

演算回路自動合成方式の検討

横田 孝義 戸次 圭介 浜田 亘曼

(株) 日立製作所 日立研究所

本文では、LSIを対象とした論理設計の効率向上を目的とした設計自動化手法の中で、主にデータベース系の機能モジュールを自動的に合成する手法について提案する。

ここではALUを例にスタティック、ダイナミック双方の回路構成手法を論理型言語を用いて蓄積し、それを用いた合成を試み、これによって必要な機能だけを持つALUが短時間に得られることを示す。また、ルールベースを用いたCMOSを対象とした論理合成手法についても提案し、これ等の手法を組み合わせることによって現実的な論理合成が可能となることを示す。

WG DA 34-3

"A METHOD FOR FUNCTIONAL LOGIC SYNTHESIS" (in Japanese)

by Takayoshi YOKOTA, Keisuke BEKKI, and Nobuhiro HAMADA
(Hitachi Research Laboratory, Hitachi Ltd., 4026 Kuji-cho,
Hitachi-shi, Ibaraki-ken, 319-12 Japan)

In this paper, we propose a method for functional logic synthesis which is based upon design models and a rule based multi-level logic synthesis method, which are implemented in logic programming language. This method is suited particularly to synthesize special purpose ALU's from a list of user specified operations which can be acquired from register transfer level behavioral specification. Results show a close agreement or less consumption of transistors compared with skilled manual designs.

演算回路自動合成方式の検討

横田 孝義 戸次 圭介 浜田 亘曼

(株) 日立製作所 日立研究所

1. はじめに

近年のVLSIの高集積化に伴い、それを活用して高機能のLSIを出来るだけ短期間に設計したいという要求が高まっている。実装、診断等の技術は古くから研究が成され実用化されているものが多いが、VLSIの機能設計や論理設計は、その作業のほとんどが人手で行われているのが現状である。

そこで本論文では、これらの機能設計や論理設計の負担を軽減するための検討結果の一つである演算回路モジュールの合成方法について述べる。

2. システム構成

本研究では、主にMOSのマイクロプロセッサ等のシステムLSIのデータバス機能モジュール自動合成、およびマイクロプログラムを前提とした制御論理の自動合成の実現方式の研究を行っている。

また、本報告で述べる演算回路合成、テクノロジー依存多段論理合成機能は、各種の機能モジュールのパラメトリックライブラリ作成と、制御論理系で必要となるデコーダ等をランダム論理で自動合成するために用いる。

本研究で開発目標としている論理設計支援システムでは論理設計者(ユーザー)は、本システムへ設計対象であるハードウェアのデータバスを図形エディタ等を介して入力し、かつ、それを用いて実現したい機能をレジスタ転送レベルの動作仕様として入力する。

これだけの情報を元に計算機上での論理設計を可能にするためには次に示すような技術課題がある。

- 1) 機能モジュールライブラリの実現と充実
- 2) 動作仕様から制御論理の合成
- 3) デコーダ等のランダム論理の合成
- 4) テクノロジ依存論理変換
- 5) 生成されるネットリストから回路図を自動作画表示
- 6) ライブラリ登録容易化
- 7) 論理検証

本論理設計支援システムは、データバス系、制御系に関わらず、設計モデル群とその詳細化手続き群を計算機内に蓄積することにより自動設計を行うことを基本としており、将来的には内部に用意されている設計モデルで対処出来なければ、設計者が新たに設計モデルを登録出来るようにすることを考えている。また、設計者の判断が必要な部分の対話性を重視しようとしている。ここで言う設計モデルとはデータバス系では例えばALUの設計モデルであり、制御論理系では基本マイクロ命令を用意した制御回路モデルを指す。このようなモデルの上で、制御系であれば動作仕様をマイクロ操作列に展開し、必要なデコーダを生成したり、データバス系であれば必要な演算機能を有するALUを合成し、その演算制御コードを制御論理に伝達するものである。

3. 演算回路自動合成

ALU、シフタ等の使用頻度の高いデータバスモジュールはライブラリの形でCADシステム内に登録しておく必要があるが、単なるデータとして格納するだけでは効率が悪く、また設計変更が生じた場合、その変更をシステム上で矛盾の混入無しに利用

することが困難となる。従って、出来るだけパラメトリックな形のライブラリとして登録する手法の確立が長期的な課題である。近年、米国においてシリコンコンパイラが注目されているのはこの方向性を示唆したものと考えられる。ここでは機能モジュールのパラメトリックな合成問題としてALUの合成を検討した。

我々は種々のLSIをケーススタディーし、標準的な機能モジュールを基本ライブラリとする検討を行っている。これ等は、通常のライブラリとしての意味を持つ以外に、その構成方法自体を登録したパラメトリックなライブラリとしての意味を持つ。

この中で、パラメトリックなライブラリは、

- 1) 機能に対してパラメトリック
- 2) 構造に対してパラメトリック

の2つの性格を有し、前者はALUの演算子に代表されるものであり、後者はレジスタ等のビット数に代表されるものである。このようにパラメトリック化を行う必然性は、単にライブラリとして蓄積するのに対し、構成手法を蓄積したほうが不要部分の削除が可能となり、ユーザ仕様に適合した最小構成のモジュールが得られる等の効果があり、ライブラリとしての蓄積効果の増大が期待されるからである。

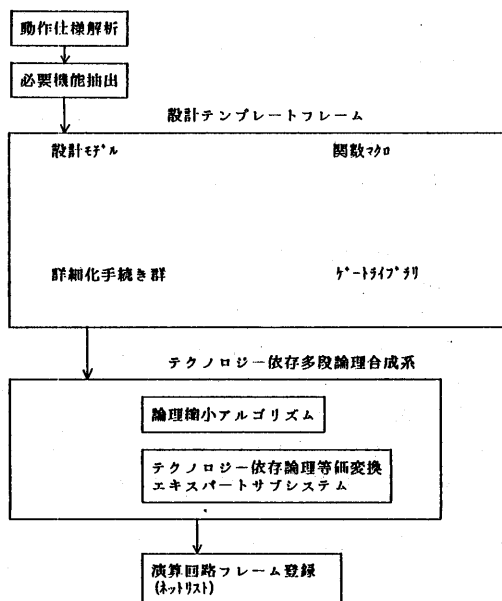


図1 演算回路合成サブシステムの構成

機能モジュール合成の具体的な内容は、所望の機能を有する機能モジュールの詳細回路情報（ネットリスト）と、機能モジュールが所望の機能を実現するための制御点情報、及び制御コードの自動生成を行うことである。

ALUの場合の構成手法はスタティックなものとダイナミックなものがあり、各々の設計モデルを用意する。このような演算回路の合成を行う演算回路合成サブシステムの構成を図1に示す。

演算回路の設計手法は設計テンプレート内に1) 設計モデル、2) 関数マクロ、3) 詳細化手続き群、4) ゲートライブラリとして分類して登録する。

また、この設計テンプレートを用いても設計出来ないランダム論理は図中のテクノロジー依存多段論理合成系によって合成する。この部分は、PLAの論理縮小アルゴリズムを利用した論理関数簡単化処理と、それをMOS LSIに適した回路に変換するルールベースからなる処理系で構成してある。

3. 1 ダイナミックALUの合成

ダイナミックな動作を伴うALUは占有面積が小さくCMOS LSIで多用される。

そこで、本節ではこのようなALUを自動合成するための処理について述べる。

ここで用いる設計モデルは、いわゆるマンチェスター桁上げ型ALUを対象としたものである。そのために、図2(A)に示すMead & Conway^{1), 2), 3)}等の提唱するビットスライスの汎用マンチェスター桁上げ型ALUに多少手を加えた物を設計モデルとして用いる。本手法では、キャリー伝播論理P、キャリー抑止論理K、ALU出力論理Rを図2(b)の仮想的な万能論理ブロックで実現することを出発点とする。ただし、万能論理ブロックの形のままでは最下キャリーを含めたALU制御コードのbit数が演算の数に無関係に13bitも要し、マイクロプログラムのメモリの増大を招き実用的でない。

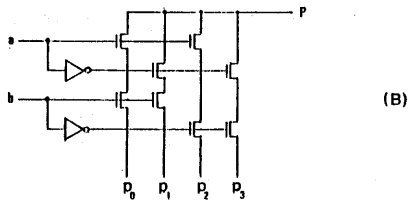
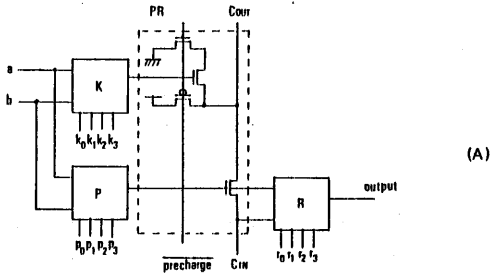
マンチェスター桁上げにおいて、キャリー伝播論理P、抑止論理Kの組合せは図2(C)で示す意味を持つ。図2に示すALUモデルにおいて、各万能論理ブロックの出力信号P、K、Rは次式で示すように入力A、Bとパラメータ P_0, P_1, P_2, P_3, K_0 。

$K_1, K_2, K_3, R_0, R_1, R_2, R_3$ によって規定される。

$$P = P_0AB + P_1\bar{A}\bar{B} + P_2\bar{A}B + P_3\bar{A}\bar{B} \dots\dots\dots (1)$$

$$K = K_0AB + K_1\bar{A}\bar{B} + K_2\bar{A}B + K_3\bar{A}\bar{B} \dots\dots\dots (2)$$

$$R = R_0PC_{in} + R_1\bar{P}C_{in} + R_2\bar{P}C_{in} + R_3\bar{P}C_{in} \dots\dots\dots (3)$$



P, K	C_{in}, C_{out}
0 0	set C_{out} to 1
0 1	set C_{out} to 0
1 0	$C_{in} \rightarrow C_{out}$
1 1	set C_{in}, C_{out} to 0

図2 ダイナミックALU設計テンプレート

図2(C)のキャリー伝播論理の決定手順は煩雑であるが自動化可能であり、一度生成して以降はそれを参照するだけで良い。そこで以下のように必要な演算の集合を入力するか、あるいは設計対象の論理装置の動作仕様から必要な演算の集合を抽出し、それ等を解析し、2進コード化した制御コードと所望の演算機能を有するALUの構造を合成する手続きをPrologで実現する。

まず、ALUで実現する可能性のある算術論理演算は、関数として定義しておく。これ等の関数はPrologを用いて定義することにより双方向に働く。

上記、(1)、(2)、(3)式を拘束条件とし

て、関数化して登録した所望の演算の入出力関係を満足させるパラメータ $P_0, P_1, P_2, P_3, K_0, K_1, K_2, K_3, R_0, R_1, R_2, R_3$ を求める。

この処理によって得られるP, K, R論理の生成結果の例を図3に示す。これを見てわかるように、and, or等の論理演算を対象とする場合の解は一意的ではなく、 $P_1=K_1=1$ の場合を除く制約を課しても、キャリー伝播についてdon't careであるため、それぞれ8種類のコードが存在する。

一方、add, sub AB等の算術演算の場合は、キャリー伝播の条件を必要とするために、解は一意的になる。万能論理ブロックのままでALUを構成する場合は、これ等の解をマイクロプログラムのALU制御フィールド内で単にbitパターン形で用いれば良いが^{27, 37}、実際には問題があり、ランダムロジックに変換する必要がある。

```

code(and, [1, 0, 0, 0], [0, 1, 1, 1], [0, 0, 0, 1]).
code(and, [1, 0, 0, 0], [0, 1, 1, 1], [0, 0, 1, 1]).
code(and, [1, 0, 0, 0], [0, 1, 1, 1], [1, 0, 0, 1]).
code(and, [1, 0, 0, 0], [0, 1, 1, 1], [1, 0, 1, 1]).
code(and, [0, 1, 1, 1], [1, 0, 0, 0], [0, 1, 0, 0]).
code(and, [0, 1, 1, 1], [1, 0, 0, 0], [0, 1, 1, 0]).
code(and, [0, 1, 1, 1], [1, 0, 0, 0], [1, 1, 0, 0]).
code(and, [0, 1, 1, 1], [1, 0, 0, 0], [1, 1, 1, 0]).
code(or, [1, 1, 1, 0], [0, 0, 0, 1], [0, 0, 0, 1]).
code(or, [1, 1, 1, 0], [0, 0, 0, 1], [0, 0, 1, 1]).
code(or, [1, 1, 1, 0], [0, 0, 0, 1], [1, 0, 0, 1]).
code(or, [1, 1, 1, 0], [0, 0, 0, 1], [1, 0, 1, 1]).
code(or, [0, 0, 0, 1], [1, 1, 1, 0], [0, 1, 0, 0]).
code(or, [0, 0, 0, 1], [1, 1, 1, 0], [0, 1, 1, 0]).
code(or, [0, 0, 0, 1], [1, 1, 1, 0], [1, 1, 0, 0]).
code(or, [0, 0, 0, 1], [1, 1, 1, 0], [1, 1, 1, 0]).
code(ex_or, [0, 1, 1, 0], [1, 0, 0, 1], [0, 0, 0, 1]).
code(ex_or, [0, 1, 1, 0], [1, 0, 0, 1], [0, 0, 1, 1]).

```

図3 演算制御コード

必要な演算機能の数が、加算、減算、AND, OR等の基本的な数種に限定されている場合には、ここまでで検討したマンチェスター桁上げ型ALUのモデルではモデル自体の汎用性が高過ぎるために、たとえ論理の縮小を行っても、そのハードウェア量は実際にカスタムLSIで用いられているALUと大きく異なってしまう。そこで、小規模なマンチェスター桁上げ型ALUの基本論理部を設計する場合には図4に示すExclusive-OR論理を活用し、その内部に含まれるNORとANDを活用する。こ

この関係は、以下のようなマクロライブラリーとして定義しておく。

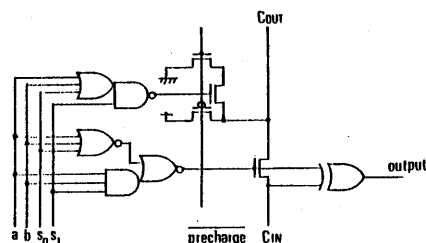
```
macro_ex_or(p_logic, [1, 0, 0, 0], codes, [s0, 1],
            [s1, 1], op, nand).
```

つまり、P 論理を逆論理で用いた場合は、K 論理

例えば、`add`, `and`, `or`, `ex-or` の4
機能を実現したい場合には以下に示す真理値データ
が得られる。

```
solution([s0, 1, s1, 0], op, ex_or, [1, 1, 1, 1]).
```

このようにして自動合成した3機能ダイナミックALUを図5に示す。



s_0, s_1	機能
0 0	$a \vee b$
0 1	-----
1 0	$a \oplus b \oplus C_{in}$
1 1	$a \wedge b$

図5 ダイナミックALUの合成例

3. 2 スタティック ALUの合成

スタティック論理を用いるALUの代表的なものとしてTTLのALUがある。

そこで、このスタティックALUを一般化した図6のALUモデルをテンプレートとして用意し、その上での論理合成を考えた。

スタティック型のALUは、 AB , $A\bar{B}$, $\bar{A}B$, $\bar{A}\bar{B}$ 等の積項を入力とする論理ブロックによってG論理（キャリー生成論理）を生成し、 A , B 等の単一項を入力とする論理ブロックによりP論理（キャリー伝播論理）を生成し、出力論理はP, Gの排他的論理和をとることによって得るモデルを採用した。

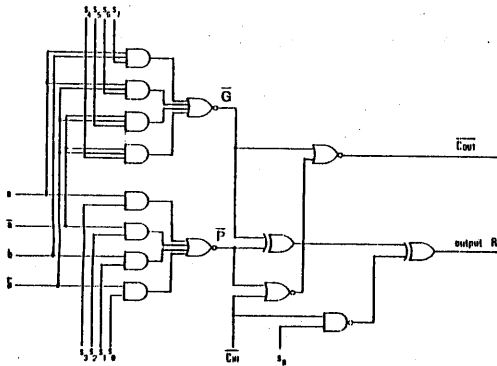


図6 スタティックALU設計テンプレート

このモデルを用いると通常の論理演算はGとPの排他的論理和の否定として得られるため所望の演算はそれを実現するための8本の制御コード $[S_0, \dots, S_7]$ をgenerate & test（試行錯誤）的に決定すれば良い。

この回路は冗長であるために各論理演算に対して1.6個の制御コードが存在する。そこでこれ等の中から結果的に最も制御線が少なく、それとともにゲート数も最小になる組合せを決定する必要があるが、算術演算を考慮すると1つの指針を得ることができる。算術演算ではキャリーの扱いが大きな拘束条件となることからALUの解の探索空間をかな

り限定することができる。

加算を例に考えると以下のような定式化が可能である。

$$\begin{aligned} \text{キャリー伝播の拘束: } C_{out} &= (A \oplus B)C_{in} + AB \\ &= (A + B)C_{in} + AB \dots (4) \\ \text{すなわち } P &= A + B \dots (5) \\ G &= AB \dots (6) \end{aligned}$$

また、加算結果はこのモデルにおいては、

$$\begin{aligned} R &= \bar{G} \oplus P \\ &= A \oplus B \dots (7) \end{aligned}$$

でなければならない。

従って、 $\bar{P} = \bar{A}\bar{B}$, $\bar{G} = \bar{A} + \bar{B} \dots (8)$ となる。

これを実現する制御コードは以下になる。

$$\begin{aligned} [S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7] &= [0, 1, 0, 1, 0, 0, 0, 1] \\ &\dots (9) \end{aligned}$$

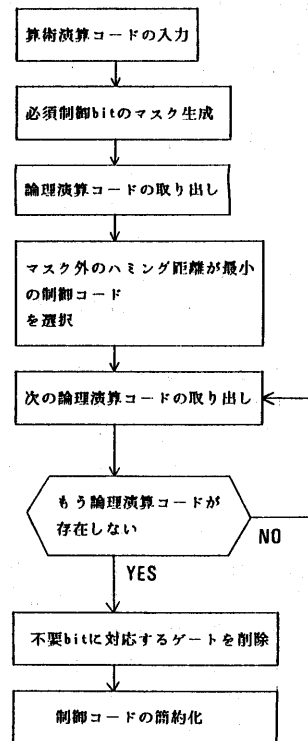
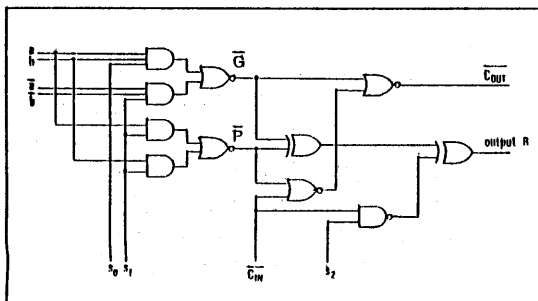


図7 スタティックALU合成の流れ

つまり、加算を含むALUを構成するためには S_1, S_3, S_7 を必ず含まなければならない。
同様に、減算等についてのキャリー伝播条件も求めることができ、キャリーを扱う場合にはスタティックの場合もダイナミックの場合と同様に制御コードが一意に決定される。従って、これ等の算術演算と、他の論理演算を含むALUを構成するためには、このような制御コードとのハミング距離が小さく、かつ、全体としての制御コードの長さが最小になるものを選択するのが妥当である。これを具体化した準最適解を求めるためのアルゴリズムを図7に示す。

このアルゴリズムを用いて合成された4機能ALUを図8に示す。



s_0, s_1	機能
0 0	-----
0 1	$a \oplus b \oplus C_{in}$
1 0	$a \wedge b$
1 1	$a \vee b$

図8 スタティックALUの合成例

3.3 CLAの構成

ここまで述べた手法はALUの基本算術論理演算機能を合成するものであったが、算術演算時の桁上げ信号の伝播の高速化手法もALU合成サブシステム内に格納する必要がある。特にCMOS回路ではマンチェスタ桁上げ型ALUの高速化を扱う必要がある。マンチェスタ桁上げの場合のルックアヘッド論理は、プリチャージされているキャリーラインをディスチャージさせる論理の設計ということになる。各ALUステージでのキャリーラインの

ディスチャージ論理は次式で与えられる。

$$\overline{C_{i+1}} = P_i \overline{C_i} + K_i \quad \dots\dots\dots (10)$$

(10)式より、4bitまでのキャリーラインのディスチャージを先見する論理を求めると以下のようになる。

$$\overline{C_1} = P_0 \overline{C_0} + K_0 \quad \dots\dots\dots (11)$$

$$\overline{C_2} = P_1 P_0 \overline{C_0} + P_1 K_0 + K_1 \quad \dots\dots\dots (12)$$

$$\overline{C_3} = P_2 P_1 P_0 \overline{C_0} + P_2 P_1 K_0 + P_2 K_1 + K_2 \quad \dots\dots (13)$$

$$\overline{C_4} = P_3 P_2 P_1 P_0 \overline{C_0} + P_3 P_2 P_1 K_0 + P_3 P_2 K_1 + P_3 K_2 + K_3 \quad \dots\dots\dots (14)$$

ここで、(14)式中の4つの項の内最初のものだけを用いるとキャリースキップ方式となる。実際のカスタムLSIの論理設計においては、これ等のルックアヘッドのための論理関数群をそのまますべて実現するとチップ面積が増大し問題となることが多いため、要求性能と面積とのトレードオフを考慮し、最適な方式選定が行われている。

これを実現するには上記論理式をシステム内に登録しておき、次に述べる論理合成を行う。

4. テクノロジー依存論理合成

CLAの基本論理式からの回路生成や、ダイナミックALUのK論理の生成等、論理設計の各部分でランダム論理の生成が必要となり、その自動化が望まれている。一般に、真理値表や論理関数から直接的にハードウェア化したのではゲート数やゲート段数が増大して不都合が多い。特にCMOSLSIの場合には、複合ゲートを巧妙に組み合わせる論理を実現するのが普通である。このようにCMOSに適した論理に変換するためには以下に示す様なAND, OR論理をNAND, NOR論理に変換することと、NAND-OR複合論理、NOR-AND複合論理を活用してトランジスタ数を削減させる必要がある。そこで、実現したい論理の真理値データを与えたら、その論理を縮小し、回路結線に変換し、それを特定のテクノロジーに変換する処理系を構成した。この部分の構成を図9に示す。

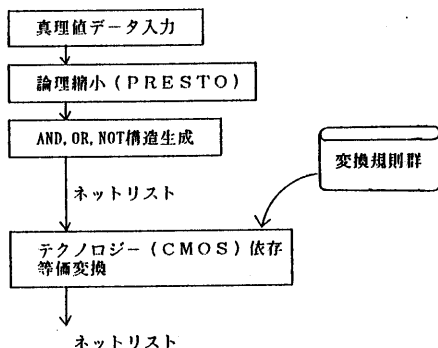


図9 テクノロジー依存論理合成系の構成

まず、真理値データが入力されると規定されていない入力の組合せに対する出力をdon't careにし、PLA論理縮小アルゴリズムを適用する。ここでは簡単のためにPRESTOを用いている。

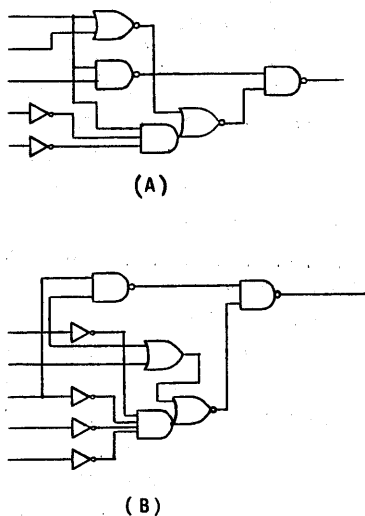


図10 K論理の合成例

次に、この2レベルの論理をそのままAND、OR、NOTのハードウェアに変換し、そのネットリストを自動生成する。その後、そのネットリストに対してCMOSテクノロジー変換を施すものである。この処理では、基本的にAND、OR、NOTから成るテクノロジー独立の回路結線データをプロダクションルールによって自動変換するものである。現在は局所的な変換をした後にCMOS特有のNAND-ORやNOR-AND型複合論理を多用してトランジスタ数の削減を図るものである。現在までにこのようなルールを30個実装した。

図10 (A)には4機能K論理、図10 (B)には5機能K論理の合成結果を示す。

5. ALU合成結果の評価

以上の検討を基にALUの合成を行って、その評価を行った。計算時間に関しては、図11に示すように、ダイナミックALUの場合7 KLIPSのPrologを用いて数10秒のオーダーであり、また計算時間の増加傾向も線形的であり、Prologの処理系の高速化を勘案するとほとんど問題にならない。スタティックALUもほぼ同様であった。

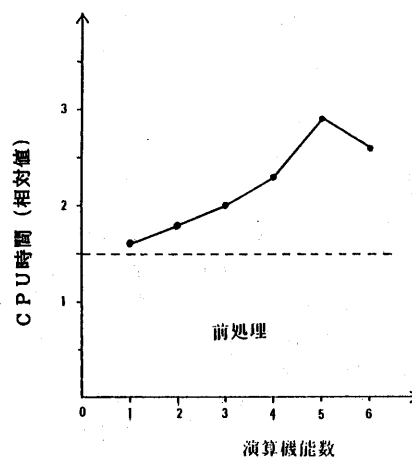


図11 処理時間

スタティックなALUは適切な比較対照が無く比較困難であるが、汎用多機能のTTLのALUより圧倒的にゲート数が少なく、また4機能のバイポーラALUと比較してもMOSトランジスタ換算で同等あるいは表1で示すように、より少ない素子数で実現出来ている。

従って、論理合成手順のような論理的な記述や、解の探索処理、回路のパターン照合等の処理は論理型言語Prologにより比較的容易に記述可能であるだけでなく、処理性能も特に問題にならないと考えられる。

表1 市販ALUとの比較

	市販品	本手法
P, G論理に要するTR数	32	24

6. 糸言論

本研究で得られた結論は以下の通りである。

1) パラメトリックな機能モジュールライブラリの作成という課題に対しALUを選び、その設計モデルをダイナミックな場合とスタティックな場合について検討し、各々の詳細化過程を含めてフレーム表現を拡張させた方式で実現し、実際に合成を試みた結果、人手による設計と同等のものが高速に得られることを確認した。

2) 上記詳細化過程で必要となるテクノロジー依存の論理変換処理をルールベースによる処理によって実現し、人手による結果と同等の物が高速に得られることを確認した。

3) 論理合成手順のような論理的な記述や、解の探索処理、回路のパターン照合等の処理は論理型言語Prologにより比較的容易に記述可能であることを確認した。

また、本研究を通して、ALU合成にとどまらず、設計問題に本質的に内在する解の探索問題の解法として冗長モデルを縮小して行くというアプローチと論理関数の実現に既存のゲートライブラリーをあてはめて問題を小さくするというアプローチは他の部分でも活用できると思われる。

<<<参考文献>>>

1. J. Newkirk and R. Mathews: The VLSI Designer's Library: Addison-Wesley, 1983.
2. C. Mead and L. Conway: Introduction to VLSI Systems: Addison-Wesley, 1980.
3. J. Mavour, M. A. Jack and P. B. Denyer: Introduction to MOS LSI Design: Addison-Wesley, 1983.
4. Kai Hwang: Computer Arithmetic, Principle, Architecture, and Design: John Wiley & Sons, 1979.
5. M. Morris Mano: Computer System Architecture: Prentice-hall Inc., 1976.
6. Aart J. de Geus et al: A Rule Based System for Optimizing Combinational Logic: IEEE Design & Test, Aug. 1985.
7. T. J. Kowalski et al: The VLSI Design Automation Assistant from Algorithm to Silicon: IEEE Design & Test, Aug. 1985.
8. Jay R. Southard: MacPitts: An Approach to Silicon Compilation, IEEE COMPUTER, Dec. 1983, pp. 75-82.
9. Brown, D. W.: A State Machine Synthesizer, Proc. 18th DA Conference, pp. 301-305 (1981)
10. 高木 茂: テンプレートを用いたALU論理合成: 信学 論文誌 Vol. J68-D, No. 7, 7月, 1985, pp. 1369-1375.
11. 横田 孝義, 戸次 圭介, 浜田 亘彦: フレーム表現によるCMOS論理機能ブロックの対話設計支援: 信学, 情報システム全国大会, 1985年11月
12. 同上: 論理設計知的支援用推論方式のプロトタイプینگ: 情処 VLSI CADへの知識工学の応用シンポジウム論文集, pp. 11-15, 1986年1月.
13. 同上: 論理設計知的支援用対話型仕様入力方式情処 第32回(昭和61年前期)全国大会講演論文集, 講演番号4L-1, 1986年3月.
14. 戸次 圭介, 横田 孝義, 浜田 亘彦: 論理設計知的支援用知識ベース構成方式: 同上, 講演番号4L-2, 1986年3月.