1F-2

Ｔｏｋｉｏによるレジスタトランスファレベル記述からの論理回路の自動合成

中村　宏・河野　真治・藤田　昌宏＊・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 を中心としたハードウェア設計支援システムを考えている。

Tokio はハードウェア記述言語であり、仕様からレジスタトランスファレベル（以下ＲＴＬと略す）までの様々なレベルを記述することができる。また既にＤＤＬトランスレータの出力からＣＭＯＳゲートアレイ用の回路を合成するツールができている［１］。従ってTokio による仕様記述からＲＴＬの記述に変換し、さらにＤＤＬトランスレータの出力に相当する状態遷移表を作成することによって仕様記述から一貫して回路の自動合成を行ういわゆるシリコン・コンパイラを実現できる。（図１）

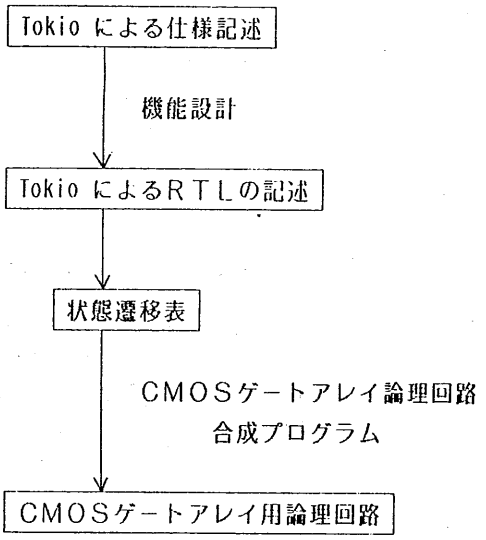

図１．システムの構成図

本稿では、Tokio の仕様記述をＲＴＬの記述に変換し状態遷移表を作成する手法を示す。

２．Tokio による仕様記述とＲＴＬの記述

ここでは簡単な例として、図２のMultiplierを考える。複雑な例については［２］を見られたい。

図２．Multiplier

Tokio による仕様記述は図３のようになる。

```
idle(A,B,C,Req,Ack):-halt(Req=1),keep(Ack=1)
                    && main(A,B,C,Req,Ack).

main(A,B,C,Req,Ack):-C<-A*B,keep(Ack=0),
          fin(Ack=1) && idle(A,B,C,Req,Ack).
```

図３．Multiplierの仕様記述

Tokio についての詳細は［３，４］を見られたい。Ｃ＜－Ａ＊Ｂの部分はＡをＢ回加算することでＣを求めるという機能設計をすると、図４のような対応するＲＴＬの記述が得られる。

```
idle(A,B,C,Req,Ack):-Req=1,!,*Ack-r=0
                    && main(A,B,C,Req,Ack).
idle(A,B,C,Req,Ack):-Req=0,!,*Ack-r=1
                    && idle(A,B,C,Req,Ack).
main(A,B,C,Req,Ack):-B≠0,!,*RA<-A,*RB<-B-1,
        *RC<-0 && main1(A,B,C,Req,Ack).
main(A,B,C,Req,Ack):-B=0,!,*RC<-0,*Ack-r<-1&&
                    idle(A,B,C,Req,Ack).
main1(A,B,C,Req,Ack):-*RB ≠0,!,*RC<-*RA+*RA,
      *RB<-B-1 && main1(A,B,C,Req,Ack).
main1(A,B,C,Req,Ack):-*RB=0,!,*Ack-r<-1&&
                    idle(A,B,C,Req,Ack).
```

図４．Tokio によるＲＴＬの記述

Automatic Synthesis from Register Transfer Description in Tokio
Hiroshi NAKAMURA, Shinji KONO, Masahiro FUJITA* , Hidehiko TANAKA
University of Tokyo ,*FUJITSU LABORATORY.

Tokio には２種類の変数がある。１つはそのclauseが定義されているinterval内でのみ定義されているlocal 変数、もう１つは、全てのinterval（即ち全ての時刻）で値が定義されているglobal変数である。後者の場合、変数名の頭に＊が付く。

local 変数はその定義から新しいclauseがcallされたとき以前の値を保持しておらず、terminalに対応すると考えられ、一方global変数は以前の値を覚えており、レジスタに対応すると考えられる。

そこで、このＲＴＬの記述ではterminalがA,B,C,Req,Ack でありレジスタ変数がRA,RB,RC,Ack-rである。また、レジスタ間の操作には転送以外にはadder とdecrement が必要なので、図５のようなMultiplierの構成ができる。

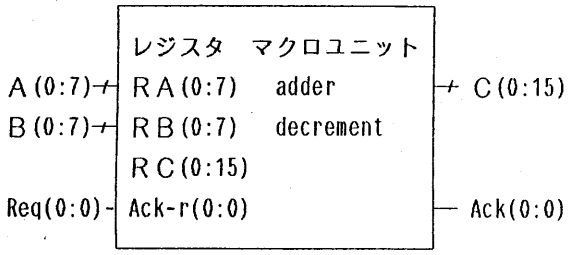

図５．Multiplierの構成

３．ＲＴＬの記述からの状態遷移図の合成

図４の記述を見た場合、全てのclauseは

現在の状態名　：－　条件，！，
　　レジスタ間転送　＆＆　次の状態名.

という形をしている。

また図４では明記していないが、条件部のinterval の長さは全て１と考えられる（この場合、そのように変換できている）。するとこの記述は状態遷移表現そのものであり、bit 幅を考慮すると、図６が得られる。

ここで問題になるのは、dec がひとつで十分であることを判定することである。この例題では、レジスタの遷移部が全て長さが１でタイミングが分かり易く、２つ以上のdecrement が重なることはない。複雑な例の場合でも、ＲＴＬの記述のレジスタ遷移部の長さを常に１にすることは可能ではあるが、必要以上のレジスタ変数を作ることになる。この問題は、ＲＴＬの記述に変換するときに考慮しなければならない［２］。

```
register(RA,RB,RC,Ack-r).
terminal(A,B,C,Req,Ack).
facility(adder,dec).
adder(adder1).
dec(dec1).
```

レジスタ名　RA(0:7)			
RANGE (0:7)			
遷移後の値	条件	現在の状態	遷移後の状態
A(0:7)	B≠0	main	main1

レジスタ名　Ack-r(0:0)			
RANGE (0:0)			
遷移後の値	条件	現在の状態	遷移後の状態
0	Req=0	idle	main
1	Req=1	idle	idle
	B=0	main	idle
	RB=0	main1	idle

図６．Multiplierの状態遷移表（一部）

４．おわりに

本稿では、Tokio による仕様記述から状態遷移表を作成する手法を述べた。今後は、その部分の実装、及び、既にできているＤＤＬトランスレータの出力からＣＭＯＳゲートアレイ用回路を合成する部分とのinterface を作成する予定である。

参考文献

［１］石曽根：情報処理学会第30回大会,4H-2,1985
［２］河野：情報処理学会第34回大会,1F-1,1987
［３］青柳：Tokio かける言語，Logic Programming Conference 8.1 ICOT,1985
［４］河野：時相論理型言語Tokio の実装，Logic Programming Conference 8.2 ICOT,1985