2003－ARC－153 (8)
2003／5／8

Responsive Multithreaded Processor 用バス機構の設計と実装

一ノ瀬 信征 † 佐藤 純一 † 山崎 信行 †

本論文では，分散リアルタイム処理用プロセッサである Responsive MultiThreaded（RMT）Processor のバス機構について述べる．RMT Processor では複数スレッドの同時実行を行うためメモリアクセスによるバス利用が多くなる．また，リアルタイム処理を扱うために I/O によるバス利用も多くなり，スループットの高いバス機構が要求される．本研究ではトランザクションを request と ack に分け，バスの占有と解放を制御することによってレイテンシの隠蔽を行い，バスのスループットを改善する．スプリットトランザクションを可能にしたことにより従来の共有バスよりもスループットを高くすることができた．

Design and Implementation of Bus mechanism for Responsive Multithreaded Processor

Nobuyuki Ichinose ,† Junichi Sato † and Nobuyuki Yamasaki †

This paper describes the bus mechanism of Responsive MultiThreaded (RMT) Processor which is a processor for distributed real-time operations. In RMT Processor, in order to perform simultaneous execution of two or more threads, the bus use by memory access increases. In order to treat a real-time operation, the bus use by I/O also increases, and the high bus mechanism of a throughput is required. In this research, a transaction is divided into request and ack, a latency is concealed by controlling occupancy and release of a bus, and the throughput of a bus is improved. The throughput was able to be made higher than the conventional share bus by having made the split transaction possible.

1. はじめに

リアルタイム性のある処理とは広義にはある処理の実行結果の価値がその実行結果だけでなく，その処理時間にも依存する処理のことを示し，狭義には時間制約を守る処理のことをいう．

実際にリアルタイム処理をコンピュータで行う際には OS が処理するタスクをスケジューリングし，各タスクが時間制約を守れるようにする．タスクのスケジューリングを行うとコンテキストスイッチのオーバーヘッドがタスクの処理時間に上乗せされる．そのためコンテキストスイッチのオーバーヘッドを減らすことがリアルタイム処理を行うために必要となる．

Responsive MultiThreaded（RMT）Processor ではハードウェア上でコンテキストスイッチをサポートしていることに加え，複数スレッドから複数の命令を同時発行し，同時実行することができる．RMT Processor は多数のスレッドをハードウェアで保持し，リアルタイム処理を行うことができる．また，スレッドに優先度をもたせることで特定スレッドの優先実行を可能にしている．RMT Processor はメモリアクセスのレイテンシを別スレッドの実行で隠蔽している．このことにより，メモリアクセスによるバスの利用が複数スレッドによって行われることになるので，通常のプロセッサに比べて単位時間あたりのバス利用率が高くなる．さらに RMT Processor は system on a chip であり，I/O のインターフェースが 1 チップにまとめられているが，RMT Processor が想定しているリアルタイム処理はセンサから入力されたデータをある時間内に処理し，それにもとづいてアクチュエーター制御を行うなど，I/O の使用が多い．

本論文では RMT Processor においてバストランザクションのレイテンシを隠蔽し，I/O インターフェースとプロセッサコア間のバストランザクションの効率化を計るバス機構について述べる．

† 慶應義塾大学
Keio University

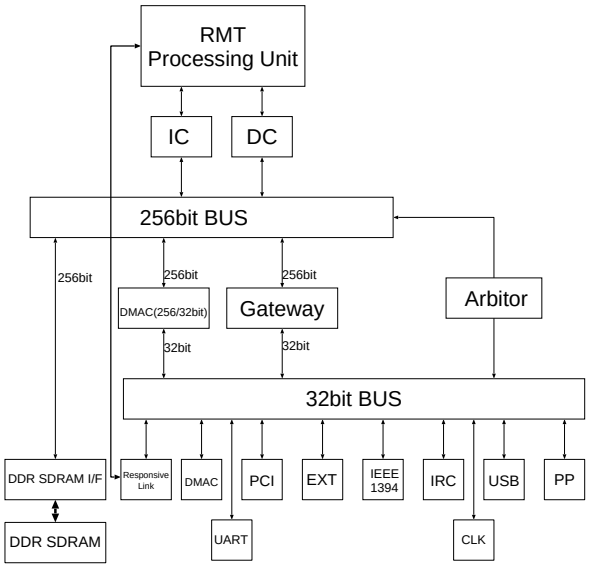

図 1　RMT Processor のバスの構成

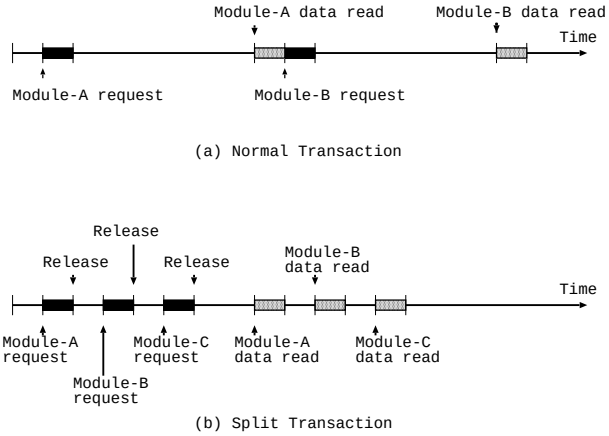

図 2　スプリットトランザクション

2. 背　　景

2.1 リアルタイムシステム

リアルタイム性を持つ処理には大きく分けて 2 種類ある．ひとつはハードリアルタイムといい，これは制限時間内に処理を完了させることが必須となっており，時間内に処理が完了しない場合には実行結果の価値がただちに 0 になる処理のことをいう．もうひとつはソフトリアルタイムといい，これは時間内に処理が完了しなくても実行結果の価値がただちに 0 になることはなく，処理の遅延に応じて実行結果の価値が減ずるものをいう．このように，リアルタイムシステムでは処理に時間的制約がついており，その制約を満たせたかどうかで処理結果の価値が左右される．

2.2 Responsive Multithreaded Processor

RMT Processor は分散リアルタイム処理をハードウェアレベルで支援するためのプロセッサであり，各種 I/O を 1 チップに集積した system on a chip である [1]．ハードウェア上で最高 40 までのスレッドを保持し，8 つのスレッドを同時に実行することが可能である．RMT Processor のバスの構成を図 1に示す．

各 I/O インターフェース（PCI，USB，IEEE1394 など）は 32bit 幅のデータ入出力がある．命令キャッシュ（IC），データキャッシュ（DC），DDR SDRAM インターフェースは 256bit 幅のデータ入出力となっている．各 I/O インターフェースを 32bit BUS に接続し，キャッシュと DDR SDRAM インターフェースを 256bit BUS に接続する．256bit BUS と 32bit BUS は図 1 の Gateway 部を介してデータ転送を行う．

RMT Processor のバス構成は通常の共有バスになっており，256bit BUS と 32bit BUS の使用権は同一となっている．そのため，バストランザクションのレイテンシがデータ転送に影響する．

2.3 スプリットトランザクション

スプリットトランザクションとは，データ転送時に相手モジュールが即座に応答できない場合に転送を中断し，別の転送を行う処理である．[2] バスの物理的な接続は単一のバスにすべてのモジュールがつながっている．処理の流れを図 2に示す．

Module A がメモリに read request を出し，メモリがアクセスされてデータを送信するまでの間，バスを占有しつづけると，バストランザクションのレイテンシの影響を受け，バスの利用効率が悪くなる（図 2（a））．これに対してスプリットトランザクションが可能な場合，Module A は request を出した後，バスを解放し，次の Module B がバス権をとれるようにする．Module B も request のみでバスを解放し，Module C がバスをとれるようにする（図 2（b））．Module A に対するデータが用意された時点で，再度 Module A がバスを占有し，データを受信する．

スプリットトランザクションにはリクエストの発生順とレスポンスの発生順が同じ in-order タイプと，レスポンスがリクエストの順を追い越す out-of-order タイプがある．

スプリットトランザクションは，転送の中断と再開の制御により，バスの制御が複雑になるがバストランザクションのレイテンシを隠蔽し，バスの利用効率を上げるためには極めて有効である．

2.3.1 RMT バスアーキテクチャの課題

RMT のバスアーキテクチャに通常の共有バスを選

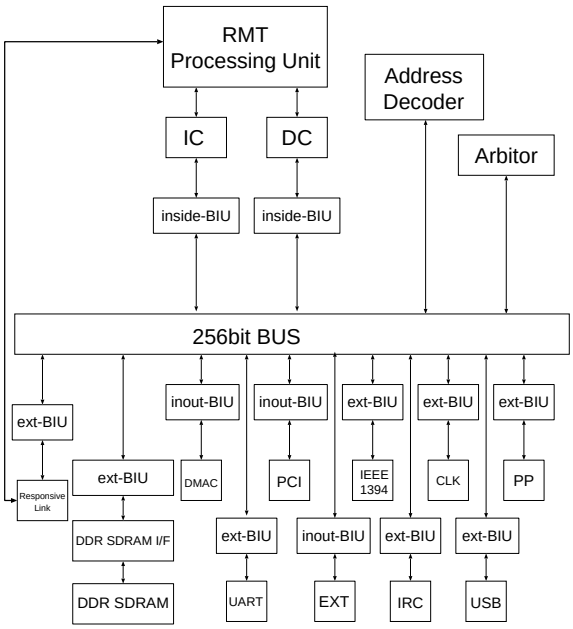

図 3 スプリットトランザクションの機能を取りつけたバス構成

択した場合，すべてのバストランザクションがシリアルに実行され，データ転送はバストランザクションのレイテンシの影響を受けるため性能が低下してしまう．そこで本研究ではバストランザクションの性能向上を達成するバス機構の設計を行う．

3. 設計と実装

3.1 設 計 方 針

本研究では，RMT Processor に適合する形でバスアーキテクチャを設計及び実装する．現段階の RMT Processor のバスアーキテクチャは図 1の構成となっている．このバス構成では，通常の共有バスと同様のアービトレーションをするため，多数のモジュールが同時にバス権を要求すると各トランザクションは完全にシリアルに行われてしまい，データ転送はバストランザクションのレイテンシの影響を直接受けてしまう．そこでレイテンシの隠蔽を行うためにスプリットトランザクションを行う機構を設計する．

3.2 設計するバスアーキテクチャ

3.2.1 バスアーキテクチャの概略

設計するバス構成は図 3 になる．以下ではバストランザクションを行う各 I/O インターフェースや S-DRAM，DC，IC をまとめてモジュールと呼称する．

既存のモジュールの設計を変更せずにスプリットトランザクションを行うために，図 3のように各種モジュールとバスの間に 1 つのモジュール（以後これを BIU と呼称）を配置し，256bit BUS で BIU 同士を接続する．スプリットトランザクションは BIU 間で 256bit BUS を介して行われ，各モジュールは通常の共有バスにおけるトランザクションの通信プロトコルでデータ転送を行い，スプリットトランザクションにおけるバスの占有と解放の制御は BIU が行う．各モジュールはそれぞれ動作が異なる．各動作に対応するために 3 種類（inside-BIU，ext-BIU，inout-BIU）の BIU を用意する．

表 1 read，write と request，ack の組み合わせ

	read 時	write 時
request	読み出し先アドレス	書き込みアドレスと 書き込みデータ
ack	読み出しデータと 読み出し完了通知	書き込み完了通知

- **inside-BIU**
  IC や DC などモジュール自体がトランザクションを行うが，他のモジュールからトランザクション要求を受けることのないモジュールに対応する BIU.
- **ext-BIU**
  SDRAM インターフェースや I/O インタフェースなどモジュール自体はトランザクションを行わないが，他のモジュールからトランザクション要求を受けるモジュールに対応する BIU.
- **inout-BIU**
  DMAC，PCI などモジュール自体がトランザクションを行い，他のモジュールからもトランザクション要求を受けるモジュールに対応する BIU.

各 BIU の違いは内部にもつバッファの種類であり，機能的に大差はないので本論文では必要のないかぎり同一のものとして扱う．

3.2.2 トランザクションの分割

バストランザクションは request トランザクションと ack トランザクションの 2 つに分割される．基本的なトランザクションの流れは次のようになる．BIU がアクセス対象の BIU に request パケットを送り，一旦バスを解放する．解放されたバスは他の BIU が使用可能になる．そして返答の準備が完了したら ack パケットを request パケットを出した BIU へ送り返してトランザクションを終了する．

read トランザクション時と write トランザクション時でパケットの中の情報が異なる．パケット内部の情報を種類別に表 1に示す．

パケットフォーマットの詳細は 3.3 節で示す．本設計では高い性能向上を得るために in-order ではな

く，out-of-order のスプリットトランザクションを採用する．

3.2.3 アービトレーション

各 BIU が同時にパケットを送信した場合，パケットの衝突が発生する．これを避けるためにバスの調停を Arbitor で行う．アービトレーションの手順としてはバスを使用する BIU が Arbitor にバス権を要求し，Arbitor がひとつの BIU を選択してバス権を与える．バストランザクション全体のスループット向上のためにアービトレーションの方針はバス権を与えるたびに優先順位を順に回していくラウンドロビン方式を使用する．

3.2.4 パケット ID の配布

本設計では out-of-order タイプのスプリットトランザクションを採用している．そのため ack が request の順を追い越す可能性があり，バストランザクションにおける coherency と consistency を維持できなくなる．よって各 BIU で out-of-order に返ってきた ack を一時的にバッファに格納し，in-order に並び替える．in-order に並び替えるためにはどの順番で request パケットが出されたかという情報が必要である．そのためバスは投入されたパケットにパケット ID を付加する．パケット ID はバス全体に一意なものとする．

3.2.5 アドレスデコード

共有バスであるためにバスを流れるパケットはすべての BIU に届く．よって BIU がパケットを送信すると，バスではそのパケットのアドレスをデコードし，目的 BIU へデータ取り込み信号を出す．取り込み信号を受け取った BIU がパケットを受け取る．

アドレスをデコードして目的 BIU を割り出す処理は request パケットで行い，ack パケットではアドレスデコードをしない．アドレスがふられていないが，ack パケットの送信先になる DC,IC が存在し，アドレスデコードだけでは送信先を決定できないため，ack パケットの送信先はその ack パケットに対応する request パケットを出したモジュールとしている．

3.2.6 BIU の役割

BIU の役割は各モジュールの設計の変更をせずにスプリットトランザクションを実現することである．それを実現するために以下の機能を設計した．

(1) 対応するモジュールからトランザクション要求を受け，request パケットを生成する．このとき BIU とモジュールの間では共有バスのプロトコルでデータ転送をする．

(2) 他の BIU から受け取った request packet をもとに対応するモジュールと共有バスでのプロトコルでデータ転送を行い，ack パケットを生成する．

(3) パケット生成後，バス権を Arbitor に要求し，バス権取得後にパケットをバスに送出する．

(4) out-of-order に届いた ack パケットを in-order に並び替えるためにバッファにデータを格納する．

(5) 受け取った ack パケットがバッファの先頭に位置したときに，その ack パケットの内容にもとづいて BIU とモジュール間のトランザクションの続きを行い，トランザクションを終了させる．

inside-BIU は（1）,（3）,（4）,（5）の処理を行い，ext-BIU は（2）,（3）の処理のみを行う．inout-BIU はすべての処理を行う．

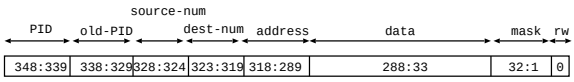

図 4 パケットフォーマット

3.3 パケットフォーマット

BIU 間でスプリットトランザクションを行う際に使用するパケットのフォーマットを説明する．パケットのフォーマットを図 4に示す．パケットの各ブロックについて示す．

- **PID**
  パケット ID．そのパケットがバスを通過した際にバスに付加される一意の番号である．PID にもとづいて BIU ではパケットを in-order に並び替える．使用する ID に 0 は使用せず，1 から 1023 までを使用する．
- **org-PID**
  パケットが ack パケットであった場合，どの request パケットに対する ack パケットであるかを示す必要がある．そのため，対応する request パケットのパケット ID を org-PID として情報を保持する．org-PID が 0 であるときはそのパケットは request パケットであることを示し，それ以外の番号の場合は ack パケットであることを示す．
- **source-num**
  パケットがどの BIU から送出されたかを示す．source-num は request パケットを受け取った BIU がその request パケットに対する ack を返すときに返答先の BIU を示す．
- **dest-num**
  パケットがどの BIU へ送信されるかを示す．
- **address**
  BIU が対応するモジュールから受信したアドレス．

表 2　各モデルのレイテンシ

モデル	トランザクション	レイテンシ
I/O モデル	write	20 サイクル
	read	15 サイクル
SDRAM モデル	write	3 サイクル
	read	10 サイクル

- **data**
  実際のデータ．RMT Processor では SDRAM と DC，IC は 256bit（8 ワード）のアクセスが可能であるために 256bit 幅にしている．
- **mask**
  マスク．1byte 単位でマスクするので 256bit 分のマスク信号として 32bit 用意する．
- **rw**
  そのパケットが read トランザクションか，write トランザクションかを示す．

4. 評　　価

4.1 評 価 方 法

本方式ではバストランザクションを並行に行うことでレイテンシを隠蔽し，全体のスループットを向上させることを目的としている．よって，トランザクションの並行実行による性能向上を調べるためにトランザクションが 1 つのモジュールに集中しない多対多のトランザクション時のスループットの計測を行う．また，1 つのモジュールにアクセスが集中した際の性能を調べるために，1 対多のトランザクション時のスループットの計測も行う．この 2 種類の評価を本方式ともとの RMT Processor のバス構成（通常の共有バス）でそれぞれ行い，性能比較をする．評価は NC-Verilog の RTL シミュレーションによって行った．スループットの計測にあたってはバスに接続するモジュールのモデルを作成し，モデル間でトランザクションを行うことにする．各モデルのレイテンシ（as 信号が入力されてから ready 信号が返ってくるまでの間）を表 2に示す．

4.2 多対多のトランザクション時

3 種類のトランザクションを仮定する．

- **type1**
  DMAC0 が I/O モデル A から 32bit データを read し，そのデータを I/O モデル B に write する．これを連続して 128 回行う．
- **type2**
  DMAC1 が I/O モデル C から 32bit データを read し，そのデータを I/O モデル D に write する．これを連続して 128 回行う．
- **type3**
  DMAC2 が SDRAM から 32bit データを read し，そのデータを I/O モデル E に write する．これを連続して 128 回行う．

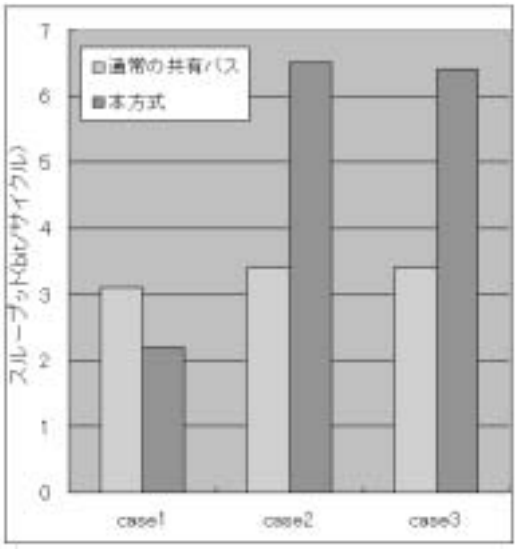

図 5　多対多のトランザクション時におけるスループット

これらのトランザクションは使用するモジュールが重ならないトランザクションである．以下の 3 つのケースにおいてスループットを計測する．

- **case1（同時実行数 1）**：type1 のみを実行
- **case2（同時実行数 3）**：type1，type2，type3 すべてを同時に実行
- **case3（同時実行数 12）**：type1，type2，type3 のトランザクションを DMAC に 4 回分ずつ設定し，合計 12 トランザクションを同時に実行

RMT Processor では 3 つの DMAC が存在し，DMAC1 つにつき 4 つのトランザクションを設定できるので最大同時実行トランザクション数は 12 になる．上述の 3 つのケースのスループットを図 5 に示す．

図5をみると共有バスに比べて本方式ではスループットが 3 つのトランザクションを同時実行した case2 において 1.91 倍，12 個のトランザクションを同時実行した case3 において 1.88 倍向上した．これは各トランザクションのレイテンシの最中に別のトランザクションを行うことによりトランザクションの並行処理が可能であることを示している．単一のトランザクションを実行した case1 ではスループットが 0.71 倍に低下している．これは request パケットの生成と ack パケットの生成がオーバーヘッドとして付加されるため並行に実行するトランザクションがない単一のトランザクションでは性能低下を招くことを示している．

4.3 1 対多のトランザクション時

6 種類のトランザクションを仮定する．

- **type1**
  DMAC0が SDRAMから 32bitデータを readし，そのデータを I/O モデル A に write する．これを連続して 128 回行う．
- **type2**
  DMAC1が SDRAMから 32bitデータを readし，そのデータを I/O モデル B に write する．これを連続して 128 回行う．
- **type3**
  DMAC2が SDRAMから 32bitデータを readし，そのデータを I/O モデル C に write する．これを連続して 128 回行う．
- **type4**
  DMAC0が SDRAMから 32bitデータを readし，そのデータを I/O モデル D に write する．これを連続して 128 回行う．
- **type5**
  DMAC1が SDRAMから 32bitデータを readし，そのデータを I/O モデル E に write する．これを連続して 128 回行う．
- **type6**
  DMAC2が SDRAMから 32bitデータを readし，そのデータを I/O モデル F に write する．これを連続して 128 回行う．

これらはいずれも SDRAM にアクセスするトランザクションである．以下の 3 つのケースにおいてスループットを計測する．

- **case1（同時実行数 1）**：type1 のみを実行
- **case2（同時実行数 3）**：type1，type2，type3 すべてを同時に実行
- **case3（同時実行数 12）**：type1，type2，type3，type4，type5，type6 をそれぞれ DMAC に 2 つずつ設定し，合計 12 トランザクションを同時に実行

これら 3 つのケースのスループットを図 6 に示す．図 6 をみると 3 つのトランザクションが 1 つのモジュールに集中した case2 においてスループットが 1.18 倍向上し，12 個のトランザクションが 1 つのモジュールに集中した case3 ではスループットが 1.43 倍向上した．

5. 結　　論

リアルタイム処理用マルチスレッドプロセッサである RMT Processor に，スプリットトランザクションを行うバスアーキテクチャを設計及び実装した．

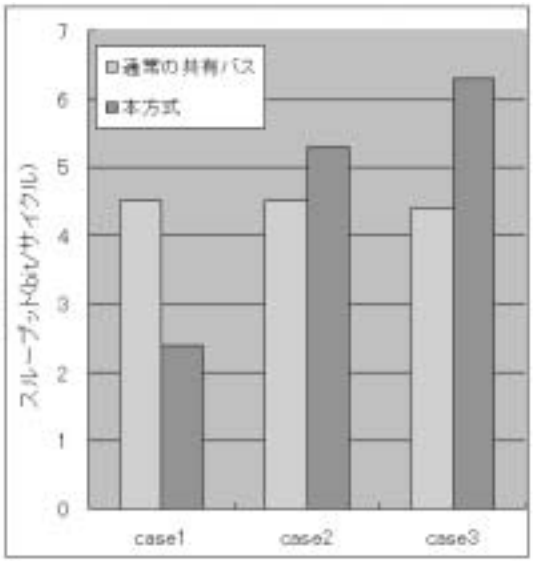

図 6　1 対多のトランザクション時におけるスループット

トランザクションを request と ack の 2 つに分解し，トランザクションの ack 待ちのときにバスを解放することで別のトランザクションを実行可能にする．複数のトランザクションを並行に行うことにより，トランザクションのレイテンシを隠蔽し，スループットを高めることが可能であることを示した．また，1 つのモジュールにアクセスが集中した場合も共有バスに比べてスループットが向上した．

6. 今後の課題

本論文では RMT Processor のバス機構としてスプリットトランザクションを設計及び実装した．しかし，スループットの向上を図っただけであり，リアルタイム処理を扱う機構を組み込んでいるわけではない．本研究の設計を土台としてリアルタイム処理を扱うバス機構を設計する予定である．

参 考 文 献

1) 山崎信行, 堀俊夫: 分散リアルタイムネットワーク用プロセッサとその応用, 情報処理, Vol. 44, No. 1, pp. 6–13 (2003).
2) Jhang, S. T. and Jhon, C. S.: A new write-invalidate snooping cache coherence protocol for split transaction bus-based multiprocessor systems, *TENCON '93. Proceeding. Computer, Communication, Control and Power Engineering.1993 IEEE Region 10 Conference on Issue*, Vol. 1, pp. 229–232 (1993).