設計自動化　50－3
(1989. 12. 14)

算術演算回路の機能情報抽出

大村昌彦　安浦寛人　田丸啓吉
京都大学 工学部 電子工学科

あらまし　我々は以前より，論理回路レベルの設計記述を機能レベルの記述に自動変換する機能情報抽出の手法を提案してきている．この技術には，機能レベルにおける論理設計検証，機能シミュレーションモデルの自動作成，ドキュメント／マニュアルの作成支援など様々な応用が考えられ，その重要性が認識されてきている．この手法の基本は，与えられた回路の機能をある入力変数の値によって幾つかの場合に分けて論理式で表わすことであるが，論理式では簡潔に表現できない機能，例えば算術演算機能，などを如何に表現するかが問題となる．本稿では，このような算術演算機能を算術式を用いて表わす手法について考察を行う．

Extraction of Functional Information from Arithmetic Units

Masahiko OHMURA, Hiroto YASUURA and Keikichi TAMARU
Department of Electronics, Faculty of Engineering, Kyoto University
Kyoto 606, Japan

Abstract　We have proposed techniques of functional information extraction that transforms the design descriptions in logic circuit level to that in function level. The techniques are applied to various phases of VLSI design, such as logic design verification in function level, automatic generation of functional simulation models and making documents or manuals, and have been regarded important. The basic idea of the technique is to represent functions of given circuits using some logic formulas divided according to the value of input variables. But a problem occurs how we express the function which can not be clearly represented by logic formulas, such as arithmetic functions. In this paper, we discuss techniques to represent such functions using arithmetic formulas.

1 まえがき

ディジタル回路の自動設計の分野においては，論理合成や自動レイアウトなどのように上位レベルの設計記述を下位レベルの記述に変換する手法が主に研究の対象とされており，現実に回路の設計に用いられているシステムも多く作成されている．また，レイアウトのような下位レベルにおいては，マスクパターン記述を回路レベルの記述に逆変換する回路抽出の手法も確立されてきている．一方，機能レベルや論理回路レベルにおいては，下位から上位レベルへの設計記述変換に関する研究は，一部の研究機関で行われてはいるものの [1, 2, 3] ，まだその手法の確立には至っていない．このような状況において我々は以前より，論理回路レベルの設計記述を機能レベルの記述に自動変換する機能情報抽出の手法を提案してきている [4].

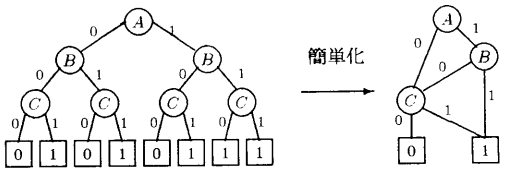
この技術には，(1) 機能レベルにおける論理設計検証，(2) 機能シミュレーションモデルの自動生成，(3) ドキュメント / マニュアルの作成支援など様々な応用が考えられ，その重要性が認識され始めてきている．昨今の回路規模の増大に伴って，論理回路レベルにおける完全な設計検証 (論理シミュレーション) はほとんど不可能になってきており，(1) に示したように機能レベルで論理の検証を行うという考えは以前から重要視されており，様々な手法の研究が行われている．しかしその多くは，論理設計に入る前に RT レベルなどで機能論理の検証を行うものであり，ゲート論理の検証を行うものではない．これに対し我々の手法では，実際に設計された論理回路レベルの記述をもとの機能レベルの記述に逆変換して，仕様として与えられた機能記述と比較するので，確実に論理設計検証が行える．(2) に挙げた機能シミュレーションモデルは，ここで設計された LSI が後に他の回路の 1 モジュールとして再利用される際，機能シミュレーションのために必要となるものである．この LSI が設計時に与えられた仕様通りに作成されていれば，機能設計の結果得られた機能記述をそのまま利用すればよいが，論理設計やレイアウト設計の途中において設計変更が生じた場合，実際に設計された回路は，元々与えられた機能を満たしていない場合がある．このような場合，一般にその変更に応じて元の機能記述も修正するということは行われないので，機能シミュレーションモデルは実際に設計された回路記述から作成されるものでなければならない．以上の他にも，機能情報抽出には様々な応用が考えられるが，レイアウトレベルからの回路抽出の技術も含め，一般に下位レベルから上位レベルへの自動記述変換の技術が発展していくと，(3) に示したドキュメントやマニュアルの作成支援，あるいは自動作成に利用できるのではないかと考えられる．

機能情報抽出の基本的手法は，与えられた回路の機能をある入力変数の値によって幾つかの場合に分けて論理式で表わすことである．このように，その値によって回路の機能を制御する働きを持つ入力変数を，コントロール系入力と呼ぶ．また，それ以外の入力をデータ系入力と呼ぶ．結局抽出された機能は，コントロール系入力の値によって分けられたデータ系入力から成る論理式で表わされる．これを表形式にしたものを，機能表と呼ぶ．一般に機能表は，真理値表ほどサイズが大きくならず，一つの論理式で表わすほど複雑にはならず，人間の目で見てわかりやすい表現となっており，先述の論理設計検証において与えられた機能記述と比較するのに適した表現である．しかし，コントロール系入力の値によって分割されてもまだその論理式が複雑な場合，あるいはその機能そのものが論理式ではわかりやすく表現できない場合には，問題が生ずる．例えば算術演算機能などは，論理式ではその機能を充分に表現出来ないので，どのように対応するかが問題となる．本稿では，このような算術演算機能を論理式ではなく算術式を用いて表現する手法について考察を行う．

2 組合せ回路の機能情報抽出システム FINES の概要

算術演算機能の抽出について述べるに先立って，本節では機能情報抽出の基本的手法について簡単に述べておく．

FINES(Functional INformation Extraction System) は，論理回路レベルの構造記述 (ネットリスト) を機能レベルの記述 (機能表または機能記述言語) に自動変換するための試作システムである．但し現在のところ，対象とする回路は組合せ回路に限定している．このシステムの特徴として，論理関数の内部表現として二分決定図 (Binary Decision Diagram:**BDD**) を用いたことが挙げられる．BDD についての詳細は，文献 [4, 5] 等を参照されたい．BDD は図 1 に示すように回路の入力変数を二分木の節点に対応させたものであるが，入力変数に順序付けがされているということ，及び簡単化して節点数を減らせるということ，という特徴を持っ



$A \cdot B + C$ の 二分決定図　　最簡形

図 1　二分決定図とその簡単化

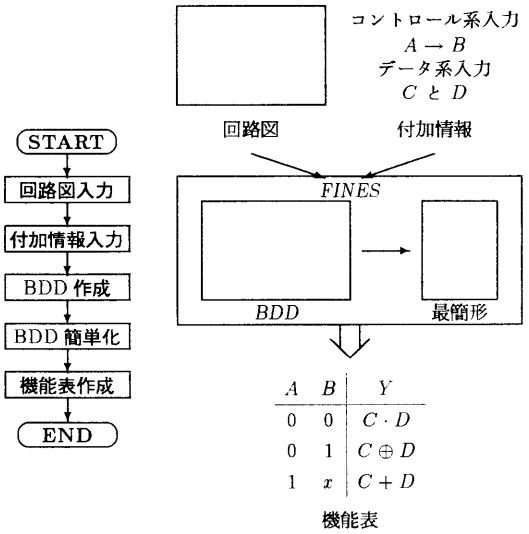


図 2　機能情報抽出の処理の流れ

ている. 入力変数の順序は簡単化の際に非常に大きな役割を果たす. 即ち, 入力変数の順序の付け方によって最簡形 (これ以上簡単化できない形) の節点数が異なり, またその順序をいったん定めれば最簡形が唯一得られる. 機能記述はこの簡単化された BDD から作成されるので, 出来るだけ簡潔な機能記述を得るためには, 出来るだけ節点数の少なくなる最簡形が得られる入力変数の順序付けを行う必要がある. この順序を知ることは一般に非常に難しいことではあるが, 我々の経験から, 回路の機能を制御する働きを持ったコントロール系入力を先に, それ以外のデータ系入力を後にするように順序付けを行えば, 比較的節点数の少ない BDD が得られることがわかっている. コントロール系入力とデータ系入力の区別は, その回路の設計者なら容易にわかることであるので, 機能情報抽出の際に手助けとなる情報として設計者自身がシステムに与える. このような情報を付加情報と呼ぶ. 更にコントロール系入力が複数ある場合は, 制御力の強い順に並べる. この順序付けもやはり付加情報として与えられる. また, データ系入力間の順序はそれほど問題ではない. 以上より, 機能情報抽出の処理の流れは図 2 のようになる.

3　算術演算機能の表現法について

前節で述べた機能情報抽出システム FINES を算術演算回路に適用した場合の問題点について, 加算回路を例にとって考察する.

1 ビット全加算器 (入力 C_0, A_0, B_0; 出力 F_0, C_1) の機能を論理式で表わすと,

$$\begin{cases} F_0 &= C_0 \oplus A_0 \oplus B_0. \\ C_1 &= C_0 \cdot (A_0 + B_0) + A_0 \cdot B_0, \end{cases}$$

となる. また, C_0 をコントロール系入力であると見なして, FINES を用いて機能表を作成すると,

C_0	F_0	C_1
0	$A_0 \oplus B_0$	$A_0 \cdot B_0$
1	$\overline{A_0 \oplus B_0}$	$A_0 + B_0$

となる. この機能記述は非常に簡潔に表わされてはいるが, 一見したところでは加算機能を表わしていることがわからない. しかし, 機能設計や論理設計に携わっている設計者ならば, このような論理式あるいは機能表を見れば, 1 ビット加算の機能を表わしていることがすぐにわかるであろう. したがって, 算術演算機能の抽出を行うためには, FINES を用いていったん論理機能の抽出を行い, 得られた結果に対応する算術演算機能で表わす, という手法が考えられる. これを自動的に行うためには, ある算術演算機能を論理式で表わすとどのようになるかをあらかじめ知識として蓄えておき, 抽出された論理式とパターンマッチングをとればよい. 実際にパターンマッチングの対象となるのは論理式ではなく内部表現の BDD である. なぜなら論理式は, BDD とは違って, 一つの関数を唯一の形で表現出来るという特徴を持っていないからである.

パターンマッチングをとるためには, 数多くの算術演算に対してその論理式表現をデータベースに蓄えておかねばならないが, 例えば 1 ビット加算と 2 ビット加算のように, 同じ機能でビット数の異なるものをそれぞれ蓄えておくのは, 非常に無駄が多い. この無駄を省くために, 一般に n ビットに対する加算機能の表現法を考える. 1 ビット全加算器の機能を BDD で表わすと, 図 3 のようになる. また, 2 ビット全加算器 (入力 C_0, A_0, A_1, B_0, B_1; 出力 F_0, F_1, C_2) の機能は, 図 4 の BDD で表わされる. ここで 2 ビット全加算器が 1 ビット全加算器を 2 つ直列に接続したものと考え, 1 ビット目から 2 ビット目への桁上げ信号を C_1 とすると, 2 ビット全加算器を表わす BDD は図 5 のようにも表わせることになる. これを見れば, F_0 と F_1 及び C_1 と C_2 の BDD がそれぞれ同じ形をしていることがわかる. これより, 一般に n ビットの全加算器の機能は図 6 に示した 2 つの BDD の繰り返しで表現出来ることがわかる. したがって, データベースにこの F_i と C_{i+1} の繰り返しパターンだけ蓄えておけば, 与えられた回路のビット数 n に応じて, $F_0, C_1, F_1, \cdots, C_{n-1}, F_{n-1}, C_n$ の BDD が順次作成され, 機能情報抽出の結果作成された出力 $F_0, F_1, \cdots, F_{n-1}, C_n$ の BDD とパターンマッチングをとることによって, この回路が n ビットの加算回路であるか否

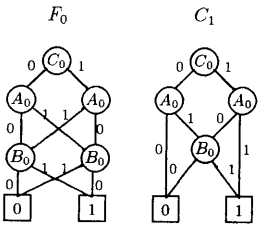


図3 1ビット全加算器のBDD

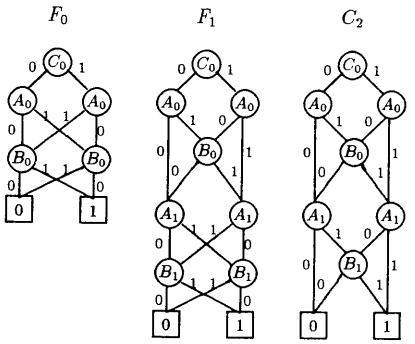


図4 2ビット全加算器のBDD

かが判定出来る. ここで注意すべき点は, パターンマッチングをとるのは与えられた回路の出力変数に対応するBDDだけであって, 中間変数として用いられた $C_1, \cdots, C_{n-1}$ のBDDはパターンマッチングには用いられないので, これらの中間変数に対応する信号線が回路に存在する必要はない, ということである. 即ち我々の機能情報抽出の手法では, 回路の入出力の関係だけから機能を決定し, 内部の構造は全く問題にしていないので, 桁上げ伝搬加算回路からも桁上げ先見加算回路からも同じ機能が抽出される.

4 FINESを用いた算術演算回路の機能情報抽出

前節では, 加算回路を例にとり算術演算機能の表現法について説明したが, 同様にその他各種の演算についても図6に示したような繰り返しパターンを記憶しておけば, 算術演算回路の機能情報抽出が行えると考えられる. 図6の繰り返しパターンは, 論理式で表わすと,

$$\begin{cases} F_i &= f(C_i, A_i, B_i) = C_i \oplus A_i \oplus B_i \ , \\ C_{i+1} &= g(C_i, A_i, B_i) = C_i \cdot (A_i + B_i) + A_i \cdot B_i \ . \end{cases}$$

$$(i = 0, \cdots, n-1) \ ,$$

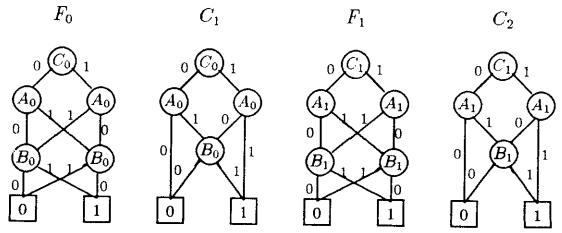
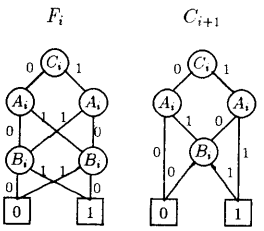


図5 中間変数 C_1 を用いた2ビット全加算器のBDD



$(i = 0, 1, \cdots, n-1)$

図6 n ビット全加算器のBDDの繰り返しパターン

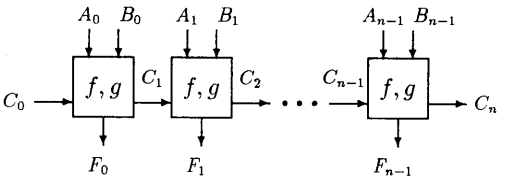


図7 一次元一方向繰り返し組合せ回路

となる. このような繰り返しパターンで表わされる演算は, 2数の加算, 減算, 論理演算, あるいはそれらに定数を加えたものや合成演算などが考えられ, いわゆる一次元一方向繰り返し組合せ回路 (combinational unilateral one-dimensional iterative circuit:**CUODIC**) [6] (図7参照) で表現出来る関数がこのように表わせる. 更に一般的に表わすと,

$$\begin{cases} F_i &= f(C_i^1, C_i^2, \cdots, C_i^k, A_i, B_i) \ , \\ C_{i+1}^1 &= g_1(C_i^1, C_i^2, \cdots, C_i^k, A_i, B_i) \ , \\ &\vdots \\ C_{i+1}^k &= g_k(C_i^1, C_i^2, \cdots, C_i^k, A_i, B_i) \ , \end{cases}$$

$$(i = 0, \cdots, n-1) \ ,$$

となる．乗算や左シフトなどはこの形式に入る．また，これまでに述べた形式は，F の第 i ビットが A, B それぞれの下位 i ビットに依存するが，右シフトや巡回シフトのように F の第 i ビットが A, B の全てのビットに依存する場合も考えられる．このように対象とする回路の構造を出来るだけ一般化することは，汎用的なシステムを作成する上で非常に重要なことではあるが，今回は図 7 で表わされるような回路に対象を絞り，FINES を用いて算術演算機能の抽出を行うシステムを試作した．以下にその処理の流れを示す．

まず通常の機能情報抽出と同様に，回路図と付加情報を与え，簡単化された BDD を作成する．但し，付加情報として新たに次のものを考える．

(i) 入出力変数のうち，2 進数構造をとる部分を指定．
n ビットの算術演算回路の場合，通常入力変数として $C_0, A_0, \cdots, A_{n-1}, B_0, \cdots B_{n-1}$，出力として $F_0, \cdots, F_{n-1}, C_n$ が存在する．このうち，$(A_0, \cdots, A_{n-1})$ を A_0 を最下位ビットとする n 桁の 2 進数 A，$(B_0, \cdots, B_{n-1})$ を B_0 を最下位ビットとする n 桁の 2 進数 B，$(F_0, \cdots, F_{n-1}, C_n)$ を F_0 を最下位ビットとする $n+1$ 桁の 2 進数 F と考える．

(ii) データ系入力の順序付け．
2 節で，データ系入力の順序付けはコントロール系入力ほど問題にならないと述べたが，2 進数データの場合はその順序も重要な意味を持つ．通常，$A_0, B_0, A_1, B_1 \cdots, A_{n-1}, B_{n-1}$ の順にすると，比較的節点数の少ない BDD の最簡形が得られる．

(iii) 最下位ビットへの外部からの桁上げ入力の指定．
入力変数のうち，外部から最下位ビットへの桁上げ入力 C_0 に相当するものがあれば，それを指定する．なければ，桁上げ入力は定数 0 となる．C_0 は，データ系入力の中で最も順序を先とする．

BDD の最簡形が得られたところで，コントロール系入力の値によって幾つかの場合に BDD を分解する．各場合について得られた BDD の F_0 の形にマッチするものをデータベースに登録された各種演算の繰り返しパターンから候補として選ぶ．そして，与えられた回路の 2 進数データのビット数 n と選ばれた繰り返しパターン f, g から順次 $C_1, F_1, C_2, F_2, \cdots, F_{n-1}, C_n$ の BDD を作成し，FINES によって得られた $F_1, F_2, \cdots, F_{n-1}, C_n$ の BDD と比較する．全ての BDD が等しければ，この回路はその繰り返しパターンを持つと判定し，データベースに登録してある名前 (算術式) を用いて機能を表現する．パターンマッチングの途中で異なるところ

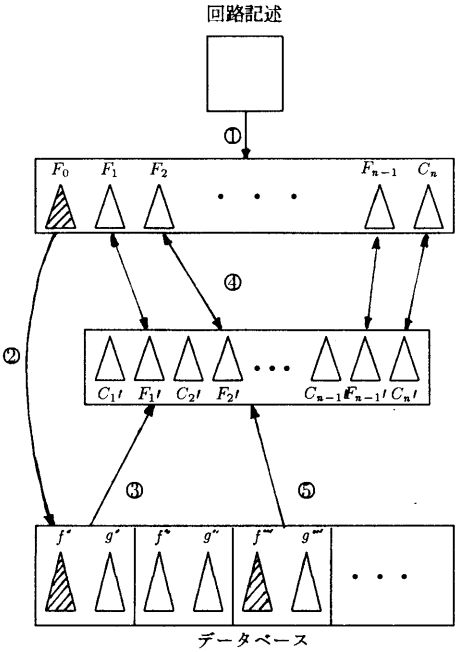


図 8　パターンマッチングの様子
① 与えられた回路から BDD を作成
② F_0 の形にマッチするパターンを候補として選び
③ $C_1', \cdots C_n'$ の BDD を順次作成
④ BDD を比較
⑤ 途中で異なるところが生じれば次の候補へ

が生じれば，次の候補に移る．候補が全てなくなれば，この回路の機能は未定義であると判定され，算術式を用いて表現することが出来ない．パターンマッチングの様子を図 8 に示す．

5　実例と問題点

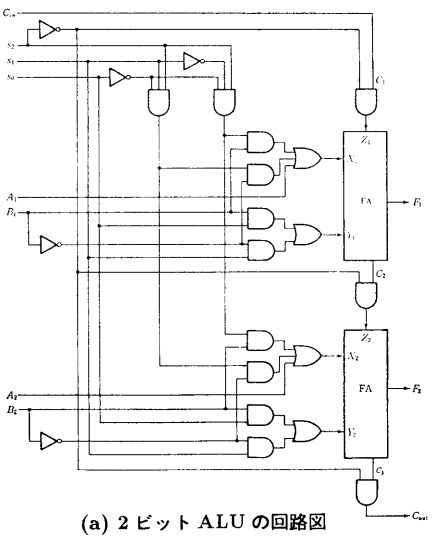
前節の手法に基づき，FINES を用いて算術演算回路の機能情報抽出を行った例を示す．

図 9 に示したのは，(a) 2 ビット ALU の回路図と (b) 抽出された機能表である．付加情報としては，入力変数の順序付け $(s_2 s_1 s_0 C_{in} A_1 B_1 A_2 B_2)$，コントロール系入力 $(s_2 s_1 s_0 C_{in})$ とデータ系入力 $(A_1 B_1 A_2 B_2)$ の区別，2 進数部分 $A = (A_1 A_2), B = (B_1 B_2), F = (F_1 F_2 C_{out})$ の指定及び桁上げ入力 C_{in} の指定をシステムに与えた．前節では，最下位ビットへの桁上げ入力は，機能の一部としてデータ系入力と見なしたが，ここではその値によって機能が変わるものとしてコントロール系入力と考えている．図 (b) に示したように，従来の機能表とは異なって，*plus* や *minus* などの算術演算記

号を用いて表現されているのが特徴である．FINES は，ソニー NWS830 (CPU68020, クロック 16.6MHz) 上の Franz Lisp インタープリタで動作するが，この例で機能表作成に要する CPU 時間は，約 3.7 秒であった．

次に図 10(a) に示した 4 ビット ALU(TTL 74181) についても同様に機能表を作成してみた．但し出力は，2 進数部分のみに注目している．この場合，同図 (b) に示すように，大半の算術演算が未定義であると見なされた．このように，登録されていない演算が未定義であると見なされるのが，本手法の大きな欠点である．しかし，未定義と見なされた演算であっても，図 7 に示した繰り返し性を持ったものについては，F_0 と F_1 の BDD の形だけからその繰り返し性を定める関数 f, g を推定することが出来る．こうして推定された f, g を用いて $C_1, F_2, \cdots, C_{n-1}, F_{n-1}, C_n$ の BDD を作成し，回路から抽出された BDD と比較すれば，この回路が繰り返しパターン f, g を持った回路であるか否かが判定出来る．この手法を図 10 の回路に適用すれば，同図 (c) のように繰り返しパターンが抽出される．この表は通常の機能表に比べれば非常に簡潔に表現されているので，登録されていない演算についてはこのように繰り返しパターンを抽出するということで欠点を補えるのではないかと考えられる．なお図 10(b) の機能表作成に要する CPU 時間は約 32.7 秒，図 (c) では約 38.5 秒であった．

回路の大規模化については，実行時間よりも使用メモリ量がネックとなり，例えば全加算器では 6 ビット (13 入力) までしか対応出来ていない．今後は本手法の基礎技術である BDD についての新しい優秀な技術 [7] を導入し，処理の高速化，使用メモリ量の削減を図っていく予定である．



(a) 2 ビット ALU の回路図

回路図は，コンピューターの論理設計 (共立出版) より引用

6 あとがき

本稿では算術演算回路の機能情報抽出の手法について説明し，一次元一方向繰り返し組合せ回路に対しては有効な手法を示した．今後はより一般的な回路の取り扱いが課題として挙げられる．また，同期式順序回路については，一次元一方向繰り返し組合せ回路で表現することができるので，今回述べた手法が順序回路にも適用出来るのではないかと考えられる．

参考文献

[1] J.Madre, *et.al.*, "Proving Circuit Correctness using Formal Comparison Between Expected and Extracted Behavior", 25th DAC, 1988.

[2] A.P.Kostelijk, "VERA, a Rule-Based Verification Assistant for VLSI Circuit Design", VLSI89, 1989.

[3] D.T.Blaauw, *et.al.*, "Automatic Generation of Behavioral Models from Switch-Level Descriptions", 26th DAC, 1989.

[4] 大村, 安浦, 田丸, "組合せ回路の機能情報抽出について", 情処研報, 88-DT-44, 1988.

[5] "Graph-Based Algorithms for Boolean Function Manipulation", IEEE Trans. Comput., C 35-8, 1986.

[6] 安浦寛人, "VLSI 用の正規集合認識法", 情報処理, 24-4, 1983.

[7] 湊, 石浦, 矢島, "共有二分決定図を用いた論理関数の処理手法について", 第 38 回情処全大, 5S-9, 1989.

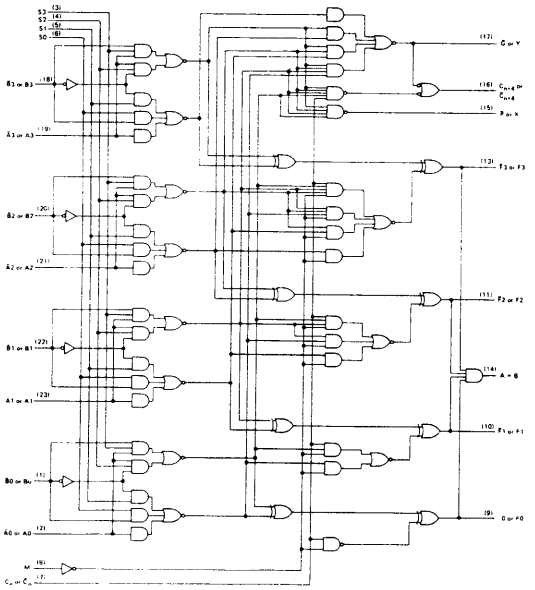
(s2 s1 s0 Cin)	F
(0 0 0 0)	A
(0 0 0 1)	A_plus_1
(0 0 1 0)	A_plus_B
(0 0 1 1)	A_plus_B_plus_1
(0 1 0 0)	A_minus_B_minus_1
(0 1 0 1)	A_minus_B
(0 1 1 0)	A_minus_1
(0 1 1 1)	A
(1 0 0 x)	A+B
(1 0 1 x)	A^B
(1 1 0 x)	A*B
(1 1 1 x)	~A

但し記号の意味は次の通り

plus	加算
minus	減算
+	論理和
^	排他的論理和
*	論理積
~	否定

(b) 抽出された機能表

図 9 算術演算機能抽出の例 (1)



(a) 4 ビット ALU の回路図

(M	S3	S2	S1	S0)	f	g
(0	0	0	0	0)	Ci^Ai	Ci*Ai
(0	0	0	0	1)	Ci^(Ai+Bi)	Ci*(Ai+Bi)
(0	0	0	1	0)	Ci^(Ai+~Bi)	Ci*(Ai+~Bi)
(0	0	0	1	1)	~Ci	Ci
(0	0	1	0	0)	Ci^Ai*Bi	Ci*Ai+~Ci*Ai*~Bi
(0	0	1	0	1)	Ci^Bi	Ci*(Ai+Bi)+~Ci*Ai*~Bi
(0	0	1	1	0)	Ci^Ai^~Bi	Ci*(Ai+~Bi)+~Ci*Ai*~Bi
(0	0	1	1	1)	Ci^(Bi+~Ai)	Ci+Ai*~Bi
(0	1	0	0	0)	Ci^Ai*~Bi	Ci*Ai+~Ci*Ai*Bi
(0	1	0	0	1)	Ci^Ai^Bi	Ci*(Ai+Bi)+~Ci*Ai*Bi
(0	1	0	1	0)	Ci^~Bi	Ci*(Ai+~Bi)+~Ci*Ai*Bi
(0	1	0	1	1)	Ci^(~Bi+~Ai)	Ci+Ai*Bi
(0	1	1	0	0)	Ci	Ai
(0	1	1	0	1)	Ci^~Ai*Bi	Ci*(Ai+Bi)+~Ci*Ai
(0	1	1	1	0)	Ci^~Ai*~Bi	Ci*(Ai+~Bi)+~Ci*Ai
(0	1	1	1	1)	Ci^~Ai	Ci+Ai
(1	0	0	0	0)	~Ai	--
(1	0	0	0	1)	~Ai*~Bi	--
(1	0	0	1	0)	~Ai*Bi	--
(1	0	0	1	1)	0	--
(1	0	1	0	0)	~Bi+~Ai	--
(1	0	1	0	1)	~Bi	--
(1	0	1	1	0)	Ai^Bi	--
(1	0	1	1	1)	Ai*~Bi	--
(1	1	0	0	0)	Bi+~Ai	--
(1	1	0	0	1)	Ai^~Bi	--
(1	1	0	1	0)	Bi	--
(1	1	0	1	1)	Ai*Bi	--
(1	1	1	0	0)	1	--
(1	1	1	0	1)	Ai+~Bi	--
(1	1	1	1	0)	Ai+Bi	--
(1	1	1	1	1)	Ai	--

(c) 抽出された繰り返しパターン

(M	S3	S2	S1	S0	CI)	F
(0	0	0	0	0	0)	A
(0	0	0	0	0	1)	A_plus_1
(0	0	0	0	1	0)	A+B
(0	0	0	0	1	1)	undefined_function
(0	0	0	1	0	0)	A+~B
(0	0	0	1	0	1)	undefined_function
(0	0	0	1	1	0)	1
(0	0	0	1	1	1)	0
(0	0	1	0	0	0)	undefined_function
(0	0	1	0	0	1)	undefined_function
(0	0	1	0	1	0)	undefined_function
(0	0	1	0	1	1)	undefined_function
(0	0	1	1	0	0)	A_minus_B_minus_1
(0	0	1	1	0	1)	A_minus_B
(0	0	1	1	1	0)	undefined_function
(0	0	1	1	1	1)	A*~B
(0	1	0	0	0	0)	undefined_function
(0	1	0	0	0	1)	undefined_function
(0	1	0	0	1	0)	A_plus_B
(0	1	0	0	1	1)	A_plus_B_plus_1
(0	1	0	1	0	0)	undefined_function
(0	1	0	1	0	1)	undefined_function
(0	1	0	1	1	0)	undefined_function
(0	1	0	1	1	1)	A*B
(0	1	1	0	0	0)	shift_1
(0	1	1	0	0	1)	undefined_function
(0	1	1	0	1	0)	undefined_function
(0	1	1	0	1	1)	undefined_function
(0	1	1	1	0	0)	undefined_function
(0	1	1	1	0	1)	undefined_function
(0	1	1	1	1	0)	A_minus_1
(0	1	1	1	1	1)	A
(1	0	0	0	0	x)	~A
(1	0	0	0	1	x)	~A*~B
(1	0	0	1	0	x)	~A*B
(1	0	0	1	1	x)	0
(1	0	1	0	0	x)	~A+~B
(1	0	1	0	1	x)	~B
(1	0	1	1	0	x)	A^B
(1	0	1	1	1	x)	A*~B
(1	1	0	0	0	x)	~A+B
(1	1	0	0	1	x)	~A^B
(1	1	0	1	0	x)	B
(1	1	0	1	1	x)	A*B
(1	1	1	0	0	x)	1
(1	1	1	0	1	x)	A+~B
(1	1	1	1	0	x)	A+B
(1	1	1	1	1	x)	A

(b) 抽出された機能表

図 10 算術演算機能抽出の例 (2)

回路図は, TTL IC 規格表 (CQ 出版社) より引用