C言語によるサイクル精度での ハードウェア／ソフトウェア協調検証手法

Cycle-accurate Hardware/Software Coverification Method Using the C Language

鈴木郁子[*1] Ikuko Suzuki
亀井克比古[*1] Katsuhiko Kamei
冨田常雄[*2] Tsuneo Tomita
森下貴弘[*3] Takahiro Morishita
山田晃久[*3] Akihisa Yamada
久保登[*3] Noboru Kubo

要　旨

本稿では，ハードウェアとソフトウェアを高速かつ高精度で協調検証する手法について述べる。本研究では，ハードウェアとソフトウェアを同じ言語（C言語）で設計し，両者を同じ開発環境で動作させることで，高速なサイクル精度のシミュレーションを実現している。これにより，システムLSIとその上で動作するソフトウェアを同時に開発する場合にも，開発の初期の段階で，システムレベルの機能と性能を十分に検証できる。

This paper proposes a hardware/software coverification method of high speed and high accuracy. In this method, the same programming language, the C language, is used for both hardware and software design, and they are verified in a single fast cycle-accurate simulation environment. The proposed method enables designers to verify the functionality and performance of a system including hardware and software at an early design stage.

まえがき

近年，急速に大規模化・多機能化している組込みシステムは，開発期間の短縮の傾向もあり，その開発に多くの課題を抱えている。特に，魅力的な新機能の実現や桁違いの性能の向上を狙い，システムLSIなどのハードウェアを新たに開発する場合には，ソフトウェア開発はハードウェア開発の計画や進捗に大きく左右される事が多く，開発の後半が混乱しやすい，これは，

（1）ハードウェアや実機ができるまでソフトウェアのデバッグやテストが実施できない。
（2）試作デバイスは完成度が低くなりがちで，これを使うソフトウェアのデバッグやテストでは，不具合をハードウェア起因かソフトウェア起因かに切り分けるのが難しく時間を要する。
（3）デバッグやテストの結果，ハードウェアそのものに不具合があったとしても，商品化までの残り期間が短い場合には再設計ができず，ソフトウェアの設計変更で対応する。

からである。ハードウェア開発とソフトウェア開発の各々の効率を上げる[1) 2) 3) 7)]だけでは，組込みシステム全体の効率化は達成できない。両者を並行に開発でき，かつ協調設計できる開発環境が求められている。

本稿では，組込みシステムの中でも，システムLSIとその上で動作するソフトウェアを対象として，ハードウェアとソフトウェアを同時に協調させて設計できる開発支援手法について述べる。まず，1章で協調設計環境の紹介と本研究の手法を提案し，2章で適用した結果を示し，3章でその効果を議論する。

1．協調設計環境の構築

ハードウェアの設計手法として、ソフトウェアと同じC言語やC++言語を用いる設計手法が注目されている[4) 5) 6) 7) 8)]。我々は，ANSI C言語を拡張したBach C言語を用いる設計環境Bachシステムを開発し，ハードウェアの設計から検証，LSI化までの一連のフローを自動化し，システムLSIの開発効率化を達成してい

*1 IC事業本部　システムLSI事業部　第2商品開発部
*2 A1238PT（B）
*3 IC事業本部　先端技術開発センター　第1開発室

る[9]。ただし，その検証はハードウェアを対象としており，大規模な組込みソフトウェアの検証には向かない。そこで，このハードウェア開発環境にソフトウェアを高速かつサイクル精度でシミュレーションできる協調設計手法の確立を試みた。なお，サイクル精度とは，ハードウェアの状態をクロックサイクル単位で示す事ができる精度である。例えば，クロックの立ち上がりに同期して動作するハードウェアにおいては，そのクロック立ち上がり時のハードウェアの状態（レジスタやメモリの状態）を観測できる精度で，ハードウェアとその上で動作するソフトウェアのパフォーマンスの測定や評価等に適している。

まず，1・1と1・2と1・3ではBachシステムの概要とC言語ベースでのハードウェアの設計フローを説明する。次に，1・4でハードウェア・ソフトウェアの協調設計を実現する方法について提案する。

1・1　C言語によるハードウェア設計

Bachシステムでは，ハードウェア記述言語としてANSI C言語を選択している。その理由は，新規LSIのアルゴリズムを検討するにはANSI C/C++でプログラムを書くのが一般的であり，通信，マルチメディア関係など数多くのアルゴリズムがANSI Cを用いて既に実現されているからである。ただし，ANSI C言語では，ハードウェアの並列動作やビットアキュレートな演算を扱えないため文法を拡張している。具体的には、任意のビット幅演算を実現するためのデータ型・ハードウェアの並列動作を明示するためのpar構文・並列に動作するスレッド間の通信などである[7]。

```
1  void main(void){
2  {
3    chan int #8 to_ckt, from_ckt;
4    par {
5      circuit(to_ckt, from_ckt);
6      tbench(to_ckt, from_ckt);
7    }
8 }
9
10 void circuit(chan int#8 to_ckt,
11                chan int#8 from_ckt)
12 {
13   unsigned #4 i;
14   int #8 x;
15   for(i = 0; i < 10; i++){
16     x = receive(to_ckt);
17     send(from_ckt, i*x);
18   }
19 }
20
21 void tbench(chan int#8 to_ckt,
22               chan int#8 from_ckt)
23 {
24   unsigned #4 i;
25   for(i = 0; i < 10; i++){
26     send(to_ckt, i);
27     putint(stdout,10,0,receive(from_ckt));
28   }
29 }
```

図1　Bach C の記述例

Fig. 1　Example of Bach C description.

一方，仕様を制限している構文や型もある。goto文，再帰的呼出し，動的にアドレスが変わるポインタである。これらは，ハードウェア化（回路実装）に適しないためBach Cから除外している。

図1は，Bach Cによる記述例である。この例にしたがい，前述のANSI C言語の拡張について説明する。

1・1・1　任意ビット幅のデータ型

Bach C言語では，(signed) int型，unsigned型の数値のビット幅を指定することができる。これにより，冗長なハードウェアの生成を防ぐことができる（**図1**の13行目、14行目参照）。

1・1・2　par構文

並列に動作するハードウェアを直観的に記述できるように，Bach C言語にはpar構文が追加されている。**図1**の4-7行目では，関数circuit()とtbench()が並列に動作することが指定されている。

1・1・3　通信

Bach C言語では，並列に動作するスレッド間でデータをやり取りするための通信を記述することができる。**図1**の16行目と26行目ではtbench()のスレッドからcircuit()のスレッドへのデータ転送が記述されており，17行目と27行目ではその逆方向のデータ転送が記述されている。通信路には同期通信路と非同期通信路がある。同期通信路では，送信側と受信側の双方が準備できるのを待ってデータ転送が行なわれ，データ転送が完了するまで次の処理は行なわれない。一方，非同期通信路では，通信相手が準備できていなくても送信，または，受信が実行され，ただちに次の処理が行なわれる。

1・2　C言語ベースのハードウェア設計環境

Bachシステムは設計支援ツールとして，ハードウェアコンパイラ（動作合成ツール），会話型デバッガ，コンパイル型シミュレータ，プロファイラを備えている。

1・2・1　ハードウェアコンパイラ

ハードウェアコンパイラはBach C記述を動作合成してRTL記述に変換する[10]。このRTL記述は，論理合成が可能な記述である。すなわち，ハードウェアの構造を強く意識しない抽象度が高いBach C言語から，LSI

化が可能なハードウェア記述まで自動変換できる。Bach Cで拡張された並列処理や任意のビット幅の演算を配慮して効率の良いRTLに合成するほか，周波数や演算器の個数などの制約条件を指定し，回路面積や実行サイクル数を最適化することもできる。

合成されたRTL記述は既存のHDL（Hardware Description Language）シミュレータでも検証できる。

1・2・2　デバッガ，シミュレータ

Bachシステムのデバッガは，Bach C記述に対して，ブレークポイントの設定やステップ実行などができる対話型デバッガである[11]。

Bachシステムのコンパイル型シミュレータは，Bach CをANSI Cに変換し，標準Cコンパイラで実行モジュールを生成し動作をサイクル精度でシミュレーションする。前述のRTLレベルのシミュレーションに比べ，検証時間が2桁以上短くなる。

Bachシステムのデバッガとシミュレータは，任意ビット幅の演算や並列処理，通信処理などのBach Cで拡張された仕様にも対応している。

1・2・3　プロファイラ

プロファイラは，Bachシステムのシミュレーションを実行中にBach記述中の各ブロックの実行回数やカバレッジを測定する。これらの情報は，ボトルネックとなる箇所や実行回数が多い箇所の特定を容易にし，ハードウェア設計時の最適化の指針となる。

1・3　Bachシステムによるハードウェア設計フロー

Bachシステムを用いたシステムLSIの典型的な設計フローを**図2**に示す。

このように，Bachシステムによる設計では，動作の正しさを検証するアルゴリズムレベルの検証から，より細かな信号レベルで機能・性能を評価するRTLレベルまで，目的に応じた精度でハードウェアの検証ができる。

1・4　ソフトウェア協調設計環境の構築

1・4・1　従来手法とその問題点

従来，ハードウェアとソフトウェアを同時に検証するには，HDLシミュレータを用いる事が多かった。これにより，動作の確認や動作速度などのパフォーマンス測定ができる。しかし，HDLシミュレータは，クロックサイクルより短い単位の信号変化をとらえて，これを接続している信号に伝播させながらシミュレーションするので，計算コストが高く，大規模な回路のソフトウェアには適さない。

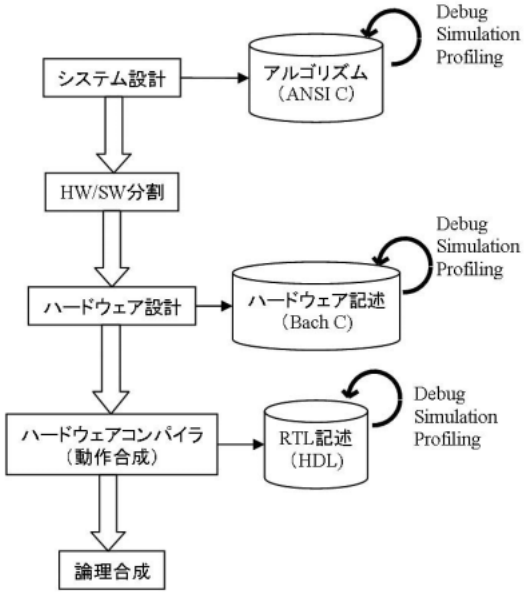

図2　Bachを用いたハードウェアの設計フロー
Fig. 2　Hardware design flow using Bach system.

1・4・2　サイクル精度モデルの生成

Bachシステムでは，抽象度の高いC言語でハードウェア設計を記述し，これを自動的にRTL記述に変換できる。この変換の仕組みをベースに，サイクル精度で実行可能なANSI C記述のモデルを生成するよう改良した。以下，具体的な手順を説明する。

（1）ハードウェアのBach C記述を字句解析・構文解析する。

（2）この解析で抽出された並列性に基づき，ブロックに分割する。

（3）各ブロックについて，ハードウェア動作の内部表現としてCDFG（Control Data Flow Graph）を作成する。
CDFGとはハードウェア動作をノードとノード間を接続する枝を用いて表現するもので，ノードはハードウェア中で動作する演算などを示し，枝はデータや制御の流れを示している。つまり，CDFGはデータの流れと演算の実行順序を表している。

（4）CDFGの各ノードに対して，要求されている動作周波数で動作可能になるようにスケジューリングを行ない，処理の実行順序を決定し，ハードウェアの動作クロックの区切りでCDFGをステート単位にまとめる。ここで，ステート単位とは，各クロックサイクル毎に実行される動作の事である。

（5）このステート単位毎の処理をサイクル精度モ

デルに変換する。このモデルはANSI-Cで記述された関数であり，ANSI C対応コンパイラでコンパイル可能なものである。これを実行することで，ハードウェア動作のシミュレーションが可能となる。

1・4・3　サイクル精度シミュレーション

ハードウェアのサイクル精度モデルはANSI Cで出力されるので，これをコンパイルして，命令セットシミュレータに組込む。一方、ソフトウェアは、クロスコンパイルし，命令セットシミュレータ上で実行させる。これにより，ハードウェアとソフトウェアの協調検証が可能になる（**図3**参照）。

2．結果

本研究の方法を実際のシステムLSIの設計と，その上で動作するソフトウェアの開発に適用した。

2・1　適用システムの概要と規模

適用したシステムは，動画データの伸長処理用アクセラレータとその制御ソフトウェアである。

表1　ハードウェアの規模
Table 1　Gate counts of circuits.

ブロック	ゲート数
Block1	30,000
Block2	187,000

本システムLSIは，長さの異なるデータと固定長のデータを効率よく処理するために，2つのモジュールからなり，それぞれのハードウェアの規模（ゲート数）は，**表1**のとおりである。

ソフトウェアはこの2つのハードウェアモジュールを用いて，動画像を伸長するもので，その規模は約6,000行である。

2・2　シミュレーション結果

図4は，本シミュレーションの実行例である。このように制御ソフトウェアのソースコードやハードウェアのレジスタの値等をリアルタイムに対話的に確認しながらシミュレーションすることができる。PentiumII 266MHzの環境でシミュレーションを行った場合の，1フレームあたりの動画伸長の実行時間を**表2**に示す。

表2　シミュレーション時間
Table 2　Simulation time.

フレーム番号	実行時間
1フレーム目	90秒
2フレーム目	48秒

また，シミュレーションの結果，測定されたサイクル数を**表3**に示す。

表3　実行サイクル数
Table 3　Number of execution cycles.

フレーム番号	サイクル数
1フレーム目	943,741
2フレーム目	506,864

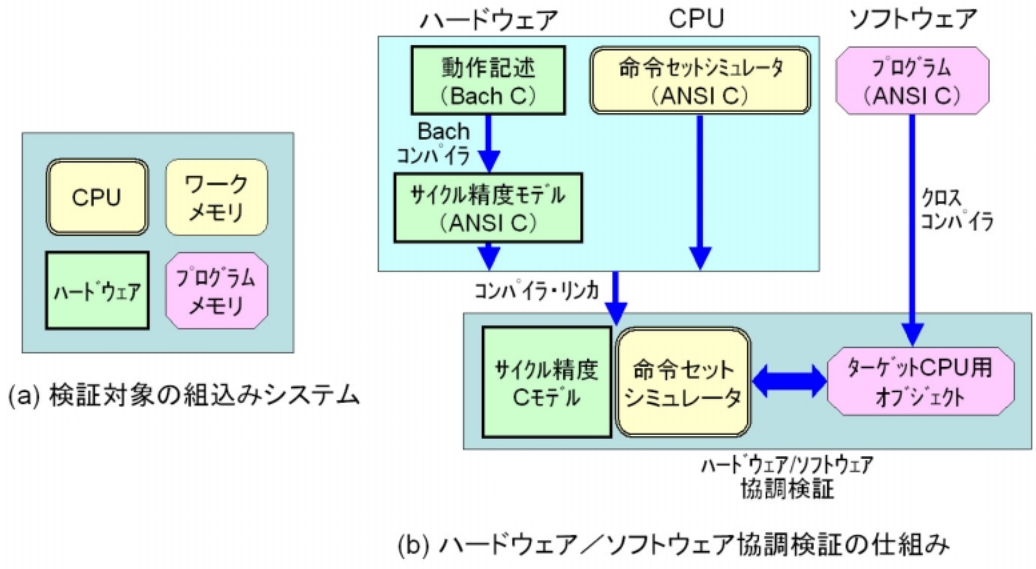

図3　ハードウェア／ソフトウェア協調検証
Fig. 3　Hardware/Software coverification.

3．考察

まず，本シミュレーション手法の有効性を，次に，ハードウェア・ソフトウェア協調設計の組込みシステム開発における有効性を，考察する。

3・1　シミュレーション速度と性能

Bach システムでは，ハードウェア設計の抽象度に応じたシミュレーション機能を用意しており，目的に応じて使い分けている。

表 4 には，3種類の抽象度によるシミュレーション時間の比較を示す。

表 4　抽象度とシミュレーション時間

Tabel 4　Design abstruction and simulation time.

回路規模	CPUtime (sec)		
ゲート数	アルゴリズム	サイクル精度	RTL
9,100	3	162	6,700
31,000	7	3,530	21,390

これから分かるようにサイクル精度レベルのシミュレーションは，RTL レベルのシミュレーションに比べ桁違いに高速である。ちなみに，2章で適用したハードウェアを RTL レベルでシミュレーションすると，1フレームを出力するのに約1時間を要する。一方，アルゴリズムレベルのシミュレーションはサイクル精度レベルのシミュレーションより高速であるが，あくまで動作の確認ができるだけであり，ハードウェア性能の評価が難しい。サイクル精度のシミュレーションでは，動作周波数に応じたサイクル数が測定でき，また，任意のサイクルでのハードウェアの動作を確認できる。これらの情報は，ハードウェアのみならずシステム全体の最適化に不可欠のものである。

同一アルゴリズムを異なる抽象度のハードウェア記述言語で比較すると、その記述量は、アルゴリズムレベル（C 言語）、サイクル精度（本手法）、RTL レベル（HDL 言語）の順に大きくなる。シミュレータは基本的には各行を 1 つずつ実行するので，記述量の差はシミュレーション速度に大きく寄与する。

アルゴリズムレベルの記述は，Bach C 言語で記述されており，実装すべき機能を簡潔に表現できる。Bach システムでは Bach C 記述をステップ実行できるシミュレータとデバッガを備えており，高速に検証できる。しかし，ハードウェアリソースへの割り当てがされていないので，その性能の評価が難しい。

RTL レベルの記述は，アルゴリズムをハードウェアの動作として記述しており，RTL シミュレータで

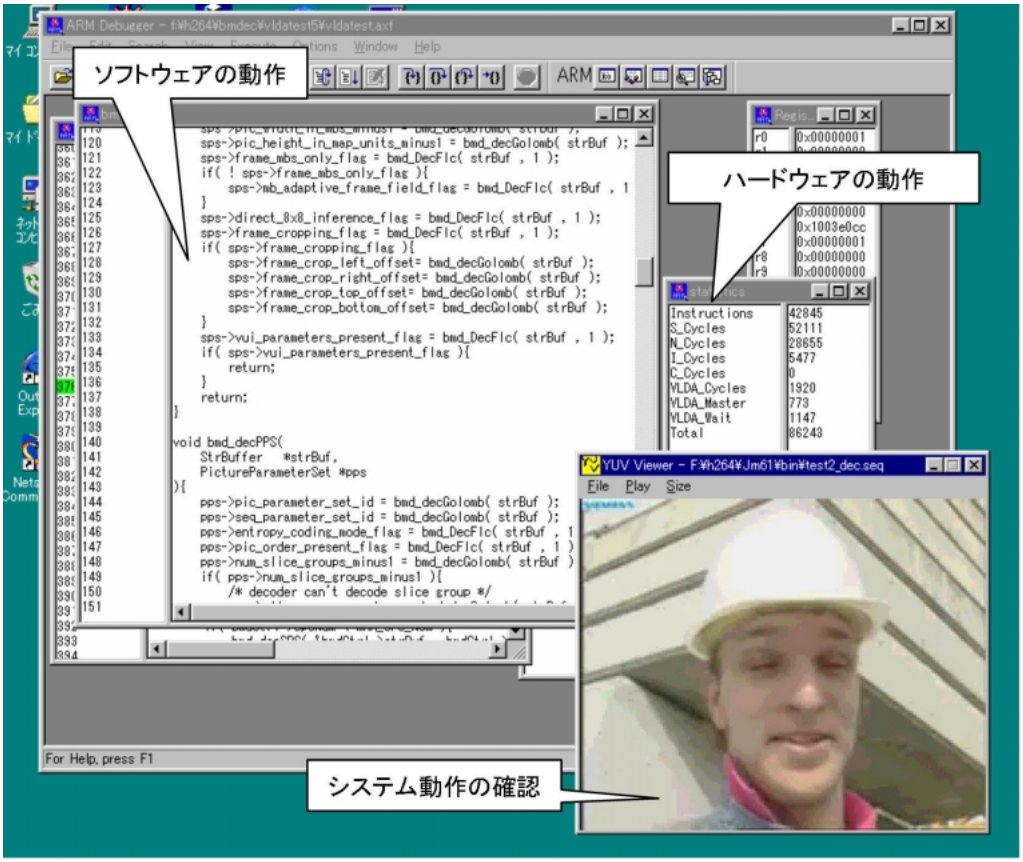

図 4　ハードウェア・ソフトウェア協調設計の実行例

Fig. 4　Snapshot of hardware/software co-design environment.

クロック単位で実行しシミュレーションでき,処理速度などの性能も評価できる。しかし,信号の伝播を逐一追う方式であるので,実行に時間がかかる。

本研究のサイクル精度の記述は,両者の中間的な抽象度と精度を有する。すなわち,比較的高速に実行し,かつ必要なハードウェア性能を測定できる。前述のようにこのサイクル精度のCモデルの記述も,Bach C記述から自動変換される。

以上のことから,システム評価には,サイクル精度のシミュレーションが有効と考えられる。

3・2　組込みシステム開発での有効性

従来は,十分に検証されたハードウェアのHDLを生成後に,HDLシミュレータを用いるか,FPGA（Field Programmable Gate Array）ボードを作成して,ソフトウェアの評価やデバッグを行っていた。しかし,前述したように,前者の方法では,処理速度に問題があり,大規模なシステムでは実用的でない。また,後者の方法では,周辺回路との結合ミスなど実装に伴う不具合が避けられず,FPGAのデバッグの負荷が重なる上に,不具合の原因の特定が難しい。

本研究の協調検証手法を用いることにより、ハードウェアの試作前に、ソフトウェアの評価とシステム全体の評価を実現できた。さらに,試作をせずに評価が可能なことから,多くの設計案を評価してシステム全体の最適化が図ることができた。これらは,短い開発期間で競争力ある製品の完成を要求される組込みシステムの開発では非常に有効と考える。

また,組込みシステムを開発するためのFPGAボードや試作デバイスボードは,製品ボードに比べると完成度・安定性にかけ固体差が大きい。このために輸送による動作不良や,電源の安定度が違う環境では動作しないなどの問題が発生する。組込みソフトウェアの開発を分散させにくい理由の１つが,動作環境のバラツキであるが,本手法は,これを解決する。

むすび

ハードウェア設計をC言語ベースで行ない,サイクル精度で実行可能なANSI C言語の動作モデルに自動変換する手法を確立した。これにより,ハードウェアとその上で動作するソフトウェアを同時に検証可能となった。また,その処理速度は従来のHDLシミュレータに比べ,約50倍速い事を確認した。さらに,動作の正しさのみならず,サイクル精度で計算量やボトルネックなどの性能指標を測定できた。

ハードウェアとソフトウェアの協調設計は,システム全体のデバッグと評価を前倒しするとともに,実装に伴う不具合を除外した論理的なデバッグができ,設計での手戻りを削減する。結果,組込みシステムの開発において早期にシステムの最適化を可能にする。

謝辞

本手法の開発にあたり、ご指導およびご協力を戴きました関係各位に深謝いたします。

参考文献

1）　鈴木，冨田，乙井，上田，“ソフトウェアトップダウン開発手法と適用例”，ソフトウェアシンポジウム1999, pp.146-152, (2000).

2）　福田，亀井，屋鋪，冨田，鈴木，神戸，“組込みソフトウェアにおける自動テストツールの開発と適用例”，ソフトウェアシンポジウム2000, pp.171-178, (2000).

3）　田中，松本，鈴木，福永，中西，福田，“Ballista methodによるプログラム強靭性評価手法”，ソフトウェアテストシンポジウム2005, pp.41-46, (2005).

4）　A. Ghosh, J. Kunkel, and S. Liao, “Hardware Synthesis from C/C++,”Din *Proc. of DATE'99*, pp.387-389,（1999）

5）　G. Arnout, “C for System Level Design,” in *Proc. of DATE'99*, pp.384-386, (1999).

6）　D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, S. Zhao, *Spec C: Specification Language and Design Methodology*, Kluwer Academic Publishers.

7）　A.Yamada, K. Nishida, R. Sakurai, A. Kay, T. Nomura, T. Kambe, “Hardware synthesis with the Bach system,” in *Proc. of IEEE ISCAS'99*, Vol. VI, pp.366-369, (1999).

8）　K. Wakabayashi, “C-based Synthesis Experiences with a Behavior Synthesize, “Cyber”,” in *Proc. of DATE'99*, pp.390-393, (1999).

9）　岡田他，“Bach C言語を用いた回路設計方法”，第14回軽井沢ワークショップ，pp.450-410,（2001).

10）　西田，岡田，大西，A. Kay, P. Boca，山田，神戸，“ハードウェアコンパイラBachの動作合成系，”DA シンポジウム'99, pp.95-100, (1999).

11）　H. Makida, T. Morishita, M. Ohnishi, K. Okada, A. Yamada, T. Kambe, P. Boca, “Verification Environment for C-Based Design,” in *Proc. of SASIMI 2003*, pp.317-322, (2003).

（2005年6月7日受理）