1U-3 RTL－Tokio：レジスタトランスファレベル動作記述言語

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、時相論理型言語Tokio[1]を中心とした、図1に示す論理設計支援システム[2]を構築中である。Tokioは時相論理に基づくため、順序性・並列性といった時間に関する記述が容易かつ厳密にでき、またアルゴリズムレベルからレジスタトランスファレベルまでの様々なレベルの動作記述が行えるハードウェア記述言語である。

本稿では、図1のシステムのTokioによるアルゴリズム記述及びRTL-Tokioによるレジスタトランスファレベルの記述について述べる。

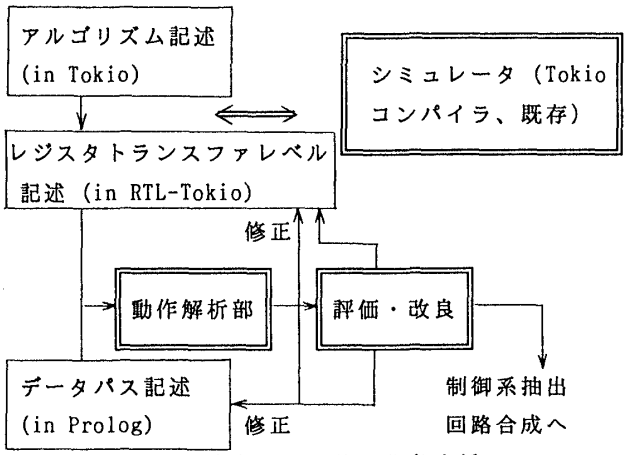

図1．Tokioを中心とした論理設計支援システム

2．システムの構成

このシステムにおけるハードウェア設計の流れは以下のようである。

まず、設計者は設計したいハードウェアの動作アルゴリズムをTokioで記述する。次に、シミュレーション等で動作の確認を行いながら記述を徐々に詳細化し、レジスタトランスファレベルの動作記述とする。レジスタトランスファレベルの記述は、ハードウェアとの対応が比較的容易にとれるようにTokioを制限したRTL-Tokioを用いて行う。

設計者は同時にデータパス等の構造の記述をPrologで行う。従来から、動作記述からデータパスを自動生成する研究が行われているが、現在のところ結果の品質は必ずしも十分ではない。そこでここでは、動作記述と構造記述は別に与え、支援システムが自動的に対応をとることにする。

次に、動作解析部において、動作記述と構造記述との間の実現可能性のチェックが行われ、さらにシミュレーションによる性能評価を経て回路合成、制御系合成へと進む。動作記述から品質のよいデータパスを自動合成できるとしても、やはり、評価・改良の部分では人手が介入することが予想されるため、動作解析部は必要であると考える。

3．TokioとRTL-Tokio

TokioはInterval Temporal Logic[3]に基礎をおく、離散時間上に定義された論理型言語であり、直観的にはPrologに時相演算子を加えたものになっている。従って、Prolog同様Tokioはアルゴリズムレベルの記述を自由に行える。しかし、自由なTokioの記述とハードウェアとの対応をとるのは大変難しい。そこで、ハードウェアとの対応が比較的に簡単にとれるRTL-Tokioを提案する。

3．1．Tokio

Tokioの詳細は紙面の都合で省略するが、TokioとPrologの違いを簡潔に述べるなら、それは時間の概念の有無であり、時相演算子及び変数の扱いに代表される。

(1)時相演算子

Tokioにはいろいろな時相演算子が定義されているが、すべてchop,nextから定義できるので、ここでは2つについてのみ述べる。

①chop（&&）　任意のインターバルを前半と後半の2つのインターバルに区切る演算子である。”pred1 && pred2”は現在のインターバルを2つに分け、前半でpred1後半でpred2を実行することを表す。

②next（@）　”@pred”は、次のインターバル（次の時刻から始まり、現在のインターバルと同時刻に終了する）でpredを実行することを表す。

(2)Tokio変数

Prologの変数は一度値が決まるとそれを保持し続けるのに対し、Tokioの変数は、各時刻内ではPrologと同様に扱われる一方、時刻が変わると値を変えてよい。Tokio変数には、時刻と共に値が変わるlocal変数と、一度値が決まると時刻の経過と共に値が変わらないglobal変数の2種類がある。global変数は頭に＊がつき全てのpredicateから参照でき、通常はレジスタ、メモリ等に対応する。変数への値の代入には、temporal assignmentと即時代入がある。

①temporal assignment　これはインターバル内で定義さ

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

れ、local変数に対しては " P <- Q "、global変数に対しては " *p <= *q "と記述される。いずれも、現在のインターバルの最初の時刻のQ(*q)の値を、インターバルの最後の時刻にP(*p)に代入する。

②即時代入　これは時刻に対して定義され、local変数に対しては " P = Q "、global変数に対しては " *p := *q "と記述される。

3．2．RTL-Tokio

RTL-TokioはTokioに以下の制限を加えた、レジスタトランスファレベルハードウェア記述言語である。

①バックトラック

Tokioには2種類のバックトラックが存在する。1つはいわゆるPrologのバックトラックであり、もう1つは時間軸方向のバックトラックである。記述対象のハードウェアは決定的に動作するので、RTL-Tokioには "バックトラックをしない" という制限が入る。

前者に対しては、以下のような構文上の制限が入る。

head(Vars) :- localConds,!, ...

localCondsは現在時刻のみで判断できる条件である必要があり、0個以上複数あっても構わない。後にカット"!"があるため、現在時刻に関してのRTL-Tokioの実行は決定的に進む。

後者に対しては、長さの指定が陽にはないインターバルの長さを次のように決める、という制限がはいる。

・インターバルの終了条件が指定されていれば、不定長。

・それ以外の場合、ほかのインターバルを起動しないインターバルの長さをすべて1とする。起動するものは、その起動されたインターバルと長さが同じになるように自身の長さを決めていく。

これは、<-, <= 等の基本動作を時間1で行うと設定することに対応する。

②変数の扱い

レジスタトランスファレベルにおいては、時刻が経過しても値を保持するglobal変数はレジスタ、メモリ等のデータに直接対応させるのが自然である。レジスタ、メモリに対するデータ転送を考えた場合即時代入は不自然であるため、RTL-Tokioでは " *p := *q " の形は許されない。また、local変数も間接的にレジスタ、メモリ等のデータあるいは転送されているデータを表すため、Tokioの変数ではProlog同様リストが許されていたがRTL-Tokioではリストは許さない。

③最小時間の対応

Tokioは離散時間上に定義され、最小時間が存在する。最小時間をハードウェアの何に対応させるかは記述上の自由であり、それ故にTokioは広い記述能力を持ち様々なレベルのハードウェア記述ができるが、RTL-Tokioでは最小時間をサイクルに対応させる。そうすることにより、先に述べた "<-, <= 等の基本動作を時間1で行う" という設定はレジスタ間の転送をサイクル1で行うことに対応し、レジスタトランスファレベルの記述としては自然である。

3．3．TokioからRTL-Tokioへの変換

TokioからRTL-Tokioへの変換は、アルゴリズムの実現法、データ構造の決定等の機能設計と共に段階的に行われる。この変換はシミュレータで動作の確認をしながら設計者が行う。RTL-TokioはTokioのサブセットなので既存のTokioのシミュレータでTokioと同様に実行できる。RTL-Tokioの記述が前節で述べた制限を満たしているか否かのチェックはハードウェアとの対応をとる際（図1の動作解析部）に行われる。

文献[4]の、reductionアルゴリズムをTokio及びRTL-Tokioで記述したものの一部を図2に載せる。

```
%%% *data((_,pointer)) == (low_id,high_id).
%%% *data((_,vertex)) == (index,value,id,mark).

:-op(400,xfy,'..').
'$function' X..index  =  Index  :- X = (Index,Value,Id,Mark).
'$function' X..value  =  Value  :- X = (Index,Value,Id,Mark).
'$function' X..id  =  Id        :- X = (Index,Value,Id,Mark).
'$function' X..mark  =  Mark    :- X = (Index,Value,Id,Mark).

%%%%%% A behaviour description in Tokio %%%%%%
subl(Id) :-
        *data((Id,vertex))..index > *vn,
        !,
        A = *q,
        append(A,[(*data((Id,vertex))..value,1,
                   *data((Id,vertex))..id)],R),
        *q <= R.

%%%%%% A behaviour description in RTL-Tokio %%%%%%
subl(Id) :-
        *data((Id,vertex))..index > *vn,
        !,
        Id <- Id, *q_tail <= *q_tail + 1
        &&
        *q((*q_tail,key)) <= (*data((Id,vertex))..value,1),
        *q((*q_tail,id)) <= *data((Id,vertex))..id,
        *q((*q_tail,pointer)) <= (null,null).
```

図2．Tokio及びRTL-Tokioによる記述例

4．おわりに

レジスタトランスファレベルを記述するTokioとして、RTL-Tokioを設定し、それによるハードウェア記述について述べた。実際にいくつかのハードウェアを記述し、その記述力は確かめてある。今後は、図1のシステムの実装を進め、Tokioによる論理設計支援環境を整備したい。

参考文献

[1]M.Fujita et al.,IEE Proc.Pt.E.,pp283-294,1986

[2]情報処理学会研究会報告,87-DA-40-18

[3]B.Moszkowski:IFIP 6th CHDL,1983

[4]R.E.Bryant:IEEE Trans. on Computer,No.8,1986