハードウェア，入出力インタフェース等の，システムを構成する要素を単一チップに集約したシステム LSI の開発がさかんである．一般的なシステム LSI は，各構成要素がバスで接続された形態（**図 1**）を基本としており，マルチメディア向けチップ等の開発においては，設計の段階でオンチッププロセッサと専用ハードウェアとの処理の分担を決定しておくことでハードウェアとソフトウェアの開発を同時に進める協調設計が広く行われている[1),2)]．

一方，Myrinet[3)] の LANai 等，ハイエンドな PC クラスタ向けネットワークのインタフェース用コントローラでは，プロセッサが中心となり，DMA コントローラ等の周辺ハードウェアをソフトウェアで制御することで通信処理を行う場合が多い．この方式は，ソフトウェア次第で様々な通信機構を柔軟に提供できるという優れた面を持つが，一方で通信にソフトウェアが介在することにより，PC クラスタにおいて重要な転送遅延や最大バンド幅に制限が生じやすいという問題がある．

これに対し，我々が提案・実装を行ったネットワーク RHiNET[4)] は，ネットワークインタフェースでの主要な通信処理を完全にハードウェア実装し，処理にソフトウェアを介在させないことで低遅延かつ高バンド幅な通信を提供する[13)]．

RHiNET のネットワークインタフェース用コントローラである Martini[8)] は，専用ハードウェア，メモリ，入出力インタフェースを備え，ハードウェア単独での通信処理が可能なチップである．Martini では，ネットワークインタフェースで必要とされるすべての処理を完全にハードウェア実装するとハードウェア規模が巨大化してしまい，実装コストの大幅な増加，ひいては全体の動作速度の低下を招いてしまうことか

† 慶應義塾大学
Keio University

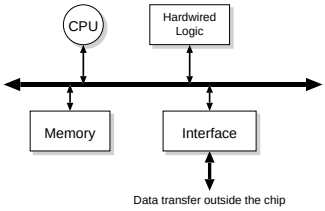


図 1 一般的なシステム LSI の接続モデル
Fig. 1 Typical connection model of a system LSI.

ら，性能が要求される使用頻度の高い基本通信処理のみをハードウェア実装し，使用頻度が低くあまり性能を必要としない例外処理等は，内部に設けたオンチッププロセッサ上でソフトウェアで処理する構成をとっている．

また，Martini では，ハードウェア実装されていない通信機構についてもオンチッププロセッサでソフトウェア処理することになるが，これらは例外処理と異なり低オーバヘッドな処理が要求される．通常，ソフトウェアで処理する通信機構はパケットヘッダの生成やネットワークへのデータの書き出し等，ハードウェア実装されている基本通信処理と共通の処理をともなうものが多く，ハードウェア実装された基本通信処理を部分的に変更・拡張するだけで実現できる場合が多い．

そこで我々は，例外処理を効率的に行い，ソフトウェアによる通信処理に高い処理能力を提供する，“乗っ取り機構” と呼ばれる新しいハードウェア/ソフトウェア協調処理の方式を提案し，Martini 上に実装を行った．乗っ取り機構は，専用ハードウェアを構成する個々のモジュールのステートや内部のレジスタの値をオンチッププロセッサが自由に変更できるようにすることで，ソフトウェアによるモジュールのステートレベルでの詳細な制御や部分的な処理のエミュレーションを可能とする機構である．

通常，ネットワークインタフェースコントローラは内部の通信処理を行うハードウェアが送信部と受信部で独立している．Martini ではそれらはさらに細かなモジュールに分かれてパイプライン構成となっており，個々のモジュールは独立して動作可能な設計になっている．あるモジュールが例外を発生した場合，通常のシステム LSI 等であれば，専用ハードウェア全体，もしくは例外を発生したブロックが停止した後，オンチッププロセッサが例外の要因を取り除き，例外を発生したモジュールの代わりに残りの処理を行う．これに対し，乗っ取り機構を利用した場合，モジュールが独立してオンチッププロセッサから制御可能であるため，例外を発生したモジュールのみが停止し，そのモジュールと無関係な他のモジュールは例外処理中も並行して動作し続けることができる．またオンチッププロセッサからモジュールのステートを自由に操作できるため，例外の原因を取り除いた後，例外を発生したモジュールを例外発生の直前の状態に遷移させ，ハードウェア処理を再開させることで，残りの処理をソフトウェアで扱う必要がなくなる．

また，乗っ取り機構を利用することで，モジュールの特定のステート間の処理をソフトウェア処理に置き換えることができる．ハードウェアの提供する基本通信処理に近いソフトウェア通信処理を実装する場合に，これを利用することで，ソフトウェアによる処理を最小限に抑え，ハードウェアを活用した効率的な処理が可能となる．

本論文では，Martini における乗っ取り機構の実装とその評価について述べる．以降，2 章で乗っ取り機構の詳細と実現方法を示し，3 章で実装対象である Martini について概要を述べる．4 章では Martini への乗っ取り機構の実装について示し，5 章でその評価結果を示す．最後に 6 章でまとめを述べる．

2. 乗っ取り機構

乗っ取り機構は，同一チップ上にオンチッププロセッサと専用ハードウェアが混在する構成のシステム LSI において，オンチッププロセッサが専用ハードウェアをモジュール単位で一時的に支配下に置き（乗っ取り），詳細に制御を行えるようにする機構である．

モジュールは乗っ取られた状態になるとステートの遷移を停止させ，リソースを解放し，ステートマシンの状態やレジスタの値をオンチッププロセッサから自由に読み書きすることを許可する．これにより，モジュールの特定のステート間での処理を，オンチッププロセッサによるソフトウェア処理に置き換えることが可能となる．これを用いることで，効率的な例外処理や，専用ハードウェアが備える処理を部分的に変更・拡張した処理が実現できる．

このように，乗っ取り機構を用いることで，従来よりも細かい粒度でのハードウェア/ソフトウェア協調処理が可能となる．

2.1 乗っ取り機構の実装

乗っ取り機構を実装するには，ハードウェアモジュールに対して

- 停止状態
- 停止状態への移行手段
- 停止状態下での制御機構

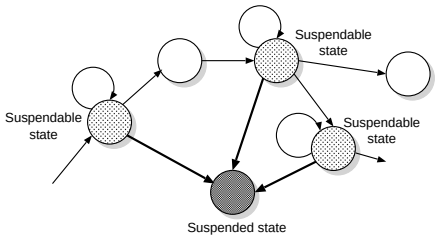


図 2 サスペンド状態
Fig. 2 Suspended state.

を実装する必要がある．

2.1.1 停止状態

これまでに述べたとおり，乗っ取り機構では，乗っ取られたモジュールはステートの遷移を停止してリソースをオンチッププロセッサに解放する必要がある．

モジュールにおいて停止する可能性のあるステートが複数ある場合，各ステートに個別に停止のための機構を設けるよりも，自発的に他のステートへ遷移せずいっさいのリソースを要求しない “停止状態” を示すステートを新たに設ける方が実装面で容易である．このようなステートを “サスペンド状態” と呼ぶ（**図 2**）．他のステートからサスペンド状態に遷移する際に，リソースを解放する機構を設けることで乗っ取り機構に必要な停止状態を実現できる．

また，乗っ取り機構では，乗っ取られたモジュールがサスペンド状態に遷移し，ソフトウェア処理が行われた後は，再びハードウェア処理を再開する必要がある．これを実現するには，サスペンド状態になったモジュールに対して入力を与えたりその出力を利用したりするような他のモジュールを，必要に応じて待機させる構造にしなければならない．

モジュールのステートのうち，サスペンド状態へ遷移した後，ハードウェア処理を正常に再開できるように設計されたステートを “サスペンダブルな” ステートと呼ぶ（図 2）．

乗っ取り対象となるモジュールのステート S がサスペンダブルであるには，以下の条件を満たす必要がある．

- 乗っ取り対象モジュールと依存のある周辺モジュールが，有限ステート遷移した時点で遷移を停止し，ステート S の次のステートにおいて発生する信号を待ち続ける．
- 依存のある周辺モジュールのステートをオンチッププロセッサが認識できるか，乗っ取り対象のモジュールがステート S に滞在している間の周辺モジュールのステートが一意に定まる．

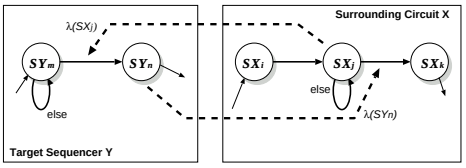


図 3 サスペンダブルなステートの例
Fig. 3 An example of the suspendable state.

以下では，サスペンダブルなステートについて例を示しつつ述べる．

まず，**図 3** に示すように，乗っ取り対象となるモジュール Y が，周辺モジュール X と依存関係にある場合を考える．それぞれのモジュールは，順序機械 MX（IX, OX, SX, σ_x, λ_x）および MY（IY, OY, SY, σ_y, λ_y）で表される．ここで，I は入力集合，O は出力集合，S は状態集合，σ は状態遷移関数，λ は出力関数である．

X は，状態 SX_i から SX_j に遷移する際に，Y に対してハンドシェイク要求出力信号 $\lambda(SX_j)$ を送信し，その後 Y が SY_n に遷移することで出されるアクノリッジ信号 $\lambda(SY_n)$ を受け取らない限り，次の状態 SX_k に遷移しないとする．このような場合，Y が SY_m からサスペンド状態に遷移しても，X は SX_j から先のステートへ進めずに待機する．このため，Y がサスペンドからの復帰時に SY_m から SY_n に遷移することで，X は正常にハードウェア処理を再開することができる．よって，Y において SY_m はサスペンダブルなステートであるといえる．すなわち，サスペンダブルな条件は，以下のとおりである．

周辺モジュール X について：

$$\sigma(SX_j, \lambda(SY_n)) = SX_k$$

$$\sigma(SX_j, \overline{\lambda(SY_n)}) = SX_j$$

乗っ取り対象モジュール Y について：

$$\sigma(SY_m, \lambda(SX_j)) = SY_n$$

$$\sigma(SY_m, \overline{\lambda(SX_j)}) = SY_m$$

のときに，状態 SY_m はサスペンダブルである．ただし，ここで $\overline{O}$ は O の補集合を表すものとする．

一方，Y が SY_n に遷移したかどうかにかかわらず，X のステートが $\lambda(SX_j)$ を出力した後に SX_j から別の状態に遷移してしまう場合，Y が SY_m からサスペンド状態に移行している間に X の処理が進んでしまう．したがって，復帰後に Y が正常なハードウェア処理を再開できない可能性が生じてしまう．このような状況でも，

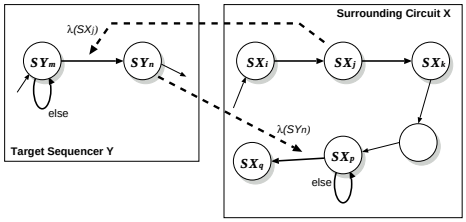


図 4 条件つきでサスペンダブルとなるステートの例
Fig. 4 An example of the suspendable state with conditions.

- X は SX_j から数ステート先の特定のステート SX_p で待機し，Y が SY_n に遷移したことを示す信号 $\lambda(SY_n)$ の入力がない限りその先のステートへは遷移しない，
- オンチッププロセッサが X の現在のステートを知ることができる，

という条件を満たす場合，Y が SY_m からサスペンド状態へ遷移した後，X が SX_p まで遷移したことを確認したうえで Y をサスペンド状態から SY_n に遷移させることができるため，安全に処理を再開できる．このことから，SY_m はサスペンダブルなステートとなる．すなわち，この場合のサスペンダブルな条件は以下のとおりである．

周辺モジュール X について：

$$\sigma(SX_p, \lambda(SY_n)) = SX_q \quad (1)$$

$$\sigma(SX_p, \overline{\lambda(SY_n)}) = SX_p \quad (2)$$

のときに状態 SY_m はサスペンダブルである．

例として，**図 4** に示すような状況を考える．図 4 では X はハンドシェイク要求信号 $\lambda(SX_j)$ を出した後，Y からのアクノリッジ信号を待たずに SX_j から先のステートに進んでしまう．ただし，X は SX_p で Y からのアクノリッジ信号 $\lambda(SY_n)$ を待つものとする．

この場合，オンチッププロセッサは，Y が SY_m からサスペンド状態に遷移した後，復帰する際に，X が SX_p まで到達しているのを確認したうえで，Y のステートを SY_m にせずに SY_n に設定する．このようにすることで，X，Y ともにハードウェア処理を正常に再開することが可能となり，SY_m はサスペンダブルなステートとなる．

2.1.2 停止状態への移行手段

乗っ取り機構では，モジュールが動作を停止しソフトウェア制御下に入る要因として，以下の 2 通りを想定している．

- モジュール自身による停止
- ソフトウェアによるモジュールの停止

前者には，例外発生等でモジュールがハードウェア処理を続行できない状態に陥った際にソフトウェアに処理の続きを依頼する場合が該当する．この場合，ハードウェアモジュールが例外を検出した段階で，リソースを解放してサスペンド状態へ遷移する構造とすればよい．

一方，後者には，ソフトウェア処理中にハードウェアを部分的に利用する場合が該当する．この場合，ソフトウェア処理によってモジュールのステートをサスペンド状態に変更すればよいが，モジュールのステートをいきなりサスペンド状態に遷移させてしまうと，その後ハードウェア処理を正常に再開できなくなる可能性がある．そこで，オンチッププロセッサがモジュールを乗っ取りたい場合，まずモジュールに対して停止要求を発行し，それを検出したモジュールがサスペンダブルなステートからサスペンド状態へ遷移する機構を設ける．これにより，その後のハードウェア処理を安全に再開できるようになる．

2.1.3 停止状態下での制御機構

あるモジュールがサスペンド状態に遷移した場合，オンチッププロセッサはそれを速やかに検出し制御下に置く必要がある．これには，各モジュールからオンチッププロセッサに対して割込み線を設け，サスペンド状態に遷移した段階で割込み線をアサートすることで実現すればよい．

また，乗っ取り機構では，サスペンド状態に遷移したモジュールは，個別にオンチッププロセッサから制御可能でなければならない．これを実現するには，オンチッププロセッサから乗っ取り対象となるすべてのモジュールに対して制御用のバスを配線すればよい．オンチッププロセッサはこれらのバスを経由してモジュール内の各種レジスタの値を読み書きすることになる．

さらに，モジュールがサスペンド状態のステートに遷移しオンチッププロセッサがそれを乗っ取ると，オンチッププロセッサはサスペンド状態へ遷移した原因を確認し，それに応じた適切な処理を行う必要がある．サスペンド状態への遷移の原因は，サスペンド時のモジュールのレジスタの値とサスペンド状態に遷移する直前のステートから判断できる．このために，1 サイクル前のステートを保持するレジスタが必要となる．

3. Martini

以下では乗っ取り機構の実装対象である Martini について述べる．Martini は PC クラスタ用ネットワークである RHiNET のネットワークインタフェース用

コントローラとして開発されたカスタム LSI である．

3.1 RHiNET

RHiNET は，オフィス等で日常業務で使用されている PC を相互接続してクラスタ化し，その余剰計算資源を利用して並列分散処理を行うことを目的としたネットワークである．RHiNET では，独自のネットワークスイッチとネットワークインタフェースを用いて，低遅延かつ高バンド幅な通信を実現する．

Martini は，第 2 世代の RHiNET である RHiNET-2 向けに開発されたネットワークインタフェース用コントローラ LSI である．RHiNET-2 のネットワークインタフェースには，Martini のほか，ワークエリア用の SDRAM と光インタコネクションモジュールが搭載される．また，ネットワークインタフェースはホストに 64 bit/66 MHz の PCI バスを介して接続される☆．

3.2 Martini のハードウェア通信機構

Martini はリモートホストのメモリへの書き込み処理（PUSH）と，リモートホストのメモリからの読み出し処理（PULL）を基本通信機構としてハードウェアで実現する．

ホストから Martini に対して通信要求が発行されると，Martini はアドレス変換やパケットヘッダ生成，DMA 要求の発行等を必要に応じて行い，パケットを組み立ててネットワークへ送出する．パケットを受信した際も，ホストに割込みをかけることなくネットワークインタフェースのみで受信パケットを処理する．これらの処理をすべてハードウェアで行うことで RHiNET-2 は高い基本通信性能を提供する．ソフトウェアによる通信起動時のオーバヘッドを極力削減するため，Martini はこれらの通信処理をユーザレベル・ゼロコピー通信[5] として提供している．

また，Martini はこれらリモート DMA を用いた通信とは別に，“Block On-The-Fly（BOTF）[6]” と “Atomic On-The-Fly（AOTF）[7]” と呼ばれる 2 種類の PIO ベースの基本通信機構を持つ．

以下では，Martini の機能と内部構造について，乗っ取り機構と関係の深い部分を中心に述べる．

3.3 Martini の構成

Martini のおおまかな構成を**図 5** に示す．Martini は PCI インタフェース部（PCII），DIMM インタフェース部（DIMMI），スイッチインタフェース部（SWIF）およびコアロジックで構成される．

☆ Martini は，メモリスロットに装着する形式のネットワークインタフェース DIMMnet-1[11] のコントローラとしても利用可能である．

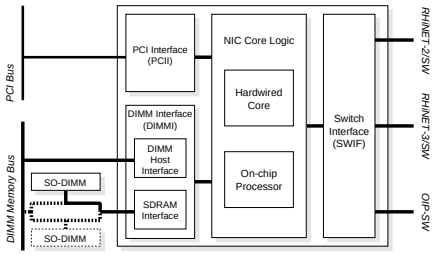


図 5 Martini の構成
Fig. 5 A block diagram of Martini.

PCII は 64 bit/66 MHz の PCI バスに対応し，ネットワークインタフェースを PCI バスに接続する際のホストインタフェースとして機能する．また，DIMMI は外部の SDRAM とのインタフェースとして機能するほか，ネットワークインタフェースを SDRAM メモリスロットに装着する際のホストインタフェースの機能も備える．SWIF は，RHiNET-2 で利用可能な 3 種類のネットワークスイッチ[12),15),16)] に対応し，接続されるスイッチに応じたパケット再送やフロー制御等のプロトコル処理を行う．

コアロジックは Martini の中核部であり，ハードワイヤードコア部とオンチッププロセッサから構成される．オンチッププロセッサは MIPS R3000 と命令互換の 32 bit RISC プロセッサであり，メモリコントローラ，割込みコントローラを内部に持つ．また，チップ内部の SRAM 上に 128 kbyte の命令メモリとデータメモリを持つ．シンプルな 5 段パイプラインの構成となっており，命令拡張はいっさい行われていない．Martini におけるソフトウェア処理はこのオンチッププロセッサが担当する．

3.4 ハードワイヤードコア部

図 6 に Martini のハードワイヤードコア部のブロック図を示す．ハードワイヤードコア部は，ホストからの通信要求やネットワークからの到着パケットの処理を行う要求処理部（Initiator Controller および Remote Controller），DMA コントローラ，ホスト仮想–物理アドレス変換用の TLB（PATLB），AOTF 処理部から構成される．

3.4.1 要求処理部

要求処理部はホスト側からの通信要求を処理する Initiator Controller（**図 7**）とネットワーク側からの到着パケットを処理する Remote Controller（**図 8**）に分かれる．

3.4.1.1 Initiator Controller

Initiator Controller はホストやオンチッププロセッサが Martini のハードウェア部に対して通信要求を発

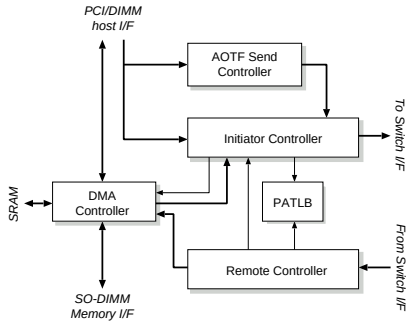


図 6 ハードワイヤードコア部のブロック図
Fig. 6 A block diagram of the hardwired core.

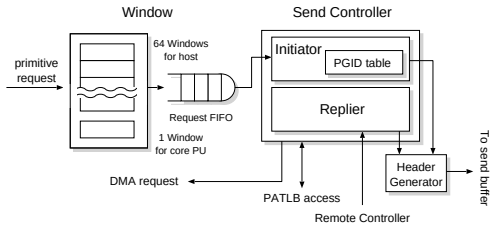


図 7 Initiator Controller のブロック図
Fig. 7 A block diagram of the Initiator Controller.

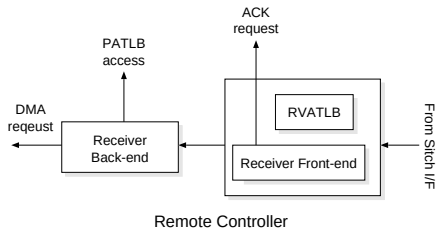


図 8 Remote Controller のブロック図
Fig. 8 A block diagram of the Remote Controller.

行する際の窓口となる Window 部と，パケット生成を行う Send Controller 部で構成される．Send Controller 部はさらに，Window から与えられた PUSH, PULL および BOTF の通信要求を処理する Initiator と，Remote Controller 側からの依頼に応じて応答パケット生成処理を行う Replier に分かれる．

Window は小規模なメモリとコントローラで構成され，メモリ部分はユーザプロセスのアドレス空間にマップされる．ホストから Window の特定のアドレスに書き込みが行われると，Window に書かれた内容が通信要求として解釈され，FIFO を経て Send Controller 内の Initiator へ渡される．Initiator は要求内容の解析を行い，通信要求が PUSH の場合は PATLB を参照して物理アドレスを獲得したうえで DMA コントローラに対して DMA 要求を発行して PUSH パケットを送出する．通信要求が PULL の場合は，リモートノードのネットワークインタフェースにデータ転送を要求する PULL パケットを送出する．また，通信要求が BOTF の場合，Window に書かれた内容をそのままパケットとして送出する．Window に書かれた通信要求がハードウェア処理できないものであった場合や，TLB でミスヒットが発生した場合に，Initiator および Replier は自らサスペンド状態となり，オンチッププロセッサに乗っ取りを依頼する．

3.4.1.2 Remote Controller

Remote Controller は，フロントエンドである Receiver Front-end（RFend）とバックエンドである Receiver Back-end（RBend）から構成される．

ネットワークから到着したパケットは，まず RFend でヘッダ解析が行われ，PUSH や PULL 等のハードウェア処理可能なパケットであれば，通信対象領域のポインタ（Segment ID/SID）を元に TLB（RVATLB）を参照して通信対象領域の仮想アドレスを求める処理が行われる．また，その際，応答を返す必要がある場合は，Initiator Controller 内の Replier に対して応答パケットの生成を要求する．

RFend で得られたヘッダ情報や仮想アドレスは RBend に渡される．RBend では，PATLB を参照して仮想アドレスから受信領域の物理アドレスを取得し，これに基づいて DMA コントローラに DMA 要求を発行する．受信パケットがハードウェア処理できないものであった場合や，アドレス変換時に PATLB や RVATLB でミスヒットが発生した場合，RFend や RBend はサスペンド状態となり，オンチッププロセッサに乗っ取りを依頼する．

4. Martini への乗っ取り機構の実装

Martini のハードワイヤードコア部は，リモート DMA を用いた通信機構において中心的な役割を果たし，他のインタフェース部等と比べて複雑なステートマシンを備えている．これを構成するモジュールは，TLB のミスヒットやハードウェア処理できないパケットの受信等の要因で例外を発生する可能性がある．

また，ハードワイヤードコア部のうち，Initiator Controller および Remote Controller が提供する機能は，ソフトウェアによる通信処理においても必要となるものが多い．

そこで，Martini では Initiator Controller 内の Initiator と Replier および Remote Controller 内の RFend と RBend の 4 モジュールを乗っ取り機構の実装対象とした．

なお，ハードワイヤードコア部にはこれ以外に

表 1 例外の種類と発生箇所
Table 1 Exceptions and requesting modules.

例外の種類	発生箇所
特殊通信命令発行	Initiator
プロテクション違反	Initiator
PATLB ミスヒット	Initiator Replier RBend
特殊パケット受信	RFend
RVATLB ミスヒット	RFend
タイムアウト	全モジュール

AOTF 処理部や DMA コントローラ等が含まれるが，これらは Initiator Controller や Remote Controller と比べて処理が単純であったり，例外処理が例外の要因を除去するだけで完了となるものであったりすることから，乗っ取り機構を通じてこれらモジュールが提供するハードウェア機能を利用することの利点が少ないと判断し，これらに対しては乗っ取り機構の実装を行っていない．

4.1 例外処理

Martini のハードワイヤードコア部や各インタフェース部を構成するモジュールは，例外が発生するとオンチッププロセッサ，もしくはホスト PC に対して割込み信号を発生させる．その際，先に述べた Initiator Controller 内の Initiator と Replier および Remote Controller 内の RFend と RBend の 4 モジュールについては，乗っ取り機構を利用した例外処理が可能な設計になっている．

表 1 に，これら 4 モジュールで発生する例外の種類と，その発生箇所を示す．

各モジュールにおいて，例外が発生しうるステートはすべてサスペンダブルなステートとして設計されている．また，これらのモジュールには，ソフトウェア側からモジュールをサスペンド状態へ遷移させるための手段として，あらかじめ指定されたステートへ遷移したらモジュール自らがサスペンド状態へ遷移するブレークポイント機構が設けられている．なお，本来ブレークポイントはサスペンダブルなステートにのみ設定可能とすべきものであるが，Martini では実験的な用途のためにすべてのステートにブレークポイントが設定可能な設計となっている．

4.2 乗っ取り機構のモジュールへの実装

以下では，Initiator と RFend を例に，乗っ取り機構のモジュールへの実装について述べる．

4.2.1 Initiator

図 9 に Initiator の状態遷移図を示す．

Initiator は，ホストから Window に通信要求が書き込まれると処理を開始し，通信要求のプロテクションの確認（PGID）と通信領域として用いるホストメモリの物理アドレスの取得（IPA および IPA2）を行った後，DMA 要求を発行し（CHK1 および CHK2），処理が完了したことをホストに通知（CONT）して処理を完了する．

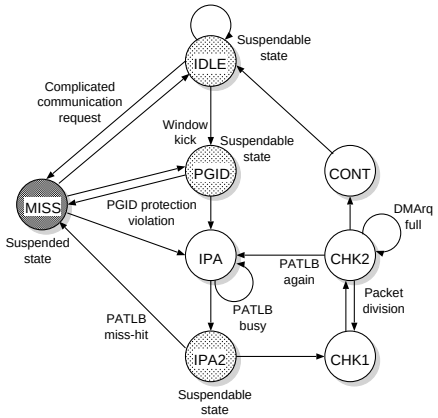


図 9 Initiator の状態遷移図
Fig. 9 State transition of the Initiator.

図 9 において，網かけとなっているステートは例外が発生する可能性のあるステートであり，サスペンダブルに設計されている．オンチッププロセッサは，Initiator からの乗っ取り要求を検出すると，Initiator 内のレジスタから Initiator がサスペンド状態に遷移する直前のステートを読み出し，それを元に乗っ取り要求の原因を判断して処理を行う．以下に，Initiator におけるサスペンドする直前のステートとそれに対応する処理の一例を示す．

IDLE Window から渡された通信要求が PUSH，PULL および BOTF のいずれでもない場合を示す．オンチッププロセッサは要求に対応するソフトウェア処理を行う．

PGID 通信要求が通信を許可されていないプロセスからのものであった場合を示す．オンチッププロセッサは現在 Initiator が保持している通信要求をキャンセルし，Initiator のステートを IDLE に設定することで，次の通信要求の処理を開始する．

IPA2 PUSH を処理する際，送信領域の物理アドレス取得の段階で PATLB がミスヒットしたことを示す．オンチッププロセッサは PATLB のミスヒットしたエントリをリフィルしたうえで，Initiator のステートを IPA に設定し，PUSH の処理を再開する．

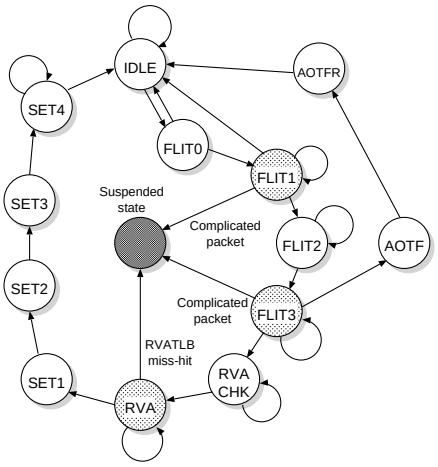


図 10 RFend の状態遷移
Fig. 10 State transitions of the RFend.

4.2.2 RFend

図 10 に RFend の状態遷移図を示す．

RFend は，ネットワークからのパケットの到着により処理を開始する．まずパケットヘッダの解析を行い（FLIT0–FLIT3），次に SID を仮想アドレスに変換するアドレス変換を行う（RVACHK および RVA）．その後，ヘッダの解析結果や変換後のアドレスが RBend や Replier へ渡される（SET1–SET4）．受信したパケットが AOTF による特殊なパケットであった場合，低遅延で受信するために別ステートで処理される（AOTF および AOTFR）．

図の網かけのステートはサスペンダブルに設計されている．以下に，RFend におけるサスペンドする直前のステートとそれに対応する処理を示す．

FLIT1 ネットワーク制御用の特殊パケットが受信された場合を示す．オンチッププロセッサはパケットに対応した処理を行い，その後 RFend のステートを IDLE に設定することで次のパケットの受信に備える．

FLIT3 ハードウェア処理できないパケットが受信された場合を示す．オンチッププロセッサはパケットに対応するソフトウェア処理を行う．

RVA RVATLB でミスヒットが発生したことを示す．オンチッププロセッサは何らかの方法で RVATLB のミスヒットしたエントリをセットしたうえで，RFend のステートを RVACHK に設定し，ハードウェアによるパケット受信処理を再開する．

5. 乗っ取り機構の評価

これまでに述べたとおり，Martini では乗っ取り機構による例外処理や，細粒度のハードウェア/ソフトウェア協調処理が可能である．以下では Martini 上で，乗っ取り機構を利用した処理について評価を行い，その有効性について検討する．

表 2 ノード PC の仕様
Table 2 Specification of each node.

CPU	Intel Pentium III 933 MHz × 2 (SMP)
Chipset	Serverworks Serverset III HE-SL
Memory	PC133 SDRAM 1 Gbyte
PCI bus	64 bit/66 MHz
OS	RedHat Linux 7.2 (kernel 2.4.21)

5.1 評価環境

乗っ取り機構の評価は RHiNET の実機を用いて行った．ただし，実機では計測不可能な一部の評価については，RTL シミュレーション[17] を用いた．

評価に用いた実機は，RHiNET-2 用のスイッチの 1 つである RHiNET-2/SW 1 台に対し，ネットワークインタフェース上に Martini を搭載したノード PC を複数台接続した構成をとっている．ノード PC の仕様を**表 2** に示す．

Martini のオンチッププロセッサとハードワイヤードコア部はともに 66 MHz で動作している．また，評価に用いたネットワークの理論上の最大伝送能力は 6 Gbps である☆．

5.2 例外処理

乗っ取り機構を導入することで，例外発生時に例外発生モジュールのみを停止させてオンチッププロセッサによる例外処理を行うことが可能となるため，例外を発生したモジュールと無関係な他のモジュールは，処理を中断することなく，例外処理中も並行して動作することができる．また，オンチッププロセッサが例外の要因を取り除いた後，処理が残っているモジュールのステートを例外発生前のステートに設定してモジュールの動作を再開させることで，これをハードウェアで処理できる．

以下では，これら例外処理における乗っ取り機構についての評価を行う．

5.2.1 例外処理の他のモジュールへの影響

以下では，Martini 上で送受信のどちらかで例外を多発させた際の，もう一方の例外の発生していない側の処理能力への影響について評価する．評価では，Martini を搭載したノード PC を 3 台用い，**図 11** に

☆ RHiNET-2/SW で利用する光ファイバは本来 8 Gbps の伝送能力を持つが，RHiNET-2/SW の製造上の問題により現在周波数を 600 MHz まで低下させて利用しているため伝送能力は 6 Gbps となっている．

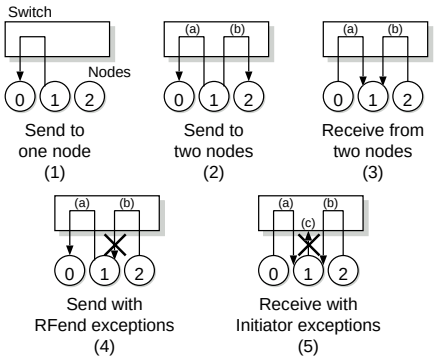


図 11 測定データ転送パターン
Fig. 11 Measured data transfer patterns.

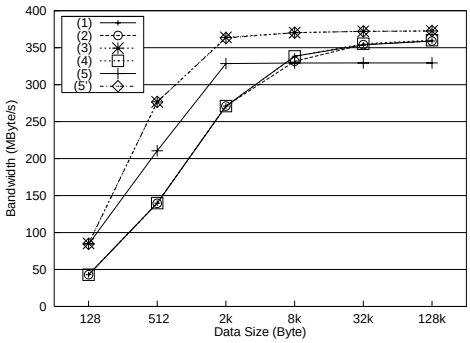


図 12 乗っ取り処理時の送受信バンド幅
Fig. 12 Bandwidth with taking over.

示す 5 通りのパターンで一定サイズのデータの PUSH を連続して発行した際の，送信ノードでの一定時間における送出量の合計を測定した．一度の PUSH で送出するデータサイズは 128 Byte，512 Byte，2 kByte，8 kByte，32 kByte，128 kByte の 6 通りについて測定を行った．

図 11 の (1) は，3 ノード中 2 ノードのみを用いて，一方が PUSH によるデータ送出を続け，もう一方がそれを受信し続けるパターンである．(2) は 1 ノードがその他 2 ノードに対して交互にデータ送出を行うパターンであり，(3) は逆に 2 ノードが 1 ノードに対して同時にデータ送出を行うパターンである．(2) では受信のみを行うノードが分散されることからノードのデータ送信能力の上限を見ることができ，また (3) では送信ノードが複数存在することからノードのデータ受信能力の上限を見ることができる．(1) での結果を (2) および (3) の結果と比較することで，1 対 1 通信におけるバンド幅が送信処理と受信処理のいずれにより制限されているかが分かる．

(4) は，(1) と同じ状況で，さらにノード 2 が送信ノード（ノード 1）に対してデータを送出し続けるパターンである．ただし，ノード 2 から送出されるパケット（図中 (b)）はノード 1 の RFend のステート RVA で必ず例外を発生させ，オンチッププロセッサにより読み捨てられる．また (5) は，(3) と同じ状況で，さらにノード 1 が PUSH 要求（図中 (c)）を発行し続けるパターンである．ただし，この要求は必ず Initiator のステート IPA2 において例外を発生させ，オンチッププロセッサによってキャンセルされる．なお，(5) については，比較のためにノード 1 で例外が発生した後，通信要求をキャンセルせずに Initiator のステートを IDLE へ戻すようにした場合（以下 (5')）についても測定を行った．この場合，ノード 1 の Initiator は一度目の PUSH 要求で IDLE–PGID–IPA–IPA2–MISS の遷移を繰り返し，例外を発生し続けることになる．(4) および (5) の結果を (1) および (3) の結果と比較することで，例外処理が，例外と無関係なモジュールへ及ぼす影響を確認することができる．

図 12 に，図 11 に示した各パターンでの送信バンド幅の総和を示す．ここで，例外処理と無関係な部分についての結果を見るために，(4) については (a) のみ，(5) および (5') については (a) と (b) のみの和となっている．

図 12 より，(1) の結果は (2) の結果とほぼ一致し，(3) よりも低い値であることから，1 対 1 通信時のバンド幅はノードのデータ送信能力の上限により制限されていることが分かる．

(4) ではノード 1 の Martini 上の RFend において例外が多発し，常時オンチッププロセッサが到着パケットを読み捨てる処理を行っているが，その間のノード 1 からノード 0 への送信バンド幅は (1) のバンド幅とほぼ一致している．このことから，ノード 1 の送信処理を行うモジュールは RFend での例外処理の影響を受けていないということがび分かる．

(5) ではノード 1 の Initiator において例外が多発し，常時オンチッププロセッサが通信要求をキャンセルする処理を行っている．この場合，他の 2 ノードから受信されるデータのバンド幅はデータサイズ 2 kByte 以上でほぼ一定値となってしまい，全体的に (3) に比べて低い．一方，(5') の結果を見ると，Initiator で例外処理のみが繰り返されている場合は (3) と全く等しいバンド幅が得られている．これより，(5) が (3) に比べて低い値を示しているのは Initiator での例外処理の多発が直接の原因でないことが分かる．乗っ取り機構と直接関係ないため詳細について触れないが，(5) でバンド幅が制限されているのはノード 1 での PUSH

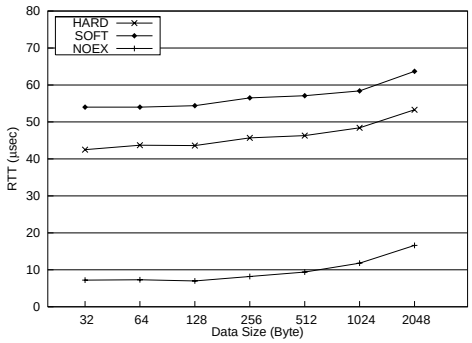


図 13 PATLB ミスヒット発生時の RTT
Fig. 13 RTT with PATLB miss.

要求の密な発行による PCI バスの混雑が原因である．

(3) および (5') の結果より，送信側もくは受信側での乗っ取り機構による例外処理は，もう一方の例外を発生していない側の処理能力に影響しないことが分かる．

5.2.2 例外処理後のハードウェア処理の再開

以下では，例外処理を行った後，例外発生モジュールを例外発生前のステートへ遷移させることでハードウェア処理を再開させることの効果について評価する．

評価は，2 ノード間での PUSH による Ping-Pong において，双方の Initiator でデータ送出時に PATLB のミスヒットが発生した際に，PATLB をリフィルした後，

- Initiator のステートを IPA に戻しハードウェアによる PUSH 処理を再開させた場合（HARD），
- パケットヘッダ生成や DMA 転送等の処理をすべてソフトウェアで続けた場合（SOFT），

のそれぞれについて，ラウンドトリップタイム（RTT）の測定を行った．結果を**図 13** に示す．参考値として，PATLB のミスヒットがない場合の RTT（NOEX）をあわせて示した．

各データサイズにおける RTT は HARD の方が SOFT よりもつねに約 10 μsec 小さい値を示しており，また，データサイズによるこれらの値の変化は NOEX の値の変化とほぼ同一である．このことから，データサイズ増加時の RTT の増加は転送データ量に起因するものと考えられ，HARD と SOFT の RTT の差は Initiator における PATLB リフィル後の PUSH 処理をすべてソフトウェアで実行した際のオーバヘッドによるものであるといえる．ここでは，乗っ取り機構を導入してハードウェア処理を再開させることで，32 Byte 転送時に約 21%，2048 Byte 転送時に約 16%の処理時間の低減を実現している．

5.3 ハードウェア未対応の通信処理

以下では乗っ取り機構によりハードウェア処理の一部をソフトウェア処理で置き換えることで実装した通信機構 VPUSH について述べ，その評価を示す．

5.3.1 VPUSH 機構

Martini の PUSH 機構では，書き込み先の領域は PUSH を要求したプロセスにより SID を用いて指定される．この SID は，受信側のネットワークインタフェース上で受信領域の開始仮想アドレスに変換されるポインタであるが，Martini の設計上，受信側ではこの SID に対応する仮想アドレスをデータ受信時に動的に変更することができない．そのため，複数ホストからの PUSH パケットを到着順に連続した領域に受信するといった処理をハードウェア単独で実現することは難しい．

一方，SCore[9] の通信ライブラリである PM[10] 等のメッセージパッシング型の通信モデルでは，キュー状のデータ受信バッファを用いて複数のホストからのデータを到着順にバッファに格納する必要がある．このような機構を RHiNET で提供する場合，通信相手ごとに独立した受信バッファを用意し，送信側のホストで PUSH のたびに送信先の先頭アドレスをずらす等の方法で擬似的に実現しなければならない[14]．

キュー状の受信バッファを擬似的に実現した場合，データ転送に PUSH をそのまま用いることができるため高い通信性能が得られる．しかし，通信相手ごとに受信バッファを用意しなければならないため，全体のノード数が増大した場合に大量のメモリを受信バッファとして確保する必要が生じてしまう．また，通信相手ごとに用意した受信バッファへのデータ到着をラウンドロビンで確認することになるため，真にパケット到着順にバッファからデータを取り出すことができない．

この問題は，データ受信先のアドレスを受信側で動的に指定可能な通信機構をソフトウェアで実装することで解決できるが，ここで必要となる処理は通常の PUSH の処理と同一のものが多い．そこで，このような通信機構として，通常の PUSH の処理の一部を改変した VPUSH 通信機構を，乗っ取り機構を用いて実装した．

5.3.2 VPUSH の概要

今回実装した VPUSH は，ホストメモリ上にリング状の受信バッファを設け，データが到着した順に先頭から格納されるように，受信時にアドレスを動的に設定する方式とした．受信バッファは，ホストが受信領域のどこまで処理したのかを示すテイルポインタ

と，どこまでが有効なデータであるかを示すヘッドポインタを持つ．テイルポインタはホストから更新し，Martini のオンチッププロセッサが必要に応じてこれをポーリングすることから Martini 内のレジスタに設ける．一方ヘッドポインタはオンチッププロセッサから更新を行うが，ホストの受信プロセスがデータ到着をポーリングする際に頻繁に読み出しを行うため，Martini 内のレジスタに置いてしまうと PCI への負荷が大きくなってしまい転送効率が下がる．そこでヘッドポインタはホストメモリ上に設け，オンチッププロセッサからは DMA を用いて更新を行う．

5.3.3 VPUSH の実装

VPUSH の受信処理はオンチッププロセッサのソフトウェアが行うが，Martini にはオンチッププロセッサがパケットの到着をポーリングする機構が備わっておらず，またパケット到着の段階からソフトウェア処理を開始してしまうとパケットヘッダの解析もソフトウェアで行う必要が生じてしまい効率が悪い．そこで，VPUSH のパケットを受信した場合，RFend がヘッダ解析を完了した段階で例外を発生させ，ソフトウェア処理を開始するようにする．

VPUSH パケットを特殊なヘッダのパケットとして実装した場合，RFend は図 10 の FLIT1 や FLIT3 からサスペンド状態に遷移してしまうため，その後の処理でソフトウェアによるヘッダの解析が必要となってしまう．

そこで，RFend がヘッダ解析を完了した段階でサスペンドするように，VPUSH では通常の PUSH パケットとまったく同一のヘッダを用い，その際ヘッダに含まれる SID を受信側で必ずミスヒットする特殊な値に設定しておくことで，RFend をヘッダ解析完了後の RVA からサスペンド状態へ遷移させる．これにより，送信側では PUSH 機構をそのまま利用して VPUSH のパケットを送出できるようになる．

RFend がステート RVA からサスペンド状態へ遷移すると，オンチッププロセッサは受信先のアドレスを計算し，RFend 内の受信領域の仮想アドレスが書かれたレジスタの内容を正しい受信先アドレスに更新する．その後の処理は通常の PUSH を受信した際の処理とまったく同一であるため，ソフトウェア処理をいったん完了し，RFend のステートを RVA に設定することで，ハードウェア処理を再開させる．すなわち，PUSH パケットの受信処理において，本来であればハードウェアで RVATLB を参照して受信領域の仮想アドレスを得る部分を，ソフトウェアでエミュレーションし，異なるアドレスを与えたことになる．

ここで，ホストの受信領域へ完全にデータを転送し終えた後にヘッドポインタを更新する処理をソフトウェアで行う必要があるため，再びソフトウェア処理に戻ることができるよう RFend にブレークポイントをセットしておかなければならない．しかし，RFend には RVA 以降 IDLE までのステートにサスペンダブルなステートがないためブレークポイントをセットすることができない．そこで，応答パケット生成モジュールの Replier にブレークポイントをセットする．Replier による応答パケット転送処理は DMA 転送が完了した直後に開始するため，ここにブレークポイントを仕掛け，ソフトウェア処理に復帰することで，ホストへの DMA 転送完了直後に確実にヘッドポインタを更新することが可能となる．

VPUSH は，以上のようにソフトウェア処理の間に部分的にハードウェアによる処理を織り交ぜることで実装されている．なお，VPUSH において送信側が一度の通信要求で送信可能なデータサイズの上限は RHiNET-2/SW が扱える最大パケット長と同じ 2048 Byte としている☆．これは，受信時に複数ホストから PUSH された転送データがインタリーブした状態でバッファに格納されてしまうことを防ぐためである．

5.3.4 VPUSH のバンド幅

VPUSH の基本的な性能として，2 ノード間での VPUSH のバンド幅を測定した．評価は例外処理の評価で用いた環境と同一の環境で行った．

図 14 に，VPUSH の送信パケットのデータサイズとバンド幅の関係を示す（VPUSH）．また，オンチッププロセッサが RFend を乗っ取った後の処理をすべてソフトウェアで処理した場合のバンド幅（VPUSH/Software）と通常の PUSH におけるバンド幅（PUSH）をあわせて示す．

VPUSH のバンド幅は転送データサイズが 2048 Byte のときに 180.3 MByte/s に達する．同データサイズの転送を完全にソフトウェアで処理した場合のバンド幅は 78.6 MByte/s であることから，ハードウェアとの協調処理を行うことでソフトウェアのみで処理するよりも大幅に高い処理能力を実現できていることが分かる．

図 15 は，RFend がデータサイズ 2,048 Byte, 512 Byte, 8 Byte の VPUSH パケットの到着を検出して処理を開始した直後からオンチッププロセッサが

☆ 単一の PUSH 要求によって送信可能なデータサイズの上限は 1 GByte である．

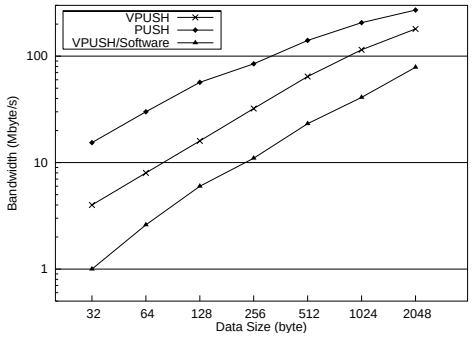
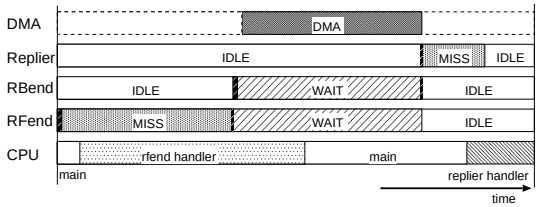
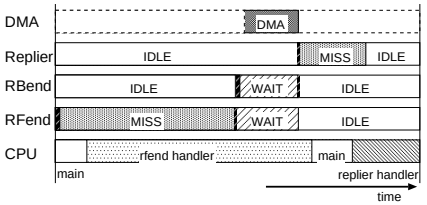


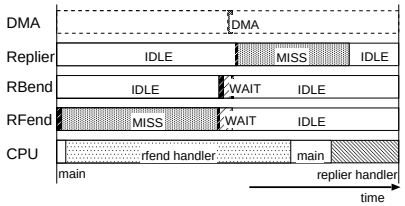
図 14 VPUSH のバンド幅
Fig. 14 Bandwidth of VPUSH.



(a) 2048 Byte 受信時



(b) 512 Byte 受信時



(c) 8 Byte 受信時

図 15 VPUSH 処理時間の内訳
Fig. 15 Breakdown of VPUSH processing.

Replier の例外処理を完了するまでの，VPUSH に関連したモジュールのステートや処理内容の変化を示している．なお，実機で各モジュールの細かな処理内容を測定することは難しいため，測定には RTL シミュレーションを用いた．

図 15 の CPU はオンチッププロセッサの処理内容を示している．main は各ハードウェアモジュールからの乗っ取り要求（割込み線）の変化をポーリングしている状態であり，rfend handler は RFend を，replier handler は Replier をそれぞれ乗っ取った際のソフトウェア処理を示している．

また，RFend，RBend，Replier について，MISS はサスペンド状態を示し，WAIT は DMA の完了待ち状態を，黒い部分はモジュールがステートを遷移しながら処理を行っている時間帯を示している．DMA は DMA 転送が開始してから完了するまでの時間帯を示している．

図 15 (a) より，2,048 Byte の VPUSH パケットを受信した際の処理時間の約 60%をオンチッププロセッサの処理が占めていることが分かる．この時間は約 6.6 μs に相当し，リソースの競合等が生じない限りパケット長によらず一定である．

図 15 (a) の DMA 転送が行われている部分に着目すると，DMA 転送が完了するよりも前にソフトウェア処理の rfend handler が完了しており，オンチッププロセッサは DMA 転送の完了を待っている Replier からの割込みを待つ，という状態になっていることが分かる．一方，8 Byte 転送時の図 15 (c) を見ると，ソフトウェアが rfend handler を実行している間に DMA 転送が完了しており，オンチッププロセッサは RFend の例外処理後，無駄なループをせずに即座に Replier の例外処理へと移っている．ここで，512 Byte 転送時の図 15 (b) を見ると，rfend hanlder と DMA 転送がほぼ同時に完了していることが分かる．このことから，512 Byte 以下の場合には，パケット処理時間がオンチッププロセッサの処理に支配され一定となるためにバンド幅がパケットサイズに比例し，512 Byte 以上の場合には，処理時間にパケット長により変化する DMA 完了待ち時間が加わってしまうためにバンド幅がパケットサイズに比例しなくなることが分かる．

5.4 乗っ取り機構によるハードウェア増加

乗っ取り機構を実装するためには，オンチッププロセッサからモジュールに対して個別にバスを設け，モジュール内にサスペンド状態となるステートを設ける必要がある．また，それ以外にもモジュール内に外部からレジスタの値を読み書きするための回路等が必要となるため，ハードウェア規模は若干増大し，遅延も加わることになる．

バスの配線に関しては，Martini の場合，乗っ取り機構を実装したことでオンチッププロセッサからの書き込みに 37 段，読み出しに 59 段分のゲート遅延が加わった．そのため，バスの途中にラッチを加え，書き込みに 2 クロック，読み出しに 3 クロック要する構造としている．

また，乗っ取り機構を追加することによるハードウェア量の増加について評価するために，Replier を通常

表 3 乗っ取り機構の有無にともなう Replier のハードウェア量の変化
Table 3 Hardware amount of Replier with/wihtout the taking over mechanism.

乗っ取り機構あり	乗っ取り機構なし
9898	9384

のものと乗っ取り機構に関する記述を除外したものとでそれぞれ論理合成を行い，ベーシックセル数の比較を行った結果を**表 3** に示す．

表 3 より，Replier において乗っ取り機構を追加した場合のハードウェア量の増加は約 5.5%程度であることが分かる．

5.4.1 乗っ取り機構の一般性

乗っ取り機構を用いた例外処理は，あらかじめ発生を想定することができるため，ハードウェア設計の段階で例外を発生する可能性のあるステートをサスペンダブルな設計としておくことができる．一方，ソフトウェアを用いた通信処理を実装する場合，処理内容によっては例外の発生がないようなステートにおいてモジュールを停止させなければならない場合があるが，Martini では例外処理を発生するステートのみがサスペンダブルに設計されているため，このようなソフトウェア処理は乗っ取り機構を用いて実装できないことになる．

これは，すべてのステートをサスペンダブルに設計した場合，周辺モジュールとのハンドシェイク制御が複雑化してしまうことと，Martini の設計段階で乗っ取り機構を用いたソフトウェア処理について十分な検討がなされていなかったことに起因する．今回実装を行った VPUSH は乗っ取り機構を利用して実装されているが，Replier の ACK パケット発行処理を行う部分はサスペンダブルな設計となっていないため，DMA 転送完了を検出するために Replier をサスペンドさせた後，Replier を元のステートに戻して ACK パケットを生成させることができず，Replier に渡された ACK 生成要求をキャンセルしてステートを IDLE に戻さざるを得ない等，いくつか制約がともなってしまっている．乗っ取り機構を利用したソフトウェア処理をより少ない制約で実現するには，ハードウェアの設計段階で，ソフトウェア処理に必要とされる機能についてある程度検討しておく必要があるだろう．

また，今回乗っ取り機構は Martini に対して実装を行ったが，乗っ取り機構は他のネットワークコントローラやストレージコントローラ等の専用ハードウェア部が処理の中心でオンチッププロセッサが補助的な処理を行うような構成のシステム LSI においては，Martini と同様の効果を得ることができるものと考えられる．

一方，マルチメディア用途等の，協調設計等によりオンチッププロセッサと専用ハードウェア部が常時同時に動作して処理を行うようなシステム LSI では，乗っ取り機構を実装することで 5.2 節で述べたような例外処理後のソフトウェア処理を軽減することは可能であると考えられる．しかし，元々設計段階でハードウェア処理とソフトウェア処理が明確に切り分けられていることから，VPUSH のようなハードウェア処理を部分的に変更した処理を行うような用途では乗っ取り機構は十分な効果を発揮できないものと思われる．

6. ま と め

システム LSI における新しいハードウェア/ソフトウェア協調処理の形態である乗っ取り機構を提案し，ネットワークインタフェース用コントローラチップである Martini 上に実装してその評価を行った．

評価の結果，乗っ取り機構を導入することで，一部のハードウェアモジュールが停止し，ソフトウェアがモジュールに代わって処理を行っている間も，他の無関係なハードウェアは並行に動作可能であることが確認され，例外の要因を取り除いた後の処理をハードウェアにより続行させることで例外処理の短縮を図ることが示された．

また，乗っ取り機構を利用してソフトウェア処理にハードウェア処理を組み込んだ例として VPUSH を実装した．VPUSH は，通常の PUSH パケットを処理するためのハードウェアを利用することで，単純にすべてをソフトウェアで処理した場合に比べて倍以上のバンド幅を実現した．

以上から，バスの配線と数%程度の回路の追加で実現できる乗っ取り機構は，ネットワークプロセッサのようなシステム LSI において有用なハードウェア/ソフトウェア協調処理の手段であると考えられる．

謝辞 本研究を行うにあたり，多大な助言をいただいた日立製作所の山本淳二氏，慶應義塾大学の西宏章氏，北村聡氏，伊豆直之氏に心より感謝いたします．

参 考 文 献

1) Xie, Y., Lin, H., Wu, Z. and Wolf, W.: CAD Techniques for Multimedia System Design, *Proc. 9th Workshop on Synthesis and System Integration of MIxed Technologies* (*SASIMI 2000*), pp.81–87 (2000).
2) Xie, Y. and Wolf,. W: Co-synthesis with custom ASICs, *Proc. 2000 conference on Asia South Pacific design automatio* (*ASP-DAC*

2000), pp.129–133 (2000).
3) Myricom, Inc.. http://www.myri.com/
4) Kudoh, T., Nishimura, S., Yamamoto, J., Nishi, H., Tatebe, O. and Amano, H.: RHiNET: A network for high performance parallel processing using locally distributed computers, *Proc. 1999 International Workshop on Innovative Architecture* (*IWIA99*), pp.69–73 (1999).
5) Tezuka, H., O'Carroll, F., Hori, A. and Ishikawa, Y.: Pin-down cache: a virtual memory management technique for zero-copy communication, *Proc. 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing* (*IPPS/SPDP 1998*), pp.308–314 (1998).
6) 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.04, pp.866–878 (2002).
7) 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, pp.10–23 (2003).
8) 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, 情報処理学会並列処理シンポジウム JSPP2002 論文集, pp.35–42 (2002).
9) Ishikawa, Y., Tezuka, H., Hori, A., Sumimoto, S., Takahashi, T., O'Carroll, F. and Harada, H.: RWC PC Cluster II and SCore Cluster System Software — High Performance Linux Cluster, *Proc. 5th Annual Linux Expo*, pp. 55–62 (1999).
10) Takahashi, T., Sumimoto, S., Hori, A., Harada, H. and Ishikawa, Y.: PM2: High Performance Communication Middleware for Heterogeneous Network Environment, *Proc. Supercomputing 2000* (*SC2000*), pp.52–53 (2000).
11) 田邊 昇, 山本淳二, 今城英樹, 上嶋利明, 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 の試作, 情報処理学会 HPC 研究会, Vol.2001, No.77, pp.99–104 (2001).
12) 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, 情報処理学会 2000 年記念並列処理シンポジウム JSPP2000 論文集, pp.173–180, (2000).
13) Watanabe, K., Otsuka, T., Tsuchiya, J., Harada, H., Yamamoto, J., Nishi, H., Kudoh, T. and Amano, H.: Performance Evaluation of RHiNET-2/NI: A Network Interface for Distributed Parallel Computing Systems, *Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid* (*CCGrid2003*), pp.318–325, (2003).
14) 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 SACSIS2003 論文集, pp.45–52, (2003).
15) Nishi, H., Nishimura, S., Harasawa, K., Kudoh, T. and Amano, H.: The Architecture and Evaluation of 3rd-generation RHiNET Switch for High-performance Parallel Computing, *IEICE Trans. Information and Systems Special Issue on Development of Advanced Computer Systems*, Vol.E-86-D, No.10, pp.1987–1995 (2003).
16) Yoshikawa, T., Hatakeyama, I., Miyoshi, K. and Kurata, K.: Optical Interconnection as an Intellectual Property of a CMOS Library, *Proc. Hot Interconnects* 9, pp.31–35 (2001).
17) 山本淳二, 渡邊幸之介, 宮脇達朗, 西 宏章, 工藤知宏, 天野英晴：PLI を用いた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グラムの協調シミュレーション, 情報処理学会研究報告, 2001-ARC-145, Nov. 2001, pp.73–78 (2001).

(平成 16 年 2 月 1 日受付)
(平成 16 年 6 月 7 日採録)

渡邊幸之介（学生会員）

平成 15 年慶應義塾大学大学院理工学研究科開放環境科学専攻前期博士課程修了．現在，同後期博士課程に在学．平成 16 年度より日本学術振興会特別研究員．PC クラスタ向けネットワークインタフェースに関する研究に従事．

大塚 智宏（学生会員）

平成 15 年慶應義塾大学大学院理工学研究科開放環境科学専攻前期博士課程修了．現在，同後期博士課程に在学．PC クラスタのネットワーク，通信ミドルウェアの研究に従事．

天野 英晴（正会員）

昭和 56 年慶應義塾大学理工学部電気工学科卒業．昭和 61 年同大学大学院理工学研究科電気工学専攻博士課程修了．現在，慶應義塾大学理工学部情報工学科教授．工学博士．計算機アーキテクチャの研究に従事．