5P-1

定理証明法によるハードウェア演算部の検証

藤田　昌宏　田中　英彦　元岡　達
（東京大学　工学部）

1.　はじめに

我々は、時相論理（temporal logic）を用いてハードウェアの仕様を記述し、処理系にProlog を使用して、ゲート回路やDDLの記述を検証できることを示した。［1、2］

仕様は命題論理の範囲で記述し、必要な全ての場合について調べることによって検証している。従って、シーケンサ等の制御回路ではよいが、ALU等の演算回路では、データのビット幅の増加に対し、調べる場合の数が指数的に増大し、処理時間が非常に長くなってしまう。そこで、演算回路の仕様は1階述語論理により記述することにし、定理証明用のLCF［4］と呼ばれるproof checkerを用いて検証することを考え、検討する。

2.　演算部と同期部

システムは、ALUのように実際に論理・算術演算を行なう演算部（function part ）と、各演算を行なうモジュール間のデータ転送のタイミングを制御し同期をとる同期部（synchronization part）に分けることができる。例えば図1に示す簡単な計算機［3］において、左下の状態遷移と示されているブロックがマシンサイクルの状態遷移全体を制御する同期部であり、その他のレジスタやそれらに付随するゲートが全て演算部となる。

同期部の検証は、並列に動作するもの全体を考えねばならず、人間にとって不得意な領域であり見落し等による誤設計も多い。従って特に自動検証が望まれる。また一般に同期部の回路規模は、かなり複雑な制御を行なう場合でも、うまく階層設計されていればそれほど大きくならないと考えられる。これに対し演算部は、細かいタイミングはそれほど問題とならず、また既に設計されているものを一部手直しして使うことも多い。つまり、演算部の検証には、過去の設計から得られた知識をかなりの部分使うことができ、従って検証手法もそれらの知識を用いるものが適合している。

以上述べたように、同期部の回路規模はそれほど大きくならないこと、及び、できるだけ自動検証が望まれることから、従来通り命題論理の範囲で記述し、しらみつぶしに必要な全ての場合を調べる方法が向いている。一方、演算部の検証は、命題論理で記述し、全ての場合を調べることは、計算機スピード・スペースの点で不可能であり、また、設計者の知識をうまく利用できる手法が望まれるため、1階述語論理の範囲で記述し、定理証明のためのpoof checkerを用いて設計者が対話的に検証していく方法が向いている。ここでは、proof checker を用いた演算部の検証について、例を用いて検討する。

3.　演算部の仕様記述

ここでは、同期回路をゲートレベルで記述し、論理的な正しさの検証を行なうことを考える。演算部の仕様は通常、『同期部から正しく制御信号が送られてくるならば、指定された時間で演算が終了する』という形になる。

一般に、演算部はデータのビット幅に応じて、同じパターンの繰り返しであることが多い。従って例えば、『プログラムカウンタの値が1だけインクリメントされる』は、次のよ

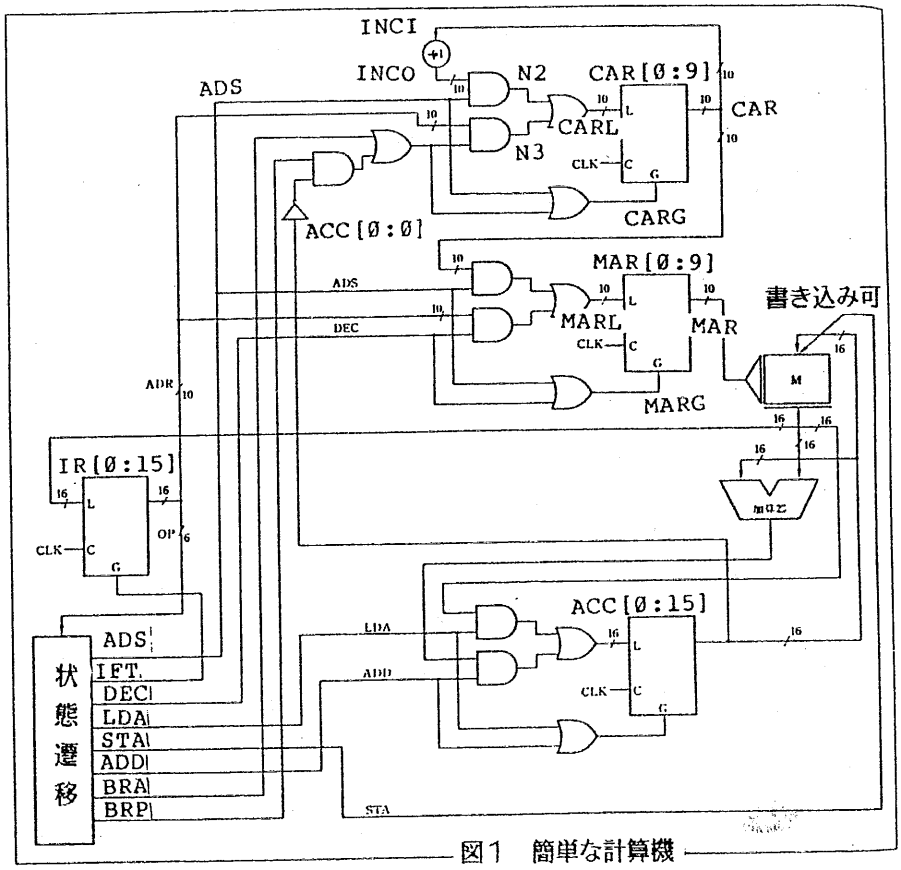

図1　簡単な計算機

うに1階述語論理で記述できる。

○(PC[0:15]) =
PC[0:15] PLUS 1

(ただし、○は時相論理の演算子[1]で、『次の時刻』を表わし、R[i:j]はレジスタRのiビット目からjビット目の要素を表わす。また、"PLUS"は加算を表わす)

この"PLUS"や"∧"、"∨"のような各種演算の性質については、公理として仕様とは別に、前もってproof checkerに与えておく。その一部を図2に示す。この公理が豊富であればあるほど、証明が容易になる。

仕様記述の例として、図1の簡単な計算機の一部を記述したものを図3に示す。

4. Proof checker:LCFについて

LCF[4]は、Edinburgh大学で開発されたproof checkerでLogics for Computable Functionsを支援するようになっている。これは、1階述語論理より推論能力は落ちるが、ここでの議論にはさしつかえない。LCFは、Frantz-lisp上のMeta-Languageで記述されており、DEC系の計算機上で動く。また、証明戦略を手続き的に記述して使用することもでき、簡単なものは自動証明することもできる。

5. 検証例

図1の設計に対し、図3の仕様を検証することを考える。まず、回路の接続情報を証明における仮定として入力していく。例えば、レジスタCARの入力CARLについては、次のようになる。

$0 \leqq \forall i \leqq 9.$
$CARL[i:i] = N2[i:i] \vee N3[i:i],$
$N2[i:i] = INCO[i:i] \wedge ADS.$
.....

また、レジスタの性質として次のことが言える。

○(CAR[0:9]) = (CARG∧CARL[0:9])
∨(CARG∧CAR[0:9])

証明手順は、図3の各仕様に対して"−>"の左辺を仮定し、公理や接続情報を用いて右辺を導きだすようにする。具体的には、『接続情報を次々と代入してまとめ、簡単化規則を適用する』ということを設計者がproof checkerに指示することで右辺を得ることができ、検証できる。

検証に必要な式変形の手間の数は、図3の場合で約40〜60ステップである。

6. おわりに

LCFでは、一定の証明手順をプログラムとして登録することができる。従って、5.で示した程度の検証は完全に自動的に行なうことができる。

検証対象を演算部に絞っているので、回路設計者の知識をうまく活用することにより、効率的に検証することができると考える。

7. 参考文献

[1] 第26情報処理全国大会、3K−1、3K−2
[2] 第27情報処理全国大会、3L−11
[3] 日経エレクトロニクス、No.225、1979
[4] Lecture Notes in Computer Science No.78、1979


```
<module> CPU ;

<control_var> ADS, IFT, DEC, LDA, STA,
              ADD, BRA, BRP ;

<data_regiter> MAR[0:9], CAR[0:9],
     M[0:1023,0:15], ACC[0:15], IR[0:15] ;

<spec>
 IR[6:15] = ADR[0:9],

 (ADS ∧ ¬IFT ∧¬DEC ∧¬LDA ∧¬STA ∧¬ADD ∧¬BRA ∧¬BRP) ->
      ( ○(MAR[0:9]) = CAR[0:9])
   ∧( ○(CAR[0:9]) = CAR[0:9] PLUS 1),

 (¬ADS ∧IFT ∧¬DEC ∧¬LDA ∧¬STA ∧¬ADD ∧¬BRA ∧¬BRP) ->
      ( ○(IR[0:15]) = M[MAR[0:9],0:15]),

 (¬ADS ∧¬IFT ∧DEC ∧¬LDA ∧¬STA ∧¬ADD ∧¬BRA ∧¬BRP) ->
      ( ○(MAR[0:9]) = ADR[0:9]),
                . . .
```

図3 図1の計算機の演算部の仕様の一部

```
BA = {0,1}
T1A  0 = 1
T1B  1 = 0
T2A  X:BA. X∨1 = 1
T2B  X:BA. X∧0 = 0
T2C  X:BA. 1∨X = 1
T2D  X:BA. 0∧X = 0
T2E  X:BA. X PLUS 0 = X
T2F  X:BA. 0 PLUS X = X
T3A  X:BA. X∨0 = X
T3B  X:BA. X∧1 = X
          .....
          .....

A1A ∀(X,Y,C):BA. ∀(i,j):INT.
 CARRY(X[i,i],Y[i,i],C) =
     (X[i:i] ∧Y[i:i]) ∨((X[i:i] ∨Y[i:i]) ∧C)
A1B ∀(X,Y,C):BA. ∀(i,j):INT.
 CARRY(X[i,j+1],Y[i,j+1],C) =
     (X[j+1:j+1] ∧Y[j+1:j+1])
    ∨((X[j+1:j+1]∨ Y[j+1:j+1]) ∧CARRY(X[i:j],Y[i:j],C))
A4  ∀(X,Y,C):BA. ∀(i,j,k):INT.
 (X[i:j] PLUS Y[i:j] PLUS C) @
 (X[j:k] PLUS Y[j:k] PLUS CARRY(X[i:j],Y[i:j],C)) =
 X[i:k] PLUS Y[i:k] PLUS C

 (@ is a concatination operator)
```

図2 演算子に対する公理の例