社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2005－SLDM－118（8）
2005／1／25

分割統治法による形式的機能検証のための インタフェイスプロトコルに対するアサーションの分割

松島　広直†　　北嶋　　暁†

† 大阪電気通信大学大学院工学研究科情報工学専攻　〒572–8530 大阪府寝屋川市初町 18–8
E-mail: †{hironao,kitajima}@sldlab.osakac.ac.jp

あらまし　本稿では，ディジタルシステム設計の RT (Register Transfer) レベルにおいて，BCA (Bus Cycle Accurate) レベルでのインタフェースプロトコルに対するアサーション群をコンポーネント単位のアサーション群に分割し，それを用いて各コンポーネントの形式的な機能検証を行う方法を提案する．本手法により，段階的設計における検証をコンポーネント単位の分割統治法により行うための，各コンポーネントに対するインタフェイスプロトコルの制約が得られる．提案手法では，コンポーネントの入出力関係に着目し，アサーション群を分割する．例題について本手法を適用し，コンポーネント単位でのインタフェイスプロトコルの制約に対して，満たすべき性質が検証可能であることが確認できた．

キーワード　形式的検証，アサーション，インタフェイスプロトコル，バスサイクルアキュレートレベル，分割統治法

A Dividing Technique of Assertions for an Interface Protocol used in a Divide and Conquer Approach of Formal Verification

Hironao MATSUSHIMA† and Akira KITAJIMA†

† Division of Information and Computer Sciences, Graduate School of Engineering,
Osaka Electro-Communication University　Hatsu-cho 18–8, Neyagawa-shi, Osaka, 572–8530 Japan
E-mail: †{hironao,kitajima}@sldlab.osakac.ac.jp

Abstract　In this paper, a dividing technique of assertions for an interface protocol in a digital system is proposed for divide and conquer approach of formal verification. In the proposed technique, assertions in the description at the bus cycle accurate level of a system are divided into the assertions that are used in the descriptions of components at RT-level for each component. Input-output relation among components in the system is considered for the division. All components in a sample producer-consumer system was verified with assertions divided by the proposed technique.

Key words　formal verification, assertion, interface protocol, bus cycle accurate level, divide and conquer approach

1. はじめに

近年，半導体集積回路製造技術の進歩により設計規模と設計の複雑さが爆発的に拡大し，それにより，設計したディジタルシステムが正しいことの機能検証が十分に行えなくなってきている．このような大規模回路設計を行う際の設計検証のコストは今や全設計工程の 5-8 割にも達していると言われる．設計規模の増大に対しては，従来の RT (Register Transfer) レベルでの設計よりも抽象レベルの高い，システムレベル設計が導入されつつある．[2] このため，抽象レベルの高い段階から段階的に検証が行える方法が望まれる．

RT レベルやそれより抽象レベルの高い BCA (Bus Cycle Accurate) レベルの検証に関して，アサーションを利用した検証手法が注目されている．[8] ここで，アサーションとは，設計された回路が満たすべき性質を指す．アサーションの記述は，命題論理式やそれを時系列上の変化しとて表した表現を用いて行う．アサーション記述言語として OVL [8], *e* 言語 [1], PSL [3], [7], SVA [4], [5] などがある．この，アサーションを用いた，アサーションベース検証では，アサーションとして回路の満たすべき性質や入力の制約などを記述し，その記述に基づいて，シミュレーションや形式的な検証を行う．シミュレーションによる方法ではシステムに起こり得る場合の網羅性を保証することは困難なため，実用的な形式的検証法が望まれている．従来の形式的検証法では，システム中の各コンポーネントを一括して扱う

ため, 検証可能な回路規模に限界があった. 抽象化を行い本質的な部分のみを検証する方法も考案されてはいるが, それでも実用的なシステムに対しては実用的な時間では検証が困難である. [10]

本稿では, 分割統治法による検証が可能になるように, BCA レベルでのアサーション群を各コンポーネントに対して分割する方法を提案する. システムを詳細化する過程で, システム全体の検証から各コンポーネント単位での検証に分けるという, 分割統治法による検証が可能になれば, 形式的検証の効率が飛躍的に向上することが期待される. 提案手法では, アサーションに着目し, BCA レベルでのシステム内におけるインタフェイスプロトコルのアサーションを, 各コンポーネントに対する入力制約および出力に関して満たすべき性質に分割する. 各コンポーネントについて, 満たすべき性質を満たすことを示すことができれば, 他のコンポーネントにおける入力制約を満たすことも保証することができ, その後の設計においてもコンポーネント単位での検証に用いることが可能となる.

提案手法を例題について適用し, 本手法により検証が行えることを確認した. 用いた例題は FIFO を用いた生産者消費者モデルである. 三つのコンポーネントから成るシステムに対して, システム内のコンポーネント間におけるインタフェイスプロトコルに関するアサーションを記述し, それを各コンポーネントに対して分割した. それを用いて, 検証器として SMV [6], [9] を用いてコンポーネントごとの検証を行い, 必要な検証項目が成立することが示せた. これにより, インタフェイスプロトコルについて, コンポーネントごとの入力制約や出力プロパティによる切り分けが可能であることの見通しを得た.

本稿の構成は次のとおりである. 2. ではまず分割統治法による検証法の基本的な考え方について述べる. 3. で分割統治法に必要となるアサーションをコンポーネント単位で分割する方法について述べる. 4. で例題に対して行った評価の内容とその結果について述べる. 5. で, 本手法を適用可能なクラスや分割統治法において他に必要となる検証項目について考察する. 6. で本稿をまとめる.

2. 分割統治法による検証

本章では, まず本研究の対象とする設計抽象レベルについて述べ, 次に, 各抽象レベル間で設計が正しいことを保証するための検証項目について述べる.

2.1 対象とする設計抽象レベル

本研究で対象とする設計抽象レベルは, UTF (Un-Timed Function) レベル・BCA レベル・RT レベルである. 図 1 に示すように, 設計は UTF から BCA レベル・RT レベルへと段階的に行うこととする. 以下, 各抽象レベルの具体的な内容について順に述べる.

2.1.1 UTF レベル

UTF レベルでは, システムを並行プロセスモデルとして表現し, 各プロセス間の通信では, 送られたデータはすぐに送信先へ到着することとする. ここでは, 一つのデータ転送を表す動作をトランザクションと呼ぶ.

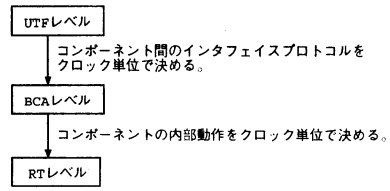

図 1 LSI 設計工程における設計抽象レベル

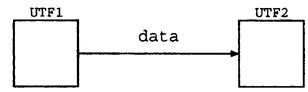

図 2 UTF レベルにおけるプロセス間のデータ送信の例

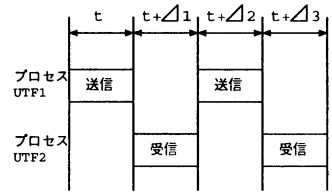

図 3 UTF レベルにおけるデータ転送を表すタイムチャートの例

プロセス間のデータ送信を行うシステムの例を図 2 に示す. この例では, プロセス UTF1 からプロセス UTF2 へデータを送信する. UTF1 から UTF2 へデータはすぐに到達する. 図 2 の例におけるデータ転送のタイムチャートを図 3 に示す. この例では, 一つの動作を行うために必要とされる仮想的な時間 (デルタ時間) により, 転送の順序が分かるようにしている.

2.1.2 BCA レベル

BCA レベルでは, UTF モデル各プロセスをコンポーネントとし, コンポーネント間の各トランザクションを実現するインタフェイスプロトコルをクロックサイクル単位で定める. 具体的には, まず, コンポーネント間でのデータや制御信号のやりとりのためのポートを定義し, それらの間の接続関係を定める. 次に, 各コンポーネントについて, データのやり取りを各ポートの入出力として表現する. トランザクションに関係しない内部動作については UTF レベルと同一である.

インタフェイスプロトコルには, 入出力のタイミングに関する制約が存在する. ここでは, この制約のことを入出力制約と呼ぶ. これは, 各ポート間でやり取りされる信号値の時間的な順序関係で表すことができる.

図 2 に対応する, BCA レベルのシステムの例を図 4 に示す. この例では, 各プロセスを実現するコンポーネントが導入され, 各コンポーネントのポートおよび接続関係が決められている. データ転送は, このポートを介して信号をやり取りすることにより行う. この際のタイムチャートの例を図 5 に示す. 一つのトランザクションがいくつかのクロックを経て行われている.

2.1.3 RT レベル

RT レベルでは, BCA レベルで定めたインタフェイスプロトコルのタイミングを保ちつつ, 各コンポーネントの内部動作をクロック単位で具体的に定める. 本レベルは論理合成可能なレ

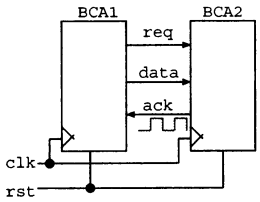

図 4　BCA レベルにおけるコンポーネント間のデータ通信の例

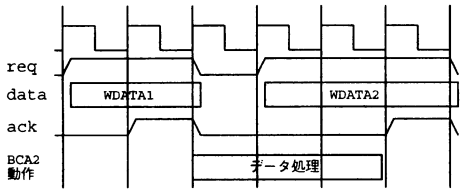

図 5　BCA レベルにおけるデータ転送を表すタイムチャートの例

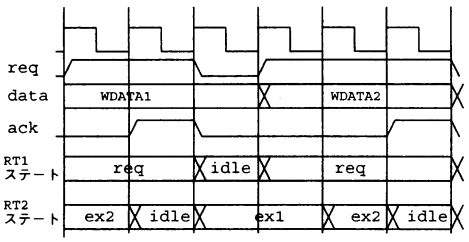

図 6　RT レベルにおけるデータ転送を表すタイムチャートの例

ベルでもある.

図 5 のモデルを RT レベルに詳細化した例を図 6 に示す. 一つのトランザクションを行うための手順は BCA レベルと変わらないが, モジュールの内部動作は各クロックに対して定まっている.

2.2 UTF レベル-BCA レベル間の検証

UTF レベルの設計記述が正しいという前提のもとで BCA レベルの設計記述が正しいことを示すためには, 以下の二つの条件を満たすことを示せばよい.

条件 A1. BCA レベル記述において, 入出力制約が満たされた場合に, UTF レベル記述での各トランザクションに対応するデータ転送が正しく行われる.

条件 A2. BCA レベル記述における任意の内部動作に対し, 対応する UTF レベル記述における内部動作が存在し, かつその内容が等しい.

2.3 BCA レベル-RT レベル間の検証

BCA レベルの設計記述が正しいという前提のもとで RT レベルの設計記述が正しいことを示すためには, 以下の二つの条件を満たすことを示せばよい.

条件 B1. BCA レベルの入出力制約が RT レベルでも満たされる.

条件 B2. RT レベル記述における任意の内部動作に対し, 対応する BCA レベル記述における内部動作が存在し, かつその内容が等しい.

この条件を満たすには, 任意のコンポーネントに対し, 以下の二つの条件を満たすことを示せばよい.

条件 C1. 入力制約が成り立つもとで, BCA レベル-RT レベル間で内部動作が等しい.

条件 C2. そのコンポーネントの入出力制約が常に成り立つ.

本条件が成立することを示す方法があれば, 分割統治法による検証が可能となる.

3. 提案手法

本章では, 前章で述べた BCA レベル-RT レベル間のインタフェイス検証において, アサーションを利用する方法を提案する. はじめに説明に使用する例題について述べ, 次にアサーション記述の内容について述べる. 最後にそれらをコンポーネントごとに分割する方法について述べる.

3.1 例　　題

例題として, 生産者消費者モデルを用いる. ここで, 生産者消費者モデルとは, 並行プロセスの基本的な要素を備えた一対一の単方向通信を行うモデルである. ブロック図を図 7 に示す. このモデルは, データを生成・送信する producer, データを一時的に保存しておく FIFO, データを FIFO から受け取る consumer から構成される. producer は consumer の状態に関係なくデータを送信することができ, consumer は producer の状態に関係なくデータを受信することができる. システムの入出力として, send・sent および receive・received がある. これらを用いて, 送信および受信のタイミングがシステム外部から指示される.

以下, 各モジュールの詳細な動作を順に示す.

3.1.1 producer

producer は有限状態機械として表現できる. 状態遷移図を図 8 に示す. producer は IDLE・WRITE・WDONE の 3 状態をとる. 図では, 各遷移における遷移条件および出力を '/' の前後に記述している. '-' は, 条件がないあるいは出力値が変化しないことを表す.

producer の入出力である各制御信号の意味を以下に示す.

- send…producer にデータの送信を要請する入力信号である. 出力 sent をアサートすることでネゲートされる.
- sent…producer が, 送信が終了したことを示す出力信号である.
- wr…データを送信することを知らせる出力信号である.
- full…送信が許可されていることを知らせる入力信号である. full 信号がアサートされているときは書き込みは行わない.

3.1.2 consumer

consumer は有限状態機械として表現できる. consumer の状態遷移図を図 9 に示す. consumer は IDLE・RE・READ・RDONE の 4 状態をとる.

consumer の入出力である各制御信号の意味を以下に示す.

- receive…consumer にデータの取り込みを要請する入力信号である. 出力 received をアサートすることでネゲートされる.
- received…consumer が, データの取り込みが終了したこ

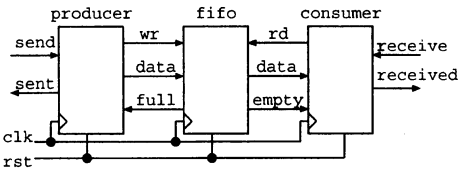

図 7　生産者消費者モデルの各コンポーネントとその接続関係

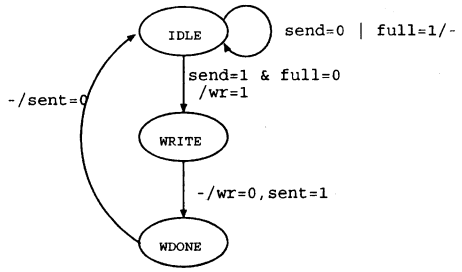

図 8　producer の状態遷移図

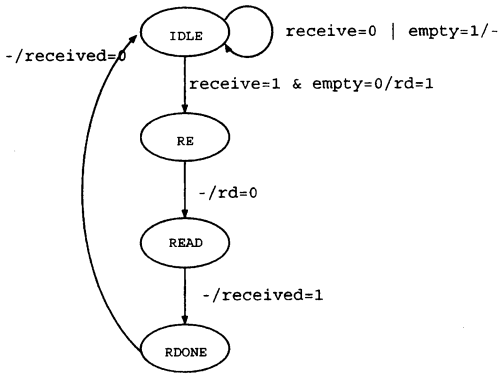

図 9　consumer の状態遷移図

とを示す出力信号である.

- rd…データの送信を要請する出力信号である.
- empty…データの取り込みが許可されていることを示す入力信号である. empty がアサートされているときはデータの取り込みは行わない.

3.1.3　FIFO

FIFO に対して, データの書き込みおよび読み出しが可能である.

FIFO は信号 wr が 1 になるとクロックに同期してデータを取り込む. 書き込まれた際に FIFO が一杯になれば信号 full を 1 にする.

読み出しの場合は信号 rd がアサートされればデータを送信する. データ送信後 FIFO が空になれば信号 empty を 1 にする.

3.2　BCA レベルにおける検証

3.2.1　アサーションの内容

アサーションとして, コンポーネント間の入出力信号を用いた入出力制約を記述する. また, システムに対する入出力制約も記述する.

アサーションとして表現できる内容については, 一般的なアサーション記述言語に用いられる概念で表現できる範囲を考える. PSL や SVA などでは, ブール式やシーケンシャル式 (ブール式とブール式の時相的関係) を用いて, これらの性質を表現する. 以下では, 例の記述の際に CTL 表現を用いるが, これらのアサーション記述言語でも表現できる範囲を想定している.

ここで, 制約の内容によっては, 過去の入出力値に依存する場合が起こり得る. 本手法では, 変数の概念を用い, ある時点の入出力値を変数を介して参照できることとする.

変数を用いたアサーションを使用する例として, FIFO に書き込まれたデータが正しい順番で出力されているか, データが破損していないかなどのチェックがあげられる.

3.2.2　例題でのアサーション記述

例題の場合, インタフェイスプロトコルに関するアサーションは次のとおりである.

($P_{BCA}1$) AG(full → AX !wr)
full がアサートされているとき, wr はアサートされない.
($P_{BCA}2$) AG(full → AF !full)
full 状態はいつか解除される.
($P_{BCA}3$) AG (!send → AF send)
send が必ずどこかでアサートされる.
($P_{BCA}4$) AG(send → AF sent)
send がアサートされたら, いつか sent がアサートされる.
($P_{BCA}5$) AG(sent → AX !send)
sent がアサートされたら send はネゲートされる.
($P_{BCA}6$) AG(empty → AX!rd)
empty がアサートされているとき, rd はアサートされない.
($P_{BCA}7$) AG(empty → AF !empty)
empty 状態はいつか解除される.
($P_{BCA}8$) AG(!receive → AF receive)
receive はどこかで必ずアサートされる.
($P_{BCA}9$) AG(receive → AF received)
receive がアサートされたらいつか received がアサートされる.
($P_{BCA}10$) AG(received → AX !receive)
received がアサートされたら receive はネゲートされる.

システムの外部からの入力制約を下記に示す.

($C_{sys}1$) send はネゲートされていたら必ずどこかでアサートされる.
($C_{sys}2$) send がアサートされたら sent がアサートされるまで send は値を保持する.
($C_{sys}3$) receive はネゲートされていたら receive は必ずどこかでアサートされる.
($C_{sys}4$) receive がアサートされたら received がアサートされるまで receive は値を保持する.

3.2.3　検証内容

この段階での検証内容としては, BCA レベルで各トランザクションが入力制約を満たす任意の時点で正しく実行されることである. この例題の場合は, producer で送信されたデータが consumer に必ず到達することを示すことになる. 今回はアサーションの分割に着目しているので, この部分の検証については

取り上げない.

3.3 アサーションの分割による検証法

各コンポーネント C について, BCA レベルのアサーション群を, 入力制約 $F_{BCA}(C)$ と出力プロパティ $P_{BCA}(C)$ に分割する.

分割は次の基準で行う.

- あるコンポーネントの入力が他のコンポーネントの振舞いに依存する場合, その外部コンポーネントの振舞いを入力制約とする.
- あるコンポーネントの出力が他のコンポーネントの振舞いに影響する場合, その振舞いを出力プロパティとする.
- あるコンポーネントの出力に依存するコンポーネントがない場合, その制約は RT レベルにおいてそのままそのコンポーネントの制約として使用する.

各コンポーネント C について, 次の条件を満たしていれば, インタフェイスプロトコルが正しく実現されていると言える.

条件 D. 入力制約 $F_{BCA}(C)$ を満たす任意の入力系列に対し, 出力プロパティ $P_{BCA}(C)$ が成り立つ.

3.3.1 例題の RT レベルにおけるアサーションの分割

BCA レベルのアサーションから分割基準に従って導き出した, RT レベルの各コンポーネントのためのアサーション入力制約, 出力プロパティを下記に示す.

producer に対するアサーション

(入力制約 pi1) full は必ずどこかでネゲートされる.(必ず書き込み可能になる) ($P_{BCA}2$ より導出)

(入力制約 pi2) send は必ずどこかでアサートされる.(必ず書き込みの要請がある)($P_{BCA}3$ より導出)

(出力プロパティ po1) AG(full → AX !wr)($P_{BCA}1$ より導出)

(出力プロパティ po2) AG(send → AF sent)($P_{BCA}4$ より導出)

(出力プロパティ po3) AG(sent → AX !send)($P_{BCA}5$ より導出)

consumer に対するアサーション

(入力制約 ci1) empty は必ずどこかでネゲートされる。(必ず読み込み可能になる)($P_{BCA}7$ より導出)

(入力制約 ci2) receive は必ずどこかでアサートされる.(必ず読み込みの要請がある)($P_{BCA}8$ より導出)

(出力プロパティ co1) SPEC AG(empty → AX !rd)($P_{BCA}6$ より導出)

(出力プロパティ co2) SPEC AG(receive → AF received)($P_{BCA}9$ より導出)

(出力プロパティ co3) SPEC AG(received → AX !receive)($P_{BCA}10$ より導出)

FIFO に対するアサーション

(入力制約 fi1) full をアサートしているとき, wr はアサートされない. ($P_{BCA}1$ より導出)

(入力制約 fi2) empty をアサートしているとき, rd はアサートされない. ($P_{BCA}6$ より導出)

BCA レベルのアサーションから RT レベルのコンポーネントのアサーションを導出する例を下記に示す.

(性質 1) AG(full → !wr) は, producer-FIFO 間で満たされなければならない性質である.

FIFO の入力である wr は producer の振舞いに依存するため, (性質 1) は producer の出力プロパティとなり, FIFO の入力制約となる. よって (出力プロパティ po1) と (入力制約 fi1) が導き出される. 入力制約に対する出力制約が満たされれば, 仮定である入力制約も成り立つ.

4. 評　　価

提案手法の有効性を確認するために検証器 SMV [6], [9] による検証実験を行った. 実験用のモデルには 3. 章の例題を用いた.

4.1 実験方法

実験では, まず BCA レベルで満たされるべき性質がシステム全体で検証を行ったときに満たされるかを確認する. 次に RT レベルで, 分割したアサーションが各コンポーネントに対して満たされるかを確認する.

入力制約は SMV の FAIRNESS 構文または外部コンポーネントとして記述し, 出力制約は SPEC 構文で記述した. 例えば producer の入力制約 pi1 と出力プロパティ po1 は次のように記述される.

(入力制約 pi1) FAIRNESS !full

(出力プロパティ po2) SPEC AG(full → !wr)

4.2 実験結果

上記の方法で検証を行なった結果, すべての性質がコンポーネントごとに満たされることが確認できた. なお, 検証に要した実行時間は Intel Pentium4 Processor 2.53GHz で合計 1 秒未満であった.

5. 考　　察

本章では, 提案手法をもとに, 分割統治法による検証に必要な項目について考察する. まず, 提案手法の適用範囲について考察する. 次に, 分割統治法による検証に必要な検証内容について考察する.

5.1 提案手法の適用範囲

提案手法では, 一対一のコンポーネント間通信を対象としている. 現状では簡単な例題に対して適用しただけであるから, 適用可能な範囲を明確にする必要がある. インタフェイスプロトコルとして記述する内容をある程度限定すれば, 分割自体を自動化できると考えられる.

近年, IP (Intellectual Property) を用いた設計において, インタフェイス部のプロトコルの間違いなどによる不具合が問題となっている場合がある. これに対しても, IP の設計記述中にインタフェイスプロトコルに対するアサーションを記述しておくことにより, 他のコンポーネントとの整合性などを確認することが可能となる. このような使い方に対しても, 本手法を拡張することで対応できると考えられる.

システム設計では通常バスを用いた通信が行われるが, 提案手法ではバスに対して必要な検証項目を明確には示していない. ただし, バス自体を一つのコンポーネントとしてとらえれば, バスに接続されている各コンポーネントと一対一の通信を行うことができる. 今後, バスを用いたデータ転送個有の問題につい

て検討する必要がある.

5.2 分割統治法による検証のための検証項目

分割統治法によりコンポーネント単位で検証が進められるようにするには, 各コンポーネントについて, 入力制約・動作内容・出力プロパティが決まればよい.

BCA レベルでは, コンポーネント間のインタフェイスプロトコルが決まるため, 各コンポーネントの入力制約は以降の抽象レベルでも変わらない. ただし, 入力制約として, アサーション記述を用いた場合では, そのアサーション記述がインタフェイスプロトコルの仕様をすべて表現しているかどうかが問題となり得る. すなわち, あるコンポーネントについて, 本来の仕様より狭い範囲でアサーションが記述されている場合, コンポーネント内での動作記述において, 必要以上の制約を課すことになる. これは, 各コンポーネントの動作内容について, 各部分動作が任意の時間かかり得るという前提のもとで検証することで解決される. しかし, この場合, 各コンポーネントの動作が有限状態では表現できなくなるため, 検証が困難となる. これに対しては, 検証可能なクラスを定めて検証方法を考える必要がある.

6. おわりに

本稿では, システム内の各コンポーネントのインタフェイスプロトコルに対するアサーションを, コンポーネント単位での検証に適用可能なように入力制約と示すべき性質について分割する方法を提案した. 例題に対して本手法を適用することにより, 例題に対して必要な検証項目が検証できた. これにより, 分割統治法による形式的検証を行うのに必要な, インタフェイスプロトコルについてのコンポーネントごとの入力制約や出力プロパティによる切り分けが可能であることの見通しを得た.

今後の課題として, 各コンポーネントの内部動作とインタフェイスプロトコルとを分離して検証する方法の考案が挙げられる.

謝　　辞

本研究を進めるにあたって多くの助言を頂きました大阪大学情報科学研究科の浜口清治助教授, 垣内洋介氏に深く感謝致します.

文　　献

[1] *e Golden Reference Guide ver 1.1.* Doulos, 2003.
[2] システムレベルデザイン. 情報処理, Vol. 45, No. 5, pp. 449–505, May 2004.
[3] accellera. *Propoerty Specification Language Reference Manual Ver 1.1*, 2004.
[4] accellera. *SystemVerilog3.1*, 2004.
[5] accellera SystemVerilog page. http//www.systemverilog.org/.
[6] Model Checking @CMU. http://www-2.cs.cmu.edu/ modelcheck/index.html.
[7] Ben Cohen. Verilog/VHDL 設計での PSL/Sugar 入門アサーション・ベース検証のための Propoerty Specification Language ガイド. VhdlCohen Publishing, 2004.
[8] Harry D.Foster, Adam C.Krolnik, and David J.Lacey. アサーションベース設計. 丸善株式会社, 原書 2 版, 2003.
[9] K.L.McMillan. *The SMV system*, 2000.
[10] Zaher S.Andraus and Karem A.Sakallah. Automatic abstraction and verification of verilog models. In *Proceedings of 41st Design Automation Conference*, 2004.