設計自動化 63－4
(1992. 7.17)

機能情報抽出の一応用 — CPU の動作検証

大村昌彦† 安浦寛人‡ 田丸啓吉†
†京都大学 工学部 電子工学科
京都府 京都市 左京区
‡九州大学 総合理工学研究科 情報システム学専攻
福岡県 春日市

あらまし 本稿では、我々が従来より提案してきている機能情報抽出の技術を設計検証に応用する手法について述べる。機能情報抽出においては、設計された回路記述に幾らかの付加情報を与えるだけで、機能記述が抽出できる。本手法は、膨大な回路記述から算術演算機能やRTレベルの記述などを抽出し、簡潔な機能記述で表すことが出来るという特長があるので、特にCPUのような回路の動作検証に適している。本手法を用いることにより、機能・論理設計の検証が容易に行なえるだけでなく、設計後に回路レベルで変更が生じた際に、それに応じて仕様を変更することにも利用できる。

An Application for Functional Information Extraction — Behavioral Verification of CPU

Masahiko OHMURA†, Hiroto YASUURA‡ and Keikichi TAMARU†
†Department of Electronics, Kyoto University
Kyoto 606-01, Japan
‡Department of Information Systems, Kyushu University
Fukuoka 816, Japan

Abstract In this paper, we will present a method to use the technique of functional information extraction which we have proposed for the design verification. Functional information extraction enables us to extract functional descriptions from designed circuit descriptions by utilizing some additional information. One of the merits of our method is that we can extract arithmetic functions or RT level descriptions from large scale circuits, and represent them concisely. Our method is suitable for behavioral verification of CPUs. We can not only verify the function design or the logic design but also change the specifications corresponding to the design change in the circuit level.

1 まえがき

LSI の設計において、設計された回路が仕様を満足しているかどうか、ということは非常に重要なことである。製造工程に入る前に全ての設計誤りを取り除くため、設計工程を階層化し、各設計段階において自動合成ツールを開発したり、設計検証を徹底的に行なうなどの努力が試みられている。特に、論理設計の分野においては、近年自動論理合成の手法が数多く提案されており [1][2] [3]、人手設計に劣らない優秀なツールも開発されつつある。しかし実際には、設計後の回路に人手で変更を加えたりすることなどが多くあり、自動合成された回路に対してでも、設計検証は必ず行なわれているのが実情である。

論理設計の検証手法としては、論理シミュレーションが広く用いられて来ている。しかし、大規模な回路においては、全ての入力パターンに対してその動作を検証することは不可能であり、ユーザが確かめたい幾つかのパターンに対して正常動作を保証しているに過ぎない。近年、シミュレーション技術も進歩しており、多くのパターンに対して高速な検証が行なえるようになってはいるものの、入力パターンを準備したり、結果を解析したりするのに手間がかかるという欠点も存在している。これに対し、シミュレーションを行なわない検証法として、形式的検証の手法が注目されている [4][5]。これは、仕様として与えられた機能と、設計された回路記述とをそれぞれ BDD [6] のような正規表現 (canonical form) で表し、比較するというものである。正規表現は、論理関数を唯一の形式で記述できるため、論理関数の照合等には非常に有効である。しかし、ALU などが持つ算術演算機能や、順序回路で実現される状態遷移やレジスタ間のデータ移動などの動作を検証するには向かない。

本稿では、我々が従来より提案している機能情報抽出 [7][8] の技術を、設計検証に利用する手法について述べる。機能情報抽出を用いれば、与えられた回路記述に幾らかの付加的な情報を与えることにより、算術演算機能や、更に高位レベルのレジスタトランスファレベルの記述などが得られるので、大規模な回路の機能を非常に簡潔に記述することが出来る。このため、本手法は特に CPU などの動作検証に向いていると言える。

第 2 節では、本手法の基礎技術について簡単に説明した後、本稿で対象とする回路について幾つかの定義を行なう。第 3 節では、CPU から動作記述を抽出する手法を説明する。第 4 節では、幾つかの場合を想定して、設計検証を行なう手法について述べる。第 5 節で本手法の評価を行ない、問題点等をまとめる。

2 準備

2.1 機能情報抽出の概要

機能情報抽出とは、論理回路レベルのネットリストからその機能情報を抽出し、機能表や機能記述言語で表す技術である。機能情報抽出を効率的に行なうためには、適切な付加情報を与えなければならない。付加情報とは、ネットリストには陽には記述されていない情報であり、組合せ回路の場合、次のようなものがある。

1. 各入力信号が、コントロール系であるかデータ系であるか。
2. コントロール系入力間の制御力の強さの順序。

更に、算術演算機能を抽出するためには、以下の付加情報が必要となる [9]。

3. 2 進数データを表す信号線が、それぞれどの 2 進数の第何ビットに対応するか。
4. 外部から最下位ビットへの桁上げ入力信号線の指定。
5. 最上位ビットから外部への桁上げ出力信号線の指定。

また、順序回路の場合、更に次の情報が必要である [10]。

6. クロック信号線の指定。
7. 各フリップフロップがコントロール系レジスタを構成するか、データ系レジスタを構成するか。
8. コントロール系レジスタの初期状態。

なお、本手法で取り扱える回路は、単一クロックが全てのフリップフロップに同時に加えられる同期式順序回路のみである。

上記のような付加情報は、その回路の設計者であれば容易に知り得る情報である。また、設計者でなくとも、回路記述における信号線の名前の付け方や回路図における端子の位置などから、ある程度推測できるような情報もある。本稿では、これらの付加情報はあらかじめ全てわかっているものとする。

2.2 本稿で取り扱う回路の構成

算術演算機能を論理式で表すと、複雑かつ記述量が膨大となるが、適切な付加情報を与えると、算術式を用いて簡潔に記述できる。また順序回路の場合、初期状態から遷移し得る状態のみが記述され、データ系レジスタの持つ値は状態とは見なさないので、状態数が非常に少なくなり、簡潔な記述が得られる。

このように、本手法の特徴は、複雑、大規模な回路の機能を簡潔に記述できる、ということである。この特徴を生かすため、本稿では機能情報抽出を CPU の動作検証に用いることを考える。

ここで対象とする CPU は、命令レジスタ (IR)、タイミングレジスタ (T)、プログラムカウンタ (PC)、メモリアドレスレジスタ (MAR)、メモリバッファレジスタ (MBR)、アキュムレータ (A)、汎用レジスタ (R) なる各レジスタから構成されるものとする。この中で、IR と T はコントロール系のレジスタで、その他のものはデータ系のレジスタであるとする。これは、2.1 で述べた付加情報の 7. に相当するものである。2.1 で述べた付加情報を与えるだけで、上記のような CPU から、レジスタ間のデータ転送記述を抽出することは可能である。しかし、そのデータ転送がどういう意味を持っているのかという情報まで抽出するためには、与えられた回路のどのレジスタが PC であり、どれが MAR であるか、などという付加情報が必要となる。

一般的な順序回路からレジスタ転送レベルの機能を抽出するのは困難であるが、本稿では対象を典型的な CPU に限定しており、回路中でどのようなレジスタが使われ、そのレジスタがどのような役割を果たすものであるかということは、あらかじめシステムに組み込んでおくことが出来る。従って、ユーザは上記のような付加情報を与えるだけで、その CPU の機能を抽出することが出来る。

コード	ニモニック	オペランド	動作
00	NOP	なし	何もしない
01	LDI	OPRD	A<-OPRD
10	LDA	ADDR	A<-M<ADDR>
11	ADD	OPRD	A<-A+OPRD

図 1: CPU の仕様

現在、さまざまな工業製品、家電製品に、単純な CPU が内蔵されている。そのような単純な機能を持った CPU でも、回路記述は膨大なものとなる場合があり、完全な設計検証は困難な作業である。このような場合に本手法を用いれば、簡潔な機能記述が抽出でき、容易に設計検証が行なえる。従って、対象回路を CPU に限定したとしても、本手法は充分実用的であると考えられる。

3 CPU からの機能記述の抽出

3.1 準備

本節では、図 1 に示す命令コード表を持つ簡単な CPU から機能情報を抽出する手法を説明する。図 2 は、図 1 の仕様から機能設計を行なった結果得られたブロック図である。図 3 は、組合せ制御論理部の論理機能である。図 2、図 3 をもとに論理設計を行なった結果得られる論理回路レベルの記述 (ネットリスト) から機能情報抽出を行なうのであるが、記述量が膨大となるためここではその記述を省略する。

この CPU は 4 命令を持つので、IR は 2 ビットで構成する。また T は 3 ビットとする。PC, MAR, MBR, A はそれぞれ 8 ビットである。IR にロード命令が与えられると、MBR の下位 2 ビットがロードされる。メモリ M は 8 ビット × 256 語を想定しているが、ここで用いる機能記述で単に M と書くことによって、MAR が示す番地の内容を表すことにする。例えば、MAR の値が 5 の時は、M はメモリの 5 番地の内容を表す。レジスタは全て、ビット幅分の D-FF から構成される。例えば、PC は 8 ビットのレジスタであるが、PC< 0 > から PC< 7 > までの 8 個の

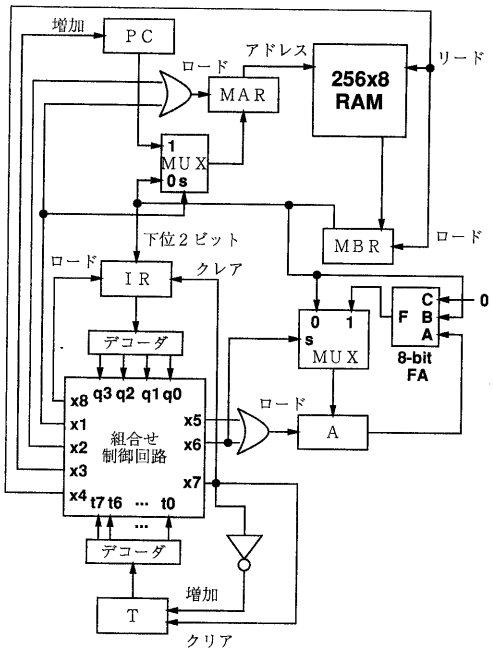

図 2: CPU のブロック図

x1 = q0 t0 + (q1 + q2 + q3) t3
x2 = q2 t5
x3 = q0 t1 + (q1 + q2 + q3) t4
x4 = q0 t1 + (q1 + q2 + q3) t4 + q2 t6
x5 = q1 t5 + q2 t7
x6 = q3 t5
x7 = q0 t3 + q1 t5 + q2 t7 + q3 t5
x8 = q0 t2

図 3: 制御論理部の機能

D-FF から構成される。なお、PC< 0 > が最下位ビットで、PC< 7 > が最上位ビットに相当する。

3.2 機能記述の抽出

3.2.1 付加情報

機能記述の抽出を行なうに先立ち、幾つかの付加情報が必要となる。2 節等で幾つか説明したが、ここでもう一度まとめておく。

- IR は、IR< 0 >, IR< 1 > なる 2 個の D-FF から構成される 2 ビットのコントロール系レジスタである。
- T は、T< 0 >, T< 1 >, T< 2 > なる 3 個の D-FF から構成される 3 ビットのコントロール系レジスタである。
- IR と T の初期値は、いずれも全ビット 0 である。
- IR と T とでは、IR の方が制御力が強い。
- PC, MBR, MAR, A はそれぞれ 8 個の D-FF から構成される 8 ビットのデータレジスタである。
- 各レジスタにおいては、配列の添字が 0 のものが最下位ビットに相当する。

3.2.2 機能表の作成

まず、与えられた回路記述から機能表を作成する。ここで機能表とは、組合せ回路の機能をコントロール系入力の値によって幾つかに分割し、それぞれを簡潔な論理式や算術式を用いて表形式で記述したものである。

最初に、順序回路のネットリストを組合せ回路に変換する。具体的には、順序回路のネットリストからクロック信号線と D-FF を取り除く。こうして内部にループを含まない組合せ回路のネットリストが得られる。この時、もともと D-FF の入力及び出力であった信号線は、それぞれ、新たに得られた組合せ回路における出力及び入力となる。ここで、IR< 0 > という D-FF の入力線を IR'< 0 > 、出力線を IR< 0 > と名付ける。IR< 0 > は、現在 D-FF が保持している値を表すので、現状態変数である。IR'< 0 > は、クロックが 1 回加えられた後に D-FF に保持される値を表すので、次状態変数である。他の FF についても同様にすると、新たに得られる組合せ回路は、IR< 1 >, IR< 0 >, T< 2 >, T< 1 >, T< 0 >, PC< 7 >, ..., PC< 0 >, MBR< 7 >, ..., MBR< 0 >, MAR< 7 >, ..., MAR< 0 >, A< 7 >, ..., A< 0 > を入力とし、IR'< 1 >, IR'< 0 >, T'< 2 >, T'< 1 >, T'< 0 >, PC'< 7 >, ..., PC'< 0 >, MBR'< 7 >, ..., MBR'< 0 >, MAR'< 7 >, ..., MAR'< 0 >, A'< 7 >, ..., A'< 0 > を出力とする 37 入力 37 出力の回路となる。ちなみに、この回路のゲート数は、607 であった。

IR			T		IR'		T'			PC'	MAR'	MBR'	A'
<1>	<0>	<2>	<1>	<0>	<1>	<0>	<2>	<1>	<0>				
0	0	0	0	0	0	0	0	0	1	PC	PC	MBR	A
0	0	0	0	1	0	0	0	1	0	PC+1	MAR	M	A
0	0	0	1	0	MBR<1>	MBR<0>	0	1	1	PC	MAR	MBR	A
0	0	0	1	1	0	0	0	0	0	PC	MAR	MBR	A
0	1	0	1	1	0	1	1	0	0	PC	PC	MBR	A
0	1	1	0	0	0	1	1	0	1	PC+1	MAR	M	A
0	1	1	0	1	0	0	0	0	0	PC	MAR	MBR	MBR
1	0	0	1	1	1	0	1	0	0	PC	PC	MBR	A
1	0	1	0	0	1	0	1	0	1	PC+1	MAR	M	A
1	0	1	0	1	1	0	1	1	0	PC	MBR	MBR	A
1	0	1	1	0	1	0	1	1	1	PC	MAR	M	A
1	0	1	1	1	0	0	0	0	0	PC	MAR	MBR	MBR
1	1	0	1	1	1	1	1	0	0	PC	PC	MBR	A
1	1	1	0	0	1	1	1	0	1	PC+1	MAR	M	A
1	1	1	0	1	0	0	0	0	0	PC	MAR	MBR	A+MBR

図 4: 抽出された機能表

ここで、IR< 1 >, IR< 0 >, T< 2 >, T< 1 >, T< 0 > の 5 入力をコントロール系入力として、機能情報抽出を行ない、機能表を作成する。例えば、IR< 1 >=0, IR< 0 >=0, T< 2 >=0, T< 1 >=0, T< 0 >=1 の時には、

```
PC'<0>=^PC<0>
PC'<1>=PC<0>@PC<1>
PC'<2>=PC<0>&(PC<1>@PC<2>)!^PC<0>&PC<2>
PC'<3>=PC<0>&(PC<1>&(PC<2>@PC<3>)!
        ^PC<1>&PC<3>)!^PC<0>&PC<3>
PC'<4>=PC<0>&(PC<1>&(PC<2>&(PC<3>@PC<4>)
        !^PC<2>&PC<4>)!^PC<1>&PC<4>)!
        ^PC<0>&PC<4>
PC'<5>=PC<0>&(PC<1>&(PC<2>&(PC<3>&(PC<4>
        @PC<5>)!^PC<3>&PC<5>)!^PC<2>&
        PC<5>)!^PC<1>&PC<5>)!^PC<0>&PC<5>
PC'<6>=PC<0>&(PC<1>&(PC<2>&(PC<3>&(PC<4>
        &(PC<5>@PC<6>)!^PC<4>&PC<6>)!
        ^PC<3>&PC<6>)!^PC<2>&PC<6>)!
        ^PC<1>&PC<6>)!^PC<0>&PC<6>
PC'<7>=PC<0>&(PC<1>&(PC<2>&(PC<3>&(PC<4>
        &(PC<5>&(PC<6>@PC<7>)!^PC<5>&
        PC<7>)!^PC<4>&PC<7>)!^PC<3>&
        PC<7>)!^PC<2>&PC<7>)!^PC<1>&
        PC<7>)!^PC<0>&PC<7>
```

という複雑な論理式が得られるが、これから算術演算機能を抽出することにより、PC'=PC+1 のように算術式を用いて簡潔に記述することが出来る。このようにして得られた機能表を図 4 に示す。なお、記号 ^,@,&,! は、それぞれ否定、排他的論理和、論理積、論理和を表し、+ は算術加算を表す。

3.2.3 状態遷移記述の抽出

図 4 は、組合せ回路の機能表としてみれば、単に入出力間の機能を記述しているに過ぎない。しかし、前述のように、この組合せ回路の入力変数は現状態変数であり、出力は次状態変数であるので、この機能表はクロックが加えられる前後の状態の変化、即ち状態遷移の様子を記述していると見なすことが出来る。ここで、レジスタ IR と T が表す値を添字に持つ文字 Q で状態を表すことにする。例えば、IR< 1 >=0, IR< 0 >=1, T< 2 >=0, T< 1 >=1, T< 0 >=1 である状態は、Q13 と表す。こうして、図 5 に示すような状態遷移図、あるいは図 6 に示した状態遷移記述が得られる。なお、ここでは機能記述言語として UDL/I [11] を用いた。

3.2.4 命令コード表の抽出

図 6 の記述を抽出することが出来れば、機能シミュレーションを行なうことにより、設計検証を行なうことが出来る。ここで、各状態で行なわれている操作の意味を考え、初期状態から数クロック後に再び初期状態に戻ってくるまでの間にどの

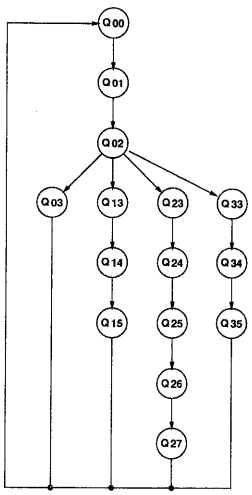

図 5: 状態遷移図

ような機能が実現されているかということが求まれば、命令コード表を作成することが出来よう。

例えば、 Q00 → Q01 → Q02 と状態が遷移する間に、

```
MAR := PC ;
PC := INC(PC) ;
MBR := M ;
IR := MBR<1:0> ;
```

という一連の操作が行なわれている。PC が実行中のプログラムのメモリ上の番地を、MAR が現在アクセス可能なメモリ上の番地を表し、MBR はメモリから読み出された値を格納するバッファであり、IR が実行する命令のコードを格納するレジスタである、ということがあらかじめわかっていれば、これらの操作によって、「メモリ上に記述されているプログラムから一つの命令を読み出し、そのコードを IR にセットしている」、ということがわかる。

次に、 Q13 → Q14 → Q15 と遷移する場合、

```
MAR := PC ;
PC := INC(PC) ;
MBR := M ;
A := MBR ;
```

という操作がなされるが、これは上と同様にして、「先程読み出した命令語の次の番地に格納されて

```
Q00:    MAR:=PC; -> Q01;
Q01:    PC:=INC(PC); MBR:=M; -> Q02;
Q02:    IR:=MBR<1:0>;
        CASE IR OF
            #2B00 -> Q03; #2B01 -> Q13;
            #2B10 -> Q23; #2B11 -> Q33;
        END_CASE;
Q03:    -> Q00;
Q13:    MAR:=PC; -> Q14;
Q14:    PC:=INC(PC); MBR:=M; -> Q15;
Q15:    A:=MBR; -> Q00;
Q23:    MAR:=PC; -> Q24;
Q24:    PC:=INC(PC); MBR:=M; -> Q25;
Q25:    MAR:=MBR; -> Q26;
Q26:    MBR:=M; -> Q27;
Q27:    A:=MBR; -> Q00;
Q33:    MAR:=PC; -> Q34;
Q34:    PC:=INC(PC); MBR:=M; -> Q35;
Q35:    A:=ADD(A,MBR,1B0); -> Q00;
```

図 6: UDL/I による状態遷移記述

いる値を A に格納している」、ということがわかる。以上よりこの命令は、命令語の次の値 (オペランド) を A に格納する命令 (即値ロード) であると解釈される。このように人手の援助があれば、図 1 に仕様として与えられた命令コード表を抽出することが可能となる。

4 設計検証への応用

ここでは、前節までに説明した手法に基づき、CPU の動作検証を行なう方法について述べる。

まず、回路全体の機能検証についてであるが、これは 3.2.4 で述べたように命令コード表を抽出することが出来れば、仕様と目で比較することにより容易に行なえる。また、命令コード表を抽出することが出来なくとも、3.2.3 で述べたように機能記述言語を抽出することが出来れば、機能シミュレーションを行なうことにより、その動作を確認することが出来る。

次に、設計変更が生じた場合の検証について考える。一般に CPU の設計においては、データパス部はモジュールジェネレータを用い、制御論理部は自動論理合成を行なうので、設計誤りの混入する余地はない。しかし、設計結果を見て、回路レベルで人手で設計変更を加えることが実際にしばしば行なわれているので、その変更が仕様に影響を与えるか否かを調べる必要がある。例えば、制御論理部の面積を小さくするために、回路レベルで設計変更を行なった結果、図 7 のように、端

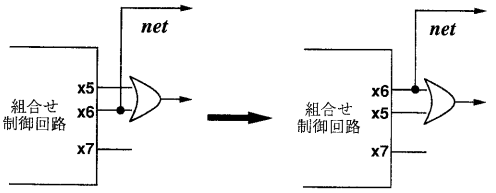

図 7: 制御論理部の端子位置の変更

子位置が入れ替わってしまったとする。この場合、図 7 における *net* の接続点を変更すれば問題は生じないが、この付け替えを忘れてしまった場合に設計誤りとなる。制御論理部には誤りが存在しないので、この部分にシミュレーションを行なっても誤りは発見できない。この誤りは、「MBR の内容を A に移す」という操作と、「MBR の内容を A に加える」という操作が入れ替わっているものである。A が全ビット 0 の時には、この誤りは表面化しないので、通常のシミュレーションでは誤りの発見は困難である。このような場合に、前節で述べた手法を用いて機能記述あるいは命令コード表を抽出すれば、容易に誤りに気付く。また逆に、ここで抽出された命令コード表を新たに仕様として、仕様変更がなされたものと見なすことも出来る。

5 評価および問題点

本手法に基づき試作したシステムの特長を以下に挙げる。

(1) 複雑な回路記述から、簡潔な機能記述を抽出することが出来る。

(2) 設計誤りの存在箇所が、ある程度推定できる場合がある。

(3) 論理関数の内部表現に BDD を用いており、処理の高速化、省メモリ化が図られている。

(1) について具体的な数字を挙げると、3 節で説明した例においては、入力として与えられるネットリストは 612 行の記述からなり、抽出される UDL/I 記述は 41 行、命令コード表は 4 行である。

(2) については、4 節で説明した例の場合、抽出された機能記述と仕様を比較すれば、`A:=MBR` と `A:=A+MBR` の操作が入れ替わっていることがわかるので、制御論理部における信号線 x5, x6 のあたりに設計誤りが存在することが予想される。

(3) については、我々の試作したシステムでは、京都大学で開発された属性エッジ付き共有二分決定グラフ (SBDD)[12] を用いており、変数シフトエッジなどを利用することにより、特に算術関数の記述がコンパクトになっている。3 節で説明した例において、図 4 の機能表を抽出するのに要した BDD の節点数は、1741 であった。

本手法の問題点としては、

- 適切な付加情報を与えなければならない。
- 同期式順序回路しか取り扱えない。
- 入力ネットリストは論理ゲートレベルで記述されねばならない。
- 算術機能を抽出するには、その論理式表現をあらかじめ登録しておかねばならない。

などが挙げられる。付加情報については、2.2 でも述べたが、現状ではユーザがあらかじめ知っているものと仮定している。設計検証を行なうユーザはその回路の設計者であるから、その回路に関する付加情報を知っていると仮定しても構わない、という考え方である。将来的には、論理設計の際に機能レベルの記述に含まれている情報を出来るだけ失うことなく回路記述に反映させることの出来るような論理合成の手法が開発されれば、設計者でなくとも回路記述から容易に付加情報を得ることが出来るようになると考えられる。

本手法は現在、同期式順序回路しか取り扱うことが出来ない。非同期回路においては、遅延情報などを考慮せねばならず、機能が複雑になるからである。しかし、非同期式順序回路においても、同期的な動作を行なっている部分の機能を抽出するだけでも有用であると考えられる。なお、本システムの入力は、論理ゲートレベルのネットリストで記述されねばならず、回路中にフィードバックループが含まれる場合には、そのループ中に必ずフリップフロップが含まれなければならない。完全 CMOS の組合せ回路については、スイッチレベルのネットリストを入力とすることも可能であるが、順序回路には対応していない。

3.2.2 において、論理式から算術機能を抽出する例を示したが、これはあらかじめインクリメン

トという演算が論理式でどのように表されるか、ということをデータベースに蓄えておいて、可能となる。現在我々のシステムでは、加算、減算、乗算、インクリメント、デクリメント等の算術演算、ビット毎の論理演算、シフト演算などが登録されている。これらの演算の合成演算等は登録されていないので抽出できないが、CPU においてそのような演算を 1 クロックで行なうようなマイクロ操作は存在しないと考えられるので、対応はしていない。

6 あとがき

本稿では、我々が従来より提案している機能情報抽出の技術を用いて、CPU の機能記述を抽出し、その動作検証を行なう手法について述べた。本手法の特徴として、膨大な回路記述から簡潔な機能記述を抽出することが出来るという点が挙げられ、CPU 等の動作検証には非常に適していると考えられる。今後の課題としては、各素子における遅延情報を考慮して、タイミング検証を行なう、ということなどが挙げられる。

参考文献

[1] S. Devadas and A. R. Newton : "Exact algorithms for output encoding, state assignment and four-level boolean minimization", IEEE Trans. CAD, pp. 13–27 (January 1991).

[2] L. Lavagno, S. Malik, R. K. Brayton and A. Sangiovanni-Vincentelli : "MIS-MV : Optimization of multi-level logic with multiple-valued inputs", Proc. ICCAD-90, pp.560-563 (Nov. 1990).

[3] T. Villa and A. Sangiovanni-Vincentelli : "NOVA : State assignment of finite state machines for optimal two-level logic implementation", IEEE Trans. CAD, pp.905-924 (Sep. 1990).

[4] J. C. Madre and J. P. Billon : "Proving Circuit Correctness using Formal Comparison Between Expected and Extracted Behaviour", Proc. 26th DAC, pp.205–210 (June 1988).

[5] K.Hamaguchi, H.Hiraishi, and S.Yajima : "Formal Verification of Sequential Circuits Based on Symbolic Model Checking for Branching Time Regular Temporal Logic", Proc. SASIMI'92, pp. 243–252 (Apr. 1992).

[6] R. E. Bryant : " Graph-Based Algorithms for Boolean Function Manipulation", IEEE Trans. Comput., Vol. C-35, No. 8, pp. 677–691 (Aug. 1986).

[7] M. Ohmura, H. Yasuura and K. Tamaru : "Extraction of Functional Information from Combinational Circuits", Proc. ICCAD-90, pp.176–179 (Nov. 1990).

[8] 大村昌彦, 安浦寛人, 田丸啓吉 : "組合せ回路の機能情報抽出", 信学論 A Vol. J 74-A, No. 2, pp. 247–255 (Feb. 1991) .

[9] M.Ohmura, H.Yasuura, and K.Tamaru : "Extraction of Arithmetic Functions from Combinational Circuits", 信学技報 VLD 90-104, pp. 51–57 (Feb. 1991).

[10] 大村昌彦, 安浦寛人, 田丸啓吉 : "順序回路からの算術演算機能の抽出", 情処研報 DA 60-12 / 信学技報 VLD 91-100, pp. 91–97 (Dec, 1991).

[11] O.Karatsu : "UDL/I: progress and effort for the real logic synthesis", Proc. SASIMI'92, pp. 105–114 (Apr. 1992).

[12] 湊真一, 石浦菜岐佐, 矢島脩三 : "論理関数の共有二分決定グラフによる表現とその効率的処理手法", 情処論, Vol. 32, No. 1, pp. 77–85 (Jan. 1991).