5F-3

Tokioによるレジスタトランスファレベル記述の動作解析ツール

中村 宏・*藤田 昌宏・河野 真治・田中 英彦
東京大学工学部・*富士通研究所

1. はじめに

我々は従来より時相論理型言語Tokio[1]を中心としたハードウェア設計支援システム[3,4]を考えている。Tokioは時相論理に基づいているために時間に関する記述が容易かつ厳密にできるハードウェア記述言語であり、また仕様からレジスタトランスファレベル(以下RTLと省略する)まで様々なレベルを記述できる。

今回、我々は自動パイプライン化支援システムの開発を開始した[2]。ここでは、[2]で述べられているシステムのうち、Tokioで記述されたRTLの記述の動作を解析するツールについて述べる。

2. TokioによるRTLの記述の解析

このツールは、Tokioで記述されたRTLレベルの記述を受取りレジスタ等の衝突の検出をするものであり、さらにデータパスが同時に指定された場合にはそのデータパス上での動作可能性も検出するもので、前者については現在Prologを用いて実装中である。

```
main :-
       stop(Req = 1), keep(Ack := 0), COUNT <- 0
    &&
       %% --- main1 --- %%
       keep(if(stage_load_ok = 1) then load),
       stop(Pipeline_Stop = 1).

load :-
       @ keep(stage_load_ok := 0),
       fin(stage_load_ok := 1),
       COUNT <- COUNT + 1, Ack := 0,
       R1 <- Input, R2 <- R1
    &&
       %% --- load1 --- %%
       if(COUNT = 2),
          then if(stage_sort_ok = 1)
                    then sort.

sort :-
       (@ keep(stage_sort_ok := 0),
        fin(stage_sort_ok := 1)),
       (COUNT <- 0, Ack := 1,
        if (R1 < R2) then (Output := R1)
                     else (Output := R2, R2 <- R1)
    &&
       %% --- sort1 --- %%
       Output := R2, Ack := 1).
```

図1. パイプラインマージソータ(2入力)の動作記述

例として図1の記述を解析する。これはパイプラインマージソータ[5]の1段目の2入力の部分であり、自動パイプライン化ツールによってシーケンシャルな記述から導出されたパイプライン化された動作記述[2]を、少し簡潔にしたものである。

まずこの記述からレジスタの状態遷移表を作成すると、図2のようになる。ここでレジスタCOUNTに着目すると、(a),(b),(c)が同時に発生することはレジスタの衝突を意味する。今それぞれの遷移における遷移条件が排他的でないので、レジスタの衝突の可能性が残る。また、それぞれの遷移において、現在の状態はmain, load, sortであり、この3つの状態の生起条件は図3の通りである。ここで、mainは一度しか呼ばれないのに対し、loadはkeep演算子の中で呼ばれ何度も生起し、sortもloadの後で何度も生起する。さらに、loadとsortの生起条件は排他的でないので、(a)と(b)あるいは(a)と(c)は同時に起こらないが、(b)と(c)は同時に起こりうるこ

レジスタ名	COUNT	Range(0:0)		
現在の状態	遷移条件	遷移後の値	遷移後の状態	
main	Req ≠1	0	main1	…(a)
load	……	COUNT+1	load1	…(b)
sort	……	0	sort1	…(c)

レジスタ名	R2	Range(0:0)	
現在の状態	遷移条件	遷移後の値	遷移後の状態
load	……	R1	load1
sort	R1≧R2	R1	sort1

図2. レジスタの状態遷移表[一部]

状態	生起条件	以前の状態
main	Req ≠1	(初期状態)
load	stage-load-ok=1 (Keep 演算子の中)	main
sort	stage-sort-ok=1	load

図3. 状態の生起条件

Analyzer for register transfer level description in Tokio
Hiroshi NAKAMURA, *Masahiro FUJITA, Shinji KONO, Hidehiko TANAKA
University of Tokyo, *FUJITSU LABORATORIES LTD.

とがわかる。同様にR2についても考えると、結局、COUNT 及びR2でレジスタの衝突が起こっていることがわかる。

さらに、図1 に対応して図４のようなデータパスを与えらえると、図５のようなデータパスに関する遷移表を作成し、同様に例えばp2でデータパスの衝突が生じることも検出できる。この部分に関してはデータパスの表現法を検討中である。

３．レジスタの衝突の回避

次に、レジスタ等の衝突の回避について考える。レジスタの衝突を回避する方法としては、以下の３つが考えられる。

①レジスタを追加する
②インターバルの長さを変更する
③アルゴリズムを変更する

例えばレジスタR2について、①の手法を用いると、sortの部分で新たにレジスタR3を追加し、図６のように変更すればよい。またレジスタCOUNT について③の手法を用いるならば、最終的に図７の記述が得られることになる。

この記述からレジスタの状態遷移表を作成すると、図８のようになり、レジスタCOUNT, R3のそれぞれの遷移において、遷移条件が排他的であるため、レジスタの衝突がないことを保証できる。

この修正は人間が行うことを仮定している。

４．おわりに

Tokio で記述されたＲＴＬの記述の動作を解析するツールについて述べた。現在のところ、Tokio による一貫したハードウェア設計支援システムを目指すために、データパスの表現もTokio を用いることを検討している。今後はその表現法を決めるとともに、ツールの実装を急ぎたい。

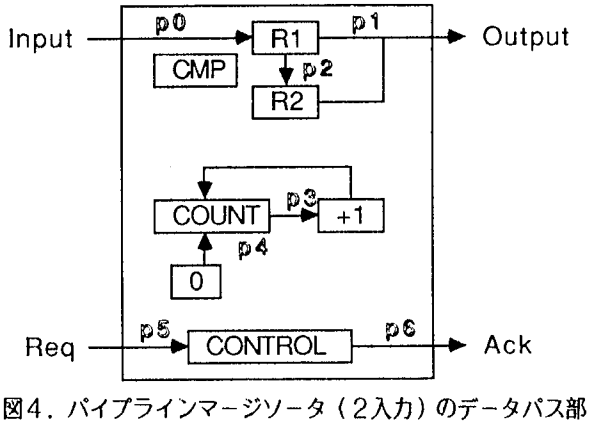

図４．パイプラインマージソータ（２入力）のデータパス部

データパス名　p２		Range(0:0)
現在の状態	遷移条件	遷移後の状態
load	……	load1
sort	R1≧R2	sort1

図５．データパスに関する遷移表

参考文献

[1]IEE Proc. E, Vol.133, No.5, pp.284-294,Nov. 1986

[2] 第３５回情報処理学会全国大会,5F-1, 1987

[3]S.Kono,et al."Implementation of temporal logic programming language Tokio",Proceednge of LPC'85,Springer-Verlag

[4] 中村他,"Tokio に基づく論理回路の検証" 情報処理学会設計自動化研究会34-1, 1986

[5]M.Kitsuregawa,et al."Relational Algebra Machine GRACE",Lecture Notes in Computer Science 147,Springer-Verlag


```
sort :-
       (@ keep(stage_sort_ok := 0),
        fin(stage_sort_ok := 1)),
       (COUNT <- 0, Ack := 1,
        if (R1 < R2) then (Output := R1, R3 <- R2)
                     else (Output := R2, R3 <- R1)
   &&
      Output := R3, Ack := 1).
```

図６．動作記述の訂正(1) ［一部］

```
load :-
       @ keep(stage_load_ok := 0),
       fin(stage_load_ok := 1),
       Ack := 0, R1 <- Input, R2 <- R1,
       (if(COUNT = 2)
           COUNT <- 1
        else
           COUNT <- COUNT + 1)
    &&
       if(COUNT = 2),
           then if(stage_sort_ok = 1)
                   then sort.
sort :-
       (@ keep(stage_sort_ok := 0),
        fin(stage_sort_ok := 1)),
       (Ack := 1,
        if (R1 < R2) then (Output := R1, R3 <- R2)
                     else (Output := R2, R3 <- R1)
   &&
      Output := R3, Ack := 1).
```

図７．動作記述の訂正(2) ［一部］

レジスタ名　COUNT		Range(0:0)	
現在の状態	遷移条件	遷移後の値	遷移後の状態
main	Req ≠1	0	main1
load	COUNT =2	COUNT+1	load1
load	COUNT ≠2	1	load1

レジスタ名　R3		Range(0:0)	
現在の状態	遷移条件	遷移後の値	遷移後の状態
load	R1＜R2	R1	load1
load	R1≧R2	R2	load1

図８．訂正後のレジスタの状態遷移表［一部］