

RTL-Tokioに基づくパイプライン化支援

3S-10

中村 宏・河野 真治・中井 正弥・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、RTL-Tokioを中心とした論理設計支援を目指している[1]。ここでTokioは時相論理(Interval Temporal Logic)に基づくハードウェア記述言語であり、RTL-TokioはTokioに制限を加えたレジスタトランスファレベル動作記述言語である。Tokio及びRTL-Tokioは時相論理に基づくため、並列性・順序性といった時間に関する記述を厳密に行うことができる。さらにRTL-Tokioは動作解析ツールによって構造記述との無矛盾性のチェックができる[2]。これらの特徴を用いた、RTL-Tokioに基づくパイプライン化支援を現在考えており、ここではその手法について述べる。

2. RTL-Tokioと動作記述

2. 1. TokioとRTL-Tokio

Tokio[3]はInterval Temporal Logic[4]に基礎をおく離散時間上に定義された論理型言語である。Tokio及びRTL-Tokio[5]の詳細は紙面の都合で省略するが、直観的にはPrologに時相演算子を加えたものと考えられる。

- ・(時相演算子を含まない)：これは現在のインターバルで実行されることを表す。
- ・chop(&&)：これは任意のインターバルを前半と後半の2つのインターバルの区切る演算子である。例えば "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。
- ・always(#)：これは現在のインターバルと終了する時刻が同じである全てのサブインターバルで実行することを表す。RTL-TokioはTokioに対し
 - ・バックトラックをしない
 - ・変数がレジスタ、メモリ等のデータに対応する
 - ・離散時間上の最小時間をレジスタ転送に対応させる
 という制限を加えた、レジスタトランスファレベル動作記述言語である。

2. 2. RTL-Tokioによる動作記述

RTL-TokioあるいはTokioでパイプライン化された動作記述を行う場合、大きく分けて、always(#)演算子を用いてそれぞれのパイプが毎時刻起動されることを記述する方法とchop(&&)を用いて、パイプライン化された動作記述を一つのオートマトンとして記述する方法がある。

簡単な例として、長さ1のステージが3つある動作記述、及びその動作を毎時刻起動するというパイプライン

動作記述を下に示す。

① sequentialな記述

```
main :- stage1 && stage2 && stage3.
```

② #(always)を用いたパイプライン記述

```
main :- #(stage1 && stage2 && stage3 && true).
```

③ &&(chop)を用いたパイプライン記述

```
main :- stage1 &&
      ( stage2 && stage3 && true ), main.
```

ここで②と③の記述のtrueは長さ0以上で必ず成功する述語であり、インターバルの長さを合わせるために用いられている。#(always)を用いる記述では、各ステージの長さが一定でない場合(例えば条件分岐等)各ステージ毎にフラグを設ける必要があり、一方、&&(chop)を用いる記述では、条件分岐が存在しても必ずしもフラグを設ける必要はないがその際状態数が増加する事になる。

3. パイプライン化手法

3. 1. パイプライン化

パイプラインには狭義のパイプラインと広義のパイプラインがある[6]。狭義のパイプラインは例えばアレイコンピュータにおけるパイプラインであり、スループットがパイプの長さあるいはアレイの長さに依存しないが、条件分岐等があると実現は難しくなる。一方、広義のパイプラインは、前の入力に対する出力が得られる前に次の入力が入るという意味のパイプラインで、スループットはパイプの長さに依存する。狭義のパイプラインは広義のパイプラインの特別な場合と考えられる。

sequentialな動作記述をパイプライン化する場合、costとperformanceという、相反する二つの制約条件下でのtrade-off pointを求める必要がある。狭義のパイプライン化を対象にした[7]などの研究では、条件分岐の無いデータフローグラフを入力とし、評価関数を用意してcost/performanceの最適解あるいは最適解に近い解を見つけるアルゴリズムが述べられている。狭義のパイプラインでは、例えばN倍のデータパスを用意すればN倍の性能が得られる場合もあり、costとperformanceのtrade-off pointをなんらかの評価関数をもとに決定するのが重要な処理になる。それに対し広義のパイプラインを考えると、データ間の依存関係を見つけ、パイプライン化できるステージを見つけることが主な処理になる。

3. 2. パイプライン化の処理の流れ

現在、広義のパイプライン化を目指したシステムを構