(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4564110号
(P4564110)
(45)発行日　平成22年10月20日(2010.10.20)　(24)登録日　平成22年8月6日(2010.8.6)

(51)Int.Cl.　　F I
G06F 11/26 (2006.01)　　G06F 11/26
G06F 11/28 (2006.01)　　G06F 11/28　340C

請求項の数 9　(全 24 頁)

(21)出願番号　特願平9-284063
(22)出願日　平成9年10月16日(1997.10.16)
(65)公開番号　特開平10-228393
(43)公開日　平成10年8月25日(1998.8.25)
審査請求日　平成14年7月2日(2002.7.2)
審判番号　不服2007-21762(P2007-21762/J1)
審判請求日　平成19年8月6日(2007.8.6)
(31)優先権主張番号　08/730866
(32)優先日　平成8年10月16日(1996.10.16)
(33)優先権主張国　米国(US)

(73)特許権者　390019839
三星電子株式会社
ＳＡＭＳＵＮＧ　ＥＬＥＣＴＲＯＮＩＣＳ　ＣＯ．，ＬＴＤ．
大韓民国京畿道水原市靈通区梅灘洞４１６
４１６，Ｍａｅｔａｎ－ｄｏｎｇ，Ｙｅｏｎｇｔｏｎｇ－ｇｕ，Ｓｕｗｏｎ－ｓｉ，Ｇｙｅｏｎｇｇｉ－ｄｏ　４４２－７４２（ＫＲ）
(74)代理人　100086368
弁理士　萩原　誠
(72)発明者　モアタズ　エー　モハメッド
アメリカ合衆国　カリフォルニア州　９５０５０　サンタクララ　ワーバートン　アベニュー７番　１７２１

最終頁に続く

(54)【発明の名称】二重プロセッサ回路の動作をシミュレーションするためのコンピュータ実行方法及び信号プロセッサシミュレータ

(57)【特許請求の範囲】
【請求項１】
二重プロセッサ回路上の並列実行をシミュレーションするための信号プロセッサシミュレータであって、
二重プロセッサの第１プロセッサ上の命令の実行をシミュレーションするための第１プロセッサシミュレータと、
前記二重プロセッサの第２プロセッサ上の命令の実行をシミュレーションするための第２プロセッサシミュレータと、
前記第１プロセッサシミュレータと前記第２プロセッサシミュレータとによりアクセスされる値を格納するための共有されるメモリと、
前記第１プロセッサに関する組み合わされた実行プログラムの命令の実行に要するクロックサイクル時間をシミュレーションするための第１クロックシミュレータと、
前記第２プロセッサに関する組み合わされた実行プログラムの命令の実行に要するクロックサイクル時間をシミュレーションするための第２クロックシミュレータと、
前記第１クロックシミュレータと第２クロックシミュレータとの前記クロックサイクル時間を比較し、アイドル状態と見なすに足る遅れを示すクロックシミュレータに該当するプロセッサシミュレータ上の命令の処理を実行させる同期装置と、
を備え、
前記同期装置は、並列処理のシミュレートを開始するため、前記第２プロセッサシミュレータを先に活性化させることにより、前記第２プロセッサシミュレータが前記命令を用

意すると、前記第１クロックシミュレータと前記第２クロックシミュレータとのクロックを初期化し、前記第１プロセッサシミュレータが前記第２プロセッサシミュレータに対しアイドル状態と見なすに足る遅れを示していれば、組み合わされた実行プログラムから前記第１プロセッサ上で実行される次の命令を取り出し前記第１プロセッサシミュレータに実行させて、並列処理をシミュレーションするコンピュータ実行段階を含む第１ループを遂行することを特徴とする信号プロセッサシミュレータ。

【請求項２】

前記第１プロセッサシミュレータは、ＲＩＳＣプロセッサをシミュレーションすることを特徴とする請求項１記載の信号プロセッサシミュレータ。

【請求項３】

前記第１プロセッサシミュレータは、ＡＲＭＵＬＡＴＯＲであることを特徴とする請求項１記載の信号プロセッサシミュレータ。

【請求項４】

前記第２プロセッサシミュレータは、ベクトルプロセッサをシミュレーションすることを特徴とする請求項１記載の信号プロセッサシミュレータ。

【請求項５】

前記共有されるメモリは、前記第１プロセッサシミュレータと第２プロセッサシミュレータとに示されることを特徴とする請求項１乃至４のいずれかに記載の信号プロセッサシミュレータ。

【請求項６】

前記共有されるメモリは、前記第２プロセッサシミュレータにより変更できる前記第１プロセッサシミュレータのレジスタ値を含むことを特徴とする請求項５記載の信号プロセッサシミュレータ。

【請求項７】

前記同期装置は、前記第２プロセッサシミュレータに関する命令に要求されるクロック周期を含む検査表を含み、前記第２プロセッサシミュレータ上の命令を実行する前に、命令に必要なクロック周期を決定するために前記検査表を検査して前記第１プロセッサ上で実行される１つ以上の命令をフェッチし、前記命令と前記フェッチした命令との実行を開始し、前記第１プロセッサシミュレータによりシミュレーションされる第１プロセッサ動作が、少なくとも前記第２プロセッサシミュレータによりシミュレーションされる第２プロセッサの動作期間以内に、前記第１プロセッサシミュレータ上の１個以上の他の命令を実行することを特徴とする請求項１乃至６のいずれかに記載の信号プロセッサシミュレータ。

【請求項８】

二重プロセッサ回路上で実行される命令を含む実行可能なプログラムと、

前記二重プロセッサ回路の第１プロセッサ上の命令実行をシミュレーションするための第１プロセッサシミュレータと、

前記二重プロセッサ回路の第２プロセッサ上の命令実行をシミュレーションするための第２プロセッサシミュレータと、

前記第１プロセッサシミュレータと第２プロセッサシミュレータとの間の命令実行をインターリーブするための同期装置と、

第１プロセッサクロックを示す第１クロック周波数と、

第２プロセッサクロックを示す第２クロック周波数と、

前記第１プロセッサシミュレータと第２プロセッサシミュレータに示されて、前記第１プロセッサシミュレータに要求される一個以上の第１レジスタ値及び前記第２プロセッサシミュレータに要求される一個以上の第２レジスタ値を含む、共有されるメモリと、を含むコンピュータ格納媒体に提供される二重プロセッサ回路の動作をシミュレーションするためのコンピュータ実行方法において、

前記実行可能なプログラムから第１命令をフェッチし、

前記第１命令が並列処理のための信号でなければ、前記第１プロセッサシミュレータ上

の第１命令実行をシミュレーションし、
前記第１命令が並列処理のための信号であれば、並列処理をシミュレーションするコンピュータ実行段階を含む第１ループを遂行し、
前記の並列処理をシミュレーションする段階が、
初期化後の各処理時間を示す前記第１クロック数と第２クロック数とを比較し、
前記第１クロック数が第２クロック数より小さければ、前記の実行可能なプログラムから前記第１プロセッサ上で実行される第２命令をフェッチし、前記第１プロセッサシミュレータ上で前記第２命令の実行をシミュレーションし、前記第１プロセッサ上の前記第２命令の実行に必要なクロック数を前記第１クロック数に加えて、
前記第１クロック数が前記第２クロック数と同一であるかまたは大きければ、前記の実行可能なプログラムから前記第２プロセッサ上で実行される第３命令をフェッチし、前記第２プロセッサシミュレータ上で前記第３命令の実行をシミュレーションし、前記第２プロセッサ上の前記第３命令を遂行するに必要なクロック数を前記第２クロック数に加えて、
前記第２プロセッサシミュレータがこれ以上必要となくなると前記第１ループに戻って、
前記の実行可能なプログラム内の第１プロセッサシミュレータと関連されているすべての命令がシミュレーションされたら、前記第１ループを終了するコンピュータ実行段階を含むことを特徴とする二重プロセッサ回路の動作をシミュレーションするためのコンピュータ実行方法。
【請求項９】
二重プロセッサ回路上で実行される命令を含む実行可能なプログラムと、
前記二重プロセッサ回路の第１プロセッサ上の命令の実行をシミュレーションするための第１プロセッサシミュレータと、
前記二重プロセッサ回路の第２プロセッサ上の命令の実行をシミュレーションするための第２プロセッサシミュレータと、
前記第１プロセッサシミュレートと第２プロセッサシミュレート間の命令の実行をインターリーブするための同期装置と、
第１プロセッサクロックを示す第１クロック周波数と、
第２プロセッサクロックを示す第２クロック周波数と、
命令を実行することに必要な時間を示す検査表と、
前記第１プロセッサシミュレータと第２プロセッサシミュレータで示されて、前記第１プロセッサシミュレータに要求される１個以上の第１レジスタ値及び前記第２プロセッサシミュレータに要求される１個以上の第２レジスタ値を含む、共有されるメモリと、を含むコンピュータ格納媒体に提供される二重プロセッサ回路の動作をシミュレーションするためのコンピュータ実行方法において、
前記の実行可能なプログラムから第１命令をフェッチし、
前記第１命令が、並列プロセッシングのための信号でなければ、前記第１プロセッサシミュレータ上の第１命令の実行をシミュレーションし、
前記第１命令が、並列処理のための信号であれば、並列処理をシミュレーションするコンピュータ実行段階を含む第１ループを遂行し、
前記の並列処理をシミュレーションする段階が、
初期化後の各処理時間を示す前記第１クロック数と第２クロック数とを比較し、
前記第１クロック数が第２クロック数より小さければ、前記の実行可能なプログラムから前記第１プロセッサ上で実行される第２命令をフェッチし、前記第１プロセッサシミュレータ上で前記第２命令の実行をシミュレーションし、前記第１プロセッサ上の前記第２命令を実行することに必要なクロック数を前記第１クロック数に加えて、
前記第１クロック数が前記第２クロック数より同一であるかまたは大きければ、前記の実行可能なプログラムから前記第２プロセッサ上で実行される第３命令をフェッチし、前記第３命令に必要な実行時間を決定するために検査表をアクセスし、前記第２プロセッサ

シミュレータ上で前記第３命令の実行をシミュレーションし、前記第２プロセッサ上の前記第３命令を遂行することに必要なクロック数を前記第２クロック数に加えて、前記第３命令に必要な実行時間を基準とし、前記実行可能なプログラムから所定の命令の実行をシミュレーションし、

前記第２プロセッサシミュレータがこれ以上必要となくなると前記第１ループに戻って、

前記の実行可能なプログラム内の第１プロセッサシミュレータと関連されているすべての命令がシミュレーションされたら、前記第１ループを終了するコンピュータ実行段階を含むことを特徴とする二重プロセッサ回路の動作をシミュレーションするためのコンピュータ実行方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、回路のシミュレーションに係るもので、特に二重プロセッサ（dual processor）回路の動作をシミュレーションするためのコンピュータ実行方法及び信号プロセッサシミュレータに関する。

【０００２】

【従来の技術】

一般的に、電子回路設計と、新しく設計された回路の大量生産との間には多くの時間間隔があるので、大量生産に費用を投資する前に、製造業者は回路設計が適合しているか否かを確認するために回路のテストを行っている。個々の回路段階に回路を製造するには費用が嵩むことから、回路のテストに物理的な制限が伴う。回路の少量生産にかかる費用を削減するために、回路設計者は少ない費用で且つ新しい回路設計の正確なテストを提供するコンピュータシミュレーションに関心を持つようになってきている。

【０００３】

従来、コンピュータシミュレータ間の信号伝達に臨時ファイルを用いて行う方法がある。この方法は、第１のシミュレータが臨時ファイルを割り当て、この臨時ファイルに記入してから割り当てを取り消し、第２のシミュレータは同じ臨時ファイルを割り当て、伝送されるデータを読み出してから当該臨時ファイルの割り当てを取り消すものである。

【０００４】

【発明が解決しようとする課題】

しかしながら、この方法では、第１、第２のシミュレータ間の情報伝達に一連の長い命令を伴うので、シミュレータの性能が低下するという問題点がある。

【０００５】

シミュレータ間の情報伝達のための他の方法として、パイプとバッファの構造を用いたものがあるが、このような構造を処理することもまた多くの費用がかかる。また、伝統的なシミュレータでは、基礎となるハードウェアに存在する情報伝達能力をシミュレーションすることができない。例えば、回路要素は臨時ファイルを通して情報を伝達しない。従って、従来のシミュレーションより性能を向上させた二重プロセッサ回路のシミュレータが要求される。また、基礎となる電子回路に存在する信号伝達動作を正確にシミュレーションするシミュレータも要求される。

【０００６】

そこで本発明は、従来のシミュレータに比して向上された性能を有し、基礎となる電子回路に存在する信号伝達動作を正確にシミュレーションできる二重プロセッサ回路の動作をシミュレーションするためのコンピュータ実行方法及び信号プロセッサシミュレータを提供することを目的としている。

【０００７】

【課題を解決するための手段】

上記課題を解決し、上記目的を達成するために、請求項１記載の発明による信号プロセッサシミュレータは、二重プロセッサ回路上の並列実行をシミュレーションするための信号

プロセッサシミュレータであって、二重プロセッサの第１プロセッサ上の命令の実行をシミュレーションするための第１プロセッサシミュレータと、二重プロセッサの第２プロセッサ上の命令の実行をシミュレーションするための第２プロセッサシミュレータと、信号プロセッサシミュレータによりアクセスされた値を格納するための共有されるメモリと、第１プロセッサに関するクロックをシミュレーションするための第１クロックシミュレータと、第２プロセッサに関するクロックをシミュレーションするための第２クロックシミュレータと、第１クロックシミュレータと第２クロックシミュレータとを比較し、後のクロックシミュレータに該当するプロセッサシミュレータ上の命令を処理する同期装置とを備えるものである。

【０００８】

第１プロセッサシミュレータは、ＲＩＳＣプロセッサをシミュレーションするものであっても良い。

【０００９】

また、第１プロセッサシミュレータは、ＡＲＭＵＬＡＴＯＲとしても良い。

【００１０】

第２プロセッサシミュレータは、ベクトルプロセッサをシミュレーションするものであっても良い。

【００１１】

前記共有されるメモリは、第１プロセッサシミュレータと第２プロセッサシミュレータとに示されるものであっても良い。

【００１２】

前記共有されるメモリは、第２プロセッサシミュレータにより変更できる第１プロセッサシミュレータのレジスタ値を含むものであっても良い。

【００１３】

同期装置は、第２プロセッサシミュレータに関する命令に要求されるクロック周期を含む検査表を含み、第２プロセッサシミュレータ上の命令を実行する前に、命令に必要なクロック周期を決定するために検査表を検査し、命令を実行してから、第１プロセッサシミュレータによりシミュレーションされる第１プロセッサ動作が、少なくとも第２プロセッサシミュレータによりシミュレーションされる第２プロセッサの動作期間程度に遂行される時まで、第１プロセッサシミュレータ上の１個以上の他の命令を実行するものであっても良い。

【００１４】

請求項８記載の発明による二重プロセッサ回路の動作をシミュレーションするためのコンピュータ実行方法は、二重プロセッサ回路上で実行される命令を含む実行可能なプログラムと、二重プロセッサ回路の第１プロセッサ上の命令実行をシミュレーションするための第１プロセッサシミュレータと、二重プロセッサ回路の第２プロセッサ上の命令実行をシミュレーションするための第２プロセッサシミュレータと、第１プロセッサシミュレータと第２プロセッサシミュレータとの間の命令実行をインターリーブするための同期装置と、第１プロセッサクロックを示す第１クロック値と、第２プロセッサクロックを示す第２クロック値と、第１プロセッサシミュレータと第２プロセッサシミュレータに示されて、前記の第１プロセッサシミュレータに要求される一個以上の第１レジスタ値及び第２プロセッサシミュレータに要求される一個以上の第２レジスタ値を含む、共有されるメモリとを含むコンピュータ格納媒体に提供される二重プロセッサ回路の動作をシミュレーションするためのコンピュータ実行方法において、実行可能なプログラムから第１命令をフェッチし、第１命令が並列処理のための信号でなければ、第１プロセッサシミュレータ上の第１命令実行をシミュレーションし、第１命令が並列処理のための信号であれば、並列処理をシミュレーションするコンピュータ実行段階を含む第１ループを遂行し、前記の並列処理をシミュレーションする段階が、クロック値と第２クロックとを比較し、第１クロック値が第２クロック値より小さければ、前記の実行可能なプログラムから第２命令をフェッチし、第１プロセッサシミュレータ上の第２命令の実行をシミュレーションし、第１プロ

セッサ上の第２命令の実行に必要な時間を第１クロック値に加えて、第１クロック値が第２クロック値より同一であるかまたは大きければ、前記の実行可能なプログラムから第３命令をフェッチし、第２プロセッサシミュレータ上の第３命令の実行をシミュレーションし、第２プロセッサ上の第３命令を遂行するに必要な時間を第２クロック値に加えて、第２プロセッサシミュレータがこれ以上必要となくなると第１ループに戻って、前記の実行可能なプログラム内の第１プロセッサシミュレータと関連されているすべての命令がシミュレーションされたら、第１ループを終了するコンピュータ実行段階を含むことを特徴とする。

【００１５】

請求項９記載の発明による二重プロセッサ回路の動作をシミュレーションするためのコンピュータ実行方法は、二重プロセッサ回路上で実行される命令を含む実行可能なプログラムと、二重プロセッサ回路の第１プロセッサ上の命令の実行をシミュレーションするための第１プロセッサシミュレータと、二重プロセッサ回路の第２プロセッサ上の命令の実行をシミュレーションするための第２プロセッサシミュレータと、第１プロセッサシミュレートと第２プロセッサシミュレート間の命令の実行をインターリーブするための同期装置と、第１プロセッサクロックを示す第１クロック値と、第２プロセッサクロックを示す第２クロック値と、命令を実行することに必要な時間を示す検査表と、第１プロセッサシミュレータと第２プロセッサシミュレータで示されて、第１プロセッサシミュレータに要求される１個以上の第１レジスタ値及び第２プロセッサシミュレータに要求される１個以上の第２レジスタ値を含む、共有されるメモリと、を含むコンピュータ格納媒体に提供される二重プロセッサ回路の動作をシミュレーションするためのコンピュータ実行方法において、前記の実行可能なプログラムから第１命令をフェッチし、第１命令が並列プロセッシングのための信号でなければ、第１プロセッサシミュレータ上の第１命令の実行をシミュレーションし、第１命令が、並列処理のための信号であれば、並列処理をシミュレーションするコンピュータ実行段階を含む第１ループを遂行し、前記の並列処理をシミュレーションする段階が、クロック値と第２クロック値とを比較し、第１クロック値が第２クロックより小さければ、前記の実行可能なプログラムから第２命令をフェッチし、第１プロセッサシミュレータ上の第２命令の実行をシミュレーションし、第１プロセッサ上の第２命令を実行することに必要な時間を第１クロック値に加えて、第１クロック値が第２クロック値より同一であるかまたは大きければ、前記の実行可能なプログラムから第３命令をフェッチし、第３命令に必要な実行時間を決定するために検査表をアクセスし、第２プロセッサシミュレータ上の第３命令の実行をシミュレーションし、第２プロセッサ上の第３命令を遂行することに必要な時間を第２クロック値に加えて、第３命令に必要な実行時間を基準とし、実行可能なプログラムから所定の命令の実行をシミュレーションし、第２プロセッサシミュレータがこれ以上必要となくなると第１ループに戻って、前記の実行可能なプログラム内の第１プロセッサシミュレータと関連されているすべての命令がシミュレーションされたら、第１ループを終了するコンピュータ実行段階を含むことを特徴とする。

【００１６】

【発明の実施の形態】

＜付録Ａに関する参考＞

本明細書の一部である表１～表５に示されている付録Ａは、本発明のコンピュータプログラム実施例として以下で詳細に説明する。付録Ａは、著作権が保護されなければならないもので、著作権者は米国特許及び商標国の特許ファイル、または記録に示されているように、誰も特許明細書をファクシミリ複写できることには異議ないが、そうでなければすべての著作権が取り消される。

【００１７】

図１は本発明の実施の形態の信号プロセッサシミュレータの概略構成を示すブロック図である。

【００１８】

この図において、コアイメージ（core image）１０２は、２個の他のプログラムのための

実行可能なファイルからなる組み合わされた実行可能なファイルである。一方のプログラムは、ＡＲＭ７プロセッサ（“ＡＲＭ７プロセッサプログラム”）を動作させるために設計されたもので、他方のプログラムはベクトルプロセッサ（“ベクトルプロセッサプログラム”）を動作させるために設計されたものである。

【００１９】

ＡＲＭ７プロセッサは、ＡＲＭプロセッサまたは“ＡＲＭ”とも呼ばれる。ベクトルプロセッサは、ベクトルコプロセッサまたは“ＶＣＰ”とも呼ばれる。それぞれのプログラムの実行可能なファイルは、コード部分領域とデータ部分領域とを含んでいる。

【００２０】

２個の他のプログラムは、共有する変数及びそれぞれの変数を有する。コアイメージ１０２の一例を表６に示した付録Ｂに表した。付録Ｂにおいて、コアイメージ１０２は、ＡＲＭ７プロセッサプログラムに関するコード部分（Ａ１－Ａ４ライン）、ＡＲＭ７プロセッサプログラムに関するデータ部分（Ａ５ライン）、ベクトルプロセッサプログラムに関するコード部分（Ａ６－Ａ７ライン）、及びベクトルプロセッサプログラムに関するデータ部分（Ａ８）を順序に含む。このコアイメージ１０２は二重プロセッサシミュレータ１０４の入力となる。

【００２１】

二重プロセッサシミュレータ１０４は、命令－ラベルシミュレータである。二重プロセッサシミュレータ１０４は、２個のプロセッサシミュレータ間の命令実行シミュレーションをインターリーブし、二プロセッサ上の並列命令実行をシミュレーションする。

【００２２】

一方のプロセッサシミュレータはＡＲＭ７・ＲＩＳＣプロセッサの動作をシミュレーションし、他方のプロセッサシミュレータはベクトルプロセッサの動作をシミュレーションする。ＡＲＭ７・ＲＩＳＣプロセッサのために設定された構造と命令は、英国ケンブリッジのＡｄｖａｎｃｅ　ＲＩＳＣ　Ｍａｃｈｉｎｅｓ Ｌｔｄ．により提供される文書番号：ＡＲＭ　ＤＤＩ　００１０Ｇの“ＡＲＭ７ＤＭ　Ｄａｔａ　Ｓｈｅｅｔ”に記述されている。ＡＲＭ７ＤＭデータシートは本明細書の参考文献として共に記述されている。

【００２３】

ＡＲＭ７レジスタと命令は、本明細書の参考文献として共に記述された米国特許出願番号０８／６９９，３０３、弁理士事件番号Ｍ－４３６８の“Methods and Apparatus for Processing Video Data ”にも記述されている。

【００２４】

ＡＲＭ７・ＲＩＳＣプロセッサシミュレータの一例であるＡＲＭＵＬＡＴＯＲは、英国ケンブリッジのAdvance RISC Machine Ltd. により常用化されたものである。

【００２５】

ベクトルプロセッサのために設定された構造と命令は、本明細書の参考文献の一つであるカリフォルニア・サンホセの三星半導体の“Preliminary MSP-IEX System Specification ”及び“MSP Architecture Specification ”に記述されている。

【００２６】

ベクトルプロセッサレジスタと命令は、米国特許出願番号０８／６９９，５９７、弁理士事件番号Ｍ－４３５５の“Single-Instruction-Multiple-Data Processing in a Multimedia Signal Processor ”に記述されている。

【００２７】

ベクトルプロセッサシミュレータの一例は、マルチメディア命令ラベルシミュレーションツール（Multimedia Instruction Level simulation Tool;MINT ）であり、これはカリフォルニア・サンホセの三星半導体の“Architecture and Design Specification of MINT ”に記述されている。

【００２８】

ＡＲＭ７・ＲＩＳＣプロセッサとベクトルプロセッサは例示的なもので、本発明を特定な

プロセッサシミュレータとして制限するものではない。本明細書を考慮すると、本発明の原理は適切なプロセッサシミュレータを採用する限り、プロセッサの特定構造でも、すべての二重プロセッサ回路のシミュレーションに使用される場合もある。例えば、ＡＲＭ７・ＲＩＳＣプロセッサは、従来の構造である多目的プロセッサに代替され得る。

【００２９】

二重プロセッサシミュレータ１０４は、ＡＲＭ７プロセッサシミュレータとベクトルプロセッサとを一つの共有するアドレス領域に丈夫に連結させる。この２個の他のプロセッサシミュレータは、傘のような構造のプロセッサの傘の骨の役割に該当するそれぞれのプロセッサシミュレータ動作の一過程として共に実行される。

【００３０】

二重プロセッサシミュレータ１０４は命令実行のシミュレーションをインターリーブする。例えば、ＡＲＭ７プロセッサシミュレータが命令実行をシミュレーションしてから、ベクトルプロセッサシミュレータが命令実行をシミュレーションし、それからＡＲＭ７プロセッサシミュレータの動作が続く。シミュレータの動作は、このように飛ばす（leap-frog）形態で並列に実行される命令がこれ以上存在しない時まで継続される。

【００３１】

二重プロセッサシミュレータ１０４は、どんなプロセッサシミュレータが一番最後のクロックキング技術を通して次に動作するかを決定する。別途のクロックシミュレータがそれぞれのプロセッサシミュレータに対して提供される。命令をフェッチする前に２個のクロックシミュレータを比較して、いずれのものが最後のものであるかを決定する。一番最後のクロックシミュレータと関連されたプロセッサシミュレータが命令の実行をシミュレーションしてから、プロセッサシミュレータとなる。

【００３２】

二重プロセッサシミュレータ１０４の動作は、２個のプロセッサシミュレータ間の同期を維持することによって、二重プロセッサ集積回路上での実行順序と比較した時に、命令が一つの命令以上から外れて実行されないようにする。このように命令の実行をシミュレーションすることで費用を低く抑えることができる。

二重プロセッサシミュレータ１０４は、コアイメージ１０２の入力で動作する二重プロセッサ集積回路から認可される結果を模写するシミュレーション結果１０６を生成する。シミュレーション結果１０６はコアイメージ１０２を処理することに必要な周期を表す作業データを含む。シミュレーション結果１０６はコードをデバッグ（debug）し、作業数値を得るための正確な情報を提供する。

【００３３】

本実施の形態による二重プロセッサシミュレータ１０４の構造を図２に詳細に示した。

【００３４】

二重プロセッサシミュレータ１０４の最上部は、デバッガーインタフェース（debugger interface）２０２である。このデバッガーインタフェース２０２はプロセッサ制御及びデバッグ命令を使用者から受ける。この場合、デバッガーインタフェース２０２はレジスタをディスプレイし、プログラム命令を目録化し、変数を変更するような機能を遂行するための命令を受ける。なお、デバッガーインタフェース２０２は、ＷＩＮＤＯＷＳ９５™使用者インタフェースのための仕様に適合するように変形され得る。ＷＩＮＤＯＷＳ９５™は、ワシントン・レドモンドのマイクロソフト社により常用化されたものである。

【００３５】

同期装置（synchronizer）２０４は、デバッガーインタフェース２０２と二重プロセッサシミュレータ１０４の残余要素間の制御を行う部分である。デバッガーインタフェース２０２は同期装置に命令を伝達し、同期装置２０４からデータが印加される。

【００３６】

同期装置２０４は、メモリ位置ＡＲＭ７ｃｌｋ２１６（ＡＲＭ７プロセッサクロックのシミュレーションのための）と、メモリ位置ＶＰｃｌｋ２１８（ベクトルプロセッサクロックのシミュレーションのための）とを含んでいる。同期装置２０４は２個のプロセッサシ

ミュレータであるＡＲＭ７プロセッサシミュレータ２０６（“ＡＲＭ７シミュレータ２０６”ともいう）とベクトルプロセッサシミュレータ２０８（“ＶＰシミュレータ２０８”ともいう）に連結される。情報は同期装置２０４と２個のプロセッサシミュレータ２０６、２０８間の両方向に伝達される。

【００３７】

ＡＲＭ７プロセッサシミュレータ２０６とＶＰプロセッサシミュレータ２０８は、メモリアクセスモジュール２１０、コプロセッサモジュール２１２及びイベントモジュール２１４を共有する。メモリアクセスモジュール２１０の一部はプロセッサシミュレータ２０６，２０８に示される。メモリアクセスモジュール２１０は、２個のプロセッサシミュレータがそれぞれの変数と共有する変数を維持する。メモリアクセスモジュール２１０はＡＲＭ７プロセッサのレジスタの内容を維持するためのメモリ位置（ARM7 reg mem）２２０と、ベクトルプロセッサのレジスタの内容を維持するためのメモリ位置（VP reg mem）２２２とを備えている。

【００３８】

コプロセッサモジュール２１２は、ＡＲＭ７プロセッサシミュレータ２０６とＶＰプロセッサシミュレータ２０８との間のデータ伝送を遂行させる役割をする。イベントモジュール２１４はリセット、排除（exception)及びインタラプトのようなイベントを処理する。メモリアクセスモジュール２１０、コプロセッサモジュール２１２及びイベントモジュール２１４は直接的に連結されない。

【００３９】

ＡＲＭ７プロセッサシミュレータ２０６は、ＶＰプロセッサシミュレータ２０８を丈夫に連結し、二重プロセッサ回路構造を精密に複写できる。例えば、ベクトルプロセッサシミュレータ２０８は、二重プロセッサ集積回路上の状態レジスタを表すメモリ位置（VP reg mem）２２０に格納された状態レジスタ変数の書き込みが可能であり、ＡＲＭ７プロセッサシミュレータ２０６は、ＡＲＭ７プロセッサシミュレータ２０６とＶＰプロセッサシミュレータ２０８との間の情報伝達のために前記と同一な状態レジスタ変数をポーリング（polling）することができる。

【００４０】

ＡＲＭ７プロセッサシミュレータ２０６は、ＶＰプロセッサシミュレータ２０８によりポーリングされたレジスタに書き込みできる。メモリアクセスモジュール２１０を通したこのような情報伝達は、ＡＲＭ７プロセッサシミュレータ２０６とＶＰプロセッサシミュレータ２０８との間の情報伝達のためにファイル、パイプ、または共有されるバッファを使用することより効率的である。このような方法には、付加的な手段管理と性能を落とす動作のアクセスが要求される。

【００４１】

例えば、ある動作システムにおいて、ファイルを通した情報伝達のためには、そのファイルを開き、そのファイルに記入してファイルを閉じるためのＶＰプロセッサシミュレータ２０８が要求され、ＡＲＭ７プロセッサシミュレータ２０６は前記ファイルを開き、このファイルから読み出しを遂行してファイルを閉じるべきである。上述のごとく、メモリアクセスモジュール２１０を利用して正常的なプロセッサ動作のシミュレーションのための情報伝達により、余分の段階を除去できる。

【００４２】

二重プロセッサシミュレータ１０４の動作を図３に詳細に図示した。

【００４３】

まず、“デバッガーインタフェースの始動”ブロック３０２でデバッガーインタフェース２０２（図２に図示されている）が始動される。“同期装置の始動”ブロック３０４で使用者は二重プロセッサ回路上で実行される命令のシミュレーションを要求し、デバッガーインタフェース２０２は同期装置２０４（図２参考）に信号を送る。

【００４４】

同期装置２０４は、ＡＲＭ７プロセッサシミュレータ２０６とＶＰプロセッサシミュレー

タ２０８との間の命令実行のシミュレーションをインターリーブする方式に同期させて、二重プロセッサ回路上で発生する並列実行に近づくようにする。

【００４５】

同期装置２０４は、駆動を開始してから“ＡＲＭ７プロセッサシミュレータの活性化”ブロック３０６で、ＡＲＭ７シミュレータ２０６（図２参考）を始動する。二重プロセッサ回路上でＡＲＭ７プロセッサとベクトルプロセッサは主従（master-slave）関係を有する。ベクトルプロセッサの動作はＡＲＭ７プロセッサで実行される命令“ＳＴＡＲＴＶＰ”（付録Ｂ、Ａ２ライン）を通して始動される。このような関係をシミュレーションするために、ＡＲＭ７プロセッサシミュレータ２０６が先に活性化される。前記の命令がベクトルプロセッサを始動するための信号として使用されるために、必ず“ＳＴＡＲＴＶＰ”と呼ばれる必要がないということは、当分野に通常の知識を有する者は理解できる。

【００４６】

ＡＲＭ７プロセッサシミュレータ２０６が活性化された後、同期装置２０４は“ＡＲＭ７部分から命令をフェッチ”ブロック３０８を遂行する。ブロック３０８で同期装置２０４は、ＡＲＭ７プロセッサプログラム（表６に示されている付録Ｂ、Ａ１－Ａ４ライン）に該当するコアイメージ１０２のコード部分から命令を検索する。

【００４７】

“ＡＲＭ７部分を終了するか？”決定３１０段階で、同期装置２０４はＡＲＭ７プログラムが終了されなかったこと、すなわちブロック３０８でＡＲＭ７命令を成功的にフェッチしたことを確認する。ＡＲＭ７プログラムが終了されると、同期装置２０４は動作を終了し、デバッガーインタフェース２０２に戻る。

【００４８】

同期装置２０４が、ブロック３０８で成功的に命令をフェッチした場合、同期装置２０４は“ＳＴＡＲＴＶＰ命令であるか？”決定ブロック３１２で、この命令が“ＳＴＡＲＴＶＰ”であるかを決定する。ブロック３０８で同期装置２０４がフェッチした命令が“ＳＴＡＲＴＶＰ”命令であると、処理過程は“並列処理のシミュレーション”ブロック３１４に移動する。

【００４９】

“ＳＴＡＲＴＶＰ”命令とぶつかる時まで同期装置２０４の制御下の単一プロセッサは、単一流れ（thread）を有する。この流れは、ＡＲＭ７プロセッサシミュレータ２０６の動作と関連する。“ＳＴＡＲＴＶＰ”命令は、ＶＰプロセッサシミュレータ２０８の動作と関連している他の流れの生成を要求する。ブロック３１４は、次に記述する“並列処理”で詳細に説明する。

【００５０】

ブロック３０８でフェッチされた命令が“ＳＴＡＲＴＶＰ”でなければ、同期装置２０４は、ＡＲＭ７プロセッサシミュレータ２０６が“ＡＲＭ７命令のシミュレーション”ブロックでフェッチされた命令の実行をシミュレーションする信号を送る機能を遂行する。

【００５１】

なお、本発明の他の実施の形態として、同期装置２０４は、ＡＲＭ７プロセッサシミュレータ２０６を使用するためのサブルーチン信号を送るようにしてもよい。この場合、同期装置２０４がＡＲＭ７プロセッサシミュレータ２０６を使用する特定な方法は、同期装置２０４とＡＲＭ７プロセッサシミュレータのプログラミング遂行に依存する。

【００５２】

ブロック３１６のプロセッシングが終了されてから、ＡＲＭ７プロセッサシミュレータ２０６は同期装置２０４に戻り、制御ループはブロック３０８に戻って、ＡＲＭ７プログラム（付録Ｂ、Ａ１ライン）から付加的な命令を読み出す。

【００５３】

＜並列処理＞

上述のごとくブロック３１４は、二重プロセッサ集積回路のＡＲＭ７プロセッサとベクトルプロセッサ上の命令の並列実行をシミュレーションする。ここで、ブロック３１４を図

４に図示した。プロセッシングは同期装置２０４がＶＰプロセッサ２０８を活性化する“ベクトルプロセッサシミュレータの活性化”ブロック４０２で開始される。ＶＰプロセッサシミュレータ２０８が命令をプロセッサするために用意してから、同期装置２０４は“シミュレータクロックの開始”ブロック４０４を実行する。

【００５４】

同期装置２０４は、２個の独立されたクロックシミュレータを時間プロセッサシミュレータ２０６，２０８に維持させるが、この中で一方のクロックシミュレータはＡＲＭ７クロック（図５参考）をシミュレーションし、他方のクロックシミュレータはベクトルクロック（図５参考）に該当する。ＡＲＭ７クロックシミュレータはメモリ位置ＡＲＭ７ｃｌｋ２１６（図２参考）を含む。ベクトルクロックシミュレータはメモリ位置ＶＰｃｌｋ２１８（図２参考）を含む。

【００５５】

ブロック４０４で同期装置２０４はメモリ位置ＡＲＭ７ｃｌｋ２１６とＶＰｃｌｋ２１８をｔ１値（図５参考）に初期化する。

【００５６】

なお、本発明の実施の形態において、ＡＲＭ７クロックは４０ＭＨｚ周波数で動作し、ベクトルクロックは８０ＭＨｚの周波数で動作する。なお、この二クロックの周波数のみに制限されるのではなく、本発明の原理はいずれの周波数のプロセッサでも使用可能であることは言うまでもない。

【００５７】

同期装置２０４は、２個のプロセッサシミュレータの中のいずれのものが命令を受けて次の命令を遂行するかを決定するために、前記の二クロックシミュレータを利用する。同期装置２０４は、続いて“二重処理流れがそのまま活性化の状態であるか？”決定ブロック４０６を実行する。両側の流れが活性化状態でなければ、プロセッサシミュレータの中の或る一つに対するプロセッシングが終了され、これにより並列処理はこれ以上必要なくなる。並列処理がこれ以上必要なくなると、ブロック３１４は終了する。

【００５８】

両側の流れがまだ活性化状態であれば、プロセッシングは“ＡＲＭ７クロックが後であるか？”を決定するブロック４０８に移動する。決定段階４０８で同期装置２０４は、ＡＲＭ７クロックシミュレータがベクトルクロックシミュレータより後であるかを決定する。このような決定はＡＲＭ７ｃｌｋ２１６とＶＰｃｌｋ２１８を比較して遂行される。例えば、ＡＲＭ７ｃｌｋ(216) ＝ｔ１で、ＶＰｃｌｋ(218) ＝ｔ３であれば（図５参考）、ＡＲＭ７クロックシミュレータがもっと後である。ＡＲＭ７クロックがもっと後であれば、これはＶＰプロセッサシミュレータ２０８が命令をプロセッシングし、ＡＲＭ７プロセッサシミュレータ２０６はアイドル（idle）状態であることを表す。二重プロセッサ回路において２個のプロセッサが存在するので、この二プロセッサが同時に動作する。

【００５９】

ＡＲＭ７プロセッサシミュレータ２０６がアイドル状態であれば、ＡＲＭ７プロセッサシミュレータ２０６は、次の命令をプロセッシングすることにより命令実行の順序が維持され、並列処理がシミュレーションされる。例えば、図５を参考すると、ＶＰプロセッサシミュレータ２０８が時間ｔ４の間にベクトルプロセッサ動作をシミュレーションし、ＡＲＭ７プロセッサシミュレータ２０６が時間ｔ２の間にＡＲＭ７プロセッサ動作をシミュレーションすると、命令を実行する次のプロセッサはＡＲＭ７プロセッサシミュレータ２０６になるはずである。ＡＲＭ７プロセッサシミュレータ２０６は少なくともＶＰプロセッサシミュレータ２０８が動作する時間だけ、動作する間に命令の処理を継続する。或る命令は終了されるための時間の量が異なるので、同期装置２０４はアイドル状態で最後まで待機しているプロセッサシミュレータがもっと多い命令を実行するように、２個のプロセッサシミュレータ間の命令実行の厳しいインターリービングから外れるようになる。

【００６０】

ＡＲＭ７クロックシミュレータがベクトルクロックシミュレータより後であれば（即ち、

ＡＲＭ７ｃｌｋ２１６＜ＶＰｃｌｋ２１８）、プロセッシングは同期装置２０４がコアイメージ１０２のＡＲＭ７プログラムのコード部分（付録Ｂ、Ａ２－Ａ４ライン）から命令をフェッチする“ＡＲＭ７命令のフェッチ”ブロック４１０に移動する。前記命令がフェッチされてから、プロセッシングは同期装置２０４がＡＲＭ７シミュレータ２０６によって、ＡＲＭ７プロセッサ命令の実行をシミュレーションするように信号を送る“ＡＲＭ７命令のシミュレーション”ブロック４１２に移動する。命令実行のシミュレーションの一部として同期装置２０４はＡＲＭ７プロセッサ上のシミュレーティングされた命令を遂行することにかかる時間をＡＲＭ７ｃｌｋ２１６に加える。例えば、図５を参考すると、ＡＲＭ７プロセッサシミュレータ２０６がＡＲＭ７ｃｌｋ(216) ＝ｔ１で命令の実行を開始し、命令を実行することに必要な時間がＡＲＭ７プロセッサ上の１クロック周期であれば、ＡＲＭ７ｃｌｋ(216) はｔ２にセットされる。ＡＲＭ７命令の実行がシミュレーションされてから、プロセッシングは決定段階４０６に移動し、命令の並列実行をシミュレーションするループを形成する。

【００６１】

この実施の形態において、ＡＲＭ７プロセッサが、ベクトルプロセッサが実行（付録Ｂ、Ａ４ライン）を完了することを待っている間、待機ループを形成することも可能である。ＡＲＭ７プロセッサシミュレータ２０６が待機ループをシミュレーションする時、両側の流れはまだ活性化状態であり、並列処理がシミュレーションされる。

【００６２】

ＡＲＭ７クロックシミュレータがベクトルクロックシミュレータより後でなければ（即ち、ＡＲＭ７ｃｌｋ２１６＞＝ＶＰｃｌｋ２１８）、処理過程は“ベクトル命令のフェッチ”ブロック４１４に移動してから、“ベクトル命令のシミュレーション”ブロック４１６に移動する。

【００６３】

なお、本発明の他の実施の形態として、ＡＲＭ７クロックシミュレータとベクトルクロックシミュレータとが同一であれば、ＡＲＭ７プロセッサシミュレータ２０６がＡＲＭ７プログラム命令の実行をシミュレーションする。図４のブロック４１４で、ＡＲＭ７クロックシミュレータが後の場合の処理過程と類似して、同期装置２０４はコアイメージ１０２のベクトルプログラムのコード部分（付録Ｂ、Ａ６－Ａ７ライン）から命令をフェッチする。ブロック４１６で、同期装置２０４はＶＰクロックシミュレータ２０８が命令の実行をシミュレーションする。ベクトルプログラム命令実行のシミュレーションの一部として、同期装置２０４はＶＰｃｌｋ(216) をベクトルプロセッサ上の命令を実行することに必要な時間だけ増加させる。ブロック４１６以後の処理過程の制御は決定ブロック４１６に移動されて命令の並列実行をシミュレーションするループを形成する。

【００６４】

本発明の実施の形態において、ベクトルプロセッサ命令“ＪＯＩＮ”（付録Ｂ、Ａ７ライン）に注目する。ＡＲＭ７命令“ＳＴＡＲＴＶＰ”に対応する“ＪＯＩＮ”は、ベクトルプログラムが終了され、ＶＰプロセッサシミュレータ２０８に対する処理流れが取り消されなければならないし、処理過程が基本ＡＲＭ７プログラムに戻るはずであるということを表す。前記の命令をベクトルプロセッサを終了する信号として使用するためには、必ず“ＪＯＩＮ”としなくても良いということは、当分野の通常の知識を有する者は理解できる。

【００６５】

ＡＲＭ７プロセッサシミュレータ２０６とＶＰプロセッサシミュレータ２０８の丈夫な連結の一つの長所は、同期装置２０４がコンテキスト（context ）とタスクスイッチング（task switching）でぶつかるようになる費用の問題を避けることにある。同期装置２０４はプロセッサシミュレータ（ブロック４１２のＡＲＭ７プロセッサシミュレータ２０６とブロック４１６のＶＰプロセッサシミュレータ２０８）に信号することができる。同期装置２０４が本来は二プロセッサ間のタスクスイッチングであることは、当分野で通常の知識を有する者は理解できる。

【００６６】
二つのプロセッサシミュレータ２０６，２０８が丈夫に連結されなければ、命令の形態がその前にシミュレーションされた形態から変更されるごとに、一方のプロセッサシミュレータは停止し、他方のプロセッサシミュレータが始動されるはずであるので、実行される新しいプロセッサが必要である（すなわち、ARM7命令からベクトルプロセッサ命令に変更され、またはベクトルプロセッサ命令からARM7命令に変更される）。
【００６７】
なお、本発明の他の実施の形態として決定ブロック４０６から開始される図４に図示したループは、クロック数を減らすために変更されることもある。図４で同期装置２０４は、それぞれの命令を実行してから、ＡＲＭ７ｃｌｋ(216) とＶＰｃｌｋ(218) とを比較する。図６は前記の変更されたループを表したもので、ブロック６０２，６０４が追加されたことを除外しては、図４のループと同一である。
【００６８】
それぞれのベクトル命令が必要とするクロック周期の数を含むクロック周期検査表（clock-cycle lookup table）（図示されていない）が同期装置２０４内に構成される。ブロック４１４が終了された後のブロック４１６の前、ベクトル命令の実行がシミュレーションされる前に“命令周期時間の検査”ブロック６０２で、同期措置２０４は命令が数個の周期を必要とするかを決定するためにクロック周期検査表を検索する。
【００６９】
一般的に、ＡＲＭ７プロセッサ上の命令は１周期を必要とし、ベクトルプロセッサ上の命令は多数のクロック周期を必要とする。ブロック６０２が終了されてから、“ベクトル命令のシミュレーション”ブロック４１６が上述のごとく実行される。ベクトル命令のシミュレーションの後、ＡＲＭ７ｃｌｋ(216) とＶＰｃｌｋ(218) を比較する代わり、“ＡＲＭ７命令の反復実行（loop）”ブロック６０４で、同期装置２０４はベクトル命令に要求されるクロック数の半分に該当する数のＡＲＭ７命令をフェッチし、その命令の実行をシミュレーションする。例えば、処理過程が時間ｔ＝１から開始され（図５参考）、８個のクロック周期を必要とするベクトルプロセッサ命令が実行されると仮定すると、ベクトルプロセッサは時間ｔ＝５で命令の実行を終了する。ＶＰクロックがＡＲＭ７クロックより２倍程度早いので、ＡＲＭ７ｃｌｋ(216) はＶＰｃｌｋ(218) より８クロック周期でない４クロック周期遅れるようになる。
【００７０】
同期装置２０４は、４個のＡＲＭ７命令をフェッチしてから、ＡＲＭ７プロセッサシミュレータ２０６に信号を送ることにより、これらの命令の遂行をシミュレーションする。図４に示したプロセッサ終了の後、同期装置２０４はブロック４１４からフェッチされたベクトル命令がシミュレーションされた後、４回のクロック比較を行う。図６に示した方法はこのような４回のクロック比較を必要としない。ブロック６０６後、処理過程は図４のブロック４１６以後の過程と同様である。
【００７１】
付録Ａのコンピュータプログラムは、本発明の実施の形態において、マイクロソフト社により常用化されたマイクロソフトビジュアルＣ＋＋４．０の統合された開発環境を利用し、コムファイルされリンクされたもので、マイクロソフト社により常用化されたマイクロソフトＷＩＮＤＯＷＳ９５™を利用する個人用コンピュータ上で使用されることである。
【００７２】
他の実施の形態において、付録Ａのコンピュータプログラムは、ＵＮＩＸ　Ｓｏｌａｒｉｓ２．５駆動システムとＧＣＣコムファイラー及びリンカーを利用してコムファイル、リンクされる。付録Ａのコンピュータプログラムに使用される特定なコンピュータ言語と、付録Ａのコンピュータプログラムにより規定されるコンピュータプロセッサが遂行される、コンピュータシステムは、本発明の重要な面ではない。他のコンピュータ言語及び他のコンピュータシステムを利用して本発明を具現できることは当分野の通常の知識を有する者は理解できる。

【００７３】
なお、上述の実施の形態は例示的なので、本発明を制限しない。例えば、前記の実施の形態では、ＡＲＭ７プロセッサとベクトルプロセッサのシミュレーションに関して記述しているが、他のプロセッサにも本発明を適用し、２個以上のプロセッサの実行をシミュレーションし、マルチプロセッサ回路が集積回路ではない場合もある。
【００７４】
また、本発明を特定の望ましい実施の形態に関連して図示、説明したが、以下の特許請求の範囲により用意される本発明の精神や分野を逸脱しない限り、本発明が多様に改造及び変化することができるということは、当業界で通常の知識を有する者は容易に知ることができる。
【００７５】
このように、実施の形態では、新しい二重プロセッサ回路シミュレータは、それぞれ分離されたシミュレータを連結し、このシミュレータ間の命令の実行をインターリーブする。この新しい二重プロセッサシミュレータは、正確なシミュレーションと基礎となる集積回路の動作において優秀な性能を提供する。
【００７６】
同期装置２０４は、ＲＩＳＣプロセッサシミュレータとベクトルプロセッサシミュレータの調整された動作を処理する。二重プロセッサシミュレータの入力は基礎となるＲＩＳＣプロセッサの実行可能なファイルと、基礎となるベクトルプロセッサの実行可能なファイルとを組み合って構成された実行可能な組合ファイルから構成される。同期装置２０４は外部ループ内のＲＩＳＣプロセッサと関連される、組み合わされた実行可能なファイルから命令をフェッチする。同期装置２０４はＲＩＳＣプロセッサと関連されたすべての命令が遂行されるとき動作を終了する。
【００７７】
同期装置２０４は、ベクトルプロセッサが初期化しなければならないフェッチされた命令信号を除外した前記フェッチされた命令実行をＲＩＳＣプロセッサシミュレータがシミュレーションするようにする。ベクトルプロセッサが要求される場合、同期装置２０４は並列処理をシミュレーションする。
【００７８】
並列処理をシミュレーションするために、同期装置２０４は、まずベクトルプロセッサシミュレータを動作させた後、ＲＩＳＣプロセッサシミュレータに該当するクロックシミュレータを初期時間にセットさせる。ベクトルプロセッサに該当するクロックシミュレータは同一な初期時間にセットされる。
【００７９】
同期装置２０４は、並列処理が要求される間、内部ループを実行する。並列処理がこれ以上要求されなければ、同期装置２０４は外部ループに戻る。内部ループで、同期装置２０４はＲＩＳＣクロックシミュレータとベクトルクロックシミュレータとを比較し、ＲＩＳＣクロックシミュレータが遅ければ、同期装置２０４は組み合わされた実行可能なプログラムからＲＩＳＣ命令をフェッチする。この命令はＲＩＳＣプロセッサシミュレータ上でシミュレーションされる。
【００８０】
シミュレーションの後、ＲＩＳＣプロセッサが命令を実行するのに必要な時間がＲＩＳＣクロックシミュレータ値に加えられる。その後、同期装置２０４は内部ループの最初に戻る。
【００８１】
ベクトルクロックシミュレータがＲＩＳＣクロックシミュレータと同一、またはＲＩＳＣクロックシミュレータより遅ければ、同期装置２０４は組み合わされた実行可能なプログラムからベクトルプロセッサ命令をフェッチする。この命令はベクトルプロセッサシミュレータ上でシミュレーションされる。シミュレーションした後、ベクトルプロセッサが命令を実行するのに必要な時間がベクトルクロックシミュレータ値に加えられる。それから

、同期装置２０４は内部ループの最初に戻る。
【００８２】
ＲＩＳＣプロセッサとベクトルプロセッサ間の情報伝達は、共有されるメモリを利用して遂行される。ＲＩＳＣプロセッサはベクトルプロセッサレジスタへの書き込みや読み出しを遂行し得る。
【００８３】
本発明によると、多数のプロセッサ回路の動作をシミュレーションする方法及び装置が提供される。本発明の実施の形態において、プロセッサ回路は二重プロセッサ集積回路である。本発明の実施の形態によると、新しい二重クロックシミュレーション方法は、丈夫に連結された２個の独立的なプロセッサシミュレータ間の命令実行シミュレーションをインターリーブする。集積回路上の二重プロセッサを表す２個の独立的なプロセッサシミュレータは、一つの共有されるアドレス空間で統合される。２個のプロセッサシミュレータは相互に情報を伝達し得るメモリを共有する。一般的に、命令実行が同期されることによりシミュレータプロセッサはシミュレーション命令実行を交替に行う。
【００８４】
このようにして、集積回路上で可能な並列実行がシミュレーションされる。別途のクロックシミュレータはそれぞれのプロセッサシミュレータに備えられる。
次の命令の実行をシミュレーションするために選択された実際のプロセッサは、最後のクロックシミュレータと関連されたプロセッサである。このように命令実行のシミュレーションが行われるようにすると、命令実行の順序が実際の二重プロセッサ集積回路の実行順序の一つの命令内にあることが確実になる。
【００８５】
【発明の効果】
本発明によると、従来のシミュレータに比して向上された性能を有する二重プロセッサ回路のシミュレータが提供され、基礎となる電子回路に存在する信号伝達動作を正確にシミュレーションするシミュレータを提供できる。
【００８６】
【表１】

[付 録 A]

```
/*  Synchronize the ARM and VCP
 *  If VCP is IDLE run/step ARM
 *  NumVcpCycles contains the number of cycles available for VCP
 * execution
 * if RUN do as many instructions as available cycles
 *  if STEP run one instruction and then break
 *  telling debugger that it was a VCP instruction
 */
    if (state->vcp.UER_Regs[VPSTATE].int_val & UER_MASK &&
    (state->NumVcpCycles > 0)) {
        /* VCP is running and there are cycles available */
        do {
            int nCount = state->vcp.CycleCount ;
            if (state->CallDebug > 0) {
                /* check for breakpoint */
                ARMword pc = state->vcp. SP_Regs[VPC].int_val;
                ARMul_Debug(state, pc, 0) ;
                if (state->Emulate < ONCE) {
                    state->Emulate = ONCE;
                    break ;
                }
            }
            VCP_DoInstr(state) ;      /* Emulate VCP instruction */
            state->NumVcpCycles -= (state->vcp. CycleCount -
                                    nCount);
    } while (state->Emulate == RUN && (state->NumVcpCycles > 0) &&
    (state->vcp.UER_Regs[VPSTATE].int_val & UER_MASK)) ;

        if  (state->Emulate == ONCE) {
            state->Emulate = STOP ;
            state->NextInstr = RESUME ;
            if  (state->EndCondition == RDIError_BreakpointReached)
               state->EndCondition = RDIError_VcpBreakpointReached ;
            else
                state->EndCondition = RDIError_VCPInstruction ;
            break ;

        }
    }
/*****************************************************************\
*       Execute the next instruction                              *
\*****************************************************************/
#   ifdef CYCLEBASED
    instr = state->instr ;
    decoded = state->decoded ;
    loaded = state->loaded ;
    pc = state->pc ;
    temp = state->temp ;
    instruction_size = INSTRUCTIONSIZE;
    has_halfword_prop = state->Processor & ARM_Halfword_Prop;

    if (state->EventSet)
        ARMul_InvokeEvent(state) ;
```

【００８７】

【表２】

［表１から続く］

```
#   ifdef MODE32
    UNDEF_Prog32SigChange ;
    UNDEF_Data32SigChange ;
    #endif

    if (state->NresetSig == LOW) {
        if (state->Reseted) {
            state->NrwSig = LOW ;
            ADDRWORD;
            state->NopcSig = LOW ;
            state->lockSig = LOW ;
            SCYCLE ;
            state->NcpiSig = HIGH ;
            state->doutenSig = LOW ;
            state->LSCActive = FALSE ;
            state->NextCycle = RESET1 ;
            state->NexecSig = 1 ;
            }
        else
            state->Reseted = TRUE ;
        }
else {
        switch (state->LastCycle) {
case NEWINSTR :
    PREFETCH(loaded,pc + instruction_size * 2) ;
        break;
    case DESTPC3 :
        /* !! NOT instruction_size * 2 */
        PREFETCH(loaded,pc + instruction_size) ;
        break ;

    case STR2 :
    case STRB2 :
    case STRH2 :
        state->doutenSig = LOW ;
        if (state->abortSig || state->Aborted) {
            if (!state->lateabtSig)
                LSBase = state->OldBase ;
            if (!state->Aborted)
                state->Aborted = ARMul_DataAbortV ;
            if (state->Processor & ARM_Abort7_Prop)
                state->NextCycle = KILLNEXT ;
            else
                state->NextCycle = ABORT1 ;
            }
        break ;
```

【００８８】

【表３】

[表２から続く]

```
case STM3 :
    state->doutenSig = LOW ;
    if (state->abortSig || state->Aborted) {
        if (state->Processor & ARM_Abort7_ProD &&
            ((state->Aborted != ARMul_DataAbortV &&
            state->abortSig) ||
            (state->Aborted == ARMul_AddrExceptnV && (LSMNumRegs
            / 4) == 1)))
            state->NextCycle = KILLNEXT ;
        else
            state->NextCycle = ABORT1 ;
        if (!state->Aborted)
            state->Aborted = ARMul_DataAbortV ;
        }
    break ;

case UNDEF :
    PREFETCH(loaded,pc + instruction_size * 2) ;
    if (!IntPending(state)) {
        if (!state->cpaSig || !state->cpbSig) {
            UNDEF_UndefNotBounced ;
            }
        state->NcpiSig = HIGH ;
        ARMul_Abort(state,ARMul_UndefinedInstrV) ;
        }
    state->NextCycle = DESTPC1 ;
    break ;

case LDC1 :
    PREFETCH(loaded,pc + instruction_size) ;
case LDCBUSY :
    state->NextCycle = CoProNextState(state,LDC2,LDCBUSY) ;
    if (state->NextCycle == LDC2) {
        BUSUSEDINCPCN ;
        if (BIT(21))
            LSBase = state->Base ;
        }
        break ;

case LDC2 :
      if (state->cpaSig == HIGH && state->cpbSig == HIGH) {
        state->LSCActive = FALSE ;
        if (state->abortSig || state->Aborted) {
            state->NextCycle = ABORT1 ;
            }
        else
            state->NextCycle = NEWINSTR ;
        }
    else
        if (state->abortSig && !state->Aborted)
            state->Aborted = ARMul_DataAbortV ;
    break ;
```

【００８９】

【表４】

[表３から続く]

```
case STC1 :
    PREFETCH(loaded,pc + instruction_size * 2) ;
case STCBUSY :
    state->NextCycle = CoProNextState(state,STC2,STCBUSY) ;
    if (state->NextCycle == STC2) {
        BUSUSEDINCPCN ;
        if (BIT(21))
            LSBase = state->Base ;
        }
    break ;

case STC2 :
case STC3 :
    if (state->cpaSig == HIGH && state->cpbSig == HIGH) {
        if (state->abortSig || state->Aborted) {
            if (state->Processor & ARM_Abort7_Prop &&
                ((state->Aborted != ARMul_DataAbortV &&
                state->abortSig) ||
                (state->Aborted == ARMul_AddrExceptnV &&
                state->LastCycle == STC2)))
                state->NextCycle = KILLNEXT;
            else
                state->NextCycle = ABORT1 ;
            }
        else
            state->NextCycle = NEWINSTR ;
        state->LSCActive = FALSE ;
        state->doutenSig = LOW ;
        }
    else
        if (state->abortSig && !state->Aborted)
            state->Aborted = ARMul_DataAbortV ;
        break ;

case MCR1 :
    PREFETCH(loaded,pc + instruction_size * 2) ;
case MCRBUSY :
    state->NextCycle = CoProNextState(state,MCR2,MCRBUSY) ;
    if (state->NextCycle == MCR2) {

        BUSUSEDINCPCN ;
        }
    break ;

case MCR2 :
    state->doutenSig = LOW ;
    break ;

case MRC1 :
    PREFETCH(loaded,pc + instruction_size * 2) ;
case MRCBUSY :
    state->NextCycle = CoProNextState(state,MRC2,MRCBUSY) ;
    if (state->NextCycle == MRC2) {
        BUSUSEDINCPCN ;
        }
    break ;
```

【００９０】

【表５】

［表４から続く］

```
    case CDP1 :
        PREFETCH(loaded.pc + instruction_size * 2) ;
    case CDPBUSY :
        state->NextCycle = CoProNextState(state,NEWINSTR,CDPBUSY) ;
        if (state->NextCycle == NEWINSTR) {
            BUSUSEDN ;
            }
        break ;
        }
    }
state->LastCycle = state->NextCycle ;

switch (state->NextCycle) { /* In a cycle based world, a new    */
                            /* instruction is just an           */
                            /* (important) state in a state     */
    case NEWINSTR :         /* machine                          */
        switch (state->NextInstr) {
        case SEQ :
        case NONSEQ :
            /* Inc PC, and fall through */
            state->Reg[15] = state->Reg[15] + instruction_size ;

        case PCINCEDSEQ :
        case PCINCEDNONSEQ :
            pc = pc + instruction_size ; /* Advance the dummy pc */
            instr = decoded ;
            decoded = loaded ;
            break ;

        case RESUME :
#ifdef MODE32
            state->Reg[15] = state->Reg[15] + PCR15DIFFERENCE ;
#else
            state->Reg[15] = (state->Reg[15] + PCR15DIFFERENCE) &
                             R15PCBITS ;
#endif
            break ;

        default : /* The program counter has been changed */
            pc = state->Reg[15] ;

#ifdef CODE16
            state->Reg[15] = PCADDRESSBITS(pc) + PCR15DIFFERENCE ;
#else
            state->Reg[15] = (pc & R15PCBITS) + PCR15DIFFERENCE ;
#endif
            instr = decoded ;
            decoded = loaded ;

            break ;

        }
        NORMALCYCLE ;

    if ((state->Inted & 3) < 3) {
        if (!(state->Inted & 1))
            ARMul_Abort(state,ARMul_FIQV) ;
        else
            ARMul_Abort(state,ARMul_IRQV) ;
        break ;
        }

#else /* Instruction based */

instruction_size = INSTRUCTIONSIZE;
has_halfword_prop = state->Processor & ARM_Halfword_Prop;

if  (state->NextInstr < PRIMEPIPE) {
    decoded = state->decoded ;
    loaded = state->loaded ;
    pc = state->pc ;
    }
```

【００９１】

【表６】

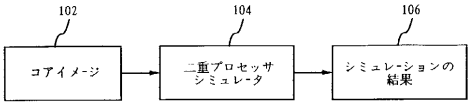
［付　録　Ｂ］

コアイメージ１０２の具体例

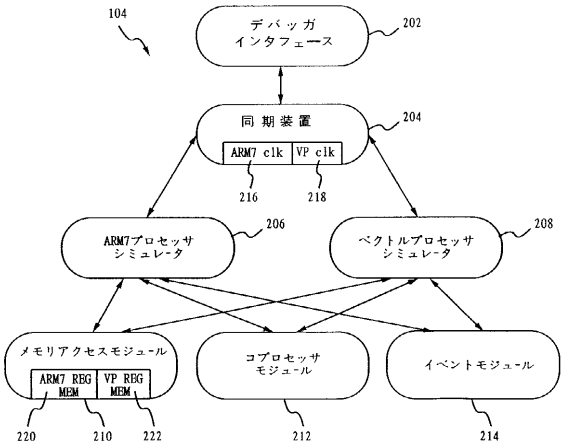
```
    /* ARM7 CODE SEGMENT */
A1.   /* ARM7 instructions */
A2.   STARTVP
A3.   /* other ARM7 instructions */
A4.   /* wait loop for VP */
    /* ARM7 DATA SEGMENT */
A5.   /* ARM7 data */
    /* VECTOR PROCESSOR CODE SEGMENT */
A6.   /* Vector processor instructions */
A7.   JOIN
    /* VECTOR PROCESSOR DATA SEGMENT */
A8.   /* Vector processor data */
```

【図面の簡単な説明】
【図１】本発明の実施の形態の信号プロセッサシミュレータの概略構成を示す図である。
【図２】図１の信号プロセッサシミュレータの二重プロセッサシミュレーションの構造を示す図である。
【図３】図１の信号プロセッサシミュレータの論理流れ図である。
【図４】図３の並列処理のシミュレーションブロックで遂行される処理過程を示す論理流れ図である。
【図５】図１の信号プロセッサシミュレータによりシミュレーションされるプロセッサのクロックを示す図である。
【図６】図４の処理過程よりクロックシミュレータの周波数を減少させるための、本発明の他の実施の形態による論理流れ図である。
【符号の説明】
１０２　コアイメージ
１０４　シミュレータ
１０６　シミュレーション結果
２０２　デバッガーインタフェース
２０４　同期装置
２０６　ＡＲＭ７プロセッサシミュレータ
２０８　ベクトルプロセッサシミュレータ
２１０　メモリアクセスモジュール
２１２　コプロセッサモジュール
２１４　イベントモジュール
２１６　メモリ位置ＡＲＭ７ｃｌｋ
２１８　メモリ位置ＶＰｃｌｋ
２２０　メモリ位置ＡＲＭ７　ＲＥＧ　ＭＥＭ
２２２　メモリ位置ＶＰ　ＲＥＧ　ＭＥＭ

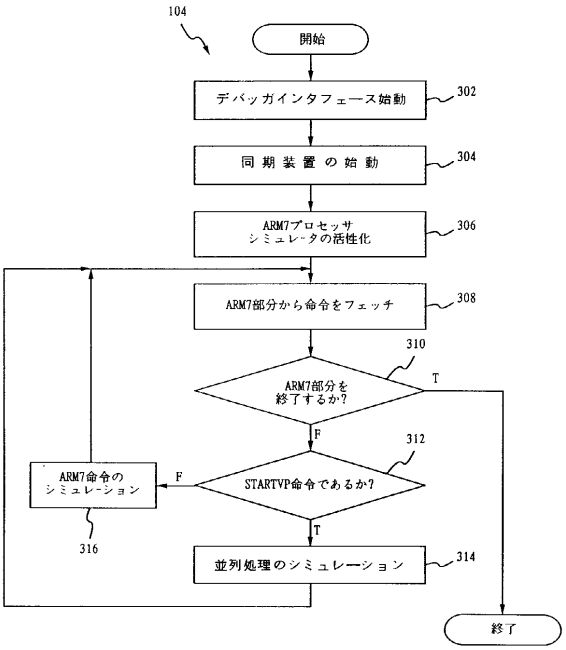
【図１】


102
104
106
コアイメージ
二重プロセッサ
シミュレータ
シミュレーションの
結果

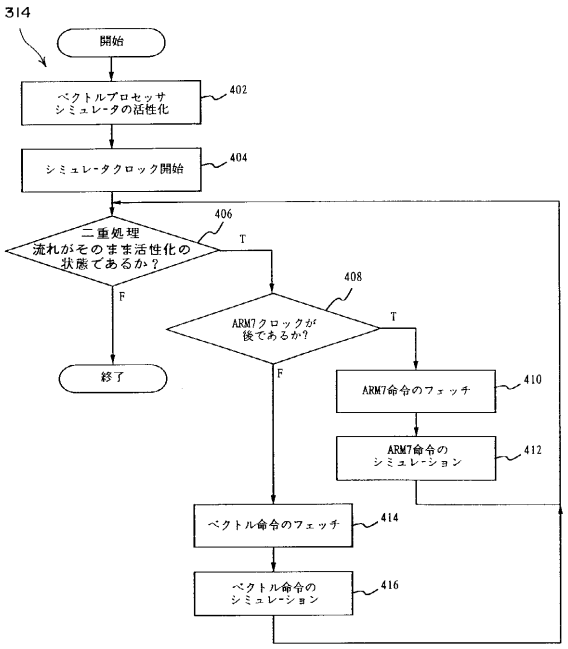
【図２】


104
デバッガ
インタフェース
202
同期装置
204
ARM7 clk
VP clk
216
218
ARM7プロセッサ
シミュレータ
206
ベクトルプロセッサ
シミュレータ
208
メモリアクセスモジュール
ARM7 REG
MEM
VP REG
MEM
220
210
222
コプロセッサ
モジュール
212
イベントモジュール
214

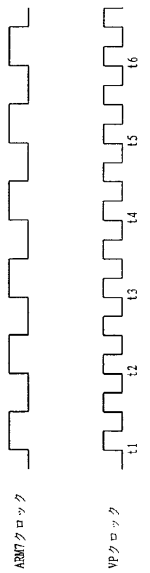
【図３】


104
開始
デバッガインタフェース始動
302
同期装置の始動
304
ARM7プロセッサ
シミュレータの活性化
306
ARM7部分から命令をフェッチ
308
310
ARM7部分を
終了するか?
T
F
312
STARTVP命令であるか?
F
ARM7命令の
シミュレーション
316
T
並列処理のシミュレーション
314
終了

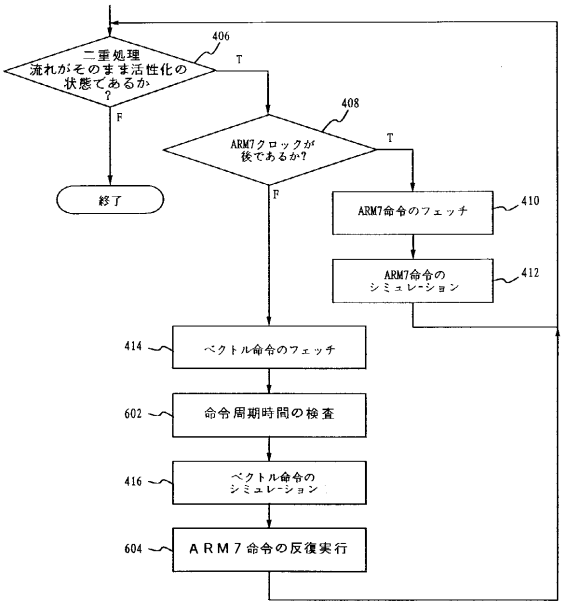
【図４】


314
開始
ベクトルプロセッサ
シミュレータの活性化
402
シミュレータクロック開始
404
406
二重処理
流れがそのまま活性化の
状態であるか?
T
F
408
ARM7クロックが
後であるか?
T
F
終了
ARM7命令のフェッチ
410
ARM7命令の
シミュレーション
412
ベクトル命令のフェッチ
414
ベクトル命令の
シミュレーション
416

【図５】


t1
t2
t3
t4
t5
t6
ARM7クロック
VPクロック

【図６】


406
二重処理
流れがそのまま活性化の
状態であるか
?
T
F
408
ARM7クロックが
後であるか?
T
F
終了
ARM7命令のフェッチ
410
ARM7命令の
シミュレーション
412
414
ベクトル命令のフェッチ
602
命令周期時間の検査
416
ベクトル命令の
シミュレーション
604
ＡＲＭ７命令の反復実行

フロントページの続き

(72)発明者　イアン　ジェイ　リッカーズ
アメリカ合衆国　カリフォルニア州　９５０５０　サンタクララ　ワーバートン　アベニュー７番　１７２１

合議体
審判長　赤川　誠一
審判官　宮司　卓佳
審判官　冨吉　伸弥

(56)参考文献　特開昭６４－０７６２３６号公報（ＪＰ，Ａ）
特開平０３－１５７７７９号公報（ＪＰ，Ａ）
特開平０７－１６０５３７号公報（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 11/26　　，G06F 11/28　340C，G06F 15/16　450D