(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4029959号
(P4029959)
(45)発行日　平成20年1月9日(2008.1.9)　(24)登録日　平成19年10月26日(2007.10.26)

(51)Int.Cl.　F I
G06F 17/50 (2006.01)　G06F 17/50 654D

請求項の数 7　(全 30 頁)

(21)出願番号　特願2001-264287(P2001-264287)
(22)出願日　平成13年8月31日(2001.8.31)
(65)公開番号　特開2003-76728(P2003-76728A)
(43)公開日　平成15年3月14日(2003.3.14)
審査請求日　平成16年6月18日(2004.6.18)

(73)特許権者　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
(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

審査官　松浦　功

(56)参考文献　特開平０８－０４４７７３（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 17/50

(54)【発明の名称】演算器アロケーション設計装置および演算器アロケーション設計方法

(57)【特許請求の範囲】

【請求項１】

ハードウェアの構成に関する情報が含まれず、処理のアルゴリズムのみが記述された動作記述から回路を合成する高位合成を行う際に、データの流れが枝、演算が節点で表されたデータフローグラフ中の演算を、スケジューリング結果に基づいて、演算器に割り付ける演算器アロケーション設計装置において、

ある演算Ａを演算器に割り付ける際に、既に他の演算Ｂが割り付けられている演算器Ｃに割り付けるために設けられるセレクタによって増加する面積と、その演算Ａのみを割り付けるために新たに追加される演算器Ｄの面積とを比較する比較手段と、

前記比較手段における比較の結果、前者の面積が小さい場合には、その演算Ａを他の演算Ｂが割り付けられている演算器Ｃに割り付けると共にセレクタを設け、後者の面積が小さい場合には、新たに演算器Ｄを追加して、その演算Ａを新たに追加された演算器Ｄに割り付ける割付手段と、

を有することを特徴とする演算器アロケーション設計装置。

【請求項２】

ハードウェアの構成に関する情報が含まれず、処理のアルゴリズムのみが記述された動作記述から回路を合成する高位合成を行う際に、データの流れが枝、演算が節点で表されたデータフローグラフ中の演算を、スケジューリング結果に基づいて、演算器に割り付ける演算器アロケーション設計装置において、

複数の演算をある１つの演算器に割り付ける際に、組み合わせ回路のみを経由するルー

プが発生するか否かを判断する判断手段と、
前記判断手段における判断の結果、前記ループが発生しない場合には、それら複数の演算をその演算器に割り付け、前記ループが発生する場合には、それら複数の演算を異なる演算器にそれぞれ割り付ける割付手段と、
を有することを特徴とする演算器アロケーション設計装置。
【請求項３】
ハードウェアの構成に関する情報が含まれず、処理のアルゴリズムのみが記述された動作記述から回路を合成する高位合成を行う際に、データの流れが枝、演算が節点で表されたデータフローグラフ中の演算を、スケジューリング結果に基づいて、演算器に割り付ける演算器アロケーション設計装置において、
複数の演算をある１つの演算器に割り付ける際に、指定した期間よりも長いフォールスパスが発生するか否かを判断する判断手段と、
前記判断手段における判断の結果、前記フォールスパスが発生しない場合には、それら複数の演算をその演算器に割り付け、前記フォールスパスが発生する場合には、それら複数の演算を異なる演算器にそれぞれ割り付ける割付手段と、
を有することを特徴とする演算器アロケーション設計装置。
【請求項４】
請求項１に記載の演算器アロケーション設計装置によって実施される演算器アロケーション設計方法であって、
前記比較手段によって、ある演算Ａを演算器に割り付ける際に、既に他の演算Ｂが割り付けられている演算器Ｃに割り付けるために設けられるセレクタによって増加する面積と、その演算Ａのみを割り付けるために新たに追加される演算器Ｄの面積とを比較する比較工程と、
前記割付手段によって、前記比較工程における比較の結果、前者の面積が小さい場合には、その演算Ａを他の演算Ｂが割り付けられている演算器Ｃに割り付けると共にセレクタを設け、後者の面積が小さい場合には、新たに演算器Ｄを追加して、その演算Ａを新たに追加された演算器Ｄに割り付ける割付工程と、
を包含することを特徴とする演算器アロケーション設計方法。
【請求項５】
請求項２に記載の演算器アロケーション設計装置によって実施される演算器アロケーション設計方法であって、
前記判断手段によって、複数の演算をある１つの演算器に割り付ける際に、組み合わせ回路のみを経由するループが発生するか否かを判断する判断工程と、
前記割付手段によって、前記判断工程における判断の結果、前記ループが発生しない場合には、それら複数の演算をその演算器に割り付け、前記ループが発生する場合には、それら複数の演算を異なる演算器にそれぞれ割り付ける割付工程と、
を包含することを特徴とする演算器アロケーション設計方法。
【請求項６】
請求項３に記載の演算器アロケーション設計装置によって実施される演算器アロケーション設計方法であって、
前記判断手段によって、複数の演算をある１つの演算器に割り付ける際に、指定した期間よりも長いフォールスパスが発生するか否かを判断する判断工程と、
前記割付手段によって、前記判断工程における判断の結果、前記フォールスパスが発生しない場合には、それら複数の演算をその演算器に割り付け、前記フォールスパスが発生する場合には、それら複数の演算を異なる演算器にそれぞれ割り付ける割付工程と、
を包含することを特徴とする演算器アロケーション設計方法。
【請求項７】
前記割付工程において、面積が大きい演算器を用いる演算から順に演算を割り付けることを特徴とする請求項４乃至請求項６のいずれかに記載の演算器アロケーション設計方法。

【発明の詳細な説明】
【０００１】
【発明の属する技術分野】
本発明は、ＬＳＩの動作記述からディジタル回路を自動合成する高位合成を行う際に、データフローグラフ中の演算をスケジューリング結果に基づいて演算器に割り付ける演算器アロケーション設計方法に関する。
【０００２】
【従来の技術】
従来から、大規模なＬＳＩを短期間で設計するために有効な技術として、高位合成技術が知られている。高位合成技術とは、ハードウェアの構造に関する情報が含まれず、処理のアルゴリズムのみが記述された動作記述から、回路を自動的に合成する技術である。従来の高位合成技術を詳細に記した文献としては、例えば”Ｈｉｇｈ Ｌｅｂｅｌ Ｓｙｎｔｅｓｉｓ” Ｋｌｕｗｅｒ ＡｃａｄｅｍｉｃＰｕｂｌｉｓｈｅｒｓ等が挙げられる。
【０００３】
以下に、従来の高位合成技術を用いて、動作記述から回路を自動的に合成する過程について、簡単に説明する。高位合成は、図１に示す手順によって行なわれる。
【０００４】
まず、ステップ１では、動作記述におけるデータの流れを解析して、データフローグラフ（ＤＦＧ）と称されるモデルを作成する。データフローグラフは、プログラムのフローチャートに類似したグラフであり、節点と枝とによって構成されている。枝はデータを表し、節点は演算を表しており、演算の入力は節点の入力枝に対応し、出力は出力枝に対応している。
【０００５】
例えば、図２に示す動作記述は、図３に示すデータフローグラフによって表される。図３に示すデータフローグラフは、２つの乗算を表す節点３１、３２と、１つの加算を表す節点３３とを含み、入力ａおよびｂを乗算した結果と、入力ｂおよびｃを乗算した結果とを加算し、その演算結果ｘを出力することを表している。
【０００６】
次に、ステップ２では、データフローグラフの節点に対応する演算をいつ実行するか、すなわち、どのクロックサイクルで実行するかを決定するスケジューリングを行う。このときには、各演算の遅延時間を考慮して、全ての節点がクロック周期内に収まるようにする必要がある。
【０００７】
図３に示すデータフローグラフのスケジューリング例を図４および図６に示す。図４は、２つの乗算と１つの加算とを１つのクロックサイクル（サイクル１）で実行するようにスケジューリングされている。例えば、加算器の遅延時間が５ｎｓ、乗算器の遅延時間が６０ｎｓの場合、クロック周期が６５ｎｓ以上であれば、図４に示すようにスケジューリングすることができる。
【０００８】
図６では、サイクル１で１つの乗算６１を実行し、次のサイクル２で残りの乗算６２と加算６５とを実行するようにスケジューリングされている。図６に示すスケジューリングも、クロック周期が６５ｎｓ以上の場合に可能である。図６において、クロックサイクルの境界をまたぐ枝６３によって表されるデータは、レジスタＲ１に記憶され、クロックサイクルの境界をまたぐ枝６４によって表されるデータは、変数ｂを保存するレジスタ（図示せず）に記憶される。
【０００９】
図４に示すスケジューリング結果は、図５に示すような回路によって実現することができる。図５に示す回路は、２つの乗算器と１つの加算器とによって構成されており、一方の乗算器にａ、ｂが入力されて乗算され、他方の乗算器にｂ、ｃが入力されて乗算され、両乗算器による乗算結果が加算器に入力されて加算され、演算結果ｘが出力される。

【００１０】
これに対して、図６に示すスケジューリング結果は、図７に示すような回路によって実現することができる。図７に示す回路は、１つのセレクタ（ｓｅｌ）７１と、１つの乗算器７５と、１つの加算器７４と、レジスタ（Ｒ１）７２と、コントローラ（ｃｏｎｔｒｏｌｌｅｒ）７３とによって構成されている。
【００１１】
セレクタ７１は、点線で表されたセレクト信号７６が１の場合に右側の入力を出力し、０の場合には左側の入力を出力する。また、レジスタ７２は、点線で表されたイネーブル信号７７が１の場合に、クロックが立ち上がる瞬間の入力の値を記憶し、０の場合には記憶している値を保持する。また、レジスタ７２からは、記憶している値が出力される。コントローラ７３は、セレクタ７１とレジスタ７２をコントロールする信号７６および７７を生成する。
【００１２】
図７に示す回路の動作を説明する。サイクル１ではセレクタ信号７６とイネーブル信号７７とは共に１であるため、ａ×ｂがレジスタ７２に記憶される。サイクル２ではセレクタ信号７６とイネーブル信号７７とは共に０であるため、ｃ×ｂが計算され、レジスタ７２に記憶されたａ×ｂの値との和ｘが出力される。
【００１３】
図５に示す回路は１サイクルで動作が終了するが、乗算器が２つ必要である。これに対して、図７に示す回路は動作のために２サイクル必要であるが、乗算器が１つで良い。このように、高位合成技術では、高速な回路が必要なときには図５に示す回路、小面積な回路が必要なときには図７に示す回路を合成することができる。
【００１４】
次に、ステップ３では、レジスタアロケーションを行なう。図６に示すスケジューリング結果において、枝６３および６４のように、クロックサイクルの境界をまたぐ枝については、その枝が表すデータをレジスタに蓄える必要がある。その理由は、同期回路において、レジスタの値は、１クロックサイクル毎に変更されるからであり、例えば乗算６１の計算結果ａ×ｂを加算６５によって使用するためには、サイクルの境界で一旦レジスタに記憶させる必要がある。レジスタアロケーションでは、このような各クロックサイクルの境界をまたぐ枝に対して、レジスタを割り付ける。以降の説明では、レジスタアロケーションの結果は、図６に示す６６のように、レジスタ名を矩形で囲んで表す。なお、枝６４に対するレジスタは、変数ｂの値を保存するレジスタを利用することができるため、図７には示していない。
【００１５】
次に、ステップ４では、スケジューリング・レジスタアロケーション結果に基づいて、ＤＦＧ中の演算を演算器に割り付ける、演算器アロケーションを行う。図６に示すスケジューリング結果では、２つの乗算６１および６２は、図７に示す１つの乗算器７５を共有して実行することができる。しかしながら、演算器を共有する方法が複数ある場合には、最適な共有方法を決定する手順が必要になり、この手順が演算器アロケーションである。
【００１６】
従来の演算器アロケーション設計方法（”Ｈｉｇｈ Ｌｅｂｅｌ Ｓｙｎｔｅｓｉｓ” Ｋｌｕｗｅｒ Ａｃａｄｅｍｉｃ Ｐｕｂｌｉｓｈｅｒｓ、特開２０００－２４２６６９号公報等）では、図６に示す２つの乗算６１、６２を図７に示す１つの乗算器７５にアロケーションする場合のように、可能な限り演算器の個数が少なくなるように演算を割り付けて、回路面積の削減を図っている。
【００１７】
次に、ステップ５では、データフローグラフの枝に基づいてデータパスを生成し、レジスタ、セレクタ等を制御する制御論理を生成して、レジスタ、演算器等のハードウェア構成と動作周期毎のレジスタ間のデータの流れと処理とが含まれるＲＴＬ（レジスタトランスファレベル）の回路を生成する。

【００１８】
【発明が解決しようとする課題】
従来の演算器アロケーション設計方法においては、演算器の個数ができるだけ少なくなるように、演算器に演算が割り付けられるため、図６に示すスケジューリング・レジスタアロケーション結果に基づいて演算器アロケーションを行った場合、図７に示すような演算器アロケーション結果が得られる。
【００１９】
しかしながら、図８に示すように、図６に示す２つの乗算６１および６２に対して、それぞれ異なる乗算器１０１および１０２を割り付ける演算器アロケーションも考えられる。図８に示す回路は、入力されたａとｂとが乗算される乗算器１０１と、入力されたｂとｃとが乗算される乗算器１０２と、乗算器１０１の乗算結果が記憶されて保持されるレジスタ（Ｒ１）１０３と、レジスタ１０３の出力と乗算器１０２の乗算結果とが加算されて演算結果ｘが出力される加算器と、レジスタ１０３のコントロール信号を生成するコントローラ（ｃｏｎｔｒｏｌｌｅｒ）とによって構成されている。
【００２０】
この図８に示す回路では、乗算器が２つ必要であるが、図７に示す演算器アロケーション結果では必要であったセレクタが不要である。よって、乗算器の面積がセレクタの面積よりも小さい場合には、図８に示す演算器アロケーション結果の方が回路全体の面積が小さくなり、逆に、セレクタの面積が乗算器の面積よりも小さい場合には図７に示す演算器アロケーション結果の方が回路全体の面積が小さくなる。
【００２１】
一般には、セレクタよりも乗算器の方が面積が大きいため、図７に示す演算器アロケーション結果の方が良い場合が多いが、セレクタよりも面積が小さい演算器を用いる演算の場合には、図８に示すように演算器を共有しない演算器アロケーション結果の方が面積が小さくなる。
【００２２】
ここで、図９に示すようなスケジューリング・レジスタアロケーション結果に基づいて演算器アロケーションを行う場合を考える。図９では、サイクル１でａとｂとを加算（１１１）し、次のサイクル２でｃとｄとを加算（１１２）して、加算１１１の結果と加算１１２の結果とを乗算する。図９において、クロックサイクルの境界をまたぐ枝によって表されるデータ（ａとｂとの加算結果）は、レジスタＲ１に記憶される。
【００２３】
この場合、図１０に示すような演算器アロケーションと、図１１に示すような演算器アロケーションとが可能である。図１０に示す回路は、入力されたａおよびｃのいずれか一方が選択されるセレクタ（ｓｅｌ）と、入力されたｂおよびｄのいずれか一方が選択されるセレクタと、両セレクタの選択結果が入力されて加算される加算器と、加算器の加算結果が記憶されて保持されるレジスタ（Ｒ１）と、レジスタからの出力と加算器の加算結果とを乗算した結果ｘを出力する乗算器と、レジスタおよびセレクタのコントロール信号を生成するコントローラ（ｃｏｎｔｒｏｌｌｅｒ）とによって構成されている。
【００２４】
また、図１１に示す回路は、入力されたａとｂとを加算する加算器と、加算結果（ａ＋ｂ）が記憶されて保持されるレジスタと、入力されたｃとｄとを加算する加算器と、加算結果（ｃ＋ｄ）とレジスタからの出力とを乗算して乗算結果ｘを出力する乗算器と、レジスタのコントロール信号を生成するコントローラ（ｃｏｎｔｒｏｌｌｅｒ）とによって構成されている。
【００２５】
図１０に示す回路では、加算１１１および１１２を１つの加算器で実現するために加算器は１つで良いが、セレクタが２つ必要である。これに対して、図１１に示す回路では、加算１１１および１１２を２つの加算器で実現するために加算器は２つ必要であるが、セレクタは不要である。よって、セレクタ２つの面積が加算器の面積よりも小さい場合には、

加算器１つによって実現される図１０に示す演算器アロケーション結果の方が回路全体の面積が小さくなり、逆に、セレクタ２つの面積が加算器の面積よりも小さい場合にはセレクタ２つによって実現される図１１に示すの演算器アロケーション結果の方が回路全体の面積が小さくなる。

【００２６】

一般には、加算器の面積は、同じビット幅のセレクタ面積の２倍前後であるため、比較のようなより小さい演算器を用いる演算の場合には当然のことながら、加算においても演算器を共有しないで別々の演算器を用いることによってセレクタの数を減らす方が、全体の面積としては小さくなる場合がある。

【００２７】

しかしながら、演算器の個数を最小化する、従来の演算器アロケーション設計方法では、図７および図１０に示すように、演算器の個数は最小であるが、多数のセレクタが必要になるために、全体の面積としてはかえって大きくなってしまう場合がある。

【００２８】

次に、図１２に示すスケジューリング結果に基づいて演算器アロケーションを行う場合を考える。図１２では、サイクル１でａとｂとを加算（１４２）し、加算結果とｃとを乗算（１４２）して演算結果ｘを出力する。次のサイクル２ではｃとｄとを乗算（１４３）し、乗算結果とｂとを加算（１４４）して演算結果ｙを出力する。

【００２９】

演算器の個数を最小化する従来の演算器アロケーション設計方法を用いた場合には、図１３に示すような回路が得られる。この図１３に示す回路は、セレクタ１５４および１５５と、加算器１５２と、乗算器１５３と、コントローラ１５１とによって構成され、加算１４１および１４４が１つの加算器１５２に割り付けられ、乗算１４２および１４３が１つの乗算器１５３に割り付けられている。

【００３０】

図１３に示す回路の動作について説明する。サイクル１では、コントローラ１５１から出力されるコントロール信号１５７と１５８は、それぞれ、１と０である。よって、セレクタ１５４はａを選択して加算器１５２に入力し、加算器１５２はａ＋ｂを計算して、結果をセレクタ１５５に出力する。セレクタ１５５は加算結果ａ＋ｂを選択して乗算器１５３に入力し、乗算器１５３は（ａ＋ｂ）×ｃを計算して、演算結果ｘを出力する。サイクル２では、コントローラ１５１から出力されるコントロール信号１５７と１５８は、それぞれ、０と１である。よって、セレクタ１５５はｄを選択して乗算器１５３に入力し、乗算器１５３はｄ×ｃを計算して、結果をセレクタ１５４に出力する。セレクタ１５４は乗算結果ｄ×ｃを選択して加算器１５２に入力し、加算器１５２は（ｄ×ｃ）＋ｂを計算して、演算結果ｙを出力する。このように、図１３に示す回路は、加算器１つと乗算器１つを用いて、図１２に示す演算結果ｘとｙとを得ることができる。

【００３１】

しかしながら、図１３に示す回路では、太線に示すように、組み合わせ回路（論理出力がその時刻の論理入力によってのみ定まる論理回路であり、インバータ、ＮＡＮＤ回路、ＮＯＲ回路等およびこれらを組み合わせた回路等が挙げられる。フリップフロップ、ラッチ等のような順序回路は、組み合わせ回路に含まれない）のみによって構成されるループ１５６が生成されている。理想的には、コントローラから出力されるコントロール信号１５７と１５８は同時に０になることはないため、セレクタ１５４と１５５が同時に左側の入力を選択することはなく、このループ１５６が活性化されることはない。但し、現実の回路では、コントローラから出力されるコントロール信号１５７と１５８との変化タイミングにずれが生じて、短時間ではあるものの、このループ１５６が活性化される場合がある。

【００３２】

このような、組み合わせ回路のみによって構成されるループ１５６は、データが同じ演算器に戻ってくるため、リングオシレータと同様の原理によって発振を引き起こすおそれが

ある。一旦発振が起こると、消費電力が増大するだけではなく、回路の動作が不安定になって、誤動作の原因ともなる。また、このようなループがあると、論理合成、フロアプラン（ブロック単位での配置）、配置配線（ブロック内のゲート単位での配置）等の工程で、遅延を正確に見積もることができない。よって、演算器の個数を最小化演算期するアロケーション設計方法によって生成された回路は、信頼性に欠けたものとなる。

【００３３】

次に、図１４に示すスケジューリング結果に基づいてアロケーションを行う場合を考える。図１４では、サイクル１でａとｂとを乗算し、乗算結果とｃとを加算（１６１）して演算結果ｘを出力する。次のサイクル２ではｃとｄとを加算（１６２）し、加算結果をｄで除算して演算結果ｙを出力する。

【００３４】

ここで、加算器の遅延は５ｎｓ、乗算器および除算器の遅延は６０ｎｓ、セレクタの遅延は１ｎｓ、クロック周期は１００ｎｓとする。図１４のサイクル１では乗算と加算を続けて行なうため、６５ｎｓの時間が必要であるが、クロック周期が１００ｎｓであるため、十分に間に合う。サイクル２でも、同様に間に合う。

【００３５】

このスケジューリング結果に基づいて、演算器の個数を最小化する従来の演算器アロケーション設計方法を用いて演算器アロケーションを行った場合、図１５に示すような回路が得られる。この図１５に示す回路は、乗算器１７１と、セレクタ１７４と、加算器１７２と、除算器１７３と、コントローラ１７５とによって構成され、２つの加算１６１および１６２が１つの加算器に割り付けられている。

【００３６】

図１５に示す回路の動作を説明する。サイクル１ではコントローラから出力されるコントロール信号１７５は０である。よって、セレクタ１７４は乗算器１７１の出力ａ×ｂを選択して加算器１７２へ入力し、加算器１７２はａ×ｂとｃを加算して演算結果ｘを出力する。ここで、ａ、ｂ、ｃからｘへのパス全ての遅延がクロック周期１００ｎｓ以下であれば、正しい演算結果ｘが出力される。ａからｘへのパスは乗算器１７１、セレクタ１７４、加算器１７２を経由するため、６６ｎｓである。また、ｂからｘ、ｃからｘは、６６ｎｓと５ｎｓであり、全てのパスの遅延はクロック周期以下である。サイクル２では、コントローラから出力されるコントロール信号１７５は１である。よって、加算器１７２へはｄが入力され、加算器１７２はｄ＋ｃを計算して除算器１７３へ出力する。除算器１７３は（ｄ＋ｃ）／ｅの値を計算して演算結果ｙを出力する。ここで、ｃ、ｄ、ｅからｙへのパス全ての遅延がクロック周期１００ｎｓ以下であれば、正しい結果演算ｙが出力される。ｄからｙへのパスはセレクタ１７４、加算器１７２、除算器１７３を経由するため、６６ｎｓである。また、ｃからｙ、ｅからｙは、６５ｎｓと６０ｎｓであり、全てのパスの遅延はクロック周期以下である。

【００３７】

この図１５に示す回路では、太線で表すａからｙへのパス１７６が存在する。このパス１７６は、乗算器１７１、セレクタ１７４、加算器１７２、除算器１７３を通るため、合計遅延は１２６ｎｓであり、仮にこのパス１７６上をデータが流れたとすると、クロック周期１００ｎｓよりも遅延が大きくなってしまうため、回路が誤動作する。しかし、コントローラから出力されるコントロール信号１７５が０のときにはａからｘのパスにデータが流れ、１のときにはｄからｙのパスにデータが流れるようにセレクタによってデータが選択され、ａからｙへデータが流れることはない。よって、ａからｙのパスの合計遅延がクロック周期を越えたとしても、回路は正常に動作するため、ａからｙへのパスを考慮する必要はない。このようなパスをフォールスパスと称する。

【００３８】

しかしながら、現状の自動論理合成ツール、フロアプランツール、配置配線ツール等では、パスがフォールスパスであるか、または実際にデータが流れるパスであるかを区別することができない。このため、クロック周期よりも長いフォールスパスに対して擬似タイミ

ングエラーが発生したり、論理合成の最適化時にフォールスパス上の演算器が、高速ではあるが面積が大きい演算器に入れ換えられて、回路面積が無駄に増加したりしていた。将来的に、フォールスパスを認識できるようなツールが実現されれば、このような問題は生じないが、現状では問題が多発している。

【００３９】

このようなフォールスパスによるタイミングエラーを検出する方法としては、例えば特開２０００－２０３５５５号公報に開示されているようなものがある。しかし、この方法は、シミュレーションを行なうことによって、タイミングエラーが実際には起こらないことを確認してフォールスパスであると判定するため、フォールスパスの検出に非常に時間がかかってしまうという問題がある。また、シミュレーションパターンに漏れがある場合には、実際に信号が伝わるパスがフォールスパスであると誤認されるおそれがある。

【００４０】

さらに、組み合わせ回路のみによって構成されるループ、長いフォールスパス等の発生を考慮せずに演算器アロケーションを行なった後に、組み合わせ回路のみによって構成されるループ、長いフォールスパス等を削除するためには、１つの演算器が複数の演算によって共有されないように、演算器を追加する必要があり、回路面積が大きくなってしまう。

【００４１】

本発明は、このような従来技術の課題を解決するためになされたものであり、回路の規模を縮小し、回路に組み合わせ回路のみからなるループが含まれないようにし、回路に長いフォールスパスが含まれないようにすることができる、演算器アロケーション設計方法を提供することを目的とする。

【００４２】

【課題を解決するための手段】

本発明の演算器アロケーション設計装置は、ハードウェアの構成に関する情報が含まれず、処理のアルゴリズムのみが記述された動作記述から回路を合成する高位合成を行う際に、データの流れが枝、演算が節点で表されたデータフローグラフ中の演算を、スケジューリング結果に基づいて、演算器に割り付ける演算器アロケーション設計装置において、ある演算Ａを演算器に割り付ける際に、既に他の演算Ｂが割り付けられている演算器Ｃに割り付けるために設けられるセレクタによって増加する面積と、その演算Ａのみを割り付けるために新たに追加される演算器Ｄの面積とを比較する比較手段と、前記比較手段における比較の結果、前者の面積が小さい場合には、その演算Ａを他の演算Ｂが割り付けられている演算器Ｃに割り付けると共にセレクタを設け、後者の面積が小さい場合には、新たに演算器Ｄを追加して、その演算Ａを新たに追加された演算器Ｄに割り付ける割付手段と、を有することを特徴とする。

また、本発明の演算器アロケーション設計方法は、前記演算器アロケーション設計装置によって実施される演算器アロケーション設計方法であって、前記比較手段によって、ある演算Ａを演算器に割り付ける際に、既に他の演算Ｂが割り付けられている演算器Ｃに割り付けるために設けられるセレクタによって増加する面積と、その演算Ａのみを割り付けるために新たに追加される演算器Ｄの面積とを比較する比較工程と、前記割付手段によって、前記比較工程における比較の結果、前者の面積が小さい場合には、その演算Ａを他の演算Ｂが割り付けられている演算器Ｃに割り付けると共にセレクタを設け、後者の面積が小さい場合には、新たに演算器Ｄを追加して、その演算Ａを新たに追加された演算器Ｄに割り付ける割付工程と、を包含することを特徴とする。

【００４３】

ＬＳＩの設計に際しては、チップサイズを可能な限り小さくすることと、演算の高速化という前提のもとで、共有できる演算器、レジスタを可能な限り共有化することによって演算器、レジスタの個数を最小化するように演算器アロケーション、レジスタアロケーションが行われ、一つのクロック周期内に可能な限り多くの演算器を割り当てるようにスケジューリングが行われる。

【００４４】

本発明にあっては、ある演算を実行させるための面積の増加に着目して、新たに演算器を割り付ける事で増加する面積（演算器の面積）Ａと、演算器の数を減らすために、同じ演算を新たな演算器を割り付けないで既に存在する演算器を用いる（この場合にはセレクタが必要になる場合が多い）ことによって増加する面積（セレクタの面積）Ｂとを比較する。そして、可能な限り小面積となるように、ＡおよびＢのうち、面積が小さくなる方を実際に割り付ける。

【００４５】

上記方法によれば、後述する実施形態１に示すように、演算器の数は少ないがセレクタが多くなって全体の面積が大きくなってしまったり、セレクタを少なくするために多数の演算器を用いて全体の面積が大きくなってしまったりすることがなくなり、演算器とセレクタの合計面積が小さくなるような演算器アロケーション結果を得ることができる。

【００４６】

さらに、本発明の演算器アロケーション設計装置は、ハードウェアの構成に関する情報が含まれず、処理のアルゴリズムのみが記述された動作記述から回路を合成する高位合成を行う際に、データの流れが枝、演算が節点で表されたデータフローグラフ中の演算を、スケジューリング結果に基づいて、演算器に割り付ける演算器アロケーション設計装置において、複数の演算をある１つの演算器に割り付ける際に、組み合わせ回路のみを経由するループが発生するか否かを判断する判断手段と、前記判断手段における判断の結果、前記ループが発生しない場合には、それら複数の演算をその演算器に割り付け、前記ループが発生する場合には、それら複数の演算を異なる演算器にそれぞれ割り付ける割付手段と、

を有することを特徴とする。

また、本発明の演算器アロケーション設計方法は、前記演算器アロケーション設計装置によって実施される演算器アロケーション設計方法であって、前記判断手段によって、複数の演算をある１つの演算器に割り付ける際に、組み合わせ回路のみを経由するループが発生するか否かを判断する判断工程と、前記割付手段によって、前記判断工程における判断の結果、前記ループが発生しない場合には、それら複数の演算をその演算器に割り付け、前記ループが発生する場合には、それら複数の演算を異なる演算器にそれぞれ割り付ける割付工程と、を包含することを特徴とする。

【００４７】

上記方法によれば、後述する実施形態２に示すように、組み合わせ回路のみを経由するループが発生しないように演算器アロケーション結果が得られるため、回路の発振によって消費電力が増大したり、回路が誤動作したりすることを防ぐことができる。また、論理合成、フロアプラン、配置配線等の工程における遅延見積りも正確に行うことができる。ループの有無は、着目演算の前演算もしくは後演算に同じ演算があるか否かによって判断することができ、同じ演算がある場合にはループがあることになる。

【００４８】

さらに、本発明の演算器アロケーション設計装置は、ハードウェアの構成に関する情報が含まれず、処理のアルゴリズムのみが記述された動作記述から回路を合成する高位合成を行う際に、データの流れが枝、演算が節点で表されたデータフローグラフ中の演算を、スケジューリング結果に基づいて、演算器に割り付ける演算器アロケーション設計装置において、複数の演算をある１つの演算器に割り付ける際に、指定した期間よりも長いフォールスパスが発生するか否かを判断する判断手段と、前記判断手段における判断の結果、前記フォールスパスが発生しない場合には、それら複数の演算をその演算器に割り付け、前記フォールスパスが発生する場合には、それら複数の演算を異なる演算器にそれぞれ割り付ける割付手段と、を有することを特徴とする。

また、本発明の演算器アロケーション設計方法は、前記演算器アロケーション設計装置によって実施される演算器アロケーション設計方法であって、前記判断手段によって、複数の演算をある１つの演算器に割り付ける際に、指定した期間よりも長いフォールスパスが発生するか否かを判断する判断工程と、前記割付手段によって、前記判断工程における

判断の結果、前記フォールスパスが発生しない場合には、それら複数の演算をその演算器に割り付け、前記フォールスパスが発生する場合には、それら複数の演算を異なる演算器にそれぞれ割り付ける割付工程と、を包含することを特徴とする。

【００４９】

フォールスパスは、ＬＳＩの動作上、実際には起こり得ないパスであるので、実際の回路ではフォールスパスがあっても問題は無い。しかし、ＬＳＩの設計が完了した後、設計検証ツールを用いた検証および論理合成ツールを用いた回路の論理の最適化が行われ、現状の設計検証ツールおよび論理合成ツールではフォールスパスを認識できないため、問題が生じる。

【００５０】

通常、設計検証ツールでは、クロック周期よりも大きい遅延を生じるような場合には、エラーと認識される。また、論理合成ツールでは、フォールスパスを、実際に動作するパスと誤認して最適化が行なわれる。この場合、レジスタからレジスタまでの全てのパスが１クロック以内の遅延で動作するように回路を設計する必要があることから、フォールスパスが１クロックよりも長い遅延を持つ場合には、フォールスパス上の演算器を高速なものに置き換えてフォールスパスの遅延が１クロック以内に収まるように最適化される。一般に、高速な演算器は面積が大きいため、回路面積が大きくなってしまうが、実際にはフォールスパスは動作しないので、このような最適化は不要であり、無駄に回路面積を大きくしてしまうことになる。

【００５１】

そこで、本発明にあっては、このような現状の設計検証ツールおよび論理合成ツールを用いた場合でもエラーが発生しないように、クロック周期±αを指定値として、これよりも時間的に長いフォールスパスであれば、設計をやり直して演算器を再度割り付ける。クロック周期±αよりも時間的に短いフォールスパスであればそのまま割り付ける。これは、フォールスパスは、実際には生じ得ないパスであるので、ＬＳＩとしては問題が無く、かつ、この場合は設計検証ツールでもエラーが発生しないので、作業上に問題が無いからである。

【００５２】

上記方法によれば、後述する実施形態３に示すように、指定した期間以上のフォールスパスが発生しないような演算器アロケーション結果が得られるため、クロック周期よりも長いフォールスパスに対して疑似タイミングエラーが発生したり、論理合成の最適化時にフォールスパス上の演算器が、高速ではあるが面積が大きい演算器に入れ換えられて、回路面積が無駄に増加したりすることを防ぐことができる。

【００５３】

本発明の演算器アロケーション設計方法は、前記割付工程において、面積が大きい演算器を用いる演算から順に演算を割り付けることを特徴としてもよい。

【００５４】

上記方法によれば、後述する実施形態４に示すように、演算器を追加してセレクタを削減する場合、演算器を追加して組わ合せ回路のみからなるループの発生を防ぐ場合、演算器を追加して指定した期間以上のフォールスパスの発生を防ぐ場合等に、面積の増加を最小限にすることができる。面積が小さい演算器から割り付けを行った場合、結果として、後で大きな演算器を使わざるを得なくなり、トータル面積が大きくなるからである。

【００５５】

【発明の実施の形態】

本発明の実施の形態について、図面に基づいて説明する。

【００５６】

（実施形態１）

図１６は、本実施形態１の演算器アロケーション設計方法について説明するためのフローチャートである。本実施形態では、データフローグラフ（ＤＦＧ）中の各演算を、スケジューリング結果に基づいて、１つずつ演算器に割り付けていく。

【００５７】
ステップ１１では、ＤＦＧ中の演算を１つ選択する。ステップ１２では、（ａ）ステップ１１で選択された演算を新たに作成した演算器に割り付け、ステップ１３では、（ｂ）既に他の演算が割り付けられた演算器を共有するように演算を割り付ける方法を試みる。このとき、他の演算が割り付けられている演算器に演算を割り付けて演算器を共有する場合には、両演算が競合しないようにセレクタが設けられる。また、上記（ｂ）の方法で割り付け可能な演算器が複数ある場合には、それぞれに対する割り付けを試みる。
【００５８】
次に、ステップ１４では、上記（ａ）の方法で演算器に演算を割り付けた場合に増加する新たな演算器の面積と、上記（ｂ）の方法で演算器に演算を割り付けた場合に増加するセレクタの面積とを比較し、増加する面積がより少ない割り付け方法を採用する。このとき、上記（ｂ）の方法で割り付け可能な演算器が複数ある場合には、増加する面積が最も少ない割り付け方法を採用する。ステップ１５では、割り付けられていない演算がまだ残っているか否かを判断し、残っている場合にはステップ１に戻り、残っていない場合には処理を終了する。
【００５９】
以下に、図１７に示すスケジューリング・レジスタアロケーション結果に基づいて、データフローグラフ中の演算を演算器に割り付ける場合を例に挙げて、本実施形態の演算器アロケーション設計方法について詳細に説明する。ここでは、加算器の面積は９、セレクタの面積は５であるとする。
【００６０】
図１７では、サイクル１でレジスタＲ１に保持されているデータとレジスタＲ２に保持されているデータとを加算（２１１）し、レジスタＲ１に格納する。次のサイクル２では、レジスタＲ１に格納されたデータとレジスタＲ３に保持されているデータとを加算（２１２）し、レジスタＲ４に格納する。次のサイクル３では、レジスタＲ４に格納されたデータとレジスタＲ５に保持されているデータとを加算（２１３）して演算結果ｘを出力する。
【００６１】
この図１７に示すスケジューリング・レジスタアロケーション結果に基づいて、図１８の２２１に示すように、加算２１１を、新たに演算器を作成して割り付ける上記（ａ）の方法によって割り付ける。この時点での面積は、加算器１個分で９となる。この時点では、まだ他の演算が割り付けられた演算器は存在しないため、他の演算と演算器を共有する上記（ｂ）の方法は行なわない。
【００６２】
次に、図１８の２２２に示すように、加算２１２を、新たに演算器を作成して割り付ける上記（ａ）の方法によって割り付ける。この場合での面積は、加算器２個分で１８となる。
【００６３】
この時点では、演算２１１が既に割り付けられている演算器があるので、図１８の２２３に示すように、加算２１２を、他の演算と演算器を共有する上記（ｂ）の方法によって割り付ける。この場合の面積は、加算器とセレクタとがそれぞれ１個ずつであるので１４となる。
【００６４】
ここで、上記（ａ）の方法による割り付け結果（図１８の２２２）の面積１８と、上記（ｂ）の方法による割り付け結果（図１８の２２３）の面積１４とを比べて、面積が小さい上記（ａ）の方法（図１８の２２３）を採用する。既に他の演算が割り付けられた演算器が複数ある場合には、それぞれの演算器に対して演算の割りつけを試み、最も面積が少ない場合を採用する。
【００６５】
次に、図１８の２２４に示すように、加算２１３を、新たに演算器を作成して割り付ける

上記（ａ）の方法によって割り付ける。この場合の面積は、加算器２個とセレクタ１個とで２３となる。

【００６６】

また、図１８の２２５に示すように、加算２１３を、他の演算と演算器を共有する上記（ｂ）の方法によって割り付ける。この場合の面積は、加算器１個とセレクタ３個とであるので２４となる。

【００６７】

ここで、上記（ａ）の方法による割り付け結果（図１８の２２４）の面積２３と、上記（ｂ）の方法による割り付け結果（図１８の２２５）の面積２４とを比べて、面積が小さい上記（ａ）の方法（図１８の２２５）を採用する。既に他の演算が割り付けられた演算器が複数ある場合には、それぞれの演算器に対して演算の割りつけを試み、最も面積が少ない場合を採用する。

【００６８】

この演算器アロケーション結果によって、図１９に示すように回路が得られる。図１９に示す回路は、レジスタＲ１～レジスタＲ５と、レジスタＲ２およびレジスタＲ３の出力のいずれか一方が選択されるセレクタと、レジスタＲ１の出力とセレクタの出力とが加算されてレジスタＲ１またはレジスタＲ４に出力される加算器と、レジスタＲ４の出力とレジスタＲ５の出力とが加算されて演算結果ｘが出力される加算器と、レジスタＲ１～Ｒ５およびセレクタのコントロール信号を生成するコントローラ（ｃｏｎｔｒｏｌｌｅｒ）とによって構成されている。

【００６９】

一方、図１７に示すスケジューリング・レジスタアロケーション結果に基づいて、データフローグラフ中の演算を演算器の個数を最小化する従来の演算器アロケーション設計方法によって演算器アロケーションを行った結果は、図２０に示すようになり、生成される回路は図２１に示すようになる。

図２０に示す従来の演算器アロケーション結果では、図１８の２２５と同様に、加算器とセレクタとの面積合計は２４となり、本実施形態１による演算器アロケーション結果の面積２３の方が小さい。

【００７０】

図２１に示す回路は、レジスタＲ１～レジスタＲ５と、レジスタＲ１およびレジスタＲ４の出力のいずれか一方が選択されるセレクタと、レジスタＲ３およびレジスタＲ５の出力のいずれか一方が選択されるセレクタと、レジスタＲ２の出力と右側のセレクタの出力のいずれか一方が選択されるセレクタと、左側のセレクタの出力と中央のセレクタの出力とが加算されてレジスタＲ１、レジスタＲ４に入力され、または演算結果ｘが出力される加算器と、レジスタＲ１～Ｒ５およびセレクタのコントロール信号を生成するコントローラ（ｃｏｎｔｒｏｌｌｅｒ）とによって構成されている。

【００７１】

図２０に示す従来の演算器アロケーション結果では、図１８の２２５と同様に、加算器とセレクタとの面積合計は２４となり、本実施形態１による演算器アロケーション結果の面積２３の方が小さい。

【００７２】

また、図２２に、全ての演算で演算器を共有しない場合の演算器アロケーション結果を示し、図２３は生成される回路を示す。

【００７３】

図２３に示す回路は、レジスタＲ１～レジスタＲ５と、レジスタＲ１の出力とレジスタＲ２の出力とが加算されてレジスタＲ１に出力される加算器と、レジスタＲ１の出力とレジスタＲ３の出力とが加算されてレジスタＲ４に出力される加算器と、レジスタＲ４の出力とレジスタＲ５の出力とが加算されて演算結果ｘが出力される加算器と、レジスタＲ１～Ｒ５コントロール信号を生成するコントローラ（ｃｏｎｔｒｏｌｌｅｒ）とによって構成されている。

【００７４】
図２２に示す従来の演算器アロケーション結果では、面積が加算器３個分であるので２７となり、本実施形態１による演算器アロケーション結果の面積２３の方が小さい。
【００７５】
以上のように、本実施形態１によれば、演算器を新たに作成した場合に増加する演算器の面積と、演算間で演算器を共有する場合に増加するセレクタの面積とを比較して、最適な演算器割り付け方法を選択し、面積が小さい演算器アロケーション結果を得ることができる。従って、演算速度を向上させたスケジューリング結果に基づいて、ＬＳＩの高集積化を図ることができる演算器アロケーションを行うことができる。
【００７６】
（実施形態２）
図２４は、本実施形態２の演算器アロケーション設計方法について説明するためのフローチャートである。本実施形態では、データフローグラフ中のある演算を、スケジューリング結果に基づいて演算器に割り付けたときに、図１３に示すような組み合わせ回路によるループ発生を防ぐ。
【００７７】
まず、前準備として、次のような手順により演算間のデータ依存関係を調べる。ステップ２１では、スケジューリング・レジスタアロケーション結果に基づいて、ＤＦＧから演算器割り付けが行われる演算を全て検出して着目演算として列挙する。ここでは、例として、図１２に示すスケジューリング結果に基づいて、図２５に示すように、各演算器１４１～１４４を着目演算として列挙する。
【００７８】
次に、ステップ２２では、各着目演算に対して、前演算と後演算とを求める。前演算は、着目演算と同一クロックサイクルに実行され、着目演算にデータを供給する演算であり、図２５に示すように、着目演算１４２の前演算は１４１、着目演算１４４の前演算は１４３となる。また、後演算は、着目演算と同一クロックサイクルに実行され、着目演算からデータを受けとる演算であり、図２５に示すように、着目演算１４１の後演算は１４２、着目演算１４３の後演算は１４４となる。
【００７９】
なお、前演算と後演算とは、着目演算との間に別の演算を含んでもよい。例えば、図２６に示すようにサイクル１で乗算３２１を行い、サイクル２で乗算３２２～３２４を行う場合に、演算３２２を着目演算とすると、演算３２３だけではなく演算３２４も後演算となる。しかし、演算３２１のように、サイクルが異なる演算は前演算または後演算には含めない。
【００８０】
次に、ステップ２３では、以下のような手順によりループを検出する。ここでは、例として、図１２に示すスケジューリング結果に基づいて、データフローグラフ中の演算を演算器アロケーションする場合について説明する。
【００８１】
例えば、乗算１４２と乗算１４３とを１つの乗算器に割り付ける場合を考える。組み合わせ回路のみからなるループが発生するのは、図１３に示すように、２つ以上の演算器を、それぞれ、２つ以上の演算が共有している場合のみである。従って、乗算１４２と乗算１４３とを１つの乗算器に割り付けただけでは、ループは発生しない。
【００８２】
ここで、乗算１４２と乗算１４３を１つの乗算器に割り付けることが他の演算の割り付けに及ぼす影響を考慮するために、加算１４２と加算１４３とを次の手順で結合して、図２７に示すように、着目演算、前演算および後演算を変更する。
【００８３】
まず、図２５において、乗算１４２と乗算１４３とを着目演算とする（３１２）および（３１３）を、図２７に示す（３３５）のように１つにまとめる。図２７の（３３５）の

前演算と後演算には、図２５の（３１２）および（３１３）のそれぞれの前演算と後演算を並列に接続する。図２５において、（３１２）は演算１４２の前演算のみを含み、（３１３）は演算１４３の後演算のみを含むため、図２７に示す（３３５）の前演算と後演算は（３３１）と（３３２）のようになる。

【００８４】

ここで、図２９に示すように、１つにまとめる演算３５１と演算３５５の両方に後演算がある場合には、図３０に示す後演算のように、演算３５１の後演算と演算３５５の後演算とを並列に接続する。１つにまとめる演算の両方に前演算がある場合についても同様である。

【００８５】

次に、図２５において、１つにまとめる乗算１４２および乗算１４３を後演算または前演算に含む（３１１）および（３１４）を、図２７に示す（３３６）および（３３７）のように更新する。図２５に示す（３１１）のように、乗算１４２が後演算にある場合は、図２７において（３３５）に新たに追加した後演算（３３２）を、（３３６）の後演算として（３３３）のように追加する。また、図２５に示す（３１４）のように、乗算１４３が前演算にある場合は、図２７において（３３５）で新たに追加した前演算（３３１）を、（３３７）の前演算として（３３４）のように追加する。

【００８６】

次に、図１２に示すスケジューリング結果の乗算１４２と乗算１４３とを１つの乗算器に割り付けた後に、引き続いて、加算１４１と加算１４４とを１つの加算器に割り付ける場合を考える。乗算１４２と乗算１４３とを割り付けた後の前演算と後演算とは、図２７に示すようになっている。

【００８７】

ここで、組み合わせ回路のみからなるループができる条件は、ある演算器Ａに対して演算Ｂと演算Ｃとを割り付けるときに、演算Ｂを着目演算としたときの演算Ｂの前演算または後演算に演算Ｃがあること、または、演算Ｃを着目演算としたときの演算Ｃの前演算または後演算に演算Ｂがあることである。

【００８８】

加算１４１と加算１４４とを１つの加算器に割り付ける場合に、図２７では、加算１４１を着目演算とした場合には後演算に加算１４４を含み、加算１４４を着目演算とした場合には前演算に加算１４１を含む。よって、加算１４１と加算１４４とを１つの演算器に割り付けることによって、組み合わせ回路のみからなるループができてしまうことが分かる。

【００８９】

ステップ２３では、以上の方法によって、複数の演算を１つの演算器に割り付けるときに、組み合せ回路のみからなるループができるか否かを判断する。そして、組み合せ回路のみからなるループができない場合にはステップ２５に進み、それら複数の演算を１つの演算器に割り付けて、ステップ２６において前演算と後演算とを更新する。一方、組み合せ回路のみからなるループができる場合にはステップ２４に進み、それら複数の演算を異なる演算器に別々に割り付ける。

【００９０】

図２７に示す例では、加算１４１と加算１４４とを１つの加算器に割り付けることによって、組み合せ回路のみからなるループができてしまうので、加算１４１と加算１４４とを別々に、２つの加算器に割り付ければよい。この場合に得られる回路は、図２８に示すようになる。図２８に示す回路は、入力されたａとｂとを加算する加算器と、加算器の出力とｄとのいずれか一方を選択して出力するセレクタと、セレクタの出力とｃとを乗算して下側の加算器に出力し、または演算結果ｘを出力する乗算器と、乗算器の出力とｂとを加算して演算結果ｙを出力する乗算器と、セレクタのコントロール信号ｋ１を出力するコントローラとによって構成されている。図２８の回路によれば、図１３に示すような組み合わせ回路によるループ１５６を含まないため、発振により回路の消費電力が大きくなった

り、動作が不安定になったりすることがない。

【００９１】

ステップ２７では、全ての演算が演算器に割り付けられたか否かを判断し、割り付けられていない演算が残っている場合にはステップ２３に戻り、残っていない場合には処理を終了する。

【００９２】

なお、本実施形態２の演算器アロケーション設計方法は、実施形態１の演算器アロケーション設計方法と組み合わせて用いることも可能である。実施形態１の演算器アロケーション設計方法では、演算を既に他の演算が割り付けられた演算器に割り付ける場合と、新たに生成された演算器に割り付ける場合とで、小面積となる方を選択する。このうち、演算を既に他の演算が割り付けられた演算器に割り付ける場合に対して、本実施形態２の演算器アロケーション設計方法を適用することによって、組み合わせ回路のみからなるループができるか否かを判断する。そして、組み合わせ回路のみからなるループができれば、その割り付け方法を行わずに、新たに演算器を生成して演算を割り付けることによって、組み合わせ回路のみからなるループの生成を防ぐことができる。

【００９３】

（実施形態３）

図３１は、本実施形態３の演算器アロケーション設計方法について説明するためのフローチャートである。フォールスパスは、実際には動作し得ないパスであり、これがあっても回路自体には問題は生じない。しかしながら、現状の設計検証ツールでは、フォールスパスであるか、実際に動作するパスであるかを判断することができないため、フォールスパスがエラーとして検出されてしまい、問題となっている。そこで、本実施形態では、演算を演算器に割り付けるときに、指定した値よりも遅延が長いフォールスパスの発生を防ぐ。

【００９４】

まず、複数の演算を１つの演算器に割り付けるときに、指定した値よりも遅延が長いフォールスパスが発生するか否かを調べる。

【００９５】

ステップ３１では、スケジューリング・レジスタアロケーション結果に基づいて、ＤＦＧから演算器割り付けが行われる演算を全て検出して着目演算として列挙し、ステップ３２では着目演算の前演算と後演算とを抽出する。これらのステップ３１およびステップ３２は、図２４に示すステップ２１およびステップ２２と同様にして行うことができる。

【００９６】

次に、ステップ３３では、着目演算、前演算および後演算のそれぞれの遅延時間を求める。前演算および後演算に複数の演算が含まれる場合には、それら全ての演算が完了するために要する遅延時間を求める。

【００９７】

次に、ステップ３４では、演算Ａと演算Ｂとを１つの演算器に割り当てる場合に生成されるフォールスパスの遅延を、

【００９８】

【数１】

$$\begin{aligned}(\text{フォールスパスの遅延}) = \max\Big(&((\text{Aの前演算の遅延}) + (\text{AまたはBの遅延}) \\ &+ (\text{Bの後演算の遅延})), ((\text{Bの前演算の遅延}) \\ &+ (\text{AまたはBの遅延}) + (\text{Aの後演算の遅延}))\Big) \cdots (\text{式}1)\end{aligned}$$

によって求める。ここで、演算Ａと演算Ｂとは同一の演算であるため、同一の遅延時間で

ある。そして、フォールスパスの遅延と、指定された値（クロック周期＋α）とを比較し、フォールスパスの遅延が大きい場合には、指定された値よりも遅延が長いフォールスパスとなる。

【００９９】

例として、図３２に示すスケジューリング結果に基づいて、データフローグラフ中の演算を演算器アロケーションする場合について説明する。ここでは、乗算器の遅延が６０ｎｓ、加算器の遅延が５ｎｓ、セレクタの遅延が１ｎｓ、クロック周期は１００ｎｓであるとする。

【０１００】

図３２では、サイクル１でａとｂとを乗算（４１１）し、乗算結果とｃとを加算（４１２）して演算結果ｘを出力する。次に、サイクル２でｃとｄとを加算（４１３）し、加算結果とｂとを乗算（４１４）して演算結果ｙを出力する。

【０１０１】

図３２に示すデータフローグラフにおいて、着目演算４１１～４１４のそれぞれの前演算および後演算は、図３３に示すようになり、各着目演算４１１～４１４と、それぞれの前演算および後演算の遅延時間は、下記表１に示すようになる。

【０１０２】

【表１】

着目節点	411	412	413	414
前演算の遅延	0	60	0	5
着目演算の遅延	60	5	5	60
後演算の遅延	5	0	60	0

ここで、図３２に示す乗算４１１と乗算４１４とを１つの乗算器に割り当てる場合を考える。上記表１から、乗算４１１の遅延は６０ｎｓであり、その前演算と後演算の遅延は０ｎｓと５ｎｓである。また、乗算４１４の遅延は６０ｎｓであり、その前演算と後演算の遅延は５ｎｓと０ｎｓである。よって、フォールスパスの遅延は、上記（式１）より、ｍａｘ（０＋６０＋０，５＋６０＋５）＝７０ｎｓとなる。

【０１０３】

フォールスパスの遅延を比較する値として、クロック周期である１００ｎｓが指定された場合、７０ｎｓは指定値１００ｎｓよりも小さいため、乗算４１１と乗算４１４とを１つの乗算器に割り付けることによって発生するフォールスパスは、論理合成時の最適化の妨げにならない。フォールスパスは、存在しても実際の回路ではデータパスとして生じ得ないので、問題はないが、このときの遅延が、クロック周期（またはその前後の値）よりも長い場合には、設計検証ツールによってエラーとして検出されてしまい、演算器アロケーションがやり直されるからである。

【０１０４】

図３２に示すスケジューリング結果に基づいて、乗算４１１と乗算４１４とを１つの乗算器に割り当て、加算４１２と加算４１３とを、それぞれ別の加算器に割り当てた回路を図３４に示す。図３４に示す回路は、入力されたｄとｃとが加算される加算器と、加算器の出力とａとのいずれか一方を選択するセレクタと、セレクタの出力とｂとを乗算して下側の加算器に出力し、または演算結果ｙを出力する乗算器と、乗算器の出力とｃとを加算して演算結果ｘを出力する加算器と、セレクタのコントロール信号ｋ１を生成するコントローラによって構成されている。

【０１０５】

この図３４に示す回路では、太線で表すｄからｘへのパス４４１がフォールスパスになる

。このパス４４１は、加算器、セレクタ、乗算器、加算器を通るため、合計遅延は７１ｎｓであり、クロック周期である指定値１００ｎｓよりも小さい。

【０１０６】

本実施形態において、表１のように着目演算、前演算、後演算の遅延を用いてフォールスパスの遅延を見積もる場合には、セレクタの遅延（１ｎｓ～２ｎｓ程度）は考慮することができないため、セレクタの遅延をマージンとして考慮して、フォールスパスの遅延を比較するための値として、例えば９５ｎｓ等、クロック周期よりも若干小さい値を指定してもよい。しかし、この方法によってフォールスパスの生成を防止した場合、多くの演算が、演算器を共有して使用することができなくなり、演算器の個数が増加する。よって、クロック周期よりも若干遅延が長いフォールスパスの発生を許容して、例えば１２０ｎｓ等、クロック周期よりも若干大きい値を指定してもよい。

【０１０７】

次に、図３２に示す加算４１２と加算４１３とを１つの乗算器に割り当てる場合を考える。上記表１から、加算４１２の遅延は５ｎｓであり、その前演算と後演算の遅延は６０ｎｓと０ｎｓである。また、加算４１３の遅延は５ｎｓであり、その前演算と後演算の遅延は０ｎｓと６０ｎｓである。よって、フォールスパスの遅延は、上記（式１）より、ｍａｘ（６０＋５＋６０，０＋５＋０）＝１２５ｎｓとなる。

【０１０８】

フォールスパスの遅延を比較する値として、クロック周期である１００ｎｓが指定された場合、１２５ｎｓは指定値１００ｎｓよりも大きいため、加算４１２と加算４１３とを１つの加算器に割り付けることによって発生するフォールスパスは、論理合成時に設計検証ツールによってタイミングエラーとして検出されてしまうおそれがある。また、論理合成ツールによって、このフォールスパス上の演算器を高速ではあるが、面積が大きい演算器に入れ換えることなどによって、遅延が１００ｎｓ以下になるように最適化される可能性もあるが、このパスはフォールスパスであるため、このような最適化は不要であり、回路面積を増大させるだけである。

【０１０９】

図３２に示すスケジューリング結果に基づいて、加算４１２と加算４１３とを１つの加算器に割り当て、乗算４１１と加算４１４とを、それぞれ別の乗算器に割り当てた回路を図３５に示す。図３５に示す回路は、入力されたａとｂとが乗算される乗算器と、乗算器の出力とｄとのいずれか一方を選択するセレクタと、セレクタの出力とｃとを加算して下側の乗算器に出力し、または演算結果ｘを出力する加算器と、加算器の出力とｂとを乗算して演算結果ｙを出力する加算器と、セレクタのコントロール信号ｋ１を生成するコントローラによって構成されている。

【０１１０】

この図３５に示す回路では、太線で表すａからｙへのパス４５１がフォールスパスになる。このパス４５１は、乗算器、セレクタ、加算器、乗算器を通るため、合計遅延は１２６ｎｓであり、クロック周期である指定値１００ｎｓよりも大きい。

【０１１１】

このように、本実施形態では、図３２に示すスケジューリング結果に基づいて演算器アロケーションを行った場合に、乗算４１１と乗算４１４とを１つの乗算器に割り付けてもクロック周期よりも長いフォールスパスは生成されないが、加算４１２と加算４１３とを１つの加算器に割り付けるとクロック周期よりも長いフォールスパスが生成されることが容易に検出される。

【０１１２】

よって、ステップ３４において、加算４１２と加算４１３とを１つの加算器に割り付ける場合のようにクロック周期よりも長いフォールスパスができる場合には、ステップ３５に進み、それら複数の演算を異なる演算器に別々に割り付ける。一方、乗算４１１と乗算４１４とを１つの乗算器に割り付ける場合のようにクロック周期よりも長いフォールスパスができない場合には、ステップ３６に進み、それら複数の演算を１つの演算器に割り付け

て、ステップ３７において前演算と後演算とを更新し、それに従ってステップ３８において前演算と後演算の遅延を更新する。これによって、クロック周期よりも長いフォールスパスが生成されることを防ぐことが可能となる。

【０１１３】

ステップ３９では、全ての演算が演算器に割り付けられたか否かを判断し、割り付けられていない演算が残っている場合にはステップ３４に戻り、残っていない場合には処理を終了する。

【０１１４】

次に、複数の演算を複数の演算器に割り付ける場合について、図３６に示すスケジューリング結果を例として説明する。図３６に含まれる演算は、全て、入力が１つである単項演算子である。円の中に「ｒ」を記した（４６１）は四捨五入演算であり、遅延は１０ｎｓである。円の中に「＋１」を記した（４６２）、（４６３）および（４６４）はインクリメント演算であり、遅延は１０ｎｓである。円の中に「＾２」を記した（４６５）および（４６６）は２乗演算であり、遅延は６５ｎｓである。円の中に「×２」を記した（４６７）は２倍演算であり、遅延は１０ｎｓである。また、クロック周期は１００ｎｓであるとする。

【０１１５】

図３６では、サイクル１でａが四捨五入（４６１）されて＋１加算（４６２）され、演算結果ｘが出力される。次のサイクル２ではｂが＋１加算（４６３）された後、さらに＋１加算（４６３）され、２乗（４６５）されて演算結果ｙが出力される。次のサイクル３ではｃが２乗（４６６）されて２倍（４６７）され、ｚに出力される。

【０１１６】

図３６に示すデータフローグラフに含まれる演算のうち、複数の演算が１つの演算器を共有する可能性のある演算は（４６２）、（４６３）、（４６４）、（４６５）および（４６６）であり、これらの演算を着目演算とした前演算と後演算とは図３７に示すようになる。また、図３７に示す各着目演算と、それぞれの前演算および後演算の遅延時間は、下記表２に示すようになる。

【０１１７】

【表２】

着目節点	462	463	464	465	466
前演算の遅延	10	0	10	10	0
着目演算の遅延	10	10	10	65	65
後演算の遅延	0	75	65	0	10

図３６に示すデータフローグラフにおいて、演算４６３と演算４６４とは同一のクロックサイクル２で実行されるため、１つの演算器を共有するように割り付けることはできない。一方、サイクルが異なる演算は、演算器を共有できるため、１つの演算器を共有できる可能性のある演算は、（４６２）と（４６３）、（４６２）と（４６４）、（４６５）と（４６６）である。

【０１１８】

これらの演算の組み合せを上記（式１）によって評価すると、それぞれ、９５ｎｓ、８５ｎｓ、８５ｎｓとなり、どの組み合せの演算が演算器を共有したとしても、クロック周期１００ｎｓを越えるフオールスパスはできないことが分かる。

【０１１９】

ここで、例えば２乗演算４６５と２乗演算４６６とを１つの２乗演算器に割り付けた場合を考える。この割り付けにより、他の演算の組み合せ（４６２）と（４６３）、（４６２

）と（４６４）によって生じるフォールスパスの長さが変わる可能性があるので、以下のような方法によって、演算４６５と演算４６６とを同じ演算器に割り付けた影響を、前演算、後演算の遅延を表す表２に反映させる。

【０１２０】

まず、演算器を共有するために、図３７に示す（４７４）の演算４６５と（４７５）の演算（４６６）とを１つにまとめ、図３８に示す（４９４）のようにする。（４９４）の前演算および後演算には、図３７に示す（４７４）と（４７５）のそれぞれの前演算および後演算を並列に接続する。ここで、（４７４）の後演算と（４７５）の前演算は無いため、図３８に示す（４９４）の前演算と後演算は（４８１）と（４８２）のようになる。

【０１２１】

このとき、図２９に示すように、１つにまとめる演算３５１と演算３５５の両方に後演算がある場合には、図３０に示す後演算のように、並列に接続する。この場合の後演算の遅延は、演算３５２と演算３５３の遅延の合計と、演算３５６の遅延の大きい方となる。１つにまとめる演算の両方に前演算がある場合も同様である。

【０１２２】

次に、遅延時間を見るために、演算４６５または演算４６６を前演算もしくは後演算に含む、図３７に示す（４７２）と（４７３）について、図３８に示す（４９４）の前演算（４８１）と後演算（４８２）を、（４９２）の後演算（４８３）、（４９３）の後演算（４８４）のように追加する。演算４６５または演算４６６を前演算もしくは後演算に含まない、図３７に示す（４７１）については、図３８に示す（４９１）のように、前演算、後演算はそのままである。

【０１２３】

その後、前演算、後演算の遅延を表す表２を、図３８に示す前演算、後演算の遅延に従って更新し、下記表３とする。

【０１２４】

【表３】

着目節点	462	463	464	(465,466)
前演算の遅延	10	0	10	20
着目演算の遅延	10	10	10	65
後演算の遅延	0	85	75	10

上記表３に基づいて、演算４６２と演算４６３の組み合わせ、演算４６２と演算４６４の組み合せを、共通の演算器に割り当てた場合に生じるフォールスパスの長さを、それぞれ、上記（式１）によって再度評価すると、１０５ｎｓと９５ｎｓとなり、演算４６２と演算４６３を１つの演算器に割り付けた場合には、クロックサイクル１００ｎｓよりも長いフォールスパスができてしまうことが分かる。これに対して、演算４６２と演算４６４とを１つの演算器に割り付けた場合には、クロックサイクル１００ｎｓより長いフォールスパスはできないことが分かる。

【０１２５】

このように、本実施形態によれば、演算４６５と演算４６６とを１つの演算器に割り付ける前の評価では、演算４６２と演算４６３とを１つの演算器に割り付けたとしてもクロックサイクル１００ｎｓより長いフォールスパスはできないが、演算４６５と演算４６６とを１つの演算器に割り付けた後の評価では、演算４６５と演算４６６とを割り付けた影響によって、クロックサイクル１００ｎｓよりも長いフォールスパスができてしまうことが検出される。

【０１２６】

図３６に示すスケジューリング結果において、演算４６５と演算４６６とを１つの演算器に割り付け、さらに、演算４６２と演算４６４とを１つの演算器に割り付けた場合の回路を図３９に示す。図３９に示す回路は、入力されたａが四捨五入される四捨五入器と、入力されたｂに＋１加算する加算器と、四捨五入演算器の出力と加算器の出力とのいずれか一方を選択するセレクタと、セレクタの出力に＋１加算して下側のセレクタに出力し、または演算結果ｘを出力する加算器５３２と、加算器５３２の出力とｃとのいずれか一方を選択するセレクタと、下側のセレクタの出力を２乗して２倍器に出力し、または演算結果ｙを出力する２乗器５３１と、２乗器５３１の出力を２倍してｚに出力する２倍器と、セレクタのコントロール信号を生成するコントローラによって構成されている。

【０１２７】

図３９に示す回路において、演算器５３１は演算４６５と演算４６６とを実行し、演算器５３２は演算（４６２）と演算（４６４）とを実行する。図３５に示す回路に含まれる最も長いフォールスパスは、太線で表すａからｚへのパス５３３である。このパス５３３は、四捨五入器、加算器、２乗器、２倍器をそれぞれ１回ずつと２つのセレクタを通る。セレクタの遅延を１ｎｓとすると、パス５３３上に配置された演算器による遅延の合計は、１０＋１０＋６５＋１０＋２＝＝９７ｎｓとなり、クロックサイクル１００ｎｓよりも小さい。

【０１２８】

このように、本実施形態の演算器アロケーション方法を用いることによって、論理合成時に設計検出ツールによってフォールスパスによる不要なタイミングエラーが発生したり、論理合成ツールによってフォールスパスによる不必要な最適化を行ったりすることなく、演算器アロケーションを行うことができる。

【０１２９】

なお、本実施形態３の演算器アロケーション設計方法は、実施形態１の演算器アロケーション設計方法と組み合わせて用いることも可能である。実施形態１の演算器アロケーション設計方法では、演算を既に他の演算が割り付けられた演算器に割り付ける場合と、新たに生成された演算器に割り付ける場合とで、小面積となる方を選択する。このうち、演算を既に他の演算が割り付けられた演算器に割り付ける場合に対して、本実施形態３の演算器アロケーション設計方法を適用することによって、指定値よりも長いフォールスパスができるか否かを判断する。そして、指定値よりも長いフォールスパスができれば、その割り付け方法を行わずに、新たに演算器を生成して演算を割り付けることによって、指定値よりも長いフォールスパスの生成を防ぐことができる。

【０１３０】

（実施形態４）

本実施形態では、データフローグラフ中の演算をスケジューリング結果に基づいて演算器に割り付ける際に、面積が大きい演算器を使用する演算から面積が小さい演算器を使用する演算へ、順次割り付けを行っていく方法について説明する。

【０１３１】

ここでは、例として、図１２に示すスケジューリング結果に基づいて演算器アロケーションを行う場合について考える。図１２に示すデータフローグラフにおいて、加算１４１と加算１４４とを１つの加算器に割り付け、乗算１４２と乗算１４３とを１つの乗算器に割り付けた場合、図１５に示すように、組み合せ回路のみからなるループができてしまうため、回路が誤動作する可能性がある。

【０１３２】

上述した実施形態２の方法では、例えば、乗算１４２と乗算１４３とを１つの乗算器に割り付けた後に、加算１４１と加算１４４とを１つの加算器に割り付けようとすると、組み合せ回路のみからなるループが発生するため、加算１４１と加算１４４とを２つの加算器に別々に割り付けることによって、図２８に示すような組み合わせ回路のみからなるループが無い回路を生成している。

【０１３３】

ここで、同じく実施形態２の方法を用いて、加算１４１と加算１４４とを１つの加算器に割り付けた後に、乗算１４２と乗算１４３とを１つの乗算器に割り付けようとした場合、加算１４１と加算１４４とが１つの加算器に割り付けられ、乗算１４２と乗算１４３とが２つの乗算器に別々に割り付けられた、図４０に示すような回路が生成される。

【０１３４】

ここで、図２８に示す回路と図４０に示す回路とは、処理内容は同じであるが、図２８に示す回路は２つの加算器と１つの乗算器とを含むのに対して、図４０に示す回路は１つの加算器と２つの乗算器とを含む。通常、乗算器の面積は加算器の面積の数倍以上であるため、図４０に示す回路の方が図２８に示す回路よりも面積が大きくなる。

【０１３５】

本実施形態では、演算器アロケーションにおいて、面積が大きい演算器を使用する演算から順に割り付けを行なうため、面積が大きい演算器に演算を割り付けた後で、面積が小さい演算器への演算の割り付けを考えることになる。よって、実施形態２の方法によって組み合せ回路のみからなるループの発生を防ぐ場合に、面積の大きい演算器に複数の演算が割り付けられ、面積の小さい複数の演算器に複数の演算が別々に割り付けられることになる。従って、図１２に示すスケジューリング結果に基づいて演算器アロケーションを行った結果は、図２８に示すようになり、図４０に示すようにはならない。このように、本実施形態によれば、組み合せ回路のみからなるループの発生を防ぐために増加する、演算器の面積を最小限に抑えることが可能であり、小面積の回路を得ることができる。

【０１３６】

なお、本実施形態４の演算器アロケーション設計方法は、実施形態３の演算器アロケーション設計方法と組み合わせて用いることも可能である。実施形態３の演算器アロケーション設計方法では、複数の演算を１つの演算器に割り付けた後に、他の複数の演算を１つの演算器に割り付ける場合に、指定値よりも長いフォールスパスができるか否かを判断して、指定値よりも長いフォールスパスができる場合には、演算器を共有せずに、異なる演算器に別々に演算を割り付ける。この場合に、本実施形態４の演算器設計アロケーション設計方法を適用することによって、面積が大きい演算器を使用する演算から順に割り付けを行なう。これにより、指定した値よりも長いフォールスパスが生成するのを防ぐために演算器を共有せずに異なる演算器に演算を別々に割り付ける場合にも、比較的面積の小さい演算器が増加することになる。よって、回路面積の増加を最小限に抑えることが可能であり、小面積の回路を得ることができる。

【０１３７】

【発明の効果】

以上詳述したように、本発明によれば、従来の演算器アロケーション設計方法のように、演算器の数は少ないがセレクタが多くなって全体の面積が大きくなってしまったり、セレクタを少なくするために多数の演算器を用いて全体の面積が大きくなってしまったりすることがなくなり、演算器とセレクタの合計面積が小さくなるような演算器アロケーション結果を得ることができる。

【０１３８】

また、本発明によれば、組み合わせ回路のみを経由するループが発生するのを防ぐことができるため、回路の発振によって消費電力が増大したり、回路が誤動作したりすることを防ぐことができる。また、論理合成、フロアプラン、配置配線等の工程における遅延見積りも正確に行うことができる。

【０１３９】

また、本発明によれば、現状の設計検証ツールおよび論理合成ツールを用いた場合でもタイミングエラーが発生したり、不要な最適化が行われないように、指定値よりも長いフォールスパスが生成されないようにすることができる。

【０１４０】

さらに、本発明によれば、演算器を追加してセレクタを削減する場合、演算器を追加して組わ合せ回路のみからなるループの発生を防ぐ場合、演算器を追加して指定した期間以

上のフォールスパスの発生を防ぐ場合等に、面積の増加を最小限にすることができる。

【図面の簡単な説明】

【図１】高位合成技術を説明するためのフローチャートである。

【図２】動作記述の一例を示す図である。

【図３】図２に示す動作記述のデータフローグラフを示す図である。

【図４】図３に示すデータフローグラフのスケジューリング結果の一例を示す図である。

【図５】図４に示すスケジューリング結果に基づいて生成される回路の一例を示す図である。

【図６】図３に示すデータフローグラフのスケジューリング結果の他の例を示す図である。

【図７】図６に示すスケジューリング結果に基づいて生成される回路の一例を示す図である。

【図８】図６に示すスケジューリング結果に基づいて生成される回路の他の例を示す図である。

【図９】データフローグラフのスケジューリング結果の一例を示す図である。

【図１０】図９に示すスケジューリング結果に基づいて生成される回路の一例を示す図である。

【図１１】図９に示すスケジューリング結果に基づいて生成される回路の他の例を示す図である。

【図１２】データフローグラフのスケジューリング結果の一例を示す図である。

【図１３】組み合わせ回路のみからなるループについて説明するための図である。

【図１４】データフローグラフのスケジューリング結果の一例を示す図である。

【図１５】フォールスパスについて説明するための図である。

【図１６】実施形態１の演算器アロケーション設計方法の手順を説明するための図である。

【図１７】データフローグラフのスケジューリング結果の一例を示す図である。

【図１８】実施形態１の演算器アロケーション設計方法の手順を説明するための図である。

【図１９】図１７に示すスケジューリング結果に基づいて生成される回路の一例を示す図である。

【図２０】従来の演算器アロケーション設計方法によるアロケーション結果の一例を示す図である。

【図２１】図２０に示すアロケーション結果によって生成される回路の一例を示す図である。

【図２２】従来の演算器アロケーション設計方法によるアロケーション結果の他の例を示す図である。

【図２３】図２２に示すアロケーション結果によって生成される回路の一例を示す図である。

【図２４】実施形態２の演算器アロケーション設計方法の手順を説明するための図である。

【図２５】着目演算と前演算と後演算について説明するための図である。

【図２６】後演算の一例を示す図である。

【図２７】図２５に示す前演算と後演算の更新例を示す図である。

【図２８】組み合わせ回路のみからなるループを含まない回路の一例を示す図である。

【図２９】後演算の一例を示す図である。

【図３０】図２９に示す前演算と後演算の更新例を示す図である。

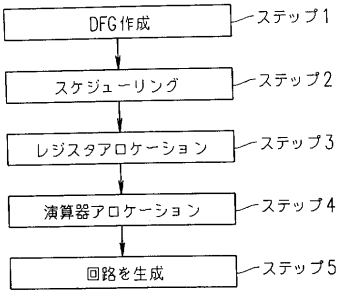
【図３１】実施形態３の演算器アロケーション設計方法の手順を説明するための図である。

【図３２】データフローグラフのスケジューリング結果の一例を示す図である。

【図３３】図３２に示す着目演算と前演算と後演算について説明するための図である。

【図３４】図３２に示すスケジューリング結果に基づいて生成される回路の一例を示す図である。
【図３５】図３２に示すスケジューリング結果に基づいて生成される回路の他の例を示す図である。
【図３６】データフローグラフのスケジューリング結果の一例を示す図である。
【図３７】図３６に示す着目演算と前演算と後演算について説明するための図である。
【図３８】図３７に示す前演算と後演算の更新例を示す図である。
【図３９】図３６に示すスケジューリング結果に基づいて生成される回路の一例を示す図である。
【図４０】図１２に示すスケジューリング結果に基づいて生成される回路の一例を示す図である。
【符号の説明】
３１、３２、６１、６２、１４２、１４３、３２１、３２２、３２３、３５３、３５６、４１１、４１４　乗算
３３、６５、１１１、１１２、１４１、１４４、１６１、１６２、２１１、２１２、２１３、３２４、３５１、３５２、３５４、３５５、４１２、４１３　加算
６３、６４　枝
７１、１５４、１５５、１７４　セレクタ
７２、１０３　レジスタ
７３、１５１　コントローラ
７４、１５２、１７２　加算器
７５、１０１、１０２、１５３、１７１　乗算器
７６、７７、１７５　コントロール信号
１５６　組み合わせ回路のみからなるループ
１７３　除算器
１７６、４４１、４５１、５３３　フォールスパス
２２１～２２５　演算器の割り付け例
３１１～３１４、３３５～３３７、４７１～４７５、４９１～４９４　着目演算と前演算および後演算の例
３３１、３３４、４８１　前演算の例
３３２、３３３、４８２～４８４　後演算の例
４６１　四捨五入演算
４６２～４６４　＋１加算
４６５、４６６　２乗演算
４６７　２倍演算
５３１　２乗器
５３２　＋１加算器

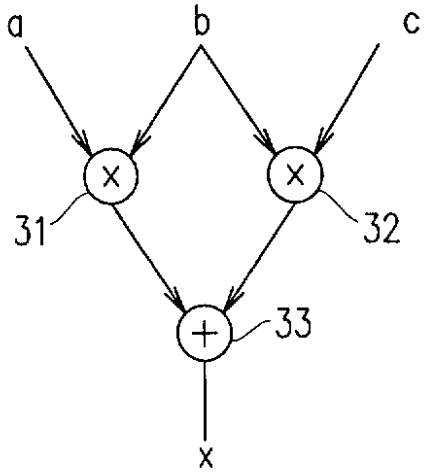
【図１】



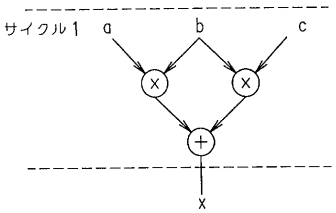
【図２】

$$x = (a \times b) + (b \times c);$$

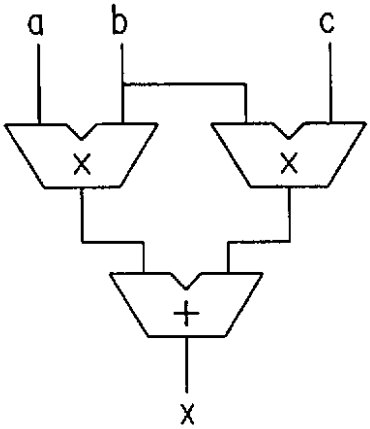
【図３】



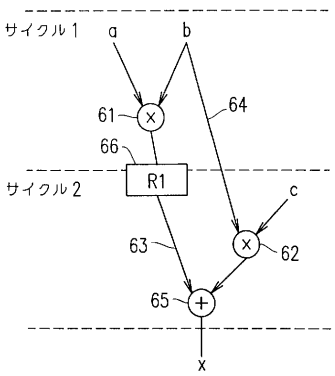
【図４】



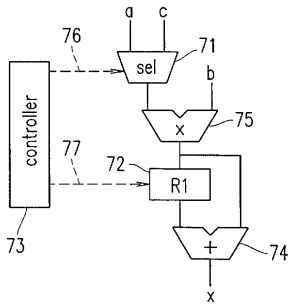
【図５】



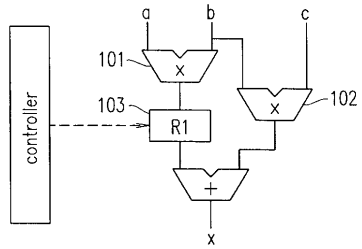
【図６】



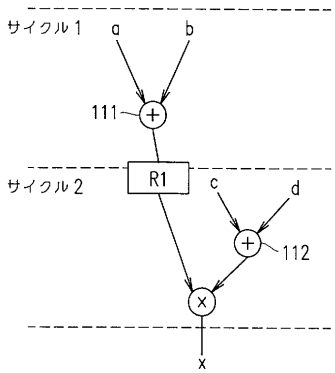
【図７】



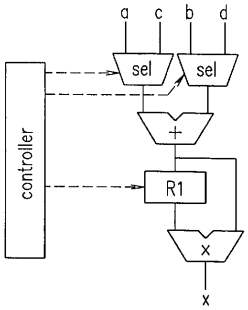
【図８】



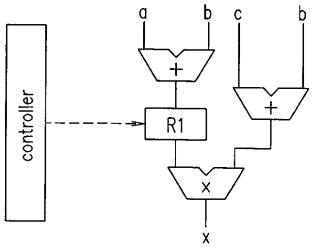
【図９】



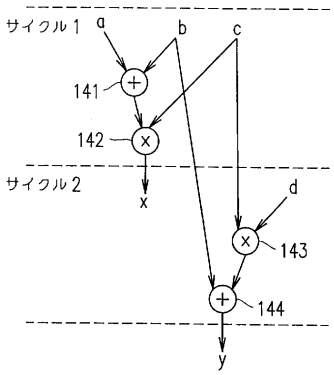
【図１０】



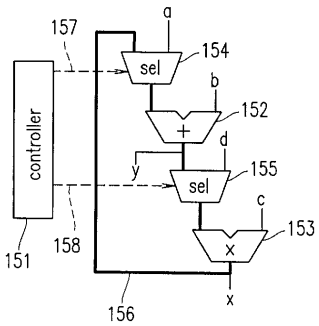
【図１１】



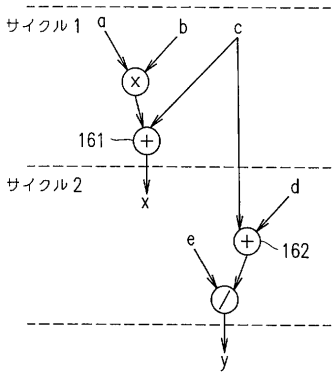
【図１２】



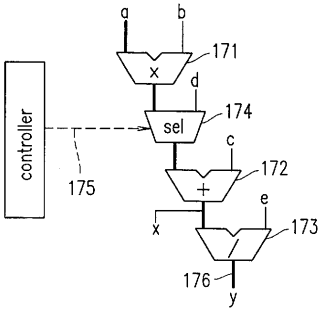
【図１３】



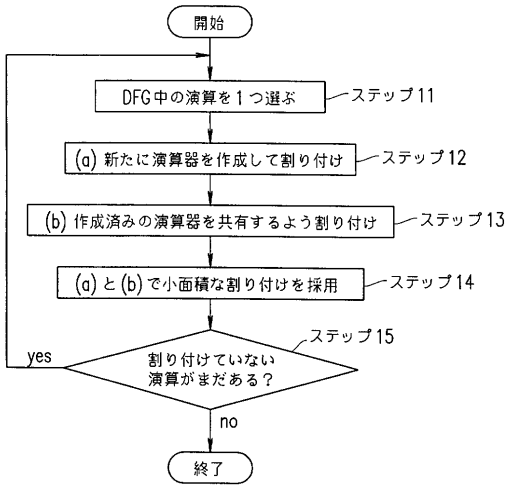
【図１４】



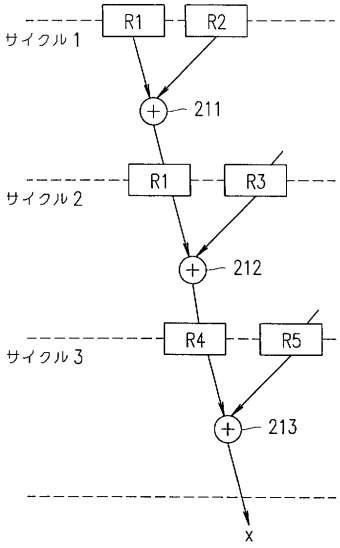
【図１５】



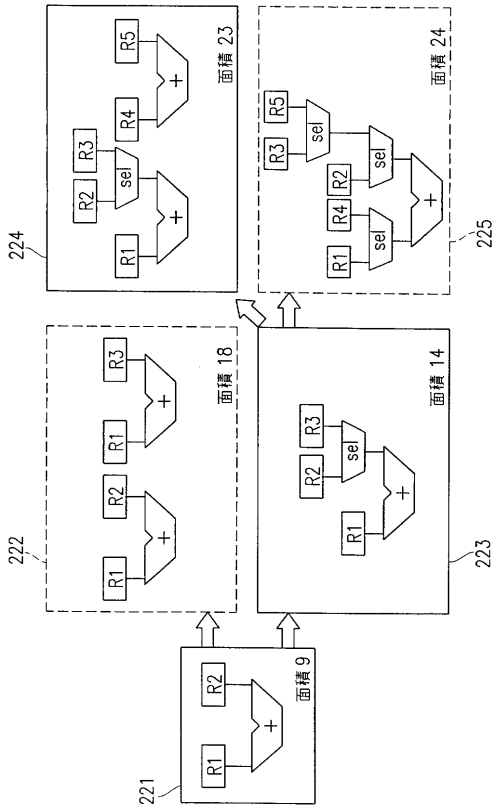
【図１６】



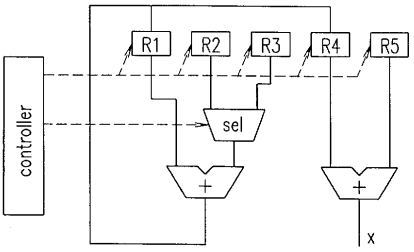
【図１７】



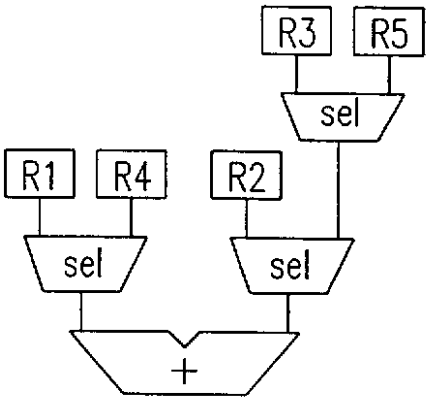
【図１８】



【図１９】

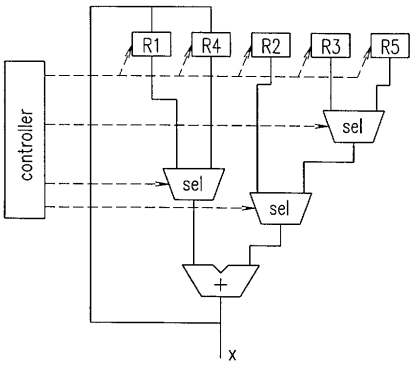


【図２０】

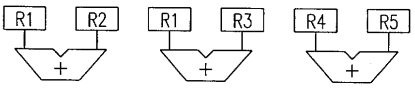


面積 24

【図２１】

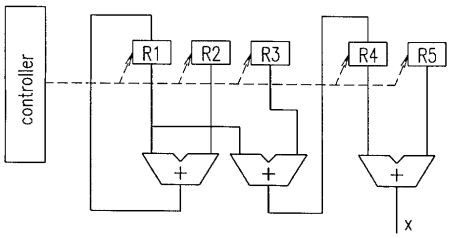


【図２２】

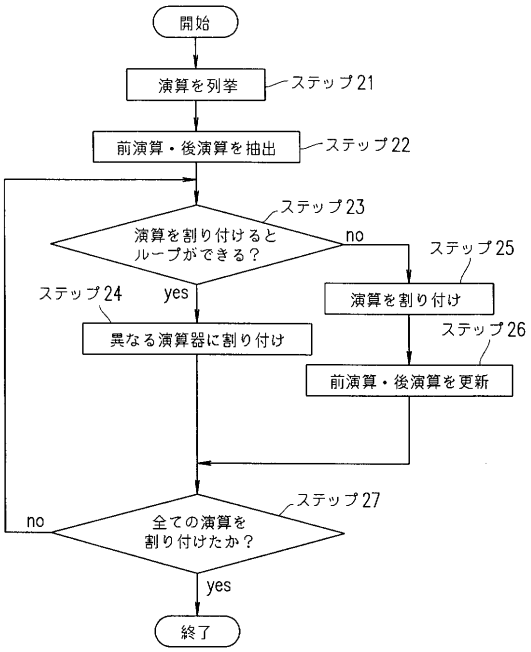


面積 27

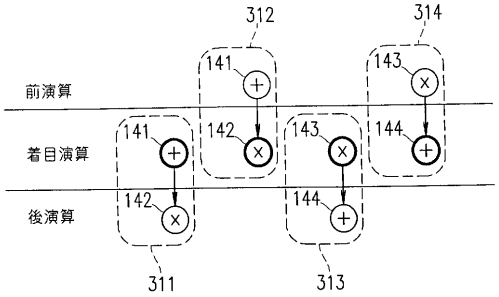
【図２３】



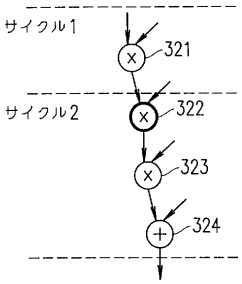
【図２４】



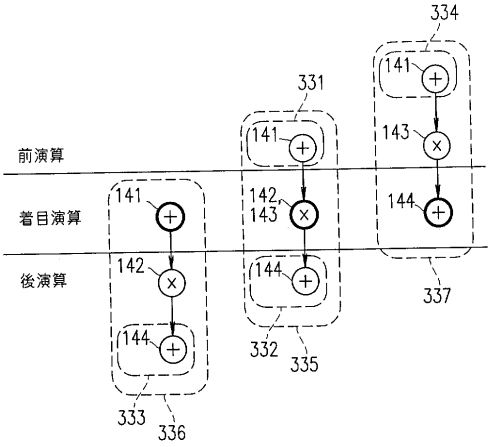
【図２５】



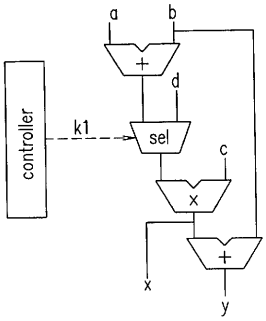
【図２６】



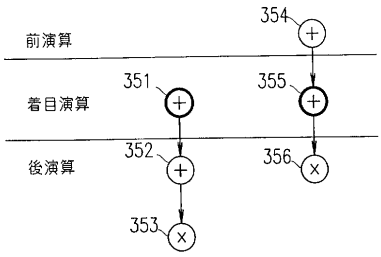
【図２７】



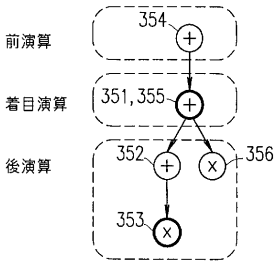
【図２８】



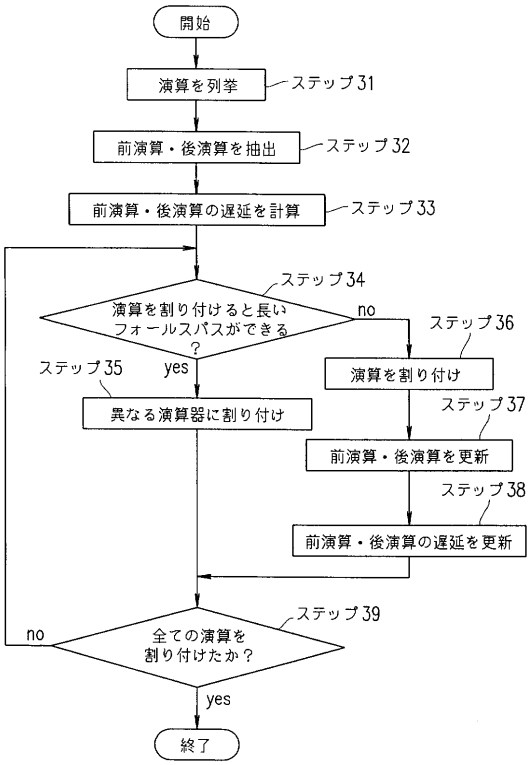
【図２９】



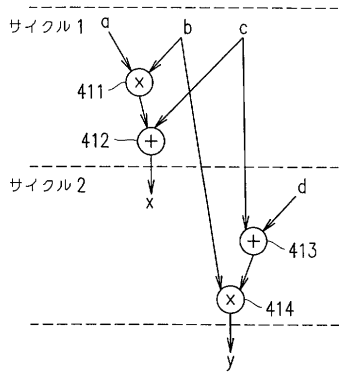
【図３０】



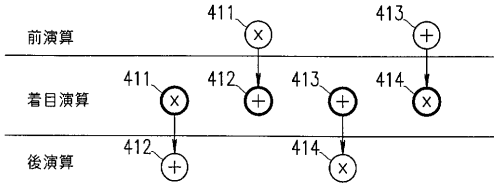
【図３１】



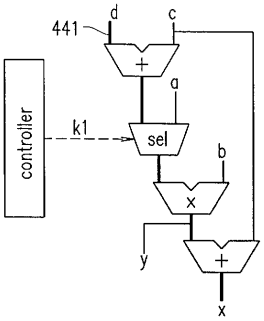
【図３２】



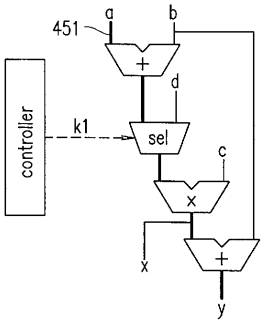
【図３３】



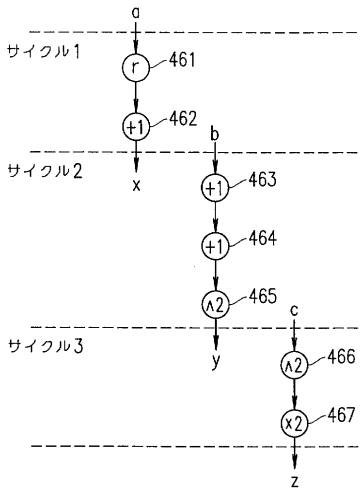
【図３４】



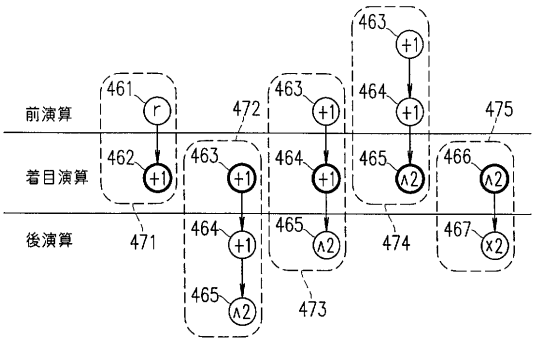
【図３５】



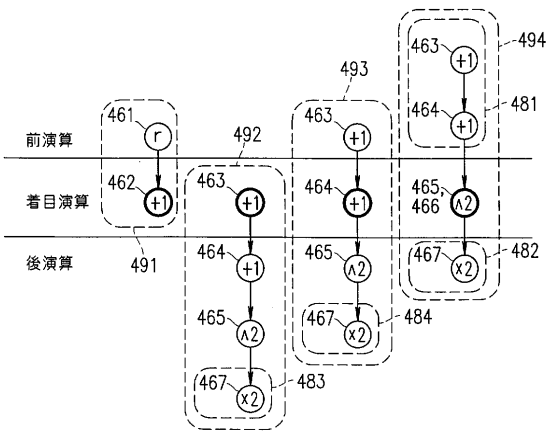
【図３６】



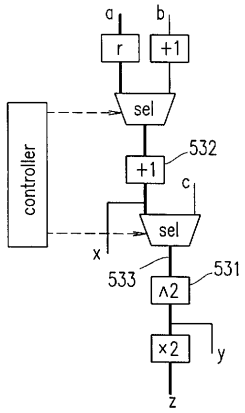
【図３７】



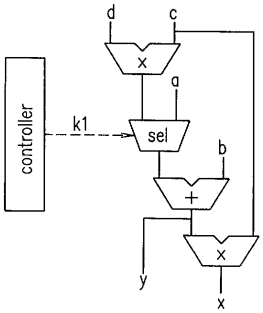
【図３８】



【図３９】


a
b
r
+1
sel
controller
532
+1
c
sel
x
533
531
∧2
×2
y
z

【図４０】


d
c
x
a
controller
k1
sel
b
+
x
y
x