(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6836065号
(P6836065)
(45)発行日　令和3年2月24日(2021.2.24)　(24)登録日　令和3年2月9日(2021.2.9)

(51)Int.Cl.　F I
G06F 9/50 (2006.01)　G06F 9/50 120Z

請求項の数 10　(全 29 頁)

(21)出願番号　特願2017-34302(P2017-34302)
(22)出願日　平成29年2月27日(2017.2.27)
(65)公開番号　特開2018-142046(P2018-142046A)
(43)公開日　平成30年9月13日(2018.9.13)
審査請求日　令和1年11月12日(2019.11.12)

(73)特許権者　000005223
富士通株式会社
神奈川県川崎市中原区上小田中４丁目１番１号
(74)代理人　100094525
弁理士　土井　健二
(74)代理人　100094514
弁理士　林　恒徳
(72)発明者　タシ　デビッド
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内
(72)発明者　藤澤　久典
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内

最終頁に続く

(54)【発明の名称】情報処理装置、ＰＬＤ管理プログラム及びＰＬＤ管理方法

(57)【特許請求の範囲】
【請求項１】
プログラムを実行するプロセッサと、
前記プロセッサからのコンフィグレーション要求に応じて、前記コンフィグレーション要求が要求する論理回路をコンフィグレーションするリコンフィグレーション領域を有するプログラマブルロジック回路装置（以下ＰＬＤ）を有し、
前記プロセッサは、
前記リコンフィグレーション領域内にコンフィグレーションされ動作中の複数の論理回路のうち、第１の論理回路の並列度を下げて第２の論理回路の並列度を上げる並列度調整を行った場合の前記複数の論理回路の第１の実行時間と、前記並列度調整を行わない場合の前記複数の論理回路の第２の実行時間とを比較し、
前記第１の実行時間が前記第２の実行時間より短い場合、前記ＰＬＤに前記並列度調整の要求を行い、短くない場合、前記ＰＬＤに前記並列度調整の要求を行わない、情報処理装置。
【請求項２】
前記プロセッサは、
前記リコンフィグレーション領域内にコンフィグレーションされ動作中の複数の論理回路のデータ転送量の測定値を取得し、
前記データ転送量の合計が前記ＰＬＤのバスのデータ転送量の上限値に達した場合、前記比較を実行する、請求項１に記載の情報処理装置。

【請求項３】
前記比較では、前記複数の論理回路の前記第１の実行時間の合計と、前記第２の実行時間の合計とを比較し、
前記第１の実行時間の合計値が前記第２の実行時間の合計値より短い場合、前記ＰＬＤに前記並列度調整の要求を行い、短くない場合、前記ＰＬＤに前記並列度調整の要求を行わない、請求項１に記載の情報処理装置。
【請求項４】
前記プロセッサは、さらに、
前記複数の論理回路の前記第１の実行時間と、前記第２の実行時間とを計算する、請求項１に記載の情報処理装置。
【請求項５】
前記プロセッサは、
前記複数の論理回路の前記第１の実行時間として、第１の論理回路の並列度を下げて第２の論理回路の並列度を上げる並列度調整を行った後、前記第２の論理回路の実行が完了後に前記第１の論理回路の並列度を上げる場合に予測される前記複数の論理回路の実行時間を計算する、請求項４に記載の情報処理装置。
【請求項６】
前記複数の論理回路は、データ処理中にメモリアクセスが発生するデータ・インテンシブ処理回路と、データ処理の最初と最後にメモリアクセスが発生するコンピュテーション・インテンシブ処理回路のいずれか一方または両方を含み、
前記プロセッサは、
前記データ・インテンシブ処理回路の並列度をＮ倍にした場合、前記実行時間を１／Ｎ倍になるよう前記第１の実行時間を算出し、
前記コンピュテーション・インテンシブ処理回路の並列度をＮ倍にした場合、前記実行時間を、前記コンピュテーション・インテンシブ処理回路のパイプライン処理におけるイニシエーション・インターバル時間が１／Ｎ倍になるよう前記第１の実行時間を算出する、請求項４に記載の情報処理装置。
【請求項７】
前記複数の論理回路は、データ処理中にメモリアクセスが発生するデータ・インテンシブ処理回路と、データ処理の最初と最後にメモリアクセスが発生するコンピュテーション・インテンシブ処理回路の両方を含み、
前記プロセッサは、
前記第１の論理回路を前記データ・インテンシブ処理回路から選択し、前記第２の論理回路を前記コンピュテーション・インテンシブ処理回路から選択する、請求項１に記載の情報処理装置。
【請求項８】
前記プロセッサは、
前記第１の実行時間が前記第２の実行時間より短くない場合、前記第２の論理回路を変更して、前記比較を再度行う、請求項７に記載の情報処理装置。
【請求項９】
プロセッサからのコンフィグレーション要求に応じて、前記コンフィグレーション要求が要求する論理回路をコンフィグレーションするリコンフィグレーション領域を有するプログラマブルロジック回路装置（以下ＰＬＤ）管理処理をプロセッサに実行させるＰＬＤ管理プログラムであって、
前記管理処理は、
前記リコンフィグレーション領域内にコンフィグレーションされ動作中の複数の論理回路のうち、第１の論理回路の並列度を下げて第２の論理回路の並列度を上げる並列度調整を行った場合の前記複数の論理回路の第１の実行時間と、前記並列度調整を行わない場合の前記複数の論理回路の第２の実行時間とを比較し、
前記第１の実行時間が前記第２の実行時間より短い場合、前記ＰＬＤに前記並列度調整

の要求を行い、短くない場合、前記ＰＬＤに前記並列度調整の要求を行わない、処理を有するＰＬＤ管理プログラム。

【請求項１０】

プログラムを実行するプロセッサと、

前記プロセッサからのコンフィグレーション要求に応じて、前記コンフィグレーション要求が要求する論理回路をコンフィグレーションするリコンフィグレーション領域を有するプログラマブルロジック回路装置（以下ＰＬＤ）を有する情報処理装置の前記ＰＬＤ管理方法であって、

前記リコンフィグレーション領域内にコンフィグレーションされ動作中の複数の論理回路のうち、第１の論理回路の並列度を下げて第２の論理回路の並列度を上げる並列度調整を行った場合の前記複数の論理回路の第１の実行時間と、前記並列度調整を行わない場合の前記複数の論理回路の第２の実行時間とを比較し、

前記第１の実行時間が前記第２の実行時間より短い場合、前記ＰＬＤに前記並列度調整の要求を行い、短くない場合、前記ＰＬＤに前記並列度調整の要求を行わない、ＰＬＤ管理方法。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は，情報処理装置、ＰＬＤ管理プログラム及びＰＬＤ管理方法に関する。

【背景技術】

【０００２】

プログラマブルロジックデバイス（Programmable Logic Device，以下PLDと称する。）は、予め複数の論理回路要素、メモリ回路要素、配線、スイッチ等が形成された集積回路に、所定の処理を実行可能な回路をコンフィグレーションするためのコンフィグレーションデータが設定または書込まれると、所定の処理を実行可能な回路をリコンフィグレーションする。このようなPLDは、例えばFPGA（Field Programmable Gate Array）などであり、コンフィグレーションデータを書き換えることで内部の回路を様々な論理回路にリコンフィグレーション可能なLSIである。以下、PLDの１つであるFPAGを例にして説明する。

【０００３】

プロセッサは、ソフトウエアの所定の処理（例えばジョブ）をハードウエアの専用回路で実行するとき、その専用回路をコンフィグレーションするためのコンフィグレーションデータをFPGAに設定または書込んでFPGA内に専用回路をコンフィグレーションし、その専用回路に所定の処理を実行させる。また、専用回路が所定の処理を終了すると、異なる処理を実行する別の専用回路のコンフィグレーションデータをFPGAに設定または書き込んでFPGA内に別の専用回路をコンフィグレーションし、別の専用回路に異なる処理を実行させる。プロセッサがソフトウエアの所定の処理をFPGAの専用回路に実行させることで、FPGAをプロセッサのアクセラレータとして利用する。これにより、プロセッサを有する情報処理装置（コンピュータ）を省電力化、高機能化できる。

【０００４】

FPGAの大規模化に伴い、FPGA内に複数の論理回路をコンフィグレーションし、複数の論理回路を並行して動作させることができる。また、FPGA内にコンフィグレーションした複数の論理回路を動作させながら、一部の論理回路をリコンフィグレーションして新たな論理回路の動作を開始するなど、複数の論理回路を非同期で動的にリコンフィグレーションし、非同期で並列に動作させることが可能になる。

【０００５】

FPGAに複数の回路をコンフィグレーションすることについては以下の特許文献に開示されている。特許文献５は先願であるが公知例ではない。

【先行技術文献】

【特許文献】

【０００６】

【特許文献１】特開２０１５－１５４４１７号公報
【特許文献２】特開２００４－３２０４３号公報
【特許文献３】特開２０１６－７６８６７号公報
【特許文献４】特開２０１５－２３１２０５号公報
【特許文献５】特願２０１６－２４８２９７
【発明の概要】
【発明が解決しようとする課題】
【０００７】
一方で、複数のユーザが、プロセッサとFPGAを搭載した情報処理装置を使用する場合、複数のユーザのプログラムの特定の処理を、FPGAにコンフィグレーションした複数の論理回路がそれぞれ処理することがある。その場合、複数のユーザのプログラムは、互いの論理回路を意識せず、それぞれの論理回路をFPGA内にコンフィグレーションし、コンフィグレーションされた複数の論理回路がFPGAを部分的に且つ動的に共用する。その結果、FPGAとメモリとの間のバスの使用帯域がバス帯域の上限値に達してバス帯域にボトルネックが発生する場合がある。
【０００８】
バス帯域にボトルネックが発生した場合、所定の論理回路の並列度を低下させ、代わりに別の論理回路の並列度を上昇させ、全体の実行時間の悪化を抑制することが考えられる。並列度を上昇した場合の実行時間の短縮は論理回路の種類によって異なる。
【０００９】
そこで，本発明の目的は，PLDの回路リソースの使用効率を向上する情報処理装置、PLD管理プログラム及びPLD管理方法を提供することにある。
【課題を解決するための手段】
【００１０】
実施の形態の第１の側面は，プログラムを実行するプロセッサと、
前記プロセッサからのコンフィグレーション要求に応じて、前記コンフィグレーション要求が要求する論理回路をコンフィグレーションするリコンフィグレーション領域を有するプログラマブルロジック回路装置（以下ＰＬＤ）を有し、
前記プロセッサは、
前記リコンフィグレーション領域内にコンフィグレーションされ動作中の複数の論理回路のうち、第１の論理回路の並列度を下げて第２の論理回路の並列度を上げる並列度調整を行った場合の前記複数の論理回路の第１の実行時間と、前記並列度調整を行わない場合の前記複数の論理回路の第２の実行時間とを比較し、
前記第１の実行時間が前記第２の実行時間より短い場合、前記ＰＬＤに前記並列度調整の要求を行い、短くない場合、前記ＰＬＤに前記並列度調整の要求を行わない、情報処理装置である。
【発明の効果】
【００１１】
第１の側面によれば，PLDの回路リソースの使用効率を向上することができる。
【図面の簡単な説明】
【００１２】
【図１】本実施の形態における情報処理装置の構成例を示す図である。
【図２】FPGAのコンフィグレーション例を示す図である。
【図３】FPGAのリコンフィグレーション領域の一例を示す図である。
【図４】複数のユーザの論理回路が動的にコンフィグレーションされそして削除される例を説明する図である。
【図５】FPGA内にコンフィグレーションされる論理回路の並列度の制御例を示す図である。
【図６】FPGAにおけるバス帯域のボトルネックを説明する図である。
【図７】第1の実施の形態におけるFPGA管理方法による並列度の制御例を示す図である。

【図８】CI処理回路の場合の実行時間について説明する図である。
【図９】並列度Piを変化させた場合の、CI処理回路の実行時間の違いを示す図である。
【図１０】第1の実施の形態におけるFPGA管理プログラムのフローチャート図である。
【図１１】プロセッサが管理するユーザ回路のパラメータを示す図表である。
【図１２】ユーザ回路の並列度調整処理S8のフローチャート図である。
【図１３】工程S13Aの並列度を増加する処理を示すフローチャート図である。
【図１４】図１２の工程S15,S15B,S15C,S15Dの詳細なフローチャートを示す図である。
【図１５】工程S17の処理を示すフローチャート図である。
【図１６】第1の具体例を示す図である。
【図１７】第２の具体例を示す図である。
【発明を実施するための形態】
【００１３】
図１は、本実施の形態における情報処理装置の構成例を示す図である。情報処理装置であるサーバ１０は、OSやアプリケーションプログラムやミドルウエアプログラムを実行するプロセッサまたはCPU(Central Processing Unit)１１と、DRAMなどのメインメモリ１２を接続するCPUバスなどの第1のバスBUS_1を有する。さらに、サーバ１０は、マウス、キーボード、表示パネルなどのI/Oデバイス（１３）、ネットワークNETに接続されるNIC（Network Interface Card）１４、そして、OS、アプリケーションプログラムAPL、データDATAなどを記憶するHDD（Hard Disk Drive）やSSD（Solid State Drive）などの補助記憶装置１７などを有する。そして、それらと第1のバスBUS_1がPCIバスなどの第２のバスBUS_2を介して接続される。
【００１４】
さらに、サーバ１０は、任意の論理回路をリコンフィグレーション可能なPLD(Programmable Logic Drive)１５と、PLDのコンフィグレーションデータなどを記憶するメモリ１６と、それらを接続するバスである第３のバスBUS_3を有する。PLDは例えばFPGAなどであり、メモリ１６はFPGA用メモリ、第３のバスはFPGAバスである。
【００１５】
たとえば、CPU１１が実行するOSのジョブ管理プログラムが、実行中のアプリケーションプログラム中にFPGA内の論理回路により処理可能なジョブを検出した場合、CPUがその論理回路をコンフィグレーションするためのコンフィグレーションデータをFPGA内に書き込んで（または設定して）論理回路をコンフィグレーションし、その論理回路を実行する。
【００１６】
補助記憶装置１７には、FPGAを管理するFPGA管理プログラムと、論理回路をコンフィグレーションするコンフィグレーションデータC_DATAが記憶される。サーバ１０が起動する時、補助記憶装置内のOS、アプリケーションAPL、FPGA管理プログラムがメインメモリ１２に展開され、プロセッサ１１により実行される。また、補助記憶装置内のコンフィグレーションデータC_DATAはFPGA用メモリに展開される。
【００１７】
FPGA１５は、コンフィグレーションデータを変更することで様々な論理回路をコンフィグレーションすることができるので、サーバ１０が製造された後でも、コンフィグレーションデータを変更することで、様々なジョブの処理をFPGA内にコンフィグレーションした論理回路で高速に処理することができる。
【００１８】
クラウドサービス等において、複数のユーザがそれぞれのアプリケーションプログラムをサーバ１０に実行させる。その結果、サーバ１０のプロセッサ１１は、複数のユーザのアプリケーションプログラムを並列に実行する。そして、それぞれのアプリケーションプログラムの所定の処理（ジョブ）を実行する論理回路が、FPGA１５内に非同期で構築され、構築された複数の論理回路（ユーザ回路）が並列に動作してそれぞれの所定の処理（ジョブ）を実行する。

【００１９】
図２は、FPGAのコンフィグレーション例を示す図である。図２のFPGA１５は、FPGAの第３のバスBUS_3とのバスインターフェース回路BUS_IFと、コンフィグレーションデータの書込み制御及びその他の制御を行う制御回路１５１と、コンフィグレーションデータが書き込まれるコンフィグレーションデータメモリC_RAMと、書き込まれたコンフィグレーションデータにより種々の論理回路がリコンフィグレーションされるリコンフィグレーション領域RC_REGと、内部バスI_BUSを有する。
【００２０】
リコンフィグレーション領域RC_REGには、図示しないが、複数の論理回路要素、メモリ回路要素、配線、スイッチ等が予め形成される。また、リコンフィグレーション領域RC_REGは、論理的にまたは物理的に区分された複数の部分リコンフィグレーションブロックPBに区分される。そして、リコンフィグレーションされる論理回路は、各部分リコンフィグレーションブロックPBに収容できる回路ブロックをコンフィグレーション単位として、単数または複数の各部分リコンフィグレーションブロック内にコンフィグレーションされる。したがって、例えば、コンフィグレーションデータメモリC_RAMは、複数の部分リコンフィグレーションブロックPBに対応する複数の記憶領域に区分され、各記憶領域にコンフィグレーションデータC_DATAが書き込まれると、その記憶領域に対応する部分リコンフィグレーションブロックPBにそれぞれの論理回路がコンフィグレーションされる。
【００２１】
さらに、あるジョブを実行する論理回路（ユーザ回路）が複数の部分リコンフィグレーションブロックPBにコンフィグレーションされる場合がある。その場合は、複数の機能ブロック領域に対応する記憶領域に論理回路をコンフィグレーションするためのコンフィグレーションデータがそれぞれ書き込まれ、各部分リコンフィグレーションブロックにコンフィグレーションされた複数の回路により前述のジョブの処理を実行する論理回路（ユーザ回路）がコンフィグレーションされる。
【００２２】
上記のとおり、FPGA内のリコンフィグレーション領域RC_REGは、複数の部分リコンフィグレーションブロックPBでコンフィグレーションされる。そして、各ユーザのアプリケーションプログラム内の所定の処理（ジョブ）を実行する論理回路は、単一の部分リコンフィグレーションブロックPBにコンフィグレーションされる場合と、複数の部分リコンフィグレーションブロックPBにコンフィグレーションされる場合とがある。
【００２３】
リコンフィグレーション領域RC_REG内にコンフィグレーションされた論理回路には、バスインターフェースBUS_IFを介して、CPUから入力データが入力され、入力データの処理結果がCPUに出力される。また、リコンフィグレーション領域RC_REG内にコンフィグレーションされた複数の論理回路は、内部バスI_BUS、バスインターフェースBUS_IF、及びFPGAバスBUS_3を介して、FPGA用メモリ１６と動作中のデータの送受信を行う。
【００２４】
図３は、FPGAのリコンフィグレーション領域の一例を示す図である。図２に示したとおり、リコンフィグレーション領域RC_REGは、マトリクス状に配置された複数の部分リコンフィグレーションブロックPBに区分される。また、リコンフィグレーション領域RC_REGは、複数の部分リコンフィグレーションブロックPB内に構成される複数の論理回路間のデータ転送や、図２のバスインターフェースBUS_IFと部分リコンフィグレーションブロックPB内にコンフィグレーションされる論理回路との間のデータ転送のための運用回路OCを有する。運用回路OCは、ネットワーク配線と、ネットワークスイッチと、ルーティング回路など含む。
【００２５】
図３の例では、複数の部分リコンフィグレーションブロックPBのうち、左側の３×３の部分リコンフィグレーションブロックPBにコンフィグレーションされた回路によりユーザAの論理回路UC_Aがコンフィグレーションされ、右側の２×４の部分リコンフィグレーシ

ョンブロックPBにコンフィグレーションされた回路によりユーザBの論理回路UC_Bがコンフィグレーションされる。また、回路がコンフィグレーションされていない８個の部分リコンフィグレーションブロックPBが無色で示されている。

【００２６】

図４は、複数のユーザの論理回路が動的にコンフィグレーションされそして削除される例を説明する図である。時間T1では、FPGA内のリコンフィグレーション領域RC_REGには論理回路はコンフィグレーションされていない。次に、時間T2で、ユーザAの論理回路が２つの部分リコンフィグレーションブロックにコンフィグレーションされジョブの実行を開始する。その後、時間T3で、ユーザBの論理回路が６個の部分リコンフィグレーションブロックにコンフィグレーションされ実行開始する。時間T3の後でユーザAの論理回路が処理を完了し、時間T4で、ユーザCの論理回路が４個の部分リコンフィグレーションブロックにコンフィグレーションされ実行開始する。その後、時間T5でユーザBの論理回路が処理を終了し、時間T6でユーザDの論理回路が４個の部分リコンフィグレーションブロックにコンフィグレーションされ実行開始する。それぞれコンフィグレーションされた論理回路は、処理が完了すると、例えば、論理回路をコンフィグレーションしていた部分リコンフィグレーションブロックが開放され、他の論理回路をコンフィグレーション可能な状態に開放される。その場合、例えば、解放された部分リコンフィグレーションブロックに他の論理回路がコンフィグレーションされるまでは、コンフィグレーションデータメモリC_RAM内のコンフィグレーションデータは削除されず、再度同じ論理回路のコンフィグレーション要求が発生すると、コンフィグレーション済みの論理回路が有効化される。

【００２７】

図４に示すとおり、FPGAのリコンフィグレーション領域内には、同じユーザのまたは異なるユーザの異なる論理回路が非同期でコンフィグレーションされ、コンフィグレーションされた論理回路がジョブの実行を行う。そして、前述のサーバ１０内のFPGA管理プログラムが、FPGA内に論理回路をリコンフィグレーションする制御を行う。

【００２８】

図５は、FPGA内にコンフィグレーションされる論理回路の並列度の制御例を示す図である。PLDの１つであるFPGAには、コンフィグレーションデータを設定することで論理回路がコンフィグレーションされ、その論理回路がジョブを実行し、CPUのアクセラレータの機能を有する。しかし、FPGA内の論理回路は、コンフィグレーションデータでリコンフィグレーションされたルックアップテーブルやスイッチング回路でコンフィグレーションされるため、通常のカスタム集積回路よりも動作速度が低い。そのため、FPGA内の論理回路をCPUのアクセラレータとして利用するための１つの方法として、FPGA内に同じ論理回路を複数個コンフィグレーションし、複数個の論理回路で並列動作することが考えられる。

【００２９】

例えば、FPGA管理プログラムを実行するプロセッサは、あるジョブの処理を実行する論理回路をFPGA内にコンフィグレーションする場合、リコンフィグレーション領域RC_REGに空きがあれば、同じ論理回路を複数個コンフィグレーションするようにFPGAを制御し、複数個の同じ論理回路に並列にジョブの実行を行わせる。

【００３０】

図５の例では、時間T11で、FPGA管理プログラムを実行するプロセッサは、ユーザAの論理回路UC_Aを６個の部分リコンフィグレーションブロックにコンフィグレーションし、ユーザBの論理回路UC_Bを２個の部分リコンフィグレーションブロックにコンフィグレーションする。そして、その後の時間T12では、プロセッサは、２つ目のユーザBの論理回路UC_B2を２個の部分リコンフィグレーションブロックにコンフィグレーションし、２個の論理回路UC_B，UC_B2に並列動作を行わせる。同様に、時間T13では、プロセッサは、２つ目のユーザAの論理回路UC_A2を６個の部分リコンフィグレーションブロックにコンフィグレーションし、２個の論理回路UC_A，UC_A2に並列動作を行わせる。これにより、FPGA内の論理回路の動作速度を高速化することができる。

【００３１】

例えば、ユーザの論理回路が加算器であり、1個の加算器がＮサイクルで演算を完了する場合、２個の加算器をコンフィグレーションして並列に加算演算すれば、Ｎ／２サイクルで演算を完了する。これが論理回路の並列度を増加してジョブの実行時間を短縮する例である。

【００３２】

［バス帯域のボトルネック］

図６は、FPGAにおけるバス帯域のボトルネックを説明する図である。FPGA内にコンフィグレーションされた論理回路は、FPGAバスBUS_3を介して図１、２に示したFPGA用メモリ１６にアクセスする。FPGA用メモリ１６には、リコンフィグレーションされる論理回路のコンフィグレーションデータと、コンフィグレーションされた論理回路がアクセスするデータとが格納される。したがって、FPGA管理プログラムを実行するプロセッサが、FPGAにある論理回路のコンフィグレーションを要求したとき、FPGA内の制御回路がFPGA用メモリにアクセスし、論理回路のコンフィグレーションデータをダウンロードする。さらに、FPGA内にコンフィグレーションされた論理回路がそれぞれのジョブを実行すると、各論理回路がFPGA用メモリ内に格納されているデータにアクセスする。したがって、PFGA内にコンフィグレーションされた論理回路は、FPGAバスBUS_3が提供可能な帯域のうち、それぞれのデータ転送量に対応する帯域を使用する。

【００３３】

図６の例では、時間T21で、FPGAのリコンフィグレーション領域RC_REG内に、ユーザ１，３，４のユーザ回路UC_1、UC_3、UC_4が並列度１でコンフィグレーションされ、ユーザ２のユーザ回路UC2が並列度２でコンフィグレーションされている。FPGAバスBUS_3の提供可能な帯域（データ転送量の上限値）が例えば1350MB/Sであり、ユーザ１，２，３，４のユーザ回路UC_1，UC_2、UC_3，UC_4の平均データ転送量がそれぞれ100MB/S、200MB/S、200MB/S、300MB/Sとする。図６の状態では、コンフィグレーションされたユーザ回路UC_1～UC_4の平均データ転送量の合計値が100+200*2+200+300=1000MB/Sである。したがって、合計値1000MB/Sは上限値1350MB/Sに達していない。この状態では、FPGAバスBUS_3にボトルネックは発生しておらず、各ユーザ回路は予測されたデータ転送量で動作し、ジョブの実行時間も予測された実行時間になる。

【００３４】

一方、時間T22では、FPGA管理プログラムを実行するプロセッサが、FPGAの制御回路にユーザ２の論理回路UC_2の並列度を４に増加する要求を行い、論理回路UC_2の並列度が４に増加されている。この場合、プロセッサは、リコンフィグレーション領域内に論理回路UC_2の並列度を４に増加するために必要な部分リコンフィグレーションブロックの空きがあり、且つ論理回路UC_2のデータ転送量の予測が低くかったため、並列度を４に増加してもバス帯域の上限値を超えることはないと予測されていたと考えられる。

【００３５】

しかしながら、実際には、動作中の論理回路のデータ転送量の合計値が100+200*4+200+300=1400MB/Sとなり、FPGAバスの上限値1350MB/Sを超えてしまい、PFGAバスの帯域にボトルネックが発生する場合がある。その結果、並列度を４に増加されたユーザ２の論理回路UC_2は、ジョブの実行に必要な帯域を使用することができず、ユーザ２の論理回路UC_2による１つのジョブの実行時間は、予測した実行時間より長くなる。

【００３６】

上記のとおり、FPGA内のリコンフィグレーション領域RC_REG内の部分リコンフィグレーションブロックに空きがある場合、論理回路の並列度を増加させたとしても、FPGAバスの帯域が足りず論理回路のデータ転送量の合計値がバス帯域の上限値に達してバス帯域にボトルネックが発生する場合がある。その結果、並列度を増加した論理回路の性能は上がらず、リコンフィグレーション領域内の部分リコンフィグレーションブロックを無駄に使用することになる。

【００３７】

［第１の実施の形態］

図７は、第1の実施の形態におけるFPGA管理方法による並列度の制御例を示す図である。このFPGA管理方法では、ユーザの論理回路に、１つのジョブを実行するのに要する実行時間を測定する実行時間測定回路と、FPGAバスへのアクセスを監視しバスアクセスの単位時間当たりのデータ転送量の平均値を測定するデータ転送量測定回路とが含まれる。これらの測定回路は、FPGAのコンフィグレーションデータによりコンフィグレーション可能である。そして、FPGAの制御回路は、ユーザの論理回路をコンフィグレーションデータでコンフィグレーションするときに同時に測定回路もコンフィグレーションデータでコンフィグレーションする。または、測定回路を予め部分リコンフィグレーションブロックに形成しておき、部分リコンフィグレーションブロックにコンフィグレーションされる論理回路の測定回路として使用してもよい。

【００３８】

そして、FPGA管理プログラムを実行するプロセッサは、FPGA内のリコンフィグレーション領域内にコンフィグレーションされ動作中の複数の論理回路のデータ転送量の測定値を取得し、取得したデータ転送量の測定値の合計がFPGAバスのデータ転送量の上限値を超えない範囲で、リコンフィグレーション領域内にコンフィグレーションする複数の論理回路それぞれの並列数を増加する。

【００３９】

また、第1の実施の形態では、プロセッサは、取得したデータ転送量の測定値の合計がFPGAバスのデータ転送量の上限値に達した場合、複数の論理回路のうち、所定の条件を満たす論理回路の並列度を減少する。そして、プロセッサは、並列度を減少させた論理回路以外の別の論理回路のいずれかの並列度を、FPGAバスのデータ転送量の上限値を超えない範囲で、増加する。これにより、並列度を増加した論理回路の動作が予測より短い時間で終了することが期待できる。プロセッサは、並列度を増加した別の論理回路の動作終了後、上限値を超えない範囲で、並列度を減少した論理回路の並列度を増加する。これにより、当該論理回路の動作が予測より短い時間で終了することが期待できる。

【００４０】

図７の例で説明すると、プロセッサが、図６の時間T22の状態で動作中の論理回路のデータ転送量の測定値を取得し、その合計値がFPGAバスの限界値に達していることを検出する。これにより、図7の時間T23に示すとおり、プロセッサは、バスのボトルネックの原因と考えられるユーザ２の論理回路UC_2の並列度を４から２に減らす。その後、論理回路のデータ転送量が低いユーザ１の論理回路UC_1の並列度を１から４に増加する。この結果、動作中の論理回路のデータ転送量の測定値の合計が、100*4+200*2+200+300=1300MB/SとなりFPGAバスの上限値1350MB/S未満になり、バス帯域のボトルネックは解消される。

【００４１】

これにより、ユーザ１の論理回路UC_1の動作時間が短くなり短時間で動作完了することが予測される。そして、時間T24に示すとおり、ユーザ１の論理回路UC_1の動作が完了すると、プロセッサは、並列度を減少したユーザ２の論理回路UC_2を優先的に並列度２から４に増やす。そして、プロセッサは、動作中の論理回路のデータ転送量の測定値を取得し、測定値の合計200*4+200+300=1300MB/SがFPGAバスの上限値1350MB/S未満であることを検出する。この状態でも、バス帯域のボトルネックが解消され、論理回路が十分な動作を行うことができる。

【００４２】

［データ処理タイプCIとDI］

図７の時間T23でユーザ回路UC_1の並列度を増やした場合、ユーザ回路UC_1のデータ処理パターンによって、実行時間の短縮度が異なる。例えば、データ処理パターンには、データ・インテンシブ（DI：Data Intensive）と、コンピュテーション・インテンシブ（CI：Computation Intensive）とがある。一般に、DI処理回路の場合は、並列度を増やすと実行時間がそれに比例して短くなるが、CI処理回路の場合は、並列度を増やしても実行時間の短縮は少ない場合がある。

【００４３】

DI処理回路の場合、回路が動作中、データの読み出し（ロード）とデータの書込み（ストア）が常時発生し、ユーザ回路内のメモリアクセスを行うロードストアユニット（LSU）の稼働率が高く、バスの使用帯域も高くなる。したがって、バスボトルネックの原因となる。バスボトルネックが発生した場合、DI処理回路の性能が悪化し、ジョブの実行時間は長くなる傾向にある。また、DI処理回路は、一般に、並列度をＮ倍にすると実行時間は1/N倍に短縮する。

【００４４】

一方、CI処理回路の場合、メモリアクセスが回路動作の最初と最後に発生する。つまり、回路動作の最初にデータ処理に必要な入力データがメモリから読み出され、回路動作の最後にデータ処理後の出力データがメモリに書込まれる。データ処理中メモリアクセスはほとんど発生しない。したがって、CI処理回路は、バス帯域を使用しない時間が比較的長く、バスの使用帯域は小さく、バスボトルネックが発生してもCI処理回路の性能は悪化せず、ジョブの実行時間はあまり変わらない。また、CI処理回路は、一般に、並列度を増やしても実行時間の短縮は少ない。

【００４５】

図７の場合、時間T23でユーザ１の論理回路UC_1がDI処理回路の場合、その並列度を増やすと、ジョブの実行時間は増加率に応じて短くなるが、一方、論理回路UC_1がCI処理回路の場合、その並列度を増やしてもジョブの実行時間はあまり短縮されない。この理由は、通常、ユーザ回路はパイプライン構造を有し、パイプライン構造では既にデータ処理が並列化されるからである。そのため、CI処理回路の場合は、回路の並列度を増やしても、パイプラインのイニシエーション・インターバル（Initiation Interval）しか実行時間が短縮されない。一方、DI処理回路の場合は、回路の並列度を増やすと、データ処理中のメモリアクセスも並列化され、通常、並列度をＮ倍にすると実行時間は1/Nに短縮される。

【００４６】

図８は、CI処理回路の場合の実行時間について説明する図である。CI処理回路の実行時間は、前述の実行回数Nと、1回の実行で処理されるデータセットSと、入力データのロード（読み出し）時間T_{LD}と、出力データのストア（書込み）時間T_{ST}と、1つのデータセットの処理時間T_{COMP_SINGLE}と、複数のデータの処理開始間隔であるイニシエーション・インターバルT_{II}とで予測することができる。

【００４７】

図８に示されるとおり、CI処理回路は、実行開始時に入力データをメモリからロードする処理LDを行い、入力データの計算処理COMPをS回パイプライン処理し、最後に出力データをメモリにストアする処理STを実行する。そして、S回のパイプライン処理は、それぞれイニシエーション・インターバルT_{II}の間隔で処理開始される。１つのデータセットの計算処理COMPの実行時間はT_{COMP_SINGLE}であり、データセットS回のイニシエーション・インターバルT_{II}はΔiである。前述のとおり、CI処理回路では、データ処理の最初と最後にデータロードLDとデータストアSTが発生し、データ処理中COMPはメモリアクセスはほとんど発生しない。

【００４８】

そこで、１つのジョブに含まれるN回の実行回数のうちｉ番目の1回の実行時間Tiは、次のとおりである。

$$Ti = T_{LD} + \Delta i + T_{COMP\_SINGLE} + T_{ST} \qquad \text{式１}$$

ここで、Δiは、T_{II}の（S-1）倍であるので、次のとおりである。

$$\Delta i = T_{II} * (S-1)$$

【００４９】

さらに、CI処理回路の並列度をPiとすると、Δiは、次のとおり並列度Piに応じて短縮される。

$$\Delta i = T_{II} * \{(S/Pi)-1\}$$

【００５０】

図９は、並列度Piを変化させた場合の、CI処理回路の実行時間の違いを示す図である。図９の例では、データセットSがS=4である。そして、並列度PiがPi=1、Pi=2、Pi=S=4の場合のΔが示される。Pi=1の場合は、Δｉ＝３＊T_{II}、Pi=2の場合は、Δｉ＝T_{II}、そして、Pi=S=4の場合は、Δｉ＝０となる。

【００５１】

以上、CI処理回路の場合のｉ番目の実行時間Tiは、並列度Piとすると、上記の式１により次のとおりとなる。

【００５２】

【数１】

$$\Delta_i = T_{II} \times \left(\frac{S}{P_i} - 1\right)$$

$$\begin{aligned} T_i &= T_{LD} + \Delta_i + T_{COMP\_SINGLE} + T_{ST} \\ &= T_{LD} + T_{II} \times \left(\frac{S}{P_i} - 1\right) + T_{COMP\_SINGLE} + T_{ST} \end{aligned} \quad (式1)$$

【００５３】

そして、データ処理時間に比較してメモリアクセス時間T_{LD}、T_{ST}は十分に小さいので、式１からT_{LD}、T_{ST}を省略すると、実行時間Tiは次のとおりとなる。

【００５４】

【数２】

$$T_i = T_{II} \times \left(\frac{S}{P_i} - 1\right) + T_{COMP\_SINGLE} \quad (式2)$$

【００５５】

そこで、回路の並列度PiをPjに変更した場合の実行時間の差分（Pi<Pjの場合の短縮時間）は、次のとおりとなる。

【００５６】

【数３】

$$\begin{aligned} T_j - T_i &= \left(T_{II} \times \left(\frac{S}{P_j} - 1\right) + T_{COMP\_SINGLE}\right) - \left(T_{II} \times \left(\frac{S}{P_i} - 1\right) + T_{COMP\_SINGLE}\right) \\ &= T_{II} \times \frac{S \cdot (P_i - P_j)}{P_i \cdot P_j} \end{aligned} \quad (式3)$$

【００５７】

更に、式２において、CI処理回路の並列度を最大のPi＝Sにすると、実行時間Tiは次のとおり最短実行時間T_{min}となる。

【００５８】

【数４】

$$T_{min} = T_{II} \times \left(\frac{S}{P_i} - 1\right) + T_{COMP\_SINGLE} \quad (式2)$$

$$If\ P_i = S,$$

$$T_{min} = T_{II} \times \left(\frac{S}{S} - 1\right) + T_{COMP\_SINGLE} = T_{COMP\_SINGLE} \quad (式4)$$

【００５９】

そして、CI処理回路の実行回数Nのトータル実行時間T_{CI_total}は、ｉ回目の実行時の並列度をPiとすると、次のとおりである。

【００６０】

【数５】

$$T_{CI\_total} = \sum_{i=1}^{N} T_i = \sum_{i=1}^{N} \left(T_{II} \times \left(\frac{S}{P_i} - 1 \right) + T_{COMP\_SINGLE} \right) \quad (式5)$$

【００６１】

上記の式５において、並列度PiからPjに変更した場合の変更前のトータル実行時間$T_{CI_total_before}$と、変更後のトータル実行時間$T_{CI_total_after}$はそれぞれ、式５の並列度をPiとPjにした式になる。

【００６２】

さらに、N回の実行全てで並列度Piが最大値S（Pi=S）になる場合（[1:N]の全ての要素iにおいて、Pi＝S）、トータル実行時間は以下の最小値$T_{CI_total_min}$になる。

【００６３】

【数６】

$$T_{CI\_total\_min} = \sum_{i=1}^{N} T_{min} = N \cdot T_{COMP\_SINGLE} \quad (式6)$$

【００６４】

次に、DI処理回路の並列度と実行時間について説明する。DI処理回路は、データロードLDとデータストアSTが、各データセットのデータ処理COMP中にも発生する。したがって、DI処理回路は、CI処理回路のように実行時間Tiの大半が演算処理時間T_{COMP_SINGL}とはならない。そこで、本実施の形態では、DI処理回路は、並列度をN倍にすれば実行時間は1/N倍に短縮されると仮定して、実行時間の計算を行う。

【００６５】

本実施の形態では、バスボトルネックが発生して、所定のユーザ回路の並列度を減らしてバスボトルネックを解消し、その代わりに所定のユーザ回路とは別のユーザ回路の並列度を増やす。但し、並列度調整前の全ユーザ回路の実行時間の合計T_{total_before}と、並列調整後の全ユーザ回路の実行時間の合計T_{total_after}とを比較して、並列度の調整を実行するか否か判断する。すなわち、調整後のT_{total_after}が調整前のT_{total_before}より短くなる場合、ボトルネック解消のための並列度の調整を実行し、短くならない場合、並列度の調整を実行しない。

【００６６】

上記のT_{total_before}とT_{total_after}は、次のとおり、いずれもCI処理回路及びDI処理回路の実行時間の合計である。

$T_{total\_before} = T_{DI-total\_before} + T_{CI-total\_before}$

$T_{total\_after} = T_{DI\_total\_after} + T_{CI\_total\_after}$

【００６７】

CI処理回路の並列度の調整後の実行時間は、前述の式５、６により予測される。一方、DI処理回路の並列度の調整後の実行時間は、調整前の実行時間の測定値を並列度N倍の逆数1/N倍して予測される。また、CI処理回路とDI処理回路の区別は、例えば、処理プログラムのコンパイル時に判定することができる。または、使用帯域の測定値から、帯域の使用がデータ処理の最初と最後だけ発生する回路をCI処理回路、それ以外をDI処理回路と判定できる。

【００６８】

［FPGA管理プログラムの概略処理］

図１０は、第1の実施の形態におけるFPGA管理プログラムのフローチャート図である。例えば、OS（Operating System）のジョブ管理プログラムは、プロセッサが実行するユーザのアプリケーションプログラムのジョブを監視し、ジョブの処理がFPGA内の論理回路で実行可能な場合、プロセッサに新ユーザ回路のコンフィグレーション要求の割込みを発生する。

【００６９】

FPGA管理プログラムを実行するプロセッサは、OS（Operating System）から新たなユーザ回路をコンフィグレーションする要求を受信すると（S1のYES）、次のように要求を処理する。まず、プロセッサは、FPGAのリコンフィグレーション領域の総面積から動作中のユーザ回路の総面積を減じた値が、新たなユーザ回路の面積より大きいか否か判定する（S2）。FPGAのリコンフィグレーション領域の総面積は、例えば部分リコンフィグレーションブロックPBの数であり、動作中のユーザ回路の総面積は、例えば動作中のユーザ回路がコンフィグレーションされている部分リコンフィグレーションブロックの数である。

【００７０】

工程S2の判定がYESの場合、プロセッサは、FPGAに新ユーザ回路のコンフィグレーションを要求する（S3）。そして、FPGAから新ユーザ回路のコンフィグレーション完了通知があると（S4のYES）、プロセッサはFPGAにユーザ回路によるジョブ開始を通知する（S5）。一方、工程S2の判定がNOの場合、プロセッサは、FPGAに新ユーザ回路のコンフィグレーションを要求せず、新たに回路コンフィグレーション要求を要求キュー（要求の待ち行列）に格納する（S9）。要求キュー内の要求は、次のサイクルで前述の工程S1で新ユーザ回路構築要求としてチェックされる。

【００７１】

さらに、プロセッサは、FPGAからユーザ回路のジョブの実行完了通知を受信すると（S6）、FPGAにジョブの実行が完了したユーザ回路の開放通知を行う（S7）。これにより、FPGA内の制御回路は、リコンフィグレーション領域内にコンフィグレーションされたユーザ回路を解放状態にする。

【００７２】

さらに、プロセッサは、ユーザ回路の並列度調整処理S8を実行する。ユーザ回路の並列度調整処理については後述する。そして、プロセッサは、上記の工程S1～S8を繰り返し実行する。

【００７３】

［ユーザ回路のパラメータ］

並列度調整処理S8の説明をする前に、まず、プロセッサが管理するユーザ回路の各種パラメータの例について説明する。

【００７４】

図１１は、プロセッサが管理するユーザ回路のパラメータを示す図表である。図１１の図表に、FPGAのリコンフィグレーション領域内にコンフィグレーションされているユーザ回路UC_1，UC_2，UC_3，UC_4それぞれについて、論理回路の並列度P、予測コンフィグレーション時間CT_E、予測実行時間ET_E、予測使用帯域BD_E、測定実行時間ET_M、測定使用帯域BD_Mの値が示されている。

【００７５】

予測コンフィグレーション時間CT_Eは、論理回路のコンフィグレーションデータをFPGAメモリからダウンロードしてFPGA内のコンフィグレーションデータメモリC_RAMに設定するのに要する時間の予測値である。予測実行時間ET_Eは、論理回路による１つのジョブの実行完了までの時間の予測値である。予測使用帯域BD_Eは、論理回路がジョブ実行中に使用する単位時間当たりのバス帯域（データ転送量）の予測値であり、単位はMB/Sである。

【００７６】

一方、測定実行時間ET_M、測定使用帯域BD_Mは、論理回路に設けられた実行時間測定回路とデータ転送量測定回路それぞれの測定値である。

【００７７】

また、FPGAバスの帯域の上限値をBD_Lとする。この帯域上限値BD_Lは、FPGAバスの帯域であり、リコンフィグレーション領域にコンフィグレーションされた論理回路のFPGAバスへのデータ転送量の合計がこの帯域上限値BD_Lを超えることはできない。したがって、リコンフィグレーション領域にコンフィグレーションされた論理回路のデータ転送量の合計が帯域上限値BD_Lに達している場合、バス帯域にボトルネックが発生しているとみなすことができる。

【００７８】
さらに、図１１のパラメータは、ユーザ回路UC_1, UC_2, UC_3, UC_4それぞれについて、回路の処理パターンを示す回路タイプCI（Computation Intensive,コンピュテーション・インテンシブ）、DI（Data Intensive,データ・インテンシブ）と、ジョブに対するユーザ回路の実行回数Nと、１回の実行で処理されるデータセット数Sと、入力データの読み取り時間T_{LD}と、出力データの書込み時間T_{ST}と、1個のデータセットの処理時間T_{COMP_SINGLE}と、パイプライン回路のイニシエーション・インターバル（Initiation interval）時間T_{II}を有する。

【００７９】
ユーザの論理回路の並列度調整処理S8では、プロセッサは、図１１に示した値に基づいてFPGAのリコンフィグレーション領域内のユーザの論理回路の並列度を制御する。

【００８０】
図１２は、ユーザ回路の並列度調整処理S8のフローチャート図である。FPGA管理プログラムを実行するプロセッサは、一定時間待機するたびに（S10のYES）、FPGA内にコンフィグレーションされているユーザ回路の実行時間測定回路と使用帯域測定回路が測定中の測定実行時間ET_Mと測定使用帯域BD_Mを、両回路から読み出すまたはFPGA内の制御回路１５１から受信する（S11）。

【００８１】
［並列度の増加制御（１）］
そして、プロセッサは、FPGAバスの帯域上限BD_Lからユーザ回路の測定使用帯域の合計値を減じた値が、FPGA内にコンフィグレーション中のユーザ回路のいずれかの並列度を増加するために必要な最小帯域より大きいか否かを判定する（S12）。工程S12の判定がYESであれば、プロセッサは、以下に示す式１、式２を満たす範囲で、ユーザ回路の並列度を増加する（S13A）。

【００８２】
図１３は、工程S13Aの並列度を増加する処理を示すフローチャート図である。まず、プロセッサは、複数（ｎ個）のユーザ回路を所定の順（例えば測定使用帯域BD_Mが小さい順）にソートする（S131）。このソートされたユーザ回路の順番を係数ｉ= 1～nとする。そして、プロセッサは、ソートされた順番で、つまり係数順に、係数ｉ= 1～ｎの各ｉついて(S132-S135)、処理対象のi番目のユーザ回路の並列度Piを１つ増加した後の並列度PXi（＝Pi + 1）で以下の式１, 式２を満たすか否か判定する（S133）。

【００８３】
式１、式２は図１１に示されるが以下のとおりである。

$$\Sigma\ (BD\_Mj/Pj)\ *PXj\ <\ BD\_L \qquad 式１$$

$$\Sigma\ (Aj*PXj)\ \leqq\ A\_L \qquad 式２$$

ここで、Σは全ユーザ回路j=1～nの合計である。また、式１、式２のPXjは、j=iならPXj=Pj + 1、j≠iならPXj=Pjとなり、処理対象のi番目のユーザ回路だけ並列度Pjを＋１増加し、i番目ではない他のユーザ回路は増加しない並列度Pjのままである。

【００８４】
つまり、n=4,i=2の場合の式１は次の通りである。

$$(BD\_M1/P1)*P1 + (BD\_M2/P2)*PX2 + (BD\_M3/P3)*P3 + (BD\_M4/P4)*P4 < BD\_L$$

上記の左辺の第１項は(BD_M1/P1)*P1=BD_M1であり、第３, ４項も同様であるから、よって、

$$BD\_M1 + (BD\_M2/P2)*PX2 + BD\_M3 + BD\_M4 < BD\_L$$

【００８５】
さらに、式２のAjは並列度１のユーザ回路の回路面積（例えば、部分リコンフィグレーションブロックの数）、A_Lはリコンフィグレーション領域の総回路面積（例えば、部分リコンフィグレーションブロックの総数）である。n=4,i=2の場合の式２は次の通りである。

$$A1*P1 + A2*PX2 + A3*P3 + A4*P4 \leqq A\_L$$

【００８６】
式１を満たす場合、処理対象のi番目のユーザ回路だけその並列度Piを１つ増加した後の全ユーザ回路の使用帯域の合計が、FPGAバスの帯域上限値BD_Lより小さいことを意味する。式１において(BD_M2/P2)*PX2は、測定使用帯域は並列度に比例することを意味する。一方、式２を満たす場合、処理対象のi番目のユーザ回路だけその並列度Piを１つ増加した後の全ユーザ回路の使用面積の合計が、FPGAの総回路面積A_L以下であることを意味する。
【００８７】
工程S133の判定がYESなら（S133のYES）、プロセッサは、増加後の並列度PXiをそのユーザ回路UC_iの並列度Piに設定する（S134）。係数ｉ=1～ｎの全てにおいて工程S133がYESの場合、全てのユーザ回路の並列度Piが＋１されたことを意味する。
【００８８】
一方、係数iが１～ｎのいずれかで工程S133の判定がNOなら（S133のNO）、S132～S135のループを抜ける。すなわち、ユーザ回路の順に並列度を＋１増加し、あるユーザ回路で工程S133の判定がNOになると、ループS132～S135の処理を終了する。
【００８９】
そして、CPUは、ユーザ回路UC_iを新しく設定した並列度Piでリコンフィグレーションする要求をFPGAに行い、そのユーザ回路のリコンフィグレーション完了通知受信後、そのユーザ回路のジョブの実行再開を通知する（S137）。
【００９０】
図１３において、測定使用帯域が小さい順にソートし、測定使用帯域が小さいユーザ回路を優先的に並列度を増加させ、あるユーザ回路で式１を満たさない場合、再度、測定使用帯域が小さい回路の並列度を増加できるか否か判定するようにしてもよい。その場合、一般的に測定使用帯域が小さいほど並列度を＋１増加したときの使用帯域の増加量も小さい傾向にあるので、バス帯域の上限未満に抑えることができる。そこで、かかるユーザ回路の並列度をより増加させてジョブの実行時間をより短縮させ、より早くジョブの実行を完了させるためである。ユーザ回路のジョブ実行が完了すれば、その後他のユーザ回路の並列度を増加させてそれらのジョブの実行時間も短縮できる場合がある。
【００９１】
［バス帯域のボトルネックの原因と推定されるユーザ回路の並列度の低下と、他のユーザ回路の並列度の増加］
図１２に戻り、工程S12での判定がNOの場合、プロセッサは、測定使用帯域の合計がFPGAバスの帯域上限に達しているか否か判定する（S14）。この工程S14の判定がYESの場合、FPGAバスの帯域にボトルネックが発生していることを意味する。
【００９２】
そこで、プロセッサは、使用帯域が帯域上限未満になるよう、所定のユーザ回路UC_MAXの並列度を低下させる（S15）。所定のユーザ回路UC_MAXとして、第1の例として、測定使用帯域が最も大きいユーザ回路が選択される。プロセッサは、並列度の低下量として、所定のユーザ回路UC_MAXの予測使用帯域を算出し、他のユーザ回路の測定使用帯域との合計が帯域上限に満たないような値を選択する。測定使用帯域が大きいユーザ回路は、一般にDI処理回路が選択される場合が多い。
【００９３】
第２の例として、所定のユーザ回路UC_MAXとして、実行時間ET_Eと測定実行時間ET_Mの差分が最も大きいユーザ回路が選択される。このようなユーザ回路は、バスボトルネックにより予測使用帯域BD_EほどFPGAバスの帯域を使用することができていない蓋然性が高い。したがって、かかるユーザ回路の並列度を低下させることで、バスボトルネックによりユーザ回路の一部が十分に動作せずFPGA内に無駄にコンフィグレーションされている状況を改善することができる。
【００９４】
並列度を低下させるターゲットのユーザ回路の選択は、第３の例として、予測使用帯域

BD_Eと測定使用帯域BD_Mの差分が最大のユーザ回路を選択してもよい。この場合、差分が最大のユーザ回路は、バスボトルネックにより予測使用帯域BD_EほどFPGAバスの帯域を使用することができていないユーザ回路であるため、かかるユーザ回路を、並列度を減少させるターゲットに選択する。

【００９５】

さらに、第４の例として、並列度が最大のユーザ回路を選択して並列度を減少させてもよい。この場合、並列度が最大に制御されているユーザ回路は、他のユーザ回路よりもより優遇されているユーザ回路といえるので、かかるユーザ回路を、並列度を減少させるターゲットに選択する。

【００９６】

プロセッサは、所定のユーザ回路UC_MAXの並列度を低下させると共に、所定のユーザ回路UC_MAX以外の他のユーザ回路の並列度を増加する（S15）。並列度を増加させるユーザ回路の選択は、様々な例が考えられる。第1の例では、図１３と同様に、任意の順に他のユーザ回路をソートし、式１，式２を満たす範囲で順番に並列度を増加するようにする。つまり、FPGA内の所定のユーザ回路UC_MAX以外の他のユーザ回路のうち、DI処理回路とCI処理回路の区別をせず、任意のユーザ回路の並列度を増加する。

【００９７】

第２の例では、CI処理回路を優先的に選択して並列度を増加する。CI処理回路は使用帯域が小さいので、帯域上限未満を満たしつつ実行時間を短縮できる可能性がある。但し、CI処理回路の実行時間の短縮は、前述の通りあまり大きくない場合がある。

【００９８】

第３の例では、CI処理回路のうち並列度の増加による実行時間の短縮の程度が大きい回路を選択し、さらに、一部のDI処理回路を選択し、選択した両CI処理回路とDI処理回路の並列度を増加する。この場合、並列度が増加された回路の実行時間が短縮され、実行完了後にバスボトルネック解消のために並列度を低下した所定のユーザ回路の並列度を増加して、合計実行時間を短縮できれば望ましい。

【００９９】

次に、プロセッサは、工程S15での並列度の増加及び低下の調整を行わない場合の全てのユーザ回路のジョブ完了までの実行時間の合計、調整前（未調整）合計実行時間T_{total_before}と、前記調整を行った場合の同実行時間の合計、調整後（調整済）合計実行時間T_{total_after}とを計算し、$T_{total_after}<T_{total_before}$か否かを判定する（S15B）。

【０１００】

判定結果がYESの場合、プロセッサは、並列度の調整を実行する（S15C）。判定結果がNOの場合、プロセッサは、並列度の調整を実行しない（S15D）。つまり、バスボトルネック状態を許容したまま、ユーザ回路の並列度の調整を行わず、未調整のままにする。FPGA内のユーザ回路は、処理実行中FPGA内にコンフィグレーションされる。したがって、全てのユーザ回路のジョブ完了までの実行時間の合計が短くなると、FPGAの回路リソースの使用効率が高くなることを意味する。プロセッサが工程S15Bの判定を行うことで、バスボトルネック状態の解消を行うか否かを、FPGAの回路リソースの使用効率が高くなるか否かの観点で判断することができる。

【０１０１】

図１４は、図１２の工程S15,S15B,S15C,S15Dの詳細なフローチャートを示す図である。プロセッサは、所定のユーザ回路UC_MAXの並列度を下げる処理として、測定使用帯域が最大のユーザ回路UC_MAXを抽出し（S151）、そのユーザ回路UC_MAXの並列度を下げて、新たな使用帯域を計算により予測する。新たな使用帯域の予測は、例えば、並列度を1/Nに下げれば、使用帯域も1/N倍に下がると見積もる。

【０１０２】

そして、プロセッサは、予測使用帯域が上限未満になるか否か判定し（S153）、未満にならない場合は、再度工程S152を実行し更に並列度を低下させる。未満になる場合、プロセッサは、帯域上限と全ユーザ回路の予測使用帯域の合計との差分が、所定の基準値Vth

を超えているか否か判定する（S154）。この所定の基準値Vthを超えている場合は、ユーザ回路UC_MAX以外の他のユーザ回路の並列度を増加し、そのユーザ回路の新たな使用帯域を予測する（S155）。

【０１０３】

そして、プロセッサは、並列度を低下及び上昇する調整前の並列度（旧並列度）での全ユーザ回路の実行時間の合計値T_{total_before}を予測し、同時に、並列度を低下及び上昇する調整後の並列度（新並列度）での全ユーザ回路の実行時間の合計値T_{total_after}を予測する（S156）。さらに、プロセッサは、予測値を比較して、$T_{total_after}<T_{total_before}$か否かを判定する（S157）。

【０１０４】

この判定結果がYESであれば（S157のYES）、プロセッサは、ユーザ回路の新並列度をFPGAに要求し、回路リコンフィグレーション完了通知受信後、並列度を更新されたユーザ回路のジョブの再開を通知する（S161）。前述の工程S154の判定結果がNOの場合も、プロセッサは工程S161を実行する。そして、プロセッサは、ユーザ回路UC_MAXを並列度低下リストに記憶する（S162）。

【０１０５】

一方、$T_{total_after}<T_{total_before}$の判定結果がNOであれば（S157のNO）、プロセッサは、並列度を低下したユーザ回路UC_MAX以外のユーザ回路に並列度未調整のDI処理回路が含まれているか否かを判定する（S158）。含まれている場合（S158のYES）、CI処理回路の代わりにDI処理回路の並列度を上げることで、工程S157の判定結果がYESになる可能性がある。そこで、プロセッサは、並列度を増加する第1のユーザ回路のDI処理回路とCI処理回路の組み合わせを変更し（S160）、再度工程S155,S156,S157を実行する。

【０１０６】

例えば、第1のユーザ回路について、あるCI処理回路の並列度を増加する代わりに、あるDI処理回路の並列度を増加したほうが、調整後の実行時間T_{total_after}を調整後の実行時間T_{total_before}より短くできる可能性がある。例えば、並列度を増やした第1のユーザ回路のうち、回路規模が大きなCI処理回路であって、並列度増加による実行時間の短縮が少ないCI処理回路について、並列度の増加を中止し、利用可能な回路リソースを確保し、そのCI処理回路代わりにDI処理回路を選択して並列度を増加することが望ましい。

【０１０７】

そして、工程S157の判定がYESになれば、プロセッサは、工程S161,S162を実行し、並列度の調整処理を実行する。

【０１０８】

一方、第1のユーザ回路のDI処理回路とCI処理回路の組み合わせの変更をK回行っても工程S157の判定がNOになる場合は（S159のNO）、所定のユーザ回路UC_MAXの並列度の低下と第1のユーザ回路の並列度の増加の調整を行わない（S15D）。前述の工程S158の判定がNOの場合も、工程S157の判定結果が逆転する可能性は小さいので、プロセッサは、ユーザ回路の並列度の調整を行わない（S15D）。

【０１０９】

図１４のフローチャートによれば、プロセッサは、バスボトルネックを解消するための所定のユーザ回路UC_MAXの並列度の低下と、他のユーザ回路（第1のユーザ回路）の並列度の増加の調整後の状態での全実行時間の合計が、調整前の状態での全実行時間の合計より短くなる第1のユーザ回路の組み合わせを探索する。これにより、バスボトルネックの解消と、FPAGの回路リソースの有効利用の両方を達成することができる。

【０１１０】

［ユーザ回路UC_MAXの並列度を増加する制御］

図１２に戻り、プロセッサは、一定時間待機中（S10のNO）、ユーザ回路のジョブ実行完了通知を受信すると（S16のYES）、式１、式２を満たす範囲で、ユーザ回路UC_MAXの並列度を増加する制御を行う（S17）。一定時間待機中にジョブ実行完了通知を受信しない場合、プロセッサは、ユーザ回路の並列度調整処理S8を終了する。

【０１１１】
図１５は、工程S17の処理を示すフローチャート図である。ユーザ回路のジョブ実行完了通知を受信すると(図１２のS16のYES)、プロセッサは、並列度低下リストにユーザ回路UC_MAXが存在するか判定する（S171）。存在する場合（S171のYES）、プロセッサは、式1、式２を満たす範囲で、ユーザ回路UC_MAXの最大の新並列度PXを算出する（S172）。式１，式２は、図１３の式1、式２と同じである。但し、ここでは、直前にあるユーザ回路がジョブ実行を完了して開放されるので、開放されたユーザ回路は式１，式２から除かれる。また、並列度増加対象は所定のユーザ回路UC_MAXである。
【０１１２】
例えば、ユーザ２の回路UC_2が並列度低下リストに格納されていて、ユーザ１とユーザ３の回路UC_1，UC_3のジョブ実行が完了したとすると、式１、式２は次の通りになる。
(BD_M2/P2)*PX2 + BD_M4 < BD_L　　式１
A2*PX2 + A4*P4 ≦ A_L　　式２
【０１１３】
プロセッサは、上記の式を満たす範囲で、最大の新並列度PX2を算出する。これにより、ユーザ回路UC_MAX（UC_2）は、他のユーザ回路の実行完了時に優先的に並列度を増加する制御を受けることができる。
【０１１４】
そして、プロセッサは、ユーザ回路UC_MAXの新並列度PXでの論理回路のコンフィグレーションをFPGAに要求し、回路リコンフィグレーション完了通知を受信するとジョブの実行再開を通知する（S173）。また、プロセッサは、並列度を増加したユーザ回路UC_MAXを並列度低下リストから削除する（S173）。
【０１１５】
図１２に戻り、プロセッサによるユーザ回路の並列度調整制御をまとめると次のとおりである。プロセッサは、通常は一定時間ごとに測定実行時間ET_Mと測定使用帯域BD_MをFPGA内のユーザ回路の測定回路から取得する（S11）。そして、全ユーザ回路の測定使用帯域の合計と、FPGAバスの帯域上限値との差分が、並列度増加のために必要な帯域分より大きい場合（S12のYES）、プロセッサは、式1，式２を満たす範囲で、あるユーザ回路の並列度を増加する（S13A）。また、あるユーザ回路がジョブの実行を完了した場合（S16のYES）、並列度を低下したユーザ回路UC_MAXがなければ（S171のNO）、次の測定サイクルで取得した測定使用帯域BD_Mに基づいて工程S12の判定がYESになり、プロセッサは、再度、式1、式２を満たす範囲で所定のユーザ回路を優先して並列度を増加する（S13A）。
【０１１６】
一方、測定使用帯域の合計値がFPGAバスの帯域上限値に達している場合（S14のYES）、プロセッサは、FPGAバス帯域のボトルネックの原因と疑われる所定のユーザ回路UC_MAXを選択し、その並列度をバスボトルネックが解消するように低下する（S15）。この並列度を下げるユーザ回路UC_MAXには、例えば使用帯域が大きいDI処理回路が選択される。さらに、プロセッサは、残りのユーザ回路について、ユーザ回路UC_MAX以外の他のユーザ回路を式１、式２を満たす範囲で並列度を増加する（S15）。この並列度を上げるユーザ回路は、例えば、使用帯域が小さいCI処理回路が選択される。
【０１１７】
そして、並列度調整前の状態での全ユーザ回路の合計予測実行時間より、並列度調整後の状態での全ユーザ回路の合計予測実行時間が短い場合に、プロセッサは、並列度の変更を実行し（S15C）、短くない場合に、プロセッサは、並列度の変更を実行しない（S15D）。
【０１１８】
さらに、あるユーザ回路のジョブの実行が完了したら（S16のYES）、プロセッサは、並列度を低下させたユーザ回路UC_MAXの並列度を、式１、式２を満たす範囲で最大の並列度に増加する（S17）。これにより、プロセッサは、バス帯域のボトルネックの原因と見なされたユーザ回路UC_MAXの並列度を一時的に低下するが、他のユーザ回路の並列度を増加

した結果それらの合計実行時間が短くなる。そして、他のユーザ回路のジョブの実行が完了すると、一時的に並列度を低下させたユーザ回路UC_MAXの並列度を再度増加させる。その結果、バスボトルネック発生時の並列度の調整により、バスボトルネックが解消され、さらに全ユーザ回路の合計実行時間が短くなる可能性がある。

【０１１９】

［並列度調整の具体例］

図１６は、第1の具体例を示す図である。横軸が時間TIMEであり、縦方向に（１）バスボトルネックが発生しない場合の予測実行時間、（２）バスボトルネックが発生し並列度調整しない場合の予測実行時間、（３）バスボトルネックが発生し並列度調整した場合の予測実行時間をそれぞれ示す。第1の具体例は、FPAG内にユーザ回路UC-AとUC-Bがコンフィグレーションされ、ユーザ回路UC-AがDI処理回路、ユーザ回路UC-BがCI処理回路という、最も単純化した例である。

【０１２０】

図１６の（１）において、ユーザ回路UC-Aは、DI処理回路であり、並列度Ｐ＝４、１回の実行時間が８で、実行回数５でジョブが完了する。一方、ユーザ回路UC-Bは、CI処理回路であり、並列度Ｐ＝２、１回の実行時間が２５（T_{LD}＝１、イニシエーション・インターバルΔ＝２、T_{COMP}＝２１、T_{ST}＝１）で、実行回数２でジョブが完了する。

【０１２１】

（２）バスボトルネック発生、並列度調整しない場合、時刻ｔ１でバスボトルネックが発生し、ユーザ回路UC-AとUC-Bでメモリアクセスに時間がかかっている。そして、バスボトルネックが発生しても両ユーザ回路の並列度を変更していない。その結果、ユーザ回路UC－Aは実行時間が６長くなり、ユーザ回路UC-Bは実行時間が３長くなっている。

【０１２２】

（３）バスボトルネック発生、並列度調整する場合、ユーザ回路UC-Aの並列度Pを４から２に減らした結果、１回の実行時間が８から１６に２倍になっている。そして、ユーザ回路UC-Bの並列度Pを２から４に２倍に増やしたが、CI処理回路のため、イニシエーション・インターバルΔが２から１に減少しただけであり、１回の実行時間は２５から２４にわずかに短くなっただけである。そして、ユーザ回路UC-Bの実行終了後に、ユーザ回路UC-Aの並列度Pを２から８に変更し、最後の実行時間が４と短くなった。

【０１２３】

そこで、（２）と（３）の場合の２つのユーザ回路の実行時間の合計を比較すると、ユーザ回路UC-Aが２２長くなり、ユーザ回路UC-Bが１短くなっているので、（２）の調整前の合計実行時間T_{total_before}より（３）の調整後の合計実行時間T_{total_after}のほうが長くなる。したがって、バスボトルネックが発生したとき、プロセッサは、ユーザ回路の並列度の調整を行わないという判定を行う。

【０１２４】

図１７は、第２の具体例を示す図である。横軸と、（１）（２）（３）は図１６と同様である。第２の具体例は、FPAG内にユーザ回路UC-AとUC-Bに加えて更にUC-Cがコンフィグレーションされ、ユーザ回路UC-A、UC-CがDI処理回路、ユーザ回路UC-BがCI処理回路である。

【０１２５】

図１７の（１）において、ユーザ回路UC-A、UC-Bは、図１６と同じである。そして、ユーザ回路UC-Cは、DI処理回路であり、並列度Ｐ＝３、１回の実行時間が１２で、実行回数４でジョブが完了する。

【０１２６】

（２）バスボトルネック発生、並列度調整を行わない場合、時刻ｔ１でバスボトルネックが発生し、ユーザ回路UC-A、UC-B、UC-Cでメモリアクセスに時間がかかっている。そして、バスボトルネックが発生しても両ユーザ回路の並列度を変更していない。その結果、ユーザ回路UC－Aは実行時間が６長くなり、ユーザ回路UC-Bは実行時間が３長くなり、ユーザ回路UC-Cは実行時間が６長くなっている。

【０１２７】
（３）バスボトルネック発生、並列度調整を行う場合、ユーザ回路UC-Aの並列度Pを４から２に減らした結果、１回の実行時間が８から１６に２倍になっている。そして、具体例１で説明したとおりユーザ回路UC-Bの並列度を上げても実行時間の短縮は小さいため、ユーザ回路UC-Bの並列度Pは２から増加していない。その代わりに、ユーザ回路UC-Cの並列度Pを３から４に変更している。その結果、ユーザ回路UC-Bはリコンフィグレーション時間がなくなり１回の実行時間は（１）と同じ２５であり、一方、ユーザ回路UC-Cは、実行時間が１２から９に短くなっている。この例では、ユーザ回路UC-Cの使用帯域が小さかったため、帯域上限未満で並列度を増やすことができた例である。
【０１２８】
そして、ユーザ回路UC-Cの実行終了後に、ユーザ回路UC-Aの並列度Pを２から８に変更し、最後の３回の実行時間がそれぞれ４と短くなった。
【０１２９】
そこで、（２）と（３）の場合の２つのユーザ回路の実行時間の合計を比較すると、ユーザ回路UC-Aが１４長くなり、一方、ユーザ回路UC-Bが３短くなり、ユーザ回路UC-Cが１２短くなっている。その結果、（２）の調整前の合計実行時間T_{total_before}より（３）の調整後の合計実行時間T_{total_after}のほうが短くなる。したがって、バスボトルネックが発生したとき、ユーザ回路の並列度の調整を行うという判定が行われる。
【０１３０】
第２の具体例では、バスボトルネックが発生したとき、DI処理回路のユーザ回路UC-Aの並列度を下げてバスボトルネックを解消し、余ったバスの帯域とFPAG内の回路リソースを利用して、CI処理回路ではなくDI処理回路のユーザ回路UC-Cの並列度を上げた例である。この場合は、調整後の合計実行時間のほうが短くなったため、並列度の調整を行うという判定になっている。
【０１３１】
以上のとおり、本実施の形態では、バスボトルネックが発生したときに、プロセッサが、バスボトルネック解消のために所定のユーザ回路の並列度を一時的に下げ、余裕ができた回路リソースを利用して別のユーザ回路の並列度を増加し、別のユーザ回路の実行完了後に、所定のユーザ回路の並列度を増加する。但し、上記のような並列度の調整を行うか否かを、調整前の状態での予測実行時間の合計と、調整後の状態での予測実行時間の合計とを比較して判定する。
【０１３２】
通常は、バスボトルネックが発生すると、使用帯域が大きいDI処理回路の並列度を下げて、使用帯域が小さいCI処理回路の並列度を上げることで、バスボトルネックの解消と実行時間の悪化を抑制する。
【０１３３】
しかし、CI処理回路は、並列度を下げても実行時間の短縮の度合いが小さく、並列度の調整を行っても実行時間の合計が短くならない場合がある。その場合は、並列度の調整を行わないという判定が行われる。
【０１３４】
但し、CI処理回路でも、並列度の増加による実行時間の短縮の度合いに幅があり、短縮時間が長いCI処理回路を選択して並列度を増加し、さらにバスボトルネックにならない別のDI処理回路を選択して並列度を増加して実行時間を短縮させることで、バスボトルネックの解消と実行時間の悪化の抑制を達成できる場合がある。
【０１３５】
上記の実施の形態では、バスボトルネックが発生したときに、一部のユーザ回路)並列度を低下し、他のユーザ回路の並列度を増加する並列度の調整を行うか否かを、調整前の状態と調整後の状態でそれぞれの実行時間の合計を比較して判定した。しかし、バスボトルネックの発生時だけでなく、例えばFPGAの消費電力の上限値に達した場合（消費電力のボトルネックの発生）でも、消費電力の大きいユーザ回路の並列度を低下し、消費電力の

小さい他のユーザ回路の並列度を増加する並列度の調整を行うか否かを、調整前と調整後の実行時間の合計を比較して判定して行っても良い。

【０１３６】

以上の実施の形態をまとめると，次の付記のとおりである。

【０１３７】

（付記１）

プログラムを実行するプロセッサと、

前記プロセッサからのコンフィグレーション要求に応じて、前記コンフィグレーション要求が要求する論理回路をコンフィグレーションするリコンフィグレーション領域を有するプログラマブルロジック回路装置（以下ＰＬＤ）を有し、

前記プロセッサは、

前記リコンフィグレーション領域内にコンフィグレーションされ動作中の複数の論理回路のうち、第１の論理回路の並列度を下げて第２の論理回路の並列度を上げる並列度調整を行った場合の前記複数の論理回路の第１の実行時間と、前記並列度調整を行わない場合の前記複数の論理回路の第２の実行時間とを比較し、

前記第１の実行時間が前記第２の実行時間より短い場合、前記ＰＬＤに前記並列度調整の要求を行い、短くない場合、前記ＰＬＤに前記並列度調整の要求を行わない、情報処理装置。

【０１３８】

（付記２）

前記プロセッサは、

前記リコンフィグレーション領域内にコンフィグレーションされ動作中の複数の論理回路のデータ転送量の測定値を取得し、

前記データ転送量の合計が前記ＰＬＤのバスのデータ転送量の上限値に達した場合、前記比較を実行する、付記１に記載の情報処理装置。

【０１３９】

（付記３）

前記比較では、前記複数の論理回路の前記第１の実行時間の合計と、前記第２の実行時間の合計とを比較し、

前記第１の実行時間の合計値が前記第２の実行時間の合計値より短い場合、前記ＰＬＤに前記並列度調整の要求を行い、短くない場合、前記ＰＬＤに前記並列度調整の要求を行わない、付記１に記載の情報処理装置。

【０１４０】

（付記４）

前記プロセッサは、さらに、

前記複数の論理回路の前記第１の実行時間と、前記第２の実行時間とを計算する、付記１に記載の情報処理装置。

【０１４１】

（付記５）

前記プロセッサは、

前記複数の論理回路の前記第１の実行時間として、第１の論理回路の並列度を下げて第２の論理回路の並列度を上げる並列度調整を行った後、前記第２の論理回路の実行が完了後に前記第１の論理回路の並列度を上げる場合に予測される前記複数の論理回路の実行時間を計算する、付記４に記載の情報処理装置。

【０１４２】

（付記６）

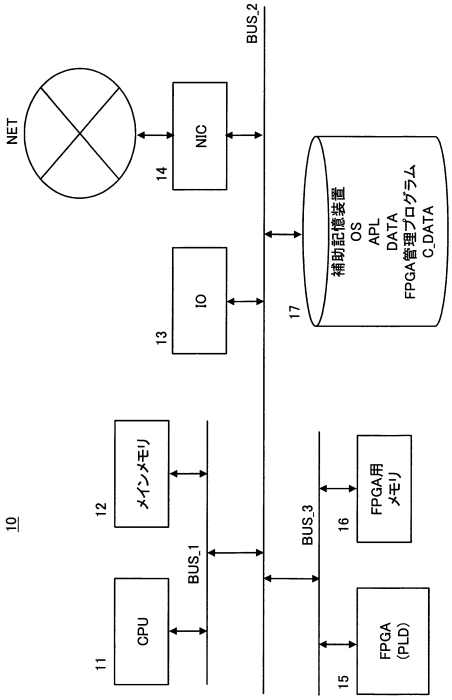
前記複数の論理回路は、データ処理中にメモリアクセスが発生するデータ・インテンシブ処理回路と、データ処理の最初と最後にメモリアクセスが発生するコンピュテーション・インテンシブ処理回路のいずれか一方または両方を含み、

前記プロセッサは、

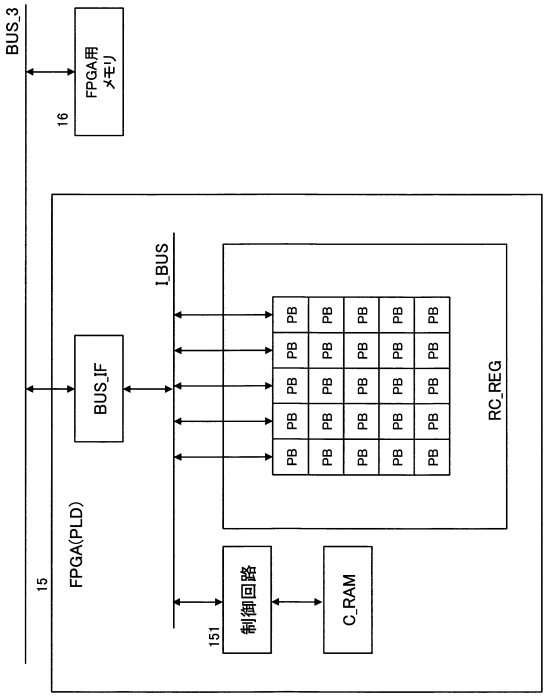
前記データ・インテンシブ処理回路の並列度をＮ倍にした場合、前記実行時間を１／Ｎ倍になるよう前記第１の実行時間を算出し、
前記コンピュテーション・インテンション処理回路の並列度をＮ倍にした場合、前記実行時間を、前記コンピュテーション・インテンション処理回路のパイプライン処理におけるイニシエーション・インターバル時間が１／Ｎ倍になるよう前記第１の実行時間を算出する、付記４に記載の情報処理装置。
【０１４３】
（付記７）
前記複数の論理回路は、データ処理中にメモリアクセスが発生するデータ・インテンシブ処理回路と、データ処理の最初と最後にメモリアクセスが発生するコンピュテーション・インテンシブ処理回路の両方を含み、
前記プロセッサは、
前記第１の論理回路を前記データ・インテンシブ処理回路から選択し、前記第２の論理回路を前記コンピュテーション・インテンシブ処理回路から選択する、付記１に記載の情報処理装置。
【０１４４】
（付記８）
前記プロセッサは、
前記第１の実行時間が前記第２の実行時間より短くない場合、前記第２の論理回路を変更して、前記比較を再度行う、付記７に記載の情報処理装置。
【０１４５】
（付記９）
前記プロセッサは、
前記第２の論理回路の変更を、前記第２の論理回路のうち前記並列度を増やした場合の実行時間の短縮の程度が少ないコンピュテーション・インテンシブ処理回路に代えて、前記データ・インテンシブ処理回路を選択する、付記８に記載の情報処理装置。
【０１４６】
（付記１０）
プロセッサからのコンフィグレーション要求に応じて、前記コンフィグレーション要求が要求する論理回路をコンフィグレーションするリコンフィグレーション領域を有するプログラマブルロジック回路装置（以下ＰＬＤ）管理処理をプロセッサに実行させるＰＬＤ管理プログラムであって、
前記管理処理は、
前記リコンフィグレーション領域内にコンフィグレーションされ動作中の複数の論理回路のうち、第１の論理回路の並列度を下げて第２の論理回路の並列度を上げる並列度調整を行った場合の前記複数の論理回路の第１の実行時間と、前記並列度調整を行わない場合の前記複数の論理回路の第２の実行時間とを比較し、
前記第１の実行時間が前記第２の実行時間より短い場合、前記ＰＬＤに前記並列度調整の要求を行い、短くない場合、前記ＰＬＤに前記並列度調整の要求を行わない、処理を有するＰＬＤ管理プログラム。
【０１４７】
（付記１１）
プログラムを実行するプロセッサと、
前記プロセッサからのコンフィグレーション要求に応じて、前記コンフィグレーション要求が要求する論理回路をコンフィグレーションするリコンフィグレーション領域を有するプログラマブルロジック回路装置（以下ＰＬＤ）を有する情報処理装置の前記ＰＬＤ管理方法であって、
前記リコンフィグレーション領域内にコンフィグレーションされ動作中の複数の論理回路のうち、第１の論理回路の並列度を下げて第２の論理回路の並列度を上げる並列度調整を行った場合の前記複数の論理回路の第１の実行時間と、前記並列度調整を行わない場合

の前記複数の論理回路の第２の実行時間とを比較し、
　前記第１の実行時間が前記第２の実行時間より短い場合、前記ＰＬＤに前記並列度調整の要求を行い、短くない場合、前記ＰＬＤに前記並列度調整の要求を行わない、ＰＬＤ管理方法。
【符号の説明】
【０１４８】
１０：情報処理装置
１１：CPU、プロセッサ
１２：メインメモリ
１５：FPGA、PLD
１６：補助記憶装置
１７：FPGA用のデータメモリ
BUS_1：CPUバス
BUS_2：PCIバス
BUS_3：FPGAバス
I_BUS：FPGA内部バス
RC_REG：リコンフィグレーション領域
OC:FPGAの運用回路
PB：部分リコンフィグレーションブロック
UC_A，UC_B：ユーザ回路
１５１：C_DATA書き込み制御回路
C_RAM：コンフィグレーションデータメモリ
P：並列度
ET_E：予測実行時間
BD_E：予測帯域
ET_M：測定実行時間
BD_M：測定帯域、使用帯域
A1、A2：ユーザ回路面積
BD_L：上限帯域
A_L：総回路面積
T_{total_after}：第１の実行時間、第１の合計実行時間、並列度調整後の合計実行時間
T_{total_before}：第２の実行時間、第２の合計実行時間、並列度調整前の合計実行時間
CI：コンピュテーション・インテンシブ処理回路
DI：データ・インテンシブ処理回路

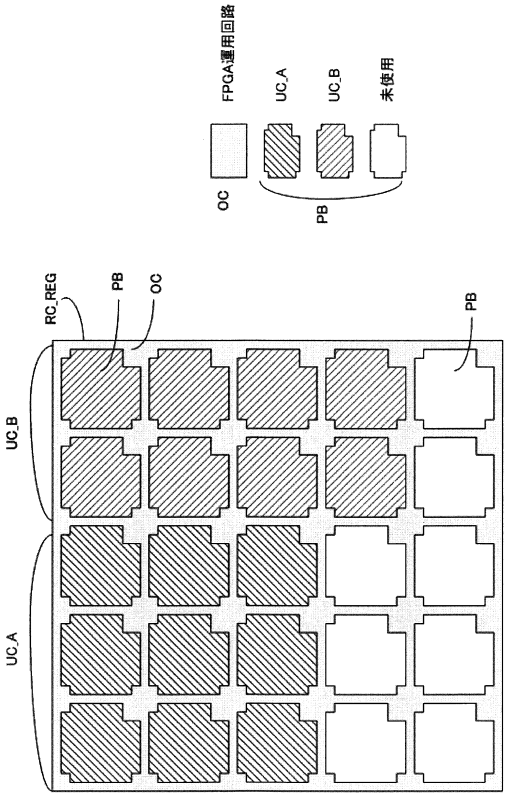
【図１】


10
NET
NIC
14
BUS_2
IO
13
補助記憶装置
OS
APL
DATA
FPGA管理プログラム
C_DATA
17
メインメモリ
12
BUS_1
BUS_3
FPGA用
メモリ
16
CPU
11
FPGA
(PLD)
15

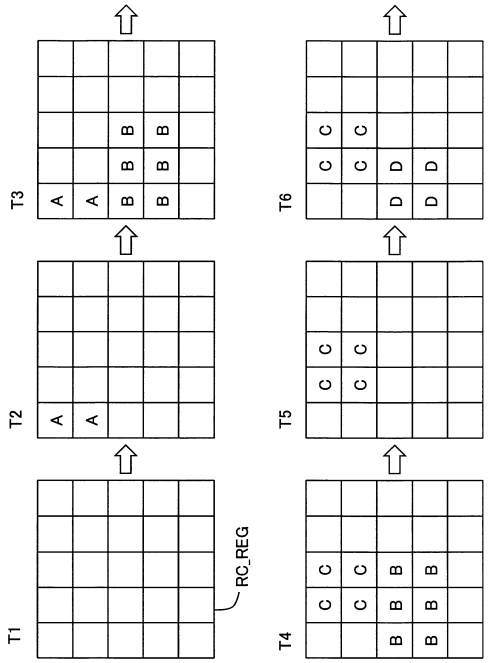
【図２】


BUS_3
FPGA用
メモリ
16
BUS_IF
I_BUS
RC_REG
PB
FPGA(PLD)
15
制御回路
151
C_RAM

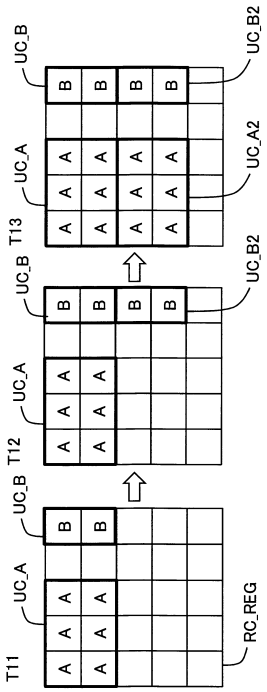
【図３】


FPGA運用回路
UC_A
UC_B
未使用
OC
PB
RC_REG
PB
OC
UC_B
UC_A
PB

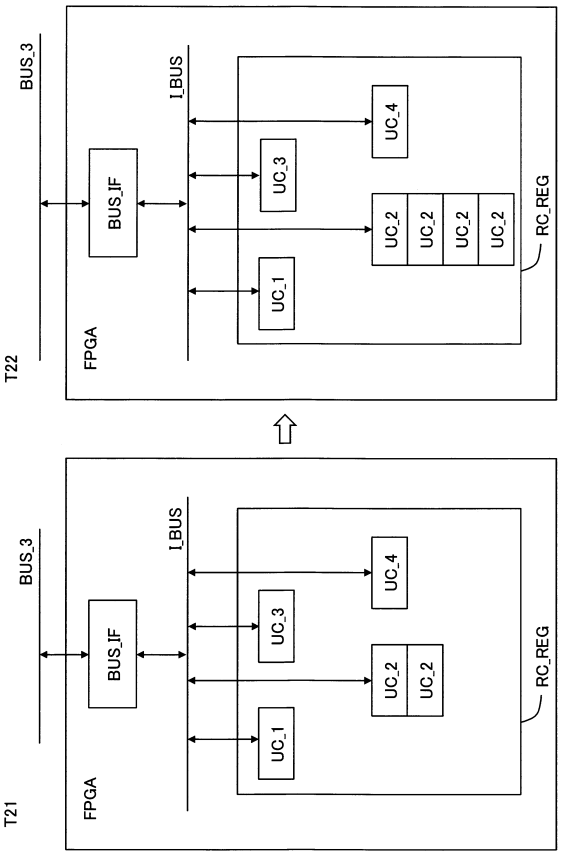
【図４】


T1
T2
T3
T4
T5
T6
RC_REG

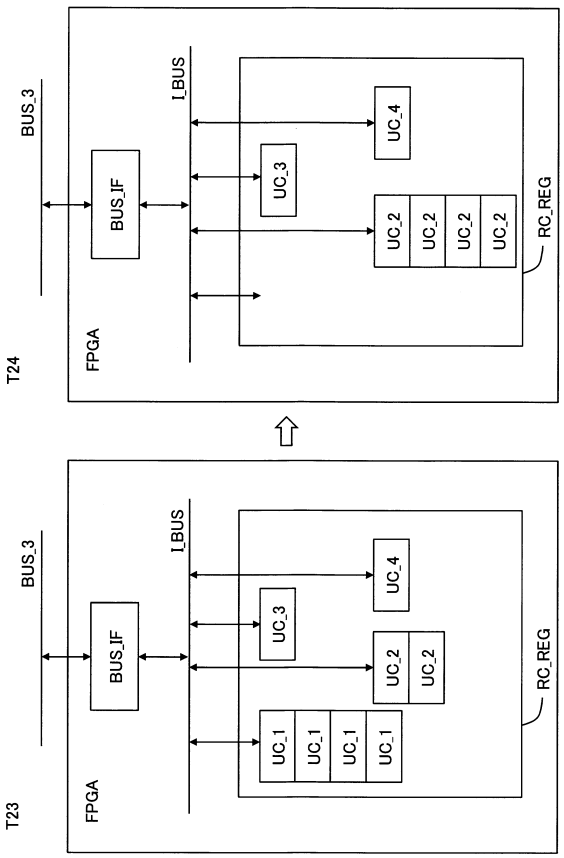
【図５】



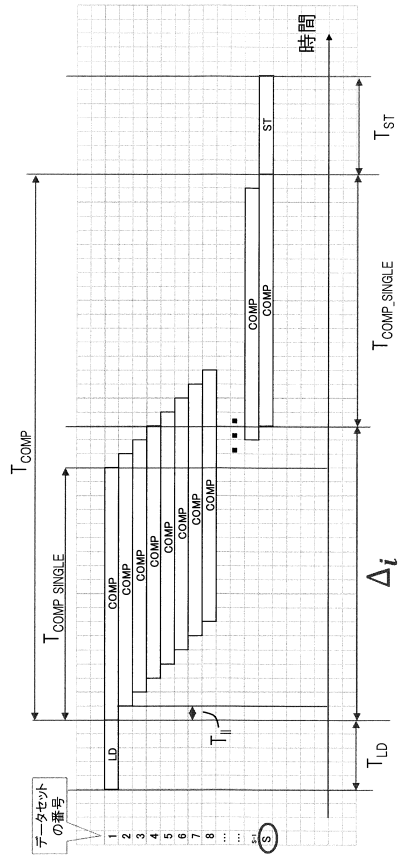
【図６】



【図７】



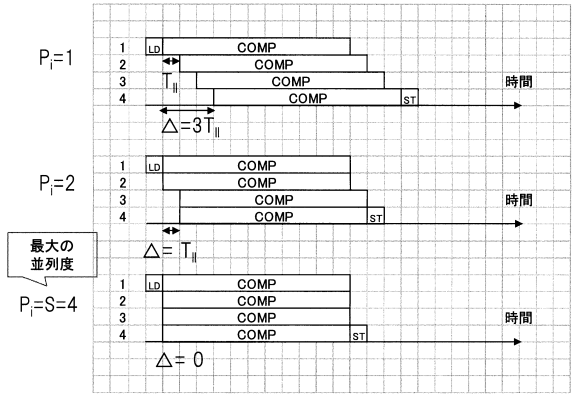
【図８】



$$T_i = T_{LD} + \Delta_i + T_{COMP\_SINGLE} + T_{ST}$$

【図９】

$$\Delta_i = T_{II} \times \left(\frac{S}{P_i} - 1\right)$$

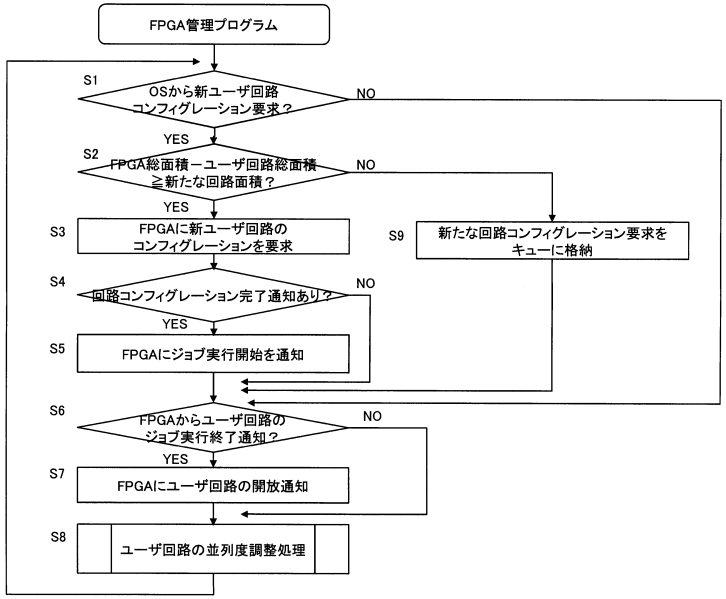


$$\Delta = T_{II} \times \left(\frac{4}{1} - 1\right) = 3 \cdot T_{II}$$

$$\Delta = T_{II} \times \left(\frac{4}{2} - 1\right) = T_{II}$$

$$\Delta = T_{II} \times \left(\frac{4}{4} - 1\right) = 0$$

【図１０】

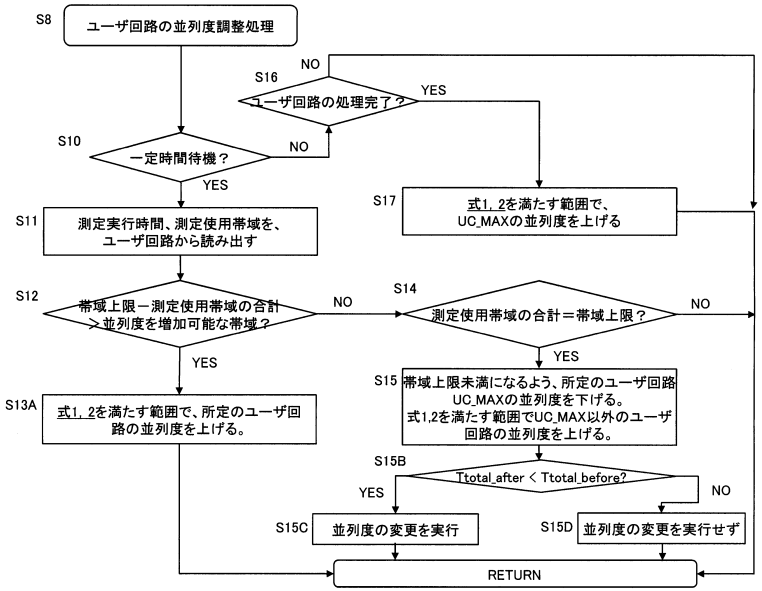


【図１１】

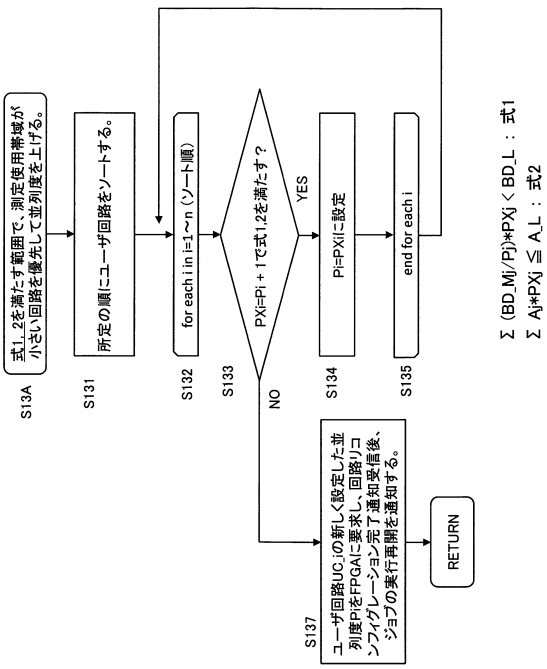
	UC_1	UC_2	UC_3	UC_4
並列度　P	P1	P2	P3	P4
予測コンフィグレーション時間　CT_E	CT_E1	CT_E2	CT_E3	CT_E4
予測実行時間　ET_E	ET_E1	ET_E2	ET_E3	ET_E4
予測使用帯域　BD_E	BD_E1	BD_E2	BD_E3	BD_E4
測定実行時間　ET_M	ET_M1	ET_M2	ET_M3	ET_M4
測定使用帯域　BD_M	BD_M1	BD_M2	BD_M3	BD_M4
回路タイプ（処理パターン）	CI	DI	DI	CI
回路の実行回数	N_1	N_2	N_3	N_4
1回の実行で処理されるデータセット数	S_1	S_2	S_3	S_4
入力データの読み取り時間	$T_{LD\_}1$	–	–	$T_{LD\_}4$
出力データの書き込み時間	$T_{ST\_}1$	–	–	$T_{ST\_}4$
1個のデータセットの処理時間	$T_{COMP\_SINGLE\_}1$	–	–	$T_{COMP\_SINGLE\_}4$
Initiation intervalの時間	$T_{II\_}1$	–	–	$T_{II\_}4$

FPGAバス帯域上限値　BD_L

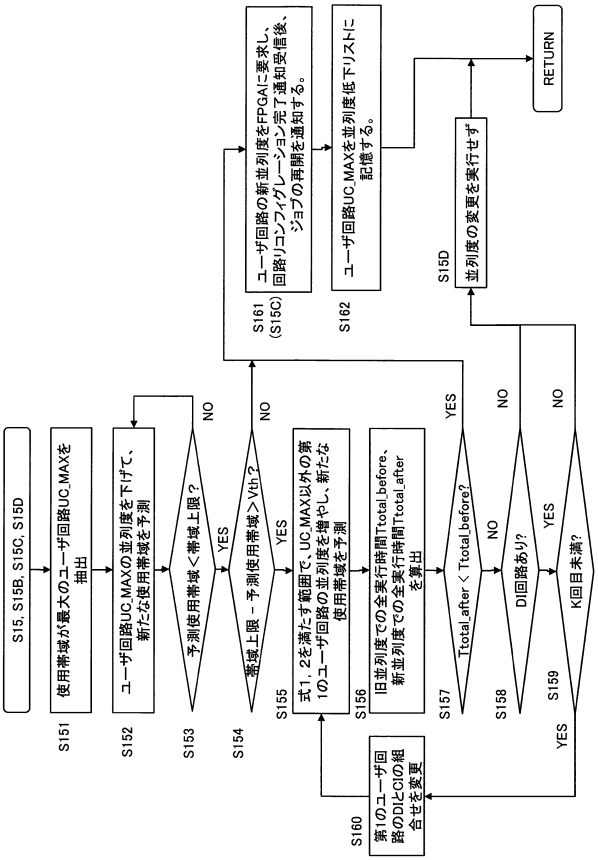
【図１２】



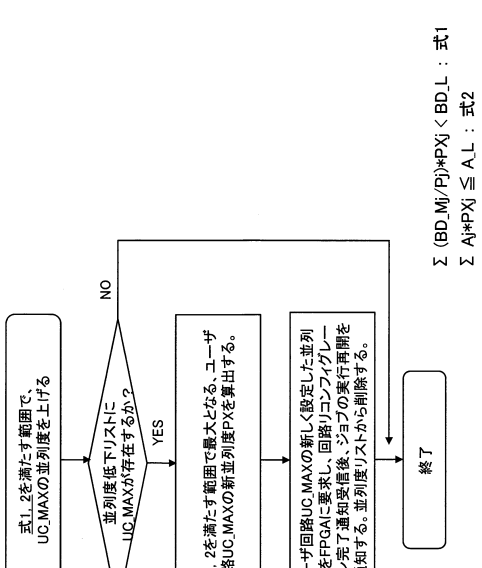
【図１３】



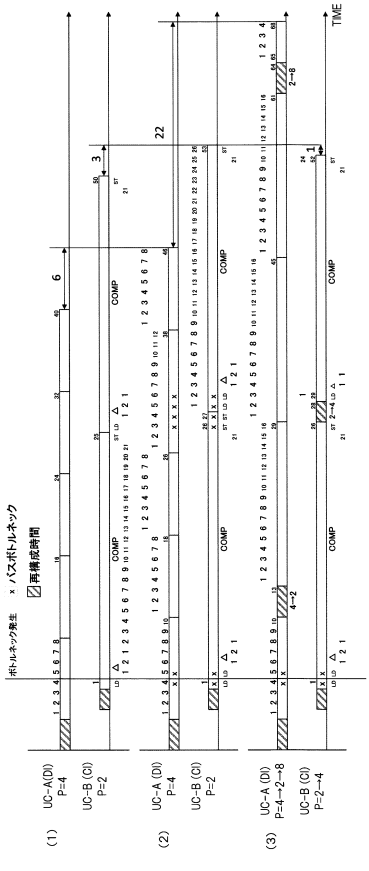
【図１４】



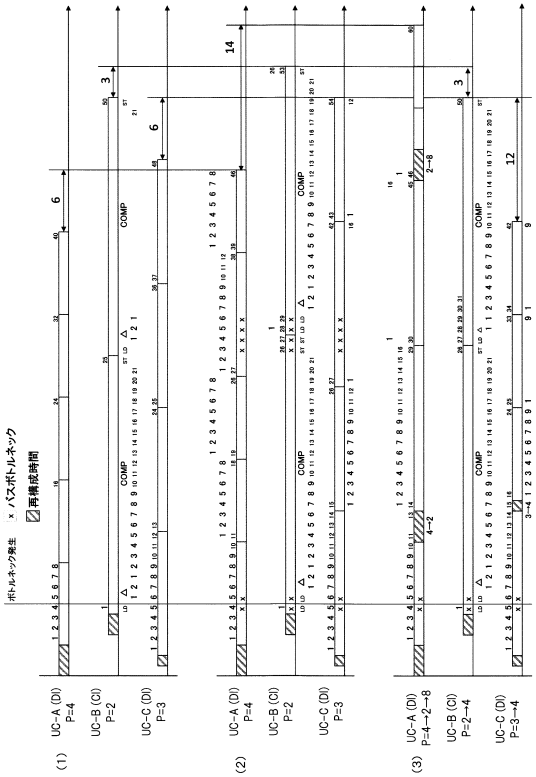
【図１５】



【図１６】



【図１７】


ボトルネック発生
バスボトルネック
非構成時間
COMP
(1)
UC-A (DI)
P=4
UC-B (CI)
P=2
UC-C (DI)
P=3
(2)
UC-A (DI)
P=4
UC-B (CI)
P=2
UC-C (DI)
P=3
(3)
UC-A (DI)
P=4→2→8
UC-B (CI)
P=2→4
UC-C (DI)
P=3→4

フロントページの続き

審査官　漆原　孝治

(56)参考文献　特開２００７－１７９３５８（ＪＰ，Ａ）
特開２０１１－２０３９２０（ＪＰ，Ａ）
特開２０１６－１１０４９９（ＪＰ，Ａ）
国際公開第２０１７／０２９７４３（WO，Ａ１）
特開２０１４－２３５７４６（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　　９／５０