(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2014-225194
(P2014-225194A)
(43)公開日　平成26年12月4日(2014.12.4)

(51)Int.Cl.		F I			テーマコード（参考）
G06F 17/50	(2006.01)	G06F	17/50	654A	5B046
H01L 21/82	(2006.01)	H01L	21/82	C	5F064

審査請求 未請求　請求項の数 10　ＯＬ　（全 18 頁）

(21)出願番号　特願2013-105024 (P2013-105024)
(22)出願日　平成25年5月17日 (2013.5.17)

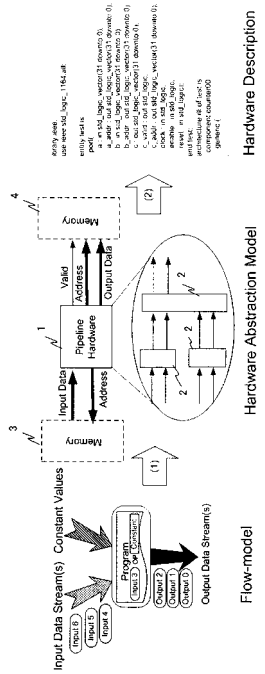
（出願人による申告）平成２４年度、独立行政法人科学技術振興機構、戦略的創造研究推進事業（個人型研究（さきがけ））に関する委託研究、産業技術力強化法第１９条の適用を受ける特許出願

(71)出願人　504171134
国立大学法人　筑波大学
茨城県つくば市天王台一丁目１番１
(74)代理人　100100549
弁理士　川口　嘉之
(74)代理人　100123319
弁理士　関根　武彦
(74)代理人　100105407
弁理士　高田　大輔
(72)発明者　山際　伸一
茨城県つくば市天王台一丁目１番１　国立大学法人筑波大学内

Ｆターム(参考)　5B046 AA08 BA02 BA03 DA06
5F064 AA04 BB40 EE47 HH01 HH06 HH08 HH10 HH11 HH13 HH14 HH15 HH17

(54)【発明の名称】ハードウェア設計装置，及びハードウェア設計用プログラム

(57)【要約】
【課題】パイプライン処理の演算内容に応じたパイプラインハードウェアのモデルを自動生成可能とする。
【解決手段】ストリームデータを処理するためのパイプライン処理に使用可能な複数のハードウェアコンポーネントの定義を含むコンポーネント情報と、前記パイプライン処理で行われる演算が記述されたプログラムと、前記演算における入力及び出力の定義を少なくとも含む定義情報とを用いて、前記演算の内容及び前記定義情報に応じた２以上のハードウェアコンポーネントを前記コンポーネント情報に基づき特定し、前記２以上のハードウェアコンポーネントが前記パイプライン処理を行うように合成されたパイプラインハードウェアのハードウェア記述言語による記述を生成する。
【選択図】図２



【特許請求の範囲】
【請求項１】
ストリームデータを処理するためのパイプライン処理に使用可能な複数のハードウェアコンポーネントの定義を含むコンポーネント情報と、前記パイプライン処理で行われる演算が記述されたプログラムと、前記演算における入力及び出力の定義を少なくとも含む定義情報とを用いて、前記演算の内容及び前記定義情報に応じた２以上のハードウェアコンポーネントを前記コンポーネント情報に基づき特定し、前記２以上のハードウェアコンポーネントが前記パイプライン処理を行うように合成されたパイプラインハードウェアのハードウェア記述言語による記述を生成する制御装置
を含むハードウェア設計装置。
【請求項２】
前記プログラムは、前記演算を行うプロセッサを識別子の記述により指定可能なストリームコンピューティング向けのプログラム言語で記述され、
前記制御装置は、前記プログラム中の演算に関して識別子の指定が記述されている場合には、当該演算によって得られる出力データの書き込みアドレスを示すカウンタ値を出力するカウンタを含む前記パイプラインハードウェアのハードウェア記述言語による記述を生成し、
前記カウンタから出力されるカウンタ値は、前記演算の実行毎に変更される
請求項１に記載のハードウェア設計装置。
【請求項３】
前記制御装置は、前記プログラムに記述された前記パイプライン処理で実行される演算の構文解析を行い、前記演算の代入部分を幹ノードとし、この幹ノードの左側に前記パイプライン処理の出力データを示す葉ノードが置かれ、且つ幹ノードの右側が前記パイプライン処理に係る複数の入力データのそれぞれを示す葉ノードと前記複数の入力データを用いて前記出力データを算出するために使用される演算子を示す枝ノードとを含む二分木構造で表された抽象構文木の生成を試行し、前記抽象構文木が生成されないときにエラーを出力する
請求項１又は２に記載のハードウェア設計装置。
【請求項４】
前記制御装置は、前記抽象構文木が生成されたときに、前記枝ノードに対する２つの入力のそれぞれにおける遅延が均等になるように遅延を挿入し、挿入した遅延を含む前記パイプラインハードウェアのハードウェア記述言語による記述を生成する
請求項３に記載のハードウェア設計装置。
【請求項５】
前記制御装置は、前記コンポーネント情報が同一の演算タイプを有する複数のハードウェアコンポーネントの定義を含むときに、演算による遅延が最も小さいハードウェアコンポーネントを前記２以上のハードウェアコンポーネントの１つとして選択する
請求項１から４のいずれか１項に記載のハードウェア設計装置。
【請求項６】
ストリームデータを処理するためのパイプライン処理に使用可能な複数のハードウェアコンポーネントの定義を含むコンポーネント情報と、前記パイプライン処理で行われる演算が記述されたプログラムと、前記演算における入力及び出力の定義を少なくとも含む定義情報とを用いて、前記演算の内容及び前記定義情報に応じた２以上のハードウェアコンポーネントを前記コンポーネント情報に基づき特定するステップと、
特定された２以上のハードウェアコンポーネントが前記パイプライン処理を行うように合成されたパイプラインハードウェアのハードウェア記述言語による記述を生成するステップと
をコンピュータに実行させるハードウェア設計用プログラム。
【請求項７】
前記プログラムは、前記演算を行うプロセッサを識別子の記述により指定可能なストリ

ームコンピューティング向けのプログラム言語で記述され、
前記制御装置は、前記プログラム中の演算に関して識別子の指定が記述されている場合には、当該演算によって得られる出力データの書き込みアドレスを示すカウンタ値を出力するカウンタを含む前記パイプラインハードウェアのハードウェア記述言語による記述を生成するステップを前記コンピュータに実行させ、
前記カウンタから出力されるカウンタ値は、前記演算の実行毎に変更される
請求項６に記載のハードウェア設計用プログラム。
【請求項８】
前記プログラムに記述された前記パイプライン処理で実行される演算の構文解析を行うステップと、
前記演算の代入部分を幹ノードとし、この幹ノードの左側に前記パイプライン処理の出力データを示す葉ノードが置かれ、且つ幹ノードの右側が前記パイプライン処理に係る複数の入力データのそれぞれを示す葉ノードと前記複数の入力データを用いて前記出力データを算出するために使用される演算子を示す枝ノードとを含む二分木構造で表された抽象構文木の生成を試行するステップと、
前記抽象構文木が生成されないときにエラーを出力するステップと
を前記コンピュータに実行させる請求項６又は７に記載のハードウェア設計用プログラム。
【請求項９】
前記抽象構文木が生成されたときに、前記枝ノードに対する２つの入力のそれぞれにおける遅延が均等になるように遅延を挿入するステップと、
挿入した遅延を含む前記パイプラインハードウェアのハードウェア記述言語による記述を生成するステップと
を前記コンピュータに実行させる請求項８に記載のハードウェア設計用プログラム。
【請求項１０】
前記コンポーネント情報が同一の演算タイプを有する複数のハードウェアコンポーネントの定義を含むときに、演算による遅延が最も小さいハードウェアコンポーネントを前記２以上のハードウェアコンポーネントの１つとして選択するステップ
を前記コンピュータに実行させる請求項６から９のいずれか１項に記載のハードウェア設計用プログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、ハードウェア設計装置，及びハードウェア設計用プログラムに関する。
【背景技術】
【０００２】
現在、ネットワークには、様々なセンサ及びデバイスが接続され、これらのセンサ及びデバイスから刻々と出力されるデータがネットワーク上でデータストリームを形成する。データストリームを形成するデータ（ストリームデータと呼ばれる）を滞りなく処理する（リアルタイムに処理する）手法として、ストリームコンピューティングがある。
【０００３】
ストリームコンピューティングでは、ストリームデータに関して所定のデータ単位が入力として決定され、このデータ単位に対する複数の処理（演算）が直列に実行されるパイプライン処理が行われる。
【０００４】
パイプライン処理を実行する典型的な環境として、ＧＰＵ（Graphical Processing Unit）を用いたソフトウェア処理がある。ＧＰＵは、並列処理を実行可能な複数のプロセッサを含んでおり、パイプライン処理の手順を記述したプログラムに従って、各プロセッサにパイプライン処理に係る複数の処理を実行させることで、パイプライン処理結果を得ることができる。

【０００５】
OpenCLのような、ＧＰＵ向けのプログラミング言語では、ＧＰＵが並列処理を実行可能な複数のプロセッサを有することを考慮した、パイプライン処理手順を記述することができる。例えば、プログラム上で、パイプライン処理に係る複数の演算を、複数のプロセッサに割り当てることができる。
【先行技術文献】
【特許文献】
【０００６】
【特許文献１】特開２０１２－１７４２６８号公報
【非特許文献】
【０００７】
【非特許文献１】“OpenCL規格を用いたFPGAデザインの導入”、[online]、アルテラ、[平成２５年５月１３日検索]、インターネット<URL: http://www.altera.co.jp/literature/wp/wp-01173-opencl_j.pdf>
【非特許文献２】Mencer, O., ASC, "a stream compiler for computing with FPGAs, Computer-Aided Design of Integrated Circuits and Systems," IEEE Transactions on (Volume:25 , Issue: 9) , pp. 603 - 1617, Sept. 2006.
【非特許文献３】Shinichi Yamagiwa, leonel Sousa, "Caravela: A Novel Stream-Based Distributed Computing Environment," Computer , vol.40, no.5, pp.70-77, May 2007.
【発明の概要】
【発明が解決しようとする課題】
【０００８】
しかしながら、ＧＰＵ及びＧＰＵを用いたシステムは、回路規模が大きく、またコストの上昇を招来する。このため、パイプライン処理を行うために、ＧＰＵ乃至ＧＰＵシステムを導入することが困難、又は非現実的である場合が少なくない。
【０００９】
このため、ストリームデータ，ストリームデータに対して行う処理（演算）の内容などに応じたパイプライン処理を実行するハードウェア（パイプライン処理用のディジタル回路：パイプラインハードウェアと呼ぶ）の開発が求められている。
【００１０】
パイプラインハードウェアを設計する際には、パイプライン処理で実行される複数の演算（四則演算等）をそれぞれ実行する複数の演算コンポーネント（「ハードウェアコンポーネント」と呼ぶ。以下単に「コンポーネント」と表記）が用意され、演算の順序に従って複数のコンポーネントが配線により接続される。パイプラインハードウェアにはクロックが入力され、クロック毎に演算が進められる。演算に係る遅延（ディレイ）はクロック数として定義される。
【００１１】
従来におけるハードウェア記述言語（hardware description language：HDL）を用いたパイプラインハードウェアの設計では、設計者がパイプライン処理における演算内容に応じて複数のコンポーネントを選択し、選択したコンポーネント間の配線接続によってパイプラインハードウェアを設計する。このとき、設計者がコンポーネント間のデータ到達タイミングを計りながら、最終的な演算結果が正しく出力されるための遅延（ディレイ）をマニュアル操作で挿入する。
【００１２】
コンポーネントは様々な遅延を持つ。例えば、加算器は、実装技術（ＬＳＩのプロセス）の違いにより遅延が異なる。さらに、積算回路については繰り返し処理が伴うことで遅延が可変となることがある。このように、パイプラインハードウェアの設計では、各コンポーネントの遅延を考慮することにより、全体の時間バランスを考慮することが要求される。

【００１３】
しかしながら、例えば、コンポーネントの入れ替え（選択変更）により、コンポーネント自体の遅延が変化した場合には、この変化をコンポーネント間に挿入されたディレイで吸収できない場合が起こり得る。この影響がパイプライン全体に及ぶ場合には、パイプラインハードウェアの設計を最初からやり直すことが要求される虞があった。このように、設計者に対して、コンポーネントの選択と、挿入する遅延の考慮とを慎重に行うことが要求され、これは設計者にとって大きな負担であった。
【００１４】
現在のところ、パイプライン処理の演算内容に応じたパイプラインハードウェアのひな形となるモデルを自動的に生成する機構はない。このため、コンポーネントの特性と、コンポーネントのパイプラインへの適用を、目的とするパイプラインハードウェアに応じて一意に決定可能な機構、特に、コンポーネントを考慮した遅延を自動的に挿入可能な機構もない。
【００１５】
本発明は、上記の事情に鑑みなされたものであり、パイプライン処理の演算内容に応じたパイプラインハードウェアのモデルを自動的に生成可能な技術を提供することを目的とする。
【課題を解決するための手段】
【００１６】
本発明は、上記課題を解決するために以下の手段を採用する。すなわち、本発明は、ストリームデータを処理するためのパイプライン処理に使用可能な複数のハードウェアコンポーネントの定義を含むコンポーネント情報と、前記パイプライン処理で行われる演算が記述されたプログラムと、前記演算における入力及び出力の定義を少なくとも含む定義情報とを用いて、前記演算の内容及び前記定義情報に応じた２以上のハードウェアコンポーネントを前記コンポーネント情報に基づき特定し、前記２以上のハードウェアコンポーネントが前記パイプライン処理を行うように合成されたパイプラインハードウェアのハードウェア記述言語による記述を生成する制御装置を含むハードウェア設計装置である。
【００１７】
また、本発明は、上記した制御装置における処理をコンピュータに実行させるハードウェア設計用プログラム，このようなプログラムを記録したコンピュータ読み取り可能な記録媒体，及び上記制御装置によるパイプラインハードウェア記述の生成方法としても特定することができる。
【発明の効果】
【００１８】
本発明によれば、パイプライン処理の演算内容に応じたパイプラインハードウェアのモデルを自動的に生成可能な技術を提供することができる。
【図面の簡単な説明】
【００１９】
【図１】図１は、ハードウェア設計用プログラムの実行によってハードウェア設計装置として機能する情報処理装置（コンピュータ）の構成例を示す図である。
【図２】図２は、ＳＰＣの仕組みを説明する図である。
【図３】図３は、ＸＭＬで記述されたフローモデルの例を示す。
【図４】図４は、カウンタ生成に係る処理を説明する図である。
【図５】図５は、コンポーネントの定義ファイルの例を示す。
【図６】図６は、コンポーネントの選択に係る説明図である。
【図７】図７は、抽象構文木の例を示す図である。
【図８】図８は、演算式“a[id] = b[id] + c[id] * d[id] - e[id]”に応じて作成されたパイプラインハードウェアのモデル（ＨＡＭ）における遅延を説明する図である。
【図９】図９は、遅延挿入の例を示す説明図である。
【図１０】図１０は、ＣＰＵによって実行されるＳＰＣにおける処理を大略して示すフロ

ーチャートである。
【図１１】図１１は、ＨＤＬによるパイプラインハードウェアの記述例を示す。
【発明を実施するための形態】
【００２０】
以下、図面を参照して本発明の実施形態について説明する。実施形態の構成及び設定は例示であり、本発明は実施形態の構成及び設定に限定されない。
【００２１】
以下、実施形態に係るハードウェア設計用プログラム、及びハードウェア設計用プログラムの実行によりハードウェア設計装置として機能する情報処理装置（コンピュータ）について説明する。
【００２２】
ハードウェア設計用プログラムは、ストリームデータを処理するためのパイプライン処理に使用可能な複数のハードウェアコンポーネントの定義を含むコンポーネント情報と、上記パイプライン処理で行われる演算が記述されたプログラムと、上記演算における入力及び出力の定義を少なくとも含む定義情報とを用いて、上記演算の内容及び上記定義情報に応じた２以上のハードウェアコンポーネントを上記コンポーネント情報に基づき特定する。
【００２３】
さらに、ハードウェア設計用プログラムは、上記２以上のハードウェアコンポーネントが上記パイプライン処理を行うように合成されたパイプラインハードウェアのハードウェア記述言語による記述を生成する。このようなハードウェア設計用プログラムを“ストリーム・パイプライン・コンパイラ（ＳＰＣ）”と呼ぶ。
【００２４】
＜情報処理装置（ハードウェア設計装置）＞
図１は、ハードウェア設計用プログラムの実行によってハードウェア設計装置として機能する情報処理装置（コンピュータ）の構成例を示す図である。情報処理装置１０として、例えば、パーソナルコンピュータ（ＰＣ），ワークステーション，専用又は汎用のサーバマシンを適用することができる。
【００２５】
図１において、情報処理装置１０は、例として、バスＢを介して相互に接続された、ＣＰＵ１１と、主記憶装置１２と、補助記憶装置１３と、入力装置１４と、出力装置１５と、通信インタフェース回路（通信Ｉ／Ｆ）１６とを備える。
【００２６】
主記憶装置１２は、ＣＰＵ１１の作業領域として使用されるメインメモリとして機能する。メインメモリは、例えば、ＲＡＭ（Random Access Memory）及びＲＯＭ（Read Only Memory）によって形成される。
【００２７】
補助記憶装置１３は、制御装置に相当するＣＰＵ１１によって実行される、ハードウェア設計用プログラムを含む各種のプログラム，及び各プログラムの実行時に使用されるデータを記憶する。補助記憶装置１３は、例えば、不揮発性記録媒体であり、例えば、ハードディスク，フラッシュメモリ，ＥＥＰＲＯＭ（Electrically Erasable Programmable Read-Only Memory）の少なくとも１つを用いて形成することができる。主記憶装置１２及び補助記憶装置１３のそれぞれは、記憶装置，記録媒体の一例である。
【００２８】
入力装置１４は、キーボード，マウスやタッチパネルのようなポインティングデバイスを含み、情報（データ）の入力に使用される。出力装置１５は、例えば、ディスプレイ装置であり、情報を画面に表示する。通信Ｉ／Ｆ１６は、ネットワークとの通信処理を司る。
【００２９】
ＳＰＣは、補助記憶装置１３にインストールされており、ＣＰＵ１１がＳＰＣを主記憶

装置１２に読み出し、ロードして実行することによって、情報処理装置１０は、ハードウェア設計装置として機能することができる。

【００３０】

＜ＳＰＣ＞

図２は、ＳＰＣの仕組みを説明する図である。図２に示すように、ＳＰＣに対する入力として、“Flow-model（フローモデル）”と呼ばれる、パイプライン処理のモデルを定義したモデル定義情報と、パイプライン処理で行われる演算が記述されたプログラムとが所定の記述言語で記述されたファイル（パイプライン処理モデルのファイル）が用意される。各ファイルは、補助記憶装置１３に記憶され、ＣＰＵ１１によるＳＰＣの実行に際して使用される。

【００３１】

ＳＰＣは、フローモデルのファイルを入力として、フローモデルにおいて定義されたパイプライン処理を行うパイプラインハードウェアの抽象化モデル（Hardware Abstraction Model：ＨＡＭ）を生成する（図２（１））。抽象化モデル（ＨＡＭ）は、フローモデルにおける演算内容に従った演算を行う演算子である２以上のコンポーネントを特定し、２以上のコンポーネントを合成することによって生成される。さらに、ＳＰＣは、ＨＡＭをハードウェア記述言語（ＨＤＬ）で記述したパイプラインハードウェアの記述（Hardware Description）を生成して出力する（図２（２））。

【００３２】

このように、ＳＰＣは、パイプラインハードウェアを設計するためのひな形として利用可能なＨＡＭ及びパイプラインハードウェアの記述を自動的に生成することができる。この点で、設計者の負担を減らすことができる。

【００３３】

図２において、パイプラインハードウェア１は、所定の複数の入力データを得て、パイプライン処理により所望の出力データを出力するハードウェアである。パイプラインハードウェア１は、パイプラインを形成する複数のコンポーネント２（ハードウェアコンポーネントとして機能する複数の部分ハードウェア）を含み、コンポーネント２間の配線接続によって合成される。

【００３４】

＜＜フローモデル＞＞

フローモデルの記述言語として、例えば、ＸＭＬ（Extensible Markup Language）を含む様々なマークアップ言語を適用可能である。

【００３５】

パイプライン処理で行われる演算が記述されたプログラムの一例として、ストリームコンピューティング向けのプログラム言語であるOpenCL（Open Computing Language）で記述されたプログラムを使用することができる。但し、パイプライン処理で行われる演算内容が特定可能に記述される限り、OpenCL以外の他のプログラム言語を適用することもできる。

【００３６】

OpenCLは、ＧＰＵ（Graphics Processing Unit）用のプログラムを記述することができる。一般に、ＧＰＵは、複数のプロセッサを有し、これらのプロセッサを用いた並列処理を行うことができる。OpenCLは、「プロセッサインデックス」，「プロセッサＩＤ」と呼ばれる“id”値を指定する構文（例えば、id=get_grobal_id(0)：括弧内の数字"０"はid値（id番号））の記述と、例えば、“c[id]=+a[id]-100+b[id]”のような、idの指定を含む演算内容の記述とにより、並列処理において個々の処理を行うプロセッサを指定することができる。

【００３７】

“フローモデル”のファイルは、パイプライン処理で行われる演算が記述されたプログラム（カーネルプログラムと呼ばれる）として、OpenCLを用いたカーネルプログラムの記述を含むことができる。カーネルプログラムにおいて、“id”を指定した演算内容が記述

される。
【００３８】
OpenCLのような、ＧＰＵのようなアクセラレータで実行されるストリームコンピューティング向けのプログラミング言語におけるパイプライン処理の手順に係る記述は、ストリームデータを順次、処理していくハードウェア構成の記述と等価である。このため、当該プログラムがカーネルプログラムとしてＳＰＣに入力され、ＳＰＣがカーネルプログラムをコンパイルすることで、パイプラインハードウェアのコンポーネント割り出し乃至選択を行うことが可能となる。
【００３９】
定義情報は、パイプライン処理に使用される入力データ，パイプライン処理の結果として出力される出力データの定義情報を含む。入力データ及び出力データの定義情報として、カーネルプログラムの関数引数を入力データ又は出力データとする定義情報が記述される。
【００４０】
図３は、ＸＭＬで記述されたフローモデルの例を示す。図３において、カーネルのタグ（<kernel> </kernel>）で挟まれた部分が、OpenCLのプログラム、すなわちカーネルプログラムの記述である。そして、カーネルプログラムに記述された引数“ａ”及び“ｂ”を入力（input）とし、演算結果の“ｃ”を出力（output）とするモデル定義情報が記述されている。図２の例では、“ａ”，“ｂ”，“ｃ”のそれぞれの定義（名称，データタイプ，データ長等）が記述されている。
【００４１】
なお、上記説明では、カーネルプログラムと定義情報とがフローモデルのファイルにおいて一つにまとめられているが、カーネルプログラムと定義情報とは個別のファイルであっても良い。
【００４２】
＜＜パイプラインハードウェアの抽象化モデル（ＨＡＭ）＞＞
次に、パイプラインハードウェアの抽象化モデル（ＨＡＭ）の詳細について説明する。ＳＰＣによって生成されるＨＡＭは、以下のように規定（設定）される。
・入力は、“（入力）データ”，“（入力）データのアドレス”，“クロック”，及び“リセット”を含む。
・出力は、“（出力）データ”，“（出力）データの有効性を示すValid信号”，“（出力）データのアドレス”を含む。
・パイプラインハードウェアに対する入力は、メモリ（図１のメモリ３を参照）からの読み出しによって行われる。
・出力データは、メモリ（図１のメモリ４を参照）に書き込まれる。
・入力時における“（入力）データのアドレス”から読み出された“（入力）データ”のメモリからの読み出し時間“T_{ad}”は、ＳＰＣの合成変数として与えられる。
・パイプラインハードウェアへの入力データを“入力ポート”，パイプラインハードウェアからの出力データを“出力ポート”と呼ぶ。
【００４３】
[入出力に係る規定]
パイプラインハードウェアに対する入力データ（入力ポート）及び出力データ（出力ポート）に関して、例えば、以下のように規定される。
・パイプラインハードウェアを構成するための演算式（例えば、図２のフローモデルにおける“c[id]=+a[id]-100+b[id]”）に現れる入力ポート（ａ及びｂ）は必ず演算式の右辺になくてはならない。
・上記演算式において、出力ポート（ｃ）は左辺にだけ現れなくてはならない。
【００４４】
パイプライン構造を作る際に、或る演算の出力データが他の演算の入力データとされると、パイプラインが再帰構造を有する状態となり、過去のデータ（或る演算の出力データ

）を再利用するためのメモリが必要となる。そこで、本実施形態では、メモリの使用を回避すべく、データが入力データ（入力ポート）と、出力データ（出力ポート）とのどちらであるかを規定する。

【００４５】

上記規定が満たされない場合には、エラーが出力される。例えば、ＣＰＵ１１が、出力装置１５にエラーを表示する。これによって、プログラム中でのメモリ発生可能性（再帰構造）を検出でき、設計者（プログラマ）に修正を促すことで、メモリを使用しないＨＡＭを生成できる。また、再帰構造の排除によって、ＨＡＭ生成の際におけるコンポーネントの合成が容易になる。

【００４６】

[カウンタの生成]

入力とされるプログラム（上記したフローモデル中のカーネルプログラム）で使用される入力データ（入力ポート）及び出力データ（出力ポート）の各アドレスは、カーネルプログラムで記述されたプロセッサＩＤ値（id値：「識別子」に相当）に基づいて指定される。プログラムの構造において、プロセッサＩＤは連続的な値を有し、カウンタのインクリメントによってプロセッサＩＤが変更される。変更されたプロセッサＩＤが次のアドレスとして機能する。

【００４７】

図４は、カウンタ生成に係る処理を説明する図である。例えば、図４に示すようなカーネルプログラムを含むフローモデルのファイルがＳＰＣに供給されたと仮定する。この場合、ＳＰＣ（を実行するＣＰＵ１１）は、ファイルからカーネルプログラムを抽出し、構文解析を行う。

【００４８】

このとき、“id=get_global_id(0)”のようなプロセッサＩＤを指定する構文がある場合には、ＳＰＣは、演算式（“c[id]=+a[id]-100+b[id]”）に基づき、図４の下段に図示するようなパイプラインハードウェアのモデルを生成する。

【００４９】

図４において、カウンタ値は、入力ポート“ａ”及び“ｂ”、並びに出力ポート“ｃ”に対するアドレスとして供給される。“ａ”のアドレス“a_addr”及び“ｂ”のアドレス“b_addr”は、入力ポート側にあるメモリ３（図２）のアドレスとして機能する。そして、アドレスとして指定された箇所から読み出された“ａ”の値は、即値“１００”とともに、減算器（Subtract）に入力される。“ｂ”の値は、減算器の出力とともに、加算器（adder）に入力される。そして、加算器から出力ポートである“ｃ”の値が出力される。“ｃ”の値は、アドレス“c_addr”として指定されたカウンタ値に対応する、出力ポート側のメモリ４（図２）の記憶領域に書き込まれる。そして、所定の契機（例えば、“ｃ”の出力“書き込み”）で、カウンタ値がインクリメント（現在のカウンタ値に１を加算）される。そして、次のカウンタ値に対応するメモリ３の記憶領域から、入力ポート“ａ”及び“ｂ”を取得し、次のカウンタ値に対応するメモリ４の記憶領域に出力ポート“ｂ”を書き込むことができる。

【００５０】

なお、カウンタ値としての“ａ”のアドレス“a_addr”及び“ｂ”のアドレス“b_addr”は、同じ値であるが、ポート毎に異なるメモリ空間（記憶領域）をアクセスするようにして、異なるデータが入力ポートとして得られるようにすることができる。例えば、入力ポート“ａ”の読み出し用メモリと入力ポート“ｂ”の読み出し用メモリとが物理的に別にされることが考えられる。或いは、ポート毎に異なるバンクを持ったデュアルポートメモリを適用することが考えられる。また、入力ポート“ａ”及び“ｂ”に関して、意図して同じメモリ空間にアクセスする場合もあり得る。

【００５１】

このように、実施形態は、プログラムが演算を行うプロセッサを識別子の記述により指定可能なストリームコンピューティング向けのプログラム言語で記述され、且つプログラ

ム中の演算に関して識別子の指定が記述されている場合に、当該演算によって得られる出力データの書き込みアドレスを示すカウンタ値を出力するカウンタを含む前記パイプラインハードウェアのハードウェア記述言語による記述を生成し、カウンタから出力されるカウンタ値は、演算の実行毎に変更される構成を含む。

【００５２】

なお、「インクリメント」の用語は、現在のカウンタ値に１を加算することであり、このような動作を行う場合に、カウンタの回路構成が最も簡易となる。但し、実施形態では、“ id ”番号の指定構文と、当該“ id ”番号を用いた演算式があるときに、その演算毎に異なるアドレスを生成及び出力するカウンタが生成されるようにすれば良い。このため、カウンタ値は、演算毎に、カウンタ値を変更するようにされていれば良い。このとき、カウンタ値は、例えば“ 規則的に増加又は減少 ”するように変更可能であり、１回のカウンタ値の変更において増加又は減少する数値は、１でも２以上であっても良い。また、id値として採り得る値は、本実施形態では連続する値としているが、離散値（規則的に増加、減少する値）であっても良い。

【００５３】

カウンタを利用した簡易なアドレス生成回路の作成によって、ＨＡＭ（パイプラインハードウェア）の回路構成を簡潔にすることができる。なお、カウンタ値は、採り得る値の最大値となったときには、次に採り得る値の最小値が出力されるようにすることができる。また、カウンタ値とメモリの記憶領域とが関連づけられ、或るカウンタ値に対して対応するメモリの記憶領域に対する読み出し／書き込みが行われるようにすることができる。

【００５４】

[コンポーネントの指定]

ストリームデータを処理するためのパイプライン処理に使用可能な複数のハードウェアコンポーネントの定義を含むコンポーネント情報として、例えば、コンポーネントの定義ファイルが、コンポーネント毎に用意される。コンポーネント定義ファイルは、以下のような情報を含む。

・コンポーネントの実装のファイル（例えば、ＨＤＬファイル）。

・クロック，イネーブル，リセットのそれぞれの入力と、各入力に合致する実装（コンポーネント）におけるポート（端子）の識別情報（例えばポート名）との組（関連）。

・Valid出力と、Valid出力に対応する実装（部分ハードウェア）上のポート識別情報との組（関連）

・演算において、演算式の左側（左辺）に位置すべき出力ポート，及び演算式の（右辺）に位置すべき入力ポートにそれぞれ対応する実装上のポート識別情報との組（関連）。

・演算に要求されるクロックサイクル数（遅延）

・演算タイプ（加算，減算，積算など）

【００５５】

コンポーネント定義ファイルは、例えば、フローモデルのＸＭＬファイルから独立したＸＭＬファイルとしてＳＰＣに供給される。但し、フローモデルのＸＭＬに含まれてＳＰＣに供給されるようにしても良い。或いは、コンポーネント定義ファイルは、データベース（ＤＢ）上で管理され、必要に応じてＳＰＣによりアクセス（参照）されるようにしても良い。コンポーネント定義ファイルとして、演算タイプが同一であるが仕様が異なる複数のコンポーネントに関する複数のファイルを含むことができる。ＤＢは、例えば、補助記憶装置１３に記憶される。

【００５６】

図５は、コンポーネント定義ファイルの例を示す図であり、減算器（Subtract）のコンポーネント定義ファイルが例示されている。当該コンポーネント定義ファイルの記述として、コンポーネント名“ sub ”，演算タイプ（Category：SUB），及びRTL（Register Transfer Level）ファイル名，が記述されている。コンポーネント定義ファイルには、各入力データ“ ａ ”及び“ ｂ ”の定義と、出力データ“ ｃ ”の定義と、イネーブル（enable），有効（Valid），クロック（ck），リセット（reset）及び遅延（delay）に係る記述が含

まれる。

【００５７】

図６は、コンポーネントの選択に係る説明図である。ＳＰＣは、実行時において、複数のコンポーネント定義ファイルを取得及び参照して、目的のパイプラインの形成に適合する定義ファイルを選択することができる。例えば、図６では、ＳＰＣは、カウンタ（Counter）、加算器(adder)，減算器(Subtract)，積算回路(Multiplier)，遅延（Delay），ネゲート(negate)のような様々なコンポーネントの定義ファイルを受け取り、その中から、演算に適合したコンポーネントを選択（抽出）し（図６では、加算器（１），減算器，及び積算回路）、これらを合成（配線接続）したパイプラインハードウェアを生成する様子が図示されている。

【００５８】

このとき、或る演算タイプに関して複数の定義ファイルがある場合には、複数の定義ファイルから、目的に合致した定義ファイルを選択する。例えば、図６の例では、加算器に関して二つのコンポーネント定義ファイル（adder(1)，adder(2)）が得られている。ここで、例えば、最小クロックサイクル数でパイプラインを作成する（遅延を最小にする）ことが目的とされる場合には、ＳＰＣは、クロックサイクル数が少ない順で複数の定義ファイルをソートし、最小クロックサイクル数のコンポーネントの定義ファイル（図６の例では、“adder(1)”）を選択する。これによって、最短の遅延時間でパイライン処理の演算結果を出力可能なパイプラインハードウェアを作成することができる。

【００５９】

このように、コンポーネントの定義ファイルにおいて、同タイプのコンポーネント間で目的に応じた適用の優先順位を決定可能な情報（優先情報）が含まれることで、目的に応じたコンポーネントを自動的に選択する（特定する）ことが可能となる。

【００６０】

[遅延挿入（出力タイミング調整）]

ＳＰＣは、入力プログラム（カーネルプログラム）に記述されたパイプライン処理で実行される演算の構文解析を行い、二分木構造を有する抽象構文木（Abstract Syntax Tree：ＡＳＴ）を作成する。ＡＳＴの作成は、ＳＰＣによるカーネルプログラムのコンパイル時に、bisonやyaccのようなパーサジェネレータを用いて行うことができる。

【００６１】

図７は、抽象構文木の例を示す図であり、プログラム中の演算式“a[id] = b[id] + c[id] * d[id] - e[id]”に対する抽象構文木を例示する。図７に示すように、ＡＳＴの幹ノードには、代入（例えば等式における等号）が指定される。代入部分（幹ノード）の左側の葉ノードには出力ポート“ａ”が指定される。一方、代入部分の右側には、葉ノードと枝ノードとが配置される。なお、ノード間を結ぶ線（リンク）は、ハードウェアにおける信号線として宣言される。

【００６２】

葉ノードには、入力ポート，プログラム中のローカル変数，即値が指定される。図７の例では、入力ポート“ｂ”，“ｃ”，“ｄ”及び“ｅ”の葉ノードが配置されている。枝ノードには、演算式に従った演算子が配置され、演算子に応じたコンポーネントがマッピングされる。例えば、図６において、“ｃ”及び“ｄ”を入力とする演算子“*”には、コンポーネントとして、積算回路がマッピングされる。

【００６３】

また、枝ノードには、葉ノード又は下位の枝ノードの出力（演算結果）が入力される。例えば、演算子“－”の枝ノード（減算器がマッピングされる）は、入力ポート“ｅ”の値と、演算子“*”の演算結果とが入力される。そして、演算子“+”（加算器がマッピングされる）には、入力ポート“ｂ”の値と、演算子“－”の演算結果とが入力される。

【００６４】

このようにして、図６の下側に示すような、パイプライン処理に応じた２以上のコンポーネントとしての積算回路（積算器），減算器，及び加算器が合成されたパイプラインハ

ードウェアのモデルが生成される。

【００６５】

ＡＳＴの生成に当たっては、上記した入出力に係る規定に従い、幹ノードの左側に入力ポートが現れたり、右側に出力ポートが現れたりした場合には、パイプラインが再帰構造を有することになる。この場合、ＳＰＣを実行するＣＰＵ１１は、処理を停止して、エラーを出力装置１５に出力する。

【００６６】

このようにして、演算の代入部分を幹ノードとし、この幹ノードの左側にパイプライン処理の出力データを示す葉ノードが置かれ、且つ幹ノードの右側が前記パイプライン処理に係る複数の入力データのそれぞれを示す葉ノードと前記複数の入力データを用いて出力データを算出するために使用される演算子を示す枝ノードとを含む二分木構造で表された抽象構文木の生成が試行され、このような抽象構文木が生成されない場合に、エラーが出力される。

【００６７】

これに対し、構文解析によって、入出力の規定に従ったＡＳＴが生成された場合には、ＳＰＣを実行するＣＰＵ１１は、幹ノードの右側に関し、各深さにおいて、枝ノードからの出力タイミングが同じとなるように、遅延を挿入する処理を行う。

【００６８】

図７は、上記した演算式“a[id] = b[id] + c[id] * d[id] - e[id]”に応じて作成されたパイプラインハードウェアのモデル（ＨＡＭ）における遅延を説明する図である。各入力ポート“ｂ”～“ｅ”のメモリ３（図２）からの読み出し時間は、それぞれ遅延時間T_{ad}で同じであると仮定する。

【００６９】

そして、時間T_{ad}の経過後に積算器（演算子“*”）が積算結果を出力するまでの遅延時間がT_1で、T_1から減算器（演算子“－”）が減算結果を出力するまでの遅延時間がT_2で、T_2から加算器（演算子“+”）が加算結果を出力するまでの遅延時間がT_3である。

【００７０】

このとき、積算器への入力タイミングは同じであるが、減算器及び加算器では、２つのデータの入力タイミングがそれぞれ異なる。このため、２つのデータの双方が入力されるまで、演算処理を開始することができない。この結果、減算器及び加算器の少なくとも一方から正確な演算結果が出力されない虞がある。

【００７１】

そこで、ＳＰＣは、以下のようにして、遅延（遅延時間）を自動的に挿入する処理を行う。図８は、遅延挿入（出力タイミング調整）の説明図である。図８の上側には、図７に示した各遅延時間をＡＳＴの各リンクにあてはめた状態を示す。

【００７２】

これに対し、各枝ノードに至る２つの入力に関する遅延量が同じになるように、遅延を挿入する。具体的に説明すると、ＳＰＣは、（ａ）枝ノードにおける二つの入力に係る遅延の差を算出し、（ｂ）差分が生じた場合には、その差分を、遅延が小さい側に挿入する。

【００７３】

例えば、図８の下側のＡＳＴにおいて、演算子“+”の枝ノードに係る２つの入力に対する各遅延はそれぞれT_{ad}であるので、遅延の挿入は行われない。これに対し、演算子“－”の枝ノードに注目すると、演算子“+”側の入力に係る遅延時間は、$T_{ad}+T_1$であるのに対し、入力ポート“ｅ”側の入力に係る遅延時間はT_{ad}である。このため、差分$|T_1+T_{ad}-T_{ad}|$が生じる。そこで、当該差分を、演算子“*”の枝ノードによる遅延Ｄ（“*”）として、入力ポート“ｅ”側に挿入する。

【００７４】

これによって、演算子“－”の枝ノードに対するデータの入力タイミングを一致させることができるので、正確な演算結果を出力することが可能となる。同様に、演算子“+”

の枝ノードに着目した場合には、入力ポート“ｂ”側の入力に関して、遅延の差分Ｄ（“－”）＝|T_2＋Ｔ（“＋”）－T_{ad}|が挿入される。なお、演算子“＋”の枝ノードから演算結果が出力されるまでの遅延は、Ｔ（“＋”）＝T_1＋T_2＋T_3＋T_{ad}である。

【００７５】

以上のように、ＡＳＴの右側において、各枝ノードへの２つの入力のそれぞれにおける遅延が均等になるように遅延が挿入される。これによって、各枝ノード、すなわちコンポーネント（演算器）における出力タイミングを一致させることが可能となることで、二つの入力のタイミングを合わせることができる。これによって、正確な演算結果が適正なタイミングで出力されるようにすることができる。

【００７６】

＜ＳＰＣの処理フロー＞

図９は、ＣＰＵ１１によって実行されるＳＰＣにおける処理を大略して示すフローチャートである。図９において、ＣＰＵ１１は、例えば、入力装置１４を用いた操作に応じて、ＳＰＣの実行を開始すると、補助記憶装置１３に記憶されたパイプライン処理のプログラム及びモデル定義情報（フローモデルのＸＭＬファイル）を取得する（０１）。

【００７７】

次に、ＣＰＵ１１は、補助記憶装置１３に記憶された、複数のコンポーネントの定義ファイル（ＸＭＬファイル）を取得する（０２）。

【００７８】

次に、ＣＰＵ１１は、フローモデルのＸＭＬファイル中のカーネルプログラム（0penCL）を取り出してコンパイルする。このとき、プログラムからプロセッサＩＤ（id）を指定する構文が見つかると（０２ＡのＹｅｓ）、ＣＰＵ１１は、図４を用いて説明した手法を用いてカウンタの生成を行う（０３）。すなわち、0penCLの“get_global_id”関数を発見したときに、カウンタを生成する。このとき、生成されたカウンタの値は、代入先の変数を使う部分、すなわち、出力データをメモリに書き込むポートに接続する。これによって、出力データの書き込みアドレスがカウンタ値によって制御される。また、カウンタ値は、入力ポートに対するアドレスとして使用することもできる。なお、プロセッサＩＤを含む構文が発見されない場合（０２ＡのＮｏ）には、ＣＰＵ１１は、処理を０４に進める。

【００７９】

次に、ＣＰＵ１１は、図７を用いて説明した手法で、ＡＳＴを作成する（０４）。このとき、作成されたＡＳＴが入出力に係る規定に合致するか否かを判定する（０５）。ＡＳＴが規定に合致しない場合（０５，ＮＧ）には、エラー出力が行われ（０６）、処理が終了する。これによって、設計者（プログラマ）に対し、パイプラインの見直しを行う機会を提供することができる。

【００８０】

これに対し、ＡＳＴが規定に合致する場合（０５，ＯＫ）には、ＣＰＵ１１は、コンポーネント及びポートのマッピングを行う（０７）。すなわち、ＣＰＵ１１は、図６に示したように、複数のコンポーネントの定義ファイルから、ＡＳＴの生成により得られた演算子と演算タイプが合致するコンポーネントを抽出する。このとき、ＣＰＵ１１は、或る演算子について複数のコンポーネントのファイルが抽出された場合には、所定の優先順位の決定ルール（例えば、クロックサイクル数の小さい順）に従って、クロックサイクル数（すなわち遅延）が最も小さいコンポーネントを選択する。そして、選択したコンポーネントを、その定義ファイル中の情報に基づいて演算子にマッピングする。また、ＣＰＵ１１は、或る演算子について１つだけコンポーネントが抽出された場合には、そのまま当該コンポーネントを演算子に対してマッピングする処理を行う。

【００８１】

次に、ＣＰＵ１１は、マッピングされたコンポーネント間の配線接続を、ＡＳＴ及びコンポーネント定義ファイル内の情報に基づき行うことで、複数のコンポーネントを合成する。これによって、ＨＡＭが作成される（０８）。

【００８２】
ＨＡＭが作成されると、次に、ＣＰＵ１１は、遅延（タイミング）調整を行う（０９）。すなわち、ＣＰＵ１１は、図８～図１０を用いて説明した手法で、遅延の挿入を行う。このとき、遅延は、ＤＦＦを用いた遅延フリップフロップや、ゲートを複数段つないだ遅延器のような、所定の遅延回路の記述がＨＡＭに含められる。
【００８３】
そして、ＣＰＵ１１は、ＨＤＬの生成及び出力処理を行う（１０）。ＨＤＬとして、例えば、ＶＨＤＬや Verilog HDLを用いることができる。ここでは、ＶＨＤＬが使用されていると仮定する。
【００８４】
ＣＰＵ１１は、フローモデルのファイルで定義した入出力とプログラムの情報から、ＶＨＤＬにおけるentity宣言を出力する。また、ＣＰＵ１１は、ＨＡＭにおいて使用されているコンポーネントについて、ＶＨＤＬにおけるcomponent宣言を出力する。また、ＣＰＵ１１は、ＡＳＴにおけるノード間の接続（リンク）をsignal（信号線）として宣言し、さらに、architecture宣言の中で、ＡＳＴにおけるコンポーネント（ノード）の接続状態に基づき、パイプライン構造を作成する。このようにして、パイプラインハードウェアのＶＨＤＬによる記述（Hardware Description）が生成される。当該記述は、出力装置１５にて出力（表示又は印刷）されることができる。
【００８５】
図１１は、ＨＤＬによるパイプラインハードウェアの記述例を示す。図１１に示す記述例は、図４の下側に示したパイプラインハードウェアのモデルに対応する。当該記述から、回路図を表すことも可能である。
【００８６】
＜実施形態の効果＞
実施形態によれば、パイプライン処理における演算の内容が記述されたプログラム，パイプライン処理の入出力に係る定義情報，及びコンポーネント情報を用いて、パイプラインハードウェアのＨＤＬによる記述が自動的に作成される。このため、ＧＰＵを用いたパイプライン処理のプログラムをパイプラインハードウェアのＨＤＬ記述に自動的に変換することが可能となる。これによって、ハードウェア設計の作業負担の軽減を図ることができる。
【００８７】
また、上記したように、カウンタの生成によって、少なくともパイプラインの出力に関して簡易なアドレス制御を行うことができ、パイプラインハードウェアの回路構成の複雑化を抑えることができる。また、同タイプの複数のコンポーネントの中から、目的に沿った優先順位が最も高いコンポーネントが選択されるようにすることで、例えば、パイプライン処理に要する時間を最短にすることができる。また、コンポーネントへの２つの入力間で遅延が均等になるように遅延の挿入が行われることで、正確な演算結果が得られるようにすることができる。
【００８８】
このように、実施形態に係るハードウェア設計装置を用いることで、ＧＰＵを用いたシステムよりも回路規模が小さい、すなわち、ＧＰＵシステムよりも小型化及び簡易化されたパイプラインハードウェアを容易に設計することが可能となる。従って、ストリームデータ，及びストリームデータに対する処理の内容に特化した、様々なパイプラインハードウェアを容易に得ることが可能となる。そして、設計されたパイプラインハードウェアの回路規模は小さいため、その適用に係るコストを抑えることができる。
【００８９】
なお、実施形態では、１つの演算式に基づくパイプライン処理を例示したが、２以上の演算式に基づくパイプライン処理に対しても、本実施形態に係るハードウェア設計装置を適用することができる。この場合、或る演算式の次の演算式において、或る演算式の出力が次の演算式の入力として扱われる。このように、ハードウェア設計装置は、複数の演算

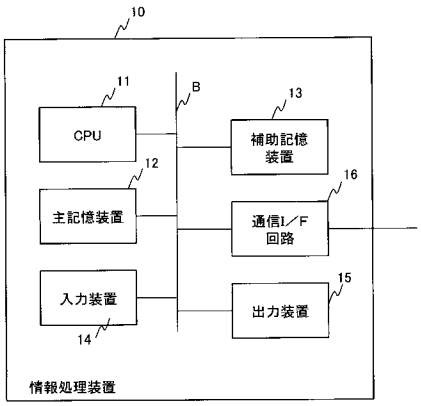
式を含むパイプライン処理に適用可能である。もちろん、１つの演算式に対するパイプラインハードウェアを直列に接続して、複数の演算式に応じたパイプライン処理を行うパイプラインハードウェアを構築することも可能である。

【符号の説明】

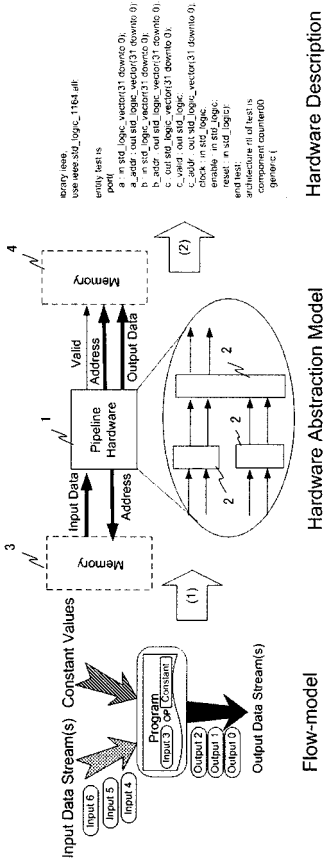
【００９０】

１０・・・情報処理装置（コンピュータ）
１１・・・ＣＰＵ（制御装置）
１２・・・主記憶装置
１３・・・補助記憶装置
１４・・・入力装置
１５・・・出力装置

【図１】



【図２】

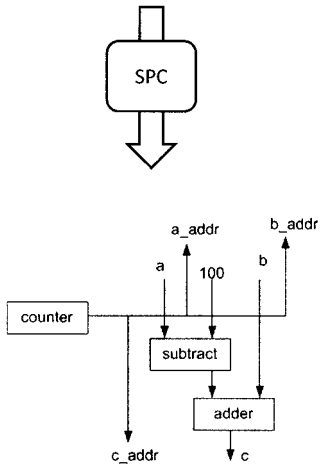


【図３】

```
<?xml version="1.0" encoding="ASCII"?>
<FlowModel xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <Input>
  <Name>a</Name>
  <DataType>INT</DataType>
  <Length>1024</Length>
  <Index>0</Index>
 </Input>
 <Input>
  <Name>b</Name>
  <DataType>INT</DataType>
  <Length>1024</Length>
  <Index>1</Index>
 </Input>

 <Output>
  <Name>c</Name>
  <DataType>INT</DataType>
  <Length>1024</Length>
  <Index>0</Index>
 </Output>

 <Kernel>
  __kernel void test(__global int *a, __global int *b, __global int *c
            )
  {
  int id;
   id = get_global_id(0);
   c[id] = + a[id] -100 + b[id]  ;
   return;
  }
 </Kernel>

 <KernelName>test</KernelName>
 <LangType>SHADERLANG_OPENCL1.0</LangType>
 <RuntimeType>RUNTIME_OPENCLCPU</RuntimeType>
 <Binary />
 <TotalNumThreads>1024</TotalNumThreads>
 <TotalNumBlocks>128</TotalNumBlocks>
 <SwapCounter>0</SwapCounter>
</FlowModel>
```

【図４】

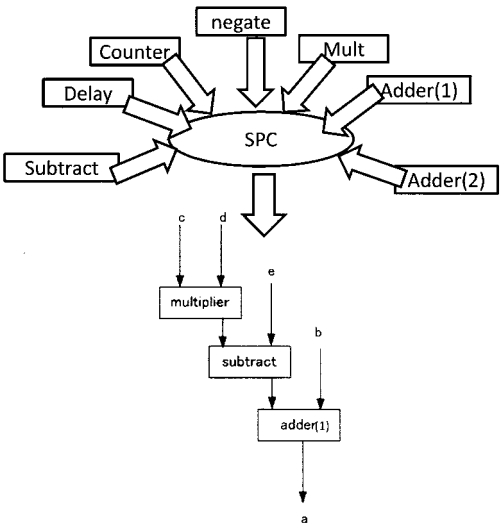
```
__kernel void test(__global int *a, __global int *b, __global int *c)
{
   int id;

   id = get_global_id(0);
   c[id] = + a[id] -100 + b[id]  ;
   return;
}
```



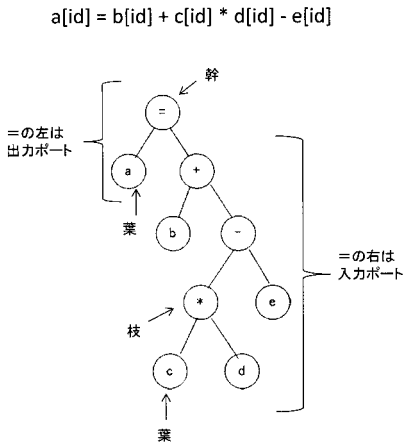
【図５】

```
<?xml version="1.0" encoding="UTF-8"?>
-<Component>
   <Name>sub</Name>    <Category>SUB</Category>
   <RTLFile>sub_2.vhd</RTLFile>
-<Port>
   <Name>a</Name>    <Format>INT</Format>
   <Length>32</Length>    <Direction>INPUT</Direction>
   <Type>DATA_LEFT_TYPE</Type>
 </Port>
-<Port>
   <Name>b</Name>    <Format>INT</Format>
   <Length>32</Length>    <Direction>INPUT</Direction>
   <Type>DATA_RIGHT_TYPE</Type>
 </Port>
-<Port>
  <Name>c</Name>    <Format>INT</Format>
   <Length>32</Length>  <Direction>OUTPUT</Direction>
   <Type>DATA_RESULT_TYPE</Type>
 </Port>
-<Port>
   <Name>enable</Name>    <Format>INT</Format>
   <Length>1</Length>    <Direction>INPUT</Direction>
   <Type>ENABLE_TYPE</Type>
   <Polarity>POSITIVE</Polarity>
 </Port>
-<Port> <Name>valid</Name>    <Format>INT</Format>
   <Length>1</Length>    <Direction>OUTPUT</Direction>
   <Type>VALID_TYPE</Type>  <Polarity>POSITIVE</Polarity>
 </Port>
-<Port>
   <Name>clk</Name>    <Format>INT</Format>
   <Length>1</Length>    <Direction>INPUT</Direction>
   <Type>CLK_TYPE</Type>  <Polarity>POSITIVE</Polarity>
 </Port>
-<Port>
   <Name>reset</Name>    <Format>INT</Format>
   <Length>1</Length>    <Direction>INPUT</Direction>
  <Type>RESET_TYPE</Type>  <Polarity>NEGATIVE</Polarity>
</Port>
 -<GenericPort> <Name>delay</Name> <Format>INT</Format>
  <Length>8  </Length>    <Value>2</Value>
  </GenericPort>
-<Delay> <Enable2Valid>2</Enable2Valid>
</Delay>
</Component>
```

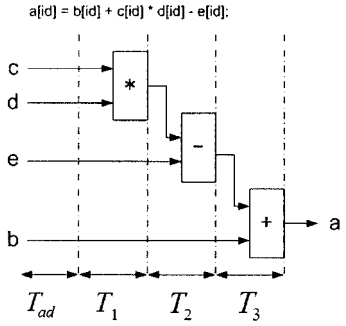
【図６】



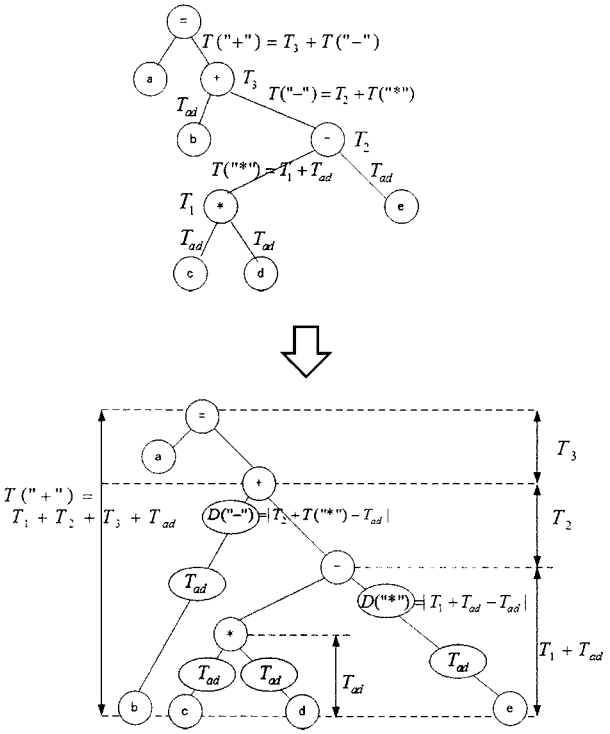
【図７】


a[id] = b[id] + c[id] * d[id] - e[id]
幹
=の左は
出力ポート
=の右は
入力ポート
葉
枝
葉

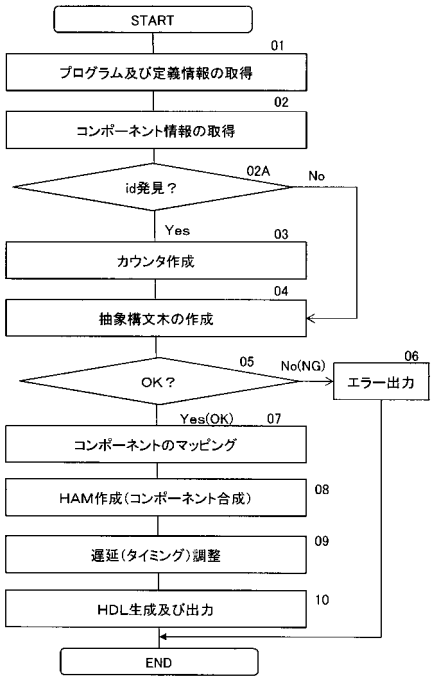
【図８】


a[id] = b[id] + c[id] * d[id] - e[id];
c
d
e
b
a
T_{ad} T_1 T_2 T_3

【図９】


$T("+") = T_3 + T("-")$
$T("-") = T_2 + T("*")$
$T("*") = T_1 + T_{ad}$
$T("+") = T_1 + T_2 + T_3 + T_{ad}$
$D("-") = |T_2 + T("*") - T_{ad}|$
$D("*") = |T_1 + T_{ad} - T_{ad}|$
T_3
T_2
$T_1 + T_{ad}$

【図１０】


START
01
プログラム及び定義情報の取得
02
コンポーネント情報の取得
02A
id発見？
No
Yes
03
カウンタ作成
04
抽象構文木の作成
05
OK？
No(NG)
06
エラー出力
Yes(OK)
07
コンポーネントのマッピング
08
HAM作成(コンポーネント合成)
09
遅延(タイミング)調整
10
HDL生成及び出力
END

【図１１】

```
library ieee;
use ieee.std_logic_1164.all;

entity test is
  port(
    a : in std_logic_vector(31 downto 0);
    a_addr : out std_logic_vector(31 downto 0);
    b : in std_logic_vector(31 downto 0);
    b_addr : out std_logic_vector(31 downto 0);
    c : out std_logic_vector(31 downto 0);
    c_valid : out std_logic;
    c_addr : out std_logic_vector(31 downto 0);
    clock : in std_logic;
    enable : in std_logic;
    reset : in std_logic);
end test;
architecture rtl of test is
  component counter00
    generic (
      max_val : in std_logic_vector(31 downto 0);
      init_val : in std_logic_vector(31 downto 0));
    port (
      nreset : in std_logic;
      clk : in std_logic;
      enable : in std_logic;
      count_out : out std_logic_vector(31 downto 0));
  end component;
  component sub
    generic (
      delay : in std_logic_vector(7 downto 0));
    port (
      nreset : in std_logic;
      clk : in std_logic;
      valid : out std_logic;
      enable : in std_logic;
      c : out std_logic_vector(31 downto 0);
      b : in std_logic_vector(31 downto 0);
      a : in std_logic_vector(31 downto 0));
  end component;
  signal id : std_logic_vector(31 downto 0);
  signal node1 : std_logic_vector(31 downto 0);
  signal node4 : std_logic_vector(31 downto 0);
  signal node6 : std_logic_vector(31 downto 0);
  signal node7 : std_logic;
begin
  Inst2 : counter00
    generic map (
      max_val =>
B"00000000000000000000010000000000",
      init_val =>
B"00000000000000000000000000000000")
    port map (
      nreset => reset,
      clk => clock,
      enable => enable,
      count_out => node1);
  id <= node1;
  c_addr <= id;
  a_addr <= id;
  Inst3 : sub
    generic map (
      delay => B"00000010")
    port map (
      nreset => reset,
      clk => clock,
      valid => node7,
      enable => enable,
      c => node4,
      b => B"00000000000000000000000001100100",
      a => a);
  b_addr <= id;
  Inst5 : add
    generic map (
      delay => B"00000010")
    port map (
      nreset => reset,
      clk => clock,
      valid => c_valid,
      enable => node7,
      c => node6,
      b => b,
      a => node4);
  c <= node6;
end rtl;
```