

(19) 日本国特許庁 (JP)

(12) 特 許 公 報 (B2)

(11) 特許番号

特許第6059413号

(P6059413)

(45) 発行日 平成29年1月11日 (2017. 1. 11)

(24) 登録日 平成28年12月16日 (2016. 12. 16)

(51) Int. Cl.

F I

G O 6 F 9/30 (2006. 01)

G O 6 F 9/30 3 1 O B

G O 6 F 9/45 (2006. 01)

G O 6 F 9/44 3 2 2 F

請求項の数 55 (全 34 頁)

(21) 出願番号	特願2008-508300 (P2008-508300)	(73) 特許権者	507364838
(86) (22) 出願日	平成18年4月28日 (2006. 4. 28)		クアルコム、インコーポレイテッド
(65) 公表番号	特表2008-539485 (P2008-539485A)		アメリカ合衆国 カリフォルニア 9 2 1
(43) 公表日	平成20年11月13日 (2008. 11. 13)		2 1 サン ディエゴ モアハウス ドラ
(86) 国際出願番号	PCT/GB2006/001556		イブ 5 7 7 5
(87) 国際公開番号	W02006/114642	(74) 代理人	110000855
(87) 国際公開日	平成18年11月2日 (2006. 11. 2)		特許業務法人浅村特許事務所
審査請求日	平成21年4月28日 (2009. 4. 28)	(72) 発明者	アースラン、トクラル、サティ
審査番号	不服2013-9418 (P2013-9418/J1)		イギリス国、エディンバラ、グランジェ、
審査請求日	平成25年5月22日 (2013. 5. 22)		レルガス プレース 1 2
(31) 優先権主張番号	0508589.9	(72) 発明者	ミルワード、マーク、ジョン
(32) 優先日	平成17年4月28日 (2005. 4. 28)		イギリス国、ブリストル、ソーンベリー、
(33) 優先権主張国	英国 (GB)		ストリームリーズ、 コルン スクエア
(31) 優先権主張番号	0604428.3		1 0
(32) 優先日	平成18年3月6日 (2006. 3. 6)		
(33) 優先権主張国	英国 (GB)		最終頁に続く

(54) 【発明の名称】 再構成可能命令セル・アレイ

(57) 【特許請求の範囲】

【請求項 1】

従属および独立の両方のプログラム命令の第一のデータ経路を有するプログラム命令を実行するプロセッサであって、

相互接続ネットワークと、

それぞれが前記相互接続ネットワークに接続される異種の複数の命令セルであって、該命令セルの各々は少なくとも加算、減算、乗算、除算あるいはシフトの処理を行う該複数の命令セルと、

構成メモリへのインターフェースであって、前記構成メモリは、前記従属プログラム命令及び独立プログラム命令の前記第一のデータ経路の少なくとも1つから、前記命令セルの回路へのマッピングがそれぞれ符号化された構成命令を記憶するものである前記インターフェースと、

復号モジュールであって、受けた構成命令を復号して前記相互接続ネットワークおよび命令セルの少なくともいくつかの第二のデータ経路を構成し、これにより前記第一のデータ経路を前記命令セルの回路にマップした前記プログラム命令を実行する、復号モジュールと、を備え、

前記命令セルの少なくとも一つは、実行フロー制御命令セルを含み、前記実行フロー制御命令セルはプログラム・カウンタ及び前記インターフェースを管理して、クロック・サイクル毎に前記構成命令の一つを前記復号モジュールに与えるように構成されている、

プロセッサ。

10

20

【請求項 2】

前記復号モジュールは前記命令セルの少なくともいくつかを前記相互接続ネットワークを通して直列に接続することにより、前記相互接続ネットワークおよび命令セルの少なくともいくつかのデータ経路を構成する、請求項 1 記載のプロセッサ。

【請求項 3】

前記復号モジュールは更に、複数のプログラム・スレッドのデータ経路から前記命令セルの対応する複数の独立回路へのマッピングを符号化する構成命令を受け、更に構成命令を復号して前記相互接続ネットワークおよび命令セルの少なくともいくつかのデータ経路を構成し、これにより複数のプログラム・スレッドのデータ経路を前記命令セルの対応する複数の独立回路にマップしたプログラム・スレッドを互いに独立に同時に実行する、請求項 1 または 2 のいずれか記載のプロセッサ。

10

【請求項 4】

前記プロセッサはクロック・サイクルを有するクロック信号を与えるクロック・モジュールを更に備え、また前記復号モジュールは前記構成命令を復号して前記相互接続ネットワークおよび命令セルの少なくともいくつかのデータ経路をクロック・サイクル毎に構成する、請求項 1 から 3 のいずれか一項記載のプロセッサ。

【請求項 5】

前記クロック・モジュールは可変クロック・サイクルを与える、請求項 4 記載のプロセッサ。

【請求項 6】

20

前記クロック・モジュールはイネーブル信号を与える、請求項 4 または 5 のいずれか記載のプロセッサ。

【請求項 7】

前記イネーブル信号は命令セルのイネーブル入力に与えられる、請求項 6 記載のプロセッサ。

【請求項 8】

前記クロック・モジュールは複数のクロック信号を与える、請求項 4 から 7 のいずれか一項記載のプロセッサ。

【請求項 9】

前記複数のクロック信号の少なくともいくつかは複数の命令セルに別々に与えられる、請求項 8 記載のプロセッサ。

30

【請求項 10】

前記復号モジュールは更に構成命令を復号してクロック信号を構成する、請求項 4 から 9 のいずれか一項記載のプロセッサ。

【請求項 11】

前記復号モジュールは更に構成命令を復号してクロック・サイクルを構成する、請求項 10 記載のプロセッサ。

【請求項 12】

前記クロック信号の活動時間は構成される、請求項 10 または 11 のいずれか記載のプロセッサ。

40

【請求項 13】

前記クロック信号のデューティ・サイクルは構成される、請求項 10 から 12 のいずれか一項記載のプロセッサ。

【請求項 14】

前記イネーブル信号は前記実行フロー制御命令セルのイネーブル入力に与えられる、請求項 6 記載のプロセッサ。

【請求項 15】

前記構成セルの少なくとも 1 つはその出力をクロック・サイクル毎に変えるレジスタ命令セルを含む、請求項 14 に記載のプロセッサ。

【請求項 16】

50

前記イネーブル信号は前記レジスタ命令セルのイネーブル入力に与えられる、請求項 1 5記載のプロセッサ。

【請求項 1 7】

前記イネーブル信号は前記実行フロー制御命令セルのイネーブル入力と前記レジスタ命令セルのイネーブル入力の両方に与えられる、請求項 1 5または請求項 1 6のいずれか記載のプロセッサ。

【請求項 1 8】

前記プロセッサは更に少なくとも1つの入力／出力ポートを備え、また前記命令セルの少なくとも1つは前記少なくとも1つの入力／出力ポートにアクセスする入力／出力レジスタ命令セルを含む、請求項 1 から請求項 1 7のいずれか一項記載のプロセッサ。

10

【請求項 1 9】

前記プロセッサは更に少なくとも1つのスタックを備え、また前記命令セルの少なくとも1つは前記少なくとも1つのスタックにアクセスするスタック・レジスタ命令セルを含む、請求項 1 から請求項 1 8のいずれか一項記載のプロセッサ。

【請求項 2 0】

前記命令セルの少なくとも1つはデータ・メモリ位置にアクセスするメモリ命令セルを含む、請求項 1 から請求項 1 9のいずれか一項記載のプロセッサ。

【請求項 2 1】

前記データ・メモリ位置はマルチポート・メモリ位置を含む、請求項 2 0記載のプロセッサ。

20

【請求項 2 2】

メモリ命令セルの1対以上は前記クロック・サイクルの周波数より高い周波数でクロックするデータ・メモリ位置の1対以上に同じクロック・サイクル内にアクセスする、請求項 2 0または請求項 2 1のいずれか記載のプロセッサ。

【請求項 2 3】

前記復号モジュールは更に構成命令を復号して、その前の構成命令から復号されたクロック信号の構成に依存するクロック信号を構成する、請求項 4 から請求項 2 2のいずれか一項記載のプロセッサ。

【請求項 2 4】

前記命令セルの少なくとも1つは命令セルの間のデータの経路を選択するマルチプレクサ命令セルを含む、請求項 1 から請求項 2 3のいずれか一項記載のプロセッサ。

30

【請求項 2 5】

前記命令セルの少なくとも1つは組合せ論理命令セルを含む、請求項 1 から請求項 2 4のいずれか一項記載のプロセッサ。

【請求項 2 6】

命令セルの機能はプログラム命令に対応する、請求項 1 から請求項 2 5のいずれか一項記載のプロセッサ。

【請求項 2 7】

命令セルの前記回路は1つの実行フロー制御命令セルを含む命令セルの集合を含む、請求項 1から請求項 2 6のいずれか一項記載のプロセッサ。

40

【請求項 2 8】

請求項 1 から請求項 2 7のいずれか一項記載のプロセッサの構成命令を生成するコンパイラであって、

プログラム命令を受けて前記プログラム命令のデータ経路を識別するフロントエンド・モジュールと、

従属および独立の両方のプログラム命令のデータ経路を前記命令セルの回路としてマップするスケジューリング・モジュールと、

命令セルの回路の前記構成を構成命令として符号化した前記構成命令を出力するマッピング・モジュールと、を含むコンパイラ。

50

【請求項 2 9】

前記プログラム命令は高水準コンパイラから出力されるアセンブラ・コードを含む、請求項 2 8記載のコンパイラ。

【請求項 3 0】

前記プログラム命令は高水準言語のソース・コードを含む、請求項 2 8記載のコンパイラ。

【請求項 3 1】

前記スケジューリング・モジュールは更にその相互依存性に従って前記プログラム命令をスケジュールする、請求項 2 8 から請求項 3 0のいずれか一項記載のコンパイラ。

【請求項 3 2】

前記フロントエンド・モジュールおよびスケジューリング・モジュールの少なくとも 1 つは更に一時的結果を記憶するためのレジスタへのアクセスを命令セルの間の相互接続にマップする、請求項 2 8 から請求項 3 1のいずれか一項記載のコンパイラ。

【請求項 3 3】

前記スケジューリング・モジュールは更に単一クロック・サイクル内に実行する前記プログラム命令のデータ経路をマップする、請求項 2 8 から請求項 3 2のいずれか一項記載のコンパイラ。

【請求項 3 4】

前記スケジューリング・モジュールは更に複数のプログラム・スレッドのデータ経路を前記命令セルの対応する複数の同時の回路としてマップする、請求項 2 8 から請求項 3 3のいずれか一項記載のコンパイラ。

【請求項 3 5】

前記スケジューリング・モジュールは更に操作連鎖を用いて前記データ経路をマップする、請求項 2 8 から請求項 3 4のいずれか一項記載のコンパイラ。

【請求項 3 6】

前記スケジューリング・モジュールは更に単一クロック・サイクルをスケジュール中は操作連鎖を用いる、請求項 2 8 から請求項 3 5のいずれか一項記載のコンパイラ。

【請求項 3 7】

前記スケジューリング・モジュールは更に複数の実行可能リストの各実行可能リストを処理中は操作連鎖を用いる、請求項 2 8 から請求項 3 6のいずれか一項記載のコンパイラ。

【請求項 3 8】

前記スケジューリング・モジュールは更にレジスタ割当てを用いて前記データ経路をマップする、請求項 2 8 から請求項 3 7のいずれか一項記載のコンパイラ。

【請求項 3 9】

前記スケジューリング・モジュールは更に単一クロック・サイクルをスケジュール中はレジスタ割当てを用いる、請求項 3 8記載のコンパイラ。

【請求項 4 0】

前記スケジューリング・モジュールは更に複数の実行可能リストの各実行可能リストを処理中はレジスタ割当てを用いる、請求項 3 8 か請求項 3 9 に記載のコンパイラ。

【請求項 4 1】

前記スケジューリング・モジュールは更に複数のスケジューリング・アルゴリズムを用いて前記データ経路をマップする、請求項 2 8 から請求項 4 0のいずれか一項記載のコンパイラ。

【請求項 4 2】

前記スケジューリング・モジュールは更に前記複数のスケジューリング・アルゴリズムの 1 つから選択した出力を用いて前記データ経路をマップする、請求項 4 1項記載のコンパイラ。

【請求項 4 3】

コンピュータのプログラム命令をプロセッサにより実行する方法であって、

10

20

30

40

50

従属命令及び独立命令のデータ経路を有するプログラム命令内のデータ経路をコンパイラ中のモジュールにより識別するステップと、

プログラム・カウンタと構成命令を格納する構成メモリとを管理して、各クロック・サイクル毎に前記構成命令の一つを実行するステップであって、前記構成命令は、従属命令のデータ経路と独立命令のデータ経路から複数の異種の命令セルの一つ以上の回路へのマッピングが符号化された、前記実行するステップと、

前記構成命令に基づいて、相互接続ネットワーク及び異種の複数の命令セルの少なくともいくつかを接続するステップと、

該命令セルの各々は少なくとも加算、減算、乗算、除算あるいはシフトの処理を行い、前記命令セルの少なくともいくつかを前記相互接続ネットワークを通して直列に接続することにより各命令セルを前記相互接続ネットワークに接続し、これにより従属命令および独立命令のデータ経路をマッピングするステップと、を含むプログラム命令を実行する方法。

10

【請求項 4 4】

クロック・サイクルを有するクロック信号を前記命令セルの少なくとも 1 つに与えるステップを更に含み、前記相互接続ネットワークおよび異種の複数の命令セルの少なくともいくつかを構成する際はクロック・サイクル毎に行う、請求項 4 3 記載のプログラム命令を実行する方法。

【請求項 4 5】

前記相互接続ネットワークおよび異種の複数の命令セルの少なくともいくつかを構成する際はその後のクロック・サイクルを構成するステップを更に含む、請求項 4 3 または請求項 4 4 に記載のプログラム命令を実行する方法。

20

【請求項 4 6】

前記相互接続ネットワークおよび異種の複数の命令セルの少なくともいくつかを構成する際は、前記クロック・サイクルの活動時間が構成される、請求項 4 4 または請求項 4 5 に記載のプログラム命令を実行する方法。

【請求項 4 7】

前記相互接続ネットワークおよび異種の複数の命令セルの少なくともいくつかを構成する際は、前記クロック信号のデューティ・サイクルが構成される、請求項 4 4 から請求項 4 6 に記載のプログラム命令を実行する方法。

30

【請求項 4 8】

前記相互接続ネットワークおよび命令セルの少なくともいくつかを構成する前記ステップは、

- ・ 従属および独立の両方のプログラム命令のデータ経路を前記命令セルの回路としてマップするステップと、
- ・ 命令セルの前記回路の構成を構成命令として符号化するステップと、
- ・ 前記構成命令を復号して前記相互接続ネットワークおよび命令セルの少なくともいくつかを構成するステップと、

を含む、請求項 4 3 から請求項 4 7 のいずれか一項記載のプログラム命令を実行する方法。

40

【請求項 4 9】

マップする前記ステップはその相互依存性に従って前記プログラム命令をスケジュールするステップを含む、請求項 4 8 記載のプログラム命令を実行する方法。

【請求項 5 0】

マップする前記ステップは一時的結果を記憶するためのレジスタへのアクセスを命令セルの間の相互接続にマップするステップを含む、請求項 4 8 または請求項 4 9 に記載のプログラム命令を実行する方法。

【請求項 5 1】

マップする前記ステップは単一クロック・サイクル内に実行するプログラム命令のデータ経路をマップするステップを含む、請求項 4 8 から請求項 5 0 のいずれか一項記載のプ

50

プログラム命令を実行する方法。

【請求項 5 2】

コンピュータにロードすると請求項 2 8 から請求項 4 2 のいずれか一項記載のコンパイラを構成するプログラム命令を含むコンピュータ・プログラム。

【請求項 5 3】

記憶媒体上で実現され、コンピュータ・メモリ内に記憶され、または電氣的または光学的なキャリア信号で運ばれる、請求項 5 2 記載のコンピュータ・プログラム。

【請求項 5 4】

請求項 4 3 から請求項 5 1 のいずれか一項記載の方法をコンピュータに実行させるプログラム命令を含むコンピュータ・プログラム。

10

【請求項 5 5】

前記コンピュータ・プログラムは、前記コンピュータに結合された記憶媒体上で実現されるか、前記コンピュータの内部メモリ内に記憶されるか、あるいは電氣的または光学的なキャリア信号で伝送可能である請求項 5 4 記載のコンピュータ・プログラム。

【発明の詳細な説明】

【技術分野】

【0001】

本発明はコンピュータ・プロセッサに関するもので、特に再構成可能プロセッサに関するものである。

【背景技術】

20

【0002】

現在の半導体デバイスの構造は将来のデータ処理および処理能力の要求に対処できないので、将来の新しいハードウェア構造についてのニーズが数年前から認識されている。代表的な例は大量のオーディオおよびビデオ・データを処理する必要のある次世代ネットワーク用の移動体装置である。このため、電力消費およびシリコン面積に関して効率を犠牲にしても高い処理能力を保持するためにプロセッサの性能に新たな要求が加わった。ハードウェアも将来の変化する標準に対して高い適応性を備える必要がある。

【0003】

再プログラム可能性および柔軟性は、製作後にシステムを変更することができるので設計コストを下げる主な要因である。標準 C/C++ などの高水準プログラム記述によりプログラム可能なこの柔軟性を有することは設計サイクルを減らしまた大きなシステムを実現するのに重要である。

30

【0004】

すでに確立されている方法には、書き替え可能ゲート・アレイ (FPGA)、デジタル信号処理プロセッサ (DSP)、および特殊用途向け集積回路 (ASIC) 構造などがある。しかしこれらは全て欠点を有する。すなわち、FPGA は電力消費が非常に大きく、ASIC は製作後の非反復エンジニアリング・コストが高くまた柔軟性が低く、DSP は処理能力が低いなどである。超長命令語 (VLIW) DSP 構造は並列処理において優れている。しかし、これらは一般的なプログラムに見られるように命令レベル並列処理 (ILP) の量が限られるという制約がある。これまで研究機関から提案された多くの方法は、主として、柔軟性および大量の並列処理に関して再構成可能型計算枠組により高性能を実現するものであった。これらの方法は計算能力および柔軟性では顕著な成果を達成したが、電力節約が十分でない、またはプログラムが非常に難しいという欠点がある。

40

【0005】

いくつかの構造は、RISC プロセッサと粗粒度の再構成可能なアレイとの結合を試みている。例えば、モルフォシス (Morphosys) (H. Singh 他「モルフォシス: データ並列および計算強化アプリケーションのための統合された再計算可能システム (Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications)」, IEEE Trans.

50

on Comp. , 49 (5) : 465-481, 2000年5月) や、SMEXPP (V. Baumgarte 他「PACT XAPP-A 自己再構成可能データ処理構造 (PACT XAPP-A Self-reconfigurable data processing architecture)」、Proc. , ERS A01) などである。しかしこれらは特殊な特注の低水準言語を用いるのでアレイのプログラミングが困難である。また、プログラミングが難しいことやアレイとプロセッサとの間のデータ転送量が大きいなどの、従来のプロセッサとアレイとの間の緩い結合に起因する他の制限もある。

【0006】

ADRES 構造 (Bingfen Mei 他「ADRES: 密に結合された VLIW プロセッサおよび粗粒度の再構成可能マトリックス (ADRES: An Architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix)」、Proc. , FPL '03) は、メモリおよびレジスタ・ファイルを共用することによりプロセッサと再構成可能構造とを密に結合するもので、高水準 C 言語を用いるのでプログラミング・モデルは簡単である。RaPID 構造 (C. Ebeling 他「RaPID 再構成可能構造を用いた OFDM 受信機の実現 (Implementing an OFDM Receiver on the RaPID Reconfigurable Architecture)」、Proc. , FPL '03) は高水準 RaPID-C 言語を用いてプログラムすることができるが、この言語は ANSI-C と互換性がなく、並列操作を手動でスケジュールする必要がある。

【0007】

Chameleon/Montium 双対プロセッサの方法 (Paul M. Heysters 他「エネルギー効率と柔軟性と性能の間のバランス (Balancing between Energy-Efficiency, Flexibility and Performance)」、Proc. ERS A '03) および Elixent (英国ブリストル) の D-Fabrix (TM) は、複数の ALU を相互接続して特殊な C コンパイラを用いてデータ経路を作るので、時間のかかる計算の速度が上がる。Elixent (TM) の場合は、プログラミングは Handel-C により行う。設計者がコード内で並列処理を宣言しなければならないのでこれは HDL に近い。

【発明の開示】

【発明が解決しようとする課題】

【0008】

したがって、提案されている再構成可能なハードウェア・インフラストラクチャは、コストが低く、設計時間が短く、電力消費が少なく、性能が高い、というニーズに十分応えていないのでどれも広く用いられていない。

【0009】

プログラミング環境では、スレッドは、理想的には別個のデータの集合において「並列に」動作する、相関のない独立のタスクである。マルチスレッド・プログラムを実行する際に、スレッドを実行するためのまたは相互に同期させるための予め決められたまたは既知の順序はない。プログラムを分析することにより既知のパターンを構築することができるのであれば、なぜこれをマルチスレッド (MT) プログラムとして実現するのか疑問である。

【0010】

マルチスレッド・プログラムは、コンパイル時には分からない動的な情報に依存する。そのため、マルチスレッド・プログラムの実行フローを静的に分析して、全ての活動的なサポートされたスレッドを満足させるように割当てを決定することは非常に難しい。各スレッドは他の活動的なスレッドについての事前の知識なしに独立にマップされる。

【0011】

従来、マルチスレディングは完全に分離された実行コアを用いている。従来、単一コア

10

20

30

40

50

でのMTはハードウェア資源をタイム・スライスすることにより行っている。これは時間的マルチスレディングと呼ぶことが多い。スーパースカラ・コンピュータ、VLW（超長命令語）、および再構成可能計算機の出現により、単一コア内でも真のMTが可能になった。本質的に、並列構造は多数の計算資源および／または相互接続資源（再構成可能型計算枠組の場合）を用いることに依存する。これらの資源を無駄なく働かせることは挑戦的なタスクである。

【0012】

多くの並列構造の問題はコアの利用度が低いことである。この影響を減らすために、従来の方法は単一コア内で同時マルチスレディング（SMT）を用いている。通常、SMTはコアの実行フロー制御機構の拡張であって、2つ以上の同時スレッドによりコアの資源の多くを共用する。SMTの実現方法は実施の形態によって非常に変わるが、所定のクラスのデバイスについての概要を定義することはできる。

10

【0013】

資源を共用するには、SMTは構造の重要な構成要素を何らかの形で複写する必要がある。これは一般に次のものを含む。

1. フロー制御機構
2. 構造状態。

【0014】

コアの利用度を高くするだけでなく、SMTはキャッシュ待ち時間と取出しおよび復号待ち時間とに有用な操作を行うことによりこれらの時間を隠すことができる。1つのスレッドの機能停止条件があっても、他のスレッドを走らせて資源を無駄なく働かせることができる。

20

【0015】

再構成可能プロセッサ上で走る全てのアプリケーションを記述するには高水準言語を用いることが望ましい。しかし、従来のソフトウェアは本来逐次的である。これらの高水準言語は従来のCPU用に開発されたものなので、適したスケジューリング・アルゴリズムは逐次コード用である。これは、本来なら逐次のコードを用いて、利用可能な並列処理を取り出して利用すると言う挑戦的なタスクを行う。高い抽象レベルからのアルゴリズムの実現を自動化して再構成可能な計算システム上で走らせる設計フローは、汎用ソフトウェア・コードを高度に並列の再構成可能デバイス上で変換して資源を最大限に使用することのできる効率的なスケジューリング・アルゴリズムを必要とする。

30

【0016】

従来のリスト・スケジューリングは、命令セルに基づく構造において直接用いることができない。なぜなら、

- 1) 従属命令の並列処理を扱わない、
 - 2) スケジューリング・アルゴリズムの中でレジスタ割当てを考慮しない、
 - 3) 再構成可能な機能ユニットの時間効果および経路選択相互接続遅れを考慮に入れないので、再構成可能デバイス内のデータ経路遅れを変える可能性がある、
- からである。

【課題を解決するための手段】

40

【0017】

本発明の1つの態様の目的は、コストが低く、設計時間が短く、電力消費が少なく、性能が高いプロセッサを提供することである。

【0018】

本発明の第1の態様では、従属および独立の両方のプログラム命令のデータ経路を有するプログラム命令を実行するプロセッサを提供する。このプロセッサは、

- ・ 相互接続ネットワークと、
- ・ それぞれが相互接続ネットワークに接続される異質の複数の命令セルと、
- ・ 復号モジュールであって、従属プログラム命令のデータ経路と独立プログラム命令のデータ経路の少なくとも1つから命令セルの回路へのマッピングをそれぞれ符号化する構

50

成命令を受け、更に構成命令を復号して相互接続ネットワークおよび命令セルの少なくともいくつかを構成し、これによりデータ経路を命令セルの回路にマップしたプログラム命令を実行する、復号モジュールと、
を備える。

【0019】

好ましくは、復号モジュールは命令セルの少なくともいくつかを相互接続ネットワークを通して直列に接続することにより、相互接続ネットワークおよび命令セルの少なくともいくつかを構成する。

【0020】

好ましくは、復号モジュールは更に、複数のプログラム・スレッドのデータ経路から命令セルの対応する複数の独立回路へのマッピングを符号化する構成命令を受け、更に構成命令を復号して相互接続ネットワークおよび命令セルの少なくともいくつかを構成し、これにより複数のプログラム・スレッドのデータ経路を命令セルの対応する複数の回路にマップしたプログラム・スレッドを互いに独立に同時に実行する。

【0021】

好ましくは、プロセッサはクロック・サイクルを有するクロック信号を与えるクロック・モジュールを更に備え、また復号モジュールは構成命令を復号して、相互接続ネットワークおよび命令セルの少なくともいくつかをクロック・サイクル毎に構成する。

好ましくは、クロック・モジュールは可変クロック・サイクルを与える。

好ましくは、クロック・モジュールはイネーブル信号を与える。

好ましくは、イネーブル信号は命令セルのイネーブル入力に与えられる。

好ましくは、クロック・モジュールは複数のクロック信号を与える。

【0022】

好ましくは、複数のクロック信号の少なくともいくつかは複数の命令セルに別々に与えられる。

好ましくは、復号モジュールは更に構成命令を復号してクロック信号を構成する。

好ましくは、復号モジュールは更に構成命令を復号してクロック・サイクルを構成する。

。

好ましくは、クロック信号の活動期間は構成される。

好ましくは、クロック信号のデューティ・サイクルは構成される。

【0023】

好ましくは、プロセッサは更にプログラム・カウンタと構成メモリへのインターフェースとを備え、構成メモリは構成命令を記憶し、命令セルの少なくとも1つはプログラム・カウンタおよびインターフェースを管理して、クロック・サイクル毎に構成命令の1つを復号モジュールに与える実行フロー制御命令セルを含む。

好ましくは、イネーブル信号は実行フロー制御命令セルのイネーブル入力に与えられる。

。

好ましくは、命令セルの少なくとも1つはその出力をクロック・サイクル毎に変えるレジスタ命令セルを含む。

好ましくは、イネーブル信号はレジスタ命令セルのイネーブル入力に与えられる。

好ましくは、イネーブル信号は実行フロー制御命令セルのイネーブル入力とレジスタ命令セルのイネーブル入力の両方に与えられる。

【0024】

好ましくは、プロセッサは更に少なくとも1つの入力／出力ポートを備え、また命令セルの少なくとも1つは少なくとも1つの入力／出力ポートにアクセスするための入力／出力レジスタ命令セルを含む。

好ましくは、プロセッサは更に少なくとも1つのスタックを備え、また命令セルの少なくとも1つは少なくとも1つのスタックにアクセスするスタック・レジスタ命令セルを含む。

好ましくは、命令セルの少なくとも1つはデータ・メモリ位置にアクセスするためのメ

10

20

30

40

50

メモリ命令セルを含む。

好ましくは、データ・メモリ位置はマルチポート・メモリ位置を含む。

【0025】

好ましくは、メモリ命令セルの1対以上はクロック・サイクルの周波数より高い周波数でクロックするデータ・メモリ位置の1対以上に同じクロック・サイクル内にアクセスする。

好ましくは、復号モジュールは更に構成命令を復号して、その前の構成命令から復号されたクロック信号の構成に依存するクロック信号を構成する。

好ましくは、命令セルの少なくとも1つは命令セルの間のデータの経路を選択する多重命令セルを含む。

一般に命令セルの少なくとも1つは組合せ論理命令セルを含む。

好ましくは、命令セルの機能はプログラム命令に対応する。

好ましくは、命令セルの回路は1つの実行フロー制御命令セルを含む命令セルの集合を含む。

【0026】

本発明の第2の態様では、第1の態様のプロセッサの構成命令を生成するコンパイラを提供する。このコンパイラは、

- ・ プログラム命令を受けてプログラム命令のデータ経路を識別するフロントエンド・モジュールと、
 - ・ 従属および独立の両方のプログラム命令のデータ経路を命令セルの回路としてマップするスケジューリング・モジュールと、
 - ・ 命令セルの回路の構成を構成命令として符号化した構成命令を出力するマッピング・モジュールと、
- を含む。

【0027】

好ましくは、プログラム命令は高水準コンパイラから出力されるアセンブラ・コードを含む。

または、プログラム命令は高水準言語のソース・コードを含む。

好ましくは、スケジューリング・モジュールは更にその相互依存性に従ってプログラム命令をスケジュールする。

好ましくは、フロントエンド・モジュールおよびスケジューリング・モジュールの少なくとも1つは更に一時的結果を記憶するためのレジスタへのアクセスを命令セルの間の相互接続にマップする。

好ましくは、スケジューリング・モジュールは更に単一クロック・サイクル内に実行するプログラム命令のデータ経路をマップする。

【0028】

好ましくは、スケジューリング・モジュールは更に複数のプログラム・スレッドのデータ経路を命令セルの対応する複数の同時の回路としてマップする。

好ましくは、スケジューリング・モジュールは更に操作連鎖を用いてデータ経路をマップする。

好ましくは、スケジューリング・モジュールは更に単一クロック・サイクルをスケジュール中は操作連鎖を用いる。

好ましくは、スケジューリング・モジュールは更に複数の実行可能リストの各実行可能リストを処理中は操作連鎖を用いる。

【0029】

好ましくは、スケジューリング・モジュールは更にレジスタ割当てを用いてデータ経路をマップする。

好ましくは、スケジューリング・モジュールは更に単一クロック・サイクルをスケジュール中はレジスタ割当てを用いる。

好ましくは、スケジューリング・モジュールは更に複数の実行可能リストの各実行可能

10

20

30

40

50

リストを処理中はレジスタ割当てを用いる。

好ましくは、スケジューリング・モジュールは更に複数のスケジューリング・アルゴリズムを用いてデータ経路をマップする。

好ましくは、スケジューリング・モジュールは更に複数のスケジューリング・アルゴリズムの1つから選択した出力を用いてデータ経路をマップする。

【0030】

本発明の第3の態様では、プログラム命令を実行する方法を提供する。この方法は、

- ・ 従属および独立の両方の命令のデータ経路を有するプログラム命令内のデータ経路を識別するステップと、

- ・ 相互接続ネットワークおよび異種の複数の命令セルの少なくともいくつかを構成し、命令セルの少なくともいくつかを相互接続ネットワークを通して直列に接続することにより各命令セルを相互接続ネットワークに接続し、これにより従属および独立の両方の命令のデータ経路をマップしたプログラム命令を実行するステップと、を含む。

10

【0031】

好ましくは、この方法は更にクロック・サイクルを有するクロック信号を命令セルの少なくとも1つに与えるステップを含み、命令セルを構成するステップはクロック・サイクル毎に実行する。

好ましくは、構成するステップはその後のクロック・サイクルを構成するステップを更に含む。

20

好ましくは、クロック・サイクルの活動期間は構成される。

好ましくは、クロック・サイクルのデューティ・サイクルは構成される。

【0032】

好ましくは、相互接続ネットワークおよび命令セルの少なくともいくつかを構成するステップは、

- ・ 従属および独立の両方のプログラム命令のデータ経路を命令セルの回路としてマップするステップと、

- ・ 命令セルの回路の構成を構成命令として符号化するステップと、

- ・ 構成命令を復号して相互接続ネットワークおよび命令セルの少なくともいくつかを構成するステップと、を含む。

30

【0033】

好ましくは、マップするステップはその相互依存性に従ってプログラム命令をスケジュールするステップを含む。

好ましくは、マップするステップは一時的結果を記憶するためのレジスタへのアクセスを命令セルの間の相互接続にマップするステップを含む。

好ましくは、マップするステップは単一クロック・サイクル内に実行するプログラム命令のデータ経路をマップするステップを含む。

【0034】

本発明の第4の態様では、コンピュータにロードすると第2の態様のコンパイラを構成するプログラム命令を含むコンピュータ・プログラムを提供する。

40

好ましくは、第4の態様のコンピュータ・プログラムは、記憶媒体上で実現され、コンピュータ・メモリ内に記憶され、または電氣的または光学的なキャリア信号で運ばれる。

本発明の第5の態様では、第3の態様をコンピュータに実行させるプログラム命令を含むコンピュータ・プログラムを提供する。

好ましくは、第5の態様のコンピュータ・プログラムは、記憶媒体上で実現され、コンピュータ・メモリ内に記憶され、または電氣的または光学的なキャリア信号で運ばれる。

【実施例】

【0035】

ここで再構成可能命令セル・アレイ (RICA) と呼ぶ本発明の1つの実施の形態につ

50

いて以下に説明する。

図1を参照すると、RICA構造のコア要素1は、プログラマブル・スイッチのネットワーク4を通して命令セル(IC)2を相互接続してデータ経路を作る。CPU構造と同様に、この実施の形態では、IC2および相互接続4の構成はサイクル毎に変更して命令の異なるブロックを実行することができる。図1に示すように、RICAはハーバード構造CPUと同じであって、プログラム(構成)メモリ6とデータ・メモリ8とは別個である。

【0036】

RICAの場合は、処理するデータ経路は相互接続可能なICの再構成可能なコアであって、構成メモリ6は、復号モジュール11を介して相互接続内部のICおよびスイッチの両方を制御する構成命令10(すなわち、ビット)を含む。データ・メモリ8とのインターフェースはMEMセル12が形成する。多数のこれらのセルは同じクロック・サイクル中に複数のメモリ位置から効率的に同時に(時間多重化)読取りおよび書込みを行うことができる。これは、データ・メモリ上の複数のポートを用いることにより、また構成可能コアより高速でクロックすることにより行うことができる。更に、いくつかの特殊なREG IC14を入出力ポート16にマップして外部環境とインターフェースすることができる。

【0037】

再構成可能RICAコア1の特徴は、設計時に完全にカスタマイズできることと、アプリケーションの要求に従って設定できることである。これは、システムのビット幅(この実施の形態では4ビットから64ビットまで任意に設定できる)や、アレイの柔軟性(ICの選択および展開される相互接続により設定する)などのオプションを含む。またこれらのパラメータは、達成可能な並列処理の範囲と、面積や最大処理能力や電力消費などのデバイス特性に影響を与える。RICAコアを含むチップを製作した後で、システムは汎用プロセッサ(GPP)と同様に任意のコードを実行するよう容易に再プログラムすることができる。

【0038】

RICA内のICアレイは種類が異なり、各セルは少数の操作に限定される。例えば、

ADD	加算、減算
MUL	乗算(符号付きまたは符号なし)
DIV	割り算(符号付きまたは符号なし)
REG	レジスタ
I/O REG	外部入出力ポートにアクセスするためのレジスタ
MEM	データ・メモリからの読取り/書込み
SHIFT	シフト操作
LOGIC	論理演算(XOR、AND、ORなど)
COMP	データ比較
JUMP	分岐(およびシーケンサ機能)

などである。

【0039】

このため、アレイ内のセルの総数を増やすと一層多くの並列計算を行うことができる。なぜなら、かかる小さなセルを追加することのオーバーヘッドは相互接続が占める追加の面積に関係するだけからである。また種類の異なるセルを用いるので、頻度の高い操作に用いるICを追加することによりアプリケーション領域に合うようにアレイを調整することができる。各ICに1命令をマップしてよい。アセンブリ命令と同様に、セルは2入力および1出力を有してよい。これにより、一層効率的な相互接続構造を作ることができるし、また必要な構成ビットの数を減らすことができる。この実施の形態のために開発された命令セルはOpenRISCおよびARM7(TM)(英国ケンブリッジのアーム社)などのGPPに見られるのと同じ命令集合をサポートする。ただし、命令セルはこれらのプロセッサからの命令集合をサポートすることに限定されない。したがってこの配列では

、R I C Aは任意のG P P／D S Pシステムとバイナリ互換性を持つことができる。

【0040】

上記のように、レジスタ18は標準の命令セルとして定義してよい。この配列では、レジスタ・メモリ要素は独立に動作するようにアレイ内に分散してよい。これは高度の並列処理を行うのに絶対必要である。R I C Aアレイをプログラムするには、プログラムのアセンブリ・コードをスライスして単一可変長クロック・サイクル内に実行する命令のブロックにする。一般に、本来は逐次G P P用に生成されたこれらの命令は、中間結果を一時的に記憶するためのレジスタへのアクセスを含む。R I C A構造の場合は、これらの読取り／書込み操作は単に配線に変換される。これによりレジスタの使用効率が高まる。レジスタのこの配置を用いることにより、R I C Aは操作のパイプラインをプログラム可能にするので、長い組合せ計算を複数の可変長クロック・サイクルに容易に分解することができる。レジスタを分散すると、各レジスタにスタックを割り当てて分散スタック方式を形成することができる。

10

【0041】

特殊なI Cは実行フロー制御すなわちJ U M Pセル20を含む。これはC P Uに見られる命令コントローラと同様に、プログラム・カウンタ22および構成メモリ6とのインターフェースを管理する。

【0042】

別の特殊な構成要素は再構成レート・コントローラ(R R C)24クロック・モジュールである。R R Cはクロック発生器26から信号を受けてイネーブル信号を生成する。この信号はレジスタI C 18および実行フロー制御I C 20のイネーブル入力に送られて、可変クロック・サイクルを与える。「クロック・ゲーティング」は実施の形態に特有であり、使用できる多数の他の周知の方法がある(非同期イネーブル信号など)。普通のC P Uでは、プロセッサをクロックできる最高クロック周波数はプログラム可能なデータ経路内の最長の可能な遅れにより決定する。

20

【0043】

例えば、C P Uが乗算器(加算などの演算より実行時間がかなり長い)を有する場合は、最高クロック周波数はそれを演算するのに十分な時間を与えなければならない。問題は、かかるクロックを用いた場合に、プロセッサの命令サイクルは加算器だけに用いられるためにプロセッサは必要以上に長い時間待たされることになり、このために全体の最大処理能力が制限されることである。従来のC P Uではこの問題を解決するために、乗算器が必要とする周波数より高い周波数でC P Uをクロックし、同時に乗算器をパイプライン化する。このため、実行に複数サイクル必要になる。

30

【0044】

R I C A構造でも同様の問題が起こる。なぜなら、柔軟性が大きいために多くのレベルの計算を行うデータ経路が作られて、多くの遅れが必要になる可能性がある。最も長いデータ経路が指定する最高周波数でR I C Aをクロックすると、達成可能な最大処理能力が制限されることになる。

【0045】

イネーブル信号を生成する前にR R Cが待つクロック・サイクルの量は、アレイの構成命令10の一部としてデコーダ11が構成することができる。これとレジスタおよびプログラム・カウンタへのクロック・ゲーティング技術とを組み合わせる(実行フロー制御セルを介して)ことにより、ソフトウェアの一部としてプログラム可能な可変クロック・サイクルを実際に達成することができる。

40

【0046】

別の特殊なセルは条件付き組合せ経路を与えるマルチプレクサ命令セルである。ハードワイヤード比較器およびマルチプレクサを含むセルを設けることにより、本発明は簡単なマルチプレクサであるとコンパイラが識別する条件付き移動を実現することができる。更に、R I C Aとして実現したとき、本発明は並列多重実行データ経路を実現することもできる。かかるスパニング・ツリーは条件付き操作で実行の並列処理レベルを高めるのに有

50

用であり、したがって操作を終えるのに必要な時間を減らすことができる。

【0047】

図2は、(a) 島型および(b) マルチプレクサに基づく命令セル間の相互接続の例を示す。図2において、プログラマブル・スイッチ21はIC25の出力ポートと入力ポートの間の指向的接続23を行う。相互接続の設計は種々の規則を考慮してよい。例えば、各ICは1出力および2入力だけを有することや、ICの出力を決してその入力にループバックしないこと（組合せループを避けるため）などである。スイッチの回路設計およびトポロジに種々の周知の方法を利用することができる。例えば、図2aに示すような通常のFPGAに見られる島型メッシュや、図2bに示すようなマルチプレクサに基づくクロスバーなどである。

10

【0048】

RIC A構造の特徴は、プログラム可能セルおよびそのプログラム可能相互接続を共にクロック・サイクル毎に動的に再構成できることである。これらのセルの数およびタイプはアプリケーションにおいてパラメータ化可能であり、UMC0.13 μ mCMOS技術ライブラリで実現されている。RIC Aの再構成可能構造を図2aに示す。RIC A構造は、明確な32ビット（ただし、パラメータ化可能なので32ビットに制限されない）プログラマブル機能ユニット、汎用レジスタ、メモリ・セル、および島型メッシュ再構成可能相互接続から成る。RIC A構造の基本的なコア要素は、CPU命令と同様に1つの操作を実行するようにプログラムすることができる32ビットのプログラム可能ICである。例えば、ADDセルは符号付き／符号なしの加算または減算機能としてプログラムすることができ、ICは、データ経路内の操作連鎖を可能にする島型メッシュ構造を通して相互接続される。

20

【0049】

図3に示すサンプルのCコード30を一般的な逐次プロセッサ上で実行するには19サイクル（32で示す）必要である。しかし同じコードをTIC6203などのVLW DSP用にコンパイルすると、15サイクル（34で示す）で実行する。なぜなら、VLW構造は最大8つの独立命令を同時に実行できる（6個のALUと2個のマルチプレクサが利用できる）からである。4つの同時乗算および4つのメモリ・アクセスが可能な場合は、サイクル数は8に減少する。コードが簡単なこととFPGAなどのハードワイヤード方式を用いて達成できるものとを考慮に入れると、これでも多い。従属命令が存在すると、コンパイラはクロック・サイクル数を更に減らすことはできない。

30

【0050】

図4は前に示したCコード30をわずか2サイクル（40および42で示す）で実行するのに必要な命令セルの構成を示す。これは例えば構造が、4xADD、4xRAM、4xMUL、および2xREGを同時に実行することができる14個の演算要素を備える場合である。これはVLWプロセッサが直面する限界を克服し、より高度の並列処理を可能にする。サイクル1（40で示す）に示すように、最も長い遅れ経路44は、RAMアクセスが2回、乗算が1回、および簡単な算術演算が数回と等価である。これは、擬似同時RAMアクセスを単一の可変クロック・サイクル内の妥当な時間内に達成できる場合だけの事例である。同じサイクル中により多くの命令をどれだけ実行できるかを比べると、これは一般的なDSP内のクリティカル・パスよりあまり長くない。したがって、計算のタイプによるが、かかる配列をサポートする構造はVLWと同様な処理能力を一層低い電力消費で達成することができる。

40

【0051】

本発明の或る態様は、RIC Aアレイをハードウェアで生成するための、この実施の形態では領域特有のアレイを生成するのに利用できるツールに基づく自動ツール・フローを与える。これらのツールは必要なアレイの特性を捕らえて、RIC Aコアの合成可能なレジスタ・トランスファ・ランゲージ(RTL)定義を生成する。これは、確認や合成やレイアウトのための標準のシステム・オン・チップ(SoC)ソフトウェア・フローで用いることができる。

50

【0052】

図5は本発明の或る実施の形態に係るプログラミング・フローを示す。標準GNU Cコンパイラ（gcc）を用いてC/C++コード50をコンパイルして、どのICを用いる必要があるかを記述するアセンブリ・フォーマット52を作る。ただし命令は順次に実行すると仮定する。RICAコンパイラはこのアセンブリを処理して、一連のネットリスト54を作る。各ネットリストは1クロック・サイクル内に実行する命令のブロックを含む。ネットリストへの分割は命令をスケジューリングして命令の間の依存性を分析した後に行う。ネットリスト内では、従属プログラム命令は直列に接続され、独立プログラム命令は並列に接続される。スケジューリング・アルゴリズムはアレイ内のIC資源や相互接続資源やタイミング制限を考慮に入れる。最も多くのICを用いまた同時に最長経路遅れを減らして最小にすることにより、プログラム処理能力を最高にすることを目指す。

10

【0053】

相互接続構造に従って、コンパイラの経路選択部分は、場所および経路ツールVPRに見られるような標準の確立された技術に基づいてよい。更にコンパイラは、gccが生成した一時的レジスタを簡単な配線で置き換えて除くなどの厳しい最適化も行う。コンパイラは一連の命令セル56の構成を生成し、これを構成命令6として符号化して構成メモリ10内に記憶する。

【0054】

種々のアルゴリズムについて設計空間を迅速に分析しかつハードウェアの柔軟性を最大にするには完全なツール連鎖が有用である。また、ツール連鎖が現在の種々の設計方式と統合するのに適していることは利点である。図6にRICAシステムの設計方法を記述する。

20

【0055】

RICA再構成可能構造に適用できる柔軟性のレベルは2つある。

- ・ 製作前—選択された数のハードウェア資源について高水準C/C++コードをコンパイルする。RTLシミュレーションによりまたはRICAシミュレータを通して決定された必要な性能が満たされない場合は、高水準コードを修正しまたはセル資源の混合を変えてよい。ハードウェア資源を調整すると用いる特定の領域に合うように構造を調整することができるので、電力および資源を節約することができる。この段階は柔軟性の最高のレベルを有する。

30

【0056】

- ・ 製作後—アレイ資源は固定される。したがって、製作プロセス中にまたはその後に引き続きアルゴリズムを変えるには、その固定された資源についてコードを単に再コンパイルする。更新されたコードについて前のコードと同様の操作が必要なが多く、大多数のアルゴリズムは変わらずに残る。これは、生成されたコードが所定の構造についてなおかなり最適であることを意味する。

【0057】

RICA構造のプログラミングは種々のツールを集めて行う。命令セル（IC）を使うと高水準プログラムをRICA構造にマップするのに必要な全体の作業が非常に簡単になる。標準CPUと同様にしてアレイをプログラムできるので、最適化されたコンパイラや言語パーサなどの利用可能な既存の技術をプロセッサに再使用することができる。

40

【0058】

図6において、段階1で、高水準コンパイラ604は高水準コード602を中間アセンブリ言語フォーマット606に変換する。このステップは標準GNU Cコンパイラ（gcc）が行う。これはC/C++コードをコンパイルして（他のフロント・エンドの中で）、どのICを用いる必要があるかを記述するアセンブリ・フォーマットに変換する。命令を逐次に（すなわち、毎サイクル1命令）実行すると仮定してgccの出力を生成する。コンパイラはRICAで利用できる並行処理については知らない。しかし、機械記述定義を用いる従来のプロセッサで一般的であるように、gccを調整して、利用可能なレジスタの最大数や利用可能な命令セル（これは許容される操作を定義する）などの目標のR

50

ICAのいくつかの制限を考慮に入れる。コンパイラは、レジスタ割当て、スタック処理、およびプロローグ／エピローグ定義などの事項を自動的に処理する。更に、ループ展開やループ・ピーリングやループ融合などの、後で有用になる全ての最適化を実行する。

【0059】

段階2（RICAコンパイラ608）で、RICA構造に関係する全ての最適化を行う。RICAコンパイラ・プロセスはgcc606のアセンブリ出力を受け、プログラムを表す一連のネットリストを作る。各ネットリストは単一のクロック・サイクル内に実行する命令のブロックを含む。ネットリストへの分割は命令をスケジューリングして命令の間の依存性を分析した後に行う。従属命令はネットリスト内で直列に接続され、独立命令は並列に走る。またスケジューリング・アルゴリズムはアレイ内の資源やタイミングの制約を考慮に入れる。最も多くのICを用いまた同時に最長経路遅れを減らして最小にすることにより、プログラム処理能力を最大にすることを目指す。最後に、gccが生成した一時的レジスタを簡単な配線で置き換えて除くなどの厳しい最適化も行う。

10

【0060】

このように、RICAコンパイラは複数の操作段階を行って構造のコードを最適化し、また構成命令ストリームを生成する。これは、他の点では逐次のアセンブリ・コード内で並行処理を識別して抽出するという問題を解決するのに必要である。

【0061】

アレイ内の分散されたレジスタにより、同じ回路の異なるパイプライン・オプションを持つデータ経路を構築することができる。パイプライン計画の変更は最大処理能力または電力消費に影響を与える可能性がある。通常、パイプラインのこのような柔軟性はCPU構造では利用できない（FPGAでは確かに存在する）ことであって、コンパイラは最良のパイプライン配列を選択して、コンパイルされたCコードの性能を改善する機会が与えられる。他の方法ではできないことであるが、これにより例えば電力消費の少ないパイプライン方式を選ぶことができる。

20

【0062】

段階3（場所および経路）は分割および資源のスケジューリングを行うループの一部なので、実際には段階2のRICAコンパイラ608内で始まる。なぜなら、相互接続資源はネットリストの経路選択性と最長経路のタイミングに影響を与える可能性があるからである。しかしここではこれを別個のステップとして述べる。なぜなら、場所および経路ツールVPR624などの標準ツールを再使用してネットリスト622をアレイ628にマップできるからである。

30

【0063】

段階4（構成メモリ）では、コンパイラはマップされたネットリストから構成メモリ26（この場合は構成RAM）の必要な内容を生成する。次に、従来のEDAツールでシミュレーションおよび性能分析を行い、またはRicaシミュレータを用いてコードの機能性をテストすることもできる。

【0064】

他のツールを設計フローに接続して、オープン・ソース・グラフ視覚化ソフトウェアであるGraphViz620に適したフォーマット618をRICAコンパイラが出力できるようにするなどにより設計プロセスを助けて、コードおよびスケジューリングされたステップの依存性の視覚表示を作ることができる。RICA内のセルの組合せや数は設計者のアプリケーション領域および利用可能なシリコン資源に従って変わることがあるので、ハードウェア生成ユーティリティ642を用いて機械記述に基づくカスタムRTLコード644を生成する。

40

【0065】

コンパイル・プロセスには複数の段階が関係する。RICAコンパイラのフロント・エンド610は、加算、論理、割り算、シフト、乗算などの命令を処理する命令パーサである。この段階では修正されたGCCアセンブラ・コードの字句解析を行い、機能、宛先レジスタ、ソース・レジスタ、およびセル構成データを抽出する。新しい命令をそれぞれ内

50

部パターンに追加し、無限の量の資源（レジスタ、加算器など）を有することに基づいて命令の完全なネットリストを作る。配線名の適切な生成とレジスタの割当てとを行ってレジスタの再使用を明らかにし、スケジューラが必要とする全ての情報を保持する。プロセッサ上のレジスタの数は固定されているので、コンパイラはレジスタを再使用するのにグラフ着色などの方法を用いる。またRICAコンパイラのフロント・エンド部分は、修正されたgccアセンブラ・コードからのラベルの処理に適した機構を有する。これは、分岐または飛越し命令が来たときに、生成されてスケジュールされたアセンブリ・コードはラベル位置を処理する必要があるからである。

【0066】

RICAコンパイラは、追加の資源および時間の制約を考慮した操作連鎖リスト・スケジューリング（CLS）アルゴリズムの考え方に基づいて再構成可能資源スケジューリング・アルゴリズム（RRS）を実現する。これは、利用可能な時刻を昇順に（ゼロから開始して）処理して、次に移る前の或る時刻にできるだけ多くの命令をスケジュールする。

【0067】

リスト・スケジューリング（LS）は基本ブロックのための最も普通に用いられるコンパイラ・スケジューリング法である。その主な考え方は、次のクロック・サイクルに移る前の或るクロック・サイクルでできるだけ多くの操作をスケジュールすることである。この手続きは全ての操作のスケジューリングが終わるまで続ける。この方法の主な目的はDFG内の操作の全実行時間を最小にすることである。もちろん、リスト・スケジューリングでは順位関係も重視する必要がある。或るサイクル t でDFG内の先行操作が実行を完了した操作を全ていわゆる実行可能リスト $R[t]$ に入れる。実行可能リストはスケジューリングに利用可能な操作を含む。

【0068】

$R[t]$ 内の全ての操作についてサイクル t で未利用の資源が十分ある場合は、これらの操作をスケジュールすることができる。しかし $R[t]$ 内の各操作で適当な資源を利用できない場合は、サイクル t でどの操作をスケジュールするか、およびどの操作を保留して後でスケジュールするようにするかについて選択を行わなければならない。通常、この選択は発見的に行い、各発見で特定のタイプのリスト・スケジューリングを定義する。高度な方法で優先順位を計算することにより、最も重要なセルを最初にスケジュールし、そのセルを適当な資源に割り当てる。通常、LSは優先順位ベクトルを用いてどのタスクを最初に考えるか決定する。

【0069】

スケジューリング・プロセス612では、RRSスケジューリング・アルゴリズムは実行可能リストを用いて操作を追跡することができる。スケジューリングに利用できる操作は実行可能リスト内にある。操作に使える空いた資源が実行可能リスト内にある場合は、レジスタ割当ておよび経路選択情報を考慮した上でこの操作をスケジュールしてよい。しかし操作に使える空いた資源が実行可能リスト内に十分ない場合は、スケジューリング・アルゴリズムはこのサイクルでどの操作をスケジュールするか選択しなければならない。その操作が資源およびクロック・サイクルの制約を満たしている限り、RRSスケジューリング・アルゴリズムは実行可能リスト内の移動度および並列処理により決まる最高の優先順位の操作を選択する。

【0070】

優先順位の高い操作は移動度が小さく、並列処理が高い。移動度はスケジュール間隔の長さで定義される。高い並列処理は最初にスケジュールすべき独立命令を決定する。RRSでは利用できる従属並列処理がRICA構造内にあるので、操作連鎖と呼ばれるコンパイラ技術を用いる。操作のスケジューリングおよび連鎖はレジスタの再使用に影響を与えるので、スケジューリング・アルゴリズムはレジスタ割付けも行う。資源および時間の制約は機械記述616内で与える。機械記述616では、プレスケジューリング推定経路選択時間を用い、またVPRからのポストスケジューリング経路選択時間遅れを用いる。時間の制約はプログラム可能計算ユニット遅れだけでなく、経路選択遅れも含む。

【0071】

スケジューリングの前と後のコードの簡単な例を以下に示す。g c cが生成する最初のアセンブリ・コードは次の通りである。

```

1   L o a d   r 1, x
2   L o a d   r 2, y
3   M u l t   r 3, r 2, r 1
4   L o a d   r 4, A
5   M u l t   r 2, r 4, r 1
6   A d d     r 5, r 2, r 4
7   M u l t   r 1, r 2, r 5
8   L o a d   r 3, B
9   A d d     r 7, r 1, r 3

```

10

【0072】

このコードを実現するのに9サイクルおよび6個のレジスタが必要である。目標のR I C Aが3つのロード、2つの乗算、および2つの加算を有する場合は、R I C Aを目標とするスケジューラが生成するコードは、コードを終了するのに2可変長クロック・サイクルと4個のレジスタを用いるだけである。このスケジュールされたアセンブリ・コードは次の通りである。

ステップ1：

```

1   L o a d   w i r e 1, X
2   L o a d   w i r e 2, Y
3   M u l t   r 1, w i r e 1, w i r e 2
4   L o a d   w i r e 3, A
5   M u l t   w i r e 4__r 2, w i r e 1, w i r e 3
6   A d d     r 3, w i r e 4, w i r e 3

```

20

ステップ2：

```

7   M u l t   w i r e 5, r 2, r 3
8   L o a d   w i r e 6, B
9   A d d     r 4, w i r e 5, w i r e 6

```

【0073】

R I C Aコンパイラの最終段階はマップ614である。これはスケジュールされたステップのネットリスト内の機能セルをハードウェア・セル・レイアウト622に割り当ててマップし、資源割当て仕様に基づいて所定のハードウェア構造のための構成命令の適当なビット・ストリーム626を作る。この構成ビット・ストリーム626は経路選択相互接続のための構成から成る。これはV P R 6 2 4およびコンパイラからのセル機能構成を通して生成される。この出力は最終的にハードウェアをプログラムするのに、またはシミュレータがシステム性能を分析するのに用いてよい。またコンパイラは構成データの人間読取り可能な形式630を出力する。

30

【0074】

シミュレータ632はアルゴリズムの迅速な開発を助ける、設計環境の重要な1つの態様であり、高い抽象レベルで資源を割り当てることにより、時間のかかるR T Lハードウェア・シミュレーションを行う前にトレードオフの範囲を決めることができる。シミュレータは人間読取り可能な. n e t f i l e 6 3 0と、各セルの機能性および性能分析ルーチンのためのコードを含むシミュレーション・テンプレート・ファイル634とを用いて、その与えられた. n e t f i l eについてR I C A構造に基づいて挙動レベルCモデル636を生成する。このCモデルはg c cコンパイラ638に与えられる。g c cコンパイラ638が出力する性能特性640および機械記述616を通るフィードバック・ループによって、ネットリストをハードウェアに割り当てた後の経路選択遅れを考慮に入れる。

40

【0075】

50

スケジュールされたステップ毎に、シミュレータはそのステップの依存性に従って操作の順序を再整理する。次に、これらの再整理された全てのステップとその相互の接続とを組み合わせ、全システムを表すデータ経路の集合を形成する。これをコンパイルすると、各ステップでのループ、レジスタ、およびメモリなどの機能性と全体の実行されたステップとを共に評価することができる。得られる他のパラメータとしては、各命令セルのモデル・ライブラリを有することに基づく電力および面積の推定値がある。

【0076】

図7aおよび図7bは本発明の利点を示す。RICAは粗粒度FPGAと同じ柔軟性と、DSPと同じプログラム可能性とを有する。RICAは粗粒度の構成可能構造を用いるので、一般的な精密粒度FPGAより電力消費が少ない。VLIW DSPは独立命令を並列に実行できるだけであるが、RICAは直列に接続できるので独立および従属の両方のアセンブリ命令を同じクロック・サイクル内に実行することができる。これを図7aおよび図7bに示す。ここではハードウェア資源は4個の加算器と2個の乗算器を含むものとする。

【0077】

図7aを参照すると、VLIWプロセッサは独立命令702だけを実行することができるので5クロック・サイクル必要である704。図7bを参照すると、RICAは資源の制約の下で独立命令706および従属命令708を実行できるので、費やすのは2クロック・サイクルだけである710。したがって、RICAの処理能力はVLIW DSPより高くなる。同じ「周波数」で走る場合は、または両者の内部の計算要素の遅れが同じ場合は、RICAの方が処理能力が高い。RICAはVLIW DSPより最大周波数が低いので、制御に集中したアプリケーションによっては最大処理能力が低い可能性がある。

【0078】

また、RICAは高度の操作連鎖技術を用いてレジスタおよびメモリの使用量を減らして、電力消費を大幅に下げる。操作連鎖は周知の技術であって、拡張されたクロック・サイクル内で2つの操作を順次に結合して、アプリケーションに必要なレジスタを減らす。従来の高水準分析(HLS)は変換プロセスであって、挙動記述をレジスタ・トランスファ・レベル(RTL)の構造的特徴記述に変換するものである。このプロセスでスケジューリングは重要なタスクである。スケジューリングは、制御データ・フロー・グラフ(CDFG)を時間ステップに分割することにより、スケジュールされたCDFGモデルを生成する。スケジューリング・ルーチンを実行した後、レジスタ割当てを用いて設計でのレジスタを最小にする。

【0079】

高度の再構成レート・コントローラ・セルに関する本発明の1つの態様の或る実施の形態について以下に説明する。

図8を参照すると、高度の再構成レート・コントローラ(RRC)802はプログラム・カウンタ804、メモリ806、808、およびレジスタ810に送る別個のイネーブル信号を適当な時刻に生成して複数のクロック信号を生成する。構造の他の部分に送るイネーブル信号を生成する前にRRCが待つクロック・サイクル812の量は、アレイの構成命令814の一部としてプログラム可能である。

【0080】

図9を参照すると、RRC命令ユニットのこの実施の形態では、ユニットの構成ビットは複数のタグ・フィールドを含む。すなわち、ジャンプ・セル・クリティカル・パス・タグ902、Rmemセル・パス・タグ904-906、および組合せセル・クリティカル・パス・タグ908（これはレジスタの活動化を決定する）である。

レジスタを制御する組合せセル・クリティカル・パス・タグは図1に関して上に説明したのと同様に動作する。すなわち、タグはレジスタ・セルをいつクロックするかを決定する。

【0081】

ジャンプ（実行フロー制御）セルがそのステップの最長のクリティカル・パスの一部で

10

20

30

40

50

ない場合は、現在のステップ内の最長のクリティカル・パスが終わる前に次のステップの先取りおよび復号を開始することができる。これを行うには、ジャンプ・セル出力値が有効になるのはいつかを決定するための追加の情報が必要である。したがって、この例では4ビット902をRRCユニットの構成に割り当てる。これはRICA内部クロック・サイクルをいくつ待つかに対応する。すなわち、ジャンプ・セルのクリティカル・パスが20nsでクロック分解能が2nsの場合は、ジャンプ構成フィールドを10にセットする。

【0082】

RICAコア内にある読取りインターフェース・メモリ・セル毎に、対応する読取りメモリ・タグ（例えば、904, 906）をRRC構成命令に割り当てる。これらのタグを用いてメモリ・セルのタイミング管理を行うので、データはステップの計算中の適当な時間にだけ読み取られる。

10

図10を参照すると、Rmemタグの最初の部分は、ステップの開始から、インターフェース・メモリ読取り命令セル102, 112の入力（例えば、アドレス104, 114およびオフセット106, 116）をメモリ108, 118に送るべきときまでの遅れ時間を定義するのに用いる。

【0083】

またこれらの各タグはカスケード状態を示す1ビットを有する。カスケード・ビットをハイにセットすると、メモリ読取りはその前のインターフェース・メモリ・セル読取りアクセスに依存するので、その前のインターフェース・メモリ・セル読取り遅れ時間が終わってからはじめてそのタイミング遅れを開始しなければならないことを意味する。

20

図10に示す例では、両方のRmemセル・カスケード・ビットはローにセットされているので、両者はその遅れ時間を同時に開始する。しかし図11に示す例では、Rmem[1]112のカスケード・ビットはハイにセットされているので、Rmem[0]102がその遅れを終了した後にはじめてRmem[1]112の遅れを開始する。更に、Rmem[0]102の出力はRmem[1]112の入力への配線に直接マップされる。

【0084】

再構成可能型計算枠組構造での同時マルチスレッディング（SMT）に関する本発明の1つの態様の或る実施の形態について以下に説明する。

本発明のこの実施の形態はRICA再構成可能型計算枠組構造内のSMTを提供する。

30

同時マルチスレッディングが単一の再構成可能格子に与える効果を示す1つの例として、ここでは二次元マルチホップ直接ネットワークを介して接続される計算ユニットの集合を用いる。相互接続方式の他の例としてクロスババーまたは区分された共用バスがある。

【0085】

同じプログラム内に共存する他のスレッドとは独立に、各スレッドをスケジューリングしてアレイにマップする。時間的マルチスレッディングでは、任意の所定の時刻に活動的なスレッドは1つだけである。或るSMT環境では、これらのスレッドはアレイの異なる部分を占め、並列に動作することができる。

【0086】

図12は競合せずに同じ再構成可能格子126を共用する2つのスレッド122, 124を示す。これはアレイを最大限に使用する理想的な場合である。このようなスレッドを強制的に分離するには、コンパイル時にスレッドの1つをコアの一隅に片寄せ、他の1つをコアの反対側に割り当ててよい。多くの場合、2つの同時のスレッドは1つ以上の共通資源を同時に要求する。相互接続資源「競合」の1つの形を図13に示す。この例では、共通資源132は2つのスレッド134および136の一方に割り当てられ、他方は待つ必要がある。

40

【0087】

一般的なスレッドは所定の構成内におそらく1つ以上の独立なデータ経路を有している。厳密に同期させなければ、各独立の競合しないデータ経路は他と並列に動作することができる。その結果、スレッドは部分的に（すなわち、そのデータ経路の一部だけが）実行

50

される。部分的実行を用いることによりコアの利用度を更に高め、その結果、性能を高めることができる。

【0088】

図14は、種々のシナリオの下に走る2つのスレッド（スレッド₀142およびスレッド₁144）の実行時間線図を示す。シナリオとしては時間的MT（図14a）、SMT（図14b）、およびSMT（図14c）があり、部分的実行をサポートする。図14には構成ロード146および実行148の状態をスレッド毎に示し、また保留状態150および部分的実行状態152も示す。

【0089】

リスト・スケジューリングに基づく新しい操作連鎖再構成可能スケジューリング・アルゴリズム（CRS）を提供する本発明の或る実施の形態について以下に記述する。これは分散された高性能命令セルに基づく再構成可能システムで利用可能な、命令レベルの並列処理を最大にするものである。他の一般的なスケジューリング法とは異なり、これは配置および経路選択、レジスタ割当て、および高度の操作連鎖コンパイル法を用いて一層高性能のスケジュールされたコードを生成する。この方法を用いたスケジュールは、VLWと少なくとも同等な処理能力を、非常に低い電力消費で達成する。

【0090】

本発明のこの実施の形態は、再構成可能システムを目標にした効率的な操作連鎖スケジューリング・アルゴリズムを提供する。この連鎖再構成可能スケジューリング（CRS）アルゴリズムはリスト・スケジューリング・アルゴリズムに基づいて高度の操作連鎖法を用い、また資源および時間の制約の下でレジスタ割当て、電力消費、配置および経路選択遅れの影響を考慮する。CRSアルゴリズムを用いると各スケジュール・ステップでの従属および独立の命令セル（IC）の数が最大になりまた同時に最長経路遅れのクロック・サイクルの総数が最小になるので、プログラム処理能力を高めて電力消費を下げるができる。

【0091】

従来のHLSプロセスとは異なり、本発明のこの実施の形態はスケジューリング、経路選択、命令セルの結合、およびレジスタ割当てを組み合わせ、命令セルに基づく構成に適合させる。本発明のこの実施の形態は新しい効率的な多目的スケジューリング最適化を行って、新しい再構成可能構造の処理能力を高くすると共に電力を低くする。

【0092】

従来のリスト・スケジューリングは命令セルに基づく構造で直接用いることができない。なぜなら、

- 1) 従属命令の並列処理を行うには効率的な操作連鎖が必要であり、
 - 2) スケジューリング・アルゴリズムにおいてレジスタ割当てを考慮する必要がある、
 - 3) スケジューリング・アルゴリズムは再構成可能機能ユニットおよび経路選択相互接続遅れの影響も考慮に入れなければならないが、これは再構成可能デバイス内のデータ経路遅れを変える可能性がある、
- からである。

【0093】

図15はスケジューリング・アルゴリズムの或る実施の形態の流れ図を示す。

スケジューラは実行可能リストを生成し1502、スケジューリングのためにこのリストから操作を選択する1504。優先順位割当て機構を処理するのに3つのスケジューリング法を用いる。すなわち、IPRS1506、CPRS1508、およびMPRS1602である。MPRSの場合は、アルゴリズムのこのステップ1510でIPRSを用い、レジスタ割当ての後のステップ1530でCPRSと組み合わせる。

【0094】

1506から1520までのアルゴリズムの部分は操作をスケジュールしまたは除去するのに用いる。1512でスケジューリングを行う場合は、操作連鎖を行い1514、実行可能リストを更新する1516。このステップ1518で実行可能リスト内の他の操作

10

20

30

40

50

が利用可能でかつ機能ユニットが利用可能な場合は、実行可能リストから次の操作を選択する1504。

【0095】

ステップ1512で、1506から1520までのアルゴリズムの部分をレジスタの除去に用いると決定した場合は、ステップ1514から1518ではなく、スケジュールされた操作の除去を行う1520。

1522で、この構成ステップ（またはクロック・サイクル）でレジスタが十分ない場合は、ステップ1506から1510にループバックして操作の除去を行い、例えば、フラグをセットしてレジスタ除去を行うことを決定テスト1512で示す。

【0096】

ステップ1522でレジスタが十分ある場合はレジスタ割当てを行い1524、次にIPRS1526およびCPRS1528の場合はノーアクション、MPRSの場合はステップ1530でCPRSを用いて更に操作のスケジューリングを行う。

スケジューリングの3つの方法を用いることにより、最適な方法を用いてコードの全部または一部をスケジュールして、目標のRICA構造の構成ビットを生成することができる。最適スケジューラ出力を選択した1532後で、スケジューラは次の構成ステップに進む1534。

【0097】

CRSアルゴリズムと簡単なリスト・スケジューリングとの比較を示すために、CRSアルゴリズムの別の例を以下に示す。これは簡単なリスト・スケジューリング・アルゴリズムに、@印をつけた新しい行を追加したものである。

方法： CRSスケジューリング

入力： アセンブリ・コードであって、

空間機械記述（資源、クロック・サイクル）

VPRアルゴリズムにより生成される経路選択遅れ

をスケジュールする操作を表すもの

出力： 高速で並列のネットリスト

Begin

ステップ1： 制御フロー・グラフ（CFG）を構築する

ステップ2： データ・フロー・グラフ（DFG）を構築する

ステップ3： 名前を変えて逆依存／出力依存をなくす

ステップ4： セルの柔軟性に基づいて命令に優先順位を割り付ける

ステップ5： 命令の選択とスケジューリングを繰り返す

【0098】

Cycle = 1 ;

Ready [Cycle] = DFGのルート ;

Ready [Cycle + 1] = ϕ ;

Scheduled [Cycle] = ϕ ;

@Available_Register [Cycle] = {出力レジスタである
が基本ブロックの入力レジスタとしては用いない}

【0099】

While (Ready Lists $\neq \phi$) // All Ready [i]
lists

{

 If (Ready [Cycle] = ϕ) then

 {

@ 利用可能なハードウェア資源を初期化する ;

 Cycle = Cycle + 1 ;

 }

 While (Ready [Cycle] $\neq \phi$) {

10

20

30

40

50

```

@ Readyから操作を優先順に除き、直列低パワー最適化法を用いて操
  作を選択する
  If (ヨサイクル上のtype (x) のフリー・イシュー・ユニット
@   &&操作を同じサイクル内で連鎖してよい) {
    S (op) = サイクル;
    F (op) = FinishTime (op);
    Scheduled [Cycle] = Scheduled [Cycle]
      ∪ {op};
    For (DFG内の次の操作s) {
      If (sが実行可能) then
        Ready [Cycle] = Ready [Cycle] ∪ s;
    }
  }
else
  Ready [Cycle+1] = Ready [Cycle+1] ∪ {op}
}
【0100】
@ If (操作はレジスタおよびScheduled [Cycle] 内の次の操作
  を用いる)
@ then { // release register
  操作の出力を保存するレジスタを除去して利用可能なレジスタ・リスト内に
  入れる}
@ N = このクロック・サイクルで必要なレジスタの数
@ A = このクロック・サイクルで利用可能なレジスタの数
@ if (N < A)
@   レジスタを割り付けて操作の出力を保存する
@ else {
@

```

N ≤ A 30

になるまでいくつかのスケジュールされた操作を優先順に除去す
る；

@ レジスタを割り付けて操作の出力を保存する；}

【0101】

Θ C P R Sを用いて別のスケジューリングを行う（このステップはM P R Sにだ
け用いる）

```

@ 最も長いクリティカル・パスおよび可変クロック・サイクルを計算する
}

```

@ 資源の結合ーハミング距離を用いる。

End Method

【0102】

スケジューラの設計入力のアセンブリ・コードまたは他の中間表現なので、依存グラフ
(DG)はレジスタを除去して命令依存を表すことにより生成される。従来のプロセッサ
のコンパイラは高度レジスタ割当てアルゴリズムを用いて固定の数の物理的レジスタを再
使用するので、DGは出力依存および逆依存と呼ばれる擬似データ依存を有することがあ
る。

【0103】

操作Aが作った値を操作Bが読み取る場合は、BはAに真に依存する。これはリード・

50

アフタ・ライト (RAW) と呼ばれる。操作 A が読み取ったレジスタに操作 B が値を書き込んで、A が読み取った値を壊した場合は、B は A に逆依存する。これはライト・アフタ・リード (WAR) と呼ばれる。操作 A と操作 B の両方が同じレジスタに値を書き込む場合は、B は A に出力依存する。これはライト・アフタ・ライト (WAW) と呼ばれる。

【0104】

レジスタ名の変更は、WAR/WAW競合を起こすレジスタの名前を変えることにより擬似依存性を除く方法である。しかしこの方法はレジスタ・スピリング (register spilling) の原因になることがある。この問題を解決するには、レジスタが不足しているので追い出された値を記憶するためにロード命令および格納命令を用いる必要がある。このためにメモリ・アクセスが増えて、性能が下がりまた電力消費が増える。RICA内の機能ユニットは同じクロック・サイクル内で逐次連鎖するので、操作連鎖法を用いるとレジスタおよびメモリの使用が減る。

10

【0105】

CRSスケジューリング・アルゴリズムにおける設計項目がコンパイルされまたスケジュールされたアセンブリ・コードの場合は、スケジューラは不必要なレジスタを除去するなどの追加のタスクを処理しなければならない。レジスタ割当ては高水準合成でスケジュールした後に行う。操作のスケジューリングおよび連鎖はレジスタ割当てに影響を与えるので、CRSスケジューリング・アルゴリズムはレジスタ割付けも考慮する。

20

【0106】

制御フロー・グラフは、2つの基本ブロックの間に正しいデータ依存を生成した異なる基本ブロックの命令の並列処理を見つけるのに用いられる基本的なデータ構造である。まず、コードを基本ブロックの集合に分割する。これは或るラベルで、またはジャンプ命令の後に始まる。基本ブロックはCFGのノードから成る。その後で、CFGのエッジを現在のブロックから分岐の全ての可能な目標基本ブロックに加える。CFGノードの入力／出力ポートは基本ブロックの入力／出力レジスタに関係する。入力レジスタは、他の基本ブロックが書き込んだ後でこの基本ブロックが読み取る値を保存する。出力レジスタはこの基本ブロックが計算した後で他の基本ブロックが読み取る値を保存する。CLSスケジューリング内の利用可能なレジスタは出力レジスタだけを含む。なぜなら、入力レジスタは他の基本ブロックが用いていて同じ値を保っているはずだからである。

30

【0107】

CRSスケジューリング・アルゴリズムには3つの方法が用いられ、クロック・サイクルの数が最小 (処理能力が最高) または電力消費が最低の出力コードがスケジューラの出力として選択される。

第1の方法はクリティカル・パス再構成可能スケジューリング (CPRS) と呼ばれるものである。この文脈では、CFGの或るノードから出力ノードまでの最も長い経路をそのクリティカル・パスと呼ぶ。全てのクリティカル・パスの中で最も長いものは、スケジュールの残りの部分を実行するのに必要な全時間の下限値を与える。優先順位の高い操作の方が移動度が小さい。移動度はスケジュール間隔の長さとして定義される。(移動度=最も遅い (ALAP) - 最も速い (ASAP) である)。したがって、CPRSスケジューリングでは、最も長いクリティカル・パスを持つノードをサイクルtでスケジュールするものとして選択する。

40

【0108】

第2の方法は独立命令優先度再構成可能スケジューリング (IPRS) と呼ばれるものである。IPRSでは、サイクルtでスケジュールするものとして選択するのは、他の利用可能なノードに比べてこのクロック・サイクルのクリティカル・パスの増加が最も小さいノードである。

【0109】

最後に、混合優先度再構成可能スケジューリング (MPRS) と呼ばれる第3の方法もCRSスケジューラで用いられる。これはCPRSアルゴリズムとIPRSアルゴリズム

50

とを結合したものである。MPRSでは、まずIPRS法を用いてノードをスケジュールし、レジスタ割当ての後でCPRSアルゴリズムを再び実行する（図1の□印のついた線はMPRSだけに用いられる）。固定のクロック・システムではCPRSアルゴリズムの実行時間が最も短くなる。CRSアルゴリズムは可変クロック・サイクルを用いるので、CPRSスケジューリング法はIPRSスケジューリング法に比べて実行時間が長くなることがある。CRSアルゴリズムはレジスタ割当てと組み合わせるので、レジスタの数が有限であることがIPRS法を制限することがある。MPRSスケジューリングはCPRSおよびIPRSスケジューリングより優れたスケジューリングを生成するかも知れない。

【0110】

図16に示す例を用いてこれらのスケジューリング法を示す。この構造は3xADDと2xREGとを同時に実行することができる5つの演算要素を備え、加算は1クロック・サイクルで実行すると仮定して、各加算器162のASAP時間およびALAP時間を角括弧で、例えば図16の164および166で示す。下の表1は3つの異なるCRSアルゴリズムを比較して、図16の機能を走らせるのに必要なクロック・サイクル数に与える影響を見る。

【表1】

表1：スケジューラの比較（3個の加算器と2個のレジスタ）

実行ステップ	CPRS		IPRS		MPRS	
	サイクル	SC.	サイクル	SC.	サイクル	SC.
1	2	[1,2,5]	1	[1,2]	2	[1,2,5]
2	2	[3,6]	1	[5,3]	2	[3,6]
3	2	[4,8,7]	1	[6]	2	[4,8,7]
4	1	[9]	2	[4,8,7]	1	[9]
5			1	[9]		
ET	7		6		7	

【0111】

表1から表3において、ETは実行時間の全クロック・サイクルを指し、SCはスケジュールされたセルを指す。表1はCPRSスケジューリングのETがIPRSスケジューリングのETより長く、MPRSスケジューリングのETと同じであることを示す。しかし異なるアルゴリズムのETは機械記述に大きく依存する。以下の表2および3は異なる機械記述に基づく異なるスケジューリング結果を与える。

【表 2】

表 2 : スケジューラの比較 (4 個の加算器と 2 個のレジスタ)

実行ステップ	CPRS		IPRS		MPRS	
	サイクル	SC.	サイクル	SC.	サイクル	SC.
1	2	[1, 2, 5, 3]	1	[1, 2]	1	[1, 2, 5, 3]
2	2	[6, 4, 8, 7]	2	[5, 3, 6]	1	[6, 4, 8, 7]
3	1	[9]	3	[4, 8, 7, 9]	2	[9]
ET	5		6		5	

10

【表 3】

表 3 : スケジューラの比較 (5 個の加算器と 5 個のレジスタ)

実行ステップ	CPRS		IPRS		MPRS	
	サイクル	SC.	サイクル	SC.	サイクル	SC.
1	3	[1, 2, 5, 3, 6]	2	[1, 2, 3, 4, 5]	2	[1, 2, 3, 4, 5]
2	3	[4, 8, 7, 9]	3	[6, 7, 8, 9]	3	[6, 7, 8, 9]
ET	6		5		5	

20

【0112】

表 1 および表 2 に示すように、CPRS アルゴリズムの ET は IPRS アルゴリズムより長く (表 1) または短く (表 2) なる。これはレジスタの数が原因である。しかし、レジスタが十分ある場合 (表 3) は、IPRS アルゴリズムの ET 時間は短い。この例では機能的資源は加算器だけを含むので、MPRS アルゴリズムを用いた結果は CPRS と同じ (表 1、表 2) か、または IPRS と同じ (表 3) である。しかし再構成可能な命令セルに基づく機械の場合のように資源が大きく変わる場合に適用すると、これは異なる結果を生成する。コンパイラは全ての方法を用いて入力コードをスケジューリングし、その要求に関して適当な出力コードを選択する。

30

【0113】

基本ブロック毎に、多数の操作の間のデータ依存を表すデータ・フロー・グラフ (DFG) を、CRS スケジューリング・アルゴリズムへの入力として用いる。設計をプロセッサ上で走らせるときの 1 つの制限は、単一サイクル内に実行することができる操作の数である。単一サイクル内に実行する操作の数を最大にするため、操作連鎖および可変クロック・サイクルを CRS スケジューリング・アルゴリズムに追加する。これにより使用するレジスタの総数は減り、メモリ・アクセスが減るので電力消費が減り、可変クロック・サイクルなので処理能力が増す。操作を連鎖するとき、スケジューリング・アルゴリズムはクリティカル・パスへの操作の影響だけでなく、プログラム可能な経路選択遅れも考慮しなければならない。

40

【0114】

ツール・フロー内のスケジューリング・アルゴリズムはプレスケジューリングおよびポ

50

ストスケジューリングを含む。プレスケジューリング・アルゴリズムはクリティカル・パスへの操作および推定経路選択時間の影響を考慮する。プレスケジューリング後のネットリストをVPRなどの標準の配置および経路選択ツールに与える。VPRツールまたは他の経路選択ツールから与えられる対応する経路選択情報を用いて、より正確なタイミング情報をポストスケジューリングに与えて、設計要求を満たすネットリストを生成する。CRSアルゴリズムは性能を最適化するための第1の段階に用いる。その後、この電力コスト・モデルとしてハミング距離を用いて資源結合を生成して、命令セル構成バス内の遷移活動を推定する。ハミング距離は2つの2進ストリングの間のビット数の差である。最小ハミング距離を持つ資源結合方式を選択して電力消費を減らす。

【0115】

10

異なるCLSスケジューリング・アルゴリズム(CPRS、IPRS、およびMPRS)を用いてまた本発明の再構成可能命令セルに基づく構造を目標にしてベンチマーク・テストを行い、各CLSアルゴリズムの性能を説明する。下の表4は実行時間およびエネルギー消費を示す。ベンチマークは構造上を同じ周波数(125MHz)で行い、また同じハードウェア資源を用いた。この再構成可能構造の電力消費値は、UMC0.13 μ m技術を用いたSynopsys PrimePowerによるポストレイアウト・シミュレーションの後に得たものである。

【表 4】

表4：異なるスケジューリング法の比較

ベンチマーク	方法	実行時間 (μ s)	エネルギー消費 (μ J)
2D-DCT	CPRS	2.352	0.1309
	IPRS	2.192	0.108
	MPRS	2.192	0.112
FIR	CPRS	1.998	0.263
	IPRS	1.98	0.248
	MPRS	1.98	0.249
IIR	CPRS	0.194	0.015
	IPRS	0.186	0.0154
	MPRS	0.188	0.0156
最小誤差	CPRS	9.286	1.59
	IPRS	9.086	1.519
	MPRS	9.286	1.529
OFDM	CPRS	42.496	0.7980
	IPRS	43.224	0.8003
	MPRS	41.6	0.7905
ピタゴ	CPRS	501.488	0.2296
	IPRS	452.162	0.225
	MPRS	452.1	0.227
ドライストーン	CPRS	283.046	1.004
	IPRS	281.08	0.878
	MPRS	282.06	0.925

【0116】

表4から分かるように、全てのベンチマークについてCPRS、IPRS、およびMPRSアルゴリズムから得られる実行時間および電力消費値は少し異なる。ここで、スケジューリングされたコードはエンド・ユーザの設計基準（例えば、電力、処理能力）により選択してよい。処理能力を高くするには、実行時間が最も短い出力コードを選択する。最適解はアプリケーションに大きく依存し、CPRS、IPRS、またはMPRSアルゴリズムにより生成される。これを表4に示した。多くのベンチマークで、CPRSスケジューリング・アルゴリズムは他のスケジューリング・アルゴリズムに比べて使用するレジスタは少ないがエネルギー消費が多い。

【0117】

スケジューリングおよびシミュレーション分析から、C P R Sスケジューリング・アルゴリズムが生成するコードは他のアルゴリズムに比べて組合せ論理経路が長いために電力消費が多い。しかし、C P R Sアルゴリズムの方がエネルギー消費が少ない場合もある。重ねて述べるが、エネルギー消費はアプリケーションに依存する。コンパイラは全ての方法を用いて入力コードをスケジュールし、その要求に適した出力コードを選択する。C L Sスケジューリング・アルゴリズムは他のD S P/V L I Wプロセッサに比べてエネルギー消費が少なく、処理能力は同程度と思われる。

【0118】

本発明の種々の態様は低電力アプリケーションのための、命令セルに基づく再構成可能型計算枠組構造を提供する。R I C Aの開発のためにトップダウン方式を用いたが、これは柔軟で、プログラムが容易で、低電力の構造のための重要な設計決定であることが分かった。これらの機能から、R I C Aは最新の低電力デバイスの主な設計要求を本質的に解決する構造である。

10

【0119】

再構成可能アレイと同様で、高水準プログラム・ソフトウェアと同等のシリコン構造を設計することにより、本発明は粗粒度のF P G A構造と同じ高性能と、D S Pと同じ柔軟性、低コスト、およびプログラム可能性を達成する。R I C A構造はC P Uと同様であるが、データ経路としてI Cに基づく再構成可能コアを用いることによりD S PおよびV L I Wより優れた利点（並列処理の一層のサポートなど）を持つ。再構成可能コアは独立および従属の両方のアセンブリ命令を含むブロックを同じクロック・サイクル内に実行することができる。これは、従属命令がプログラム内のI L Pの量を制限するのを防ぐ。D S P構造より優れている他の点は、集中レジスタ・ファイルをなくしかつ分散メモリ要素を用いて並列にレジスタにアクセスできるようにするためにメモリ・アクセスが減ることである。

20

【0120】

結果から分かることは、R I C AはF P G Aに基づく実施の形態よりはるかに性能が優れている一方で、直接の高水準再プログラミングが可能である。初期性能の結果を見ると、主なV L I Wおよび低電力D S Pプロセッサに比べて電力消費は最大で6倍も低くなる。

ここに述べた本発明の範囲から逸れない範囲で、他の変更および改善を加えてよい。

30

【図面の簡単な説明】

【0121】

本発明の或る実施の形態について添付の図面を参照して説明する。

【図1】本発明の或る実施の形態に係るプロセッサ構造の略図を示す。

【図2】aは、命令セルの間の島型に基づく相互接続の例の略図を示す。bは、命令セルの間のマルチプレクサに基づく相互接続の例の略図を示す。

【図3】Cコード、V L I W D S Pコード、および逐次アセンブラ・コード（A S M）のブロックを示す。

【図4】本発明の或る実施の形態に係る図3のアセンブラ命令の実行の略図を示す。

【図5】本発明の或る実施の形態に係るプログラミング・フローの略図を示す。

40

【図6】本発明の或る実施の形態に係る設計法の略図を示す。

【図7】aは、V L I W D S Pの命令スケジューリングの略図を示す。bは、本発明の或る実施の形態の命令スケジューリングの略図を示す。

【図8】高度の再構成レート・コントローラ・ユニットの略図を示す。

【図9】高度の再構成レート・コントローラの構成ビットの構造の一例の略図を示す。

【図10】独立メモリ命令セル操作の略図を示す。

【図11】従属メモリ命令セル操作の略図を示す。

【図12】非競合スレッドの同時マルチスレッディングの略図を示す。

【図13】共通の相互接続資源要求を有する競合スレッドの同時マルチスレッディングの略図を示す。

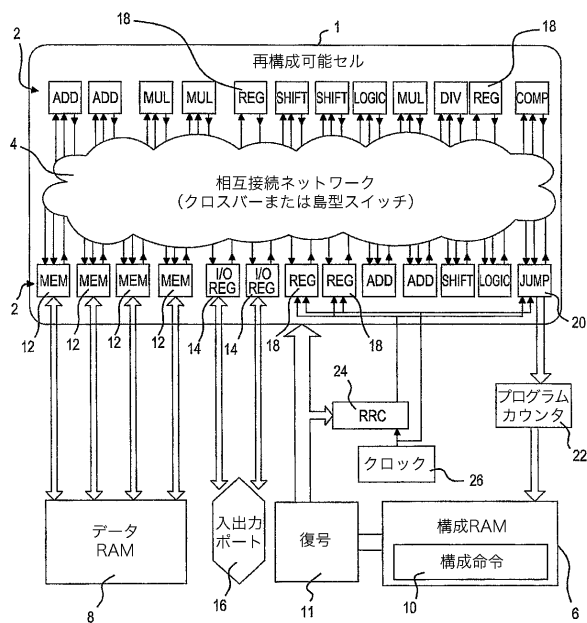
50

【図14】 aは、時間的マルチスレッディング・シナリオの下に走る2スレッドの実行タイミング図の略図を示す。 bは、同時マルチスレッディング・シナリオの下に走る2スレッドの実行タイミング図の略図を示す。 cは、部分実行をサポートする同時マルチスレッディング・シナリオの下に走る2スレッドの実行タイミング図の略図を示す。

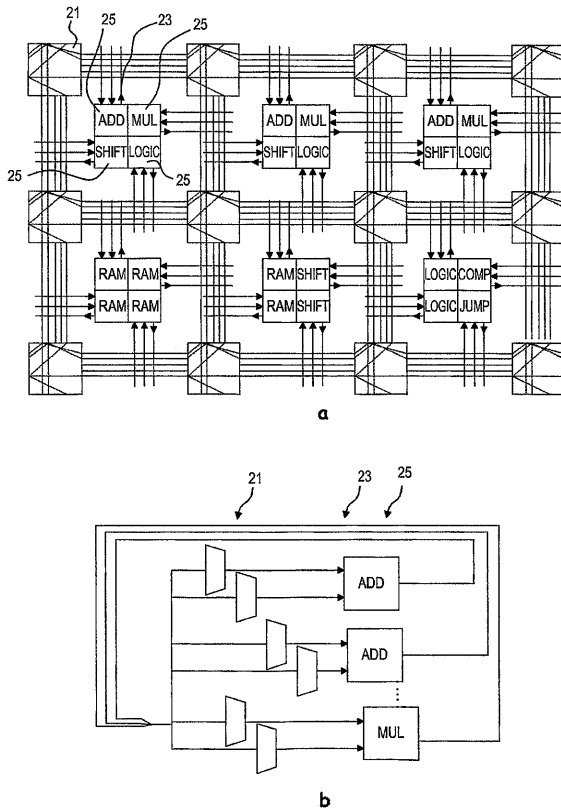
【図15】 操作連鎖再構成可能スケジューリング・プロセスの流れ図の略図を示す。

【図16】 入力データ制御フロー・グラフの略図を示す。

【図1】



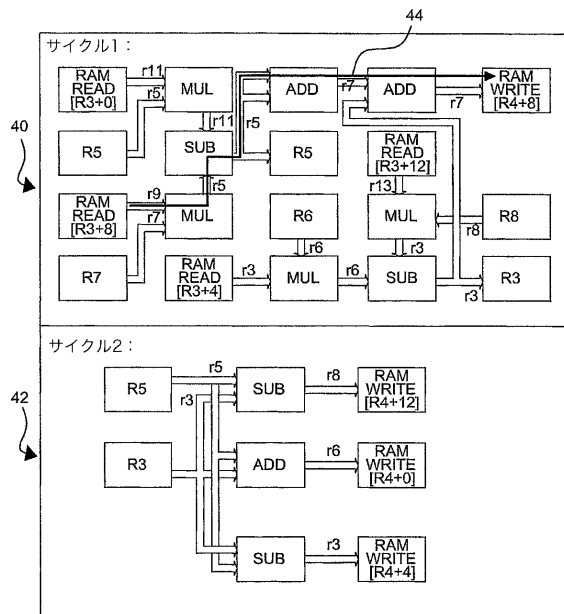
【図2】



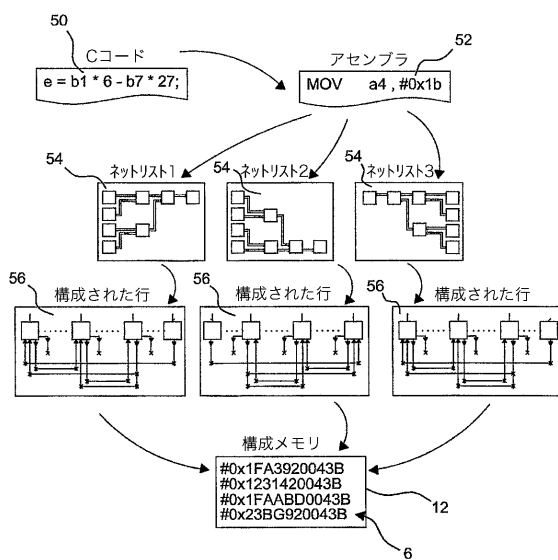
【図 3】

Cコード	逐次アセンブラ
b0 = in_mem [add+0];	LD [r3+0] → r11
b1 = in_mem [add+1];	LD [r3+8] → r9
b2 = in_mem [add+2];	MUL r11, r5 → r11
b3 = in_mem [add+3];	LD [r3+12] → r13
e = b0 * f0 - b2 * f2;	LD [r3+4] → r3
f = b1 * f1 - b3 * f3;	MUL r3, r6 → r6
out_mem [add+0] = e + f;	MUL r9, r7 → r5
out_mem [add+1] = e - f;	MUL r13, r8 → r3
out_mem [add+2] = f + 2*e;	SUB r11, r5 → r5
out_mem [add+3] = f - e;	ADD r5, r5 → r7
TMS320C6x VLIW ASM	
LDH *+A4 (2) → A7	SUB r6, r3 → r3
LDH *+A4 (6) → A3	SUB r5, r3 → r8
LDH *+A4 (4) → A0	ADD r7, r3 → r7
LDH *A4 → A5	ADD r5, r3 → r6
MPY A7, B6 → B5	LD r8 → [r4+12]
MPY A3, B8 → B6	SUB r3, r5 → r3
MPY A5, A6 → A3	LD r6 → [r4+0]
SUB B5, B6 → B5	LD r3 → [r4+4]
SUB A3, A0 → A0	LD r7 → [r4+8]
RET B3	
MV B5 → A3	
ADDAH A3, A0 → A4	
ADD B5, A0 → B5	
STH B5 → *+B4	
STH A0 → *+B4 (2)	

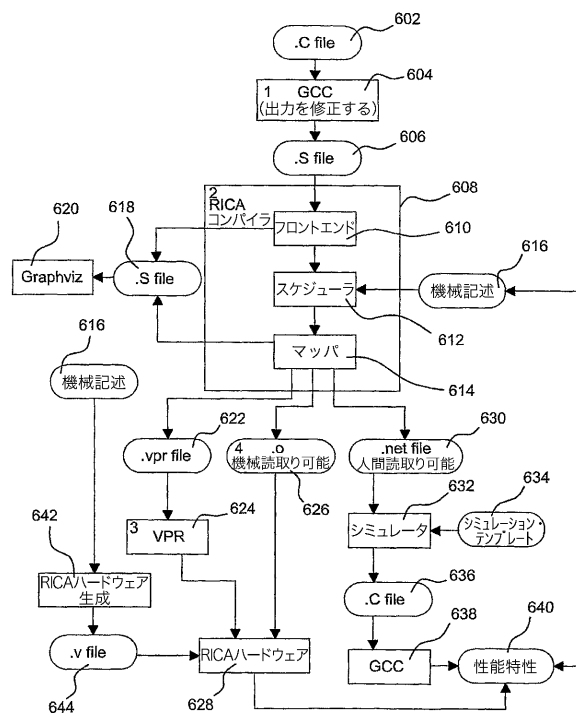
【図 4】



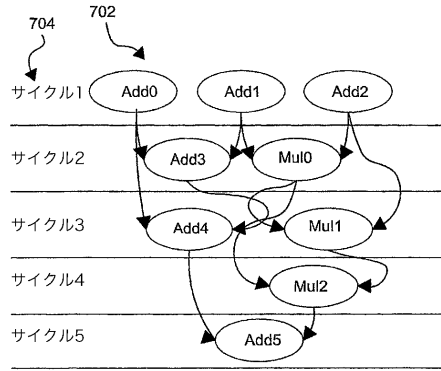
【図 5】



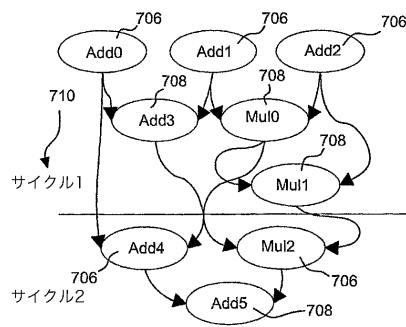
【図 6】



【図 7】

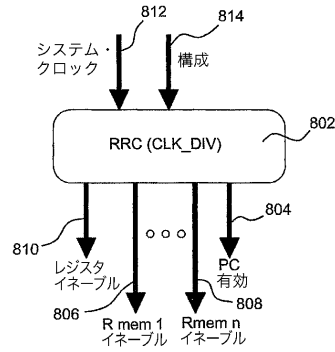


a

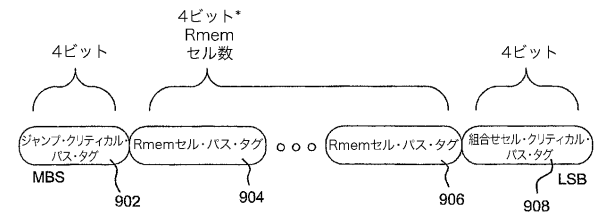


b

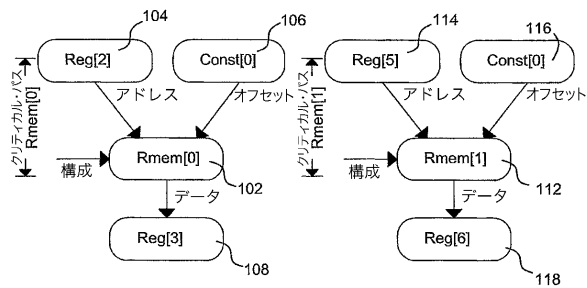
【図 8】



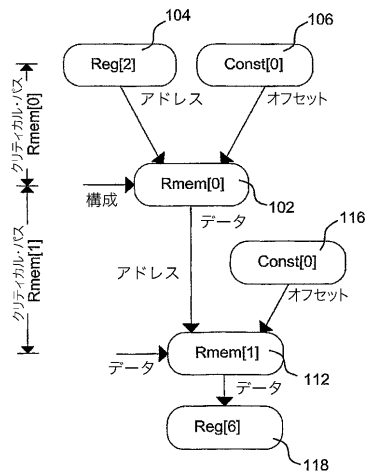
【図 9】



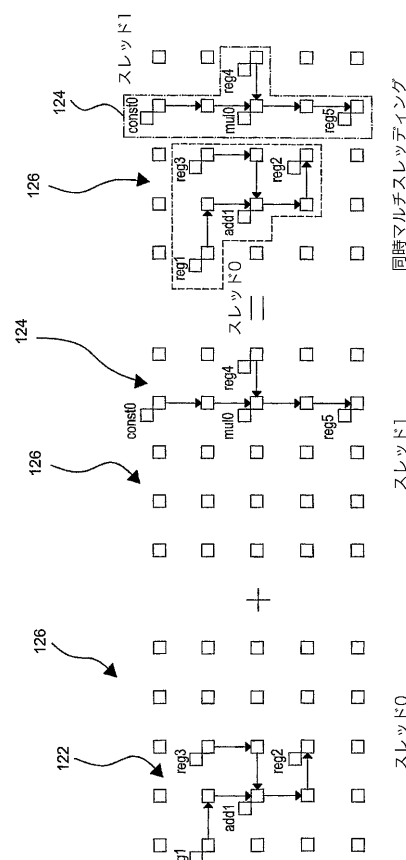
【図 10】



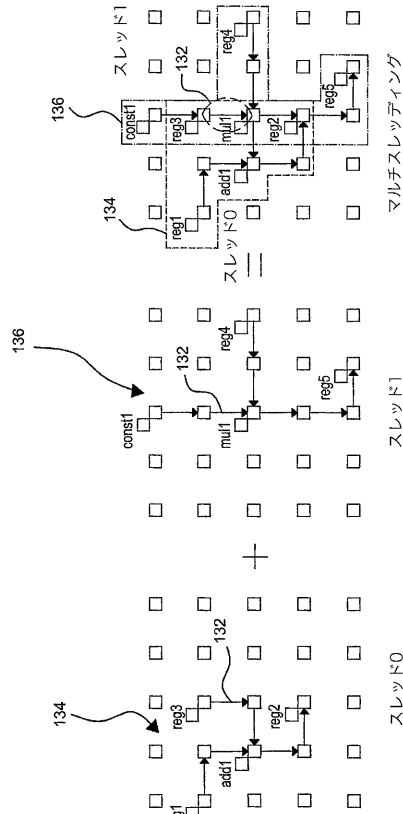
【図 11】



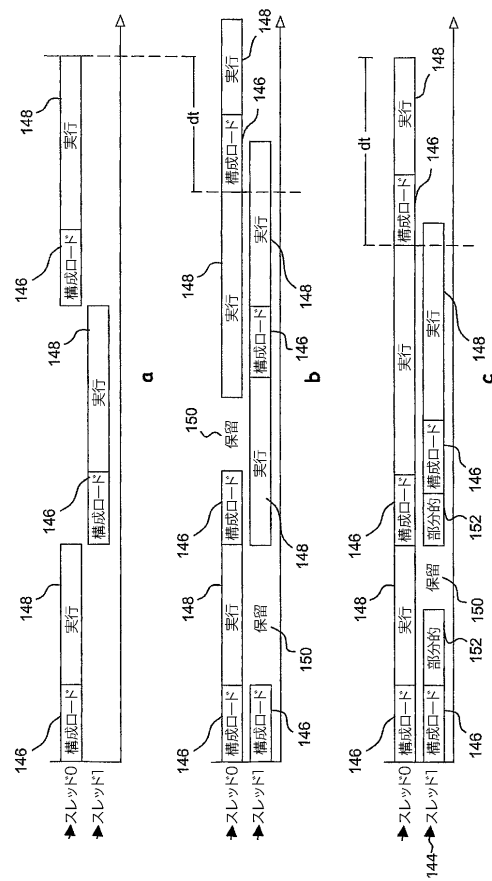
【図 12】



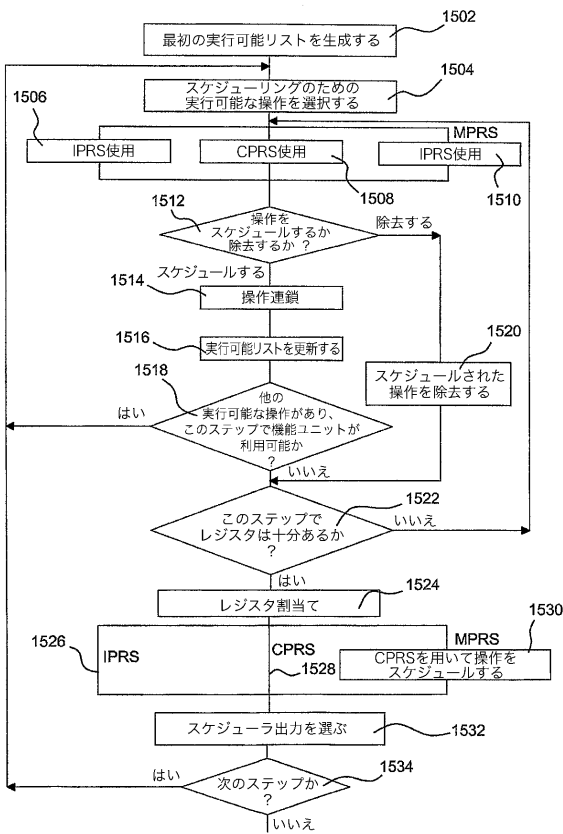
【図 13】



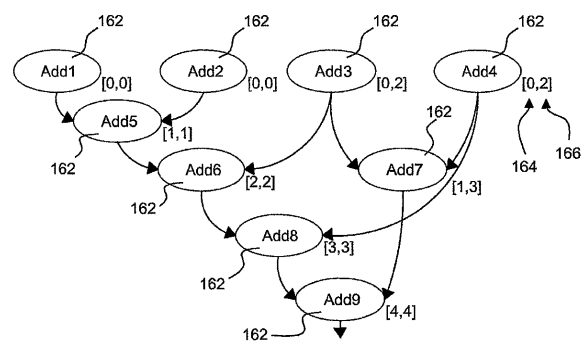
【図 14】



【図 15】



【図 16】



フロントページの続き

- (72)発明者 ハワム、サミ
イギリス国、エディンバラ、フォレスト ヒル 1/4
- (72)発明者 ヌージアス、イオアニス
イギリス国、エディンバラ、コミストン プレース 1/6
- (72)発明者 イ、イン
イギリス国、エディンバラ、ニュー バトル テラス ナンバー 3

合議体

審判長 辻本 泰隆

審判官 高木 進

審判官 須田 勝巳

- (56)参考文献 国際公開第2004/015568 (WO, A1)
特表2005-535055 (JP, A)
特開2000-112758 (JP, A)
特開昭60-215248 (JP, A)
特開2004-7472 (JP, A)
特開平9-50367 (JP, A)
特開平9-181599 (JP, A)
Francisco Barat, Murali Jayapala, Tom Vander Aa, Rudy Lauwereins, Geert Deconinck, and Henk Corporaal, "Low Power Coarse-Grained Reconfigurable Instruction Set Processor", LNCS, Field-Programmable Logic and Applications, 2003年9月, Vol. 2778, p230-239
Pascal Benoit, Lionel Torres, Gilles Sassatelli, Michel, Gaston Canbon, "Automatic task scheduling/loop unrolling using dedicated RTR controllers in coarse grain reconfigurable architectures", Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (LPDPS'05), [online], 2005年4月, [retrieved on 2012-02-28]. Retrieved from the Internet, URL, <<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1419997>>

- (58)調査した分野(Int.Cl., DB名)

G06F 9/30