設計自動化 44－3
（1988. 10. 28）

組合せ回路の機能情報抽出について

大村昌彦　安浦寛人　田丸啓吉

京都大学工学部

本報告では，論理回路図からその機能情報を抽出する手法を提案する．この手法は，高位レベルにおける設計検証に役立つばかりでなく，設計ドキュメントやマニュアルの自動生成にも利用できる．われわれは，組合せ回路の結線情報から機能情報を抽出するシステムを試作した．回路の入力変数に関する付加的な情報を与えることにより，質の良い機能表が効果的に作成できる．本報告では，情報抽出の基礎的概念を示し，試作したシステムの詳細について述べる．

ON FUNCTIONAL INFORMATION EXTRACTION OF COMBINATIONAL CIRCUITS

Masahiko OMURA, Hiroto YASUURA and Keikichi TAMARU

Department of Electronics, Faculty of Engineering, Kyoto University
Kyoto 606, JAPAN

We propose a technique to extract functional information from logic circuits. This technique is useful for design verification in higher level and automatic generation of design documents and manuals. We made a prototype system for extracting functional information from net-list information of combinational circuits. Using some additional information on input variables of a circuit, our system effectively generates feasible functional tables. This paper shows a basic concept of functional extraction and a detail of the prototype system.

1. まえがき

ディジタル回路の設計自動化においては，仕様記述，機能レベル，論理回路，電子回路，レイアウトなどの各設計過程間の自動設計（自動変換）の技術の確立が必要であり，多くの自動設計システムの研究開発がおこなわれてきた．また，設計対象の大規模化，複雑化に伴い，各過程での設計検証やドキュメントの作成などの自動化も非常に重要な問題となってきている[1]．

一般に自動設計は2つの設計レベル間の記述の自動的な変換と考えられる．従来，研究開発の主な対象とされてきたのは，論理回路からレイアウトを自動生成するように，いわば上位レベルから下位レベルへの変換であった．しかし，設計レベル間の記述の変換という立場からは，上位レベルから下位レベルへの変換の他に，同位レベル間の変換や下位レベルから上位レベルへの変換も考えられる．実際に，レイアウト検証のためのレイアウトから回路情報を抽出するシステム[2]や，テクノロジ変換のための論理回路変換[3]は，下位から上位あるいは同位のレベル間の変換の例となっている．

上位レベルの記述から下位レベルの記述への変換は，一般に抽象的な情報を具体化する操作に対応し，上位レベルで不足していた情報を付加していくことと考えられる．一方，下位レベルから上位レベルへの変換は，多くの情報の中から必要な情報だけを取り出すことを意味すると考えられる．即ち，上位レベルから下位レベルへの変換（設計）が情報の生成あるいは創造であるのに対し，下位レベルから上位レベルへの変換は情報の抽出であると考えられる．ある意味では，創造よりも抽出の方がより自動化しやすいと考えられる．

このような下位レベルの記述から上位レベルの情報を抽出する手法は，設計検証の有力な手段となる．設計検証を設計されたもののレベル，即ちより多くの情報を含む下位レベルにおいて行うと，多大の手間を要する．そこで，より情報の少ない上位レベルにおいて検証を行うことが得策であると考えられる[4]．また，上位情報の抽出は，設計のドキュメントやマニュアルの作成にも利用できる．実際，設計の各過程におけるドキュメントの作成は多大な人手と時間を要する仕事であり，この作業の自動化は今後ますます重要になると考えられる．

以上のような観点に基づいて，本報告ではディジタル回路設計過程の高位レベルにおいて下位レベルの情報から上位レベルの情報を抽出する手法について考察する．具体的には，論理回路レベルの情報から機能レベルの情報を抽出することに問題を限定し，組合せ回路の回路情報からその回路の機能情報を抽出するシステムを試作し，機能情報の抽出手法に関する基本的な考察を行う．

2. 組合せ回路の機能情報抽出

情報抽出の手法を述べるに先立って，ここでは問題の定式化を行う．まず情報を抽出される側，即ち論理回路レベルの情報の表現法を考える．組合せ回路の持つ情報としては，その回路を構成する各素子（論理ゲート）の種類，外部端子の種類，及びそれらの接続関係（結線情報），あるいはある入力パターンに対して出力がどのようになるか（機能情報），入力信号が出力に到達するまでにどれほどの時間がかかるか（遅延情報），など種々のものが考えられる．これらの情報のうち，論理ゲートや外部端子の種類，結線情報などは，論理回路レベルで直接的に表現され得る情報である．このような論理回路レベルの情報を表現する手段としては，論理回路図やハードウェア記述言語などが用いられている[1]．

機能情報や遅延情報，あるいは回路の使用法などの機能レベル，仕様記述レベルの情報は，論理回路図や対応する言語記述には陽に表現されてはいない．これらの情報は，普通はシミュレーション等により，論理回路レベルの記述から抽出されるものである．機能情報や遅延情報を設計された回路の仕様と比較することで，設計検証がおこなえる．また，論理回路レベルの記述から回路の使用法などの仕様記述レベルの情報が抽出できれば，ドキュメントやマニュアルの作成などにおいて大きな手助けとなる．ここでは，最も基本的な機能レベルの情報抽出を対象にして，情報抽出の基礎的な技術の確立を目指す．

次に抽出された情報の表現法について考える．遅延情報は入力線名，出力線名，及びその入力から出力まで信号が伝搬するのに要する時間を並べて，表にするなり言語で記述するなりして表すことができる．一方，どのような入力に対して出力がどのようになるかという論理関数としての機能情報の表現法としては，論理式，真理値表，グラフ表現[5]，ハードウェア記述言語などが知られている．

最も一般的に用いられているのは論理式である．この表現は，入力信号の数が少なくしたがって短い式で表されている場合は簡潔で見やすいが，入力信号の数が多くなってくると複雑になり，冗長な変数を含んでいても見逃すことがある．また一意性をもたないため，二つの論理式が同一の論理関数を表すか否かの判定すらNP完全な問題となってしまう．

真理値表やグラフによる表現は，全ての入力パターンに対して出力が0になるか1になるかを，それぞれ表，グラフの形式で示したものである．真理値表による表現は，入力信号の数nに対して2^nの大きさになるため入力数が多い場合には非実用的である．しかしある入力の値によらず出力が同じになるようないくつかの入力パターンがあれば，それらをまとめて簡単化し，表のサイズを小さくすることができる．このように簡単化された真理値表では，出力の欄に0，1の値だけではなく簡単な論理式も含まれるため，表のサイズが小さくなるばかりでなく簡潔でわかりやすい表現となっている．したがってICの規格表などではこの形式がよく用いられている[6]．この簡単化された真理値表を機能表（Function Table）と呼ぶことにする．

グラフによる表現は，真理値表を変形したものと考えられる[5][7]．グラフの各節点に入力変数名を与え，その節点から出る2本の枝はそれぞれその入力が0，1の場合に対応し，葉は0，または1の値をとる．このグラフによる表現も簡単化することができ，しかも簡単化されたグラフは，変数の順序を固定すれば一意性をもつことが知られている[5]．したがって，設計検証や論理関数の等価性を調べる問題においてはこの表現が適していると思われる．しかしこのグラフ表現がどういう機能を表しているかということを知るためには，グラフの根から葉まで各節点がどのような値をとっているか調べなければならず，人間にとってはあまりわかりやすい表現とは言えない．

ハードウェア記述言語では，論理式やプログラム言語風の条件文などを組み合わせて機能情報を表すことができるが，基本的には論理式あるいは機能表を書き換えたものとみなせる．

本稿で述べる情報抽出は，論理回路図などの論理回路レベルの記述から，上で述べたような機能レベルの情報を得ることである．この際，論理回路レベルでは表せないが機能情報を抽出するための手助けとなる情報があれば，それをユーザが与えてもよいこととする．この様な情報を付加情報と呼ぶ．このことについては後に詳しく説明するが，実はこの付加情報の有無，あるいはその良し悪しが情報抽出の質を大きく左右する．ここではとりあえず，情報抽出とは論理回路レベルの記述に付加情報を与えて機能レベルの情報を得ることとしておく．

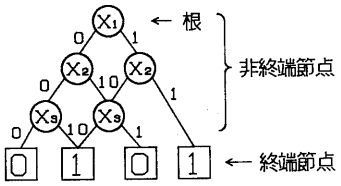


図3．2　グラフ表現の例

3. 情報抽出の手法

3.1. 概要

ここでは，組合せ回路の機能情報抽出の手法について述べる．上で述べたように，抽出された情報は機能表で表されるのが人間にとって最もわかりやすく，また簡単化したグラフ表現を用いればその関数が一意に表現できる．そこで，図3.1に示すように，与えられた論理回路図をいったんグラフ表現に変換し，それを簡単化した後，グラフから機能表を作成する，という手法をとる．これらの過程において，付加情報が利用される．

3.2. グラフ表現とその簡単化

3.2.1. 論理関数のグラフ表現[5)]

まず，本手法の基本となる論理関数のグラフ表現について説明する．n入力1出力論理関数fを考えその入力変数を

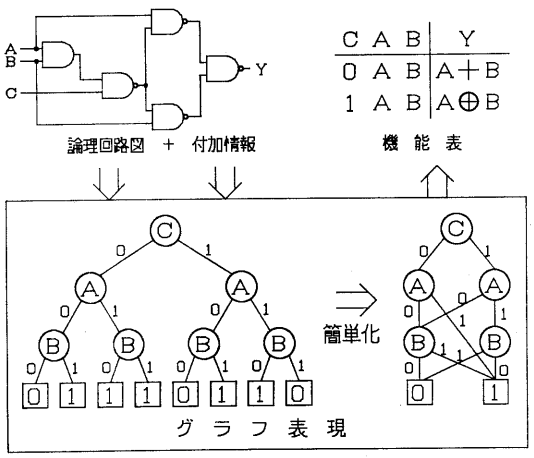


図3．1　機能情報抽出の概要

$x_1, \ldots, x_n$ とする．論理関数fを表すグラフGは，図3.2に示すように2種の節点を含む節点集合Vからなる閉路を持たない有向グラフで，次の条件を満たすものである．節点集合Vは非終端節点集合Nと終端節点集合Tからなり，$V = N \cup T$, $N \cap T = \phi$ である．入次数が0の節点が唯一存在し，これを根と呼ぶ．根はNに含まれる場合もTに含まれる場合もある．$v \in T$は出次数0で $value(v) \in \{0,1\}$ なる値を属性として持ち，図3.2では四角の節点で表されている．一方$v \in N$は出次数2で，$index(v) \in \{1, ..., n\}$ なる変数番号と2つの子供を示すポインタ low(v)，high(v) $\in$ Vを属性として持ち，枝 (v,low(v))，(v,high(v)) がその節点からの出力枝となる．非終端節点は図3.2では丸い節点で表されている．$index(v) = i$ ということは節点vが変数 x_i に対応していることを意味する．また子供 low(v)，high(v) は，x_i が0及び1の値をとる場合に指す節点を示す．図3.2に示したように，各非終端節点vからは0及び1なる数字が添えられた2本の枝が出ており，それぞれ low(v)，high(v) につながっている．$low(v) \in N$ の時は $index(v) < index(low(v))$でなければならない．同様に $high(v) \in N$ の時は $index(v) < index(high(v))$でなければならない．$v \in T$ に対しても $index(v) = n+1$ と定義してやればこれらのことは任意の節点で成り立つ．

次に，vを根とするグラフの表す関数 f_v を求める方法について述べる．まずvが終端節点の時は，$value(v) = 1$ なら $f_v = 1$ なる恒真関数，$value(v) = 0$ なら $f_v = 0$ なる恒偽関数を表す．またvが非終端節点の時には，$index(v) = i$ とおくと，$f_v = \overline{x_i} \cdot f_{low(v)} + x_i \cdot f_{high(v)}$ で再帰的に求められる．ここで $f_{low(v)}$，$f_{high(v)}$ は，それぞれ low(v)，high(v) を根とするグラフが表す関数であり，このようなグラフはもとのグラフのサブグラフと呼ばれる．サブグラフはあるグラフにおいて任意の節点で定義される．

3.2.2. グラフ表現の簡単化

一般に回路記述から作成されるグラフは，2^n 個の終端節点及び $2^n - 1$ 個の非終端節点を持つ．このグラフは以下のアルゴリズムによって簡単化され，そのサイズを小さくして表現することができる[5)]．

まずグラフの同形ということについて定義する．2つのグラフG,G′ において，Gの節点集合Vから G′ の節点集合 V′ の上への1対1写像 σ を考える．2つの節点v ∈ V，v′ ∈ V′ に対して

$\sigma(v) = v'$ なら

Vも V′ も value(v) = value(v′) なる終端節点

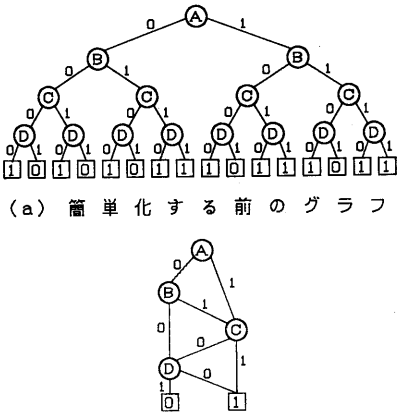
または index(v) = index(v′)，$\sigma(low(v)) = low(v')$，

$\sigma(high(v)) = high(v')$ なる非終端節点

が成り立つとき，この2つのグラフG,G′ は同形であるという．あるグラフにおいて low(v) = high(v) なる節点vが存在せず，かつ同形である2つ以上のサブグラフが含まれなければ，そのグラフは簡単化されているという．例えば図3.3(a)のグラフを簡単化すると同図(b)のようになる．グラフの簡単化については次の重要な定理が証明されている．即ち，『任意の論理関数fに対し，f を表す簡単化されたグラフは唯一存在し，それ以外のfを表すグラフはそれより多くの節点を含む』，[5]というものである（証明 略）．この定理から，簡単化されていないグラフは節点数を減らしてサイズを小さくすることができる．

ここで注意しておかねばならないのは，図3.4に示すように同じ関数であっても変数の順序が異なればそのグラフ表現が異なってくる，ということである．それは単に各節点に対応する変数の名前が変わるだけでなく，グラフの簡単化の度合が異なってくる場合もあることを意味する．したがってこの定理は変数の順序が一定である時にのみ成り立つものである．

簡単化を行う手順は，同形であるサブグラフは1つだけを残して取り除き，low(v) = high(v) なる節点は省略して，vを指す全てのポインタが low(v) を指すようにする．この操作を行ったことによって新たにそのようなサブグラフや節点が生じた場合には，再びこの操作を行い，簡単化されるまで繰り返す．



（a） 簡 単 化 す る 前 の グ ラ フ

（b） 簡 単 化 さ れ た グ ラ フ

図3．3 グラフの簡単化の例

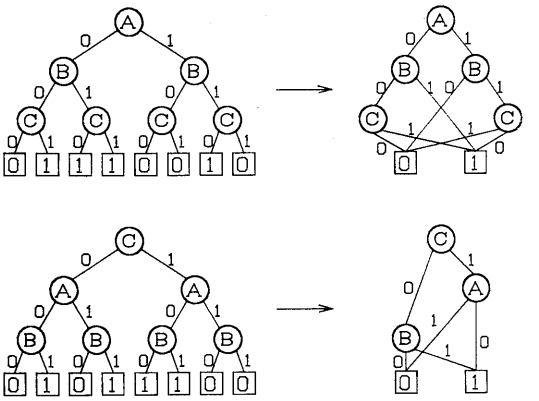


図3．4 変数の順序によって簡単化されたグラフのサイズが異なる例

3.3. 機能表作成の手法

3.3.1. 付加情報について

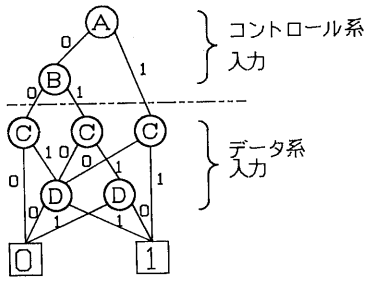
機能表は簡単化されたグラフから作成することができるが，一般に節点数の少ないグラフからはわかりやすい機能表が得られるのではないかと考えられる．そこで，簡単化されたグラフのサイズ（節点数）が最も小さくなるような入力変数の順序があらかじめわかっていれば，それを付加情報として与えることが考えられる．しかし，一般にそのような順序を知ることは非常に難しく，ユーザに大きな負担がかかる．これを少しでも軽減するために，他の付加情報を考える．

図3.1の機能表は，入力Cが0か1かによって，出力にAとBとの論理和または排他的論理和が現れることを示している．これより，入力Cは回路の出力を制御する働きをしていると考えられる．そこで，このような入力をコントロール系入力と呼ぶことにする．一方，入力A，Bは，入力Cによって制御されるデータであるとみなされるので，これらをデータ系入力と呼ぶ．このように，回路の入力をコントロール系のものとデータ系のものに分類し，グラフ作成の際に，入力をコントロール系，データ系の順に並べれば，グラフを簡単化したときに節点数が比較的少なくなり，わかりやすい機能表が得られる．また，データ系入力については，それらの間の順序はそれほど重要ではない（但し，数値データのようにデータ系入力が構造をもつ場合は，その順序が重要となる．）．したがって，ここで与えるべき付加情報というのは，コントロール系入力とデータ系入力の区別，及びコントロール系入力が複数個ある場合には，それらの間の順序，ということになる．コントロール系入力とデータ系入力の区別は，回路図から容易に読み取れる場合が多く，また，ユーザ自身がある程度これに関する情報を持っている場合も多いと考えられるので，ユーザの負担はかなり軽減できると思われる．

3.3.2. 機能表への変換

次に簡単化されたグラフ表現を機能表の形式に直してやることを考える．ここで得られる機能表は，入力の部分の幾つかが入力変数名などの記号で与えられ，出力がそれらの論理式，あるいは0，1の値で表されたものである．言い換えれば，このグラフの表す論理関数において，幾つかの変数を0か1に置き換えて出力を簡単な論理式で表しそれらを並べたもの，ということになる．

グラフ表現において，ある変数を0（または1）に置き換えるということは，index(v) = i なる節点vを指すポインタを全て low(v)（または high(v)）を指すように変えてやることを意味する．それによって新たに，より節点数の少ないグラフが得られる．このグラフの表す論理関数を3.2.1.で述べた手法に従って求めると，$x_i = 0$（または $x_i = 1$）の時の論理式が得られることになる．特に，根vに対応する変数 x_1 を0（または1）に置き換えると，low(v)（または high(v)）を根とするサブグラフが得られる．一般に x_1 は，先に述べたコントロール系入力であるから，機能表では x_1 は0と1に分けられるべきである．また，low(v)（あるいは high(v)）に対応する変数がコントロール系入力であれば，これも0と1に分けて考える．そして全てのコントロール系入力を0と1に分解して得られるいくつかのサブグラフを，3.2.1.で述べた手法にしたがって論理式に変換すると，機能表が得られる．簡単化されたグラフでは図3.5に示すように，ある場合に0でも1でも出力に影響しないコントロール入力は冗長な節点として削除されているから，この入力は0と1の場合に分けられず，機能表には入力変数名がそのまま現れて，わかりやすい表現となっている．



A	B	C	D	Y
0	0	C	D	C・D
0	1	C	D	C⊕D
1	B	C	D	C＋D

機 能 表

図3．5 機能表への変換

4. 機能情報抽出システムの試作

4.1. システムの概要

3．で述べた手法に基づいて機能情報抽出システムを試作した．対象とする回路は組合せ回路のみである．まずこのシステムの概要について述べる．

システムの構成は，本体，記号シミュレータ，† 及び回路図入力用の図面作成ツールからなる（図4.1参照）．今回試作したのは本体のみで，他の2つは既成のものを利用した．記号シミュレータは，与えられた回路記述に対して各入力に0，1，または入力変数名などの記号を与えてやると，出力がそれらの論理式，あるいは0，1の値で表されるものである．図面作成ツールは市販のスケマチックデザインツール††を利用した．これは論理回路図の図面を作成する設計ツールで，作成された図面からEDIF形式のネットリストを出力することができる．

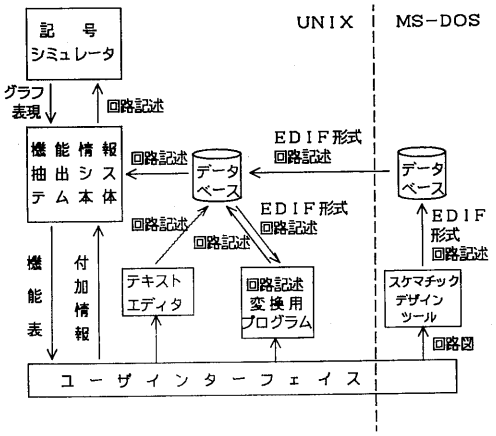


図4．1 システムの構成図

† 京都大学工学部情報工学教室矢島研究室で開発されたものである．
†† OrCAD SYSTEMS CORPORATION のOrCAD/SDT™を用いた．

システム本体への入力は，回路記述と付加情報であり，出力は機能表である．回路記述の作成には2通りある．ひとつは，テキストエディタを用いて直接回路記述を作成してやる方法である．もうひとつは図面作成ツールを用いて回路図を入力し，それから得られるEDIF形式のネットリストを回路記述に変換する方法である．この変換用プログラムは，本体に含まれている．また付加情報は，プログラムを実行させてから適宜入力する．

次にシステムの動作環境などについて説明する．図面作成ツールは MS-DOS 上で動作するツールであり，それ以外のものは UNIX(CPU 68020) 上で動作する．したがってこのシステムを利用するときには，まず MS-DOS 上で作成したEDIF形式のネットリストを UNIX 上に移し，システムの入力回路記述への変換を行っておく．本体及び記号シミュレータのプログラムは，記号処理が中心となるので LISP(Franz Lisp)で実現した．

システムの動作原理は，まず回路記述を記号シミュレータに与えて回路の出力を入力の論理式で表す．次にこれをグラフ表現に変えて簡単化し，機能表に直す．この機能表作成の際に，データ系入力の指定という付加情報を与える．即ち，入力変数のうちの幾つかをデータ系入力であると指定することによって，機能表の出力欄の論理式にデータ系入力以外の変数が含まれないようにする．またデータ系入力は，入力の順序としては最後にくるものとする．したがって節点数が最小となる入力変数の順序を見つけるには，データ系入力以外の入力について全ての順序を調べてやればよい．

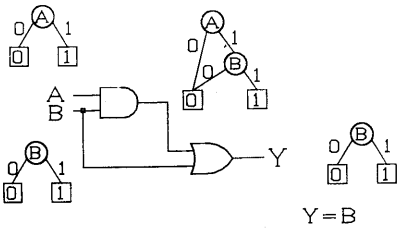


図4．2　記号シミュレータの使用例

```
(IN A B C)                ←入力変数名のリスト
(OUT Y)                   ←出力変数名のリスト
(G1 AND2 0 A B D)
(G2 NAND2 0 D C E)        論理ゲートに
(G3 NAND2 0 A E F)        関する情報
(G4 NAND2 0 B E G)
(G5 NAND2 0 F G Y)
```

図4．3　記号シミュレータの入力回路記述

4.2. 記号シミュレータの利用

ここで用いた記号シミュレータの使用例を図4.2に示す[8]．これは入力変数などをすべてグラフで表現し，途中の演算もすべてグラフに対して行う．そして演算を行うたびにグラフの簡単化を行い，出力にはその論理関数を表す簡単化されたグラフが得られる．そしてこのグラフ表現を論理式表現に変換する命令も用意されている．そこで，この命令を用いる代わりに機能表に変換することを考えた．それは 3.3.で述べたようにコントロール系入力を0，1に分解してゆき，得られたサブグラフを論理式に変換するという手法である．この記号シミュレータはグラフ表現の簡単化を有効に利用しており，不必要にグラフが大きくなることがなく，実行時間や要するメモリなどの点で優れている．

記号シミュレータには回路記述と入力変数の値を与えてやらねばならない．回路記述は図4.3に示すようなもので，入力変数のリスト，出力変数のリスト，及び各論理ゲートに関するリストからなる．論理ゲートに関するリストは左から順に，ゲートの名前，種類，遅延時間，入力線名，・・・，出力線名となっている．ここで考えているのは組合せ回路の機能情報抽出であるから，遅延時間は0にしておけばよい．記号シミュレータは，まずこの回路をLISPのプログラムに変換する．次に，各入力変数にその変数名を値として与えてやると，簡単化されたグラフを出力する．記号シミュレータの仕事はここまでである．次に3.3.2.で述べた手法に基づいて，グラフを機能表に変換する．

4.3. 機能表の作成例

このシステムを用いて実際に機能表を作成した例を示す．図4.4は4ビットのセレクタである．図(b)は，図(a)の回路図から得られるEDIF形式のネットリストを変換したもので，このシステムの入力である．また，入力A，B，C，Dがデータ系入力であるという付加情報も与える．これだけの入力を与えると，このシステムはまず，コントロール系入力EN，S1，S2の順序を定めるため，この3入力の全ての順序（3! = 6 通り）についてグラフを作成し，簡単化を行う．すると，(EN S2 S1) と (EN S1 S2) が簡単化されたグラフの節点数を最小にする順序であることがわかる．そこで，入力の順序を (EN S2 S1 A B C D) として機能表を作成すると，図(c)のようになる（入力順序を (EN S1 S2 A B C D) としても，ほぼ同様の機能表が得られる．）．この機能表は，入力ENが0の時は入力S2，S1の値に応じてデータ系入力のうちの一つが出力にあらわれ，入力ENが1の時は他の入力によらず出力が0になることを示しており，イネーブル入力つきのセレクタの機能がうまく抽出されていると思われる．

図4.5(a)は4ビットのプライオリティエンコーダ，同図(b)は10進デコーダであるが，これらのような多出力関数においても，このシステムはわかりやすい機能表を作成する．

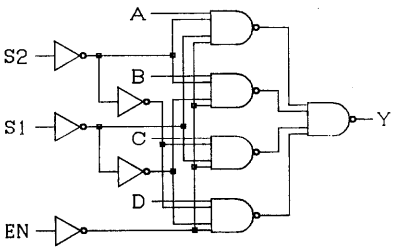
なお，これらの機能表作成に要するCPU時間は，ゲート数20程度の回路で0.1～0.3秒程であり，それほど膨大な時間ではない．しかし，現在使用しているLISPの処理系の限界で，このシステムは10入力50ゲート程度の回路までしか適用できない．

4.4. 問題点

このシステムの問題点について検討する．まず，付加情報がない場合にどうするかという問題である．基本的には，全ての入力順序についてグラフを作成し，簡単化して節点数が最小となるものを選んで機能表に変換すればよいが，それでは非常に時間とメモリ容量を要するし，わかりやすい機能表が得られるとも限らない．そこで論理回路図から自動的に付加情報を得ることが考えられる．例えば，入力から出力までの段数によって入力順序を決めるとか，ファンアウト数の多い入力はコントロール系入力であるとみなす，とかである．また，回路図における位置情報や対称性なども付加情報を得るための有力な手がかりとなりそうであるが，それらをどのように利用して付加情報を得るかという具体的な検討は，今後の課題である．

次に，算術演算機能をどのように表現するかという問題が挙げられる．即ち，このシステムを用いると，単純な論理機能は抽出されるが，算術演算の様な高度な機能は抽出されない．これは，論理機能として抽出された情報を如何に算術機能としてみなすかということで，ここでおこなったような回路図からの単純な機能抽出とは別次元の問題とも考えられる．特に，数値データの符号化に関する情報をシステムに与える必要があると考えられ，その表現法なども問題となる．

最後に，順序回路をどのように表現するかということが挙げられる．順序回路はある入力に対して一意に出力が決まらず，状態を考えねばならない．また，クロック入力といった新しい種類の入力も考慮せねばならず，組合せ回路のようにはいかない．しかし，順序回路も組合せ回路と記憶部分に分けて考えることができるので，今回述べた手法を発展させて，順序回路の機能情報を抽出することもできると考えている．



（a） セレクタの論理回路図

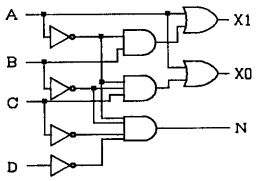
```
(IN EN D C S1 B S2 A)
(OUT Y)
(U4A INV 0 N4 N6)
(U1A INV 0 S1 N9)
(U3A INV 0 N9 N13)
(U0A INV 0 EN N17)
(U2A INV 0 S2 N4)
(U5A NAND4 0 A N4 N9 N17 N2)
(U6A NAND4 0 B N4 N13 N17 N7)
(U7A NAND4 0 C N6 N9 N17 N11)
(U8A NAND4 0 D N6 N13 N17 N18)
(U9A NAND4 0 N2 N7 N11 N18 Y)
```

（b） 回路記述

```
(EN S2 S1 A B C D) (Y)
(0 0 0 A B C D) (A)
(0 0 1 A B C D) (B)
(0 1 0 A B C D) (C)
(0 1 1 A B C D) (D)
(1 S2 S1 A B C D) (0)
```

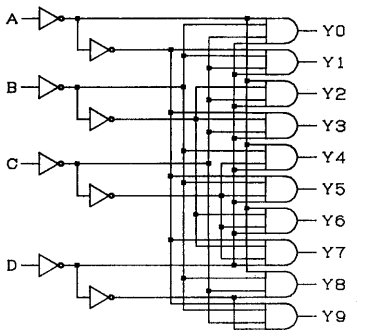
（c） 機能表

図4．4 機能表作成の例（1）




```
(D C B A) (N X0 X1)
(0 0 0 0) (1 0 0)
(0 0 0 1) (0 0 0)
(0 0 1 A) (0 1 0)
(0 1 B A) (0 0 1)
(1 C B A) (0 1 1)
```

（a） プライオリティ・エンコーダ




```
(D C B A) (Y9 Y8 Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0)
(0 0 0 0) (0 0 0 0 0 0 0 0 0 1)
(0 0 0 1) (0 0 0 0 0 0 0 0 1 0)
(0 0 1 0) (0 0 0 0 0 0 0 1 0 0)
(0 0 1 1) (0 0 0 0 0 0 1 0 0 0)
(0 1 0 0) (0 0 0 0 0 1 0 0 0 0)
(0 1 0 1) (0 0 0 0 1 0 0 0 0 0)
(0 1 1 0) (0 0 0 1 0 0 0 0 0 0)
(0 1 1 1) (0 0 1 0 0 0 0 0 0 0)
(1 0 0 0) (0 1 0 0 0 0 0 0 0 0)
(1 0 0 1) (1 0 0 0 0 0 0 0 0 0)
(1 0 1 A) (0 0 0 0 0 0 0 0 0 0)
(1 1 B A) (0 0 0 0 0 0 0 0 0 0)
```

（b） 10進デコーダ

図4．5 機能表作成の例（2）

5. あとがき

本稿では，ディジタル回路設計における下位レベルの記述を，上位レベルの記述に変換するということを提案し，組合せ回路図から機能表として表現された機能情報を抽出する手法について述べた．また，その手法に基づいて試作したシステムを用い，わずかな付加情報を与えるだけでかなり質のよい機能表が得られる例をいくつか示した．これからの課題としては，加算乗算などの算術演算機能をどのように表現するかということや，順序回路への対応などが考えられる．

6. 謝辞

本稿作成にあたり，いろいろと御討論頂いた本学田丸研究室の皆様に感謝致します．また，記号シミュレータのプログラムを提供して頂いた本学情報工学教室矢島研究室の皆様に感謝致します．

[参考文献]

1) 渡辺，浅田，可児，大附：ＶＬＳＩの設計Ｉ，岩波書店(1985)
2) K.Yoshida : Layout Verification, T.Ohtsuki Ed. Layout Design and Verification, North Holand (1986), pp.237-265
3) E.E.Hollis : Design of VLSI Gate Array ICs, Prentice-Hall(1987), p.173
4) 丸山，上原：方式・機能・論理設計の検証，情報処理，25-6(1984), pp.1062-1070
5) R.E.Bryant : Graph-Based Algorithms for Boolean Function Manipulation, IEEE Tran. on Comput., C.35-8(1986), pp.677-691
6) TTL IC 規格表，CQ出版社(1986)
7) S.B.Akers : Binary Decision Diagrams, IEEE Tran. on Comput., C.27-6(1978), pp.509-516
8) 北嶋，高木，矢島：論理関数のグラフ表現を用いた記号シミュレーション，信学技報，VLD87-113(1987)，pp.47-52