6Q-10

# 時相論理型言語Ｔｏｋｉｏによる論理回路検証系の評価

中村　宏・藤田　昌宏＊・田中　英彦
東大工学部　＊富士通研

## 1．はじめに

近年の素子技術や実装技術の進歩に伴い、ハードウェアシステムは大規模複雑なものとなってきており、論理設計段階での支援が重要になってきている。我々は時相論理（Temporal Logic）［１］を用いた論理設計手法を提案し、論理型言語Prolog を用いた実験システム［２］を作成してきたが、スピード及びメモリの点で必ずしも満足のいくものではなかった。今回、論理式をカバーで表現することによる手続き型言語による実装を試みた。ここではそのシステムの評価を簡単に報告する。

## 2．システムの構成

このシステムは、図１のような構成をしている。ＨＳＬはハードウェア記述言語、Tokio［３，４］は時相論理に基づく言語であり、設計をＨＳＬで仕様をTokioで記述する。検証すべきことは、設計が仕様を満たすことであり、このシステムではそれを状態遷移図上で検証する［５］。カバー表現は論理式をコンパクトに表現する表現法である。斜線部はProlog でそれ以外はＣ及びYacc，Lexを用いて実装する。

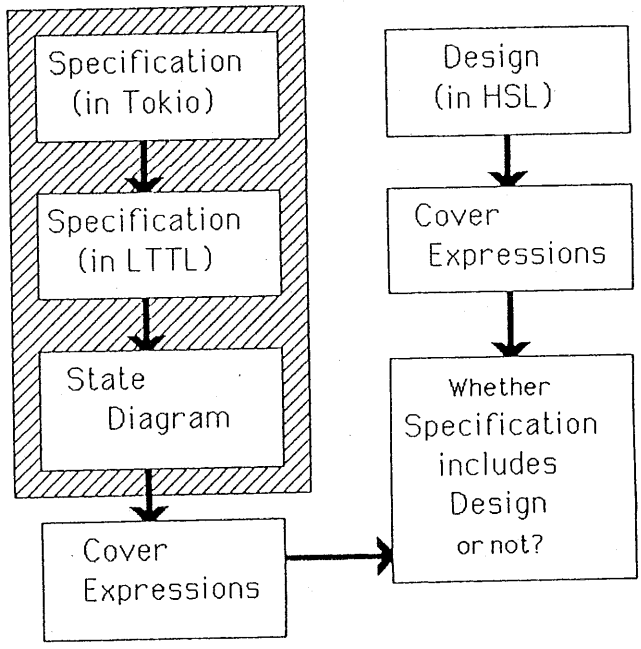


図１　検証システムの構成図

## 3．カバー表現［６］

カバー表現は論理式を整数として表現するコンパクトな表現法であり、システムを実装する際のメモリー効率の向上に役立つ。

・キューブとカバー

キューブは１つの積項を、カバーは積和形で表された論理式を表現する。n 入力m 出力の１つの積項p に対し、大きさが２ビットの要素n 個と大きさ１ビットの要素をm 個を持つベクタがキューブである。そして与えられた論理式を積和形に変形しキューブの集合として表現したものがカバーである。

例　$f1 = AB + \overline{B}C + AC$

$f2 = \overline{B}C + \overline{C}\overline{D}$

という論理式は、

| A | B | C | D | f1 | f2 | |
|---|---|---|---|---|---|---|
| 01 | 01 | 11 | 11 | 1 | 0 | $AB$ |
| 11 | 10 | 01 | 11 | 1 | 1 | $\overline{B}C$ |
| 01 | 11 | 01 | 11 | 1 | 0 | $AC$ |
| 11 | 11 | 10 | 10 | 0 | 1 | $\overline{C}\overline{D}$ |

という４つのキューブの集合からなるカバーで表現される。

カバー表現を用いると論理式の積が各要素ごとのビット積で計算できるので処理が高速化できる。

・onカバーとoff カバー

ある出力変数に対してそれを１とするような入力変数の論理式をonカバーといい、０とするような入力変数の論理式をoff カバーという。

---

Evaluation of Logic Design Verifier based on Tokio

Hiroshi NAKAMURA, Masahiro FUJITA*, Hidehiko TANAKA
University of Tokyo ,*Fujitsu Labolatory

## 4．HSLからカバー表現への変換部

図２の回路を例に説明する。

onカバーとoff カバーは全ての出力変数に対して計算されるので、図２の回路では”Infin のビット幅＋１（Hear）”個の出力変数に対して計算されることになる。しかし、例えば”Call→◇Hear”という仕様を検証する時には、Infin という出力変数に対するカバーは不要であり、Hearに関するカバーのみが必要である。ある特定の出力変数に対するカバーのみを求めればよい時には、その出力変数に関係するゲートのみに回路を絞り込んで計算すればよく、この例ではand3とand4のゲートが絞り込みによって除外される。

このシステムのHSLからカバー表現への変換部は、この絞り込み手法を実現しているため、無駄な計算をせずに済み、スピード、メモリ所要の向上に役立つ。

## 5．評価

現在図１の構成図においてHSL→カバー表現の変換部と２つのカバー表現から設計が仕様を満たすことを調べる部分ができており、前者は約3600行、後者は約400 行である。後者については、仕様記述に対応するカバー表現を手で作成することによって、動作の確認をしてある。

ここでは以前に作成したProlog によるシステムと今回のC言語によるシステムで、設計記述を変換する部分について比較・評価をする。

例題は図２の回路であり、Message からInfin までのビット幅を１，４，８の３通りにして絞り込みを行なわなかった場合の結果が表１であり、ビット幅を８に設定してHearについて絞り込んだ場合の結果が表２である。測定はVAX11／730 上で行なった。

システムの構成の違いから、今回実装したC言語による変換部は出力変数に対する入力変数の論理式（カバー）を計算して出力するのに対し、Prolog による変換部は出力変数に対する入力変数の論理式を導出するデータを出力するのみであり、処理内容は前者の方が多いが、処理時間はオーダーのレベルで短くなっている。

表１と表２を比較して絞り込みの効果がProlog によるシステムの方が顕著である理由は、Prolog による実装ではビット幅を展開する前に絞り込みを行なうのに対し、C言語による今回の実装ではビット幅を展開してから絞り込みを行なうからである。

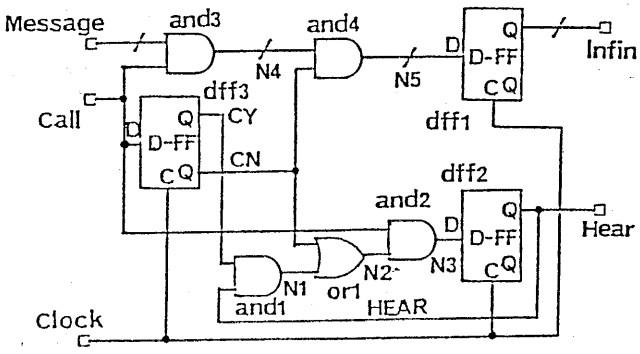


図２　回路例

| | CPU time (VAX11/730) | |
|---|---|---|
| ビット幅 | Prolog | C |
| 1 | 34．6sec | 0．9sec |
| 4 | 102．3sec | 4．2sec |
| 8 | 246．8sec | 7．2sec |

表１　評価結果（絞り込まない場合）

| | CPU time (VAX11/730) | |
|---|---|---|
| ビット幅 | Prolog | C |
| 8 | 18．1sec | 3．5sec |

表２　評価結果（Hearについて絞り込み）

## 6．参考文献


［１］B．Moszkowski：”Reasoning about Digital Circuits”，Stanford Univ. Report STAN_CS_83_ 970 （1985）

［２］M．Fujita:”Logic Design Assistance with Temporal Logic”，IFIP 7th CHDL

［３］青柳：”Tokioかける言語”，Logic Programmimg Conference 8.1 （1985）

［４］河野：”時相論理型言語Tokioの実装”，Logic Programming Conference 8.2 （1985）

［５］中村：情報処理学会第32回全国大会,5U－2

［６］藤田：”時相論理を用いた論理設計検証およびテスト生成のカバー表現に基づく高速化”，電子通信学会　FTS研究会　Nov．（1985）