(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5283810号
(P5283810)
(45)発行日　平成25年9月4日(2013.9.4)　(24)登録日　平成25年6月7日(2013.6.7)

(51)Int.Cl.　　F I
G06F 9/38 (2006.01)　　G06F 9/38 370C
G06F 9/30 (2006.01)　　G06F 9/30 350A

請求項の数 17　(全 34 頁)

(21)出願番号　特願2001-503043(P2001-503043)
(86)(22)出願日　平成12年6月15日(2000.6.15)
(65)公表番号　特表2003-502728(P2003-502728A)
(43)公表日　平成15年1月21日(2003.1.21)
(86)国際出願番号　PCT/GB2000/002331
(87)国際公開番号　WO2000/077627
(87)国際公開日　平成12年12月21日(2000.12.21)
審査請求日　平成19年6月4日(2007.6.4)
審判番号　不服2011-22822(P2011-22822/J1)
審判請求日　平成23年10月24日(2011.10.24)
(31)優先権主張番号　99304659.8
(32)優先日　平成11年6月15日(1999.6.15)
(33)優先権主張国　欧州特許庁(EP)

(73)特許権者　511076424
ヒューレット－パッカード　デベロップメント　カンパニー　エル．ピー．
Ｈｅｗｌｅｔｔ－Ｐａｃｋａｒｄ　Ｄｅｖｅｌｏｐｍｅｎｔ　Ｃｏｍｐａｎｙ，　Ｌ．Ｐ．
アメリカ合衆国　テキサス州　７７０７０　ヒューストン　コンパック　センタ　ドライブ　ウエスト　１１４４５
(74)代理人　110000039
特許業務法人アイ・ピー・エス
(72)発明者　オルギエッティ・アンドレア
イギリス、ビーエス１６、１エーディー、ブリストル、バークレイズ・ヒル　２０

最終頁に続く

(54)【発明の名称】プロセッサおよびコプロセッサを含むコンピュータ・システム

(57)【特許請求の範囲】
【請求項１】
第１のプロセッサと、
前記第１のプロセッサへのコプロセッサとして使用するための第２のプロセッサと、
メモリと、
バースト命令に従ってデータ・バーストにおいて前記メモリに書き込みまたは読み取るデータを入れるための、少なくとも１つのデータ・バッファと、
前記バースト命令を実行するためのバースト・コントローラと、
前記バースト・コントローラによる実行のためにバースト命令を順番に提供するためのバースト命令エレメントと、
コプロセッサ命令およびバースト命令の実行を、前記コプロセッサ命令およびバースト命令が実行するデータの可用性により、同期化するための同期化機構と
を含み、
バースト命令が前記第１のプロセッサによって前記バースト命令エレメントへ提供され、前記バースト・コントローラによって実行されたバースト命令に従って前記少なくとも１つのデータ・バッファを通じて、データが、前記第２のプロセッサへの入力データとして前記メモリから読み取られ、前記第２のプロセッサからの出力データとして前記メモリへ書き込まれ、
前記同期化機構は、
特定のバースト命令の実行により増分され、特定のコプロセッサ命令の実行により減分

される第１のカウンタを少なくとも含み、
前記第１のカウンタを、第１の低しきい値を越えてさらに減分することができないとき、前記第２のプロセッサの関連付けられた実行のためのコプロセッサ命令がストールあるいは防止され、
前記第１のカウンタを、第１の高しきい値を越えてさらに増分することができないとき、前記少なくとも１つのバッファから前記メモリへのデータの関連付けられた格納のためのバースト命令がストールあるいは防止される
コンピュータ・システム。
【請求項２】
前記第２のプロセッサの実行を順番に制御するためのコプロセッサ命令を提供するためのコプロセッサ命令エレメントをさらに含み、前記コプロセッサ命令が前記第１のプロセッサによって提供される、
請求項１に記載のコンピュータ・システム。
【請求項３】
コプロセッサ・コントローラをさらに含み、
前記コプロセッサ・コントローラがコプロセッサ命令を前記コプロセッサ命令エレメントから受信し、前記第２のプロセッサの実行を、受信されたコプロセッサ命令に従って制御し、前記コプロセッサと前記少なくとも１つのデータ・バッファの間の通信を制御する、
請求項２に記載のコンピュータ・システム。
【請求項４】
前記同期化機構が、前記少なくとも１つのデータ・バッファにまだロードされていないデータにおける前記第２のプロセッサの実行を必要とするコプロセッサ命令の実行をブロックするように適合され、前記少なくとも１つのデータ・バッファから前記メモリへのデータの格納のためのバースト命令の実行を、このようなデータが前記第２のプロセッサによって前記少なくとも１つのデータ・バッファへ提供されていない場合にブロックするように適合される、
請求項１に記載のコンピュータ・システム。
【請求項５】
前記同期化機構は、
特定のコプロセッサ命令の実行により増分され、特定のバースト命令の実行により減分される第２のカウンタをさらに含み、
前記第２のカウンタを、第２の低しきい値を越えてさらに減分することができないとき、前記少なくとも１つのバッファから前記メモリへのデータの関連付けられた格納のためのバースト命令がストールあるいは防止される、
請求項１に記載のコンピュータ・システム。
【請求項６】
前記第２のカウンタを、第２の高しきい値を越えてさらに増分することができないとき、前記第２のプロセッサの関連付けられた実行のためのコプロセッサ命令がストールあるいは防止される、
請求項５に記載のコンピュータ・システム。
【請求項７】
前記バースト命令エレメントは、命令キューである、
請求項１に記載のコンピュータ・システム。
【請求項８】
前記バースト命令エレメントは、さらに加えられたプロセッサである、
請求項１に記載のコンピュータ・システム。
【請求項９】
前記バースト命令エレメントは、プログラム可能な状態機械である、
請求項１に記載のコンピュータ・システム。

【請求項１０】
前記第１のプロセッサは、コンピュータ装置の中央処理装置である、
請求項１に記載のコンピュータ・システム。
【請求項１１】
コンピュータ・システムを動作する方法であって、
第１のプロセッサ、および前記第１のプロセッサへのコプロセッサとして動作する第２のプロセッサによる実行のためのコードを提供することと、
前記第２のプロセッサによって実行されるタスクを提供することとしての、前記コードの一部の識別することと、
前記タスクを提供するコードを、コプロセッサ・コントローラによる実行のためのコプロセッサ命令で置換すること
を含み、
前記コプロセッサ命令は、
前記第２のプロセッサによる前記タスクの実行を制御するように決定され、
前記コードおよび前記タスクから、少なくとも１つのデータ・バッファにより、前記第２のプロセッサによるアクセスのためにデータ・バーストにおいてメイン・メモリからデータを読み取り、そこへ書き込むことができるようにするためのバースト命令を決定することと、
前記少なくとも１つのデータ・バッファと前記メイン・メモリの間でデータの転送を制御するバースト・コントローラによるバースト命令の実行と共に、前記コプロセッサ上で前記タスクを実行することと
を含み、
前記タスクの実行において、コプロセッサ命令の実行とバースト命令の実行の間の同期化が、同期化機構によって達成され、
前記同期化機構は、
特定のバースト命令の実行により増分され、特定のコプロセッサ命令の実行により減分される第１のカウンタを少なくとも含み、
前記第１のカウンタを、第１の低しきい値を越えてさらに減分することができないとき、前記第２のプロセッサの関連付けられた実行のためのコプロセッサ命令がストールあるいは防止され、
前記第１のカウンタを、第１の高しきい値を越えてさらに増分することができないとき、前記少なくとも１つのバッファから前記メモリへのデータの関連付けられた格納のためのバースト命令がストールあるいは防止される
方法。
【請求項１２】
バースト命令を決定する前記ステップが、前記バースト命令を、前記第１のプロセッサによって実行される前記コードの一部内に含めることをさらに含む、
請求項１１に記載の方法。
【請求項１３】
バースト命令を決定する前記ステップが、前記コードから前記第２のプロセッサによってアクセスされるメモリ・アドレスを決定すること、および少なくとも１つのデータ・バッファにより、前記第２のプロセッサによるアクセスのためにデータ・バーストにおいてメイン・メモリからデータを読み取り、そこへ書き込むことができるように、前記第２のプロセッサによって行われるメモリ・アクセスを編成することをさらに含む、
請求項１１に記載の方法。
【請求項１４】
前記同期化機構が、前記第１の命令の正しい実行のために完了が必要である第２の命令が完了するまで、前記第１の命令をブロックすることを含む、
請求項１１に記載の方法。
【請求項１５】

前記コプロセッサ命令エレメントは、命令キューである、
請求項２に記載のコンピュータ・システム。
【請求項１６】
前記コプロセッサ命令エレメントは、さらに加えられたプロセッサである、
請求項２に記載のコンピュータ・システム。
【請求項１７】
前記コプロセッサ命令エレメントは、プログラム可能な状態機械である、
請求項２に記載のコンピュータ・システム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、メイン・プロセッサおよびコプロセッサを含むコンピュータ・アーキテクチャに関し、詳細にはこのようなアーキテクチャにおけるコプロセッサによるメモリ・リソースの使用に関する。
【背景技術】
【０００２】
マイクロプロセッサをベースとするコンピュータ・システムは、典型的には、ＣＰＵのような汎用マイクロプロセッサに基づいている。このようなマイクロプロセッサは、幅広い範囲の計算タスクを処理するように十分に適合されるが、これらは必然的にすべてのタスクに合わせて最適化されるのではない。タスクが計算に集中する場合（媒体処理など）、ＣＰＵはしばしば実行に困難を生じる。
【０００３】
この問題への標準の手法の１つは、個々の計算的に困難なタスクを処理するように特に適合されたコプロセッサを使用することである。このようなコプロセッサは、ＡＳＩＣ（特定用途向けＩＣ）を使用して構築することができる。これらは特定の計算タスクのために構築され、したがってこのようなタスクに合わせて最適化することができる。しかし、これらは使用において柔軟性がなく（これらが特定のタスクのみのために設計されるので）、一般に製造に時間がかかる。解決策としては、ＦＰＧＡ（フィールド・プログラム可能ゲート・アレイ）など、所与の計算タスクに特に適合された構成でプログラムすることができる、柔軟性のあるハードウェアの構築がある。このような構造が構成可能であるだけでなく、再構成可能である場合は、さらなる柔軟性が達成される。このような再構成可能な構造の一例がＣＨＥＳＳアレイであり、これは、国際特許出願第ＧＢ９８／００２６２号、国際特許出願第ＧＢ９８／００２４８号、１９９８年１２月１１日出願の米国特許出願第０９／２０９，５４２号、およびその欧州の相当する欧州特許出願第９８３０９６００．９号において論じられている。
【発明が解決しようとする課題】
【０００４】
このようなコプロセッサの使用は、このような計算の効率を著しく改善することができるが、従来のアーキテクチャの構成がコプロセッサの有効性を抑制する可能性がある。計算をなおより有効にコプロセッサに移すことができる装置を達成することが、特にこれらの計算が大量のデータを処理することを含む場合、望ましい。
【課題を解決するための手段】
【０００５】
したがって、第１のプロセッサ、第１のプロセッサへのコプロセッサとして使用するための第２のプロセッサ、メモリ、バースト命令に従ってデータ・バーストにおいてメモリへ書き込まれるかあるいはそこから読み取られるデータをバッファに入れるための少なくとも１つのデータ・バッファ、バースト命令を実行するためのバースト・コントローラ、バースト・コントローラによる実行のためにバースト命令を順番に提供するためのバースト命令エレメントを含み、それによってバースト命令が第１のプロセッサからバースト命令エレメントへ提供され、バースト・コントローラによって実行されたバースト命令に従

って少なくとも１つのデータ・バッファを通じて、データが第２のプロセッサによってメモリから読み取られ、そこへ書き込まれるコンピュータ・システムが提供される。

【０００６】

この装置は特に、コプロセッサが大きいブロックのデータで作業する場合、特にこのようなブロックのメモリ・アドレスが定期的に変わる場合に有利である。この装置により、このようなブロックを、メイン・プロセッサ（それらを使用するにあまり適していないシステム・コンポーネント・エレメントである）の関与を最小にして、有効にメイン・メモリに出し入れすることができる。

【０００７】

コプロセッサがデータ・バッファと類似の方法で制御される場合、特に効率的な構造を達成することができる。これは、第２のプロセッサの実行を制御するためのコプロセッサ命令を順番に提供するためのコプロセッサ命令エレメントにより行うことができる（コプロセッサ命令は初めに第１のプロセッサによって提供される）。コプロセッサ・コントローラがコプロセッサ命令をコプロセッサ命令エレメントから受信し、それに従って第２のプロセッサの実行を制御するのがよい。このコプロセッサ・コントローラは、コプロセッサと少なくとも１つのデータ・バッファの間の通信を制御することができ、たとえば、バスがコプロセッサ・コントローラとデータバッファの間に存在する場合、コプロセッサ・コントローラが、第２のプロセッサからバスに別々のデータ・ストリームの出入りのアクセスを制御することができる。

【０００８】

コプロセッサおよびバースト命令が実行するデータの可用性により、コプロセッサとバースト命令の実行を同期化するための同期化機構がある場合、特定の利点を得ることができる。これは、コプロセッサがコプロセッサ命令に基づいて実行する場合、特に十分に実施される。有効な手法は、データ・バッファにまだロードされていないデータに対し第２のプロセッサの実行を必要とするコプロセッサ命令の実行をブロックし、データが第２のプロセッサによってデータ・バッファへ提供されていない場合、データ・バッファからメモリへのデータの格納のためのバースト命令の実行をブロックするように、同期化機構を適合させることである。同期化機構を実行するための特に有効な方法は、カウンタを使用することであり、このカウンタは、適切なバーストおよびコプロセッサ命令を通じて増分あるいは減分することができ、特定の命令をさらに減分できない場合はブロックする。

【０００９】

さらなる態様では、本発明はコンピュータ・システムを動作する方法を提供し、これは、第１のプロセッサによる実行のためのコードを提供すること、第１のプロセッサへのコプロセッサとして動作する第２のプロセッサによって実行されるタスクをコードから抽出すること、コードおよびタスクから、少なくとも１つのデータ・バッファにより、第２のプロセッサによるアクセスのためにデータ・バーストにおいてメイン・メモリからデータを読み取り、そこへ書き込むことができるようにするためのバースト命令を決定すること、および少なくとも１つのデータ・バッファとメイン・メモリの間でデータの転送を制御するバースト・コントローラによるバースト命令の実行と共に、コプロセッサ上のタスクの実行を含む。

【００１０】

コードからのタスクの抽出に続いて、コプロセッサ・コントローラによる実行のためのコプロセッサ命令が、第２のプロセッサによるタスクの実行を制御するために決定されるのがよい。

【００１１】

タスクの実行の場合、コプロセッサ命令の実行とバースト命令の実行の間の同期化が同期化機構によって達成されるのがよい。この同期化機構は、第１の命令の正しい実行のために完了が必要である第２の命令が完了するまで、第１の命令をブロックすることを、有効に含むことができる。この機構は、適切なバーストまたはコプロセッサ命令を通じて増分あるいは減分することができるカウンタを使用することができる。

【００１２】
本発明の特定の実施形態を添付の図面を参照して、以下に説明する。
【発明を実施するための形態】
【００１３】
図１は、本発明の第１の実施形態によるシステムの基本エレメントを示す。本質的に、このシステムはプロセッサ１およびコプロセッサ２を含み、これらは、最大の計算効率のために、計算をプロセッサ１とコプロセッサ２の間で区分することができるよう構成されている。プロセッサ１は本質的にいかなる汎用プロセッサ（たとえば、ｉ９６０）でもよく、コプロセッサ２は本質的に、著しくより高い有効性により計算の一部を処理することができるいかなるコプロセッサでもよい。ここで記載された特定のシステムでは、本質的に計算全体がプロセッサ１によるよりもコプロセッサ２によって処理されるが、本発明はこの特定の構成に限定されるものではない。
【００１４】
特に記載されたシステムでは、コプロセッサ２が再構成可能ＦＰＧＡの形式であり、これについては以下でさらに述べられるが、コプロセッサ２の他の形式、たとえば、ＡＳＩＣＳ、ＤＳＰなどを代りに使用することができる（対応する修正を計算モデルに行うことが必要）。プロセッサ１およびコプロセッサ２は共にＤＲＡＭメイン・メモリ３へのアクセスを有するが、プロセッサ１はより高速なアクセス・メモリ４のキャッシュ、典型的にはＳＲＡＭへのアクセスも有する。ＤＲＡＭ３への効率的なアクセスが、情報の「バースト」の効率的なローディングおよび格納のためにＤＲＡＭと通信するように適合された「バースト・バッファ」メモリ５によって提供され、バースト・バッファについては以下でさらに記載される。バースト・バッファ５への命令が、バースト命令待ち行列６を通じて提供され、バースト・バッファ５がバースト・バッファ・コントローラ７の制御下で動作する。バースト・バッファのアーキテクチャは、以下に述べられる理由のため、コプロセッサ２に関連付けられたアーキテクチャにおいて、ミラーリングされる。コプロセッサ２への命令がコプロセッサ命令待ち行列８において提供され、コプロセッサがコプロセッサ・コントローラ９の制御下で動作する。バースト・バッファおよびコプロセッサの動作、およびそれらの関連付けられた命令待ち行列の同期化が、プロセッサ１自体による一般の方法ではなく、特定の機構によって達成される。この実施形態では、この機構がロード／実行セマフォ(semaphore)１０および実行／格納セマフォ１１を含み、これらは以下に記載される方法で動作する（他のこのような同期化機構が可能であり、これらについても以下に述べられる）。
【００１５】
システム・アーキテクチャにおけるエレメントの説明
このシステムの個々のエレメントが、以下でより詳細に論じられる。プロセッサ１は一般に計算を制御するが、計算自体におけるステップのいくつか（または、記載された実施形態では、すべて）がコプロセッサ２において実行されるような方法で制御する。プロセッサ１が、バースト命令待ち行列６を通じて特定のタスクのための命令、すなわち、バースト・バッファ・コントローラ７の構成、およびバースト・バッファ・メモリ５とメイン・メモリ３の間のデータの転送のための命令を提供する。さらに、コプロセッサ命令待ち行列８を通じて、プロセッサ１はさらなるタスクのための命令、すなわち、コプロセッサ・コントローラ９の構成、およびコプロセッサ２上の計算の開始のための命令を提供する。コプロセッサ２上で実行されるこの計算は、バースト・バッファ・メモリ５を通じてデータにアクセスする。
【００１６】
コプロセッサ命令待ち行列８の使用が有効にプロセッサ１をプロセッサ２の動作から減結合し、バースト命令待ち行列６の使用が有効にプロセッサ１をバースト・バッファ５から減結合する。この構成の特定の詳細は、以下でより詳細に論じられる。この減結合については、本発明のこの実施形態のための計算モデルに関連して、以下でさらに論じられる。

【００１７】
コプロセッサ２が、実際の計算のいくつかまたはすべてを実行する。特に適したコプロセッサはＣＨＥＳＳ　ＦＰＧＡ構造であり、これは、国際特許出願第ＧＢ９８／００２６２号、国際特許出願第ＧＢ９８／００２４８号、１９９８年１２月１１日出願の米国特許出願第０９／２０９，５４２号、およびその欧州の相当する欧州特許出願第９８３０９６００．９号に記載されており、それらの出願の内容が参照により本明細書に組み込まれる。このコプロセッサは再構成可能であり、４ビットＡＬＵのチェッカーボード・アレイおよびスイッチング構造を含み、それにより、コプロセッサが構成可能であり、１つの４ビットＡＬＵからの出力を使用して別のＡＬＵへ命令することができる。ＣＨＥＳＳアーキテクチャは特に、パイプライン計算に有効であり、ここでは入力および出力データ・ストリームと対話するように有効に適合される。コプロセッサ・コントローラ９（この動作が以下でさらに論じられる）が、高レベルの制御命令（計算の詳細に関する命令ではなく、コプロセッサ２の制御全体のための命令、たとえば「ｎサイクル実行する」）を、コプロセッサ命令待ち行列８から受信する。ＣＨＥＳＳコプロセッサ２はコプロセッサ・コントローラ９の制御下で動作し、バッファ・バースト５との対話を通じてデータを受信し、格納する。したがって、ＣＨＥＳＳコプロセッサ２は入力ストリーム上で動作して、出力ストリームを生成する。これは、ＣＨＥＳＳコプロセッサの動作が非常に予測可能なので、効率的なプロセスにすることができる。このモデルに従った計算の詳細な動作が、後で論じられる。

【００１８】
プロセッサ１が、従来の方法によるＳＲＡＭにおける高速アクセス・メモリ・キャッシュ４へのアクセスを有するが、メイン・メモリはＤＲＡＭ３として提供される。ＤＲＡＭへの有効なアクセスが、バースト・バッファ５によって提供される。バースト・バッファは、欧州特許出願第９７３０９５１４．４号、および１９９８年１月６日出願の対応する米国特許出願第０９／３，５２６号において記載されており、それらの出願が参照により本明細書に組み込まれる。バースト・バッファ・アーキテクチャについては本明細書で簡単に記載されるが、このアーキテクチャの十分な詳細については、先の出願を参照していただきたい。

【００１９】
この実施形態で使用されたバースト・バッファ・アーキテクチャのバージョンのエレメント（前記の出願において論じられるように、変形も可能）が、図２および図３に示される。バースト・バッファ構成エレメントがプロセッサ１と通信する接続１２が提供される。メモリ・バス１６が、メイン・メモリ３（図２においては図示せず）への接続を提供する。このメモリ・バスはキャッシュ４と共有することができ、この場合、メモリ・データパス・アービタ５８が、キャッシュ４との通信も可能にする。

【００２０】
この装置におけるバースト・バッファの全体の役割は、計算をコプロセッサ２上で実行できるようにすることであり、このコプロセッサ２とメイン・メモリ３の間のデータ転送を、両方が各システム構成エレメントの効率を最大にすると同時に全体のシステム効率を最大にする方法で行うことを含む。これは、以下のいくつかの技術の組み合わせによって達成される。

【００２１】
1)以下に記載されるようなバースト・バッファ５を使用した、ＤＲＡＭへのバースト・アクセス、
2)「ダブル・バッファリング」とよばれる技術を使用した、コプロセッサ２上の計算の同時実行、およびメイン・メモリ３とバースト・バッファ・メモリ５の間のデータ転送、および
3)プロセッサ１の実行をコプロセッサ２およびバースト・バッファ・メモリ５の実行から、命令待ち行列の使用を通じて切り離すことである。

【００２２】

「ダブル・バッファリング」は、たとえば、コンピュータ・グラフィックスにおいて知られている技術である。本明細書で使用される形式では、バースト・バッファ・メモリ５の一部からのデータを消費すること、すなわち読み取ること、他のデータを同じメモリの異なる領域へ生成すること、すなわち書き込むことを含み、先に書き込まれた領域を読み取り、その逆もできるスイッチング機構を有する。

【００２３】

バースト・バッファの利点は、従来のＤＲＡＭ構造の特徴の有効な利用である。ＤＲＡＭは、正方行列におけるメモリ位置のアレイを含む。アレイにおけるエレメントにアクセスするには、行が最初に選択され（あるいは「開かれ」）なければならず、その後に適切な列の選択が続く。しかし、行が選択された後、その行における列への連続アクセスは、単に列アドレスを提供することによって実行することができる。行を開くことおよびその行にローカルな一連のアクセスを実行することの概念が、「バースト」と呼ばれる。媒体集中計算（典型的には、いかなるデータ依存アドレス指定もなしに長いアレイにアクセスする、規則正しいプログラム・ループを使用するアルゴリズムを含む）におけるように、データが規則正しい方法で構成されるとき、バーストの有効な使用が劇的に計算速度を向上させることができる。バースト・バッファは、バーストの効率的な使用を通じてＤＲＡＭからのデータにアクセスするように適合された、新しいメモリ構造である。

【００２４】

システムがいくつかのバースト・バッファを含むことができる。典型的には、各バースト・バッファがそれぞれのデータ・ストリームへ割り振られる。アルゴリズムが、異なる数のデータ・ストリームを有するので、固定量のＳＲＡＭ２６がバースト・バッファ・メモリ領域としてバースト・バッファに使用可能であり、この量が、必要とされるバッファの数に従って分割される。たとえば、固定ＳＲＡＭの量が２Ｋバイトである場合、かつ、アルゴリズムが４つのデータ・ストリームを有する場合、メモリ領域を４つの５１２バイトのバースト・バッファに区分することができる。

【００２５】

このタイプのアーキテクチャでは、バーストが、以下によって定義されたアドレスの組を含む。

【００２６】

【数１】

バースト＝｛Ｂ＋Ｓ×ｉ｜Ｂ，Ｓ，ｉ∈Ｎ∧０≦ｉ＜Ｌ｝

【００２７】

ただし、Ｂは転送の基底アドレスであり、Ｓはエレメントの間のストライド(stride)であり、Ｌは長さであり、Ｎは自然数の組である。この式において明示的に定義されないが、バースト順序が、０からＬ－１まで増分するｉによって定義される。したがって、バーストは、次の３個のエレメントからなる集合によって定義することができる。

【００２８】

（ｂａｓｅ＿ａｄｄｒｅｓｓ，ｌｅｎｇｔｈ，ｓｔｒｉｄｅ）

ソフトウェアでは、バーストをエレメントサイズによって定義することもできる。これは、バーストをバイト、ハーフワード、またはワードのサイズにすることができることを意味する。ストライドの単位は、このことを考慮しなければならない。「サイズド・バースト」は、次の形式の４個のエレメントからなる集合によって定義される。

【００２９】

（ｂａｓｅ＿ａｄｄｒｅｓｓ，ｌｅｎｇｔｈ，ｓｔｒｉｄｅ，ｓｉｚｅ）

「チャネル・バースト」は、サイズがメモリへのチャネルの幅である、サイズド・バーストである。コンパイラが、ソフトウェア・サイズド・バーストからチャネル・バーストへのマッピングを担う。チャネル・バーストは、次の４個のエレメントからなる集合によって定義することができる。

【００３０】

（ｂａｓｅ＿ａｄｄｒｅｓｓ，ｌｅｎｇｔｈ，ｓｔｒｉｄｅ，ｗｉｄｔｈ）

チャネル幅が３２ビット（または４バイト）である場合、チャネル・バーストが常に以下の形式である。

【００３１】

（ｂａｓｅ＿ａｄｄｒｅｓｓ，ｌｅｎｇｔｈ，ｓｔｒｉｄｅ，４）

あるいは、３個のエレメントからなる集合（ｂａｓｅ＿ａｄｄｒｅｓｓ，１ｅｎｇｔｈ，ｓｔｒｉｄｅ）に短縮される。

【００３２】

このメモリの制御およびバースト・バッファの割り振り（および解放）が、ソフトウェア処理によって高レベルで処理される。この実施形態では、「ダブル・バッファリング」が使用されるが、他の戦略も確かに可能であり、この判断は記憶の効率と簡潔性の間のトレードオフを含む。バースト・バッファ・メモリ領域２６がメイン・メモリ３からデータをロードし、そこへデータを格納し、これはメモリ・データパス・アービタ５８を通じて行い、これがＤＭＡコントローラ５６の制御下で動作し、バースト命令待ち行列６を通じて受信される命令に応答する。データが、バースト・バッファ・メモリ領域２６とプロセッサ１またはコプロセッサ２の間で、接続手段１２を通じて交換される。図３に示すように、バースト・バッファ・システム５のための制御インタフェースは、１組のテーブルに基づいている。すなわち、バースト・バッファ・メモリへバーストし、そこからバーストするためのメイン・メモリの領域を記述するメモリ・アクセス・テーブル（ＭＡＴ）６５、およびバースト・バッファ・メモリの領域を記述するバッファ・アクセス・テーブル（ＢＡＴ）６６である。この実施形態では、デュアルポートＳＲＡＭの同次領域が、バースト・バッファ・メモリ領域２６のために使用される。

【００３３】

ＭＡＴおよびＢＡＴを使用しなかったバースト・バッファ構成（これも欧州特許出願第９７３０９５１４．４号に記載されているようなもの）を、本発明の代替実施形態において使用することができ、ＭＡＴおよびＢＡＴにおいて暗示的に符号化されたパラメータ（ソース・アドレス、宛先アドレス、長さ、ストライド）が、次いで、発行されたバースト転送毎に明示的に指定されなければならない。直接のアドレス、長さおよびストライドではなく、ＭＡＴおよびＢＡＴを使用するための主な理由は、これが全体のコード・サイズを著しく減らすことである。本発明に関連して、これは典型的には重要というよりも有用である。

【００３４】

プロセッサ１から発信するバースト命令が、バースト命令待ち行列６により、バースト・バッファ５へ提供される。バースト命令待ち行列６からの命令が、バッファ制御エレメント５４によって処理されて、ＭＡＴ６５およびＢＡＴ６６におけるスロットが参照される。バッファ・コントローラは、８個のバースト制御レジスタ５２からの制御入力も受信する。これらの２つのテーブルに含まれた情報が、実行時間で共に結び付けられて、完全なメイン・メモリ対バースト・バッファのトランザクションが記述される。出力がバッファ・コントローラ５４から直接メモリ・アクセス（ＤＭＡ）コントローラ５６へ、よってメモリ・データパス・アービタ５８へ提供されて、メイン・メモリ３とバースト・バッファ・メモリ領域２６の間のトランザクションが実施される。

【００３５】

重要なバースト命令は、データをメイン・メモリ３からバースト・バッファ・メモリ領域２６へロードするため、かつ、データをバースト・バッファ・メモリ領域２６からメイン・メモリ３へ格納するために使用されるものである。これらの命令は「ロードバースト」および「ストアバースト」である。ロードバースト命令が、データ・ワードのバーストを、メモリ３における決定された位置からバースト・バッファのそれへ転送させる。対応するストアバースト命令もあり、これは、データ・ワードのバーストを、バースト・バッファの１つからメモリ３へ、メモリ３における特定のアドレスで開始して、転送させる。図１のアーキテクチャでは、追加の同期化命令も必要とされ、これらは以下でさらに論じられる。

【００３６】
命令のロードバーストおよびストアバーストは、通常のロードおよび格納命令とは異なり、これらは転送が起こっていなくとも単一のサイクルで完了する。本質において、ロードバーストおよびストアバースト命令が、メモリ・インタフェース１６へバーストを実行するように伝えるが、これらはバーストが完了するまで待機しない。
【００３７】
基本動作は、２つのテーブル・エントリへ、メモリ・アクセスおよびバッファ・アクセス・テーブルのそれぞれにおける１つを索引付けする命令を発行することである。メモリ・アクセス・テーブルへの索引が、転送のメモリ側で使用された基底アドレス、エクステントおよびストライドを検索する。バッファ・アクセス・テーブルへの索引が、バースト・バッファ・メモリ領域内の基底アドレスを検索する。図示の実施形態では、マスキングおよびオフセットが索引値へ、コンテキスト・テーブル（これは欧州特許出願第９７３０９５１４．４号においてさらに論じられている）によって提供されるが、代りに実アドレスを使用することが可能である。直接メモリ・アクセス（ＤＭＡ）コントローラ５６がパラメータを２つのテーブルから渡され、これらを使用して、要求された転送を指定する。
【００３８】
表１は、可能な命令セットを示す。
【００３９】
【表１】

演算コード	パラメータ値	コメント
BB_LOADBURST	mat_index（整数）、 bat_index（整数）、 block_increment（ブール）	データのバーストをメイン・メモリからバースト・バッファ・メモリへロードし、任意選択でメイン・メモリにおける基底アドレスを増分する
BB_STOREBURST	mat_index（整数）、 bat_index（整数）、 block_increment（ブール）	データのバーストをバースト・バッファ・メモリからメイン・メモリへ格納し、任意選択でメイン・メモリにおける基底アドレスを増分する
BB_LX_INCREMENT	N/A	LX セマフォの値を増分する
BB_XS_DECREMENT	N/A	XS セマフォの値を減分する
BB_SET_MAT	entry（整数）、 memaddr（整数）、 extent（整数）、 stride（整数）	MAT エントリを所望の値に設定する
BB_SET_BAT	entry（整数）、 bufaddr（整数）、 extent（整数）	BAT エントリを所望の値に設定する

表１：バースト・バッファのための命令セット

【００４０】
ストアバースト命令（ＢＢ＿ストアバースト）が、ＭＡＴおよびＢＡＴにおけるパラメータを索引付け(index)し、これが要求された転送の特性を定義する。ｂｌｏｃｋ＿ｉｎｃｒｅｍｅｎｔビットが設定される場合、ＭＡＴにおいて索引付けされたエントリのｍｅｍａｄｄｒフィールドが、転送が完了するときに自動的に更新される（以下で論じられるように）。
【００４１】
ロードバースト命令（ＢＢ＿ロードバースト）も、ＭＡＴおよびＢＡＴにおけるパラメ

ータを索引付けし、再度これが要求された転送の特性を定義する。前のように、ｂｌｏｃｋ＿ｉｎｃｒｅｍｅｎｔビットが設定される場合、ＭＡＴにおいて索引付けされたエントリのｍｅｍａｄｄｒフィールドが、転送が完了するときに自動的に更新される。

【００４２】

必要とされた同期化命令が、Ｌｏａｄ－Ｅｘｅｃｕｔｅ ＩｎｃｒｅｍｅｎｔおよびｅＸｅｃｕｔｅ－Ｓｔｏｒｅ Ｄｅｃｒｅｍｅｎｔ（ＢＢ＿ＬＸ＿ＩＮＣＲＥＭＥＮＴおよびＢＢ＿ＸＳ＿ＤＥＣＲＥＭＥＮＴ）として提供される。ＢＢ＿ＬＸ＿ＩＮＣＲＥＭＥＮＴの目的は、特定のデータのバーストにおけるコプロセッサ２の実行が、必要とされたデータがロードバースト命令の後に続いてバースト・バッファ・メモリ５に到着した後に起こるようにすることである。ＢＢ＿ＸＳ＿ＤＥＣＲＥＭＥＮＴの目的は、ストアバースト命令の実行が、結果がメイン・メモリ３に格納され戻される（コプロセッサ２上の）計算の完了の後に続くようにすることである。

【００４３】

この実施形態では、これらの命令が動作する特定の機構は、２つのカウンタの組であり、それぞれ次のものを追跡する。すなわち、a)ストアバーストを受信する用意ができているバースト・バッファ・メモリ５における領域の数、およびb)完了されたロードバースト命令の数、である。

【００４４】

コプロセッサ２によるデータの要求が、ＬＸカウンタを減分することによって実行されるが、データの可用性は、ＸＳカウンタを増分することによって信号で通知される。これらのカウンタは、２つの特性を満たさなければならない。すなわち、これらがただ１つのシステム構成エレメントへいかなる所与のときにもアクセス可能でなければならないこと、およびこれらが使用不可能なデータを要求する処理を中断する能力を有していなければならないことである。

【００４５】

要求されるものにもっとも近く合致する既存の概念はセマフォであり、Ｄｉｊｋｓｔｒａ（「［Ｄｉｊｋｓｔｒａ １９６８］ Ｅ．Ｄｉｊｋｓｔｒａ，「Ｃｏ－ｏｐｅｒａｔｉｎｇ Ｓｅｑｕｅｎｔｉａｌ Ｐｒｏｃｅｓｓｅｓ」，Ｆ．Ｇｅｎｕｙｓ（編集者），Ｐｒｏｇｒａｍｍｉｎｇ Ｌａｎｇｕａｇｅｓ，Ｎｅｗ Ｙｏｒｋ：Ａｃａｄｅｍｉｃ Ｐｒｅｓｓ，（１９６８），４３－１１２ページ）に記載されている。したがって、用語「セマフォ」は、本発明の実施形態において使用されたカウンタを記載するために使用されるが、これらのカウンタがＤｉｊｋｓｔｒａによって記載されたセマフォには等しくないが広く類似していることに留意されたい。

【００４６】

セマフォの基本原理は以下の通りである。セマフォは整数値を含む。Ｗａｉｔ（）命令をセマフォ上で実行することはこの値をディクリメント(decrement、減分)するが、Ｓｉｇｎａｌ（）命令を実行することはこれをインクリメント(increment、増分)する。Ｗａｉｔ（）を、値がすでに０であるセマフォ上で実行すると、セマフォの値が増分されるまで、Ｗａｉｔ（）を実行しようと試みているソフトウェア処理またはハードウェア構成エレメントが停止する。

【００４７】

この実施形態では、ＢＢ＿ＸＳ＿ＤＥＣＲＥＭＥＮＴ命令がＸＳセマフォ（図１の１１）上でＷａｉｔ（）のように動作するが、ＢＢ＿ＬＸ＿ＩＮＣＲＥＭＥＮＴ命令はＬＸセマフォ（図１の１０）上でＳｉｇｎａｌ（）のように動作する。以下で記載されるように、コプロセッサ・コントローラ９が、反対に、Ｗａｉｔ（）をＬＸセマフォ１０上で、Ｓｉｇｎａｌ（）をＸＳセマフォ１１上で実行する。これらの命令の意味は、Ｄｉｊｋｓｔｒａの論文に記載されたものと同じにすることができるが、Ｓｉｇｎａｌ（）およびＷａｉｔ（）動作の構成全体は元の論文に記載されたものとは著しく異なる。システムの正確さに必要な、あるイベントの相対的時間順序が守られるようにするため、これらの命令が適切な順序で（以下でさらに論じられるように）発行される。

【００４８】
メモリ・アクセス・テーブル（ＭＡＴ）６５が、以下で図３を参照して記載される。これは、バースト・トランザクションに含まれたメイン・メモリ位置に関連する情報を保持するメモリ記述子テーブルである。ＭＡＴにおける各エントリは、メイン・メモリへのトランザクションを記述する、索引付けされたスロットである。この実施形態では、ＭＡＴ６５が１６個のエントリを含むが、異なる実施がもちろん可能である。各エントリが以下の３個のフィールドを含む。
１．メモリ・アドレス（ｍｅｍａｄｄｒ）。メイン・メモリにおける関連領域の開始アドレス。この位置が物理メモリ空間にあることが理想的であり、これは、仮想アドレス変換が２つの物理ページにまたがるバースト要求の結果となる可能性があり、これがメモリ・コントローラに難点を引き起こすからである。
２．エクステント（ｅｘｔｅｎｔ）。転送のエクステント。これは転送の長さであり、ストライドで乗算され、転送された最後のアドレス＋１を与える。転送の長さは、エクステントをストライドにより除算することによって計算され、これは、転送が完了した後で、関連するＢＡＴ６６（以下参照）のｂｕｆｓｉｚｅフィールドへ自動的にコピーされる。
３．ストライド（ｓｔｒｉｄｅ）。転送における連続したエレメントの間の間隔。
【００４９】
ｍｅｍａｄｄｒ：チャネル・バーストの第１のエレメントの３２ビット、符号なし、語調整されたアドレスである。
【００５０】
ｅｘｔｅｎｔ：ｅｘｔｅｎｔレジスタにおけるパラメータは、バースト転送の範囲を包含するアドレス・オフセットである。転送がＳのストライドによって分離されたＬ個のエレメントを必要とする場合、エクステントはＳ＊Ｌである。
【００５１】
ｓｔｒｉｄｅ：パラメータｓｔｒｉｄｅは、アクセスの間でスキップされたバイト数である。転送ストライド間隔の値は、１から１０２４までの範囲に制限される。１０２４より大きい値は自動的に１０２４に切り捨てられる。このレジスタの読取りが、バーストのために使用された値を戻す（すなわち、切り捨てが必要であった場合、切り捨てられた値が戻される）。ストライドはメモリ・バス幅の倍数でなければならず、この場合は４バイトである。自動切り捨て（丸めなし）が、この位置合わせを実施するために実行される。
【００５２】
ＭＡＴスロットによって含まれた値の一例は、以下のようになる可能性がある。
【００５３】
｛０ｘ１ｆｅｅｌｂａｄ，１２８，１６｝
これは、３２語（３２個の４バイト語）バーストの結果となり、各語が４語（４個の４バイト語）によって分離される。
【００５４】
バースト命令の自動増分標識ビットは、ＭＡＴ６５にも関連している。このビットがバースト命令において設定される場合、もしバーストが３２を越えて継続しているならば、開始アドレス・エントリがポイント・トゥ・ポイントで、次のメモリ位置へ増分される。これは、長いシーケンスのメモリ・アクセスにおいて次のバーストのための開始アドレスを計算することにおける、プロセッサ・オーバーヘッドを節減する。
【００５５】
バッファ・アクセス・テーブル（ＢＡＴ）６６が、以下で図３を参照して記載される。これもまたメモリ記述子テーブルであり、この場合はバースト・バッファ・メモリ領域２６に関する情報を保持する。ＢＡＴ６６における各エントリが、バースト・バッファ・メモリ領域２６へのトランザクションを記述する。ＭＡＴ６５の場合のように、ＢＡＴ６６が１６個のエントリを含むが、もちろんＭＡＴ６５の場合のように変わる可能性がある。この場合は各エントリが以下の２つのフィールドを含む。

１．バッファ・アドレス（ｂｕｆａｄｄｒ）。バッファ領域におけるバッファの開始。
２．バッファ・サイズ（ｂｕｆｓｉｚｅ）。最後の転送で使用されたバッファ領域のサイズ。

【００５６】

バッファ・アドレス・パラメータｂｕｆａｄｄｒは、バッファ領域におけるチャネル・バーストの第１のエレメントのためのオフセット・アドレスである。バースト・バッファ領域は、ハードウェアによってプロセッサのメモリ空間の領域へ物理的にマッピングされる。これは、バースト・バッファ領域にアクセスするときにプロセッサが絶対アドレスを使用しなければならないことを意味する。しかし、ＤＭＡ転送は単にオフセットを使用し、そのため、ハードウェアが、要求されたいかなるアドレス解決をも管理することが必要である。不当に位置合わせされた値は、切り捨てによって自動的に位置合わせすることができる。このレジスタの読取りが、バーストのために使用された値を戻す（すなわち、切り捨てが必要であった場合、切り捨てられた値が戻される）。デフォルト値は０である。

【００５７】

パラメータｂｕｆｓｉｚｅは、もっとも最近のバーストによって占有されたバッファ領域内の領域のサイズである。このレジスタは、そのエントリを目標としたバースト転送の完了時に、自動的に設定される。格納された値がバーストの長さであり、これは、０の値が使用されていないバッファ・エントリを指示するからであることに留意されたい。このレジスタに書き込むことができるが、これは、バッファが保存されリストアされるときに、文脈切り換え後にのみ有用である。デフォルト値は再度０である。

【００５８】

ＭＡＴおよびＢＡＴエントリをプログラムすることが、ＢＢ＿ＳＥＴ＿ＭＡＴおよびＢＢ＿ＳＥＴ＿ＢＡＴ命令の使用を通じて実行される。エントリ・パラメータが、現在の命令が参照するＭＡＴ（またはＢＡＴ）におけるエントリを決定する。

【００５９】

バースト・バッファ・アーキテクチャおよびその制御のための機構のさらなる詳細が、欧州特許出願第９７３０９５１４．４号および対応する米国特許出願第０９／３，５２６号において提供される。上記で提供された詳細は、主として、バースト・バッファ・システムのアーキテクチャエレメントを示し、バースト・バッファ・システムが実施することができる機能効果を、それが提供する入力および出力と共に示すように意図される。バースト・バッファ・システムが、特定のタイプの計算モデルに合わせて最適に適合され、これは、本明細書で、本発明の記載された実施形態のための計算モデルに展開される。この計算モデルについてさらに説明する。

【００６０】

バースト命令待ち行列６が上で記載された。この実施形態の著しい態様は、命令が類似の方法でコプロセッサへ、コプロセッサ命令待ち行列８を通じて提供されることである。コプロセッサ命令待ち行列８は、コプロセッサ・コントローラ９との接続において動作し、これは、コプロセッサがプロセッサ１から命令を受信する方法、およびそれがデータをバースト・バッファ・システム５と交換する方法を決定する。

【００６１】

コプロセッサ命令待ち行列８の使用は、プロセッサ１自体が計算自体から切り離される重要な効果を有する。したがって、計算中に、プロセッサ・リソースが他のタスクの実行のために使用可能である。プロセッサ１の動作がストールされることにつながる可能性のある唯一の状況は、命令待ち行列６、８の一方が命令で満たされることである。この場合は、プロセッサ１が、いずれかの待ち行列のための命令を、命令が消費されるよりも速い速度で生成するときに起こる可能性がある。この問題の解決策は入手可能である。事前定義の時間量の後、あるいは、いずれかの待ち行列において占有されたスロットの数が事前定義の量まで減分された事実によってトリガされた割り込みの受信の上で、文脈切り換えを実行してこれら２つの待ち行列にサービスするために戻るように、プロセッサ１へ要求することによって、有効性を改善することができる。反対に、プロセッサ１が、命令が消

費される速度に遅れないでいることができないために、２つの待ち行列の一方が空になる場合、これらの命令の消費者（コプロセッサ・コントローラ９またはバースト・バッファ・コントローラ７）は、新しい命令がプロセッサ１によって生成されるまで、ストール（機能停止）する。

【００６２】

修正を、プロセッサ１からのそれ以上の関与がまったく必要とされないようにするアーキテクチャにも提供することができ、これらについては本明細書の最後の部分において論じられる。

【００６３】

コプロセッサ・コントローラ９の基本機能は、データをバースト・バッファ・メモリ５からコプロセッサ２へ（およびコプロセッサ２からバースト・バッファ・メモリ５へ）取り出すこと、コプロセッサの動作を制御すること、およびコプロセッサ２の実行をバースト・バッファ・メモリ５からの適切なロードまたはそれへの格納と同期化することである。これらの機能を達成するには、コプロセッサ・コントローラを、本質において、ある規則に従ってアドレスを生成することができる、相対的に簡素な状態機械にすることができる。

【００６４】

図４は、コプロセッサ・コントローラ９を、アーキテクチャの他の構成エレメントとの関係において示し、その構成エレメント、およびアーキテクチャ全体における他のエレメントとの接続も示す。その厳密な機能は、コプロセッサ２およびその初期化要件（ある場合は）によって必要とされた入力および出力のタイプに依存し、そのため、以下に記載されたものから詳細において変わる可能性がある。ＣＨＥＳＳコプロセッサの場合、これらの入力および出力が、バースト・バッファ・メモリ５と交換された入力および出力データ・ストリームである。

【００６５】

コプロセッサ・コントローラ９は、次の２つの主なタスクを実行する。すなわち、1)コプロセッサ２とバースト・バッファ・メモリ５との間の通信の制御、および2)制御有限状態機械４２の使用を通じたシステム状態の維持である。

【００６６】

コプロセッサ２がストリームにおけるデータにアクセスし、そのそれぞれがいくつかの制御レジスタ４１の１つとの関連付けを与えられる。これらのレジスタ４１のためのアドレスが、制御有限状態機械４２によってアドレス指定ロジック４３と共に、有限状態機械４２によって生成されたシーケンスに従って、周期的な様式において生成される。

【００６７】

有限状態機械４２内のクロックの各チックで、有限状態機械は、レジスタ４１の（多くとも）１つが、そのために生成された新しいアドレス、およびレジスタ４１がバースト・バッファ・メモリ５をアドレス指定できるようにするために使用されたアドレスを有するための許可を与える。同時に、適切な制御信号が有限状態機械４２によって生成され、マルチプレクサ４４へ送信されて、適切なアドレスが正しい読み／書き信号と共にバースト・バッファ・メモリ５へ送信されるようにする。特定の読み／書き信号が各レジスタ４１に、全体の計算中で変化しない値と共に関連付けられる。

【００６８】

レジスタ４１のために得られたアドレスがメモリをアドレス指定するために使用された後、一定量がその値へ追加され、これは一般にコプロセッサ２とバースト・バッファ・メモリ５との間の接続の幅と同じである。つまり、この接続の幅が４バイトである場合、カウンタ４１に行われた増分が４となる。これは、本質的に、バースト・バッファのプログラミングにおける「ストライド」に比較可能である。

【００６９】

上述のコプロセッサ・コントローラの機構は、単一のバスに沿った異なるデータ・ストリームの多重化を可能にする。各データ・ストリームは、それ自体のポートを通じて単一

の共有バスへアクセスするとみなすことができる。

【００７０】

このシステムが、通信の完全性が保証されるように動作するためには、バスの他方の端でコプロセッサ２が、同期した方法でこのバスから読み取り、このバスへ書き込みする用意ができていることが必要である。アプリケーション・ソフトウェア（および具体的には、コプロセッサ２を構成するアプリケーション・ソフトウェアの一部まで）の責任は、つぎのことを保証することである。すなわち、1) ２つのストリームが同時にバスにアクセスしようと試みないこと、および2)コプロセッサ２の実行がバースト・バッファ・メモリ５とのデータ転送と同期であることである。

【００７１】

この後者の要件は、コプロセッサ２が、２つのデバイスの間の接続上で、バースト・バッファ・メモリ５によって配置されたデータを読み取るため、かつその逆のための用意ができることを保証する。

【００７２】

通常は、複数の物理線をＣｈｅｓｓアレイ２とバースト・バッファ・メモリ５の間に設けることができるが、一般の多重化の必要性はなお残る。コプロセッサ２とバースト・バッファ・メモリ５の間の物理接続の数が、コプロセッサ２のための論理Ｉ／Ｏストリームの合計数以上でない限り、２つ以上の論理ストリームが同じワイヤ上で多重化されなければならないことが常に真となる。（バースト・バッファ・メモリ５に有利に使用されるように）高速ＳＲＡＭの設計に関係した技術的な理由が、コプロセッサ２との複数の接続の使用を防止する。

【００７３】

コプロセッサ・コントローラ９は、コプロセッサ２を含むＣＨＥＳＳアレイの実行を制御するようにも動作し、それが指定数のクロック・サイクルで実行するようにする。これは、コプロセッサ２におけるパイプラインの内部状態に影響を与えない方法において、ＣＨＥＳＳアレイを、その内部クロックを「ゲーティング」すること（つまり、停止すること）によって「フリーズ」する前に、指定数のサイクルだけチッキングする、制御有限状態機械４２におけるカウンタによって達成される。このチックの数が、以下に記載されたＣＣ＿ＳＴＡＲＴ＿ＥＸＥＣ命令を使用して指定される。

【００７４】

コプロセッサ・コントローラ９が、プロセッサ１によって、コプロセッサ命令待ち行列８の使用を通じてプログラムされる。このコプロセッサ・コントローラ９のための可能な命令セットが、以下の表２において示される。

【００７５】

【表２】

演算コード	パラメータ値	コメント
CC_CURRENT_PORT	n（整数）	次の CC_PORT_xxx コマンドが参照するポート番号
CC_PORT_PERIOD	（整数）	ポートの活動の周期
CC_PORT_PHASE_START	$_{start}$（整数）	ポートの活動の段階開始
CC_PORT_PHASE_END	$_{end}$（整数）	ポートの活動の段階終了
CC_PORT_TIME_START	t_{start}（整数）	ポートの活動の開始サイクル
CC_PORT_TIME_END	t_{end}（整数）	ポートの活動の終了サイクル
CC_PORT_ADDRESS	$addr_{start}$（整数）	ポートのための初期アドレス
CC_PORT_INCREMENT	$addr_{end}$（整数）	ポートのためのアドレス増分
CC_PORT_IS_WRITE	rw（ブール）	読み／書きフラグ
CC_START_EXEC	n_{cycles}（整数）	コプロセッサ２の実行を、決定されたサイクルの数だけ開始／再開する
CC_LX_DECREMENT	N/A	LX セマフォの値を減分する
CC_XS_INCREMENT	N/A	XS セマフォの値を増分する

表２：コプロセッサ・コントローラ命令セット

【００７６】

前記の命令では、命令フォーマットの異なる選択を行うことができる。１つの可能なフォーマットは３２ビットの数字であり、１６ビットが演算コードを符号化し、１６ビットが上述の任意選択のパラメータ値を符号化する。

【００７７】

個々の命令の意味は、以下の通りである。

・ＣＣ＿ＣＵＲＲＥＮＴ＿ＰＯＲＴは、ポートの１つを、すべての後続のＣＣ＿ＰＯＲＴ＿ｘｘｘ命令の受信者として、次のＣＣ＿ＣＵＲＲＥＮＴ＿ＰＯＲＴまで、選択する。

・ＣＣ＿ＰＯＲＴ＿ＰＥＲＩＯＤ（）は、現在のポートの活動化の周期を、整数パラメータの値に設定する。

・ＣＣ＿ＰＯＲＴ＿ＰＨＡＳＥ＿ＳＴＡＲＴ／ＣＣ＿ＰＯＲＴ＿ＰＨＡＳＥ＿ＥＮＤ（$_{start\ end}$）は、現在のポートの活動化の段階の開始／終了を、整数のパラメータ（$_{start\ end}$）の値に設定する。

・ＣＣ＿ＰＯＲＴ＿ＴＩＭＥ＿ＳＴＡＲＴ／ＣＣ＿ＰＯＲＴ＿ＴＩＭＥ＿ＥＮＤ（$t_{start}\ t_{end}$）は、現在のポートの活動の最初／最後のサイクルを設定する。

・ＣＣ＿ＰＯＲＴ＿ＡＤＤＲＥＳＳ（$addr_{start}$）は、現在のポートの現在のアドレスを、整数のパラメータ$addr_{start}$の値に設定する。

・ＣＣ＿ＰＯＲＴ＿ＩＮＣＲＥＭＥＮＴ（$addr_{incr}$）は、現在のポートのアドレス増分を、整数のパラメータ$addr_{incr}$の値に設定する。

・ＣＣ＿ＰＯＲＴ＿ＩＳ＿ＷＲＩＴＥ（ｒｗ）は、現在のポートのためのデータ転送方向を、ブール・パラメータｒｗの値に設定する。

・ＣＣ＿ＳＴＡＲＴ＿ＥＸＥＣ n_{cycles}は、コプロセッサ・コントローラ２の実行を、関連付けられた整数パラメータn_{cycles}によって指定されたクロック・サイクルの数だけ開始する。

・ＣＣ＿ＬＸＳ＿ＤＥＣＲＥＭＥＮＴは、ＬＸセマフォの値を（以前に記載されたように、中断の方法において）減分する。

・ＣＣ＿ＸＳＳ＿ＩＮＣＲＥＭＥＮＴは、ＸＳセマフォの値を増分する。

【００７８】

カウンタ４２の現在値、t_{cur}が、t_{start} t_{cur}＜t_{end}であり、start（t_{cur} mod）＜$_{end}$である場合、ポートが能動（つまり、バースト・バッファ・メモリ５との通信の制御を有する）として定義される。これは、たとえば、２つのストリームが、等しい周期、言わば５で存在し、一方が最初の４サイクルのためのＢＢメモリの制御を有し、他方が残りのサイクルのための制御を有する、システムの可能性を可能にする。

【００７９】

このアーキテクチャを使用するアルゴリズムを実行する処理は、最初にコプロセッサ２のプログラミング、次いでコプロセッサ・コントローラ９およびバースト・バッファ・コントローラ７のプログラミングまたは初期化と、その後に続くアルゴリズムの実際の実行を含む。

【００８０】

コプロセッサ２の初期化では、デバイスの実際の実施形態に特定の手段によって、構成がコプロセッサ自体にロードされることが、一般にもっとも直接になる。

【００８１】

コプロセッサ・コントローラ９のプログラミングでは、そのステップは次の通りである。

１．メイン・コプロセッサ・コントローラ９が、前に記載されたように、Ｃｈｅｓｓアレイにおいて存在する各論理ストリーム毎の合計数、周期、段階およびアドレス増分に従って構成される。所望の機能を実行するためのコプロセッサ・コントローラ９のプログラミングの一例が、下に示される。

２．コプロセッサ・コントローラ９の構成における次のステップは、アドレス構成である。各論理ストリームの特性（周期、段階）がアルゴリズム中で同じであり続ける可能性が高いが、バースト・バッファ・メモリ５におけるコプロセッサ・コントローラ９によってアクセスされた実アドレスは変わる。それはこの可変性であり、これは、バースト・バッファ・コントローラ７がダブル・バッファリングを、バースト・バッファ・アーキテクチャ内で直接の方法で実行できるようにする。このダブル・バッファリングの効果は、先に述べられたように、コプロセッサ２に、それが連続ストリームと対話中である印象を与えることであるが、実際にはバッファが連続的に交換されている。

【００８２】

バースト・バッファ・コントローラ７も構成される必要がある。これを行うには、メイン・メモリ３からバースト・バッファ・メモリ５へのデータの転送を構成するために、適切なコマンドがバースト命令待ち行列６へ送信されなければならない。これらの命令（ＢＢ＿ＳＥＴ＿ＭＡＴおよびＢＢ＿ＳＥＴ＿ＢＡＴ）が、ＢＡＴおよびＭＡＴ内で適切なエントリを、コプロセッサ・コントローラ９のプログラミングに適合する方法で構成する。この実施形態では、ＭＡＴおよびＢＡＴエントリをプログラムするための命令が、バースト命令待ち行列６を通じて発行される。代替の可能性は、プロセッサ１が読み書きするメモリ・マップ・レジスタの使用となる。この実施形態の場合のように、メモリ・マップ・レジスタから読み取られる可能性がなく（それらが存在しないので）、プロセッサ１がバースト・バッファ・コントローラ７の状態を照会することができないが、これは著しい制限ではない。さらに、この目的のためのバースト命令待ち行列６の使用は、バースト転送の実行によりＭＡＴおよびＢＡＴエントリを構成するための命令をインターリービングし、したがって、プロセッサ１の監視なしに正しい時間的意味を維持する可能性を可能とする。

【００８３】

これらのステップが実行された後、ＣＨＥＳＳアレイの実際の実行を開始することができる。この実施形態では、ＣＨＥＳＳアレイに指定数のサイクルで実行するように命令することのみが必要である。これは、正確な数のサイクルをパラメータとして、コプロセッサ命令待ち行列８におけるＣＣ＿ＳＴＡＲＴ＿ＥＸＥＣ命令へ書き込むことによって達成され、次いでこのデータをコプロセッサ・コントローラ９へ渡すことができるようにする

。この値がコプロセッサ・コントローラ９に転送された１クロック・サイクル後、コントローラが値をバースト・バッファ・メモリ５とコプロセッサ２のＣＨＥＳＳアレイの間で転送することを開始し、ＣＨＥＳＳアレイの実行を可能にする。

【００８４】

しかし、重要なステップを、計算に関係する命令がそれぞれの命令待ち行列に配置される前に追加しなければならない。これは、必要な同期化機構が、同期化およびダブル・バッファリングの原理をうまく実施するために適切であるようにするためである。この機構における基本エレメントは、コプロセッサ・コントローラ９がＬＸセマフォの値を減分しようと試み、上述のロジックに従って、それがそうできるようになるまでコプロセッサの動作を中断することである。このセマフォの初期値は０である。すなわち、したがって、コプロセッサ・コントローラ９およびコプロセッサ２がこの段階で「フリーズ」される。成功したロードバースト命令の後にＬＸセマフォの値がバースト・バッファ・コントローラ７によって増分されるときにのみ、コプロセッサ２がその実行を開始（あるいは再開）できるようになる。この効果を達成するため、ＣＣ＿ＬＸ＿ＤＥＣＲＥＭＥＮＴ命令がコプロセッサ命令待ち行列８において、「コプロセッサ２の実行を開始する」（ＣＣ＿ＳＴＡＲＴ＿ＥＸＥＣ）命令の前に挿入される。以下に示されるように、対応する「ＬＸセマフォを増分する」（ＢＢ＿ＬＸ＿ＩＮＣＲＥＭＥＮＴ）命令が、バースト命令待ち行列６において、対応するロードバースト命令の後に挿入される。

【００８５】

ＣＨＥＳＳ論理ストリームおよびバースト・バッファ・メモリ５の間の実際のデータの転送が、先に記載されたようなコプロセッサ・コントローラ９のプログラミングに従って実行される。

【００８６】

カウンタ４２が実行しなければならないチックの数は、１つまたは複数の入力バーストを消費するためにどれだけの時間がかかるかに依存する。システムの正確さを保証することは、アプリケーション・ソフトウェアに任される。カウンタ４２のプログラミングは、バッファが消費された後にコプロセッサ２の実行が停止するようにしなければならない。コプロセッサ命令待ち行列８における次の命令は、次のデータのバーストがバースト・バッファ・メモリ５に到着しているようにするため、同期化命令（つまり、ＣＣ＿ＬＸ＿ＤＥＣＲＥＭＥＮＴ）でなければならない。この命令（および場合によっては、必要とされたデータが使用可能になるまでの待機期間）の後に続いて、この新しいデータのバーストの初期アドレスがデータ・ストリームに割り当てられ（ＣＣ＿ＰＯＲＴ＿ＡＤＤＲＥＳＳ命令により）、実行がＣＣ＿ＳＴＡＲＴ＿ＥＸＥＣ命令を介して再開される。この手順は出力ストリームの場合に類似している（重要な違いは、データがメイン・メモリ３からバースト・バッファ・メモリ５に到着するために必要としたものに等しい待機期間がなくなることである）。

【００８７】

計算モデル

計算モデル全体の例を図５を参照して説明する。この例は、アルゴリズムがこのアーキテクチャにおいて使用するためにどのように記録することができるかを指示し、一例として簡素なベクトル加算を使用し、これは従来のマイクロプロセッサ向けに以下のようにＣでコーディングすることができる。

【００８８】

【表３】

```
int a[1024], b[1024], c[1024];
for(i=0; i<1024; i++)
  a[i]=b[i]+c[i];
```

【００８９】

図１のアーキテクチャ上で元のベクトル加算ループ・ネストと同じ機能性を達成するようプロセッサ１を走らせるＣコードの一部分は、次の通りである。

【００９０】
【表４】

```
0:  int a[1024], b[1024] c[1024];
1:  int eo, not_eo, k;
2:  /*Port 0 specification: port #, increment, xfer size, period,
3:  phase start, phase end, start time, end time, r/w*/
4:  CIQ_STREAM( 0, 4, 4, 3, 0, 1, 0, 3*BLEN*MAXK+3, 0 );
5:  /*Port 1 specification*/
6:  CIQ_STREAM( 1, 4, 4, 3, 1, 2, 0, 3*BLEN*MAXK+3, 0 );
7:  /*Port 2 specification*/
8:  CIQ_STREAM( 2, 4, 4, 3, 2, 3, 0, 3*BLEN*MAXK+3, 1 );
9:  BIQ_SET_MAT(0, &b[0], BLEN*4, 4);
10: BIQ_SET_MAT(1, &c[0], BLEN*4, 4);
11: BIQ_SET_MAT(2, &a[0], BLEN*4, 4);
12: BIQ_SET_BAT(0, 0x0000, BLEN*4); BIQ_SET_BAT(1, 0x0100, BLEN*4);
13: BIQ_SET_BAT(2, 0x0200, BLEN*4); BIQ_SET_BAT(3, 0x0300, BLEN*4);
14: BIQ_SET_BAT(4, 0x0400, BLEN*4); BIQ_SET_BAT(5, 0x0500, BLEN*4);
15: for( k = 0; k < MAXK; k++ )
16: {
17:   /*Even or odd iteration? - For double buffering*/
18:   eo = k&0x1;
19:   CIQ_LXD(2);
20:   CIQ_SA(0,(BLEN*4*eo));
21:   CIQ_SA(1,((2*BLEN*4)+BLEN*4*eo));
22:   CIQ_SA(2,((4*BLEN*4)+BLEN*4*eo));
23:   /*Start Chess*/
24:   CIQ_ST(3*BLEN);
25:   CIQ_XSI(1);
26:   /*BB stuff*/
27:   /*Load A*/
28:   BIQ_FLB(0,eo);
29:   /*Load B*/
30:   BIQ_FLB(2,2+eo);
31:   BIQ_LXI(2);
32:   if( k >= 1 )
33:   {
34:     not_eo = (eo==0)?1:0;
35:     BIQ_XSD(1);
36:     BIQ_FSB(4,4+not_eo);
37:   }
38: }
39: eo = MAXK & 0x1;
40: not_eo = (eo==0)?1:0;
41: BIQ_XSD(1);
42: BIQ_FSB(4,4+not_eo);
```

【００９１】
この構成では、３つのポートがコプロセッサ・コントローラ９において使用され、すなわち、各入力ベクトルのためのもの（ｂおよびｃ）、および出力ベクトルのためのもの（ａ）である。行４、６および８の文は、これらの３つのポートを初期化するためのコード・マクロである。これらは、拡張されるとき、次の表のコマンドになる（これは行４を参照し、他の拡張されたマクロは直接類似している）。
【００９２】
【表５】

ＣＣ＿ＣＵＲＲＥＮＴ＿ＰＯＲＴ（０）；
ＣＣ＿ＰＯＲＴ＿ＩＮＣＲＥＭＥＮＴ（４）；
ＣＣ＿ＴＲＡＮＳＦＥＲ＿ＳＩＺＥ（４）；
ＣＣ＿ＰＯＲＴ＿ＰＥＲＩＯＤ（３）；

ＣＣ＿ＰＯＲＴ＿ＰＨＡＳＥ＿ＳＴＡＲＴ（０）；
ＣＣ＿ＰＯＲＴ＿ＰＨＡＳＥ＿ＥＮＤ（１）；
ＣＣ＿ＰＯＲＴ＿ＳＴＡＲＴ＿ＴＩＭＥ（０）；
ＣＣ＿ＰＯＲＴ＿ＥＮＤ＿ＴＩＭＥ（３＊ＢＬＥＮ＊ＭＡＸＫ＋３）；
ＣＣ＿ＰＯＲＴ＿ＩＳ＿ＷＲＩＴＥ（０）；

【００９３】

このコードは、ポート０が、カウンタ４２の３チック毎、正確にはチック０、３、６．．．３＊ＢＬＥＮ＊ＭＡＸＫ＋３の４バイトのデータを読み取り、それが読み取るアドレスを毎回４バイト増分するという効果を有する。ＢＬＥＮ＊ＭＡＸＫは、合計する２つのベクトルの長さ（この場合、１０２４）であり、ＢＬＥＮはＤＲＡＭからの単一のデータのバーストの長さ（たとえば、６４バイト）である。これらの値で、ＭＡＸＫは１０２４／６４＝１６に設定される。

【００９４】

行９から１４は、バースト・バッファ転送のためのＭＡＴおよびＢＡＴを確立し、これらのテーブルにおけるエントリをメイン・メモリ３およびバースト・バッファ・メモリ５におけるアドレスに結合する。コマンドＢＩＱ＿ＳＥＴ＿ＭＡＴ（０，＆ｂ［０］，ＢＬＥＮ＊４，４，ＴＲＵＥ）はコード・マクロであり、これはＢＢ＿ＳＥＴ＿ＭＡＴ（０，＆ｂ［０］，ＢＬＥＮ＊４，４）に拡張され、ＭＡＴにおけるエントリ０をアドレス＆ｂ［０］へ結合し、バースト長をＢＬＥＮ＊４バイトに（つまり、整数が３２ビットの場合、ＢＬＥＮ整数）、ストライドを４に設定する。後に続く２行は、ｃおよびａに類似し、関係する。行ＢＩＱ＿ＳＥＴ＿ＢＡＴ（０，０ｘ０００００，ＢＬＥＮ＊４）は、ＢＢ＿ＳＥＴ＿ＢＡＴ（０，０ｘ０００００，ＢＬＥＮ＊４）に拡張され、ＢＡＴのエントリ０をバースト・バッファ・メモリ５におけるアドレス０ｘ０００００へ結合する。後に続く２行は再度類似している。

【００９５】

この点まで、計算は行われていないが、コプロセッサ・コントローラ９およびバースト・バッファ・コントローラ７が設定されている。行１５から３８のループ・ネストは、実際の計算が行われるところである。このループはＭＡＸＫ回繰り返され、各反復がＢＬＥＮエレメント上で動作し、処理されたＭＡＸＫ＊ＢＬＥＮエレメントの合計を与える。このループは、コプロセッサ命令待ち行列８へ送信された命令の組ＣＩＱ＿ｘｘｘで開始して、コプロセッサ２およびコプロセッサ・コントローラ９の動作を制御し、その後に、バースト・バッファ・コントローラ７およびバースト・バッファ・メモリ５を制御することが目的である、バースト命令待ち行列６へ送信された１組の命令が続く。これら２組の相対的な順序は、原理においては重要でなく、これは、異なるシステムエレメントの間の同期化が、セマフォによって明示的に保証されるからである。互いの後に実行する２つの異なるループを有すること（２つの命令待ち行列が十分深かったと仮定して）、または、２つの異なる制御のスレッドを有することさえも可能となる。

【００９６】

ＣＩＱ＿ｘｘｘ行は、ソース・コードを書くことを簡約にするコード・マクロである。これらの意味は、次の通りである。

ＣＩＱ＿ＬＸＤ（Ｎ）：Ｎ個のＣＣ＿ＬＸＳ＿ＤＥＣＲＥＭＥＮＴ命令をコプロセッサ命令待ち行列８に挿入する。

ＣＩＱ＿ＳＡ（ポート、アドレス）：ＣＣ＿ＣＵＲＲＥＮＴ＿ＰＯＲＴ（ポート）およびＣＣ＿ＰＯＲＴ＿ＡＤＤＲＥＳＳ（アドレス）命令をコプロセッサ命令待ち行列８に挿入する。

ＣＩＱ＿ＳＴ（ｃｙｃｌｅｎｏ）：コプロセッサ２にカウンタ４２のｃｙｃｌｅｎｏチックだけ実行させるために、ＣＣ＿ＥＸＥＣＵＴＥ＿ＳＴＡＲＴ（ｃｙｃｌｅｎｏ）命令を挿入する。

ＣＩＱ＿ＸＳＩ（Ｎ）：Ｎ個のＣＣ＿ＸＳＳ＿ＩＮＣＲＥＭＥＮＴ命令をコプロセッサ命令待ち行列８に挿入する。

【００９７】
上に示したコードのネットの効果は、以下の通りである。すなわち、1)ＬＸＳセマフォ上で対応するロードバーストと同期化すること、2)コプロセッサ２上で計算を、カウンタ４２の３＊ＢＬＥＮチックだけ開始すること、および3)ＸＳＳセマフォ上で対応するストアバーストと同期化することである。
【００９８】
ＢＩＱ＿ｘｘｘ行は再度、ソース・コードを書くことを簡約にするコード・マクロである。これらの意味は、次の通りである。
ＢＩＱ＿ＦＬＢ（ｍａｔｅ，ｂａｔｅ，ＴＲＵＥ）命令をバースト命令待ち行列６に挿入する。
ＢＩＱ＿ＬＸＩ（Ｎ）：Ｎ個のＢＢ＿ＬＸ＿ＩＮＣＲＥＭＥＮＴ命令をバースト命令待ち行列６に挿入する。
ＢＩＱ＿ＦＳＢ（ｍａｔｅ，ｂａｔｅ，ＴＲＵＥ）：ＢＢ＿ストアバースト（ｍａｔｅ，ｂａｔｅ，ＴＲＵＥ）命令をバースト命令待ち行列６に挿入する。
ＢＩＱ＿ＸＳＤ（Ｎ）：Ｎ個のＢＢ＿ＸＳ＿ＤＥＣＲＥＭＥＮＴ命令をバースト命令待ち行列６に挿入する。
【００９９】
上に示したコードのネットの効果は、メインＤＲＡＭメモリ３からバースト・バッファ・メモリ５へ２つのバーストをロードし、次いでＬＸセマフォ１０の値を増分して、コプロセッサ２がその実行を上述のように開始できるようにすることである。最初のものを除くすべての反復において、コプロセッサ２の計算の結果が、次いで、ストアバースト命令を使用してメイン・メモリ３に戻すように格納される。２番目の反復が、最初の反復において実行された計算の結果を格納することを待機する必要は厳密にはないが、これはコプロセッサ２とバースト・バッファ・メモリ５の間の並行性を向上させる。
【０１００】
２つの変数ｅｏおよびｎｏｔ＿ｅｏの使用は、先に記載されたダブル・バッファリング効果を可能にするためにここで使用された機構である。
【０１０１】
行３９から４２は、バースト・バッファ・メモリ５からメイン・メモリ３への最後のバースト転送を実行し、ループ本体の最初の反復におけるストアバースト命令の不在を補償する。
【０１０２】
結果として生じる時間線は図６の通りである。ロードバースト６０１は最初の動作であり（これらが完了されるまで、コプロセッサ２がロード／実行セマフォによってストールされるので）、これらが完了されるとき、コプロセッサ２が６０２を実行開始することができる。バースト命令待ち行列６における次の命令は別のロードバースト６０１であり、これが、最初の２つのロードが終了するとすぐに実行される。次いで、バースト命令待ち行列６における次の命令がストアバースト６０３であり、これは、ＸＳセマフォ１１が、コプロセッサ２上の最初の計算が完了したことを信号で知らせるまで待機しなければならない。この処理は、ループ中で継続する。
【０１０３】
上に示された例は非常に簡素なアルゴリズムの場合であるが、これは、より複雑である計算において必要とされる基本原理を例示する。当業者は、上に示された手法、原理および技術を、より複雑なアルゴリズムをこのアーキテクチャによる実行に適合させるために図１のアーキテクチャをプログラムすることに使用できる。
【０１０４】
計算のためのツールチェーン
計算モデルの原理を、ハンド・コーディングによって直接の様式で活用することができる。つまり、手動でＣコードを書いて、従来の方法でシステム構成エレメントの適切な動作をスケジュールするように適合されたＣＰＵ上で実行して（命令を適切な待ち行列に配

置し、記載されたようにシステム構成エレメントを動作に設定し）、コプロセッサのための適切な構成を、そのコプロセッサを構成するための標準の合成ツールに従って提供することである。ＣＨＥＳＳのような、構成可能またはＦＰＧＡに基づいたプロセッサでは、このツールが一般にハードウェア記述言語となる。ＣＨＥＳＳに使用するための適切なハードウェア記述言語はＪＨＤＬであり、たとえば、Ｐｅｔｅｒ Ｂｅｌｌｏｗｓおよび Ｂｒａｄ Ｈｕｔｃｈｉｎｇｓによる１９９８年４月の「ＪＨＤＬ－An HDL for Reconfigurable Systems」Ｐｒｏｃｅｅｄｉｎｇｓ ｏｆ ｔｈｅ ＩＥＥＥ Ｓｙｍｐｏｓｉｕｍ ｏｎ Ｆｉｅｌｄ－Ｐｒｏｇｒａｍｍａｂｌｅ Ｃｕｓｔｏｍ Ｃｏｍｐｕｔｉｎｇ Ｍａｃｈｉｎｅｓに記載されている。

【０１０５】

好ましい代替手法は、特定のツールチェーンがこの計算アーキテクチャのために使用されることである。このようなツールチェーンおよびその実際の動作のエレメントを次に簡単に説明する。

【０１０６】

ツールチェーンは、従来の順次コードから、有効な動作に特に適合されたコードへ変換すること、およびシステム構成エレメントの相互運用の機能を有する。例示的ツールチェーンは入力としてＣコードを受信し、出力として次のものを提供する。すなわち、1)計算の実行のためのＣＨＥＳＳコプロセッサ構成、2)データをシステム・メモリとバースト・バッファ・メモリの間で移動するためのバースト・バッファ・スケジュール、および3)データをＣＨＥＳＳコプロセッサとバースト・バッファ・メモリの間で移動するためのコプロセッサ・コントローラ構成である。

【０１０７】

ツールチェーン自体は２つの構成エレメントを有する。第１のものはフロントエンドであり、これはＣコードをその入力として取り、注釈付きのディペンデンスグラフをその出力として提供する。第２の構成エレメントはバックエンドであり、これはフロントエンドによって生成されたディペンデンスグラフを取り、これらからＣＨＥＳＳ構成、バースト・バッファ・スケジュール、およびコプロセッサ・コントローラ構成を生成する。

【０１０８】

フロントエンドの主なタスクは、計算を、それがコプロセッサ２において起こるように、適切に記述するグラフを生成することである。実行された主なステップの１つは、値に基づいた依存解析であり、これはＷ．ＰｕｇｈおよびＤ．Ｗｏｎｎａｃｏｔｔによる１９９３年１２月の「An Exact Method for Analysis of Value－based Array Data Dependences」, University of Maryland, Institute for Advanced Computer Studies－Dept. of Computer Science, University of Marylandに記載されている。生成された出力は、ＣＨＥＳＳアレイにおいて実施されるデータフローの記述、および（ロードバースト命令を介して）入力としてロードされるか、あるいは（ストアバースト命令を介して）出力として格納される必要のあるすべてのアドレスの表現、およびデータがメイン・メモリ３から検索され、それへ格納されなければならない順序の表現である。これは、バースト・バッファ・コントローラ７のための効率的なスケジュールが導出される基礎である。

【０１０９】

一例として、４タップＦＩＲフィルタのためのＣコードを仮定する場合、次の表のようになる。

【０１１０】

【表６】

```
int i, j, src[], kernel [] , dst [] ;
for (i=0 ; i<1000; i++)
  for (j=0 ; j<4; j++)
```

dst［i］＝dst［i］＋src［4+i-j］*kernel［j］；

【０１１１】

テキスト・ファイルとして提供された、フロントエンド、出力への入力として、次の形式を有する。

【０１１２】

【表７】

```
loop:0<=i<999 #loop nest description
loop:0<=j<4
16:str/0/0/20/ #store instruction
LOD:
#Array:d[1/0/0] at line 11
20:ldc/16/0/0/ #load constant
22:str/0/0/26/ #store instruction, which
LOD: 4 <= j #writes its outputs to main
#Array:d[1/0/0] at line 13 #memory if 4<=j
26:add/22/27/31/ #addition
27:lod/26/0/0/ #load instruction, taking its inputs
Dep(16): [0][0] / Range: j <= 0 #from instruction 16 if j<=0
Dep(22): [0][1] / Range: 1 <= j #from instruction 22 otherwise
LID:
#Array:d[1/0/0] at line 13
31:mul/26/32/37/ #multiplication
32:lod/31/0/0/ #load instruction
Dep(32): [1][1] / Range: 1 <= i && 1 <= j
LID: i <= 0 || j <= 0 && 1 <= i #which takes its inputs from main
#Array:src[1/-1/0] at line 13 #memory if i <= 0 || j <= 0 && 1 <= i
37:lod/31/0/0/
Dep(37): [1][0] / Range: 1 <= i #load instruction
LID: i <= 0 #taking its inputs from main memory if
#Array:kernel[0/1/0] at line 13 #i<=0
```

【０１１３】

このテキスト・ファイルは注釈付きのグラフの表現である。グラフ自体が、図７に示される。このグラフは、フロントエンド・アルゴリズムによって判明された依存性を明瞭に示す。エッジ８１が、依存性が存在する条件、および適用可能である場合の依存性の距離によりマーク付けされる。記述は、必要とされる機能性を有するハードウェア構成エレメントを生成するに必要なすべての情報を含む。

【０１１４】

コンパイル・ツールチェーンのバックエンドは、ある基本機能を有する。１つは、フロントエンドから得られた、拡張されたディペンデンスグラフをスケジュールし、時間変更することである。これは、十分に機能的なＣＨＥＳＳ構成を得るために必要である。スケジューリングは、拡張されたディペンデンスグラフにおけるノード８２のそれぞれが活動化される時点を決定することを含み、時間変更は、たとえば、エッジが値を適切な瞬間に伝播するようにするための遅延の挿入を含む。スケジューリングは、シフト線形スケジューリングを使用して実行することができ、これはハードウェア合成において幅広く使用される技術である。時間変更は、ハードウェア合成における共通で静かな直接のタスクであ

り、適切な数のレジスタを回路に追加して、回路における異なる経路が適切な時点で交わるようにすることのみを含む。この点で、コプロセッサ２（ここでは、ＣＨＥＳＳコプロセッサ）の機能性の完全な記述を有する。この記述が図８に示される。次いで、この記述を適切なツールへ渡して、この機能性を有するＣＨＥＳＳコプロセッサをプログラムするために必要な一連の信号（一般に「ビットストリーム」と呼ばれる）を生成することができる。

【０１１５】

バックエンドの必要とされる別の機能は、バースト・バッファおよびコプロセッサ・コントローラ・スケジュールの生成である。ＣＨＥＳＳ構成が得られた後、これにメイン・メモリからの値を入れる必要があるとき、および値をメイン・メモリへ戻すように格納でき、バースト・バッファ・スケジュールを確立できるときが明らかである。したがって、バースト・バッファ・メモリ５にロードされ、そこから格納される必要のあるすべてのデータのアドレス空間を、バースト・バッファ・コントローラ７が動作できる固定のデータのバーストに分割することを含む、ステップが提供される。

【０１１６】

たとえば、上に提示されたＦＩＲの例では、入力アレイ（ｓｒｃ［］）が適切なサイズのいくつかのバーストに分割され、アルゴリズムのために必要とされたすべてのアドレス範囲が包含されるようになる。このツールチェーンは長さB_{len}のバーストを使用して（B_{len}は２の累乗であり、このツールチェーンへの実行パラメータとして指定される）、できるだけ多くの入力アドレス空間を包含する。このバースト長でそれ以上達成できないとき、ツールチェーンは、長さを減分するバースト、すなわち、$B_{len}/2$、$B_{len}/4$、$B_{len}/8$、．．．、２、１を、このアルゴリズムのために必要とされたあらゆる入力アドレスが唯一のバーストに属するまで使用する。

【０１１７】

これらのバーストのそれぞれについて、ロードされたデータのいずれかが必要とされる、反復空間におけるもっとも早い点が計算される。すなわち、各入力バーストに対して、反復空間において関連付けられた１点があり、そこでは、それより早い反復が、バーストによってロードされたデータのいずれをも必要としないことが保証される。コプロセッサ２の実行が反復空間におけるこの点に到達するときを検出することは容易である。したがって、次のものが作成される。すなわち、1)データをバースト・バッファ・メモリ５に移動するための、関連アドレスのためのロードバースト命令、および2)コプロセッサ２の実行が関連ロードバースト命令と同期化されることを保証するための、対応する同期化点（ＣＣ＿ＬＸ＿ＤＥＣＲＥＭＥＮＴ／ＢＢ＿ＬＸ＿ＩＮＣＲＥＭＥＮＴの組）である。

【０１１８】

計算および通信の効果的なオーバーラップを達成するには、バスを介したデータの転送に関連付けられた待ち時間を隠すために、ロードバースト命令が予め発行されなければならない。

【０１１９】

アルゴリズムによって包含されなければならないすべての出力アドレス空間が、類似のロジックに従って出力バーストに区分される。再度、出力空間が、可変長のバーストに区分される。

【０１２０】

ツールチェーンは、1)関連アドレスのためのストアバースト命令、および2)対応する同期化点（ＢＢ＿ＸＳ＿ＤＥＣＲＥＭＥＮＴ／ＣＣ＿ＸＳ＿ＩＮＣＲＥＭＥＮＴの組）を作成する。

【０１２１】

この点で、我々は、次のことに関連する情報を有する。すなわち、1)ロードバーストおよびストアバースト命令の相対的順序、およびそれらの実行のパラメータ（アドレスなど）、および2)コプロセッサ２上で実行される計算に相対的なそれらの位置である。

【０１２２】

次いで、この情報が使用されて、上述のＦＩＲの例のように、全体の計算を編成するための適切なＣコードが生成される。

【０１２３】

実際のコード生成段階（つまり、プロセッサ１上で実行するためのＣコードの排出）を、ｈｔｔｐ：／／ｗｗｗ．ｃｓ．ｕｍｄ．ｅｄｕ／ｐｒｏｊｅｃｔｓ／ｏｍｅｇａ／で入手可能である、Ｏｍｅｇａ　Ｌｉｂｒａｒｙ　ｏｆ　ｔｈｅ　Ｕｎｉｖｅｒｓｉｔｙ　ｏｆ　Ｍａｒｙｌａｎｄに含まれたコード生成ルーチンと、その後に続いて、これらのルーチンの総称出力を上述の形式に変換する、カスタマイズされたスクリプトを使用して実施することができる。

【０１２４】

実験結果－画像たたみこみ

画像たたみこみアルゴリズムが、次の表のループ・ネストによって記述される。

【０１２５】

【表８】

```
for（i=0；i＜ＩＭＡＧＥ＿ＨＥＩＧＨＴ；i++）
for（j=0；j＜ＩＭＡＧＥ＿ＷＩＤＴＨ；j++）
for（k=0；k＜ＫＥＲＮＥＬ＿ＨＥＩＧＨＴ；k++）
for（l=0；1＜ＫＥＲＮＥＬ＿ＷＩＤＴＨ；l++）
Dest［i，j］＋＝Source［（i+1）-k，（j+1）-l］*C［k，l］；
```

【０１２６】

境界条件を簡約するために、ソース画像を、垂直方向においてＫＥＲＮＥＬ＿ＨＥＩＧＨＴ－１画素、水平方向においてＫＥＲＮＥＬ＿ＷＩＤＴＨ－１画素だけ拡張するために、複製が使用された。２つのカーネル、すなわち共にメディアン・フィルタ（median filtering）を実行する３ｘ３カーネルおよび５ｘ５カーネルが、システム性能を評価することに使用される。

【０１２７】

図９および図１０は、（ＢＢＣとして示す）本発明の一実施形態によるアーキテクチャの性能を、バースト・バッファ（ＢＢとして示す）を使用した従来のプロセッサ、および従来のプロセッサおよびキャッシュの組み合わせ（キャッシュとして示す）に対するものとして例示する。２つのバージョンのアルゴリズムが実施され、１つは３２ビットの画素により、１つは８ビットの画素による。同じ実験測定値が、異なる画像サイズで、８ｘ８から１２８ｘ１２８までの範囲で、異なるバースト長でとられた。

【０１２８】

図から分かるように、ＢＢＣ実装が、ＢＢおよびキャッシュ実装に勝る高い性能上の利点を示した。このアルゴリズムは相対的に複雑であり、ＢＢおよびキャッシュ実装におけるシステムの性能全体が非常にコンピュータ制約的(computer bound)であり、アルゴリズムが高度に複雑であるためにＣＰＵがついていけない。本発明の実施形態を使用すると、計算がＣＨＥＳＳアレイ上で（その固有の並行性により）実行されるので計算が大いに効果的であり、性能はあえていうとＩＯ制約的(I0 bound)である。最も、ＩＯもバースト・バッファの有効な使用を通じて効率的である。マルチメディア命令（ＭＩＰＳ　ＭＤＭＸなど）がＢＢまたはキャッシュ実施におけるＣＰＵの性能を向上させることができ、これは、それらがいくつかの算術命令の並列実行を可能にできるからである。それでもなお、性能向上の結果は、この構成において専用コプロセッサを使用して得られた性能レベルに到達する可能性が低い。

【０１２９】

修正および変形形態

プロセッサ１をコプロセッサ２およびバースト・バッファ・メモリ５から切り離す機能を、命令待ち行列６、８以外によって達成することができる。有効な代替物は、２つの待ち行列を、図１２に記載されたような、命令をバースト・バッファ・メモリ５およびコプロセッサ２へ発行することに完全に専用にされた２つの小型プロセッサ（それぞれが各待

ち行列用）と置換することである。バースト命令待ち行列が（図１の実施形態を参照して）バースト・コマンド・プロセッサ１０６によって置換され、コプロセッサ命令待ち行列が、コプロセッサ・コマンド・プロセッサ１０８によって置換される。これは、これらの２つの構成エレメントによって実行された唯一のタスクとなるので、これらがコプロセッサ２およびバースト・バッファ７からそれぞれ切り離される必要はなくなる。コマンド・プロセッサ１０６、１０８のそれぞれが、コマンドをコプロセッサまたはバースト・バッファ（適切なように）へ発行することによって動作でき、次いで、そのコマンドがその実行を完了するまで何も行わず、別のコマンドを発行することなどができる。これは、設計を複雑にするが、メイン・プロセッサ１をその残りの、命令を待ち行列へ発行する単純なタスクから解放する。プロセッサ１によって実行される唯一の作業は、次いで、これらの２つのプロセッサの初期設定となり、これは計算の開始直前に行われる。したがって、計算中に、プロセッサ１が完全にコプロセッサ２およびバースト・バッファ・メモリ５の実行から切り離される。

【０１３０】

２つの従来の、しかしより小型のマイクロプロセッサ（あるいは、別法として、２つの制御の独立スレッドを実行する唯一のプロセッサ）を使用することができ、それぞれが適切なコード（ループ・ネスト）の関連部分を実行する。別法として、外部挙動がコードの関連部分の実行を反映する、２つの汎用状態機械を合成することができる（つまり、これらが同じ命令のシーケンスを提供する）。このような状態機械のハードウェアの複雑さおよびコストは、同等の専用プロセッサのものよりも著しく低くなる。このような状態機械が、メイン・プロセッサ１によって、上述のものに類似した方法でプログラムされる。主な違いは、イベントの繰り返しも符号化されることである。すなわち、これは、プロセッサ１が、多少の（複雑である場合は）命令における１つのアルゴリズムの挙動を符号化できるために必要である。イベントのｘ回の繰り返しを得るために、プロセッサ１がｘ個の命令を待ち行列に挿入する必要はないが、この繰り返しパラメータを命令定義において符号化しなければならない。

【０１３１】

上に示したように、特に有効な機構は、有限状態機械（ＦＳＭ：Finite StateMachine）を待ち行列の代りに使用して、メイン・プロセッサ１の実行をコプロセッサ２およびバースト・バッファ・コントローラ７の実行から切り離すことである。この機構が以下でより詳細に論じられる。

【０１３２】

図１に例示されたアーキテクチャでは、異なるＩ／Ｏストリームの実行を駆動するための命令を、コプロセッサ２の実行のための命令と混合することができる。これは、システム構成エレメントの間の相互関係がコンパイル時に知られており、したがって、異なるシステム構成エレメントへの命令を正しい順序でソース・コードにおいてインタリーブすることができるので、可能である。

【０１３３】

２つの状態機械を、まったく同じ方法による実行のためのこれらの命令を発行するように、構築することができる。このような状態機械の一方は、コプロセッサ２の挙動を制御し、必要とされるようなＣＣ＿ｘｘｘ＿ｘｘｘ命令を発行し、他方はバースト・バッファ・コントローラ７の挙動を制御し、必要とされるようなＢＢ＿ｘｘｘ＿ｘｘｘ命令を発行する。

【０１３４】

このような状態機械を、いくつかの異なる方法において実施することができる。１つの代替手法が図１３に示される。上で提示されたベクトル加算の例を参照して、この状態機械１５０（コプロセッサ２のためのものであるが、バースト・バッファ・コントローラ７のための同等のマシンが直接類似している）が、以下のパターンから構築された命令のシーケンスを実施する。

【０１３５】

【表９】
ＣＣ＿ＬＸ＿ＤＥＣＲＥＭＥＮＴ、
ＣＣ＿ＬＸ＿ＤＥＣＲＥＭＥＮＴ、
ＣＣ＿ＳＴＡＲＴ＿ＥＸＥＣ、
ＣＣ＿ＸＳ＿ＩＮＣＲＥＭＥＮＴ
【０１３６】
メインの状態機械１５０が効果的に、より簡素な状態機械１５１、１５２、１５３に分割され、それぞれが１種類の命令の実行を制御する。周期および段階（これが、コプロセッサ２とバースト・バッファ・コントローラ７の間で通信するＩ／Ｏストリームに関連付けることができる周期および段階とは、何の関係も有していないことに留意されたい）が、より簡素な状態機械のそれぞれに関連付けられる。状態機械１５０のハードウェアは、典型的には、意図されたアプリケーションの要件を満たすために十分な数の、このようなより簡素な状態機械のアレイを含む。
【０１３７】
イベント・カウンタ１５４が定義される。イベント・カウンタ１５４の役割は、（この場合、コプロセッサ２のための）命令が順番に送信されることを可能にすることである。イベント・カウンタ１５４が増分される毎に、Ｍ＊周期$_i$＋段階$_i$＝イベント・カウンタの値であるような値Ｍが存在する場合、状態機械ｉ（すなわち、より簡素な状態機械１５１、１５２、１５３の１つ）が、比較ロジック１５５を通じた実行のために選択され、その命令が実行される。アプリケーション・ソフトウェアの責任は、２つの異なる状態機械がこの式を満たすことができないようにすることである。この命令の実行が完了されるとき、イベント・カウンタ１５４が再度増分される。このイベントのシーケンスは、次のように要約することができる。
【０１３８】
１：イベントカウンタを増分する、すなわち、ＥＣ＋＋
２：Ｍ＊周期$_i$＋段階$_i$＝ＥＣであるようなＭが存在する場合、状態機械ｉを実行のために選択する
３：このような状態機械ｉが発見された場合、状態機械ｉによって記述された命令を実行する（これは、中断動作を含むことができる）
４：１へ戻る
【０１３９】
命令の実行に関係する多少の余分なパラメータ（読み／書きするためのアドレス、ＣＣ＿ＳＴＡＲＴ＿ＥＸＥＣのための実行の長さなど）が、状態機械１５０において符号化されなければならない。複数の状態機械が、典型的には異なるパラメータを有する所与の命令を発行することができることに留意されたい。
【０１４０】
このシステムは、周期的挙動を生成するために特に十分に動作する。しかし、イベントが一度だけ起こらなければならない場合、これは容易に、無限周期および有限段階を有する簡素な状態機械において符号化することができ、唯一の帰結は、この簡素な状態機械が一度だけ使用されることである。
【０１４１】
この手法はそれ自体を変更することができる。たとえば、この機構に柔軟性を追加するには、可能な選択肢は、１つまたは複数の簡素な状態機械の実行を所定の「時間ウィンドウ」に制限するために、「開始時間」および「終了時間」パラメータを簡素な状態機械に追加することである。
【０１４２】
これらの状態機械のプログラミングは、システムの初期化中に、たとえば、プロセッサ１によって割り当てられたメモリマップ・レジスタの使用を通じて起こる。代替物は、これらの状態機械をメイン・メモリ３の事前定義の領域から、おそらくは専用チャネルおよび直接メモリ・アクセス（ＤＭＡ）機構の使用を通じて、プログラムするために必要なす

べてのパラメータのローディングとなる。

【０１４３】

２つの専用マイクロプロセッサを使用する、提案された他の代替機構は、図１のアーキテクチャのためのプログラミング・モデルに著しい修正を必要としない。すなわち、メイン・プロセッサ１をプログラムするために使用された同じ技術を、コプロセッサ２のために意図されたコマンドをバースト・バッファ・コントローラ７のために意図されたものから分割する追加ステップと共に、使用することができる。実現可能であるが、この構成は、状態機械の手法に関して不利である可能性がある。これらのプロセッサに、システムの複雑さに加えて、メイン・メモリ３または他のＤＲＡＭへのアクセスを提供することが必要となる。システムのコストおよび複雑さは、２つのマイクロプロセッサをこのように追加すること（およびそれらが非常に簡素な計算を実行するためにのみ存在することにおいて、十分に利用しないこと）によっても増大される。

【０１４４】

図１およびその代替物のアーキテクチャを越えた様々な開発も、本発明の本質の原理から逸れることなく行うことができる。３つのこれらの開発の分野が以下に記載される。すなわち、パイプライン、データ依存条件／知られていない実行時間、およびメモリへの非アフィン・アクセスである。

【０１４５】

パイプライン・アーキテクチャは、アプリケーションがそれらの入力データ・ストリーム上で複数の変換が実行されることを必要とする値を有する。たとえば、たたみこみの直後に続いて相関を行うことができる。この種類の構成を収容するために、アーキテクチャおよび計算モデルへの変更が必要となる。アーキテクチャ上では、逐次的にバッファされたＣＨＥＳＳアレイ、またはより大きい区分されたＣＨＥＳＳアレイ、または計算段階の間に再構成されたＣＨＥＳＳアレイを提供することができる。図１１Ａおよび図１１Ｂは、このようなアプリケーションを処理するために有効であり、複数のＣＥＥＳＳアレイを含む、異なるパイプライン・アーキテクチャを示す。図１１Ａは、プロセッサ１４３から命令された互い違いのＣＨＥＳＳ／バースト・バッファ・パイプラインによる構成、およびメイン・メモリ１４４とのデータの交換を示し、ＣＨＥＳＳアレイ１４１がデータを第１の組のバースト・バッファ１４２から受信して、これを第２の組のバースト・バッファ１４５へ渡し、この第２の組のバースト・バッファ１４５がさらにＣＨＥＳＳアレイ１４６と対話する（潜在的には、このパイプラインを、さらなる組のＣＨＥＳＳアレイおよびバースト・バッファにより継続させることができる）。同期化はより複雑になり、近接したＣＨＥＳＳアレイの間、および近接したバースト・バッファの組の間の通信を含むが、同じ汎用パイプラインを後に続けて、バースト・バッファの効率的な使用、およびＣＨＥＳＳアレイの間の効率的な同期化を可能にすることができる。すなわち、セマフォを使用して、パイプラインの逐次段階によって実行された計算の正確さを保証することができる。

【０１４６】

図１１Ｂは、異なるタイプの計算パイプラインを示し、２つのＣＨＥＳＳアレイ１５１、１５６の間にＳＲＡＭキャッシュ１５５を有し、第１の組のバースト・バッファ１５２へ提供されたロード、および第２の組のバースト・バッファ１５７によって提供された格納を有する。プロセッサ１５３およびメイン・メモリ１５４の役割は、本質的に他の実施形態から不変である。同期化はこの構成においてそれほど困難でない可能性があるが、この構成が並行性をそれほど効果的でなく活用する可能性がある。

【０１４７】

上述のようなアーキテクチャにおけるコプロセッサの効率的な使用上の１つの制約は、コプロセッサ実施の実行時間が知られるべきであることである（効率的なスケジューリングを可能にするため）。これは、多数の媒体処理ループについて達成可能である。しかし、実行時間がコンパイル時に知られていない場合、ツールチェーンにおけるスケジューリング要件を緩和させる必要があり、プロセッサ、コプロセッサおよびバースト・バッファ

の間の同期化および通信プロトコルにおいて、適切な許容が行われる必要がある。コプロセッサ・コントローラは、この状況のための特定の構成も必要とする。

【０１４８】

別の拡張は、バースト・バッファ・メモリへの非アフィン参照を可能にすることである。上で使用されたバースト・バッファ・モデルでは、すべてのアクセスがＡＩ＋Ｆのタイプであり、ただしＡは定数行列、Ｉは反復ベクトル、Ｆは定数ベクトルである。この制限されたアクセス・モデルの使用により、コプロセッサ・コントローラおよびプロセッサが予め、どのデータがいずれかの所与の時点で必要とされるかを知ることができ、論理ストリームの効率的な作成が可能となる。このアーキテクチャへのこの有意性は、全体として、どのように非アフィン・アクセスを完全に任意の方法で提供することができるかが不明瞭である（同期化機構が失敗するように思われる）が、非アフィン・アレイ・アクセスを使用してルックアップ・テーブルを参照することが可能になるということである。これは、ルックアップ・テーブルをバースト・バッファにロードすることによって行うことができ、次いで、コプロセッサが、後続アクセスのためのルックアップ・テーブルの開始に相対的なバースト・バッファ・アドレスを生成することができる。このようなアドレスを、それらが使用されるときより十分に前もって生成できるようにすること（場合によっては、これを、同期化機構への精練化によって達成することができる）、およびこのタイプの回帰参照を支持するように論理ストリーム機構を修正することが必要となる。

【０１４９】

したがって、図１のアーキテクチャへの多数の変形形態および拡張を、特許請求の範囲に記載されたような本発明から逸脱することなく、実行することができる。

【図面の簡単な説明】

【図１】本発明の第１の実施形態によるシステムの基本エレメントを示す図。

【図２】図１のシステムにおいて使用されたバースト・バッファ構造のアーキテクチャを示す図である。

【図３】図２のバースト・バッファ構造のさらなる機能を示す図である。

【図４】図１のシステムにおいて使用されたコプロセッサ・コントローラの構造、および他のシステム構成エレメントとの関係を示す図である。

【図５】図１のシステム上で使用可能な計算モデルを例示するための一例を示す図である。

【図６】図５の例のための計算およびＩ／Ｏ動作のための時間線を示す図。

【図７】図１のシステムのためのコードを提供するために有用なツールチェーンのフロントエンドからの出力として提供された、注釈付きグラフを示す図。

【図８】図７における仕様から導出された、コプロセッサの内部構成を示す図。

【図９】３２ビット・ピクセルを使用する５ｘ５画像たたみこみのための代替アーキテクチャの性能を示す図である。

【図１０】８ビット・ピクセルを使用する５ｘ５画像たたみこみのための、図９を生成するために使用された代替アーキテクチャの性能を示す図である。

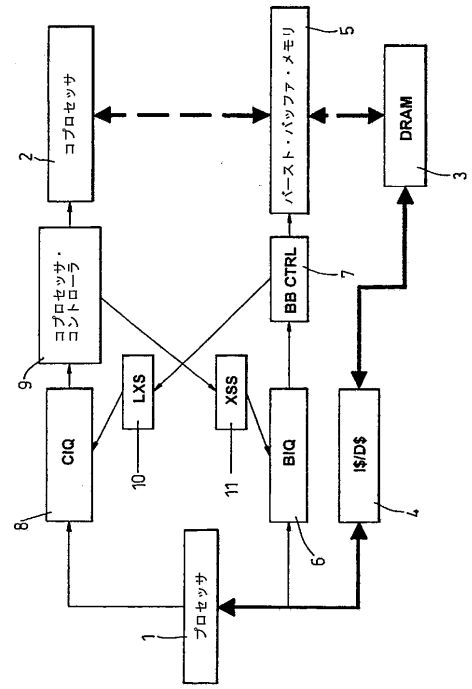
【図１１Ａ】本発明のさらなる実施形態を使用する代替パイプライン・アーキテクチャを示す図である。

【図１１Ｂ】本発明のさらなる実施形態を使用する代替パイプライン・アーキテクチャを示す図である。

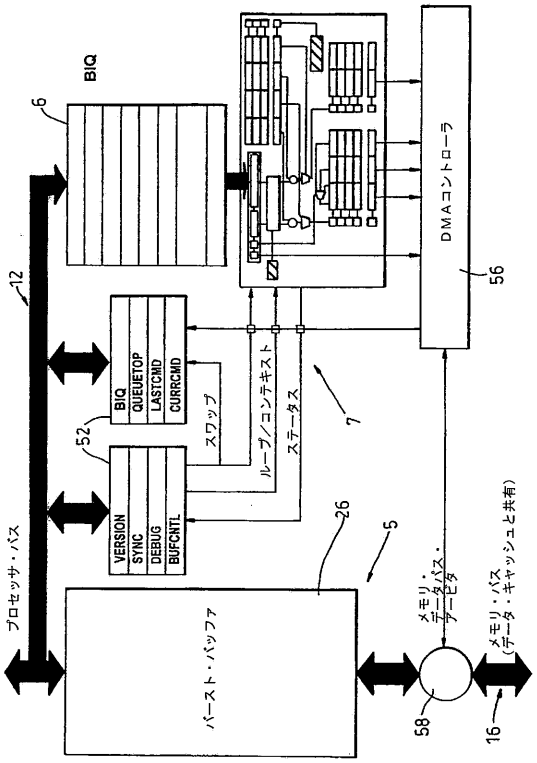
【図１２】図１のアーキテクチャにおける、コプロセッサ命令待ち行列およびバースト命令待ち行列の代替として使用可能な、２つの補助プロセッサを示す図。

【図１３】図１のアーキテクチャにおけるコプロセッサ命令待ち行列の代替としての状態機械の実施を示す図である。

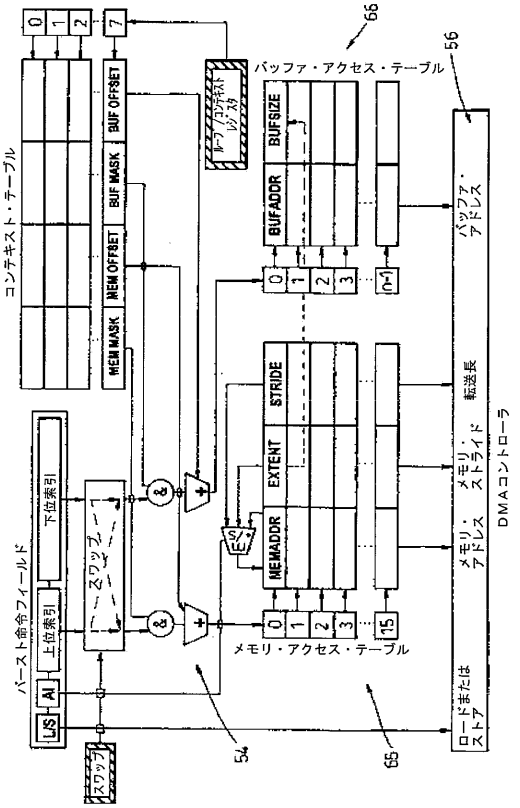
【図１】


コプロセッサ
バースト・バッファ・メモリ
DRAM
コプロセッサ・コントローラ
BB CTRL
CIQ
LXS
XSS
BIQ
I$/D$
プロセッサ

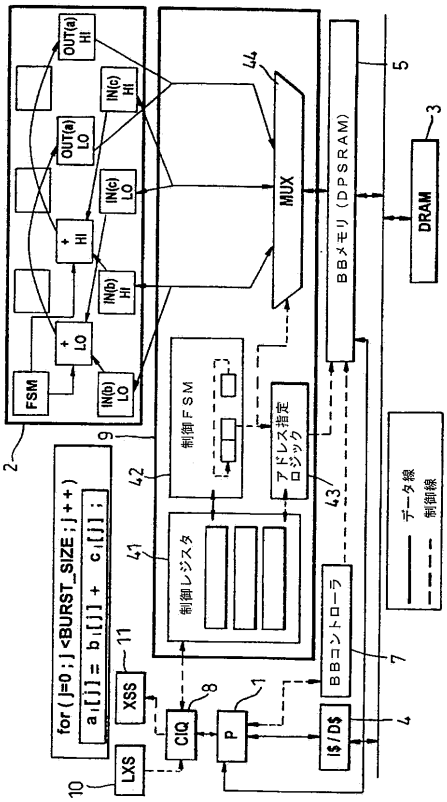
【図２】


BIQ
DMAコントローラ
BIQ
QUEUETOP
LASTCMD
CURRCMD
スワップ
ループ/コンテキスト
ステータス
VERSION
SYNC
DEBUG
BUFCNTL
プロセッサ・バス
バースト・バッファ
メモリ・バス
データ・キャッシュと共有)

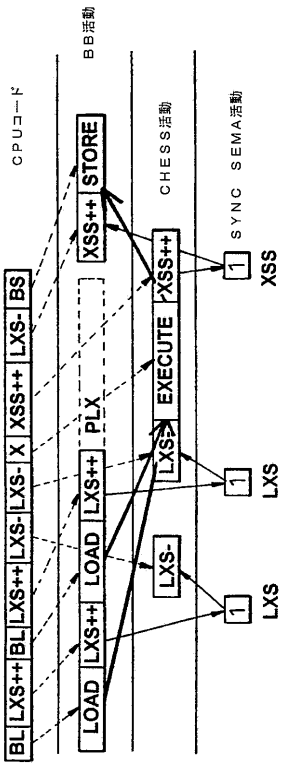
【図３】


コンテキスト・テーブル
バッファ・アクセス・テーブル
BUF OFFSET
BUF MASK
MEM OFFSET
MEM MASK
BUFSIZE
BUFADDR
STRIDE
EXTENT
MEMADDR
バースト命令フィールド
下位索引
上位索引
スワップ
メモリ・アクセス・テーブル
メモリ・アドレス
メモリ・ストライド
転送長
バッファ・アドレス
ロードまたはストア
DMAコントローラ

【図４】


FSM
MUX
BBメモリ（DPSRAM）
DRAM
制御FSM
制御レジスタ
アドレス指定ロジック
for (j=0 ; j <BURST_SIZE ; j++)
XSS
LXS
CIQ
P
BBコントローラ
I$/D$
データ線
制御線

【図５】



【図６】

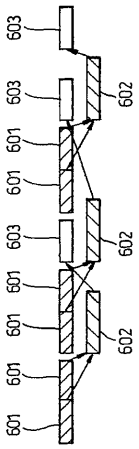
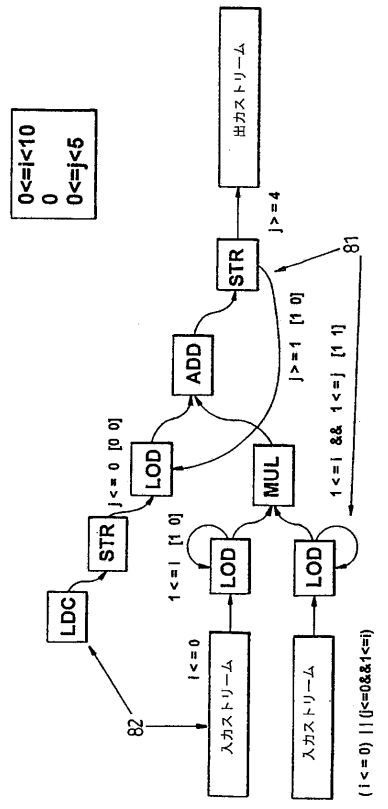
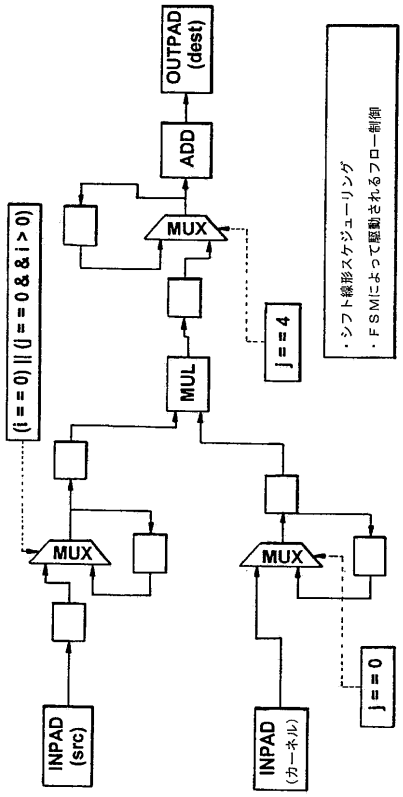


Fig. 6

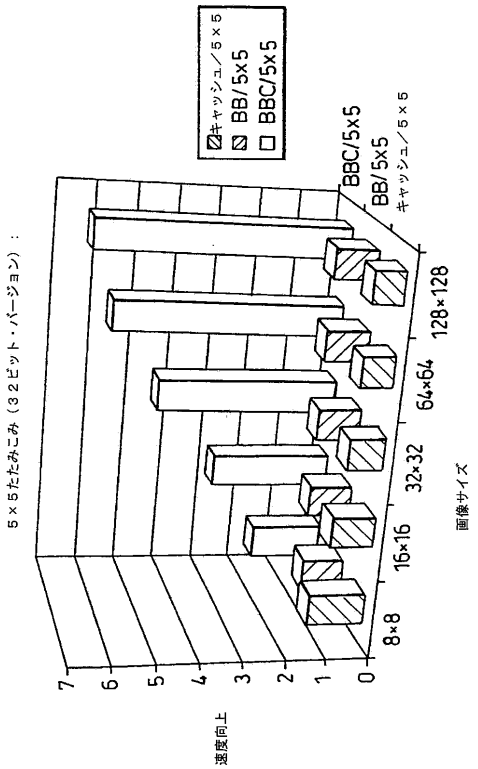
【図７】



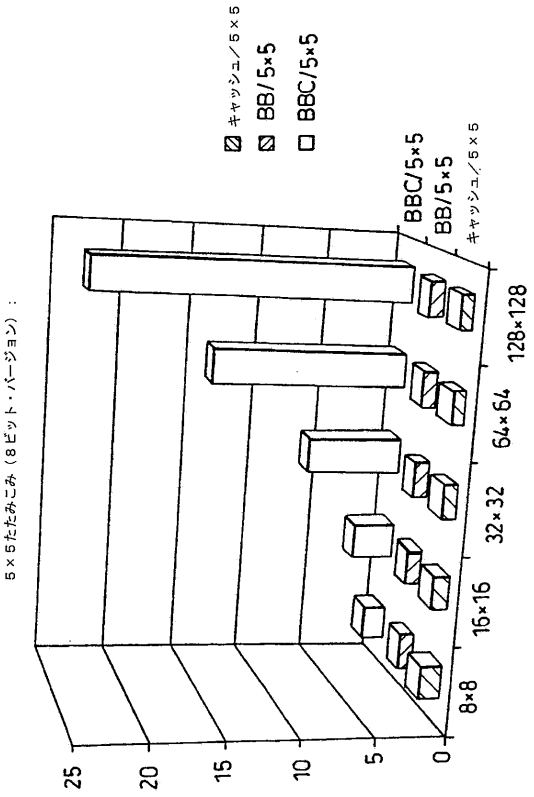
【図８】



【図９】



【図１０】



【図１１Ａ】

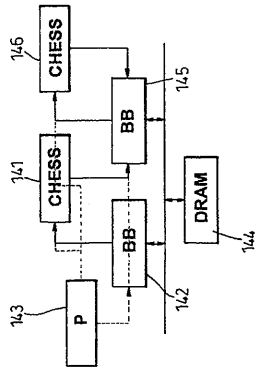


Fig. 11A

【図１１Ｂ】

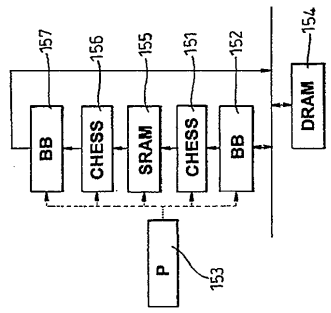
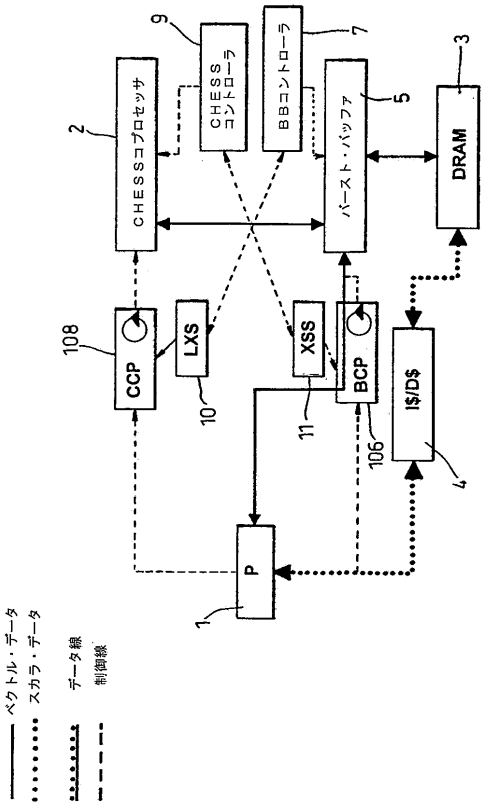
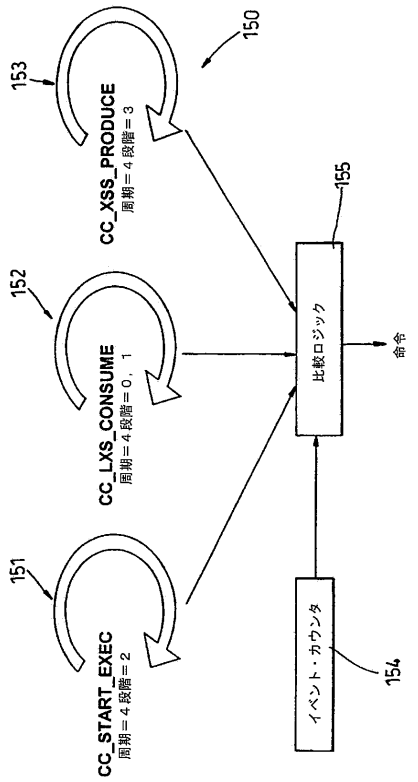


Fig. 11B

【図１２】



【図１３】


150
153
CC_XSS_PRODUCE
周期＝４段階＝３
152
CC_LXS_CONSUME
周期＝４段階＝０，１
151
CC_START_EXEC
周期＝４段階＝２
155
比較ロジック
命令
154
イベント・カウンタ

フロントページの続き

(72)発明者　マッカーシー・ドミニク・ポール
イギリス、ビーエイチ６、５ディーキュー、ボーンマス、サウスボーン、アメウッド・ロード　１９

合議体
審判長　金子　幸一
審判官　田中　秀人
審判官　長島　孝志

(56)参考文献　特開平１０－２３２８２６（ＪＰ，Ａ）
特開平１１－２５１４４２（ＪＰ，Ａ）
特開昭５６－４８６２（ＪＰ，Ａ）
特開平７－２８１８３６（ＪＰ，Ａ）
特開平１１－６５９８９（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F9/38