設 計 自 動 化 60−13
（1991．12．12）

端子位置を考慮した論理多段化の一手法

矢野 純一† 安浦 寛人‡ 田丸 啓吉†

†京都大学 工学部

‡九州大学大学院総合理工学研究科

概説

従来の論理合成では回路の論理のみを考えて回路合成を行なっていた。しかし、論理合成の段階でレイアウトが考慮できれば性能の良いチップが得られると考えられる。本稿ではこのようなレイアウトを考慮した論理合成の一手法として端子位置を考慮した論理多段化の手法を提案する。まず、端子位置を考慮した論理合成を行なうための新たなパラメータである入力変数間の距離を定義し、それを使った論理多段化の手法について説明する。最後に実験により本手法の効果を確かめている。

A Method of Logic Synthesis with Consideration for Terminal Location

Junichi YANO†, Hiroto YASUURA‡, and Keikichi TAMARU†

†Department of Electronics, Kyoto University

‡Department of Information Systems, Kyushu University

ABSTRACT

In the conventional logic synthesis, only logical information is used for synthesis. If layout information is taken into consideration in logic synthesis phase, it is expected final layout of the circuit becomes better. In this paper, we propose a method of logic synthesis with consideration for terminal location as one of layout-driven logic synthesis methodology. A new parameter, distance among input variables, is introduced and a method of logic synthesis using the parameter is presented. We show several examples to evaluate the new method.

1 はじめに

近年のLSIの高集積化にともなってLSIの自動設計の必要性はますます大きくなっている。LSIの設計は一般に仕様から機能設計、論理設計、レイアウト設計というように階層的に設計が行なわれる。これらのうち論理回路からレイアウトを設計するレイアウト設計についてはすでに自動設計ツールが実用化しておりLSIの設計に広く使用されている。またRTレベルから論理回路を設計する論理設計についても多くの研究が行なわれてきており[1]、一部実用化に至っている。

論理合成の合成する対象回路は順序回路と組合せ回路に分類することができるが、順序回路の合成は状態割り当てと組合せ回路の合成という問題に分割することができる。よって論理合成においては組合せ回路の合成問題が基本となる。組合せ回路の合成法として回路を2段回路で構成する場合と多段回路で構成する場合がある。2段回路は設計が簡単である反面、レイアウト面積、が大きくなるという欠点を持つ。それに対して2段回路をさらに多段化した多段回路では設計が複雑になるがレイアウト面積は小さくなる。これらの2段回路の合成、多段化などに関して多くの研究が行われ、報告されている。しかし従来の論理合成ツールではそのほとんどが回路のトポロジーの情報のみを考えて回路の合成を行っており、回路のレイアウトなどのジオメトリーの情報は考慮していないのが普通であった。これは、論理設計の段階でレイアウトを考慮しようとすると問題がより複雑になってしまうためである。しかし論理設計の段階でレイアウトを考慮することが出来ると、最終的に出力されるチップの性能(レイアウト面積、遅延)の向上を図ることが出来ると思われる[4][5]。

本稿ではそのようなレイアウトを考慮した論理多段化の一手法として、端子位置を考慮した論理多段化という手法を提案する。そして回路の多段化時に使用する評価関数を従来のリテラル数に加えて変数間の距離というパラメータを加えたものとし、それを実際の合成ツールに組み込んでその有効性を確かめている。変数間の距離とは外部入力変数や中間変数に対応する端子やゲートの位置を多段化時に考慮するために今回新たに提案するパラメータであり、論理の多段化におけるレイアウトの影響を考慮するために利用する。

以下に本稿の構成を示す。まず第2章でレイアウトを考慮した論理合成について述べ、新たなパラメータとしての入力変数間の距離を定義し、そしてそれを導入したときの論理多段化の手法及びその評価関数について述べる。第3章では論理多段化のアルゴリズムについて述べ、回路の変形時に必要な回路の再配置について述べる。第4章では実際に作成したプログラムで本手法の効果を調べ、実験結果について考察する。そして第5章でまとめる。

2 端子位置を考慮した論理多段化

2.1 レイアウトを考慮した論理合成

従来の論理合成ではレイアウトを考慮することはほとんど行われていなかったが、自動合成ツールによって作成された回路のレイアウトにおいて、その面積の50%以上が配線面積になっていることを考えると配線による性能(面積、遅延)の差は無視できないものであり、論理設計の段階でレイアウトを考慮することは重要であると思われる。また優れたLSI設計者がLSIを手設計する場合、論理設計が終ってレイアウト設計を行う段階でフロアプランを行うのではなく、機能設計の段階でフロアプランを行う(図1)というほうがより好ましい。そしてこのような場合には機能設計から論理設計へレイアウト情報が渡されるので、このレイアウト情報を有効に活用した論理合成を行なうべきである。このような観点からもレイアウトを考慮した論理合成は重要であるということが出来る。

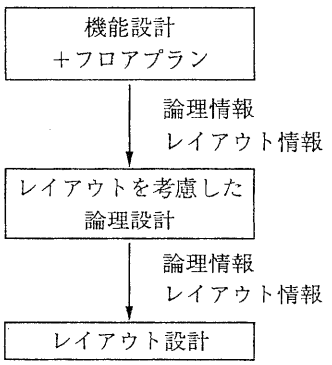

図1: フロアプランを先に行なうLSIの設計

レイアウトを考慮した論理合成は、合成する回路の性質

によって以下の2つの合成法に分類できる。

1. RTL記述に回路の論理情報を記述するだけでなく、レイアウト情報として各素子、モジュールの配置、配線情報までも記述しておく。そして論理合成の段階ではそれらの配置、配線情報を出来るだけ変更することなく回路の合成を行う。

2. 回路の論理情報以外には回路のレイアウトの形状、大きさ、入出力端子位置等の外部とのインターフェースのみをレイアウト情報として与えておく。そして論理合成の段階ではそれらの情報を考慮して合成を行う。そして内部の各素子の配置は論理合成側が行う。

これらのうち1.は回路の論理が規則的であり、RTL記述の段階でレイアウトのかなりの情報が記述できる場合、例えばデータパス系の回路を合成する場合などに適用できると考えられる。ただしこの場合は機能記述言語自体にレイアウトに関する情報を記述できるようになっていなければならない[2]。特にデータパス系の回路では実際に各素子の配置が最終的な回路の性能に大きく影響することが知られており、人手設計、モジュールジェネレーターなどでは論理合成とレイアウト合成を同時に行なっている。

これに対してコントロール系の回路では回路の論理は必ずしも規則的ではなく、レイアウトの情報としては外部端子位置やモジュールの形状などの情報しか与えることができないので、2.のような手法を取るのがよいと思われる。この場合は従来の論理合成の手法を部分的に変更することで対応できると考えられる。

今回提案する手法はコントロール系の回路の合成を対象として考えており、すなわち上記2.の場合を仮定している。

今回行なった外部端子位置を考慮した論理多段化においては、論理多段化を行なう際に以下のようなことを仮定している。

1. 回路のレイアウトの形状は矩形である

2. 回路のレイアウトはスタンダードセル(ゲートアレイ)で実現される

3. 回路の論理は2段簡単化された形で与えられる

4. 回路のレイアウト情報としては回路のレイアウトの大きさ(縦、横の寸法)、入出力端子位置(レイアウトの周上にある)が与えられる

そしてこのような仮定のもとでできるだけレイアウト面積(ゲート面積、配線面積)を小さくするような多段論理回路を生成し、また各素子の概略的な位置も出力する。

2.2 変数間の距離

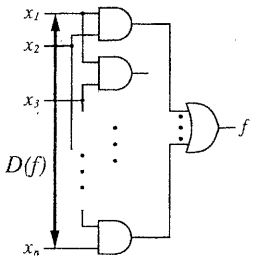

図2: 入力変数間の距離 D

レイアウトを考慮した論理合成を行うには、与えられたレイアウト情報を活かさなくてはならない。しかしレイアウトを考慮することによって処理時間が大幅に増大することはさけなくてはならない。そこで今回提案する手法では論理多段化の手法は従来使用されてきた手法を使い、その評価関数のみを変更することによってレイアウトを考慮できるようにした。これによって処理時間の大きな増加は避けられている。

従来論理関数の評価関数としてはテラル数が主に使用されてきたが、これだけではレイアウトを考慮することはできない。そこでここでは新たな評価関数として入力変数間の距離というものを導入する。論理関数 f に対する入力変数間の距離とは論理関数 f(外部入力変数及び中間変数によって表されている) $f(x_1, x_2, x_3, \cdots, x_n)$ に対して

$$D(f) = \max_{1 \le x_i, x_j \le n} d(x_i, x_j)$$

で定義されるものである(図2)。ただし $d(x_i, x_j)$ は変数 x_i と x_j に相当する外部入力端子や、ゲートのレイアウト上の座標間の距離[1]である。そしてこのような入力変数間の距離 D がレイアウトにおける配線長をある程度反映すると考えられる。例えばファンインの同じANDゲートでもそれらの入力変数の変数間の距離 D が小さい方が配線長が短くなるのでより優れたものであるということが出来る。

[1]ここで座標間の距離とはユークリッド距離のことである

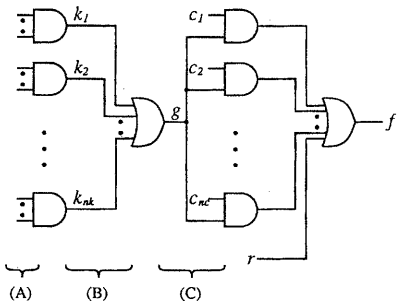

図 3: 因数分解による多段化

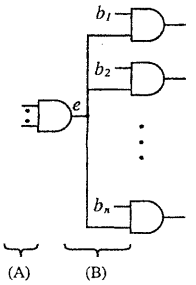

図 4: ゲートの抽出

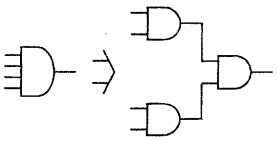

図 5: ファンイン調整

2.3 多段化手法及び評価関数

今回提案する手法の論理多段化の手法には一般的に使用されている Weak division 法を利用した因数分解 [3] を改良して使用している。因数分解による論理回路の多段化とは、与えられた各論理式の共通因子 (カーネル) を新たに中間変数で表し、共通因子を中間変数で置き換えることによって論理関数の簡単化を行うものである。ここで論理関数 f においてカーネル $g = \sum_{i=1}^{n_k} k_i$ で $f = \sum_{i=1}^{n_c} \sum_{j=1}^{n_k} c_i \cdot k_j + r$ を置き換えたときに f は以下のようになる。

$$f = \sum_{i=1}^{n_c} \sum_{j=1}^{n_k} c_i \cdot k_j + r = \sum_{i=1}^{n_c} c_i \cdot g + r$$

$$g = \sum_{i=1}^{n_k} k_i$$

n_c コカーネルの数　n_k カーネルの積項数
c_i 各コカーネル　k_j カーネルの各積項

この式を AND-OR 回路で構成したものが図 3である。さてこのときの評価関数であるが、まずリテラル数の減少だけを考えた場合の評価関数 fm は次式のようになる。

$$fm = (n_c - 1) \cdot \sum_{i=1}^{n_k} lk_i + (n_k - 1) \cdot \sum_{i=1}^{n_c} lc_i - n_c$$

lk_i カーネルの各積項に含まれるリテラル数
lc_i 各コカーネルに含まれるリテラル数

次に前節で述べた入力変数間の距離を導入した評価関数を考える。関数 f を中間変数 g で置き換えたことによって新たに生じる配線は

1. カーネルの各積項を実現する AND ゲートの入力線 (図 3(A))
2. カーネルの各積項の論理和をとる OR ゲートの入力線 (図 3(B))
3. 中間変数とコカーネルの積をとる AND ゲートの入力線 (図 3(C))

の 3 つがある。そして各配線長を L_A, L_B, L_C とすると、これらは入力変数間の距離 D を使ってそれぞれ近似的に次のように表すことができる。

$$L_A = \sum_{i=1}^{n_k} D(k_i)$$

$$L_B = D\left(\sum_{i=1}^{n_k} k_i\right)$$

$$L_C = \sum_{i=1}^{n_c} D(c_i)$$

そしてリテラル数の減少にこの入力変数間の距離で表現した各配線長を評価関数に加えることによって、評価関数 fm は次のようになる。

$$fm = (n_c - 1) \cdot \sum_{i=1}^{n_k} lk_i + (n_k - 1) \cdot \sum_{i=1}^{n_c} lc_i - n_c - K(L_A + L_B + L_C)$$

ただし K は定数であり、リテラル数の増減と配線長の増減のトレードオフを決定するものである。これはテクノロジーやレイアウト手法によって決定される。

次に Weak division とは別に多段化の手法として複数の AND(OR) ゲートから共通な入力 e を括り出すというゲートの抽出 (図 4) ということを行っている。この場合の評価関数についても考える。まずリテラル数の減少だけを考えると評価関数 fm は

$$fm = (n-1) \cdot l_e - n$$

n ゲート数
l_e 共通な入力の数

となる。そしてこのような変形を行ったことによって新たに生じる配線は図 4(A) および (B) であるが、それらの配線長 L_A, L_B はそれぞれ

$$L_A = D(e)$$
$$L_B = \sum_{i=1}^{n} D(b_i)$$

e 共通な入力
b_i 各ゲートの非共通な入力

となる。そしてこれをも加えた評価関数は以下のようになる。

$$fm = (n-1) \cdot l_e - n - K(L_A + L_B)$$

またその他の回路の変形としてファンイン制限を越えたゲートの分解 (図 5) もあり、これも同様な評価関数を与えることによってレイアウトを考慮した分解が出来る。

3 多段化のアルゴリズム

3.1 多段化の処理

多段化の処理は図 6に示すような流れで進められる。そして各段階では次のような手順で処理が行われる。

1. 与えられた論理式から全ての因子を取り出す。
2. 取り出した因子の中から評価関数が最大となるような因子の組を選択する。
3. 選択された因子によって回路の論理の変換を行い、各素子の再配置を行う。

そしてこのような処理を評価関数が 0 以下となるまで回路に適用し、回路の多段化を行う。次節以降で各処理について具体例を示しながら説明する。

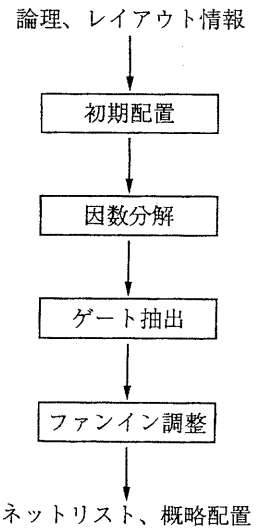

図 6: 多段化の処理の流れ

3.2 因子の取り出し

まず回路の論理が与えられると全ての因子が取り出される。その際には評価関数を計算するのに必要な情報 (リテラル数、入力変数間の距離) も同時に計算しておく。例えば 4 入力 (それぞれ a, b, c, d とし、それぞれの座標を 1, 2, 3, 4 とする) の関数 f(閾値 3 のスレッシュホールド関数)

$$f = abc + abd + acd + bcd$$

に関してそのコカーネル c、それに対応するカーネルの各積項 k、またそれぞれに対する入力変数間の距離などは表 1のようになる。

表 1: 関数 f の因子

c	lc	d_c	k	lk	d_k
ab	2	1	c, d	1, 1	0, 0
ac	2	2	b, d	1, 1	0, 0
ad	2	3	b, c	1, 1	0, 0
bc	2	1	a, d	1, 1	0, 0
bd	2	2	a, c	1, 1	0, 0
cd	2	1	a, b	1, 1	0, 0

3.3 因子の選択

すべての因子を取り出した後に、取り出された因子の中から評価関数が最大となる因子の組をヒューリスティックなアルゴリズム [6] で選択する。そのときには前章で挙げた評価関数にリテラル数、入力変数間の距離等を代入して評価関数を求める。ただし因数分解の際の評価関数中の

$$L_B = D\left(\sum_{i=1}^{n_k} k_i\right)$$

だけはその都度計算する。関数 f の各因子による評価関数 fm は次のようになる。

$$\begin{aligned} f &= ab(c+d) + acd + bcd \quad fm = 1 - 2K \\ &= ac(b+d) + abd + bcd \quad fm = 1 - 4K \\ &= ad(b+c) + abc + bcd \quad fm = 1 - 4K \\ &\vdots \\ &= cd(a+b) + abc + abd \quad fm = 1 - 2K \end{aligned}$$

そしてこれらのうち評価関数が最大となる (カーネル, コカーネル)$= (c+d, ab)$ or $(a+b, cd)$ が因子として選択される。

3.4 回路の再配置

因子が選択されるとまず回路の論理の変形が行なわれる。すなわち関数 f の例においては

$$\begin{aligned} f &= abg + acd + bcd \\ g &= c + d \end{aligned}$$

のようになる。今回の手法においては、回路の論理式の変形を行うときに各素子の再配置も行わなくてはならない。これは次の理由によるものである。

- 入力変数間の距離を計算する際に中間変数のレイアウト上の座標が必要である。
- 最終的な出力として多段化した回路のネットリストに加えて、各素子のレイアウト上の位置も出力する。

再配置においては総配線長が最小となる配置が最もよいと思われるが、再配置を行うことによって計算時間が大幅に増大することは好ましくない。また、各素子の配置は因子の選択後の回路の変形ごとに変化するのでそれほど厳密な配置を求める必要はない。よって再配置のアルゴリズムはある程度精度は低くてもよいが、処理時間が大きくかかるものであってはならない。そこで以下のようなアルゴリズムで各素子の再配置を行っている。

1. 各素子を適当な位置に配置する。
2. 各素子をそれらの入出力に接続されているゲート、外部入出力端子の重心の位置へ移動する。
3. 各素子の位置がほぼつりあえば終了、そうでなければ 2 へ戻る。

また多段化の初期時の 2 段回路の配置も同じアルゴリズムで配置を行っている。

4 評価、考察

本手法で提案したアルゴリズムの有効性を確かめるために、入力変数間の距離を評価関数に加えた場合と、加えなかった場合について種々の回路 [7] に対して多段化の結果を比較した。評価項目は概略配線長である。ただし多段化の終了時には、各素子の配置はかなり偏ったものとなっている場合がほとんどである。実際のレイアウトを考えるとこのような状態の配置で概略配線長を比較しても意味がないので、各ゲートの大きさ、スタンダードセルの列の数等を考慮して実際のレイアウトにおける配置に近い状態にしてから (図 7に例を示す) 概略配線長を比較した。その結果が表2である。この表における概略配線長とはレイアウトを考慮しなかった場合の概略配線長を 100 としたときのレイアウトを考慮したときの概略配線長である。そして各回路に対して 10 通りずつ外部入出力端子位置を適当に与えて実験を行ないそれぞれの概略配線長を記録し、それらの平均値、最大値、最小値を表にしてある。

この表より、従来のリテラルだけを評価関数として使った場合に比べて入力変数間の距離というパラメータを加えた評価関数を使用した方が概略配線長が平均で 0 ～ 9% 減

表 2: 実験結果

回路名	入力数	出力数	積項数	概略配線長		
				平均	最大	最小
xor5	5	1	16	95.9	102.4	89.5
inc	7	9	34	95.8	99.4	91.3
rd53	5	3	32	96.5	101.6	90.4
rd73	7	3	141	92.1	98.1	86.2
5xp1	7	10	75	99.8	103.5	96.5
9sym	9	1	87	96.3	100.3	91.3
sao2	10	4	58	91.6	104.1	62.0

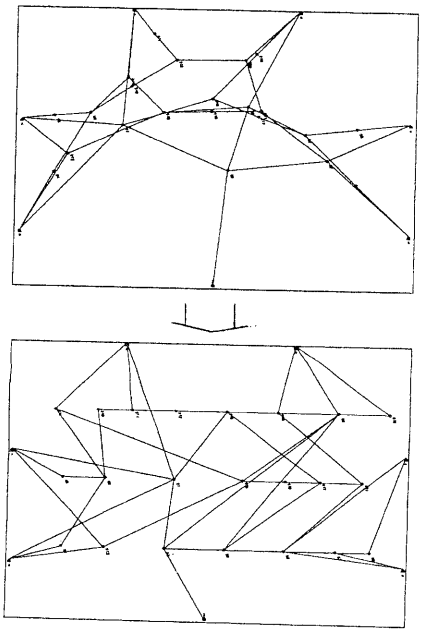

図 7: 評価のための配置の変更

少している。ただし最悪の場合には従来の手法より悪い結果が出ているものもある。また例題によって今回の手法の効果もまちまちである。

5 おわりに

本稿では、レイアウトを考慮した論理多段化の一手法として端子位置を考慮した論理多段化の手法を提案した。そして新しく導入した入力端子間の距離という評価関数について述べ、多段化における効果を実験によって確かめた。

本手法を有効にするためには 2 で述べたように

- 機能設計の段階でフロアプランを行い、コントロール系回路についてはそのモジュールのレイアウトの大きさ、形状、端子位置を決定する。

ことが必要である。また

- レイアウト合成ツールと接続し、回路のネットリストだけでなく、各素子の配置データを渡せるようにする。

ことも必要である。また今後の課題としては

- 組合せ回路の多段化の他の手法へ本手法の応用する。
- 論理合成の他の問題 (順序回路合成など) への拡張する。

などがある。

参考文献

[1] R.K.Brayton, *"Multi-level Logic Synthesis"*, ICCAD-90 Tutorial, 1990.

[2] V.G.Moshnyaga, 安浦 寛人, *"A Language for Designing of VLSI Module Generators."*, 本研究会予稿集.

[3] R.K.Brayton and C.T.McMullen, *"The Decomposition and Factorization of Boolean Expressions"*, Proc. of ICCAS-82, pp.49-54, 1982.

[4] M.Pedram and N.Bhat, *"Layout Driven Technology Mapping"*, Proc. of 28-th DAC, pp.99-105, 1991.

[5] P.Abouzeid, K.Sakouti, G.Saucier and F.Poirot, *"Multilevel Synthesis Minimizing the Routing Factor"*, Proc. of 27th DAC, pp.365-368, 1990.

[6] R.Brayton, R.Rudell, A.Sangiovanni-Vincentelli, and A.Wang, *"Multi-level Logic Optimization and The rectangle Covering Problem"*, Proc. of ICCAD-87, pp66-69, 1987.

[7] R.Lisanke, *" Logic Synthesis and Optimization Benchmarks User Guide."*, 1991.