3S-9

時相論理型言語 Tokio に基づく論理設計支援システム

中井 正弥　中村 宏　河野 真治　田中 英彦
東京大学 工学部

1 はじめに

我々は、時相論理型言語 Tokio[1] を中心とした、論理設計支援システム(図1)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なえるハードウエア記述言語である。

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なわれているが結果の品質は必ずしも十分ではない。そこで、動作記述と構造記述は別々に与え、支援システムでそれらの間の整合性をチェックすることとする。

このシステムにおける設計の流れは、以下の通りである。

まず、設計者は設計したいハードウエアの動作アルゴリズムを Tokio で記述する。シミュレーションなどで動作の確認をしながら記述を詳細化していき、Tokio をハードウエアとの対応が取り易いよう制限した RTL-Tokio[2] のレベルまで落し、レジスタトランスファレベルの記述とする。

設計者は同時にデータパス等の構造の記述を Prolog で行なう。Tokio は Prolog を包含するのでこれも Tokio で書かれることになる。

その後、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をチェックし、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。

本稿では、この動作解析部の構成、処理の流れ、実行例などについて述べる。

2 動作記述中の Tokio 変数と構造記述との関係

Tokio 変数には local 変数と global 変数の二種があり、時刻内では Prolog の変数と同様に扱われる一方、時刻が変わると値を変えて良い。local 変数は時間軸上に展開された、各々が独立な Prolog 変数と考えてよい。* で始まる global 変数は、全ての predicate から参照でき、一度値が決まると次の代入までその値を保持する。これはレジスタやメモリに対応する。

動作記述中でのレジスタやメモリの記述には global 変数を用い、動作記述中と構造記述中ではレジスタ名、メモリ名は設計者が一致させることとする。

Temporal Assignment の対象となっている local 変数(例えば、 A <- *reg1 における A) は何らかの register を意味してはいるが、構造記述中のどの register に対応するかはわからないので、構造との対応は取らないこととする。

設計者が定義した演算と、それと同じ意味の演算を導く Operation Rules も与える。

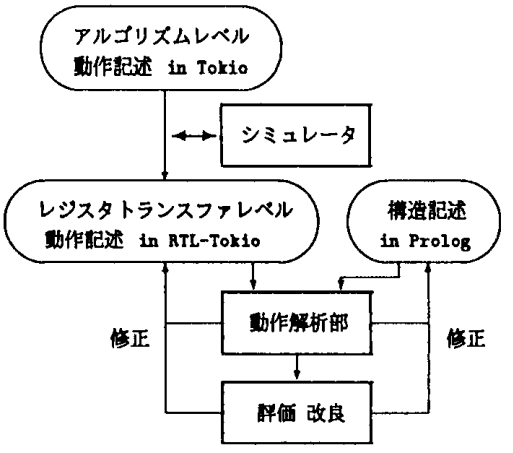

図 1: 論理設計支援システム

3 動作解析部

3.1 構成

動作解析部の構成は、図2に示す通りである。各ツールの機能について述べる。

- Translator
  動作記述をデータ転送表とインターバル遷移表に変換する。
- Facility Checker
  データ転送表中のデータ転送を実現するパスを構造記述中から探し、ファシリテイ使用表を得る。
- Time Tracer
  インタバル遷移表をたどり同時に起こり得るサブインタバルの組を見つけ、ファシリテイ使用表によりそれらでのパスの衝突をチェックする。

各ツールは、Prolog 上に実装されている。

3.2 Translator

動作記述(RTL-Tokio) 中の clause を順に読み込み、各サブインタバルにそれぞれ固有の名前をつけデータ転送を取りだしデータ転送表を得る。また predicate call を分岐条件とともに取りだし、インターバル遷移表を得る。

---

[0]Logic Design Assistance System based on Tokio
Masaya NAKAI Hiroshi NAKAMURA Shinji KONO
Hidehiko TANAKA the University of Tokyo

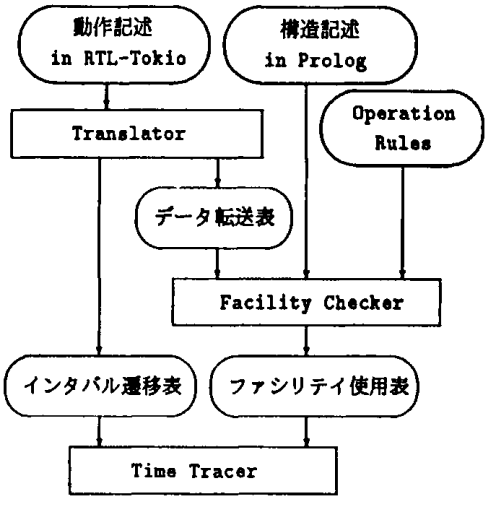

図 2: 動作解析部

3.3 Facility Checker

まず、データ転送表中の個々のデータ転送に対して、local 変数があればその実体の register にまで落とし、(predicate の引数で、call する側から値を渡す local 変数は、場合により異なる register に対応することがある。この場合、どこから呼ばれたかによって場合分けして処理する。) 同じ意味の演算を導き、(例えば、 multi(X,2) なら add(X,X) , shift-left(X)) 構造記述中から演算が可能なファシリテイを見つけ、 Source Register から Facility In へのパスと Facility Out から Destination Register へのパスを探す。

その後サブインタバル内でのデータ転送どうしでのパスの衝突をチェックし、サブインタバルとして、どのファシリテイ、パスを使うかを示したファシリテイ使用表を得る。

データ転送が、複数のパスで実現可能な場合、全ての可能性をチェックし、複数の候補をとっておく。

3.4 Time Tracer

インタバル遷移表を条件分岐を考慮しながら、前からたどり、同時に起こり得るサブインタバルの組を見つける。その後、ファシリテイ使用表を用いて、それらが使うファシリテイがコンフリクトしているかどうかチェックする。

これは、現在製作中である。

3.5 実行例

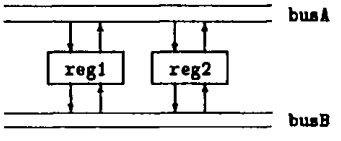

図 3: サンプルデータパス

```
data_equal,[main/0,1,1],[A,[*,reg2]],0.
data_trans,[sub/1,1,1],1,[* reg1,fin],[X,0].
```

図 4: データ転送表の例

```
predicate,[main/0,1],□,[],1.
predicate,[sub/1,1],[X],[],1.
state_call,[main/0,1,1],0,sub,1,[A].
sub_int,[main/0,1,1],□,[],undef.
sub_int,[sub/1,1,1],□,□,1.
```

図 5: インタバル遷移表の例

次に、実行例を示す。次に示すプログラムと図3のデータパスを解析したものである。

```
main :- !, A = *reg2, sub(A).
sub(X) :- !,*reg1 <= X.
```

まず、トランスレータによって、図4のデータ転送表と図5のインタバル遷移表に変換される。その後、次に示す process を経て、図5のファシリテイ使用表を得る。

```
[sub/1,1,1]
data_trans  id:1  * reg1 <- X
  Called by  [[main/0,1,1,0]]
  Register level  * reg1 <- * reg2
    Search path for [reg1,in] <- [reg2,out]
    Success. Path is [[reg2busA,[busA,in]],
                          [busAreg1,[reg1,in]]]
    Success. Path is [[reg2busB,[busB,in]],
                          [busBreg1,[reg1,in]]]
  Success.
```

パスの候補が二つ示されている。

4 おわりに

今後は、このツールの完成を急ぐとともに、このツールをパイプライン化支援などに利用していきたい。

参考文献

[1] M.Fujita et al.,IEE Proc.Pt.E.,pp283-294,1986
[2] 中村他、情処第37回全国大会1U-3,1988

```
facility,[[sub/1,1,1],[[main/0,1,1,0]]],
    [[reg2busA,[busA,in]],[busAreg1,[reg1,in]]].
facility,[[sub/1,1,1],[[main/0,1,1,0]]],
    [[reg2busB,[busB,in]],[busBreg1,[reg1,in]]].
```

図 6: ファシリテイ使用表の例