(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2010-15339
(P2010-15339A)
(43)公開日　平成22年1月21日(2010.1.21)

| (51)Int.Cl. | | F I | | | テーマコード（参考） |
|---|---|---|---|---|---|
| G06F 17/50 | (2006.01) | G06F | 17/50 | 664A | 5B046 |
| | | G06F | 17/50 | 654M | |
| | | G06F | 17/50 | 672A | |

審査請求 未請求　請求項の数 13　ＯＬ　（全 19 頁）

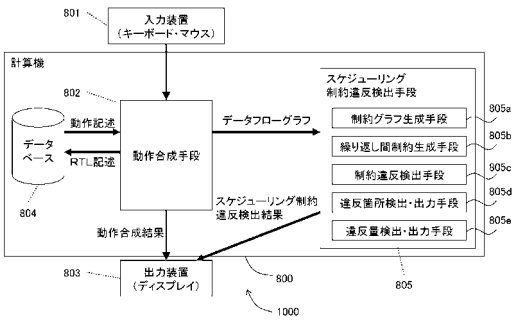
(21)出願番号　特願2008-174004 (P2008-174004)
(22)出願日　平成20年7月2日 (2008.7.2)

(71)出願人　000005049
シャープ株式会社
大阪府大阪市阿倍野区長池町２２番２２号
(74)代理人　100078282
弁理士　山本　秀策
(74)代理人　100062409
弁理士　安村　高明
(74)代理人　100107489
弁理士　大塩　竹志
(72)発明者　岡田　和久
大阪府大阪市阿倍野区長池町２２番２２号
シャープ株式会社内

Ｆターム(参考)　5B046 AA08　BA02　JA01

(54)【発明の名称】動作合成装置、動作合成方法、プログラム、記録媒体、および半導体集積回路の製造方法

(57)【要約】
【課題】動作合成装置において、ループをパイプライン化したときに、回路が正しく動作するかどうかを動作合成システムが判断することを可能とし、正しく動作しないと判断したときには、動作記述のどこを修正すればよいかを設計者に知らせることを可能とする。
【解決手段】動作記述からデータフローグラフを生成し、該データフローグラフに基づいて動作合成を行なってハードウェア回路構成を作成する動作合成装置１０００において、ループ処理をパイプライン化することによりスケジューリング制約違反が生じるかどうかを検出する手段８０５を備えた。
【選択図】図１



【特許請求の範囲】
【請求項１】
回路で実行される演算処理のアルゴリズムを記述した動作記述から、各種演算処理を行う演算処理部を示す各節点と、該節点間でのデータの流れを示す入出力枝とを含むデータフローグラフを生成し、該データフローグラフに基づいて動作合成を行なってハードウェア回路構成を作成する動作合成装置であって、
複数の節点を含むループ処理をパイプライン化したデータフローグラフを作成する動作合成手段と、
該ループ処理のパイプライン化により該データフローグラフのスケジューリング制約違反が発生するか否かを検出するスケジューリング違反検出手段とを備えた動作合成装置。
【請求項２】
請求項１に記載の動作合成装置において、
前記ループ処理における第１節点で生成した値を、該ループ処理のｎ回後の繰り返し時に、該ループ処理における第２節点で使用するとき、該ループ処理の繰り返し時間間隔であるスループットの値をｔｐとして、該第１節点での演算開始から該第２節点での演算開始までに必要な時間である重みに対してｔｐ×ｎの値を減算して繰り返し間制約を生成する繰り返し間制約生成手段を備えた動作合成装置。
【請求項３】
請求項２に記載の動作合成装置において、
前記繰り返し間制約生成手段で生成した繰り返し間制約を含む、演算の順序と時間関係を表す制約グラフ中に、前記重みの合計が正である閉ループがあるか否かの判断により制約違反があるかを判断する制約違反検出手段を備えた動作合成装置。
【請求項４】
請求項２に記載の動作合成装置において、
生成した繰り返し間制約枝のうち１つを前記制約グラフから削除し、該削除により制約違反がなくなればその枝がスケジューリング制約違反の原因と判断し、あるいは該削除後に制約違反があれば、別の繰り返し間制約枝を該制約グラフから削除する処理の繰り返しにより、スケジューリング制約違反の原因箇所を検出する原因箇所検出手段を備えた動作合成装置。
【請求項５】
請求項１の動作合成装置において、
前記スケジューリング制約違反の原因となっている繰り返し間制約枝に対して、前記重みを１減らし、該重み削減により、制約違反がなくなれば減らした重みの合計値をスケジューリング制約違反の量として検出し、あるいは、該重み削減後に制約違反がまだあるならば、さらに重みを１減らす処理の繰り返しにより、スケジューリング制約違反量を検出する制約違反量検出手段を備えた動作合成装置。
【請求項６】
回路で実行される演算処理のアルゴリズムを記述した動作記述から、各種演算処理を行う演算処理部を示す各節点と、該節点間でのデータの流れを示す入出力枝とを含むデータフローグラフを生成し、該データフローグラフに基づいて動作合成を行なってハードウェア回路構成を作成する動作合成方法であって、
複数の節点を含むループ処理をパイプライン化したデータフローグラフを作成する動作合成ステップと、
該ループ処理のパイプライン化により該データフローグラフのスケジューリング制約違反が発生するか否かを検出するスケジューリング違反検出ステップとを含む動作合成方法。
【請求項７】
請求項６に記載の動作合成方法において、
前記ループ処理における第１節点で生成した値を、該ループ処理のｎ回後の繰り返し時に、該ループ処理における第２節点で使用するとき、該ループ処理の繰り返し時間間隔で

あるスループットの値をｔｐとして、該第１節点での演算開始から該第２節点での演算開始までに必要な時間である重みに対してｔｐ×ｎの値を減算して繰り返し間制約を生成する繰り返し間制約生成ステップを含む動作合成方法。
【請求項８】
請求項７に記載の動作合成方法において、
前記繰り返し間制約生成工程で生成した繰り返し間制約を含む、演算の順序と時間関係を表す制約グラフ中に前記重み合計が正である閉ループがあるか否かの判断により制約違反があるかを判断する制約違反検出工程を含む動作合成方法。
【請求項９】
請求項７に記載の動作合成方法において、
生成した繰り返し間制約枝のうち１つを前記制約グラフから削除し、該削除により制約違反がなくなればその枝がスケジューリング制約違反の原因と判断し、あるいは該削除後に制約違反があれば、別の繰り返し間制約枝を該制約グラフから削除する処理の繰り返しにより、スケジューリング制約違反の原因箇所を検出する原因箇所検出工程を含む動作合成方法。
【請求項１０】
請求項６に記載の動作合成方法において、
制約違反の原因となっている繰り返し間制約枝に対して、前記重みを１減らし、該重み削減により、制約違反がなくなれば減らした重みの合計値をスケジューリング制約違反の量として検出し、あるいは、該重み削減後に制約違反がまだあるならば、さらに重みを１減らす処理の繰り返しにより、スケジューリング制約違反量を検出する制約違反量検出工程を含む動作合成方法。
【請求項１１】
請求項６から１０のいずれかに記載の動作合成方法の各ステップをコンピュータに実行させるための処理手順が記述されたプログラム。
【請求項１２】
請求項１１に記載のプログラムが格納されたコンピュータ読み取り可能な可読記憶媒体。
【請求項１３】
回路情報に基づいて半導体集積回路を製造する方法であって、
該回路情報は、請求項１～５のいずれかに記載の動作合成装置により作成されたハードウエハ回路構成を示すものである半導体集積回路の製造方法。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、動作合成装置、動作合成方法、プログラム、記録媒体、および半導体集積回路の製造方法に関し、特に、論理回路等の設計に用いられ、動作記述から論理回路を自動合成する動作合成装置において、ループ処理のパイプライン化時の制約違反検出機能を付加したもの、このような動作合成装置に対応する動作合成方法、該動作合成方法をコンピュータにより行うためのプログラム、このプログラムを格納したコンピュータ読み取り可能な記憶媒体、および動作合成装置により作成されたハードウエハ回路構成を示す回路情報に基づいて半導体集積回路を製造する方法に関するものである。
【背景技術】
【０００２】
システムＬＳＩ等の大規模回路の設計においては、回路の動作記述からＲＴＬ（Ｒｅｇｉｓｔｅｒ　Ｔｒａｎｓｆｅｒ　Ｌｅｖｅｌ：レジスタトランスファレベル）記述を生成する、動作合成が行なわれる。動作合成は高位合成とも呼ばれる。動作合成では、ハードウェアの構造に関する情報を含まずに、処理のアルゴリズムのみを記述した動作記述から、ハードウェアが自動的に合成される。以下、従来の動作合成技術を用いて、動作記述からハードウエア構成を自動的に合成する過程について、簡単に説明する。

【０００３】
動作合成は図１０に示す手順で行なわれる。
【０００４】
まず、図１０に示すデータフロー生成では、アルゴリズム記述におけるデータの流れを解析し、データフローグラフと呼ばれるモデルを作成する（ステップＳ１１）。データフローグラフは節点と枝より成る。枝はデータを表し、節点は演算を表す。各節点は、入力枝と出力枝によって接続され、入力枝が演算に与えられるデータ、出力枝が演算結果のデータを表す。また、各節点は演算の種類の情報も持っている。
【０００５】
例えば、次のＣ言語で記述された動作記述は、図１１に示すデータフローグラフで表される。
【０００６】
ｘ　＝　ａ＊ｂ　＋　ｂ＊ｃ；
図１１では、データフローグラブＧｆ２は、２つの乗算を表す節点２１、２２と、１つの加算を表す節点２３を含み、節点２１で入力データａと入力データｂとを乗算して得れらた結果と、節点２２で入力データｂと入力データｃを乗算して得られた結果とが節点２３で加算され、その加算結果が出力データｘとして出力されることが表されている。
【０００７】
データフローグラフＧｄｆは、計算機上では、例えば、図１２に示すようなデータ構造で表される。図１２によると、節点はＮｏｄｅ構造体Ｎｓで表され、各節点固有の節点番号ｎｏｄｅ＿ｉｄを持つ。また、ｉｎ＿ｅｄｇｅ，ｏｕｔ＿ｅｄｇｅ配列には入力枝と出力枝の枝番号が複数記憶される。図１２に示す例では、節点は２入力１出力の演算であり、ｉｎ＿ｅｄｇｅは２つの要素、ｏｕｔ＿ｅｄｇｅは１つの要素を持つ。また、ｏｐ＿ｔｙｐｅには加算・減算・乗算などの演算の種類を表す番号が記憶される。
【０００８】
枝はＥｄｇｅ構造体Ｅｓで表され、各枝固有の枝番号ｅｄｇｅ＿ｉｄを持つ。ｆｒｏｍ＿ｎｏｄｅ，ｔｏ＿ｎｏｄｅには、その枝が接続された節点の節点番号が格納される。
【０００９】
これらのデータ構造により、計算機上のメモリ上にデータフローグラフの各節点間の接続が記憶される。また、データフローグラフＧｄｆ上で、節点を接続する場合や、ある節点の入出力につながる別の節点を見つける場合は、上記データベースの各要素に節点や枝の番号を登録したり、参照したりする。
【００１０】
以下では説明をわかりやすくするため、データフローグラフを視覚的に表した図１１のような図を用いて、節点を枝で接続したり、節点の上下につながる節点を見つけたりすることでデータフローグラフについて説明する。
【００１１】
次に、データフローグラフに対して、図１０に示す、スケジューリング（ステップＳ１２）やアロケーション処理（ステップＳ１３）を行なう。スケジューリングは、データフローグラフ中の各節点をいつ実行するかを決める処理である。アロケーションはバインディングとも呼ばれ、データフローグラフ中の枝が表すデータを格納するレジスタを決める処理と、データフローグラフの節点が表す演算をどの演算器を用いて行なうかを決定する処理とからなる。動作合成方法によっては、アロケーションがスケジューリングの前に行なわれる場合もある。
【００１２】
次に、図１０に示すデータパス生成では、スケジューリング結果とアロケーション結果を元に、演算器とレジスタを、必要に応じてセレクタを入れて接続し、データパスを得る（ステップＳ１４）。図１０に示すコントローラ生成では、スケジューリング結果とアロケーション結果を元に、データパス中のレジスタの読み込みタイミングやセレクタによる選択を制御するステートマシンを生成する（ステップＳ１５）。データパス生成とコント

ローラ生成はどちらを先にやってもよい。最後にデータパスとコントローラを接続し、所望のハードウェアを得る。

【００１３】

データフローグラフから回路を得る処理の詳細の例が特許文献１に示されている。

【００１４】

実際の回路では、多くの場合、動作記述中にループ処理が含まれる。ループ処理は多数回繰り返し実行されるため、回路全体の処理時間のうちの多くはループ処理が占めることになる。よって、回路の処理を高速化したい場合には、ループ処理を高速化するのが効果的である。画像処理や音声処理などのように、ある一定の時間内に処理を終えねばならない高速性が要求される回路の場合は、このような高速化は必須である。

【００１５】

このようなループ処理の問題を解決する方法の一つとして、ループ処理をパイプライン化する方法がある。

【００１６】

従来のループ処理のパイプライン化方法の一例が特許文献２に示されている。この方法を簡単に説明する。

【００１７】

例えば、図１３に示すようなループ処理を含む動作記述を考える。図１３に示す動作記述は、ｉ＝０を一つづつインクリメントしながらｉ＜１０の条件を満たす間に、ｓｕｍ＝ｓｕｍ＋ｘ（ｉ）、ｓｕｍ＝ｓｕｍ＋ｙ（ｉ）、ｓｕｍ＝ｓｕｍ＋ｚ（ｉ）の各式で示される演算を行う処理を表す動作記述である。図１４は、図１３に示す動作記述のデータフローグラフである。

【００１８】

矩形５０１はループ処理を表し、これは、矩形５０１内のデータフローグラフＧｆ５が繰り返し実行されることを意味する。この矩形５０１の上端と下端にある丸印５０９、５１０、５１１、５１２はループ処理のポートを表し、ポート５０９及び５１０が図１３に示す変数ｉに、ポート５１１及び５１２が図１３に示す変数ｓｕｍに対応する。ポート５０９に格納された変数ｉの値がインクリメント演算節点５０８でインクリメントされて１つづつ増加するように変化し、ポート５１０に至る。ポート５０９とポート５１０は同じ変数を表し、ポート５１０に至ったデータは、次の繰り返しで再びポート５０９に戻されて使用される。図１３に示す変数ｓｕｍについても同様にポート５１１及び５１２が使用される。節点５０２は配列ｘの値ｘ（ｉ）を読み出す処理（Ｒｘ）を表し、入力が配列の要素番号、出力は要素番号の配列の値である。節点５０３及び５０４も同様に配列ｙ，ｚの値ｙ（ｉ）、ｚ（ｉ）を読み出す処理（Ｒｙ）、（Ｒｚ）を表す。節点５０５、５０６、及び５０７は加算処理を表し、２つの入力値の加算結果を出力する。図１４に示すデータフローグラフＧｆ５はこれらの節点のほか、ループの終了をコントロールする比較節点なども含むべきであるが、説明の簡単化のため省略する。

【００１９】

ここで、ループ処理５０１は、４ステップで１回の処理を行うようにスケジューリングされている。図中の点線はステップの境目を表す。スケジューリングは、クロックピリオドや演算節点での遅延などを考慮して行われ、クロックピリオドが十分長ければ、配列の読み出しと加算を１ステップで処理することも可能である。このような複数の処理を１ステップで実行することはチェイニングと呼ばれる。図１４のスケジューリングでは、配列の読み出しと加算の遅延合計がクロックピリオドを超えるため、チェイニングをせず、それぞれ別のステップに割り当てている。

【００２０】

図１３に示す動作記述が表すループ処理が図１４に示すようにスケジューリングされた場合、１ステップの実行に１サイクルが必要で、ループ処理の１回の繰り返しに４サイクルかかるため、ループ処理を１０回繰り返すには４０サイクル必要である。ここで、ループをパイプライン化して実行することにより、より少ないサイクル数で処理を完了するこ

とが可能である。

【００２１】

図１５は、図１４に示すループ処理をスループット３で実行する場合の実行タイミングを表す。ここで、スループットとは、ループ処理のある繰り返しが開始されるクロックサイクルと、ループ処理のその次の繰り返しが開始されるクロックサイクルとの間隔である。また、ループ処理の１回の繰り返しを全て実行するのに必要なクロックサイクル数をレイテンシと呼ぶ。ループ処理の０ステップ目では、配列ｘの値の読出処理（Ｒｘ）５０２が実行され、図１５でもループの０ステップ目の実行箇所にＲｘの記号を記す。ループの１ステップ目では、配列ｙの値の読出処理（Ｒｙ）５０３と加算（＋）５０５が実行されるが、図１５では代表してＲｙと記す。同様に、ループの２ステップ目は配列ｚの値の読出処理（Ｒｚ）を、ループ処理の３ステップ目には加算処理（＋）５０７とインクリメント（＋１）５０８の２つが実行されるが＋を１つだけ記す。

【００２２】

図１５に示すとおり、サイクル１でループの１回目の繰り返し（ｉ　＝　０）が始まり、サイクル１で配列ｘの値Ｒｘ（０）を読み、サイクル２でｓｕｍに加算しつつ配列ｙの値Ｒｙ（０）を読み、サイクル３で変数ｓｕｍに加算しつつ配列ｚの値Ｒｚ（０）を読み出す。サイクル４では、配列ｚの値Ｒｚ（０）を変数ｓｕｍに加算するが、同時にループの２回目の繰り返し（ｉ　＝　１）が始まり、配列ｘの値Ｒｘ（１）の読み出しも行う。これを繰り返すことにより、ループ処理の１０回の繰り返しを３１サイクルで終了することができる。このように、ループ処理の１回の繰り返しが終了する前にループ処理の次の繰り返しを開始することにより、全体の処理（つまり既定数のループ処理の繰り返し）に必要なサイクル数を削減することが可能である。

【００２３】

図１４に示すデータフローグラフを図１５に示すようなタイミングで実行する回路を得る方法は、例えば特許文献３に記載されている。

【特許文献１】特開２００１－２２９２１７号公報

【特許文献２】特開２００１－１４２９３７号公報（図３）

【特許文献３】特開２００４－３２６４６３号公報

【発明の開示】

【発明が解決しようとする課題】

【００２４】

しかしながら、ループ処理をパイプライン化して実行しようとすると、処理を正しく行えなくなる場合がある。

【００２５】

例えば、図１６は、図１４に示すデータフローグラフＧｆ５のループ処理をスループット１で実行する場合のタイミングを示している。ループの１回目の繰り返し（ｉ　＝　０）がサイクル１で開始し、ループの２回目の繰り返し（ｉ　＝　１）がサイクル２で開始するなど、ループの各繰り返しを１サイクルおきに開始する。もしこのタイミングで処理を行うことが可能であるならば、ループ処理の１０回の繰り返しを１３サイクルで終了することができる。しかし、図１４に示すループ処理では、そのある繰り返しにおいて計算した変数ｓｕｍの値は、ポート５１１、５１２を通じて、該ループ処理の次の繰り返しで使用されるため、図１６に示すタイミングでループを実行することは不可能となる。

【００２６】

すなわち、１回目の繰り返しで変数ｓｕｍの値は、サイクル４（ループ処理（ｉ＝０）のステップ３）で計算するにもかかわらず、２回目の繰り返しではサイクル３（ループ処理（ｉ＝１）のステップ１）で変数ｓｕｍの値を使用してしまっており、ループ処理のある繰り返しにおける変数ｓｕｍの値の計算が、ループ処理の次の繰り返しでの使用に間に合わない。このため、図１４に示すデータフローグラフをスループット１でパイプライン化して実行しようとした回路は正しく動作しなくなってしまう。

【００２７】

このように、図１３に示す記述は、スループット１で処理することは不可能であるが、設計者がこれを直感的に判断することは難しい。さらに、実際の設計においては、回路は複数のループ記述を含み、それぞれのループが複雑な処理を含んでおり、各ループが正しく動作するかを設計者が判断するのは非常に困難である。

【００２８】

本発明は、上記のような課題を解決するためになされたもので、ループ処理をパイプライン化したときに該ループ処理を正しく実行できるかどうかを自動的に判断することができる動作合成装置及び動作合成方法、このような動作合成装置あるいは動作合成方法を用いて設計された回路情報に基づいて半導体集積回路を製造する製造方法、該動作合成方法をコンピュータにより実行するための動作合成プログラム、およびこのような動作合成プログラムを格納した記録媒体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】

【００２９】

本発明に係る動作合成装置は、回路で実行される演算処理のアルゴリズムを記述した動作記述から、各種演算処理を行う演算処理部を示す各節点と、該節点間でのデータの流れを示す入出力枝とを含むデータフローグラフを生成し、該データフローグラフに基づいて動作合成を行なってハードウェア回路構成を作成する動作合成装置であって、複数の節点を含むループ処理をパイプライン化したデータフローグラフを作成する動作合成手段と、該ループ処理のパイプライン化により該データフローグラフのスケジューリング制約違反が発生するか否かを検出するスケジューリング違反検出手段とを備えたものであり、そのことにより上記目的が達成される。

【００３０】

本発明は、上記動作合成装置において、前記ループ処理における第１節点で生成した値を、該ループ処理のｎ回後の繰り返し時に、該ループ処理における第２節点で使用するとき、該ループ処理の繰り返し時間間隔であるスループットの値をｔｐとして、該第１節点での演算開始から該第２節点での演算開始までに必要な時間である重みに対してｔｐ×ｎの値を減算して繰り返し間制約を生成する繰り返し間制約生成手段を備えていることが好ましい。

【００３１】

本発明は、上記動作合成装置において、前記繰り返し間制約生成手段で生成した繰り返し間制約を含む、演算の順序と時間関係を表す制約グラフ中に、前記重みの合計が正である閉ループがあるか否かの判断により制約違反があるかを判断する制約違反検出手段を備えていることが好ましい。

【００３２】

本発明は、上記動作合成装置において、生成した繰り返し間制約枝のうち１つを前記制約グラフから削除し、該削除により制約違反がなくなればその枝がスケジューリング制約違反の原因と判断し、あるいは該削除後に制約違反があれば、別の繰り返し間制約枝を該制約グラフから削除する処理の繰り返しにより、スケジューリング制約違反の原因箇所を検出する原因箇所検出手段を備えていることが好ましい。

【００３３】

本発明は、上記動作合成装置において、制約違反の原因となっている繰り返し間制約枝に対して、前記重みを１減らし、該重み削減により、制約違反がなくなれば減らした重みの合計値をスケジューリング制約違反の量として検出し、あるいは、該重み削減後に制約違反がまだあるならば、さらに重みを１減らす処理の繰り返しにより、スケジューリング制約違反量を検出する制約違反量検出手段を備えていることが好ましい。

【００３４】

本発明に係る動作合成方法は、回路で実行される演算処理のアルゴリズムを記述した動作記述から、各種演算処理を行う演算処理部を示す各節点と、該節点間でのデータの流れを示す入出力枝とを含むデータフローグラフを生成し、該データフローグラフに基づいて動作合成を行なってハードウェア回路構成を作成する動作合成方法であって、複数の節点

を含むループ処理をパイプライン化したデータフローグラフを作成する動作合成ステップと、該ループ処理のパイプライン化により該データフローグラフのスケジューリング制約違反が発生するか否かを検出するスケジューリング違反検出ステップとを含むものであり、そのことにより上記目的が達成される。

【００３５】

本発明は、上記動作合成方法において、前記ループ処理における第１節点で生成した値を、該ループ処理のｎ回後の繰り返し時に、該ループ処理における第２節点で使用するとき、該ループ処理の繰り返し時間間隔であるスループットの値をｔｐとして、該第１節点での演算開始から該第２節点での演算開始までに必要な時間である重みに対してｔｐ×ｎの値を減算して繰り返し間制約を生成する繰り返し間制約生成ステップを含むことが好ましい。

【００３６】

本発明は、上記動作合成方法において、前記繰り返し間制約生成工程で生成した繰り返し間制約を含む、演算の順序と時間関係を表す制約グラフ中に前記重み合計が正である閉ループがあるか否かの判断により制約違反があるかを判断する制約違反検出工程を含むことが好ましい。

【００３７】

本発明は、上記動作合成方法において、生成した繰り返し間制約枝のうち１つを前記制約グラフから削除し、該削除により制約違反がなくなればその枝がスケジューリング制約違反の原因と判断し、あるいは該削除後に制約違反があれば、別の繰り返し間制約枝を該制約グラフから削除する処理の繰り返しにより、スケジューリング制約違反の原因箇所を検出する原因箇所検出工程を含むことが好ましい。

【００３８】

本発明は、上記動作合成方法において、制約違反の原因となっている繰り返し間制約枝に対して、前記重みを１減らし、該重み削減により、制約違反がなくなれば減らした重みの合計値をスケジューリング制約違反の量として検出し、あるいは、該重み削減後に制約違反がまだあるならば、さらに重みを１減らす処理の繰り返しにより、スケジューリング制約違反量を検出する制約違反量検出工程を含むことが好ましい。

【００３９】

本発明に係るプログラムは、上述した本発明に係る動作合成方法の各ステップをコンピュータに実行させるための処理手順が記述されたものであり、そのことにより上記目的が達成される。

【００４０】

本発明に係る記録媒体は、上述した本発明のプログラムが格納されたコンピュータ読み取り可能な可読記憶媒体であり、そのことにより上記目的が達成される。

【００４１】

本発明に係る半導体集積回路の製造方法は、回路情報に基づいて半導体集積回路を製造する方法であって、該回路情報は、上述した本発明に係る動作合成装置により作成されたハードウエハ回路構成を示すものであり、そのことにより上記目的が達成される。

【００４２】

以下、本発明の作用について説明する。

【００４３】

本発明においては、動作記述からデータフローグラフを生成し、該データフローグラフに基づいて動作合成を行なってハードウェア回路構成を作成する動作合成装置において、ループ処理をパイプライン化することによりスケジューリング制約違反が生じるかどうかを検出する手段を備えたので、ループ処理をパイプライン化したときに該ループ処理を正しく実行できるかどうかを自動的に判断することができる。

【００４４】

また、本発明においては、ループ処理における第１節点で生成した値を、ループ処理のｎ回後の繰り返し時に、該ループ処理における第２節点で使用する場合、ループ処理のス

ループット値をｔｐとして、第１節点から第２節点への重みに対してｔｐ×ｎの値を減算して繰り返し間制約とし、また、繰り返し間制約を含む制約グラフ中に重み合計が正の閉ループがあるかを判断し、制約違反があるかを判断する機能を提供するので、繰り返し間制約枝のうち１つを制約グラフから削除し、もし、制約違反がなくなればその枝がスケジューリング制約違反の原因と判断し、もし、まだ制約違反があれば、別の繰り返し間制約枝に対して同じ処理を繰り返すことにより、スケジューリング制約違反の原因箇所を検出することができる。

【００４５】

また、制約違反の原因となっている繰り返し間制約枝に対して、重みを１減らし、もし、制約違反がなくなれば減らした重みの合計値をスケジューリング制約違反の量として検出し、もし、制約違反がまだあるならば、さらに重みを１減らして同じ処理を繰り返すことにより、スケジューリング制約違反量を検出することができる。

【発明の効果】

【００４６】

本発明によれば、ループ処理をパイプライン化したときに、回路が正しく動作するかどうかを動作合成システムが判断することができ、正しく動作しないと判断したときには、動作記述のどこを修正すればよいかを設計者に知らせることが可能となる。これにより、設計効率を高めることができる。

【発明を実施するための最良の形態】

【００４７】

以下、本発明の実施形態について図面を参照しながら説明する。

【００４８】

（実施形態１）

以下、本発明の実施の形態を図面を用いて説明する。

【００４９】

図１は、本発明の実施形態１による動作合成装置を説明するブロック図である。

【００５０】

この実施形態１の動作合成装置１０００は、ループをパイプライン化することによりスケジューリング制約違反が生じるかどうかを検出するスケジューリング制約違反検出手段８０５を含むものであり、以下この動作合成装置１０００について詳述する。

【００５１】

本実施形態の動作合成装置１０００は、コンピュータシステムを用いて構成したものであり、各種演算処理を行う計算機８００と、該計算機８００に対してデータの入力、指令などを行うためのキーボード，マウスなどの入力装置８０１と、該計算機８００で得られたデータを設計者などのオペレータに分かるよう出力する出力装置８０３とを有している。

【００５２】

上記計算機８００は、動作記述に基づいてデータフローグラフを作成するとともに、ＲＴＬ記述のハードウエア回路構成を作成する動作合成手段８０２と、上記動作記述およびＲＴＬ記述を格納するデータベース８０４と、該データフローグラフに基づいてスケジューリング制約違反を検出するスケジューリング制約違反検出手段８０５とを備えている。

【００５３】

ここで、動作合成手段８０２は、入力装置８０１からの設計者の指示に従い、データベース８０４中の動作記述を動作合成し、具体的な回路構成を示すＲＴＬ記述を生成してデータベース８０４に出力し、また、動作合成により得られた回路構成の概要を示す動作合成結果を出力装置８０３に出力するものである。設計者は該出力装置８０３から出力される動作合成結果に基づいて設計結果を確認することができる。この動作合成手段８０２による処理は、従来の動作合成装置と同様の処理である。

【００５４】

本実施形態の動作合成装置１０００は、上述したようにスケジューリング制約違反検出

手段８０５を備えており、上記動作合成手段８０２は、データフローグラフを生成した後、スケジューリング制約違反検出手段８０５に、生成したデータフローグラフの情報を渡すよう構成されている。

【００５５】

このスケジューリング制約違反検出手段８０５は、スケジューリング制約違反箇所を特定し、スケジューリング制約違反結果を出力するものである。ここで、上記動作合成手段８０２及びスケジューリング制約違反検出手段８０５による処理は、計算機上で実行される。これらの手段による処理は、図１に示すように同一の計算機上で実行してもよいし、別の計算機上で独立して実行してもよい。また、データベース８０４を構成する記憶媒体も、図１に示すように計算機８００の中に含まれてもよいし、計算機８００の外部デバイスとして設けてあってもよい。

【００５６】

上記スケジューリング制約違反検出手段８０５は、制約グラフを生成する手段８０５ａと、繰り返し間制約を生成する手段８０５ｂと、制約違反を検出する手段８０５ｃと、違反箇所を検出して出力する手段８０５ｄと、違反量を検出して出力する手段８０５ｅとを有している。

【００５７】

次に動作について説明する。

図２は、本実施形態の動作合成装置１０００におけるスケジューリング制約違反検出手段８０５の処理フローについて説明する図である。

【００５８】

まず、制約グラフ生成手段８０５ａは、動作合成手段８０２から入力されたデータフローグラフから制約グラフを生成する（ステップＳ９０１）。図３（ｂ）は制約グラフＧｒ１０の一例を示す。

【００５９】

制約グラフＧｒ１０は節点３１～３５とこれらの節点間の枝とからなり、節点はデータフローグラフ中の演算を示す。枝は、節点から節点への有向枝であり、重みを持つ。

【００６０】

例えば、ある節点Ａからある節点Ｂへの枝の重みが２の場合、節点Ａに対応する演算の開始後、２サイクル以上後で、節点Ｂの演算を開始することを示す。データフローグラフは節点間のデータの流れを表すのに対して、制約グラフは節点の演算の順序と時間関係のみを表すグラフである。制約グラフには特別な節点（ｓ）３１と節点（ｔ）３５とが含まれており、それぞれソースとシンクと呼び、処理の開始と終了を表す。制約グラフもデータフローグラフと同様の図１２に示すような方法で、計算機上で表される。制約グラフの場合は枝に重みを表す項目を追加する必要がある。

【００６１】

図３（ｂ）に示す制約グラフＧｒ１０を数式で表すと、図３（ａ）に示す線形計画問題と同じ意味となる。制約グラフでソース節点（ｓ）３１から他の各節点への最長経路を求めることにより、各節点の処理開始時間を求めることができる。なお、図３（ａ）において、ｔは最小化される。

【００６２】

例えば、図３（ｂ）に示す節点（ｓ）３１から節点（Ｚ）３５への最長経路は、ｓ→Ｘ→Ｙ→Ｚの経路で経路長は３である。よって、節点（Ｚ）３５を３サイクル目で実行すれば、節点Ｘや節点Ｙで生成される、節点Ｚの入力となるべきデータは全てそろっており、正しい処理が可能である。制約グラフにおける節点間の最長経路はダイクストラの方法などにより簡単に求めることができる。ソースからの最長経路で求めた結果（つまり、各節点の処理開始時間）は、各節点の処理をできるだけ早く開始するスケジューリング結果となるので、ＡＳＡＰ（Ａｓ　Ｓｏｏｎ　Ａｓ　Ｐｏｓｓｉｂｌｅ）スケジューリングと呼ばれる。

【００６３】

図１３に示す動作記述の制約グラフは、図４（ａ）のようになる。ここで、配列アクセス（配列の値の読出し）や加算の遅延を全て、簡単のため１としている。図４（ｂ）は、図４（ａ）の制約グラフＧｆ１１に対して、クロックピリオドを１とした場合のＡＳＡＰスケジューリング結果である。このスケジューリングでは、最初に全ての配列の値を読んでしまうため、レジスタが多く必要となってしまうが、レジスタ数や演算器数も考慮したより複雑なスケジューリング手段によると、図１４のようなスケジューリング結果を得ることも可能である。

【００６４】

しかし、本発明は、スケジューリング制約違反を検出するのが目的であるため、短時間で求まるＡＳＡＰスケジューリングで十分である。ＡＳＡＰスケジューリングも他の全てのスケジューリング手法も、制約グラフにおける各節点での制約を全て満たす解のうちの１つを選ぶものである。よって、ＡＳＡＰスケジューリングで全ての節点での制約を満たす解が存在しないことがわかれば、どのようなスケジューリング手法を用いても解は存在しないし、逆に、ＡＳＡＰスケジューリングで解があるとわかれば、他のスケジューリング手法でも解は必ず見つかる。

【００６５】

次に、図２に示す繰り返し間制約生成（ステップＳ９０２）について説明する。

【００６６】

ループ処理のスループットが指定されていない場合（スループットとレイテンシが等しい場合）には、図４（ａ）に示す制約グラフＧｆ１１でよいが、レイテンシよりも小さいスループットが指定された場合は、ループ処理の繰り返し間依存制約が発生するため、それに対する枝を制約グラフに加えなければならない。図１４に示すデータフローグラフＧｆ５の場合、ポート５１０とポート５１２の値がそれぞれ次の繰り返しで、ポート５０９及び５１１で使用されるため、繰り返し間依存が発生する。

【００６７】

図５に繰り返し間制約生成の一例を示す。図５（ａ）は、ループ処理のデータフローグラフＧｆ１２を表し、節点（Ｂ）１２０４で作られた値は、ポート１２０２及び１２０１を通じて、次の繰り返しで節点（Ａ）１２０３で使用される。

【００６８】

ここで、ループ処理のスループットを３、レイテンシを５、節点（Ｂ）１２０４の遅延を１とすると、図５（ｂ）に示すように、３サイクル目で実行される節点（Ｂ）１２０６で生成する値を使用する次の繰り返しの節点（Ａ）１２０７は、節点（Ｂ）の遅延が１のため、４サイクル目以降で実行されねばならない。これを同一繰り返し間での制約条件で考えると、節点（Ａ）１２０７に対して、１回前の節点（Ａ）１２０５は、スループット値のサイクル数前に実行されるので、節点（Ｂ）の遅延１からスループット値３を減算し、つまり、節点（Ａ）１２０５は節点（Ｂ）１２０６の１－３＝－２サイクル後以降で実行せねばならないことがわかる。言い換えると、節点（Ａ）１２０５は、節点（Ｂ）１２０６の２サイクル前以降で実行せねばならない。これを制約グラフＧｒ１２で表したのが図５（ｃ）である。

【００６９】

ここで、図５（ｂ）では、節点（Ｂ）が３サイクル目で実行されるとしたが、節点（Ｂ）をどこのサイクルで実行しても節点（Ａ）との相対関係は同じであるので、制約条件は変わらない。また、ループのレイテンシも制約条件には無関係である。よって、図５（ｃ）の繰り返し間依存制約は、節点（Ｂ）の遅延とループ処理のスループットのみで決まる値である。この繰り返し間依存の例は、ある繰り返しで作った値を次の繰り返しで使う場合であるが、次の次の繰り返しで使う場合にはスループット値の２倍を引けばよいし、同様にｎ回後の繰り返しで使う場合は、スループット値のｎ倍を引けばよい。

【００７０】

図１４に示すデータフローグラフＧｆ５の繰り返し間依存制約を図４（ａ）の制約グラフＧｒ１１に加えた結果を図６に示す。図６の太線の枝が繰り返し間依存制約を表す。こ

こでスループットの値をｔｐとしている。

【００７１】

次に図２に示す制約違反検出（ステップＳ９０３）について説明する。

【００７２】

図２に示す繰り返し間依存制約（ステップＳ９０２）は、他の通常の制約と全く同じに扱うことができるため、図６に示す制約グラフＧｒ１３から従来と同じ方法でＡＳＡＰスケジューリングを求めることができる。しかし、スループットの値によってはＡＳＡＰスケジューリングが求まらない場合がある。

【００７３】

例えば、図６でスループット値ｔｐ＝１の場合を考える。ソース節点（ｓ）１３００から節点（＋）１３０１への最長経路長を求めようとすると、節点（ｓ）１３００から節点（Ｒｘ）１３０４を経由して節点（＋）１３０１への経路長は１である。しかし、さらに、節点（＋）１３０１　→　節点（＋）１３０２　→節点（＋）１３０３　→節点（＋）１３０１という閉経路も存在するため、節点（ｓ）１３００→節点（Ｒｘ）１３０４　→　節点（＋）１３０１　→　節点（＋）１３０２　→節点（＋）１３０３　→　節点（＋）１３０１の経路長は３になる。さらにこの閉経路を回ると経路長はどんどん大きくなり、結局最長経路は求まらない。このように、枝の重みの合計が正の数である閉経路が存在すると最長経路は求まらず、これは制約グラフ中の全ての制約を同時に満たす解が存在しないことに対応する。逆に枝の重みの合計が０か負の数の閉経路の場合には、閉経路を通る経路が最長経路となることはないためこのような問題は生じない。図１６のように、値を作る前に参照してしまうような、不可能な繰り返し間依存制約があると、その繰り返し間依存制約を含む閉グラフの重みの合計は正の値となる。

【００７４】

重み合計が正の閉経路を求める方法としてはどのような方法を用いてもかまわないが、単純にＡＳＡＰスケジューリングを行い、ある一定時間内で結果が求まらなかった場合は重み合計が正の閉経路が存在するとするのが簡便で高速である。

【００７５】

次に図２に示す違反箇所検出・出力方法（ステップＳ９０４）について説明する。

【００７６】

重み合計が正の閉経路が存在することはわかっているが、制約グラフのどこに閉経路が存在するかがわからない場合は、図７に示す方法で、閉経路の位置を特定することが可能である。

【００７７】

手順１４０１では、図２に示す繰り返し間制約生成（ステップＳ９０２）で作った繰り返し間制約のうち任意の１つを選び、制約グラフから削除する。手順１４０２では、図２に示す制約違反検出（手順９０３）と同じ方法で制約違反を検出し、もし制約違反がまだあるなら手順１４０１にもどる。もし、制約違反がなくなれば、最後に削除した繰り返し間制約を制約違反箇所として検出する（手順１４０３）。本来は検出した繰り返し間制約を含む重み合計が正である閉ループが全体として制約違反なのであるが、繰り返し間制約を問題箇所として出力するほうが、設計者にとって着目点が特定できてわかり易い。制約グラフ中の繰り返し間制約に対応するデータフローグラフのポートを求め、そのデータフローグラフのポートが元の動作記述の変数に対応する場合はその変数名を出力してもよい。また、そのデータを生成するデータフローグラフの節点と使用する節点のそれぞれに対応する動作記述中の演算を出力してもよい。

【００７８】

例えば、図６に示す制約グラフの場合、枝１３０５が削除されると重み合計が正である閉経路がなくなるため、枝１３０５に対応する制約が制約違反箇所となり、元の動作記述における図１３に示す変数ｓｕｍを制約違反箇所として出力する。

【００７９】

次に図２に示す違反量検出・出力方法（ステップＳ９０５）について説明する。

【００８０】
図８は、違反量検出・出力方法を示す。手順１５０１では、手順９０４で求めた違反箇所について制約の重みを１減らす。手順１５０２では、図２に示す制約違反検出（ステップＳ９０３）と同じ方法で制約違反を検出し、もし制約違反がまだあるなら手順１５０１にもどる。もし、制約違反がなくなれば、減らした重みの合計を制約違反量として検出し、出力する。図６に示す制約グラフＧｒ１３で、ｔｐ＝１の場合、枝１３０５の重み０が、－２になれば重み合計が正の閉経路はなくなるので、制約違反量は２と求まる。
【００８１】
これにより、例えば、図６に示す制約グラフＧｒ１３でｔｐ＝１の場合、枝１３０５に対応する繰り返し間依存が２制約違反していることが分かる。枝１３０５の繰り返し間制約は、図１４に示すデータフローグラフＧ５のポート５１１および５１２に対して生成したものであるので、すなわち、図１３に示す動作記述で、変数ｓｕｍの生成が次の繰り返しでの使用に２サイクル間に合わないことがわかる。これにより、設計者はスループットを１から３に変更したり、あるいは、図９に示すように、変数ｓｕｍの使用を遅らせる処置を取ることにより、スループット１でも正しく動作する回路を得ることが可能となる。
【００８２】
このように本発明の実施形態では、動作記述からデータフローグラフを生成し、該データフローグラフに基づいて動作合成を行なってハードウェア回路構成を作成する動作合成装置において、ループ処理をパイプライン化することによりスケジューリング制約違反が生じるかどうかを検出する手段を備えたので、ループ処理をパイプライン化したときに該ループ処理を正しく実行できるかどうかを自動的に判断することができる。
【００８３】
また、この実施形態では、ループ処理における第１節点Ａで生成した値を、ループ処理のｎ回後の繰り返し時に、該ループ処理における第２節点Ｂで使用する場合、ループ処理のスループット値をｔｐとして、第１節点から第２節点への重みに対してｔｐ×ｎの値を減算して繰り返し間制約とし、また、繰り返し間制約を含む制約グラフ中に重み合計が正の閉ループがあるかを判断し、制約違反があるかを判断する機能を提供するので、繰り返し間制約枝のうち１つを制約グラフから削除し、もし、制約違反がなくなればその枝がスケジューリング制約違反の原因と判断し、もし、まだ制約違反があれば、別の繰り返し間制約枝に対して同じ処理を繰り返すことにより、スケジューリング制約違反の原因箇所を検出することができる。
【００８４】
また、制約違反の原因となっている繰り返し間制約枝に対して、重みを１減らし、もし、制約違反がなくなれば減らした重みの合計値をスケジューリング制約違反の量として検出し、もし、制約違反がまだあるならば、さらに重みを１減らして同じ処理を繰り返すことにより、スケジューリング制約違反量を検出することができる。
【００８５】
以上のように本実施形態では、ループをパイプライン化したときに、回路が正しく動作するかどうかを動作合成システムが判断することができ、正しく動作しないと判断したときには、動作記述のどこを修正すればよいかを設計者に知らせることが可能となる。これにより、設計効率を高めることができる。
【００８６】
なお、上記実施形態の動作合成装置１０００は、その動作合成処理をコンピュータに実行させるための処理手順が記述された制御プログラムをコンピュータ読み取り可能な可読記憶媒体に格納して、コンピュータにより実現できる。
【００８７】
また、上記実施形態１の動作合成装置により作成されたハードウエハ回路構成を示す回路情報は、回路情報に基づいて半導体集積回路を製造する方法において、該回路情報として用いることができるものである。
【００８８】

以上のように、本発明の好ましい実施形態を用いて本発明を例示してきたが、本発明は、この実施形態に限定して解釈されるべきものではない。本発明は、特許請求の範囲によってのみその範囲が解釈されるべきであることが理解される。当業者は、本発明の具体的な好ましい実施形態の記載から、本発明の記載および技術常識に基づいて等価な範囲を実施することができることが理解される。本明細書において引用した特許文献は、その内容自体が具体的に本明細書に記載されているのと同様にその内容が本明細書に対する参考として援用されるべきであることが理解される。

【産業上の利用可能性】

【００８９】

本発明は、動作合成装置、動作合成方法、プログラム、記録媒体、および半導体集積回路の製造方法の分野において、ループ処理をパイプライン化したときに該ループ処理を正しく実行できるかどうかを自動的に判断することができ、しかも、正しく実行できない場合は、記述のどこを修正すればよいかを設計者に知らせることができる。

【図面の簡単な説明】

【００９０】

【図１】本発明の実施形態１による動作合成装置を説明するブロック図である。

【図２】本実施形態１の動作合成装置を説明する図であり、動作合成方法を説明するフローチャートを示している。

【図３】本実施形態１の動作合成装置を説明する図であり、動作合成方法における制約グラフの例（図（ｂ））を、これと等価な線形計画問題（図（ａ））とともに示している。

【図４】本実施形態１の動作合成装置を説明する図であり、動作合成方法における制約グラフ（図（ａ））とスケジューリング結果（図（ｂ））とを示す図である。

【図５】本実施形態１の動作合成装置を説明する図であり、動作合成方法における繰り返し間制約（図（ａ）～図（ｃ））を説明する図である。

【図６】本実施形態１の動作合成装置を説明する図であり、動作合成方法における繰り返し間制約を加えた制約グラフを示している。

【図７】本実施形態１の動作合成装置を説明する図であり、制約違反箇所検出・出力方法のフローチャートを示している。

【図８】本実施形態１の動作合成装置を説明する図であり、制約違反量検出・出力方法のフローチャートを示している。

【図９】本実施形態１の動作合成装置を説明する図であり、修正後の動作記述例を示している。

【図１０】従来の動作合成方法のフローチャートを示す図である。

【図１１】従来の動作合成方法を説明する図であり、データフローグラフを示している。

【図１２】従来の動作合成方法を説明する図であり、データフローグラフのデータ構造を示している。

【図１３】従来の動作合成方法を説明する図であり、動作記述の例を示している。

【図１４】従来の動作合成方法を説明する図であり、ループのデータフローグラフを示している。

【図１５】従来の動作合成方法を説明する図であり、スループット３のパイプライン動作を示している。

【図１６】従来の動作合成方法を説明する図であり、スループット１のパイプライン動作を示している。

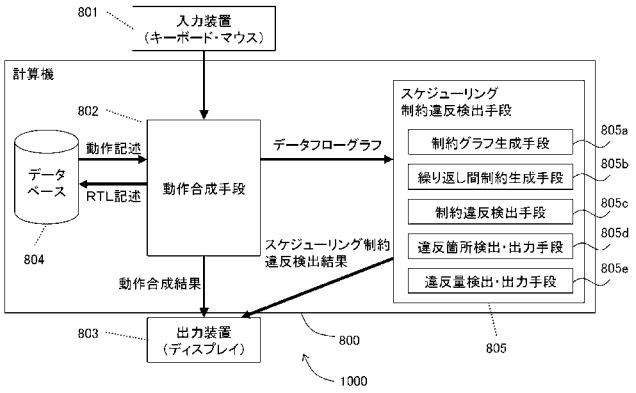
【符号の説明】

【００９１】

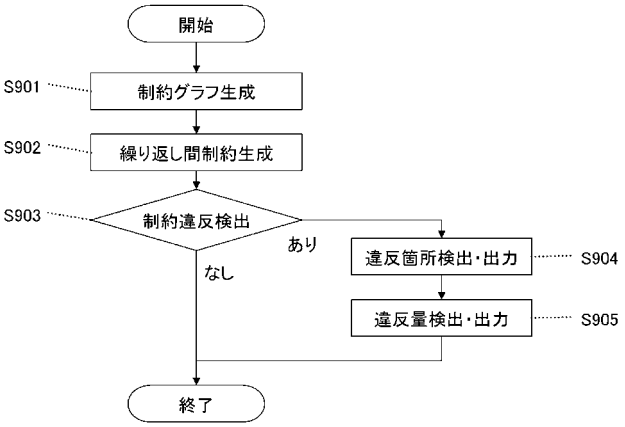
８０１　入力装置
８０２　動作合成手段
８０３　出力装置
８０４　データベース
８０５　スケジューリング制約違反検出手段

８０５ａ　制約グラフ生成手段
８０５ｂ　繰り返し間制約生成手段
８０５ｃ　制約違反検出手段
８０５ｄ　違反箇所検出・出力手段
８０５ｅ　違反量検出・出力手段
１０００　動作合成装置

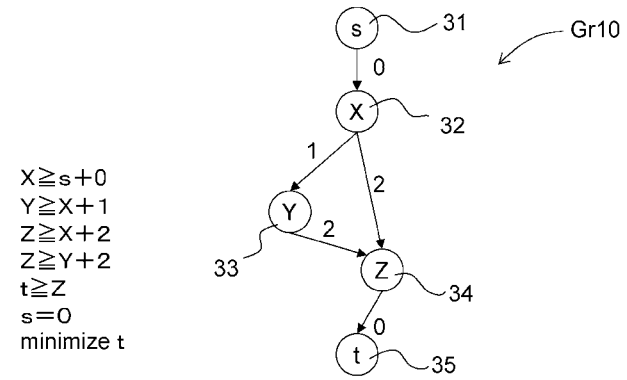
【図１】



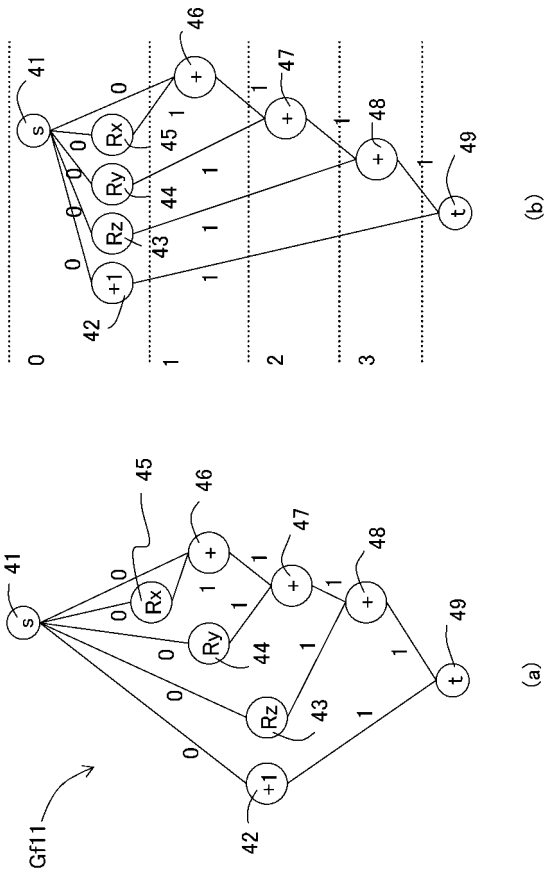

【図２】



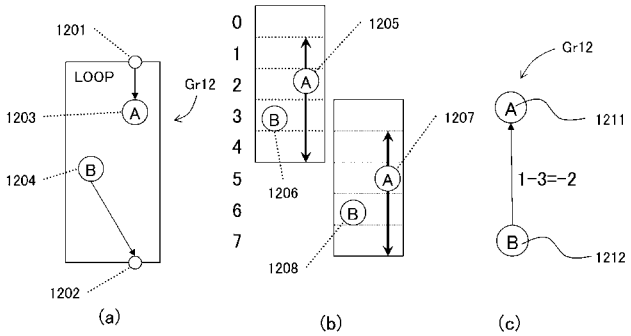
【図３】


X≧s＋0
Y≧X＋1
Z≧X＋2
Z≧Y＋2
t≧Z
s＝0
minimize t
s
X
Y
Z
t
31
32
33
34
35
Gr10
0
1
2
線形計画問題
制約グラフ
(a)
(b)

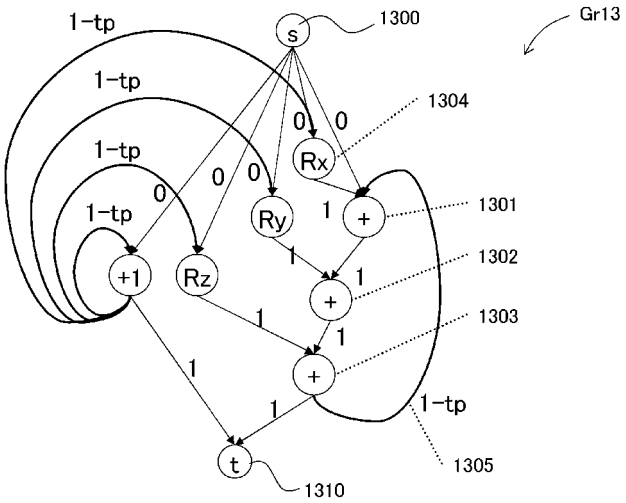
【図４】


Gf11
(a)
(b)

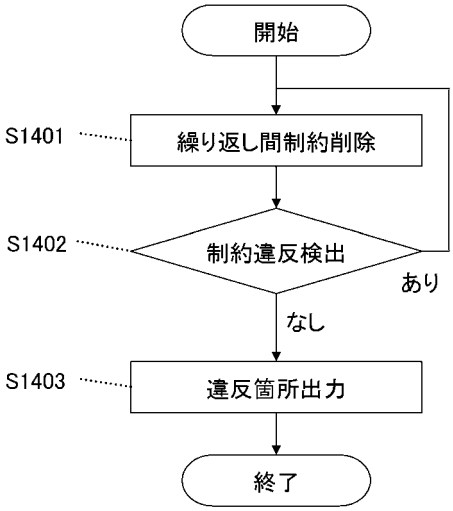
【図５】


LOOP
Gr12
1201
1202
1203
1204
1205
1206
1207
1208
1211
1212
1-3=-2
(a)
(b)
(c)

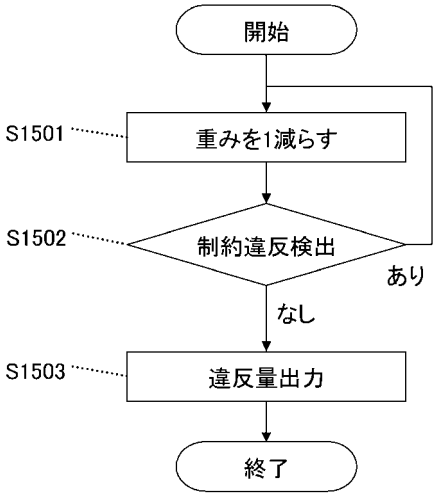
【図６】


Gr13
1-tp
1300
1301
1302
1303
1304
1305
1310

【図７】



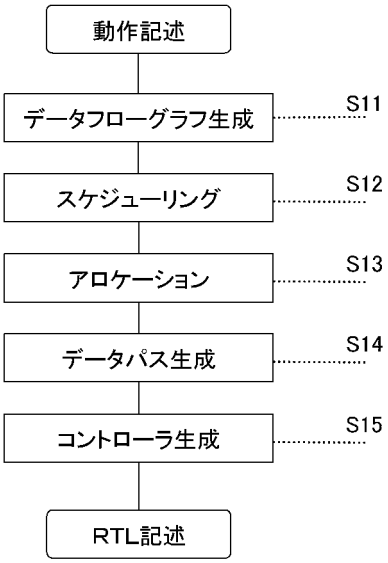

【図８】



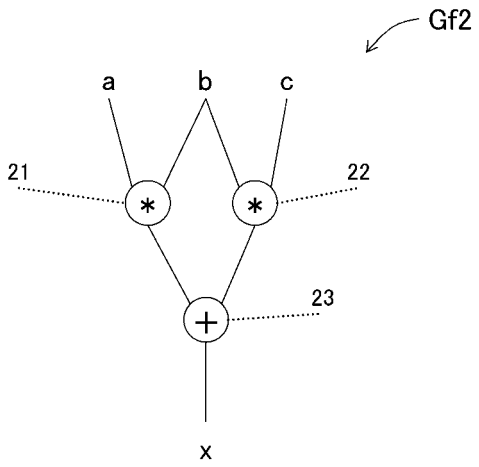

【図９】

```
sum = 0;
for (i = 0; i < 10; i++) {
    tmp =  x(i);
    tmp = tmp + y(i);
    tmp = tmp + z(i);
    sum = sum + tmp;
}
```

【図１０】



【図１１】




【図１２】

Ns

```
struct Node {
    int node_id;
    int in_edge[2];
    int out_edge[1];
    int op_type;
    int schedule;
}
```

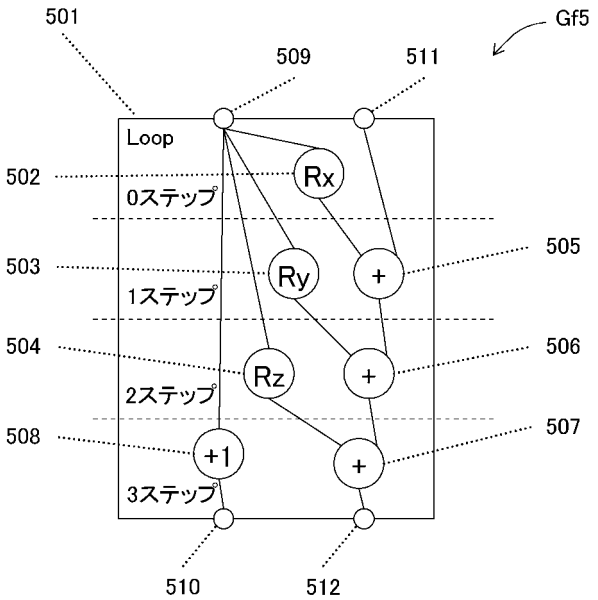
Es

```
struct Edge {
    int edge_id;
    int from_node;
    int to_node;
}
```

【図１３】

```
sum = 0;
for (i = 0; i < 10; i++) {
    sum = sum + x(i);
    sum = sum + y(i);
    sum = sum + z(i);
}
```

【図１４】



【図１５】

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| サイクル1 | Rx(0) | | | | | | | | |
| 2 | Ry(0) | | | | | | | | |
| 3 | Rz(0) | | | | | | | | |
| 4 | + | Rx(1) | | | | | | | |
| 5 | | Ry(1) | | | | | | | |
| 6 | | Rz(1) | | | | | | | |
| ・・・ | | | | ・・・ | | | | | |
| 27 | | | | | | Rz(8) | | | |
| 28 | | | | | | + | Rx(9) | | |
| 29 | | | | | | | Ry(9) | | |
| 30 | | | | | | | Rz(9) | | |
| 31 | | | | | | | + | | |

【図１６】

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| サイクル1 | Rx(0) | | | | | | | | |
| 2 | Ry(0) | Rx(1) | | | | | | | |
| 3 | Rz(0) | Ry(1) | Rx(2) | | | | | | |
| 4 | + | Rz(1) | Ry(2) | | | | | | |
| 5 | | + | Rz(2) | | | | | | |
| 6 | | | + | | | | | | |
| ・・・ | | | ・・・ | | | | | | |
| 9 | | | | | | Rx(8) | | | |
| 10 | | | | | | Ry(8) | Rx(9) | | |
| 11 | | | | | | Rz(8) | Ry(9) | | |
| 12 | | | | | | + | Rz(9) | | |
| 13 | | | | | | | + | | |