(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6703533号
(P6703533)
(45)発行日　令和2年6月3日(2020.6.3)　(24)登録日　令和2年5月12日(2020.5.12)

(51)Int.Cl.　　F I
G06F 8/40 (2018.01)　　G06F 8/40
G06F 9/455 (2006.01)　　G06F 9/455 100

請求項の数 15　(全 39 頁)

(21)出願番号　特願2017-525600(P2017-525600)
(86)(22)出願日　平成27年11月10日(2015.11.10)
(65)公表番号　特表2018-507449(P2018-507449A)
(43)公表日　平成30年3月15日(2018.3.15)
(86)国際出願番号　PCT/US2015/060025
(87)国際公開番号　WO2016/077393
(87)国際公開日　平成28年5月19日(2016.5.19)
審査請求日　平成30年9月5日(2018.9.5)
(31)優先権主張番号　14/539,985
(32)優先日　平成26年11月12日(2014.11.12)
(33)優先権主張国・地域又は機関
米国(US)
(31)優先権主張番号　14/539,975
(32)優先日　平成26年11月12日(2014.11.12)
(33)優先権主張国・地域又は機関
米国(US)

(73)特許権者　591025439
ザイリンクス　インコーポレイテッド
ＸＩＬＩＮＸ　ＩＮＣＯＲＰＯＲＡＴＥＤ
アメリカ合衆国　カリフォルニア州　９５１２４－３４００　サン　ホセ　ロジック　ドライブ　２１００
(74)代理人　110001195
特許業務法人深見特許事務所
(72)発明者　スタイルズ，ヘンリー・イー
アメリカ合衆国、９５１２４　カリフォルニア州、サン・ノゼ、ロジック・ドライブ、２１００

最終頁に続く

(54)【発明の名称】プログラム可能集積回路を対象としたヘテロジニアスマルチプロセッサプログラムコンパイル

(57)【特許請求の範囲】
【請求項１】
方法であって、
プロセッサを使用して、第１のカーネルの高レベルプログラミング言語記述から、前記第１のカーネルのレジスタ転送レベル記述を生成するステップであって、前記第１のカーネルは、ヘテロジニアスマルチプロセッサ設計の一部分であるステップと、
技術マッピング、配置、およびルーティングによって、前記ヘテロジニアスマルチプロセッサ設計のホストに対するインターフェース、および、プログラム可能集積回路内の動的に再構成可能なカーネル領域に対するインターフェースを提供する前記プログラム可能集積回路内の静的領域を提供するベースプラットフォーム回路設計と、前記第１のカーネルの前記レジスタ転送レベル記述を統合するステップと、
前記第１のカーネルの前記レジスタ転送レベル記述から、前記プロセッサを使用して、前記第１のカーネルのハードウェア実施態様を指定する第１の構成ビットストリーム、および、前記第１のカーネルのレジスタマップを含む前記構成ビットストリームのサポートデータを生成するステップであって、前記第１のカーネルの前記ハードウェア実施態様は、前記カーネル領域内で実装されるように適合されているステップと、
複数のファイルを含むバイナリコンテナを生成するステップであって、前記複数のファイルは、前記第１の構成ビットストリームを含む第１のファイルと、前記サポートデータを含む第２のファイルとを含み、前記第２のファイルの前記レジスタマップは、前記プログラム可能集積回路内で実装される前記第１のカーネルと通信している前記ホストによっ

て使用可能であるステップとを含む、方法。
【請求項２】
前記カーネル領域は、ランタイム中に、前記静的領域を損傷を受けないままにしながら、異なるカーネルのハードウェア実施態様を実装するように、動的に再構成可能である、請求項１に記載の方法。
【請求項３】
前記ヘテロジニアスマルチプロセッサ設計のランタイム中に前記プログラム可能集積回路内で前記第１のカーネルの前記ハードウェア実施態様のインスタンスを作成する、前記第１のカーネルの前記構成ビットストリームをロードするステップをさらに含み、前記ホストは前記バイナリコンテナからの前記構成ビットストリームを前記プログラム可能集積回路へと提供する、請求項１に記載の方法。
【請求項４】
前記静的領域は、
前記カーネル領域に結合されており、前記プログラム可能集積回路の前記カーネル領域にクロック信号を与えるように構成されている第１の相互接続回路ブロックと、
前記第１の相互接続回路ブロックに結合されているバスダイレクトメモリアクセスコントローラと、
前記バスダイレクトメモリアクセスコントローラに結合されているバスエンドポイントとを含む、請求項２に記載の方法。
【請求項５】
前記静的領域は、前記カーネル領域をメモリコントローラに結合する第２の相互接続回路ブロックを含む、請求項４に記載の方法。
【請求項６】
前記第１の構成ビットストリームを生成するステップは、
前記第１のカーネルの回路の前記ハードウェア実施態様を指定する部分構成ビットストリームとして前記第１の構成ビットストリームを生成するステップを含む、請求項１に記載の方法。
【請求項７】
前記第１の構成ビットストリームを生成するステップは、
前記第１のカーネルの前記ハードウェア実施態様および前記ベースプラットフォーム回路設計に対応したベースプラットフォーム回路を指定する全構成ビットストリームとして前記第１の構成ビットストリームを生成するステップを含む、請求項１に記載の方法。
【請求項８】
システムであって、
実行可能動作を開始するようにプログラムされているプロセッサを備え、前記実行可能動作は、
第１のカーネルの高レベルプログラミング言語記述から、前記第１のカーネルのレジスタ転送レベル記述を生成するステップであって、前記第１のカーネルは、ヘテロジニアスマルチプロセッサ設計の一部分であるステップと、
技術マッピング、配置、およびルーティングによって、前記ヘテロジニアスマルチプロセッサ設計のホストに対するインターフェース、および、プログラム可能集積回路内の動的に再構成可能なカーネル領域に対するインターフェースを提供する前記プログラム可能集積回路内の静的領域を提供するベースプラットフォーム回路設計と、前記第１のカーネルの前記レジスタ転送レベル記述を統合するステップと、
前記第１のカーネルの前記レジスタ転送レベル記述から、前記第１のカーネルのハードウェア実施態様を指定する第１の構成ビットストリーム、および、前記第１のカーネルのレジスタマップを含む前記構成ビットストリームのサポートデータを生成するステップであって、前記第１のカーネルの前記ハードウェア実施態様は、前記カーネル領域内で実装されるように適合されているステップと、
複数のファイルを含むバイナリコンテナを生成するステップであって、前記複数のファ

イルは、前記第１の構成ビットストリームを含む第１のファイルと、前記サポートデータを含む第２のファイルとを含み、前記第２のファイルの前記レジスタマップは、前記プログラム可能集積回路内で実装される前記第１のカーネルと通信している前記ホストによって使用可能であるステップとを含む、システム。

【請求項９】

前記カーネル領域は、ランタイム中に、前記静的領域を損傷を受けないままにしながら、異なるカーネルのハードウェア実施態様を実装するように、動的に再構成可能である、請求項８に記載のシステム。

【請求項１０】

前記実行可能動作は、

前記ヘテロジニアスマルチプロセッサ設計のランタイム中に前記プログラム可能集積回路内で前記第１のカーネルの前記ハードウェア実施態様のインスタンスを作成する、前記第１のカーネルの前記構成ビットストリームをロードするステップをさらに含み、前記ホストは前記バイナリコンテナからの前記構成ビットストリームを前記プログラム可能集積回路へと提供する、請求項８に記載のシステム。

【請求項１１】

前記実行可能動作は、

前記ヘテロジニアスマルチプロセッサ設計のランタイム中に前記プログラム可能集積回路内で前記第１のカーネルの前記ハードウェア実施態様の複数のインスタンスを作成する、前記第１のカーネルの前記構成ビットストリームをロードするステップをさらに含み、前記ホストは前記バイナリコンテナからの前記構成ビットストリームを前記プログラム可能集積回路へと提供する、請求項９に記載のシステム。

【請求項１２】

前記静的領域は、

前記カーネル領域に結合されており、前記プログラム可能集積回路の前記カーネル領域にクロック信号を与えるように構成されている第１の相互接続回路ブロックと、

前記第１の相互接続回路ブロックに結合されているバスダイレクトメモリアクセスコントローラと、

前記バスダイレクトメモリアクセスコントローラに結合されているバスエンドポイントとを含む、請求項９に記載のシステム。

【請求項１３】

前記静的領域は、前記カーネル領域をメモリコントローラに結合する第２の相互接続回路ブロックを含む、請求項１２に記載のシステム。

【請求項１４】

前記第１の構成ビットストリームを生成するステップは、

前記第１のカーネルの回路の前記ハードウェア実施態様を指定する部分構成ビットストリームとして前記第１の構成ビットストリームを生成するステップを含む、請求項８に記載のシステム。

【請求項１５】

前記第１の構成ビットストリームを生成するステップは、

前記第１のカーネルの前記ハードウェア実施態様および前記ベースプラットフォーム回路設計に対応したベースプラットフォーム回路を指定する全構成ビットストリームとして前記第１の構成ビットストリームを生成するステップを含む、請求項８に記載のシステム。

【発明の詳細な説明】

【技術分野】

【０００１】

発明の分野

本開示は、集積回路（ＩＣ）に関し、より詳細には、プログラム可能ＩＣをヘテロジニアスマルチプロセッサ設計に組み込むことに関する。

【背景技術】
【０００２】
背景
ヘテロジニアスマルチプロセッサフレームワークは、クロスプラットフォームであり、最新のプロセッサ、サーバ、手持ち式／内蔵デバイスなどの並列プログラミングをサポートする基準をもたらす。「ＯｐｅｎＣＬ」と称されるＯｐｅｎ Ｃｏｍｐｕｔｉｎｇ Ｌａｎｇｕａｇｅは、ヘテロジニアスコンピューティングプラットフォームで実行することができるプログラムを書くためのヘテロジニアスマルチプロセッサフレームワークの一例である。ヘテロジニアスコンピューティングプラットフォームは、中央処理装置（ＣＰＵ）、グラフィックス処理ユニット（ＧＰＵ）、デジタル信号プロセッサ（ＤＳＰ）などを含み得る。
【０００３】
ヘテロジニアスマルチプロセッサプログラム、たとえば、ＯｐｅｎＣＬプログラムは、ホストシステム上で実行する一部分と、デバイス上で実行する１つまたは複数の他の部分とを含む。一般的に、ホストシステムはＣＰＵを含み、一方で、デバイスはＧＰＵ、ＤＳＰなどとして実装され得る。カーネルとして参照される場合がある、デバイス上で実行する部分は、ＯｐｅｎＣＬ、ＯｐｅｎＣＬ Ｃ、または、ヘテロジニアスマルチプロセッサフレームワークもしくはＯｐｅｎＣＬに対して適合されている別の高レベルプログラミング言語でコード化され得る。ホスト上で実行する部分は、たとえば、ＣまたはＣ＋＋でプログラムされ得、様々なデバイスでヘテロジニアスマルチプロセッサ環境を制御する。
【０００４】
上述した環境は本質的にヘテロジニアスであるが、各特定のデバイスは、ＤＳＰであろうとＧＰＵであろうと、静的なアーキテクチャを有する。比較すると、フィールドプログラマブルゲートアレイ（ＦＰＧＡ）のようなプログラム可能ＩＣは、ハードウェアアクセラレーションの目的に使用することができる、極度に柔軟なハードウェアアーキテクチャを有する。しかしながら、プログラム可能ＩＣをデバイスとして利用するためには、プログラム可能ＩＣ内に実装される回路が、ホストと対話し、ヘテロジニアスマルチプロセッサ環境のコンテキスト内で動作することが可能でなければならない。
【発明の概要】
【課題を解決するための手段】
【０００５】
発明の概要
方法は、プロセッサを使用して、ヘテロジニアスマルチプロセッサ設計の第１のカーネルのレジスタ転送レベル（ＲＴＬ）記述を生成するステップと、ヘテロジニアスマルチプロセッサ設計のホストに対するインターフェースを提供するプログラム可能集積回路（ＩＣ）内の静的領域を提供するベースプラットフォーム回路設計と、第１のカーネルのＲＴＬ記述を統合するステップと、第１のカーネルのＲＴＬ記述から、プロセッサを使用して、第１のカーネルのハードウェア実施態様を指定する第１の構成ビットストリームおよび構成ビットストリームのサポートデータを生成するステップとを含む。方法はまた、第１の構成ビットストリームおよびサポートデータを、バイナリコンテナ内に含めるステップをも含む。
【０００６】
方法は、プロセッサを使用して、ヘテロジニアスマルチプロセッサ設計の第１のカーネルのＲＴＬ記述を生成するステップと、ヘテロジニアスマルチプロセッサ設計のホストに対するプログラム可能ＩＣ内の静的インターフェースを提供するベースプラットフォーム回路設計と、第１のカーネルのＲＴＬ記述を統合するステップと、第１のカーネルのＲＴＬ記述から、プロセッサを使用して、第１のカーネルのＲＴＬ記述のサポートデータを生成するステップとを含む。方法はまた、第１のカーネルのＲＴＬ記述およびサポートデータを、バイナリコンテナ内に含めるステップをも含む。
【０００７】

システムは、実行可能動作を開始するようにプログラムされているプロセッサを含むことができる。実行可能動作は、ヘテロジニアスマルチプロセッサ設計の第１のカーネルのＲＴＬ記述を生成するステップと、ヘテロジニアスマルチプロセッサ設計のホストに対するインターフェースを提供するプログラム可能ＩＣ内の静的領域を提供するベースプラットフォーム回路設計と、第１のカーネルのＲＴＬ記述を統合するステップと、第１のカーネルのＲＴＬ記述から、第１のカーネルのハードウェア実施態様を指定する第１の構成ビットストリームおよび構成ビットストリームのサポートデータを生成するステップとを含む。方法はまた、第１の構成ビットストリームおよびサポートデータを、バイナリコンテナ内に含めるステップをも含むことができる。

【０００８】

この概要部分は、特定の概念を紹介するためにのみ設けられており、特許請求される主題のいかなる重要または本質的な特徴を特定するためではない。本発明の構成の他の特徴は、添付の図面および以下の詳細な説明から明らかとなろう。

【０００９】

本発明の構成は、例として添付の図面に示されている。しかしながら、図面は、本発明の構成を、図示されている特定の実施態様のみに限定するものとして解釈されるべきではない。以下の詳細な説明を検討し、図面を参照すれば、様々な態様および利点が明らかとなろう。

【図面の簡単な説明】

【００１０】

【図１】集積回路（ＩＣ）の例示的なアーキテクチャを示すブロック図である。

【図２】例示的なデータ処理システム（システム）を示すブロック図である。

【図３】図２の対象プラットフォームの例示的なアーキテクチャを示すブロック図である。

【図４】図２および図３の対象プラットフォームを含むヘテロジニアスマルチプロセッサランタイムシステムの例示的な層を示すブロック図である。

【図５】図３のＩＣ内に実装される例示的な回路を示すブロック図である。

【図６】カーネル領域の例示的な実施態様を示すブロック図である。

【図７】ヘテロジニアスマルチプロセッサ設計のカーネルを実装する例示的な方法を示す流れ図である。

【図８】プログラム可能ＩＣ内でヘテロジニアスマルチプロセッサ設計のカーネルを実装するための例示的なプロセスを示すブロック流れ図である。

【図９】プログラム可能ＩＣ内での実装のために、ヘテロジニアスマルチプロセッサコンピューティング言語において指定されるカーネルを処理する例示的な方法を示す流れ図である。

【図１０】ヘテロジニアスマルチプロセッサコンピューティング言語において指定されるカーネルの例示的な処理を示すブロック図である。

【図１１】ヘテロジニアスマルチプロセッサコンピューティング言語において指定されるカーネルの例示的な処理を示すブロック図である。

【図１２】ヘテロジニアスマルチプロセッサコンピューティング言語において指定されるカーネルの例示的な処理を示すブロック図である。

【図１３－１】図１３－２と合わせて、ヘテロジニアスマルチプロセッサコンピューティング言語において指定されるカーネルの処理を示す図である。

【図１３－２】図１３－１と合わせて、ヘテロジニアスマルチプロセッサコンピューティング言語において指定されるカーネルの処理を示す図である。

【図１４】ヘテロジニアスマルチプロセッサコンピューティング言語において指定されるカーネルの例示的な処理を示すブロック図である。

【図１５】例示的なディレクトリ構造の図である。

【図１６】カーネル実行の例示的な方法を示す流れ図である。

【発明を実施するための形態】

【００１１】
図面の詳細な説明
本開示は新規の特徴を規定する特許請求の範囲によって締めくくられているが、本開示内に記載される様々な特徴は、図面と併せて本明細書を考慮することで、より良好に理解されるであろう。本明細書に記載されるプロセス（複数可）、機械（複数可）、製造物（複数可）およびそれらの任意の変形は、例示を目的として与えられている。本開示内に記載される特定の構造的および機能的詳細は、限定的に解釈されるべきではなく、特許請求の範囲の根拠としてのみ、および、実質的に任意で適切に詳述される構造において記載される特徴を様々に利用することを当業者に教示するための代表的な根拠としてのみ、解釈されるべきである。さらに、本開示内で使用される用語および語句は、限定的であるようには意図されておらず、記載される特徴を理解できる記述をすることを意図している。
【００１２】
本開示は、集積回路（ＩＣ）に関し、より詳細には、プログラム可能ＩＣをヘテロジニアスマルチプロセッサシステムに組み込むことに関する。本開示内に記載されている本発明の構成によれば、プログラム可能ＩＣは、１つまたは複数のカーネルを実装するために、ヘテロジニアスマルチプロセッサ設計内で使用することができる。ヘテロジニアスマルチプロセッサのカーネルのうちの１つまたは複数は、コンパイルして、プログラム可能ＩＣのプログラム可能回路を使用して実装されるハードウェアへと変換することができる。これに関連して、カーネルは、中央処理装置（ＣＰＵ）以外のプロセッサに実行のためにオフロードされる実行可能プログラムコードとして実装されるのとは対照的に、回路を使用して実装されるため、プログラム可能ＩＣを使用して実装されるカーネルは、ハードウェアで加速される。ハードウェアへと合成されているヘテロジニアスマルチプロセッサ設計のカーネル部分は、ヘテロジニアスマルチプロセッサ設計のホストと協調して動作する。
【００１３】
一態様において、プログラム可能ＩＣは、ベースプラットフォームを提供することができる。カーネル（複数可）は、ベースプラットフォームで、および／または、ベースプラットフォームと協調して実装することができる。ベースプラットフォームは、カーネルが、プログラム可能ＩＣが結合される対象プラットフォームおよびホストと通信するために必要なインフラストラクチャを提供する。ベースプラットフォームは、たとえば、対象プラットフォームの供給元によって実装または決定されてもよい。したがって、ベースプラットフォームは、使用されるプログラム可能ＩＣの特定のモデルまたはタイプ、および、プログラム可能ＩＣとともに使用される対象プラットフォームのモデルまたはタイプに従って変化してもよい。
【００１４】
本明細書において記載される本発明の構成は、データ処理システムによって実施される方法またはプロセスとして実装されてもよい。一実施例において、方法は、１つまたは複数のカーネルがプログラム可能ＩＣのプログラム可能回路内で実施されるヘテロジニアスマルチプロセッサ設計の実施態様向けに意図されてもよい。別の実施例において、方法は、プログラム可能ＩＣを使用して実装されるカーネルを含むヘテロジニアスマルチプロセッサシステムの動作、たとえば、ランタイム動作向けに意図されてもよい。
【００１５】
別の態様において、本発明の構成は、ＣＰＵを有するデータ処理システムとして実装されてもよい。データ処理システムは、１つまたは複数のカーネルがプログラム可能ＩＣのプログラム可能回路内で実施されるヘテロジニアスマルチプロセッサ設計の実施態様向けに意図されている方法、たとえば、コンパイル時方法を実施することができる。データ処理システムはまた、プログラム可能ＩＣを含むこともできる。その場合、データ処理システムは、プログラム可能ＩＣを使用して実施されるカーネルを含むヘテロジニアスマルチプロセッサシステムの動作、たとえば、ランタイム動作向けに意図されている方法を実施することができる。

【００１６】
また別の態様において、本発明の構成は、ＩＣとして実装されてもよい。ＩＣは、ベースプラットフォームを含むことができる。ＩＣはまた、ベースプラットフォームと協調して動作する、内部に実装されている１つまたは複数のカーネルを含むように構成することもできる。ＩＣは、ＩＣ内で実装されるカーネル（複数可）および／または様々なホスト対話を含む動作のランタイム方法を実施することができる。
【００１７】
また別の態様において、本発明の構成は、実行されると、プロセッサおよび／またはシステムに、本明細書において記載される様々な方法および／またはプロセスを実施および／または開始させるプログラムコードを記憶している非一時的コンピュータ可読記憶媒体として実装されてもよい。
【００１８】
図解を単純かつ明瞭にする目的で、図面内に示されている要素は、必ずしも原寸に比例して描かれているとは限らない。たとえば、いくつかの要素の寸法は、明瞭にするために他の要素に対して誇張されている場合がある。さらに、適切であると考えられる場合、対応する、類似した、または同様の特徴を示すために、参照符号は複数の図面の間で繰り返される。
【００１９】
図１は、ＩＣの例示的なアーキテクチャ１００を示すブロック図である。一態様において、アーキテクチャ１００は、フィールドプログラマブルゲートアレイ（ＦＰＧＡ）タイプのＩＣ内に実装される。アーキテクチャ１００がプロセッサを含む場合、アーキテクチャ１００はまた、ＳＯＣタイプのＩＣをも表す。ＳＯＣは、プログラムコードを実行するプロセッサ、および、１つまたは複数の他の回路システムを含むＩＣである。回路システムは、プロセッサと同じ基板内に実装される。回路システムは、互いに、および、プロセッサと協調して動作することができる。
【００２０】
図示されているように、アーキテクチャ１００は、様々な異なるタイプのプログラム可能回路、たとえば、論理、ブロックを含む。たとえば、アーキテクチャ１００は、マルチギガビット送受信機（ＭＧＴ）１０１、構成可能論理ブロック（ＣＬＢ）１０２、ランダムアクセスメモリブロック（ＢＲＡＭ）１０３、入出力ブロック（ＩＯＢ）１０４、構成およびクロッキング論理（ＣＯＮＦＩＧ／ＣＬＯＣＫＳ）１０５、デジタル信号処理ブロック（ＤＳＰ）１０６、特殊Ｉ／Ｏブロック１０７（たとえば、構成ポートおよびクロックポート）、および、デジタルクロックマネージャ、アナログ－デジタル変換器、システムモニタリング論理などの他のプログラム可能論理１０８を含む、多数の異なるプログラム可能タイルを含むことができる。
【００２１】
いくつかのＩＣにおいて、各プログラム可能タイルは、各隣接するタイル内の対応する相互接続要素（ＩＮＴ）１１１への、および、当該ＩＮＴ　１１１からの標準化された接続を有する、プログラム可能相互接続要素（ＩＮＴ）１１１を含む。それゆえ、ＩＮＴ　１１１は、ともに利用されると、図解されているＩＣのプログラム可能相互接続構造を実装する。各ＩＮＴ　１１１はまた、図１の上部に含まれている実施例によって示されるように、同じタイル内のプログラム可能論理要素への、および、当該要素からの接続をも含む。
【００２２】
たとえば、ＣＬＢ　１０２は、単一のＩＮＴ　１１１に加えて、ユーザ論理を実施するようにプログラムすることができる構成可能論理要素（ＣＬＥ）１１２を含むことができる。ＢＲＡＭ　１０３は、１つまたは複数のＩＮＴ　１１１に加えて、ＢＲＡＭ論理要素（ＢＲＬ）１１３を含むことができる。一般的に、１つのタイル内に含まれるＩＮＴ　１１１の数は、タイルの高さに依存する。図示されているように、ＢＲＡＭタイルは、５つのＣＬＢと同じ高さを有するが、他の数（たとえば、４つ）も使用されてもよい。ＤＳＰ

タイル１０６は、適切な数のＩＮＴ　１１１に加えて、ＤＳＰ論理要素（ＤＳＰＬ）１１４を含むことができる。ＩＯＢ　１０４は、ＩＮＴ　１１１の１つのインスタンスに加えて、たとえば、Ｉ／Ｏ論理要素（ＩＯＬ）１１５の２つのインスタンスを含むことができる。当業者にとって明らかになるように、たとえば、ＩＯＬ　１１５に接続されている実際のＩ／Ｏパッドは一般的には、ＩＯＬ　１１５の領域に閉じ込められない。

【００２３】

図１に示す実施例において、たとえば、領域１０５、１０７、および１０８から形成される、ダイの中心付近の列状領域は、構成、クロック、および他の制御論理に使用することができる。この列から延伸する水平領域１０９は、クロックおよび構成信号をプログラム可能ＩＣの幅にわたって分散させるために使用される。

【００２４】

図１に示すアーキテクチャを利用するいくつかのＩＣは、このＩＣの大部分を構成する規則的な列状構造を乱す追加の論理ブロックを含む。追加の論理ブロックは、プログラム可能ブロックおよび／または専用回路であってもよい。たとえば、ＰＲＯＣ　１１０として示されている任意選択のプロセッサブロックが、ＣＬＢおよびＢＲＡＭのいくつかの列にわたる。

【００２５】

一態様において、ＰＲＯＣ　１１０は、ＩＣのプログラム可能回路を実装するダイの一部分として作製される専用回路として、たとえば、ハードワイヤードプロセッサとして実装される。ＰＲＯＣ　１１０は、個々のプロセッサ、たとえば、プログラムコードを実行することが可能な単一のコアから、１つまたは複数のコア、モジュール、コプロセッサ、インターフェースなどを有するプロセッサシステム全体まで複雑に様々な異なるプロセッサタイプおよび／またはシステムのいずれかを表すことができる。

【００２６】

別の態様において、ＰＲＯＣ　１１０は、アーキテクチャ１００から省かれ、記載されている他の様々なプログラム可能ブロックのうちの１つまたは複数と置き換えられる。さらに、そのようなブロックは、プログラム可能論理の様々なブロックを使用して、ＰＲＯＣ　１１０と同様にプログラムコードを実行することができるプロセッサを形成することができるという点において、「ソフトプロセッサ」を形成するために利用されてもよい。

【００２７】

「プログラム可能回路」という語句は、ＩＣ内のプログラム可能回路要素、たとえば、本明細書において記載されている様々なプログラム可能もしくは構成可能回路ブロックもしくはタイル、ならびに、ＩＣにロードされる構成データに従って様々な回路ブロック、タイル、および／もしくは要素を選択的に結合する相互接続回路を指す。たとえば、図１に示す、ＣＬＢ　１０２およびＢＲＡＭ　１０３のような、ＰＲＯＣ　１１０の外部にある部分は、ＩＣのプログラム可能回路と考えられる。

【００２８】

一般的に、プログラム可能回路の機能は、構成データがＩＣにロードされるまで確立されない。構成ビットセットを使用して、ＦＰＧＡのようなＩＣのプログラム可能回路をプログラムすることができる。構成ビット（複数可）は一般的に、「構成ビットストリーム」として参照される。一般的に、プログラム可能回路は、最初に構成ビットストリームをＩＣにロードしなければ動作可能または機能可能でない。構成ビットストリームは実効的に、プログラム可能回路内の特定の回路設計を実装またはインスタンス化する。回路設計は、たとえば、プログラム可能回路ブロックの機能態様、および、様々なプログラム可能回路ブロックの間の物理接続を指定する。

【００２９】

「ハードワイヤード」または「ハード化」されている、すなわち、プログラム可能でない回路は、ＩＣの一部分として製造される。プログラム可能回路とは異なり、ハードワイヤード回路または回路ブロックは、ＩＣの製造後に構成ビットストリームをロードすることによって実装されるのではない。ハードワイヤード回路は一般的に、たとえば、最初に

構成ビットストリームをＩＣ、たとえば、ＰＲＯＣ　１１０にロードすることなく機能可能である専用回路ブロックおよび相互接続を有すると考えられる。

【００３０】

場合によっては、ハードワイヤード回路は、ＩＣ内の１つまたは複数のメモリ要素内に記憶されているレジスタ設定または値に従って設定または選択することができる１つまたは複数の動作モードを有することができる。動作モードは、たとえば、構成ビットストリームをＩＣにロードすることによって設定することができる。この機能があるにもかかわらず、ハードワイヤード回路は、ＩＣの一部分として製造されるときに動作可能であり、特定の機能を有するため、プログラム可能回路とは考えられない。

【００３１】

ＳＯＣの場合、構成ビットストリームは、プログラム可能回路内に実装されるべきである回路、および、ＰＲＯＣ　１１０またはソフトプロセッサによって実行されるべきであるプログラムコードを指定することができる。いくつかの事例において、アーキテクチャ１００は、構成ビットストリームを適切な構成メモリおよび／またはプロセッサメモリにロードする専用構成プロセッサを含む。構成プロセッサは、含まれる場合のＰＲＯＣ　１１０とは異なり、ユーザプログラムコードを実行しない。他の事例において、アーキテクチャ１００は、ＰＲＯＣ　１１０を利用して、構成ビットストリームを受信し、構成ビットストリームを適切な構成メモリにロードし、かつ／または、プログラムコードを実行のために抽出することができる。

【００３２】

図１は、プログラム可能回路、たとえば、プログラマブルファブリックを含むＩＣを実装するために使用することができる例示的なアーキテクチャを示すように意図されている。たとえば、列内の論理ブロックの数、列の相対幅、列の数および順序、列内に含まれる論理ブロックのタイプ、論理ブロックの相対サイズ、および、図１の上部に含まれている相互接続／論理実施態様は、単なる例示である。実際のＩＣにおいて、たとえば、ユーザ回路設計の効率的な実施を容易にするために、ＣＬＢが現れるところであればどこであっても、一般的に、ＣＬＢの２つ以上の隣接する列が含まれる。しかしながら、隣接するＣＬＢ列の数は、ＩＣの全体サイズによって変化してもよい。さらに、ＩＣ内でのＰＲＯＣ　１１０のようなブロックのサイズおよび／または位置付けは、例示のみを目的としており、限定することを意図していない。

【００３３】

図２は、例示的なデータ処理システム（システム）２００を示すブロック図である。一態様において、システム２００は、プログラム可能ＩＣ内の回路としての、ヘテロジニアスマルチプロセッサ設計のカーネル、たとえば、プログラムを実装するようにプログラムすることができるコンパイル時システムを表すことができる。本明細書における定義としては、「ヘテロジニアスマルチプロセッサ設計」は、ホストシステム上で実行する一部分、および、異なるデバイスまたはプロセッサ上で実行するカーネルと称される少なくとも１つの追加の部分を含むプログラムである。ヘテロジニアスマルチプロセッサ設計の一実施例は、ＯｐｅｎＣＬプログラムまたは設計である。一実施例において、ホスト上で実行する部分は、異なるデバイスまたはプロセッサ上で実行する部分とは異なるプログラミング言語で指定することができる。プログラム可能ＩＣは、図１を参照して説明されているようなアーキテクチャを有することができる。

【００３４】

別の態様において、システム２００は、プロセッサがホストとして機能し、プログラム可能ＩＣが１つまたは複数のカーネルを実装するランタイムヘテロジニアスマルチプロセッサシステムを表すことができる。本明細書において定義されている「ヘテロジニアスマルチプロセッサシステム」は、２つ以上のプロセッサを含むコンピューティングシステムである。２つ以上のプロセッサは、異なるタイプのプロセッサとすることができる。たとえば、ヘテロジニアスマルチプロセッサシステムは、中央処理装置（ＣＰＵ）、グラフィックス処理ユニット（ＧＰＵ）、デジタル信号プロセッサ（ＤＳＰ）、ＦＰＧＡのような

プログラム可能ＩＣなどを含んでもよい。ヘテロジニアスマルチプロセッサシステムは、ＯｐｅｎＣＬシステムであってもよい。

【００３５】

図示されているように、システム２００は、システムバス２１５を通じてメモリ要素２１０または他の適切な回路に結合されている少なくとも１つのプロセッサ、たとえば、中央処理装置（ＣＰＵ）２０５を含む。システム２００は、メモリ要素２１０内にプログラムコードを記憶する。プロセッサ２０５は、システムバス２１５を介してメモリ要素２１０からアクセスされるプログラムコードを実行する。一態様において、システム２００は、プログラムコードを記憶および／または実行するのに適しているコンピュータまたは他のデータ処理システムとして実装される。しかしながら、システム２００は、本開示内で記載されている機能を実施することが可能である、プロセッサおよびメモリを含む任意のシステムの形態で実装されてもよいことが諒解されるべきである。さらに、システム２００は、１つまたは複数のネットワーク化データ処理システム、たとえば、サーバとして実装されてもよい。

【００３６】

メモリ要素２１０は、たとえば、ローカルメモリ２２０および１つまたは複数の大容量記憶デバイス２２５のような、１つまたは複数の物理メモリデバイスを含む。ローカルメモリ２２０は、一般的にプログラムコードの実際の実行中に使用されるランダムアクセスメモリ（ＲＡＭ）または他の非持続性メモリデバイス（複数可）を指す。大容量記憶デバイス２２５は、ハードディスクドライブ（ＨＤＤ）、ソリッドステートドライブ（ＳＳＤ）、または他の持続性データ記憶デバイスとして実装されてもよい。システム２００はまた、実行中にプログラムコードが大容量記憶デバイス２２５から取り出されなければならない回数を低減するために、少なくともいくらかのプログラムコードの一時記憶を可能にする１つまたは複数のキャッシュメモリ（図示せず）をも含むことができる。

【００３７】

キーボード２３０、ディスプレイデバイス２３５、およびポインティングデバイス２４０のような入出力（Ｉ／Ｏ）デバイスが、任意選択的にシステム２００に結合されてもよい。Ｉ／Ｏデバイスは、直接的に、または、介在するＩ／Ｏコントローラを通じてシステム２００に結合することができる。システム２００が介在する私的または公衆ネットワークを通じて、他のシステム、コンピュータシステム、遠隔プリンタ、遠隔記憶デバイス、および／または、対象プラットフォーム２６０に結合されることを可能にするために、ネットワークアダプタ２４５もまた、システム２００に結合することができる。システム２００とともに使用することができる種々のタイプのネットワークアダプタ２４５の実施例は、モデム、ケーブルモデム、Ｅｔｈｅｒｎｅｔ（登録商標）カード、および、ワイヤレス送受信機である。システム２００が対象プラットフォーム２６０を含む前述したシステムのいずれかのような別のシステムに結合されることを可能にするために、ユニバーサルシリアルバスポート、ＦｉｒｅＷｉｒｅポート、周辺構成要素相互接続（ＰＣＩ）および／またはＰＣＩ　Ｅｘｐｒｅｓｓ（ＰＣＩｅ）ポートなどのような通信ポート２５０もまた、システム２００に結合することができる。

【００３８】

一態様において、メモリ要素２１０は、電子設計自動化（ＥＤＡ）アプリケーション２５５を記憶する。ＥＤＡアプリケーション２５５は、たとえば、システム２００がコンパイル時システムを表す実施態様において記憶され得る。ＥＤＡアプリケーション２５５は、１つまたは複数の異なる構成要素またはモジュールを含むことができる。実行可能プログラムコードの形態で実装されるＥＤＡアプリケーション２５５は、システム２００によって実行される。そのため、ＥＤＡアプリケーション２５５は、システム２００の一体部分と考えられる。ＥＤＡアプリケーション２５５、ならびに、ＥＤＡアプリケーション２５５を実行している間にシステム２００によって使用、生成、および／または生成される任意のデータ項目は、システム２００の一部分として利用されるときに機能性を付与する機能データ構造である。コンパイル時システムとしては、ホストアプリケーション２５８

が、システム２００から除外され得る。

【００３９】

コンパイル時システムの場合、ユーザは、ＥＤＡアプリケーション２５５を実行するシステム２００を通じて作業する。システム２００は、ヘテロジニアスマルチプロセッサ設計２７５を入力として受信し、ヘテロジニアスマルチプロセッサ設計２７５の１つまたは複数のカーネルを、ＩＣ２７０内で実装することができる回路へと合成することができる。システム２００は、バイナリコンテナ２８０を生成および出力することができる。一態様において、バイナリコンテナ２８０は、部分または全体のいずれかにかかわらず、その内部の内容の記述および１つまたは複数の構成ビットストリームを含むことができる。別の態様において、バイナリコンテナ２８０は、その内部の内容の記述、１つまたは複数の実行可能シミュレーションファイル、および／または、レジスタ転送レベル（ＲＴＬ）もしくはハードウェア記述言語シミュレータ内でシミュレートすることができる１つもしくは複数のＲＴＬファイルを含むことができる。その場合、バイナリコンテナ２８０は、実行可能シミュレーションファイル（複数可）および／またはＲＴＬファイル（複数可）に加えて、部分または全体のいずれかにかかわらず、１つまたは複数の構成ビットストリームを含むことができる。バイナリコンテナ２８０は、メモリ要素２１０内に記憶されてもよく、かつ／または、ネットワークアダプタ２４５および／もしくは通信ポート２５０によって別のシステムに提供されてもよい。

【００４０】

別の態様において、メモリ要素２１０は、ホストアプリケーション２５８を記憶する。ホストアプリケーション２５８は、たとえば、システム２００がヘテロジニアスマルチプロセッサランタイムシステムを表す実施態様において記憶され得る。ホストアプリケーション２５８は、１つまたは複数の異なる構成要素またはモジュールを含むことができる。実行可能プログラムコードの形態で実装されるホストアプリケーション２５８は、システム２００によって実行される。そのため、ホストアプリケーション２５８は、システム２００の一体部分と考えられる。ホストアプリケーション２５８、ならびに、ホストアプリケーション２５８を実行している間にシステム２００によって使用、生成、および／または生成される任意のデータ項目は、システム２００の一部分として利用されるときに機能性を付与する機能データ構造である。ランタイムシステムとしては、ＥＤＡアプリケーション２５５が、システム２００から除外され得る。

【００４１】

システム２００は、通信リンク２６５を通じて対象プラットフォーム２６０に結合することができる。ランタイムシステム実施態様の場合、対象プラットフォーム２６０は、システム２００に結合されるか、または、その一部分と考えられる。したがって、コンパイル時システムの場合、対象プラットフォーム２６０は除外され得ることが諒解されるべきである。引き続き対象プラットフォーム２６０について、通信リンク２６５は、通信ポート２５０および／またはネットワークアダプタ２４５に結合するように動作可能である様々な異なる有線および／または無線接続のいずれかとして実装されてもよい。

【００４２】

対象プラットフォーム２６０は、回路が実装されているプリント回路基板のような回路基板として実装されてもよい。対象プラットフォームは、たとえば、システム２００内の、または、システム２００の外部の通信ポート２５０のための機械的コネクタに差し込むことができるカードとして実装されてもよい。対象プラットフォーム２６０は、通信リンク２６５に結合するコネクタを含むことができる。コネクタは、対象プラットフォーム２６０の回路を使用して、ＩＣ　２７０に結合することができる。

【００４３】

ＩＣ　２７０は、ソケット、レセプタクル、ＩＣ　２７０を対象プラットフォーム２６０に直接的にはんだ付けすることのような別の取り付け技法などを使用して、対象プラットフォーム２６０に結合することができる。ＩＣ　２７０は、対象プラットフォーム２６０を通じて通信リンク２６５に結合する。一態様において、ＩＣ　２７０はプログラム可

能ＩＣである。ＩＣ　２７０は、たとえば、図１を参照して説明されているアーキテクチャを使用して実装されてもよい。別の態様において、ＩＣ　２７０は、ＳＯＣとして実装されてもよい。ＩＣ　２７０は、ヘテロジニアスマルチプロセッサ設計の１つまたは複数のカーネルを、回路として実装することができる。ヘテロジニアスマルチプロセッサ設計は、ＯｐｅｎＣＬ設計であってもよい。

【００４４】

ランタイムシステムの場合、プロセッサ２０５はホストとして動作することができる。ヘテロジニアスマルチプロセッサ設計の１つまたは複数のカーネルは、ＩＣ　２７０内に実装することができる。動作中、ＩＣ　２７０は、場合によって構成または再構成されないＩＣ　２７０の他の部分に対する妨害を引き起こすことなく、動作中である間に場合によって動的に構成または再構成することができるため、必要に応じて、ＩＣ　２７０内に新たなおよび／または異なるカーネルを実装することができる。

【００４５】

図３は、図２の対象プラットフォーム２６０の例示的なアーキテクチャを示すブロック図である。図示されているように、ＩＣ　２７０およびＲＡＭ　３４５が、対象プラットフォーム２６０に結合されている。対象プラットフォーム２６０はまた、ＩＣ　２７０に結合されているコネクタ３５０をもしくは含む。カードエッジタイプのコネクタとして示されているが、コネクタ３５０は、様々な異なるコネクタタイプのうちのいずれかとして実装されてもよいことが諒解されるべきである。さらに、対象プラットフォーム２６０は、１つまたは複数の他の構成要素（図示せず）を含んでもよい。追加の構成要素は、たとえば、コネクタ３５０とＩＣ　２７０との間に結合されてもよい。

【００４６】

ＩＣ　２７０は、静的領域３３５とカーネル領域３４０とを含む。一態様において、静的領域３３５は、ヘテロジニアスマルチプロセッサプログラミングモデルをサポートするために必要とされるインフラストラクチャＩＰを含む。一実施例において、ヘテロジニアスマルチプロセッサプログラミングモデルは、ＯｐｅｎＣＬモデルである。静的領域３３５は、たとえば、実行時間中に、カーネル領域３４０を、ＲＡＭ　３４５のような対象プラットフォーム２６０上に位置する他の構成要素、および／または、ホストのような他のシステム、たとえば、プロセッサ２０５と通信可能にリンクする。静的領域３３５は、たとえば、ホストと通信するために使用されるソフトウェアインターフェースを実装することができる。一態様において、静的領域３３５は、対象プラットフォーム２６０の供給元および／または製造元によって提供される回路実施態様であってもよい。

【００４７】

カーネル領域３４０は、ＩＣ　３３０の、カーネルが実装されている部分を表す。一態様において、カーネル領域３４０は、静的領域３３５とのメモリマップドインターフェースを有することができる。カーネル領域３４０は、静的領域３３５とは異なり、動的に生成し、静的領域３３５と統合することができる。たとえば、異なるカーネルおよびカーネルの異なる組み合わせを、ランタイム中にカーネル領域３４０内で異なる時点において実装することができる。

【００４８】

図４は、対象プラットフォーム２６０を含むヘテロジニアスマルチプロセッサランタイムシステムの例示的な層を示すブロック図である。一実施例において、ヘテロジニアスマルチプロセッサランタイムシステムは、ＯｐｅｎＣＬシステムである。図示されているように、ホストは、ホストアプリケーション内に実装されるランタイム層４０５を実行する。論じられているように、ホストは、図２を参照して説明されているシステム２００のプロセッサ２０５として実装することができる。対象プラットフォームソフトウェア層４１５が、対象プラットフォーム回路内に実装される。ランタイム層４０５は、共通の低レベルドライバインターフェース４１０を通じて対象プラットフォームソフトウェア層４１５と通信する。たとえば、ランタイム層４０５は、対象プラットフォームソフトウェア層４１５と通信するために共通の低レベルドライバ４１０において定義されている、標準的な

文書化されたアプリケーションプログラミングインターフェース（ＡＰＩ）を使用する。対象プラットフォームソフトウェア層４１５は、たとえば、カーネルドライバとして実装されてもよい。

【００４９】

対象プラットフォーム２６０の回路内で実行する対象プラットフォームソフトウェア層４１５は、対象プラットフォーム特有のプログラミングインターフェース４２０、たとえば、ハードウェアプログラミングインターフェースを通じて、静的領域３３５と通信する。静的領域３３５は、カーネル領域３４０に、クロックおよびリセット信号４３０を提供する。静的領域３３５はまた、制御レジスタ（図示せず）に結合されているメモリマップドスレーブインターフェース４４０を通じてカーネル領域３４０に情報を提供する。カーネル領域３４０は、ＲＡＭ　３４５に結合されているメモリマップドバスマスタインターフェース４３５を通じて静的領域３３５に情報を提供する。

【００５０】

図５は、図３のＩＣ　２７０内に実装される例示的な回路を示すブロック図である。より詳細には、図５は、静的領域３３５を実装するために使用することができる例示的なアーキテクチャを示す。ブロック５０５、５１０、５１５、５２０、および５２５の各々は、回路ブロックを表す。静的領域３３５の部分としてのブロック５０５～５２５の各々、および、カーネル領域３４０は、ＩＣ　２７０のプログラム可能回路内で実装することができる。

【００５１】

図示されているように、静的領域３３５は、バスダイレクトメモリアクセス（ＤＭＡ）コントローラ５１０に結合されているバスエンドポイント５０５を含むことができる。バスＤＭＡコントローラ５１０は、インターコネクト５１５に結合されている。インターコネクト５１５は、インターコネクト５２０およびカーネル領域３４０に結合する。インターコネクト５２０は、カーネル領域３４０およびメモリコントローラ５２５に結合する。メモリコントローラ５２５は、ＩＣ　２７０の外部で実装されているＲＡＭ　３４５に結合する。

【００５２】

バスエンドポイント５０５は、バスを介してヘテロジニアスマルチプロセッサ設計のホストと通信するように構成されている。バスＤＭＡコントローラ５１０は、ホストＲＡＭ、たとえば、ローカルメモリ２２０と対象プラットフォーム２６０上のＲＡＭ　３４５との間のＤＭＡ機能をサポートするために含むことができる。一態様において、バスＤＭＡコントローラ５１０は、マスタインターフェース５３０を含む。インターコネクト５１５は、スレーブインターフェース５３５ならびにマスタインターフェース５４０および５４５を含むことができる。図示されているように、スレーブインターフェース５３５は、マスタインターフェース５３０に結合されている。カーネル領域３４０は、スレーブインターフェース５５０およびマスタインターフェース５５５を含む。インターコネクト５１５のマスタインターフェース５４５は、カーネル領域３４０のスレーブインターフェース５５０に結合されている。

【００５３】

インターコネクト５２０は、スレーブインターフェース５６０および５６５ならびにマスタインターフェース５７０を含む。メモリコントローラ５２５は、スレーブインターフェース５７５を含む。図示されているように、インターコネクト５１５のマスタインターフェース５４０は、インターコネクト５２０のスレーブインターフェース５６０に結合されている。カーネル領域３４０のマスタインターフェース５５５は、インターコネクト５２０のスレーブインターフェース５６５に結合されている。

【００５４】

インターコネクト５１５および５２０は、２つ以上の他の回路ブロックをともに結合するように構成されている回路ブロックである。一態様において、インターコネクト５１５および５２０は、１つまたは複数のメモリマップドマスタデバイスを、１つまたは複数の

メモリマップドスレーブデバイスと結合する回路ブロックとして実装することができる。相互接続回路ブロック実施態様の一実施例は、英国ケンブリッジ所在のＡＲＭ（登録商標）ＬｔｄのＡＭＢＡ（登録商標）ＡＸＩバージョン４仕様に一致するものである。しかしながら、インターコネクト５１５および５２０を実装するために、他の相互接続タイプおよび／または技術が使用されてもよいことは諒解されるべきである。本開示は、与えられている例示的な相互接続回路ブロックによって限定されるようには意図されていない。

【００５５】

図５に示されているアーキテクチャ内で、バスＤＭＡコントローラ５１０およびカーネル領域３４０は、メモリコントローラ５２５に対するマスタとして機能する。インターコネクト５１５は、ホストが、たとえば、バスを介してＲＡＭ　３４５に対して読み出しおよび書き込みを行うことを可能にする。インターコネクト５２０は、メモリコントローラ５２５に対する２つのバスマスタ、すなわち、バスＤＭＡコントローラ５１０およびカーネル領域３４０の作成をサポートする。

【００５６】

カーネル領域３４０は、まず、コンパイルされたヘテロジニアスマルチプロセッサ設計カーネルのコンテナとして実装され得る。一態様において、カーネル領域３４０は、コンパイルされたカーネルのためのプレースホルダを有する階層型ＩＰとして実装することができる。１つまたは複数のカーネルが、カーネル領域３４０内に含まれ得る。一実施例において、最大１６個のカーネルが、カーネル領域３４０内に含まれ得る。ホストからのコマンドが、スレーブインターフェース５５０を通じて受信され得る。カーネル領域３４０は、マスタインターフェース５５５を通じてメモリコントローラ５２５にコマンドを与えることができる。マスタインターフェース５４５とスレーブインターフェース５５０との間の接続を通じて、カーネル領域３４０および内部に実装されている任意のカーネルにクロックおよびリセット信号が与えられる。

【００５７】

図６は、カーネル領域３４０の例示的な実施態様を示すブロック図である。図示されているように、カーネル領域３４０は追加のインターコネクト６０５および６１５を含む。インターコネクト６０５は、インターコネクト５１５のマスタインターフェース５４５に結合されているスレーブインターフェース５５０を含む。インターコネクト６０５は、カーネル回路６１０－１のスレーブインターフェース６３０に結合されているマスタインターフェース６２５をさらに含む。マスタインターフェース６２５はまた、１つまたは複数の他のカーネル回路６１０－Ｎにも結合することができ、Ｎは整数値である。

【００５８】

まとめてカーネル回路６１０－１～６１０－Ｎとして示されるカーネル回路６１０は、同じカーネル回路の複数のインスタンスを表すことができ、そのため、同じカーネルの複数のインスタンスを表すことができる。別の実施例において、カーネル回路６１０は、２つ以上の異なるカーネル回路を表すことができる。また別の実施例において、カーネル回路６１０は、第１のカーネル回路、および、１つまたは複数の追加の異なるカーネル回路の１つまたは複数のインスタンスを表すことができる。インターコネクト６１５は、各カーネル回路６１０のマスタインターフェース６４０および６４５に結合するスレーブインターフェース６５０を有する。インターコネクト６１５は、インターコネクト５２０のスレーブインターフェース５６５に結合するマスタインターフェース５５５を含む。

【００５９】

一態様において、インターコネクト６０５および６１５は、カーネル回路の最大１６個の異なるインスタンス、１６個の異なるカーネル回路、または、１６を超えないようなその組み合わせをサポートすることができる。論じられているように、カーネル領域３４０内で実装することができるカーネルおよび／またはカーネルインスタンスの特定の数は、限定ではなく例示を目的として与えられている。

【００６０】

ＯｐｅｎＣＬのようなヘテロジニアスマルチプロセッサフレームワーク内で、並列カー

ネル呼び出しは、ＮＤＲａｎｇｅとして参照される１、２、または３Ｄインデックス空間として記述され得る。ＮＤＲａｎｇｅは、複数のワークグループに分割される。ワークグループは、複数のワークアイテムを含む。たとえば、ＮＤＲａｎｇｅ内の各ポイントが、ワークアイテムとして参照される。

【００６１】

ヘテロジニアスマルチプロセッサ設計のカーネルは、１つまたは複数のコンピュートユニットにコンパイルされる。システム設計者、たとえば、ユーザは、所与のカーネルについて並列に実装されるべきであるコンピュートユニットの数を決定する。一態様において、カーネルのコンピュートユニットの数は、カーネル領域３４０内で実装され、並列に動作するカーネル回路のインスタンスの数を示す。各コンピュートユニットは、ホストによって決定され、与えられるものとしての１つのワークグループを処理することが可能である。

【００６２】

図６の実施例において、各カーネル回路ブロック６１０－１～６１０－Ｎは、ワークユニットの回路を表す。カーネル回路ブロック６１０－１～６１０－Ｎは、同じカーネルのワークユニット、たとえば、並列に動作する複数のインスタンス、または、カーネルのうちの１つまたは複数が複数のインスタンスによってカーネル領域３４０内で実装される、並列に動作する２つ以上のカーネルのワークユニットを表すことができる。

【００６３】

図７は、ヘテロジニアスマルチプロセッサ設計のカーネルを実装する例示的な方法７００を示す流れ図である。一実施例において、ヘテロジニアスマルチプロセッサ設計は、ＯｐｅｎＣＬ設計であってもよい。方法７００は、図２を参照して説明されているシステムのような、コンパイル時システムによって実施され得る。方法７００は、ヘテロジニアスマルチプロセッサ設計が、Ｃ、Ｃ＋＋、ＯｐｅｎＣＬ、ＯｐｅｎＣＬ　Ｃ、ＯｐｅｎＣＬ互換高レベルプログラミング言語、または、他の高レベルプログラミング言語で指定されるカーネルを含む状態で開始することができる。一態様において、様々な高レベルプログラミング言語のいずれかが、カーネルを指定するために使用されてもよい。さらなる態様において、カーネルを指定するために使用される高レベルプログラミング言語は、並列性または並列動作の明示的な仕様または表記をサポートするものであってもよい。システムはカーネルにアクセスすることができる。

【００６４】

ブロック７０５において、システムは、カーネルのＲＴＬ記述を生成する。ＲＴＬ記述は、ハードウェア記述言語（ＨＤＬ）を使用して指定することができる。本明細書での定義として、「ハードウェア記述言語」または「ＨＤＬ」という用語は、集積回路のようなデジタルシステムの文書化、設計、および製造を促進するコンピュータ言語である。ＨＤＬは、プログラム検証技法を、エキスパートシステム設計方法と組み合わせる。ＨＤＬを使用して、たとえば、ユーザは、電子回路を設計して指定し、回路の動作を記述し、回路の動作を検証するための試験を作成することができる。ＨＤＬは、モデル化されている電子システムの空間的および時間的構造ならびに挙動の標準的なテキストベースの表現を含む。ＨＤＬシンタックスおよびセマンティクスは、同時並行性を表現するための明示的な表記を含む。ほとんどの高レベルプログラミング言語とは対照的に、ＨＤＬはまた、デジタルシステムの最重要属性である、時間の明示的な表記をも含む。

【００６５】

ブロック７１０において、システムは、カーネルのＲＴＬ記述を、ベースプラットフォームと統合する。一態様において、ベースプラットフォームは、静的領域３３５内に実装され、図４および／または図５を参照して説明されている回路設計と同様または同じであってもよい。

【００６６】

ブロック７１５において、システムは、構成ビットストリームおよびサポートデータを生成する。構成ビットストリームは、カーネルのハードウェア実施態様、たとえば、図６

を参照して説明されているようなコンピュートユニットを指定する。一態様において、構成ビットストリームは、たとえば、カーネル、または、１つもしくは複数のカーネルのみを指定する部分ビットストリームであってもよい。別の態様において、構成ビットストリームは、カーネル、または、場合によってカーネルおよびベースプラットフォームを指定する全ビットストリームであってもよい。

【００６７】

サポートデータは、構成ビットストリームおよび／または構成ビットストリームの内容を記述する。一態様において、サポートデータは、カーネル実施態様に含まれているＩＰブロックおよび／またはコアのリストを指定することができる。別の態様において、サポートデータは、部分構成ビットストリームとして指定されるときに、カーネルのハードウェア実施態様が実装されることになる、プログラム可能ＩＣ内の２次元座標位置を指定することができる。

【００６８】

ブロック７２０において、システムは、構成ビットストリームおよびサポートデータを、バイナリコンテナ内に含める。一態様において、バイナリコンテナは、複数の個別ファイルを含むことができる。たとえば、バイナリコンテナは、１つまたは複数の構成ビットストリームおよび１つまたは複数のサポートデータファイルを含むことができる。

【００６９】

別の態様において、カーネル（複数可）のＲＴＬ記述が、バイナリコンテナ内に含まれてもよい。ＲＴＬ記述はその後、ＲＴＬシミュレータを用いて、全体的なヘテロジニアスマルチプロセッサ設計シミュレーションの一部分としてカーネル実施態様を試験するために使用することができる。たとえば、ホストは、ヘテロジニアスマルチプロセッサ設計のランタイムシミュレーション中に、ＲＴＬ記述（複数可）を含むバイナリコンテナを、ＲＴＬシミュレータに与えることができる。ＲＴＬシミュレータは、バイナリコンテナからＲＴＬ記述にアクセスすることができる。また別の態様において、試験および／またはシミュレーションを目的としてプロセッサを使用して実行することができる実行可能バージョンのカーネル（複数可）が、バイナリコンテナ内に含まれてもよい。たとえば、ホストは、ヘテロジニアスマルチプロセッサ設計のランタイムシミュレーション中に、実行可能バージョンのカーネルを含むバイナリコンテナを、シミュレータに与えることができる。カーネルの実行可能バージョンは、カーネルのハードウェア実施態様の実行可能モデルであってもよいことが諒解されるべきである。シミュレータは、バイナリコンテナからカーネルの実行可能バージョンにアクセスすることができる。したがって、バイナリコンテナは、プログラム可能ＩＣによるランタイムの構成ビットストリームとして、データ処理システムに対するシミュレーションのための実行可能バージョンであるか、および／または、ＲＴＬシミュレータを使用したシミュレーションのためのＲＴＬバージョンであるかにかかわらず、複数の異なるカーネル実施態様をサポートする。

【００７０】

バイナリコンテナは、構成ビットストリーム（複数可）のみ、カーネル（複数可）の実行可能バージョン（複数可）のみ、カーネル（複数可）のＲＴＬバージョン（複数可）のみ、構成ビットストリームおよびカーネルのＲＴＬバージョン、構成ビットストリームおよびカーネルの実行可能バージョン、カーネル（複数可）の実行可能バージョンおよびＲＴＬバージョン、または、構成ビットストリーム、カーネル（複数可）の実行可能バージョン、およびカーネル（複数可）のＲＴＬバージョンを含むことができる。サポートデータはまた、バイナリコンテナ内で実装されるカーネルバージョンの上述した組み合わせのいずれかのために含まれ得る。ヘテロジニアスマルチプロセッサ設計、特に、ＯｐｅｎＣＬ設計においてＣＰＵおよび／またはＧＰＵ供給元によって使用される既存のコンテナは、「インメモリ」で協働し、マッピングされたオブジェクトに対処する。そのようなコンテナは、カーネルのシミュレーションバージョンまたは同じコンテナ内の複数の異なるカーネルタイプをサポートしない。

【００７１】

使用されているバイナリコンテナは、複数の異なるタイプのカーネル実施態様をサポートすることができるが、一態様において、第１のコンテナが、第１のタイプのカーネル実施態様、たとえば、構成ビットストリーム、ＲＴＬ記述、または実行ファイルを含んでもよく、一方で、第２のバイナリコンテナが、異なるタイプのカーネル実施態様を含んでもよい。また別の態様において、第１のコンテナが、第１のカーネルを指定する部分構成ビットストリームを含んでもよく、一方で、第２のコンテナが、第２の異なるカーネルを指定する部分構成ビットストリームを含んでもよい。

【００７２】

ホストとカーネルとの通信のための、ＯｐｅｎＣＬのようなヘテロジニアスマルチプロセッサコンピューティング言語のための標準的なＡＰＩは、バイナリオブジェクトファイルのみをサポートする。システムによって生成されるバイナリコンテナは、すべての予めコンパイルされたカーネルが自己完結型オブジェクトを通じてアクセス可能であるべきであるというこのバイナリ要件に従う。ブロック７２０において生成されるバイナリコンテナは、ホストによって、ランタイム中にプログラム可能ＩＣ内でカーネル回路、たとえば、コンピュートユニットを実装するために使用することができる。

【００７３】

図８は、プログラム可能ＩＣ内にヘテロジニアスマルチプロセッサ設計のカーネルを実装するための例示的なプロセス８００を示すブロック流れ図である。一実施例において、ヘテロジニアスマルチプロセッサ設計は、ＯｐｅｎＣＬ設計であってもよい。プロセス８００は、ＥＤＡアプリケーション２５５を実行する、図２を参照して説明されているシステムのようなシステムによって実施され得る。一態様において、ＥＤＡアプリケーション２５５は、ＯｐｅｎＣＬコンパイラツールとして実装されてもよい。プロセス８００は、例示的なコンパイル時システム実施態様を示す。

【００７４】

図示されているように、たとえば、ヘテロジニアスマルチプロセッサアプリケーション、ＯｐｅｎＣＬ設計、または、ＯｐｅｎＣＬアプリケーションとしても参照されるヘテロジニアスマルチプロセッサ設計２７５は、ホストコード８０５およびカーネル８１５を含むことができる。１つのカーネルが図示されているが、ヘテロジニアスマルチプロセッサ設計２７５は、プロセス８００を通じて実装することができる２つ以上のカーネルを含むことができることが諒解されるべきである。ホストコード８０５は、ヘテロジニアスマルチプロセッサ設計２７５の、ホスト内で実行する部分である。ホストコード８０５は、Ｃ、Ｃ＋＋などのような高レベルプログラミング言語で指定することができる。

【００７５】

本明細書における定義として、「高レベルプログラミング言語」という用語は、命令がデータ処理システムの詳細からの強い抽象化を有する、データ処理システムをプログラムするために使用されるプログラミング言語または命令セット、たとえば、機械言語を意味する。たとえば、高レベルプログラミング言語は、メモリ管理のような、データ処理システムの動作の態様を自動化するか、または、隠すことができる。抽象化の量が、一般的に、そのプログラミング言語がどれだけ「高レベル」であるかを規定する。高レベルプログラミング言語が使用されるとき、ユーザは、高レベルプログラミング言語が実行するデータ処理システムのレジスタ、メモリアドレスなどに対処する必要がない。これに関連して、高レベルプログラミング言語は、データ処理システムのネイティブオペコードに直接的に１対１で変換される命令をほとんどまたは全く含まない。高レベルプログラミング言語の実施例は、限定はしないが、Ｃ、Ｃ＋＋、ＳｙｓｔｅｍＣなどを含む。

【００７６】

ホストコード８０５が、Ｃコンパイラ８４０または他の高レベル言語コンパイラに与えられる。Ｃコンパイラ８４０は、Ａｐｐ．ｏ　８６０として図示されているホストコード８０５のオブジェクトコードバージョンを生成する。リンカ８８５が、ヘテロジニアスマルチプロセッサランタイムライブラリ８７５、ａｐｐ．ｏ　８６０を受信し、ホストアプリケーション８９４を生成する。ヘテロジニアスマルチプロセッサランタイムライブラリ

８７５は、対象プラットフォームと通信するために使用される共通の低レベルドライバを含むことができる。ホストアプリケーション８９４は、ランタイムヘテロジニアスマルチプロセッサシステムのＣＰＵによって実行される。

【００７７】

ヘテロジニアスマルチプロセッサ高レベル合成ブロック８９０が、カーネル８１５を受信し、ｋｅｒｎｅｌ．ｈｄｌ　８９２を生成する。ｋｅｒｎｅｌ．ｈｄｌ　８９２は、カーネル８１５のＲＴＬバージョンである。システムアセンブラ８５０が、ｋｅｒｎｅｌ．ｈｄｌ　８９２およびベースプラットフォーム記述８２５を受信する。一態様において、ベースプラットフォーム記述８２５は、実際のベースプラットフォームの諸態様を記述したメタデータファイルであってもよい。言及されているように、ベースプラットフォームは、プログラム可能ＩＣ　２７０の静的領域３３５内に実装される回路である。

【００７８】

ベースプラットフォーム記述８２５から、システムアセンブラ８５０は、たとえば、対象プラットフォーム、および、カーネル実装のために使用されるべきプログラム可能ＩＣの特定のタイプを決定する。たとえば、システムアセンブラ８５０は、ベースプラットフォームに関する実施態様詳細、ならびに、ホストによって対象プラットフォームおよびベースプラットフォームと通信するために必要とされる低レベルドライバを指定するディレクトリを識別することができる。識別されるディレクトリは、ベースプラットフォームの１つまたは複数のパッケージＩＰを含むことができる。システムアセンブラ８５０は、ベースプラットフォームをカーネルと結合するインターコネクトＩＰを含む、ベースプラットフォームのパッケージＩＰを取り出すことができる。インターコネクトＩＰは、たとえば、ｋｅｒｎｅｌ．ｈｄｌ　８９２をベースプラットフォームのパッケージＩＰと統合するか、または、組み込むために必要とされる様々な相互接続回路ブロックを指定することができる。システムアセンブラ８５０は、バイナリコンテナ２８０を生成する。システムアセンブラ８５０は、バイナリコンテナ２８０内に含まれるベースプラットフォームと一体化するカーネル８１５のハードウェア実施態様を指定する構成ビットストリームを生成することができる。

【００７９】

バイナリコンテナ２８０内に含まれる各構成ビットストリームは、たとえば、カーネル８１５、または、場合によってはｋｅｒｎｅｌ．ｈｄｌ　８９２から決定される１つまたは複数のコンピュートユニットを実装することができる。論じられているように、システム設計者は、所与のカーネルについて並列に実装されるべきコンピュートユニットの数を決定する。

【００８０】

システムアセンブラ８５０は、ユーザ嗜好に応じて、前述したようにバイナリコンテナ２８０内に、ｋｅｒｎｅｌ．ｈｄｌ　８９２、たとえば、ＲＴＬシミュレーションのためのカーネル８１５のＲＴＬバージョン、および／または、シミュレーションのためのカーネル８１５の実行ファイル、たとえば、オブジェクトコードバージョンを含めることができる。システムアセンブラ８５０はまた、バイナリコンテナ２８０内にサポートデータ（図示せず）をも含める。

【００８１】

一態様において、システムアセンブラ８５０は、カーネル８１５を、ベースプラットフォームと統合する。ベースプラットフォーム記述８２５およびｋｅｒｎｅｌ．ｈｄｌ　８９２において指定されている情報を有するシステムアセンブラ８５０は、たとえば、技術マッピング、配置、ルーティングなどのような、構成ビットストリームをもたらす機能を実施することによって、カーネル８１５をベースプラットフォームと統合することができる。構成ビットストリームは、ベースプラットフォームとカーネルの両方を指定する全構成ビットストリーム、または、カーネルのみを指定する部分構成ビットストリームであってもよい。いずれにせよ、システムアセンブラ８５０は、指定されているインターコネクトＩＰを使用して、ベースプラットフォームをカーネルと結合する。

【００８２】
また別の態様において、システムアセンブラ８５０は、構成ビットストリーム以外のファイルを含めるためのバイナリコンテナ２８０を生成することができる。たとえば、言及されているように、カーネル８１５は、カーネル８１５のプロセッサが実行可能な、たとえば、オブジェクトコードのバージョンを生成するヘテロジニアスマルチプロセッサコンパイラに提供することができる。カーネル８１５の実行可能バージョン、たとえば、カーネル８１５のハードウェア実施態様の実行可能モデルは、システムアセンブラ８５０に提供することができる。システムアセンブラ８５０は、構成ビットストリームの代わりに、カーネル８１５の実行可能バージョンをバイナリコンテナ２８０内に含めることができる。別の実施例において、システムアセンブラ８５０は、構成ビットストリームの代わりに、ｋｅｒｎｅｌ．ｈｄｌ　８９２をバイナリコンテナ２８０内に含めることができる。
【００８３】
図９は、ＩＣ２７０内での実装のために、ヘテロジニアスマルチプロセッサコンピューティング言語において指定されるカーネルを処理する例示的な方法９００を示す流れ図である。一実施例において、ヘテロジニアスマルチプロセッサコンピューティング言語は、ＯｐｅｎＣＬであってもよい。方法９００は、図２を参照して説明されているシステムのような、コンパイル時システムによって実施され得る。一態様において、方法９００は、ＯｐｅｎＣＬ　Ｃ、Ｃ、Ｃ＋＋、別の高レベルプログラミング言語、または、本開示内で言及されている言語のうちの１つの派生形態および／もしくは変形形態において最初に指定されるカーネルのＲＴＬ記述の生成中に実施される様々な動作を示す。
【００８４】
ブロック９０５において、システムは、カーネルのメモリアクセスを識別およびマッピングする。ヘテロジニアスマルチプロセッサグローバルメモリは、マスタメモリバスにマッピングすることができる。たとえば、ＯｐｅｎＣＬグローバルメモリを、ＡＸＩマスタメモリバスにマッピングすることができる。カーネルパラメータは、スレーブ制御バスにマッピングすることができる。たとえば、カーネルパラメータは、ＡＸＩスレーブ制御バスにマッピングすることができる。
【００８５】
ブロック９１０において、システムは、カーネルによって利用されるパラメータを識別し、ＩＣにおけるカーネルのハードウェア実装のためにメモリマップ内にパラメータを含める。ブロック９１５において、システムは、変数を、カーネルのプライベートメモリとしてのＩＣのメモリ構造に相関付ける。ブロック９２０において、システムは、ローカルメモリ命令を、カーネルのローカルメモリとしてのＩＣのメモリ構造に相関付ける。
【００８６】
ブロック９２５において、システムは、カーネルの制御フローグラフを生成する。一態様において、システムは、カーネルをＬＬＶＭ中間表現（ＩＲ）フォーマットに変換する。ＬＬＶＭ　ＩＲフォーマットから、システムは、内部のデータフローを識別することによって、制御フローグラフを生成する。ブロック９３０において、システムは制御フローグラフを使用してカーネルの並列領域を識別する。並列領域は、制御フローグラフにおいて分離することができる。たとえば、制御フローグラフ内の各並列領域について、領域は、領域に入る１つの制御エッジおよび領域を出る１つの制御エッジを有することになる。
【００８７】
ブロック９３５において、システムは、任意選択的に、各並列領域周りに「ｆｏｒ」ループを構築する。並列領域を識別子、各々を「ｆｏｒ」ループとして表現することによって、データ並列実施態様であるカーネルが、Ｃ、Ｃ＋＋などのような逐次高レベルプログラミング言語として表現されることが可能になる。ブロック９４０において、システムは、パイプライン処理を使用して回路記述を生成する。たとえば、システムは、並列領域を「ｆｏｒ」ループとして表現することによって、Ｃ、Ｃ＋＋などのような高レベルプログラミング言語が合成されることになるため、この領域を合成することができる。
【００８８】

図１０は、ヘテロジニアスマルチプロセッサシステムの例示的なメモリアーキテクチャ１０００を示すブロック図である。一実施例において、ヘテロジニアスマルチプロセッサシステムは、ＯｐｅｎＣＬシステムである。図示されているように、ホスト１００５が、ホストメモリ１０１０を含む。ホスト１００５は、プロセッサ２０５として実装されてもよく、一方で、ホストメモリ１０１０は、メモリ要素２１０として実装されてもよい。ホスト１００５は、対象プラットフォーム２６０ならびにグローバルメモリおよびコンスタントメモリ１０１５に結合される。論じられているように、グローバルメモリおよびコンスタントメモリ１０１５へのアクセスは、メモリコントローラ（図示せず）によって可能にすることができる。グローバルメモリおよびコンスタントメモリ１０１５は、ＲＡＭ３４５として実装されてもよく、メモリコントローラがＩＣ　２７０内に実装される。しかしながら、メモリコントローラは、ＩＣ　２７０の外部にある対象プラットフォーム２６０上にあるが、ＩＣ　２７０と通信するように構成されているメモリコントローラとして実装されてもよいことが諒解されるべきである。

【００８９】

ＩＣ　２７０は、コンピュートユニット１０２０および１０２５を含む。２つのコンピュートユニットがＩＣ　２７０内に図示されているが、ＩＣ　２７０は、２つよりも少ないコンピュートユニットまたは２つよりも多いコンピュートユニットを含んでもよいことが諒解されるべきである。さらに、ＩＣ　２７０内に実装されている特定のコンピュートユニットおよびコンピュートユニットの特定の数は、ランタイム中に変化してもよい。コンピュートユニット１０２０および１０２５は、カーネル領域３４０の一部分として実装される。説明のために、静的領域３３５を図示してはいない。

【００９０】

図示されているように、コンピュートユニット１０２０は、ローカルメモリ１０３０と、処理要素１０４０および１０４５と、プライベートメモリ１０６０および１０６５とを含む。ローカルメモリ１０３０は、処理要素１０４０および１０４５によって共有される。処理ユニット１０４０および１０４５の各々は、プライベートメモリ１０６０および１０６５のうちの、共有されない個々のメモリに結合される。コンピュートユニット１０２５は、ローカルメモリ１０３５と、処理要素１０５０および１０５５と、プライベートメモリ１０７０および１０７５とを含む。ローカルメモリ１０３５は、処理要素１０５０および１０５５によって共有される。処理ユニット１０５０および１０５５の各々は、プライベートメモリ１０７０および１０７５のうちの、共有されない個々のメモリに結合される。コンピュートユニット１０２０および１０２５は両方とも、グローバルメモリおよびコンスタントメモリ１０２０にアクセスすることができる。

【００９１】

１つの例示的な実施態様において、ホストメモリ１０１０ならびにグローバルメモリおよびコンスタントメモリ１０２０は、対象プラットフォーム上のＲＡＭ、ホストＲＡＭ、および／またはホストの１つもしくは複数のキャッシュメモリを使用して実装されてもよい。ローカルメモリ１０３０および１０３５は、たとえば、１つまたは複数のＢＲＡＭ１０３を使用してＩＣ　２７０内に実装されてもよい。プライベートメモリ１０６０、１０６５、１０７０、および１０７５は、ＣＬＢ　１０２内に含まれているルックアップテーブルＲＡＭを使用して実装されてもよい。

【００９２】

ＩＣ　２７０のメモリ構造の、図１０のメモリアーキテクチャ１０００のメモリへの割り当ては、例示のみを目的としてなされている。合成中、ＩＣ　２７０のメモリ構造の可用性および必要とされるメモリの量に応じて、ＩＣ　２７０の１つまたは複数の他のメモリ構造が、プライベートメモリおよび／またはローカルメモリを実装するために使用されてもよい。

【００９３】

図１１は、ヘテロジニアスマルチプロセッサコンピューティング言語において指定されるカーネルの例示的な処理を示すブロック図である。図１１はＯｐｅｎＣＬ実施例を示し

ているが、カーネルは、ＯｐｅｎＣＬ以外の高レベルプログラミング言語で指定されてもよいこと、および、本開示内で記載されている本発明の構成は、与えられている実施例に限定されるようには意図されていないことが諒解されるべきである。一態様において、図１１は、図９のブロック９０５において実施される処理を示す。図示されているように、システムは、カーネル内で「ｇｌｏｂａｌ　ｉｎｔ」命令を識別する。「ｇｌｏｂａｌ　ｉｎｔ」命令は、ホストからカーネルに渡される特定のバッファを示す。システムは、メモリアクセスを、インターコネクト上のトランザクションとしてマッピングする。

【００９４】

別の態様において、図１１は、図９のブロック９１０において実施される処理を示す。図１１の実施例において、システムは、「ｇｅｔ＿ｌｏｃａｌ＿ｉｄ（０）；」関数によって示されているように識別子（ｉｄ）が使用されることを決定する。ホストからカーネルへと渡される、たとえば、ポインタが、カーネル内に実装されるレジスタマップ内で指定される。ｉｄのようなデータは、ホストによって、たとえば、ランタイム中にホスト内で実行しているホストアプリケーションによって、カーネルに書き込まれる。たとえば、ホストは、ｉｄのような任意の必要なデータを、カーネル回路６１０のレジスタマップの適切なレジスタに書き込むことができる。

【００９５】

システムはさらに、カーネルのプログラムコードの分析から、カーネルによって使用される任意の黙示的なパラメータを識別する。ホストからカーネルに与えられる必要があり得る黙示的なパラメータの例は、限定はしないが、ＮＤ範囲のサイズ、ワークグループのサイズなどを含む。いくつかの事例において、黙示的なパラメータは、ホストとカーネルとの間のインターフェースを通じて渡されなくてもよい。しかしながら、そのようなパラメータは、レジスタマップを通じて渡されてもよい。

【００９６】

図１２は、ヘテロジニアスマルチプロセッサコンピューティング言語において指定されるカーネルの例示的な処理を示すブロック図である。図１２はＯｐｅｎＣＬ実施例を示しているが、カーネルは、ＯｐｅｎＣＬ以外の高レベルプログラミング言語で指定されてもよいこと、および、本開示内で記載されている本発明の構成は、与えられている実施例に限定されるようには意図されていないことが諒解されるべきである。一態様において、図１２は、図９のブロック９２０および９２５において実施される処理を示す。図１２の実施例において、ブロック９２０を参照すると、「ｉｄ」のような変数が、カーネルのプライベートメモリ１０６０を実装するメモリ構造に相関付けられる。プライベートメモリの例は、パイプラインレジスタ、小型アレイ、ＢＲＡＭ、ルックアップテーブルＲＡＭなどを含み得る。ブロック９２５を参照すると、システムは、各「ｌｏｃａｌ　ｉｎｔ」メモリ命令を、カーネル内のＢＲＡＭのようなローカルメモリ１０３０と相関付ける。

【００９７】

図１３としてまとめて参照される図１３－１および図１３－２は、ヘテロジニアスマルチプロセッサコンピューティング言語において指定されるカーネルの処理を示す。一態様において、図１３は、図９のブロック９３０および９３５において実施される処理を示す。図１３－１を参照すると、カーネル１３００の例示的なＯｐｅｎＣＬ　Ｃソースコードが図示されている。図１３はＯｐｅｎＣＬ実施例を示しているが、カーネルは、ＯｐｅｎＣＬ以外の高レベルプログラミング言語で指定されてもよいこと、および、本開示内で記載されている本発明の構成は、与えられている実施例に限定されるようには意図されていないことが諒解されるべきである。

【００９８】

システムは、カーネル１３００の並列領域を、領域１３０５、１３１０、および１３１５として識別する。ブロック９３０における並列性の認識の一部分として、システムは、並列性を制御する特定の命令および／または構文を認識することができる。システムは、カーネル１３００内の「バリア」命令のインスタンスを識別することができる。「バリア」命令は、たとえば、任意のワークアイテムがバリア命令を超えて進み得る前に、すべて

のワークアイテムがバイラ命令に達しなければならないことを示す。バリア命令は、メモリフェンスまたは動悸メカニズムとして使用することができる。システムは、「ａｓｙｎｃ＿ｗｏｒｋ＿ｇｒｏｕｐ＿ｃｏｐｙ（）」命令（本明細書においては「ａｓｙｎｃ」ともいう）のインスタンスを識別することができる。「ａｓｙｎｃ」命令は、すべてのワークアイテムが、同じ引数を有するコピーに達しなければならないと指定する。したがって、一態様において、システムは、カーネル１３００内の並列性を制御する命令を識別することによって、カーネル１３００の並列領域を認識する。

【００９９】

ヘテロジニアスマルチプロセッサ実行およびメモリモデルは、領域１３０５、１３１０、および１３１５の各々が、完全に並列に、完全に逐次的に、または、様々な組み合わせで実施され得ることを保証する。観察されなければならない直列化は、並列性に直に影響を与える命令および／または構文、たとえば、バリアおよび／またはａｓｙｎｃ命令によって発生する。

【０１００】

図１３－２は、カーネル１３００に関するブロック９２５のデータフローグラフ生成を示す。並列領域１３０５、１３１０、および１３１５が示されている。領域１３０５内に含まれている「ｆｏｒ」部分の終端または戻りに対応する第４の並列領域１３０５－１が含まれている。

【０１０１】

図１４は、ヘテロジニアスマルチプロセッサコンピューティング言語において指定されるカーネルの例示的な処理を示すブロック図である。一実施例において、ヘテロジニアスマルチプロセッサコンピューティング言語は、ＯｐｅｎＣＬであってもよい。一態様において、図１４は、図９のブロック９３５および９４０において実施される処理を示す。図１４の実施例において、グレースケール変換に対応する領域１３１０の処理が示されている。各ループ反復が、１つのワークアイテムを処理する。ループ全体が１つのワークグループを処理する。ループは、パイプラインとして実施することができ、クロックサイクルごとに新たなワークアイテムがパイプラインに導入される。図示されているように、システムは、並列領域１３１０周りに「ｆｏｒ」ループ構文を作成する。図示されているようなパイプラインを使用して回路記述が生成され、ワークアイテムの各列がカーネルのパイプライン段に対応する。ワークアイテムの各行がサイクルに対応する。

【０１０２】

以下は、カーネルの例示的なレジスタマップを示す。

【０１０３】

【数１－１】

```
// 0x00 : Control signals
// bit 0 - ap_start (Read/Write/COH)
// bit 1 - ap_done (Read/COR)
// bit 2 - ap_idle (Read)
// bit 3 - ap_ready (Read)
// bit 7 - auto_restart (Read/Write)
// others - reserved
// 0x04 : Global Interrupt Enable Register
// bit 0 - Global Interrupt Enable (Read/Write)
// others - reserved
// 0x08 : IP Interrupt Enable Register (Read/Write)
// bit 0 - Channel 0 (ap_done)
// bit 1 - Channel 1 (ap_ready)
// others - reserved
// 0x0c : IP Interrupt Status Register (Read/TOW)
// bit 0 - Channel 0 (ap_done)
// bit 1 - Channel 1 (ap_ready)
// others - reserved
// 0x10 : Data signal of group_id_x
// bit 31~0 - group_id_x[31:0] (Read/Write)
// 0x14 : reserved
// 0x18 : Data signal of group_id_y
// bit 31~0 - group_id_y[31:0] (Read/Write)
// 0x1c : reserved
// 0x20 : Data signal of group_id_z
// bit 31~0 - group_id_z[31:0] (Read/Write)
// 0x24 : reserved
// 0x28 : Data signal of global_offset_x
// bit 31~0 - global_offset_x[31:0] (Read/Write)
// 0x2c : reserved
// 0x30 : Data signal of global_offset_y
```

【０１０４】

【数１－２】

```
        // bit 31~0 - global_offset_y[31:0] (Read/Write)
        // 0x34 : reserved
        // 0x38 : Data signal of global_offset_z
        // bit 31~0 - global_offset_z[31:0] (Read/Write)
        // 0x3c : reserved
        // 0x40 : Data signal of matrix
        // bit 31~0 - matrix[31:0] (Read/Write)
        // 0x44 : reserved
        // 0x48 : Data signal of maxIndex
        // bit 31~0 - maxIndex[31:0] (Read/Write)
        // 0x4c : reserved
        // 0x50 : Data signal of s1
        // bit 31~0 - s1[31:0] (Read/Write)
        // 0x54 : reserved
        // 0x58 : Data signal of s2
        // bit 31~0 - s2[31:0] (Read/Write)
        // 0x5c : reserved
        // (SC = Self Clear, COR = Clear on Read, TOW = Toggle on
 Write, COH = Clear on Handshake)
```

【０１０５】

図８を参照して説明されているようなヘテロジニアスマルチプロセッサＨＬＳ　８９０が、ＲＴＬにコンパイルされる各カーネルの上記で示されているようなカスタムレジスタマップを生成する。ホストは、対象プラットフォーム上に位置するデバイスメモリ内のバッファ、たとえば、ＯｐｅｎＣＬバッファのアドレス、カーネルに対するスカラー引数、および、カーネルを制御するための制御信号を渡すためにレジスタマップを使用することができる。レジスタマップはまた、ホストによって、ＯｐｅｎＣＬ仕様によって必要とされる場合に、グループｉｄおよびカーネルに対するグループオフセットを渡すために使用することもできる。一態様において、レジスタマップは、生成されるバイナリコンテナ内に含まれてもよい。たとえば、レジスタマップは、以前に説明されているバイナリコンテナに組み込まれるサポートデータの一部分であってもよい。

【０１０６】

以下は、例示的なプラットフォームメタデータファイルを示す。

【０１０７】

【数２－１】

```
        <platform name="vc690-admpcie7v31slot" ipiboard=""
 cfplatform="">
            <description>Alphadata ADM-PCIE-7V3 Partial Reconfiguration
 Single
```

【０１０８】

【数２－２】

```
        DIMM</description>
        <device name="cpu0" type="2">
          <core name="cpu0" type="cpu" numComputeUnits="1"/>
        </device>
        <device name="fpga0" type="8"
fpgaDevice="virtex:xc7vx690t:ffg1157:-2">
          <core name="OCL_REGION_0" type="clc_region"
clockFreq="100MHz"
          numComputeUnits="10">
            <port name="M_AXI_GMEM0" portType="addressable"
mode="master"
            base="0x00000000" range="0x40000000"
dataWidth="512"/>
            <port name="S_AXI_CONTROL0" portType="addressable"
            mode="slave" base="0x0" range="0x00010000"
dataWidth="32"/>
          </core>
        </device>
      </platform>
```

【０１０９】

一態様において、上記で示されているプラットフォームメタデータファイルは、図８を参照して説明されており、システムリンカ８３０に与えられるベースプラットフォーム記述８２５の一実施態様である。図示されているように、プラットフォームメタデータファイルは、プログラム可能ＩＣが結合される対象プラットフォームまたは基板のタイプを指定する。さらに、プラットフォームメタデータファイルは、基板上のプログラム可能ＩＣの特定の特徴、たとえば、モデルおよび／またはタイプ、ならびに、特定の領域、たとえば、静的領域のクロック周波数を示す。リンカ８３０は、プラットフォームメタデータファイルから対象プラットフォームを識別し、プラットフォームメタデータファイルにおいて指定される、対象プラットフォームのために命名されているディレクトリ構造にアクセスすることができる。

【０１１０】

図１５は、例示的なディレクトリ構造１５００である。示されている最上位ディレクトリは、システムリンカ８３０によってプラットフォームメタデータファイルから読み出すことができる対象プラットフォームと同じ名前を使用する。この実施例において、最上位ディレクトリは、「Ｂｏａｒｄ　Ｎａｍｅ」と呼ばれる。しかしながら、上記で与えられている例示的なプラットフォームメタデータファイルを参照すると、最上位ディレクトリは、「ＶＣ６９０」またはその派生形態として指定され得る。いずれにせよ、システムリンカ８３０は、図１５のディレクトリ構造を使用してプラットフォームＦＰＧＡ　８４５を取得する。説明のため、ディレクトリは図１５において太字にされている。たとえば、「Ｂｏａｒｄ　Ｎａｍｅ」、「ｄｒｉｖｅｒ」、「ｉｐｉ」、および「ｌｏｃａｌ＿ｌｉｂ」がディレクトリである。リストされている残りの項目は、ファイルおよび／またはパッケージである。

【０１１１】
システムは、任意の新たに追加される対象プラットフォームを自動的に配置することができ、ファイルは正確にパッケージされ、システムの指定されているプラットフォームディレクトリに追加される。図示されている実施例において、「ｄｒｉｖｅｒ＿ｆｉｌｅ．ｓｏ」は、ホストによってバスを介して対象プラットフォームと通信するために使用される低レベルドライバである。図示されているように、ｄｒｉｖｅｒ＿ｆｉｌｅ．ｓｏは、「ｄｒｉｖｅｒ」ディレクトリ内に位置する。図１５の実施例において「ｐｌａｔｆｏｒｍ．ｘｍｌ」として参照されるプラットフォームメタデータファイルが、ルートディレクトリに配置される。図３、図４、および図５を参照して説明されているようなＩＣの静的領域において使用される任意のパッケージＩＰを、「ｌｏｃａｌ＿ｌｉｂ」ディレクトリ内に記憶することができる。「ｂｐ．ｔｃｌ」と呼ばれるベースプラットフォームブロック図ＴＣＬファイル、ならびに、静的領域回路設計に対する任意の最上位設計制約ファイル、たとえば、タイミング制約および／または物理制約が、「ｉｐｉ」ディレクトリ内に含まれる。

【０１１２】
図１５において「ｄｒｉｖｅｒ＿ｆｉｌｅ．ｓｏ」として図示されている共通の低レベルドライバは、複数の関数を有するＡＰＩを含むことができる。共通の低レベルドライバＡＰＩ（以降「ドライバＡＰＩ」という）は、ホスト内で実行しているヘテロジニアスマルチプロセッサランタイムプログラムが、対象プラットフォームと通信することを可能にする。ドライバＡＰＩは、たとえば、カーネルの制御ポートを通じてプログラム可能ＩＣ内で実装されているものとして、バッファを割り当ておよび／または割り当て解除し、バッファをホストメモリから対象プラットフォームメモリへと移行し、対象プラットフォームメモリをホストメモリへと移行し、カーネルと通信して、プログラム可能ＩＣへとダウンロードされる構成ビットストリームをサポートする。

【０１１３】
ドライバＡＰＩはまた、アドレス空間をもサポートする。アドレス空間は、対象プラットフォームの周辺機器にアクセスするために使用することができる。対象プラットフォーム上の各周辺機器は、たとえば、アドレス空間のそれ自体のメモリマッピングされた範囲を有することができる。対象プラットフォームは任意選択的に、対象プラットフォームのすべての周辺機器に対処するために使用することができるフラットメモリ空間を有することができる。

【０１１４】
Ｔｈｅ ｄｒｉｖｅｒ＿ｆｉｌｅ．ｓｏは、対象プラットフォーム上で読み出しまたは書き込みすることができる、バッファ、たとえばＤＭＡバッファの最小サイズのような、様々な量をサポートすることができる。さらに、「ｅｎｕｍｓ」として参照される、１つまたは複数の列挙アドレス空間がサポートされ得る。メモリ動作は、フラットアドレス指定または相対アドレス指定を使用することができる。例示的なｅｎｕｍｓは、限定はしないが、ＸＣＬ＿ＡＤＤＲ＿ＳＰＡＣＥ＿ＤＥＶＩＣＥ＿ＦＬＡＴ、ＸＣＬ＿ＡＤＤＲ＿ＳＰＡＣＥ＿ＤＥＶＩＣＥ＿ＲＡＭ、ＸＣＬ＿ＡＤＤＲ＿ＫＥＲＮＥＬ＿ＣＴＲＬ、およびＸＣＬ＿ＡＤＤＲ＿ＳＰＡＣＥ＿ＭＡＸを含み得る。

【０１１５】
ドライバＡＰＩは、限定はしないが、以下を含む複数のデバイスアクセス動作をサポートする。

【０１１６】
・ｘｃｌＤｅｖｉｃｅＨａｎｄｌｅ ｘｃｌＯｐｅｎ（ｃｏｎｓｔ ｃｈａｒ ＊ｄｅｖｉｃｅＮａｍｅ）
・ｖｏｉｄ ｘｃｌＣｌｏｓｅ（ｘｃｌＤｅｖｉｃｅＨａｎｄｌｅ ｈａｎｄｌｅ）
・ｉｎｔ ｘｃｌＧｅｔＤｅｖｉｃｅＩｎｆｏ（ｘｃｌＤｅｖｉｃｅＨａｎｄｌｅ ｈａｎｄｌｅ，ｘｃｌＤｅｖｉｃｅＩｎｆｏ ＊ｉｎｆｏ）（ｘｃｌＤｅｖｉｃｅＨａｎｄｌｅ ｈａｎｄｌｅ）

ドライバＡＰＩは、動作「ｉｎｔ　ｘｃｌＬｏａｄＢｉｔｓｔｒｅａｍ（ｘｃｌＤｅｖｉｃｅＨａｎｄｌｅ　ｈａｎｄｌｅ，　ｃｏｎｓｔ　ｃｈａｒ　＊ｆｉｌｅＮａｍｅ）」によって、構成ビットストリームロード動作をサポートする。そのため、ホストは、ランタイム中に必要に応じて１つまたは複数の異なるカーネルをハードウェア内に実装するために、全体であるかまたは部分的であるかを問わず、構成ビットストリームのＩＣへのロードを開始することができる。

【０１１７】

ドライバＡＰＩは、対象プラットフォームのメモリを管理するための様々な動作を可能にする。対象プラットフォームの供給元は、たとえば、以下のＡＰＩによってメモリ管理を可能にすることが必要とされる。

【０１１８】

・ｕｉｎｔ６４＿ｔ　ｘｃｌＡｌｌｏｃＤｅｖｉｃｅＢｕｆｆｅｒ（ｘｃｌＤｅｖｉｃｅＨａｎｄｌｅ　ｈａｎｄｌｅ，　ｓｉｚｅ＿ｔ　ｓｉｚｅ）

動作「ｘｃｌＡｌｌｏｃＤｅｖｉｃｅＢｕｆｆｅｒ」は、指定サイズのバッファを対象プラットフォーム上に割り当て、対象プラットフォームＲＡＭにおける割り当てられたバッファのオフセットを、返り値として返す。オフセットは、バッファハンドルとして機能する。ＯｐｅｎＣＬランタイムはその後、返されたハンドルをＯｐｅｎＣＬカーネルに渡す。ＯｐｅｎＣＬカーネルは、返されたハンドルを使用して、対象プラットフォームＲＡＭ内の割り当てられたバッファに対してバスマスタ読み出しおよび／または書き込み動作を実施する。ホストは対象プラットフォームＲＡＭに対して直接書き込まない。空きブロックが残っていない場合、この関数は－１を返すことになる。

【０１１９】

・ｖｏｉｄ　ｘｃｌＦｒｅｅＤｅｖｉｃｅＢｕｆｆｅｒ（ｘｃｌＤｅｖｉｃｅＨａｎｄｌｅ　ｈａｎｄｌｅ，　ｕｉｎｔ６４＿ｔ　ｂｕｆ）

動作「ｘｃｌＦｒｅｅＤｅｖｉｃｅＢｕｆｆｅｒ」は、ｘｃｌＡｌｌｏｃＤｅｖｉｃｅＢｕｆｆｅｒによって以前に割り当てられているメモリを解放する。解放されたメモリは、ｘｃｌＡｌｌｏｃＤｅｖｉｃｅＢｕｆｆｅｒに対する別の呼び出しのために後に再使用することができる。ｘｃｌＡｌｌｏｃＤｅｖｉｃｅＢｕｆｆｅｒによって以前に割り当てられていないバッファハンドルが渡される結果として、エラー状態が発生する。

【０１２０】

・ｓｉｚｅ＿ｔ　ｘｃｌＣｏｐｙＢｕｆｆｅｒＨｏｓｔ２Ｄｅｖｉｃｅ（ｘｃｌＤｅｖｉｃｅＨａｎｄｌｅ　ｈａｎｄｌｅ，　ｕｉｎｔ６４＿ｔ　ｄｅｓｔ，　ｃｏｎｓｔ　ｖｏｉｄ　＊ｓｒｃ，　ｓｉｚｅ＿ｔ　ｓｉｚｅ，　ｓｉｚｅ＿ｔ　ｓｅｅｋ）

動作「ｘｃｌＣｏｐｙＢｕｆｆｅｒＨｏｓｔ２Ｄｅｖｉｃｅ」は、ホストバッファの内容を、対象プラットフォーム上に常駐する宛先バッファへとコピーする。要素ｓｒｃは、ホストバッファポインタを参照し、ｄｅｓｔは、デバイスバッファハンドルを参照する。エラーが発生する結果として、ｘｃｌＡｌｌｏｃＤｅｖｉｃｅＢｕｆｆｅｒによって以前に割り当てられていないｄｅｓｔハンドルが渡されることになる。要素ｓｅｅｋは、ｄｅｓｔハンドルにおけるオフセットを指定する。ｓｉｚｅ＋ｓｅｅｋが以前に割り当てられているデバイスバッファのサイズよりも大きい場合、ｓｉｚｅが渡される結果として、エラーが発生する。与えられている実施例において、バッファを移行するためにＰＣＩｅＤＭＡが使用される。

【０１２１】

・ｓｉｚｅ＿ｔ　ｘｃｌＣｏｐｙＢｕｆｆｅｒＤｅｖｉｃｅ２Ｈｏｓｔ（ｘｃｌＤｅｖｉｃｅＨａｎｄｌｅ　ｈａｎｄｌｅ，　ｖｏｉｄ　＊ｄｅｓｔ，　ｕｉｎｔ６４＿ｔ　ｓｒｃ，　ｓｉｚｅ＿ｔ　ｓｉｚｅ，　ｓｉｚｅ＿ｔ　ｓｋｉｐ）

動作ｘｃｌＣｏｐｙＢｕｆｆｅｒＤｅｖｉｃｅ２Ｈｏｓｔは、対象プラットフォーム常駐バッファからホストバッファへと内容をコピーする。要素ｓｒｓは、デバイスバッファハンドルを参照し、要素ｄｅｓｔは、ホストバッファポインタを参照する。ｘｃｌＡｌｌｏｃＤｅｖｉｃｅＢｕｆｆｅｒによって以前に割り当てられていないｓｒｃハンドルが渡

される結果として、エラーが発生する。要素ｓｋｉｐは、ｓｒｃハンドルにおけるオフセットを指定する。ｓｉｚｅ＋ｓｋｉｐが以前に割り当てられているデバイスバッファのサイズよりも大きい場合、ｓｉｚｅが渡される結果として、エラーが発生する。与えられている実施例において、バッファを移行するためにＰＣＩｅ　ＤＭＡが使用される。

【０１２２】

・ｓｉｚｅ＿ｔ　ｘｃｌＷｒｉｔｅ（ｘｃｌＤｅｖｉｃｅＨａｎｄｌｅ　ｈａｎｄｌｅ，　ｘｃｌＡｄｄｒｅｓｓＳｐａｃｅ　ｓｐａｃｅ，　ｕｉｎｔ６４＿ｔ　ｏｆｆｓｅｔ，　ｃｏｎｓｔ　ｖｏｉｄ　＊ｈｏｓｔＢｕｆ，　ｓｉｚｅ＿ｔ　ｓｉｚｅ）

動作ｘｃｌＷｒｉｔｅは、ホストバッファｈｏｓｔＢｕｆ　の内容を、対象プラットフォームアドレスマップ内の特定の位置へとコピーする。ｈｏｓｔＢｕｆの内容は、対象プラットフォームの周辺機器をプログラムするために使用される。たとえば、ホスト内で実行しているＯｐｅｎＣＬランタイムは、この動作を使用して、引数をプログラム可能ＩＣ内のカーネルに送信する。オフセットは、アドレス空間に対するものである。

【０１２３】

・ｓｉｚｅ＿ｔ　ｘｃｌＲｅａｄ（ｘｃｌＤｅｖｉｃｅＨａｎｄｌｅ　ｈａｎｄｌｅ，　ｘｃｌＡｄｄｒｅｓｓＳｐａｃｅ　ｓｐａｃｅ，　ｕｉｎｔ６４＿ｔ　ｏｆｆｓｅｔ，　ｖｏｉｄ　＊ｈｏｓｔｂｕｆ，　ｓｉｚｅ＿ｔ　ｓｉｚｅ）

動作ｘｃｌＲｅａｄは、対象プラットフォームアドレスマップ内の特定の位置からホストバッファｈｏｓｔＢｕｆへとデータをコピーする。この動作は、対象プラットフォームの周辺機器の状態を読み出すために使用される。たとえば、ＯｐｅｎＣＬランタイムライブラリは、この動作を使用して、カーネルが動作を終了したか否かを判定する。オフセットは、アドレス空間に対するものである。

【０１２４】

ホストのオペレーティングシステムは、対象プラットフォームと通信するために、カーネルＤＭＡドライバを必要とする。一態様において、共通の低レベルドライバＡＰＩは、ＯｐｅｎＣＬランタイムをドライバの詳細から分離するために、カーネルＤＭＡドライバの上に重ねられ得る。ドライバは、マルチスレッドセーフであるべきである。ＯｐｅｎＣＬランタイムは、任意選択的に、デバイスに対して同時に読み出しおよび書き込みを行うために、２つ以上のスレッドを使用する。

【０１２５】

別の態様において、ドライバＡＰＩは、割り込み機能を含むことができる。たとえば、カーネルのレジスタマップは、カーネルによってフラグを記憶することができる１つまたは複数のメモリ位置を含み得る。レジスタマップの指定メモリ位置においてフラグが検出されることによって、静的領域が、ドライバの一部分として提供される機能を通じてホストに対する割り込みをトリガし得る。

【０１２６】

上記で示されている実施例はＯｐｅｎＣＬ実施態様向けに意図されているが、任意のヘテロジニアスマルチプロセッサコンピューティング言語が使用されてもよいこと、および、ＡＰＩの一部分として記載されている様々な動作はそれに応じて適合されてもよいことが諒解されるべきである。

【０１２７】

図１６は、カーネル実行の例示的な方法１６００を示す流れ図である。方法１６００は、カーネルがプログラム可能ＩＣ内に実装されており、対象プラットフォームがホストと通信可能にリンクされている状態において開始する。ホストは、たとえば、１つまたは複数のバイナリコンテナを含むことができ、または、１つまたは複数のバイナリコンテナにアクセスすることができる。ホストはバイナリコンテナにアクセスし、バイナリコンテナからの構成ビットストリームファイルを、ＩＣ、たとえば、ＲＡＭ　３４５に提供する。ホストは、ドライバＡＰＩの一部分として説明されている構成ビットストリームロード動作を開始することができ、それによって、ＩＣは、構成ビットストリームをロードすることになり、構成ビットストリームによって指定されるカーネルを実行することになる。論

じられているように、ホストは、ランタイム中の様々な時点において１つまたは複数の異なるカーネルをＩＣ内に実装するために、部分的であるかまたは全体であるかを問わず、１つまたは複数の異なる構成ビットストリームがロードされるようにすることができる。

【０１２８】

ブロック１６０５において、ホストアプリケーションが初期化される。ホストアプリケーションは、図８において示されているようなヘテロジニアスマルチプロセッサランタイムライブラリを含む。ブロック１６１０において、ホストアプリケーションが、バッファをホストメモリ内に割り当てる。ブロック１６１５において、ホストアプリケーションが、バッファ内容をホストメモリから対象プラットフォームメモリへと送信するための転送を開始する。

【０１２９】

ブロック１６２０において、ホストアプリケーションが、スレーブインターフェースを通じてカーネルに、動作を開始するようにシグナリングする。ブロック１６２５において、ホストアプリケーションが、任意選択的に、完了信号をモニタリングするために、対象プラットフォームのポーリングを開始する。ブロック１６３０において、カーネル、すなわち、カーネルのハードウェア実施態様が実行する、すなわち、動作し始める。カーネルは、対象プラットフォームメモリからデータをロードおよび記憶する。ブロック１６３５において、カーネルは、処理の終了に応答してメモリマップレジスタ内の状態を完了に変更するか、または、割り込みを生成する。ブロック１６４０において、ホストアプリケーションが、対象プラットフォームメモリの更新済みバッファ内容、すなわち結果を、ホストメモリへと転送する。ブロック１６４５において、たとえば、ホストまたはホストアプリケーション内で実行しているヘテロジニアスマルチプロセッサランタイムが、ホストメモリからバッファを読み出す。

【０１３０】

説明を目的として、本明細書において開示されている様々な本発明の概念を完全に理解できるよう、特定の用語体系が記載されている。しかしながら、本明細書において使用されている用語は、本発明の構成の特定の態様を説明することのみを目的としており、限定的であるようには意図されていない。

【０１３１】

本開示内で定義されているものとしては、用語「ａ」および「ａｎ」は、１つまたは２つ以上を意味する。用語「複数」は、本明細書における定義としては、２つまたは３つ以上を意味する。用語「別の」は、本明細書における定義としては、少なくとも第２以上のものを意味する。用語「結合されている」は、本明細書における定義としては、別途特記しない限り、いかなる介在する要素もない直接的なものであるか、または、１つもしくは複数の介在する要素による間接的なものであるかを問わず、接続されていることを意味する。２つの要素はまた、機械的に結合されるか、電気的に結合されるか、または、チャネル、経路、ネットワーク、もしくはシステムを通じて通信可能にリンクされ得る。

【０１３２】

本明細書における定義としては、用語「自動的に」は、ユーザが介入しないことを意味する。本明細書における定義としては、用語「ユーザ」は、人を意味する。用語「および／または」は、本明細書における定義としては、関連してリストされている項目のうちの１つまたは複数のあらゆる可能な組み合わせを意味する。用語「含む」および／または「含んでいる」は、本開示において使用されるとき、記述されている特徴、整数、ステップ、動作、要素、および／または構成要素が存在することを示すが、１つまたは複数の他の特徴、整数、ステップ、動作、要素、構成要素、および／またはそれらのグループが存在することまたは加わることを除外するものではない。用語「第１の」、「第２の」などは、本明細書において様々な要素を説明するために使用されるが、文脈が別途示唆していない限り、これらの用語は、１つの要素を別の要素から区別するために使用されているに過ぎないため、これらの要素はこれらの用語によって限定されるべきではない。

【０１３３】

本明細書における定義としては、用語「～場合」は文脈に応じて、「～とき」、「～すると」、「～と判定されるのに応答して」、「～が検出されるのに応答して」、「～と判定されるのに応じて」、または「～が検出されるのに応じて」を意味する。同様に、語句「～と判定される場合」または語句「［記述されている状態または事象が］検出される場合」は、本明細書における定義としては、文脈に応じて、「～と判定されると」、「～と判定されるのに応答して」、「～と判定されるのに応じて」、「［記載されている状態または事象が］検出されると」、「［記載されている状態または事象が］検出されるのに応答して」、または「［記載されている状態または事象が］検出されるのに応じて」、を意味する。

【０１３４】

本開示内では、同じ参照符号が、端子、信号線、ワイヤ、およびそれらの対応する信号を指すために使用されている。これに関連して、用語「信号」、「ワイヤ」、「接続」、「端子」、および「ピン」は、本開示内で、場合によっては交換可能に使用される。用語「信号」、「ワイヤ」などは、１つまたは複数の信号、たとえば、単一のワイヤを通じた単一のビットの伝達、または、複数の並列ワイヤを通じて複数の並列ビットの伝達を表すことができる。さらに、各ワイヤまたは信号は、場合によって、信号またはワイヤにより接続されている２つ以上の構成要素間の双方向性通信を表すことがある。

【０１３５】

本開示内で記載されている１つまたは複数の態様は、ハードウェアまたはハードウェアとソフトウェアとの組み合わせにおいて実現することができる。１つまたは複数の態様は、１つのシステム内で集中様式で、または、複数の異なる要素がいくつかの相互接続されたシステムにわたって分散される分散様式で実現されてもよい。本明細書に記載されている方法の少なくとも一部分を実行するような任意の種類のデータ処理システムまたは他の装置が適している。

【０１３６】

１つまたは複数の態様はさらに、本明細書に記載されている方法の実施を可能にするすべての特徴を含む、コンピュータプログラム製品内に組み込まれてもよい。コンピュータプログラム製品は、コンピュータ可読データ記憶媒体を含む。本明細書における定義としては、語句「コンピュータ可読記憶媒体」は、命令実行システム、装置、またはデバイスによって使用するための、または、それらに関連するプログラムコードを含むかまたは記憶している記憶媒体を意味する。本明細書における定義としては、「コンピュータ可読記憶媒体」は非一時的であり、そのため、一時的伝播信号自体ではない。コンピュータ可読記憶媒体の例は、限定はしないが、光学媒体、磁気媒体、磁気－光媒体、ＲＡＭのようなコンピュータメモリ、たとえば、ハードディスクなどの大容量記憶デバイスなどを含み得る。

【０１３７】

図面内の流れ図およびブロック図は、本明細書に開示されている本発明の構成の様々な態様によるシステム、方法およびコンピュータプログラム製品の可能な実施態様のアーキテクチャ、機能、および動作を示す。これに関連して、流れ図またはブロック図内の各ブロックは、指定される機能（複数可）を実施するための１つまたは複数の実行可能命令を含む、モジュール、セグメント、またはコード部分を表すことができる。ブロック図および／または流れ図の各ブロック、ならびに、ブロック図および／または流れ図内のブロックの組み合わせは、指定される機能もしくは動作を実施する特殊用途ハードウェアベースシステム、または、特殊目的ハードウェアとコンピュータ命令との組み合わせによって実施することができることも留意されたい。

【０１３８】

一態様において、流れ図内のブロックは、様々なブロック内の参照符号に対応する昇順で実施されてもよい。他の態様において、ブロックは、ブロック内の参照符号とは異なる順序、または、変動する順序で実施されてもよい。たとえば、連続して図示される２つ以上のブロックは、実質的に同時に実行されてもよい。他の事例において、２つ以上のブロ

ックは時として、含まれている機能に応じて逆順で実行されてもよい。また他の事例において、１つまたは複数のブロックは、様々な順序で実施されてもよく、その結果は、直に後続しないその後のまたは他のブロックにおいて利用される。

【０１３９】

用語「コンピュータプログラム」、「ソフトウェア」、「アプリケーション」、「コンピュータ使用可能プログラムコード」、「プログラムコード」、「実行可能コード」、それらの変化形および／または組み合わせは、本開示の文脈において、データ処理システムに、直接的に、または、ａ）別の言語、コード、もしくは表記への変換、ｂ）異なる材料形態での再現のいずれかもしくは両方の後に、特定の機能を実施させるように意図されている命令セットの任意の言語、コードまたは表記における任意の表現を意味する。たとえば、プログラムコードは、限定はしないが、サブルーチン、関数、手順、オブジェクトメソッド、オブジェクト実装、実行可能アプリケーション、アプレット、サーブレット、ソースコード、オブジェクトコード、共有ライブラリ／動的読み込みライブラリ、および／または、コンピュータシステム上で実行するように設計されている他の命令シーケンスを含んでもよい。

【０１４０】

したがって、本開示全体を通じて、「処理」または「計算」または「算出」または「決定」または「表示」などのような用語を利用した記述は、コンピュータシステムのレジスタおよび／またはメモリ内で物理（電子）量として表されているデータを操作して、コンピュータシステムメモリおよび／またはレジスタもしくは他のそのような情報記憶、送信もしくは表示デバイス内で同様に物理量として表される他のデータに変換するデータ処理システム、たとえば、コンピュータシステム、または同様の電子コンピューティングデバイスの動作およびプロセスを参照する。

【０１４１】

対応する構造、材料、動作、および添付の特許請求の範囲内のすべての手段またはステッププラスファンクション要素の対応する構造、材料、動作、および均等物は、明確に特許請求されているものとしての他の特許請求されている要素と組み合わさった機能を実施するための任意の構造、材料、または動作を含むように意図されている。

【０１４２】

方法は、プロセッサを使用して、ヘテロジニアスマルチプロセッサ設計の第１のカーネルのＲＴＬ記述を生成するステップと、ヘテロジニアスマルチプロセッサ設計のホストに対するインターフェースを提供するプログラム可能ＩＣ内の静的領域を提供するベースプラットフォーム回路設計と、第１のカーネルのＲＴＬ記述を統合するステップと、第１のカーネルのＲＴＬ記述から、プロセッサを使用して、第１のカーネルのハードウェア実施態様を指定する第１の構成ビットストリームおよび構成ビットストリームのサポートデータを生成するステップとを含む。方法はまた、第１の構成ビットストリームおよびサポートデータを、バイナリコンテナ内に含めるステップをも含む。

【０１４３】

一例において、ヘテロジニアスマルチプロセッサ設計は、ＯｐｅｎＣＬ設計であり、第１のカーネルは、ＯｐｅｎＣＬで指定される。

【０１４４】

一態様において、サポートデータは、プログラム可能ＩＣ内で実装されるものとしての、第１のカーネルのハードウェア実施態様の２次元位置を含む。

【０１４５】

方法は、ヘテロジニアスマルチプロセッサ設計のランタイム中にプログラム可能ＩＣ内で第１のカーネルのハードウェア実施態様のインスタンスを作成する、第１のカーネルの構成ビットストリームをロードするステップを含むことができる。

【０１４６】

方法はまた、ヘテロジニアスマルチプロセッサ設計のランタイム中にプログラム可能ＩＣ内で第１のカーネルのハードウェア実施態様の複数のインスタンスを作成する、第１の

カーネルの構成ビットストリームをロードするステップをも含むことができる。
【０１４７】
方法は、第２のバイナリコンテナ内に、ヘテロジニアスマルチプロセッサ設計の第２のカーネルのハードウェア実施態様を指定する第２の構成ビットストリームを含めるステップをさらに含むことができる。第２のカーネルのハードウェア実施態様の少なくとも１つのインスタンスは、プログラム可能ＩＣ内で作成することができる。
【０１４８】
一態様において、第１の構成ビットストリームを生成するステップは、カーネル回路を指定する部分構成ビットストリームとして第１の構成ビットストリームを生成するステップを含むことができる。別の態様において、第１の構成ビットストリームを生成するステップは、カーネル回路およびベースプラットフォーム回路を指定する全構成ビットストリームとして第１の構成ビットストリームを生成するステップを含むことができる。
【０１４９】
方法は、プロセッサを使用して、ヘテロジニアスマルチプロセッサ設計の第１のカーネルのＲＴＬ記述を生成するステップと、ヘテロジニアスマルチプロセッサ設計のホストに対するプログラム可能ＩＣ内の静的インターフェースを提供するベースプラットフォーム回路設計と、第１のカーネルのＲＴＬ記述を統合するステップと、第１のカーネルのＲＴＬ記述から、プロセッサを使用して、第１のカーネルのＲＴＬ記述のサポートデータを生成するステップとを含む。方法はまた、第１のカーネルのＲＴＬ記述およびサポートデータを、バイナリコンテナ内に含めるステップをも含む。
【０１５０】
一例において、ヘテロジニアスマルチプロセッサ設計は、ＯｐｅｎＣＬ設計であり、第１のカーネルは、ＯｐｅｎＣＬで指定される。
【０１５１】
方法は、カーネルの実行可能バージョンを生成するステップと、カーネルの実行可能バージョンをバイナリコンテナ内に含めるステップとを含むことができる。
【０１５２】
方法はまた、ホストが、ランタイム中に、バイナリコンテナからのＲＴＬ記述をＲＴＬシミュレータに提供するステップと、ＲＴＬシミュレータ内でカーネルのＲＴＬ記述をシミュレートするステップとを含むこともできる。
【０１５３】
方法は、第２のバイナリコンテナ内に、ヘテロジニアスマルチプロセッサ設計の第２のカーネルのＲＴＬ記述を含めるステップをさらに含むことができる。
【０１５４】
システムは、実行可能動作を開始するようにプログラムされているプロセッサを含むことができる。実行可能動作は、ヘテロジニアスマルチプロセッサ設計の第１のカーネルのＲＴＬ記述を生成するステップと、ヘテロジニアスマルチプロセッサ設計のホストに対するインターフェースを提供するプログラム可能ＩＣ内の静的領域を提供するベースプラットフォーム回路設計と、第１のカーネルのＲＴＬ記述を統合するステップと、第１のカーネルのＲＴＬ記述から、第１のカーネルのハードウェア実施態様を指定する第１の構成ビットストリームおよび構成ビットストリームのサポートデータを生成するステップとを含む。方法はまた、第１の構成ビットストリームおよびサポートデータを、バイナリコンテナ内に含めるステップをも含むことができる。
【０１５５】
一例において、ヘテロジニアスマルチプロセッサ設計は、ＯｐｅｎＣＬ設計であり、第１のカーネルは、ＯｐｅｎＣＬで指定される。
【０１５６】
サポートデータは、プログラム可能ＩＣ内で実装されるものとしての、第１のカーネルのハードウェア実施態様の２次元位置を含むことができる。
【０１５７】

実行可能動作は、ヘテロジニアスマルチプロセッサ設計のランタイム中にプログラム可能ＩＣ内で第１のカーネルのハードウェア実施態様のインスタンスを作成する、第１のカーネルの構成ビットストリームをロードするステップを含むことができる。

【０１５８】

実行可能動作はまた、ヘテロジニアスマルチプロセッサ設計のランタイム中にプログラム可能ＩＣ内で第１のカーネルのハードウェア実施態様の複数のインスタンスを作成する、第１のカーネルの構成ビットストリームをロードするステップをも含むことができる。

【０１５９】

実行可能動作は、第２のバイナリコンテナ内に、ヘテロジニアスマルチプロセッサ設計の第２のカーネルのハードウェア実施態様を指定する第２の構成ビットストリームを含めるステップをさらに含むことができる。実行可能動作は、第２のカーネルのハードウェア実施態様の少なくとも１つのインスタンスを、プログラム可能ＩＣ内で作成するステップを含むことができる。

【０１６０】

一態様において、第１の構成ビットストリームを生成するステップは、カーネル回路を指定する部分構成ビットストリームとして第１の構成ビットストリームを生成するステップを含むことができる。別の態様において、第１の構成ビットストリームを生成するステップは、カーネル回路およびベースプラットフォーム回路を指定する全構成ビットストリームとして第１の構成ビットストリームを生成するステップを含むことができる。

【０１６１】

別の実施例において、ＩＣは、静的であり、ＩＣとホストプロセッサとの間のインターフェースを提供する第１の領域を含む。第１の領域は、第１のマスタインターフェースを有する第１の相互接続回路ブロックと、第１のスレーブインターフェースを有する第２の相互接続回路ブロックとを含む。ＩＣは、第１の領域に結合されている第２の領域を含む。第２の領域は、ヘテロジニアスマルチプロセッサ設計のカーネルを実装し、第１の相互接続回路ブロックの第１のマスタインターフェースに結合されており、ホストプロセッサからコマンドを受信するように構成されているスレーブインターフェースを含む。第２の領域はまた、第２の相互接続回路ブロックの第１のスレーブインターフェースに結合されているマスタインターフェースをも含み、第２の領域のマスタインターフェースは、メモリコントローラのマスタである。

【０１６２】

一実施例において、ヘテロジニアスマルチプロセッサ設計は、ＯｐｅｎＣＬ設計である。

【０１６３】

一態様において、第２の領域は、ランタイム中に、ホストプロセッサの制御下で異なるカーネルを実装するように、動的に再構成可能であり得る。別の態様において、第２の領域は、ランタイム中に、第１の領域を損傷を受けないままにしながら、ホストプロセッサの制御下で異なるカーネルを実装するように、動的に再構成可能であり得る。

【０１６４】

第１の領域は、バスエンドポイントと、バスエンドポイントに結合されているＤＭＡコントローラとを含むことができる。第１の領域は、第１の相互接続回路ブロックのスレーブインターフェースに結合されているマスタインターフェースを含むことができる。第１の相互接続回路ブロックは、第２のマスタインターフェースを含むことができる。第２の相互接続回路ブロックは、第１の相互接続回路ブロックの第２のマスタインターフェースに結合されている第２のスレーブインターフェースを含むことができる。

【０１６５】

第１の領域はまた、メモリコントローラをも含むことができる。メモリコントローラは、第２の相互接続回路ブロックのマスタインターフェースに結合されているスレーブインターフェースを含むことができる。

【０１６６】

第１の相互接続回路ブロックおよび第２の相互接続回路ブロックは、ＡＸＩ相互接続回路ブロックとして実装されてもよい。

【０１６７】

ＩＣはまた、ヘテロジニアスマルチプロセッサ設計のホストプログラムコードによってプログラムされている、ホストプロセッサにも結合することができる。

【０１６８】

ＤＭＡコントローラは、メモリコントローラに対するマスタとして構成することができる。

【０１６９】

第１の領域は、第１の相互接続回路ブロックを通じて第２の領域にクロック信号およびリセット信号を提供するように構成することができる。

【０１７０】

第２の領域は、第１の相互接続回路ブロックに結合されているメモリマップドレジスタを含むことができる。

【０１７１】

一態様において、第２の領域は、第１の相互接続回路ブロックの第１のマスタインターフェースに結合されている第２の領域のスレーブインターフェース、および、第１のカーネル回路ブロック６１０－１の入力に結合されているマスタインターフェースを有する第３の相互接続回路ブロックを含む。第２の領域はまた、第１のカーネル回路ブロック６１０－１の出力に結合されているスレーブインターフェース、および、第２のインターフェース回路ブロックの第１のスレーブインターフェースに結合されているマスタインターフェースを有する第４の相互接続回路ブロックをも含むことができる。

【０１７２】

第２の領域はまた、第３の相互接続回路ブロックのマスタインターフェースに結合されている入力、および、第４の相互接続回路ブロックのスレーブインターフェースに結合されている出力を有する第２のカーネル回路ブロックをも含むことができる。

【０１７３】

別の実施例において、方法は、ＩＣとホストプロセッサとの間のインターフェースを実装する、ＩＣ内の静的領域である第１の領域を提供するステップと、第１の領域内に、第１のマスタインターフェースを有する第１の相互接続回路ブロック、および、第１のスレーブインターフェースを有する第２の相互接続回路ブロックを含めるステップと、第１の領域に結合されている第２の領域を提供するステップとを含む。方法はまた、第２の領域内に、ヘテロジニアスマルチプロセッサ設計のカーネルを実装するステップと、第２の領域内に、第１の相互接続回路ブロックの第１のマスタインターフェースに結合されているスレーブインターフェースを含めるステップとを含むこともできる。カーネルは、ホストプロセッサからコマンドを受信するように構成されている。方法は、第２の領域内に、第２の相互接続回路ブロックの第１のスレーブインターフェースに結合されているマスタインターフェースを含めるステップをさらに含み、第２の領域のマスタインターフェースは、メモリコントローラのマスタである。

【０１７４】

一実施例において、ヘテロジニアスマルチプロセッサ設計は、ＯｐｅｎＣＬ設計である。

【０１７５】

一態様において、方法は、ＩＣのランタイム中に、ホストプロセッサの制御下で異なるカーネルを実装するように、第２の領域を動的に再構成するステップを含むことができる。別の態様において、方法は、ＩＣのランタイム中に、第１の領域を損傷を受けないままにしながら、ホストプロセッサの制御下で異なるカーネルを実装するように、第２の領域を動的に再構成するステップを含むことができる。

【０１７６】

方法は、第１の領域内に、バスエンドポイントを設けるステップと、第１の領域内に、

バスエンドポイントに結合されているＤＭＡコントローラを設け、第１の相互接続回路ブロックのスレーブインターフェースに結合されているマスタインターフェースを含めるステップとを含むことができる。第１の相互接続回路ブロックは、第２のマスタインターフェースを含むことができる。第２の相互接続回路ブロックは、第１の相互接続回路ブロックの第２のマスタインターフェースに結合されている第２のスレーブインターフェースを含むことができる。

【０１７７】

方法はまた、第１の領域内に、第２の相互接続回路ブロックのマスタインターフェースに結合されているスレーブインターフェースを有するメモリコントローラを設けるステップをも含むことができる。

【０１７８】

方法は、ホストプロセッサに、ヘテロジニアスマルチプロセッサ設計のホストプログラムコードを与えるステップをさらに含むことができる。

【０１７９】

方法はまた、第２の領域内に、第１の相互接続回路ブロックの第１のマスタインターフェースに結合されている第２の領域のスレーブインターフェース、および、第１のカーネル回路ブロックの入力に結合されているマスタインターフェースを有する第３の相互接続回路ブロックを含めるステップをも含むことができる。第４の相互接続回路ブロックを、第２の領域内に設けることができる。第４の相互接続回路ブロックは、第１のカーネル回路ブロックの出力に結合されているスレーブインターフェース、および、第２のインターフェース回路ブロックの第１のスレーブインターフェースに結合されているマスタインターフェースを有することができる。

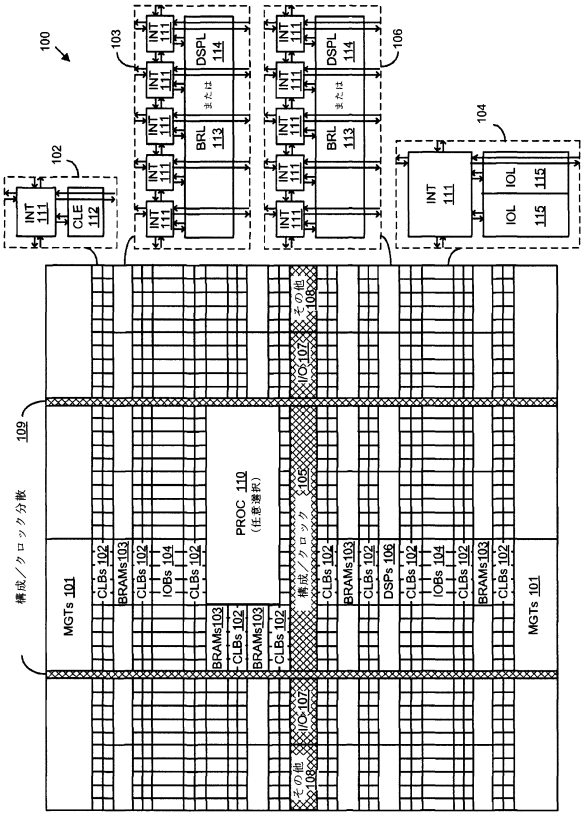
【０１８０】

方法はまた、第２の領域内に、第３の相互接続回路ブロックのマスタインターフェースに結合されている入力、および、第４の相互接続回路ブロックのスレーブインターフェースに結合されている出力を有する第２のカーネル回路ブロックを設けるステップをも含むことができる。

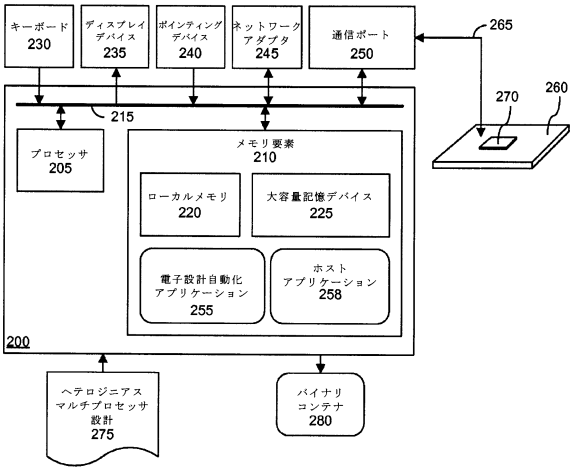
【０１８１】

本開示内に記載されている特徴は、その趣旨または本質的な属性から逸脱することなく、他の形態で具現化されてもよい。したがって、そのような特徴および実施態様の範囲を示すものとして、上記の開示ではなく、添付の特許請求の範囲を参照すべきである。

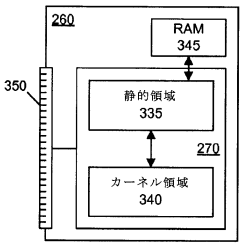
【図１】



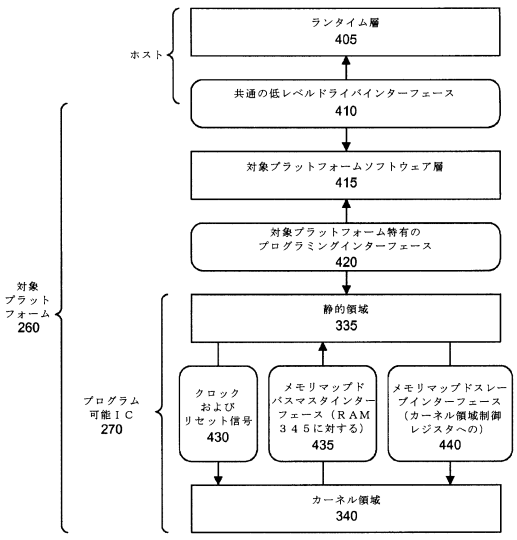
【図２】



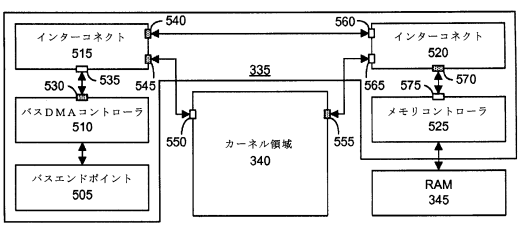
【図３】



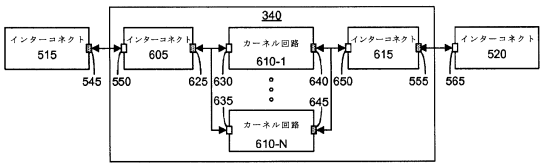
【図４】



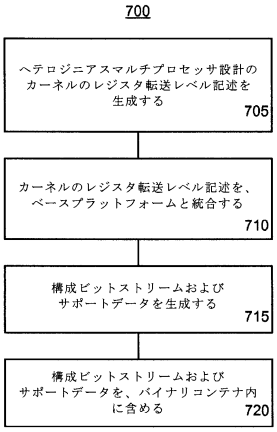
【図５】



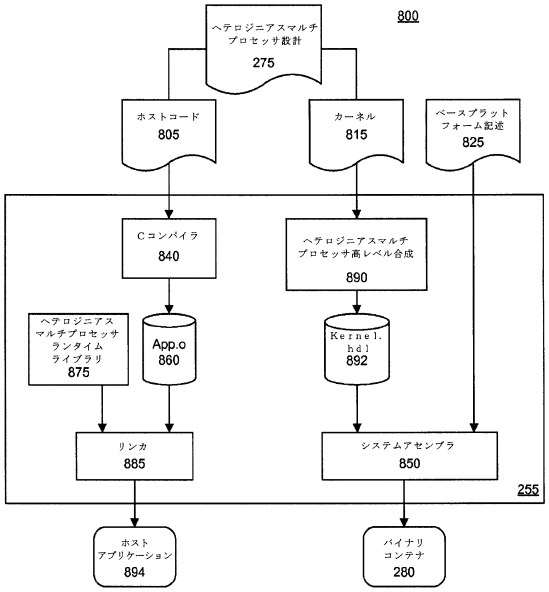
【図６】



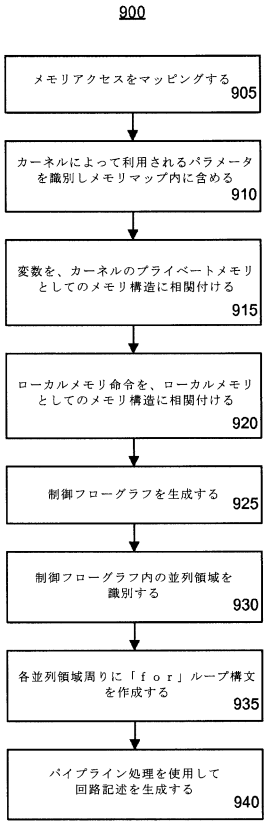
【図７】



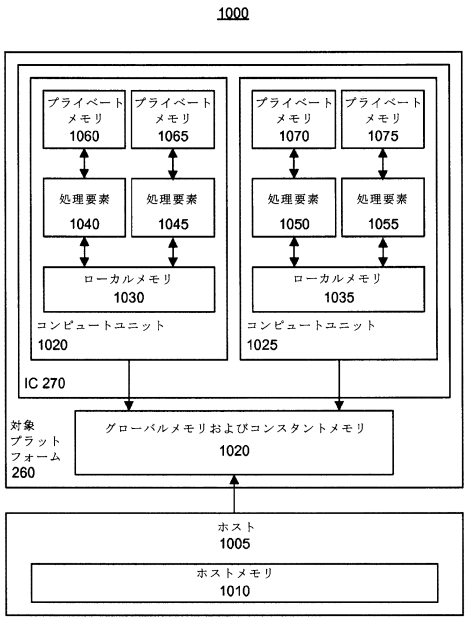
【図８】



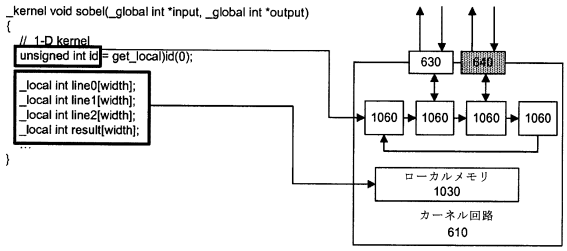
【図９】



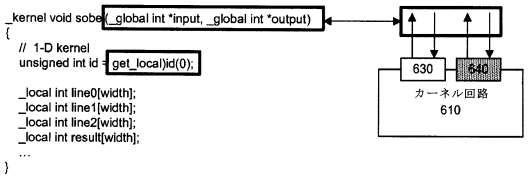
【図１０】



【図１２】



【図１１】



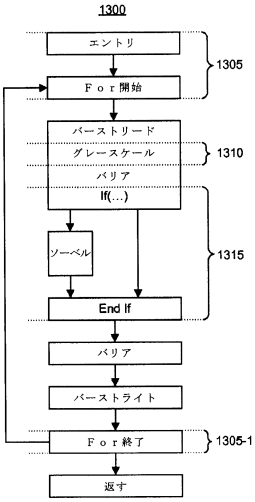
【図１３－１】

1300

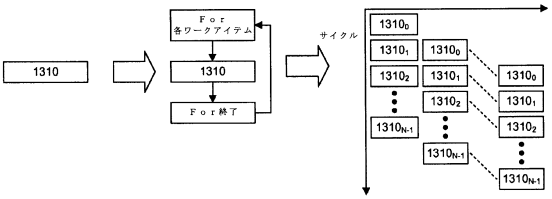
```
#define WIDTH 1920
#define LINES 1080
__kernel void sobel(__global int *input, __global int *output)
{
  // 1-D kernel                                                    1305
  unsigned int id = get_local_id(0);

  __local int line0[WIDTH];
  __local int line1[WIDTH];
  __local int line2[WIDTH];
  __local int result[WIDTH];

  // for each output line in the frame
  for (unsigned line = 1; line < LINES-1; line++)
  {
    // Fetch values
    __global int *in = input + line*WIDTH;
    event_t ev[3];
    ev[0] = async_work_group_copy(line0, in - WIDTH, WIDTH, 0);
    ev[1] = async_work_group_copy(line1, in        , WIDTH, 0);
    ev[2] = async_work_group_copy(line2, in + WIDTH, WIDTH, 0);
    wait_group_events(3, ev);                                     1310
    // Convert to Grayscale
    line0[id] = ((line0[id]       & 0xFF) + ((line0[id] >> 8)  & 0xFF) + ((line0[id] >> 16) & 0xFF) + 64) >> 2;
    line1[id] = ((line1[id]       & 0xFF) + ((line1[id] >> 8)  & 0xFF) + ((line1[id] >> 16) & 0xFF) + 64) >> 2;
    line2[id] = ((line2[id]       & 0xFF) + ((line2[id] >> 8)  & 0xFF) + ((line2[id] >> 16) & 0xFF) + 64) >> 2;

    barrier(CLK_LOCAL_MEM_FENCE);

    // Calculate Sobel Filter                                     1315
    if (id != 0 && id != width-1) {
      int gx = - line0[id-1] - 2*line1[id-1] - line2[id-1] + line0[id+1] + 2*line1[id+1] + line2[id+1];
      int gy = - line0[id-1] - line0[id] - line0[id+1] + line2[id-1] + 2*line2[id] + line2[id+1];
      if (gx < 0) gx = -gx;
      if (gy < 0) gy = -gy;
      result[id] = (gx + gy) & 0xFF;
      if (result[id] < 55)
        result[id] = 0xFFFFFFFF;
      else if (result[id] > 155)
        result[id] = 0xFF000000;
      else {
        result[id] = 255 - result[id];
        result[id] = result[id] | (result[id] << 8) | (result[id] << 16) | 0xFF000000;
      }
    }
    barrier(CLK_LOCAL_MEM_FENCE);

    event_t ev0 = async_work_group_copy(output+line*WIDTH+1, result, width-2, 0);
    wait_group_events(1, &ev0);
  }
}
```

FIG. 13-1

【図１３－２】



【図１４】



【図１５】

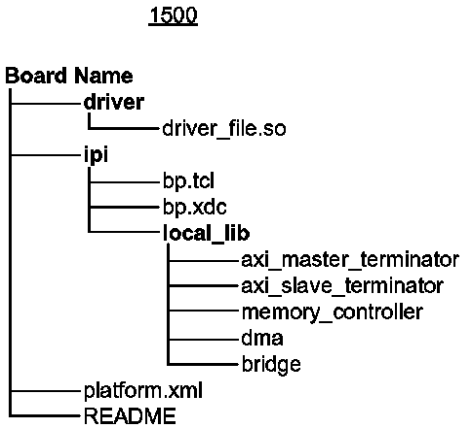
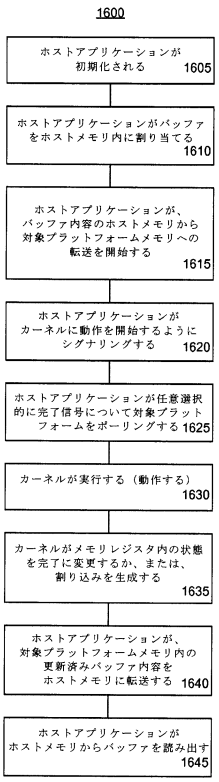


FIG. 15

【図１６】



フロントページの続き

(72)発明者 ファイフィールド，ジェフリー・エム
アメリカ合衆国、９５１２４ カリフォルニア州、サン・ノゼ、ロジック・ドライブ、２１００
(72)発明者 ビッティヒ，ラルフ・デー
アメリカ合衆国、９５１２４ カリフォルニア州、サン・ノゼ、ロジック・ドライブ、２１００
(72)発明者 ジェイムズ－ロックスビー，フィリップ・ビィ
アメリカ合衆国、９５１２４ カリフォルニア州、サン・ノゼ、ロジック・ドライブ、２１００
(72)発明者 サンタン，ソナル
アメリカ合衆国、９５１２４ カリフォルニア州、サン・ノゼ、ロジック・ドライブ、２１００
(72)発明者 バルマー，ディバダス
アメリカ合衆国、９５１２４ カリフォルニア州、サン・ノゼ、ロジック・ドライブ、２１００
(72)発明者 マルティネス・バルリナ，フェルナンド・ホタ
アメリカ合衆国、９５１２４ カリフォルニア州、サン・ノゼ、ロジック・ドライブ、２１００
(72)発明者 チョウ，シェン
アメリカ合衆国、９５１２４ カリフォルニア州、サン・ノゼ、ロジック・ドライブ、２１００
(72)発明者 ロー，クオック－ワウ
アメリカ合衆国、９５１２４ カリフォルニア州、サン・ノゼ、ロジック・ドライブ、２１００

審査官 多胡 滋

(56)参考文献 特表２０１５－５０３１６１（ＪＰ，Ａ）
特開２０１３－１６４８４７（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ ８／４０
Ｇ０６Ｆ ９／４５５