Tokio による機能記述開発支援ツール

1F-1

河野真治・中村宏・藤田昌宏＊・田中英彦
東京大学工学部　＊富士通研

1.時相論理型言語Tokio によるＲＴＬレベルの記述

論理回路設計，とくにＣＰＵの設計は，複雑であり，なんらかの形での計算機によるサポートがなければ，不可能となりつつある。とくに，複数段のパイプライン構成をもつ計算機の設計は，困難であり，従来十分なサポート・ツールがなかった。

Tokio は，全体が一つのクロックで同期をとっているとしたときの，ハードウェアの動作を，時区間時相論理(Interval Temporal Logic)〔1〕により種々なレベルで記述することができる言語である。

ここでは，レジスタを含む自由な形式で記述した，ハードウェアのアルゴリズムを，ＲＴＬレベルの適当なハードウェア記述言語に変換できる制限されたTokio 〔2〕の記述まで変換する作業をサポートするシステムについて考察する。(図1)

Tokio の論理は，時間の区間に関する論理であり，主な演算子としては，一つ短い時区間（次の時区間）を表す@，時区間が長さ0であることを示すempty ，時区間の分割を表すchop (&&) がある。

ここでいう，ＲＴＬレベルとは，レジスタを含む決定的な状態遷移図をさす。これと同等なレベルのTokio の記述は，時相論理演算子を含むHorn clause の集合であるTokio の記述のサブセットとなっている。このサブセットの定義は，以下の通りである。

① 同じヘッドを持つclauseは，互いに排他的であり，記述として決定的である。

② 条件節は，現在時刻で確定する必要があるため，時区間の終りに依存するものはかけない。

この二つの条件は，Tokio のclauseが，一つだけカットを持ち，カットの前の部分は，empty，～empty，時相論理演算子を含まない論理式からなっていることを意味する。さらに，③一つのclauseには，一つのchop演算子しか許さない。さらに，④ chop の前の時区間が，長さ1と仮定することにより，DDL〔3〕と同等な記述となる。このTokio のレベルの記述からの変換については，〔2〕に述べられている。

Free Form	*a <= *a+1.
RTL Form	adder(*a,1,A),a(A). a(A) :- empty,1,*a:=A. a(A) :- skip && a(A).
Propositional Form	use_a_at_bus_b.

図1　3種類のTokio による記述

2.Tokio におけるレジスタの取り扱い

Tokio では，レジスタは，時間が変化しても値の変らない変数として表現する。この時に，レジスタは，時相論理の外のものとして扱われる。これは，次の理由のために，レジスタを一つの時相論理変数として表現できないからである。

①レジスタの値を変える時は，特定の時区間であり，その時区間を他の時区間から区別する必要がある。

②レジスタの値が変らない時は，その時刻で，レジスタの値を変える時区間が一つもない時である。

このように，レジスタの性質は，時相論理の中での否定の概念を必要とする。これは，Horn clause を用いるTokio の範囲内では，表現しにくいものである。このレジスタは，前のTokio の標準形まで，プログラムを変換し，レジスタに関して非決定性がなくなって，初めてTokio の時相論理変数で表現できるようになる。この時，レジスタは，そのレジスタが接続されるデータパスと，それに対する制御線で表現される。

3.一般的なTokio の記述からＲＴＬレベルへの変換

Tokio の一般的な記述から，RTL レベルへの変換は，段階的に行われる。Tokio の記述は，非決定的なものも含むので，必ずも，すべての記述がRTL レベルまで落ちるわけではない。

Translation tools between hierarchical hardware descriptions in Tokio
Shinji Kono, Hiroshi Nakamura, Masahiro Fujita†, Hidehiko Tanaka
The university of Tokyo, Faculty of Engineering　†Fujitsu Lab.

①Tokio の chop, empty, length, @以外の演算子を展開する。

②一つのclauseに一つのchopが来るように述語を増やしながら分解する。

ここまでの方法は，Tokio からPrologへのコンパイラが行っていることと同じである。この時に，レジスタへのアクセスは，1クロック内に限定される。

③各clauseが決定的になるようにカットを付加する。このときに，条件節が，時区間の終了時刻に依存しないことが必要である。ここを自動的に行うことはできない。

この条件を満たすプログラムのみが，決定的であり，未来の信号線からの情報を使わずに動作するハードウェアを表現している。

ここまでのTokio の表現は，時区間の長さが1でないことと，レジスタの衝突が考慮されていないこと以外は，通常の状態遷移図と同じようになっている。

4.レジスタ・アクセスの衝突

次に，データパスを決定し，時区間の長さの決定を行う必要がある。基本的には，chopによる時区間の長さを調節し，レジスタ・アクセスの衝突を回避する。（図2）この作業には，命題時相論理による制御回路の合成を使う。まず各レジスタ・アクセスに対してデータパスを割り振る。このデータパスは，バスでもよいし，直結でもよいし，マルチプレクサでもよい。このデータパスの記述についてもTokio により行うことができる。

次にそれぞれのアクセス条件について一つずつ排他制御を表す命題時相論理変数を割り当てる。実際の時区間の決定は，このようにして，もとのプログラムからレジスタ・アクセスを命題時相論理変数に置換えた時相論理式が恒真になるように，排他制御に関する時相論理変数の状態遷移図を合成すればよい。この方法は，既に示されている〔4〕。この状態遷移を生成する制御のプログラムともとのプログラムを合せたものは，時間区間に関しても決定的になったプログラムである。この記述では，レジスタの競合は，決定的な状態遷移図により回避されている。この部分は，自動的に行うことは，可能だが，計算量の点から命題時相論理式自身をインタラクティブに変換することが望ましい。

最後に，時区間の長さが1になるように，新しい述語を生成しながら，記述の論理式を展開すればよい。これで求める標準形が得られた。

5.パイプライン制御の場合

通常，アルゴリズムの記述は，一つの制御の流れにつ

```
(. . . , *a <= 1, . . && . . ),
(. . . . , *a <= *a+1, . . && )
```

図2　レジスタの衝突

いて書かれる。しかし，ハードウェア上では，この動作が例えば，3段のパイプラインで行われる。このパイプラインの記述をTokio で実現するには，もとのアルゴリズムの記述を，時間的にシフトしながら，3つ並列に動作させたものに，置き換えればよい。（図3）あとの変換は，まったく同じ作業となる。この時，計算量は，通常3乗となるが，パイプライン実行にする時には，それぞれのステージの相互作用は，多少とも低いはずなので，（そうでなければ，もともとパイプライン化は困難）それよりは，小さいと期待される。

現在，並列推論エンジンのVector Unifier〔5〕の3段のパイプラインの部分の例題を作成している〔6〕。

```
run:-  stage_1,
       (skip && stage_1),
       (skip && skip && stage_2).
```

図3　パイプライン実行

参考文献

〔1〕8.2,Proc.of the LOgic Prog.Conf.,ICOT,1985
〔2〕情報処理学会第34回全国大会，1f-2,1986
〔3〕T.Uehara,6th Computer Hardware Description Language and their Applications,1983
〔4〕P,Wolper,22nd Annual Symposium on Foundation on Computer Science, Oct, 1981
〔5〕情報処理学会第34回全国大会，4p-4,1987
〔5〕情報処理学会第34回全国大会，2f-2,1987