2X-7

Tokioに基づくパイプライン化支援ツールの実装

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

我々は従来より時相論理型言語Tokio[1]を用いたハードウェア機能設計支援を進めてきた[2.3]。Tokio は時相論理に基づいているため、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（以下RTLと省略する）までの様々なレベルの動作記述を行うことができるハードウェア記述言語である。

ここでは、現在構築中の、Tokio に基づいたパイプライン化支援システム[4,5] について述べる。

2. システムの構成

システムの構成は図1のようである。ここで、Tokio は時相論理型言語であり、簡単にはPrologに時相演算子を導入したものといえる。従って一種のプログラミング言語とも考えられ、アルゴリズムレベルの動作記述もできる。しかし、反面Tokio で自由に記述されたものとハードウェアとの対応をとるのは困難なので、RTLの動作記述言語として、Tokio に制限を加えたRTL-Tokio を設定した[6]。RTL-Tokio は従来のTokio に含まれるものである。

3. 処理の流れ

3. 1. パイプライン化

設計者はまずシーケンシャルなアルゴリズムレベル動作記述をTokio で行いシミュレータで動作を確認しながら徐々に詳細化を行いRTLの記述をする。次にシーケンシャルな動作記述から①ステージの切り出し②各ステージの終了条件③各ステージの終了延期条件を考慮してパイプライン化動作記述へ変換する。

簡単な例として、"a+b+(b>>1)"の計算を考えると、シーケンシャルなRTLの記述は図2のようになる。ここで、load,stage1,stage2を切り出してパイプライン化すると図3のようになる。ここで、load-ok,stage1-ok,stage2-ok の3変数は0か1の値を持ち各々のステージが実行中であるか否かを示す制御変数である。またmore(…) の部分はもし次のステージが実行中であるとそのステージの終了をその前のステージが待ち合わせる終了延期条件を意味する。

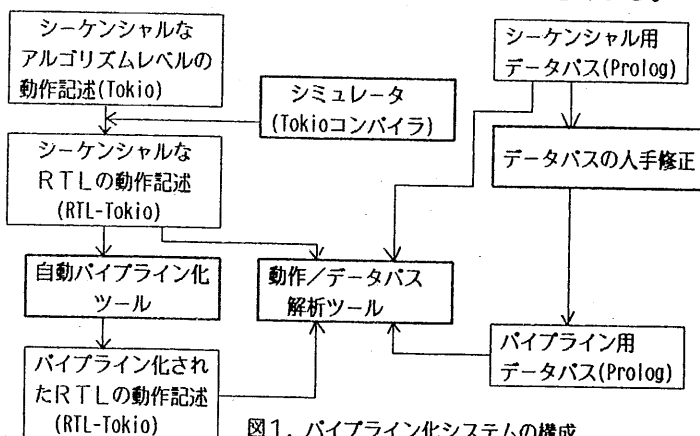

図1. パイプライン化システムの構成

```
start :- *req, !, *ack <= 0
     &&
        main.

main :- load && stage1 && stage2.

load :- *rA <= *input_A,
        *rB <= *input_B.

stage1 :- *tmp1 <= *rA + *rB,
          *tmp2 <= *b >> 1.

stage2 :- *output <= *tmp1 + *tmp2.
```

図2. シーケンシャルなRTL動作記述

```
start :- *req, !, *ack <= 0
     &&
     % if possible always execute load
        keep(if (*load_ok) then main),
        stop(*pipelinestop).

main :- load && stage1 && stage2.

load :- @keep(not *load_ok),
        *rA <= *input_A,
        *rB <= *input_B,
        more(not *stage1_ok),
        fin(*load_ok).

stage1 :-
        @keep(not *stage1_ok),
        *tmp1 <= *rA + *rB,
        *tmp2 <= (*b >> 1),
        more(not *stage2_ok),
        fin(*stage1_ok).

stage2 :-
        @keep(not *stage2_ok),
        *output <= *tmp1 + *tmp2,
        fin(*stage2_ok).
```

図3. パイプライン化されたRTL動作記述

Implementation of a tool for pipeline design based on Tokio
Hiroshi NAKAMURA, *Masahiro FUJITA, Shinji KONO, Hidehiko TANAKA
University of Tokyo, *FUJITSU LABORATORIES LTD.

3．2．動作／データパス解析

ここではRTL-Tokio で記述されたＲＴＬの動作記述、及びPrologで記述されたデータパス記述を受けとり、与えられたデータパス上でＲＴＬの動作記述が実現可能かを解析する。処理の流れは図４のようであり[6]、Prologで実装されている。

①動作記述－＞状態遷移表

ここでは、RTL-Tokio の記述を解析し、状態(Tokioのインターバルに相当し、パイプラインにおけるステージに当たる）の遷移表を作成する。その際、各インターバルの長さは以下の規則で決定される。

・length指定があればその通り

・ほかのインターバルを呼ばず、かつkeep,@演算子を含まないインターバルの長さは１

・"," で結ばれたインターバルの長さは同じ、"&&" で結ばれたインターバルの長さは単純に足算し、ボトムアップに決めていく

・上の２つの規則で矛盾が生じる場合はインターバルの長さを延ばす方向で修正する

②ファシリティ使用表への変換

ここでは、各インターバルでのファシリティの使用状況を表にする。動作記述中のあるデータに対する処理を実現するファシリティの組が複数あったり、データ転送を実現するデータパスが複数ある場合の管理もする必要がある。

③解析部

あるファシリティに対し複数のデータ転送がある場合が問題であり、それは以下の４通りに分類される。

	インターバル	入力パス	実現
a.	同じ	同じ	可能
b.	同じ	異なる	不可能
c.	異なる	同じ	可能
d.	異なる	異なる	？？？

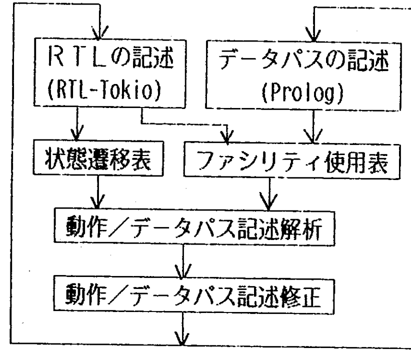

図4．動作／データパス解析部

a,c の場合にはデータの衝突は無い。b の場合は明らかにデータが衝突する。d の場合は、その異なるインターバルが同時に起こり得るかを、状態遷移表を用いて時刻を遡ることにより検出する。

④修正部

データの衝突が起こる場合の修正には、動作記述の修正とデータパスの修正がある。後者はデータが衝突しているファシリティを増やすものであり、容易であるが、ハードウェア量の増大をもたらす。ここでは前者の修正について先程の例を用いて述べる。

図３の動作記述を実現するのに、仮にadder が１つしかないという制約のあるデータパスが与えられ、その制約が変更できないとする。すると、ファシリティ使用表からstage1とstage2でadder を共有することが解り、この２つのステージは同時に起こり得るので、ここでデータの衝突が起こる。従ってstage1とstage2が同時に生起しないように動作記述を変更すれば良い。この例のように、２つのステージ各々の処理に要する時間が１と決まっている場合は、パイプラインを２クロックごとに起動すれば良いことがわかる。従って、load, stage1, stage2の動作記述の部分を図５のように修正すれば良いことになる。しかし、一般には各ステージの処理に要する時間は変化する。その場合には競合しているステージをまとめて１つのステージと考え、そのステージを実行中には新たにそのステージに入ることを禁止する、というように修正する。stage1とstage2をまとめると同じ部分は図６のように修正される。

```
load :- @keep(not *load_ok),
        length(2),
        ( true,length(1) &&
        *rA <= *input_A,
        *rB <= *input_B ),
        more(not *stage1_ok),
        fin(*load_ok).

stage1 :- length(2),
        @keep(not *stage1_ok),
        ( true ,length(1) &&
        *tmp1 <= *rA + *rB,
        *tmp2 <= (*b >> 1) ),
        more(not *stage2_ok),
        fin(*stage1_ok).

stage2 :- length(2),
        @keep(not *stage2_ok),
        ( *output <= *tmp1 + *tmp2
        && true, length(1) ),
        fin(*stage2_ok).
```

図5．動作記述の修正（１）

```
load :- @keep(not *load_ok),
        *rA <= *input_A,
        *rB <= *input_B,
        more(not *stage_ok),
        fin(*load_ok).

stage :-
        @keep(not *stage_ok),
        ( *tmp1 <= *rA + *rB,
          *tmp2 <= (*b >> 1)
        &&
        *output <= *tmp1 + *tmp2),
        fin(*stage_ok).
```

図6．動作記述の修正（２）

4．おわりに

ＲＴＬの動作記述をRTL-Tokio で行うパイプライン化支援システムについて述べた。

データ衝突がある場合は、動作記述及びデータパス記述のどちらを修正するにしてもその修正がパイプラインの性能に大きな影響を与える場合があるので、複数の修正候補をあげその中からシミュレーションで最適と思われるものを選ぶことが考えられる。

参考文献

[1]IEE Proc. E. Vol.133,No.5, pp283-294, 1986

[2]S.Kono et al., Proceedings of LPC'85 Springer-Verlag

[3]H.Nakamura et al.,IFIP VLSI'87

[4] 第35回情報処理学会全国大会,5F-1

[5] 第35回情報処理学会全国大会,5F-2

[6] 情報処理学会研究会報告,87-DA-40-18