

3X-3

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

1 はじめに

ディジタル回路の自動設計においては、上位レベルから下位レベルへの設計記述の自動変換だけではなく、下位レベルから上位レベルへの記述の変換も考えられる。この技術は、設計検証やドキュメントの作成などに利用でき、今後ますます重要になると考えられる。

そこで我々は、論理回路レベルから機能レベルへの設計記述変換(機能情報抽出)に着目し、その基礎的手法並びにそれを応用した技術の研究を行ってきた[1][2]。そして、対象を組合せ回路に限定して、機能情報抽出システム FINES(Functional Information Extraction System)を試作した[1]。これは、論理回路レベルの構造記述(ネットリスト)から、機能表を自動作成するというものであったが、その後の改良により、機能レベルの記述言語も作成できるようになった[2]。しかし、結局回路の機能そのものは論理式で表現されていたため、*AplusB*のような算術式がわかりやすく表現できないという欠点があった。

本稿では、このような算術演算をうまく抽出してわかりやすく表現する方法について検討する。

2 組合せ回路の機能情報抽出システム FINES

FINES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(機能表または記述言語)に自動変換するものであるが、内部のデータ構造として、二分決定図(Binary Decision Diagram: 以下BDD)[3]を用いている。BDDは、回路の入力変数を順序付けして、その節点に対応させている。機能記述は、簡単化されてサイズの小さくなったBDDから作成されるが、そのサイズをできるだけ小さくするためには、入力変数に適当な順序を与えてやらねばならない。一般にこの順序を知ることは非常に難しいが、コントロール系入力(制御系入力)の順序を先に、データ系入力(データ系入力)の順序を後にし、更にコントロール系入力の中で、制御力の強い順に順序を付ければ、簡単化されたグラフのサイズが比較的小さくなることが経験的にわかっている。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、設計者自身が機能情報抽出の際に付加情報としてシステムに与える。

3 算術演算機能の表現

3.1 論理式による表現

算術演算機能の表現について、フルアダーを例にとって考えてみる。FINESを用いて2ビットフルアダー(入力 CI, A_0, A_1, B_0, B_1 、出力 F_0, F_1, CO)の機能表を作成すると、

CI	CO
0	$A_1 \cdot (B_1 + A_0 \cdot B_0) + \overline{A_1} \cdot B_1 \cdot A_0 \cdot B_0$
1	$A_1 \cdot (B_1 + A_0 + B_0) + \overline{A_1} \cdot B_1 \cdot (A_0 + B_0)$
F_1	
F_0	
$A_1 \oplus B_1 \oplus A_0 \cdot B_0$	
$A_0 \oplus B_0$	
$A_1 \oplus B_1 \oplus (A_0 + B_0)$	
$A_0 \oplus B_0$	

のようになり、これはあまりわかりやすい表現とは言えない。これより、この回路の機能を論理式で表わすには無理があるということがわかる。

3.2 BDDによる表現

FINESでは、内部のデータ構造として、BDDを用いている。これを用いると、上記の2ビットフルアダーの機能は図1のように表現できる。

この2ビットフルアダーが2つの1ビットフルアダーを直列に接続したものであると考えると、その間を伝搬する桁上げ信号が存在する。これを C_1 とおくと、図1のBDDは図2のようなになる。但し、 CI を C_0 、 CO を C_2 とおき直した。

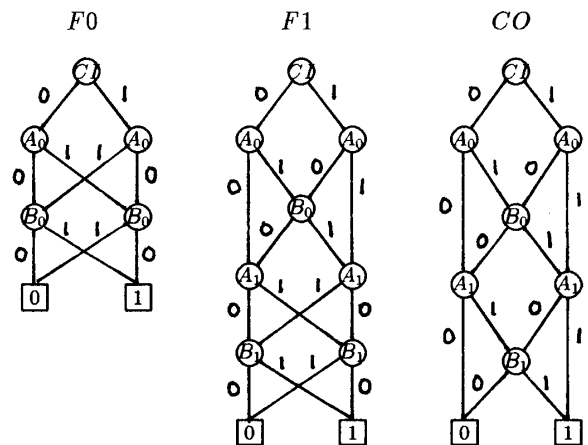
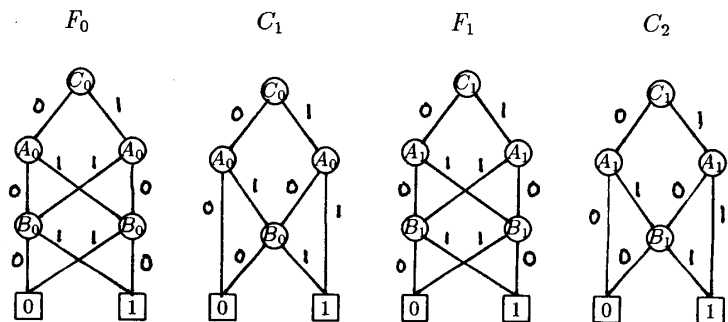


図1: 2ビットフルアダーのBDD表現

図2: 中間変数 C_1 を用いた2ビットフルアダーのBDD表現

Extraction of Functional Informations from
Arithmetic Units

Masahiko OHMURA, Hiroto YASUURA,
and Keikichi TAMARU
Kyoto University

この図を見ればわかるように、 F_0 と F_1 及び C_1 と C_2 のBDDは、それぞれ全く同じ形をしている。これは一般に n ビットのフルアダーに拡張することができる。このように、桁上げ信号を中間変数として導入することにより、加算の機能は図2に示すような簡単な構造のBDDで表現されることがわかる。逆に言うと、ある回路の機能をBDDで表わした時に図2のようになったとすると、その回路はフルアダーであると判断できる。

ここで重要なことは、論理式の代わりにBDDを用いたことにより、表現の唯一性が保証されるということである。即ち、入力変数の順序を一定に定めておけば、ある論理関数を表わすBDDは、唯一の形に簡単化することができる[3]、というものである。 $FINES$ を用いれば、回路の各出力 $F_0, F_1, \dots, F_{n-1}, C_n$ に対して簡単化されたBDDが得られるから、図2のようなBDDがフルアダーであるということをおそらく登録しておけば、これらと比較することによって、与えられた回路がフルアダーか否か判断できる。

4 算術演算機能抽出の手法

4.1 演算の型

前節で述べたフルアダーのように、中間変数を用いることによって、 n ビットの演算が1ビットの演算の繰り返し構造で表わされるものは、1ビット分のBDDをシステムに登録しておけば、 n ビットのBDDが即座に作成できる。このような演算のBDDをいくつか登録しておき、機能情報抽出で作成されたBDDとパターンマッチングをとることにより、算術演算機能の抽出ができると考えられる。本システムでは、このような型の演算として $AplusB$ などの12種の算術演算及び16種の論理演算を登録した。

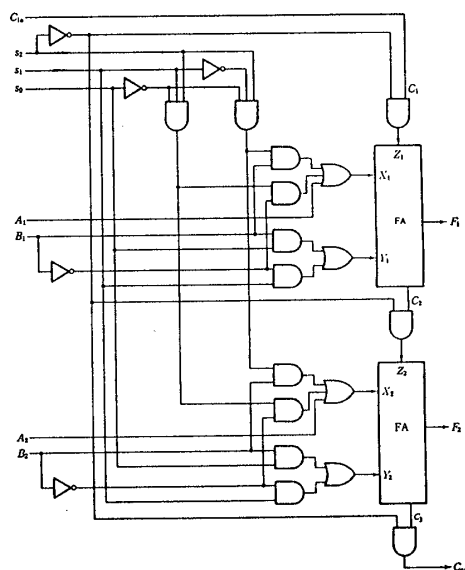
なお、このような型の演算は正規表現で表わすことができる[4]。同期式順序回路もやはり正規表現で表わせるので、同様の手法によって順序回路の機能情報抽出を行える可能性もある。また、乗算は正規表現では表わせないの、本手法で取り扱うのは難しいと考えられる。

4.2 算術演算機能抽出の手法

ここでは、算術演算機能抽出の手法について説明する。まず、与えられた回路から $FINES$ を用いてBDDを作成し、登録された28種の演算のBDDとパターンマッチングをとる。この時、ここで定義した演算は全て、 F_0 が A_0 と B_0 のみの論理式からなっているということに着目し、まず F_0 の形から28種のうちの幾つかに候補を絞る。次に、各演算固有の規則にしたがって、中間変数 C_i を導入し、次々と F_i を求めてゆく。こうして、求められた $F_0, F_1, \dots, F_{n-1}, C_n$ のBDDが機能情報抽出の結果作成されたBDDと等しければ、その機能を持つものとみなす。もし途中で異なる場所が生じれば次の候補に移り、全ての候補がなくなれば未定義関数であるとみなす。

5 実例

算術演算機能抽出の実例を示す。図3は2ビットのALUであり、3個のセレクト入力と1個の桁上げ入力により、8つの算術演算及び4つの論理演算が実現される。付加情報としては、入力変数の順序とデータ系入力の指定だけでなく、入出力変数のうち2進数データと見る部分も指定しなければならない。これらの付加情報は、回路設計者にとっては回路図から容易に得られるものであり、それほど大きな負担ではない。



s_2	s_1	s_0	C_{in}	F
0	0	0	0	A
0	0	0	1	$Aplus1$
0	0	1	0	$AplusB$
0	0	1	1	$AplusBplus1$
0	1	0	0	$AminusBminus1$
0	1	0	1	$AminusB$
0	1	1	0	$Aminus1$
0	1	1	1	A
1	0	0	C_{in}	$A+B$
1	0	1	C_{in}	$A \oplus B$
1	1	0	C_{in}	$A \cdot B$
1	1	1	C_{in}	$\bar{A}$

図3: 算術演算機能抽出の例

6 おわりに

本稿では、算術演算回路の機能情報抽出について考察を行い、幾つかの算術演算機能が抽出できるシステムを紹介した。しかし現状ではまだまだ不十分な点が多く、これからいかにしてより多くの算術演算を取り扱えるようにしてゆくかが問題である。

参考文献

- [1] 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, Vol.88, No.78(1988年10月), pp.17-24.
- [2] 大村 安浦 田丸, “組合せ回路の機能情報抽出による機能シミュレーションモデルの自動生成”, 信学春全大予稿集(1989年3月), p.253.
- [3] R.E.Bryant, “Graph-Based Algorithms for Boolean Function Manipulation”, IEEE Trans. on Comput., Vol.C-35, No.8(Aug. 1986), pp.205-210.
- [4] 安浦, “VLSI用の正規集合認識法”, 情報処理, Vol.24, No.4(1983年4月), pp.567-571.