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

2007－ARC－171 (4)
2007－EMB－ 3 (4)
2007／1／22

動作合成とFPGAを利用したCベース協調設計・検証手法

柴田 誠也† 本田 晋也‡ 冨山 宏之‡ 高田 広章‡
†名古屋大学 工学部電気電子・情報工学科
‡名古屋大学 大学院情報科学研究科情報システム学専攻
Email: {shibata, honda, tomiyama, hiro}@ertl.jp

概要
本研究では，ハードウェアとソフトウェアの協調設計・検証を効率的に行う手法を提案する．効率的な検証のため，複数のツールを用いる上で必要なソフトウェア・ハードウェア間通信のための通信ライブラリを作成した．通信ライブラリにより，設計記述に変更を加えることなく検証を行うことが出来る．MPEG4 デコーダシステムの設計を行い，検証に要した時間および記述の変更量から本手法の有効性を示す．

A C-Based Hardware/Software Codesign and Coverification Method with High-Level Synthesis and FPGA Emulator

Seiya Shibata† Shinya Honda‡ Hiroyuki Tomiyama‡ Hiroaki Takada ‡
† Dept. of Information Engineering, Nagoya Univ.
‡ Graduate School of Information Science, Nagoya Univ.

Abstract This paper presents a C-based hardware/software codesign and coverification method. For efficient verification, we prepared a communication library which enables hardware and software easily communicate with each other. With our communication library, there is no need to change the hardware/software description through a series of verification processes. We show the effect of presented method by designing an MPEG4 decoding system.

1 はじめに

近年，組込みシステム開発においては，製品の企画から市場への投入までの時間（Time-to-Market）の短縮が重要となっている．Time-to-Market の短縮には，設計生産性の向上による開発期間の短縮が不可欠である．設計生産性を向上させる方法としては，設計対象をできるだけ高い抽象度で設計する方法があり，ハードウェア設計においては C 言語もしくはシステムレベル言語と動作合成技術を用いた動作レベルの設計が適用されている [1].

また，設計したシステムの検証を高速に行う手法として FPGA エミュレータの利用が提案されている．ハードウェアの RTL 記述の検証には RTL シミュレータの利用が一般的であるが動作速度が遅く，高速にシミュレーションを行おうとすると精度を犠牲にする必要がある．これに対し FPGA エミュレータは高速に動作し，同時に高い精度で検証を行うことが出来る [2][3]. [2] は FPGA にハードウェアの内部信号を観測する機能を持たせ，ホスト計算機との通信によりサイクル精度で高速にシミュレーションを行う手法を提案している．

しかし上記のような設計技術や検証技術を用いる場合，ソフトウェア・ハードウェア間の通信インタフェースがそれぞれ異なり，技術を適用するごとに通信インタフェースのための記述を変更する必要がある．例えば RTL シミュレータを用いてコシミュレーションを行う時には，設計した記述に対しコシミュレーション用の記述追加・変更を行わなければならない．これに加えて FPGA を用いた検証を行うには，コシミュレーション用とは別の記述追加・変更が必要となる．検証技術を変更するたびに記述を変更しなければならない環境では記述変更による不具合も発生しかねず，設計・検証に手間がかかり Time-to-Market の短縮は難しい．

そこで本研究では各技術をひとつの設計・検証手法としてまとめ，通信インタフェースと通信ライブラリを整備した．通信インタフェースにより設計・検証手法を通して同一の通信記述を用いることができ，通信ライブラリによって，新規に記述を作成することなく各検証技術の通信方法に対応することが可能になる．これにより複数の検証技術を用いる上での手間を省き，設計・検証に要する時間を短縮することが出来る．

通信を考慮した設計・検証の効率化には [3] があり，これは FPGA エミュレータを使用し，FPGA・ホスト計算機間の通信記述を効率的に作成する方法を示している．しかしハードウェアの内部信号を観測する機能がなく，RTL シミュレーションの利用につ

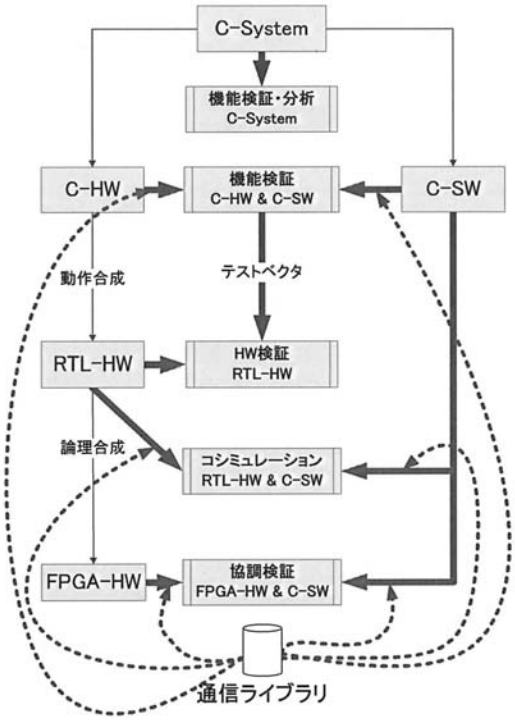


図 1: 動作合成と FPGA を利用した協調設計・検証手法

いても言及されていない．ハードウェアを検証する上で内部信号の観測は不可欠であり，設計・検証手法のなかで重要な位置を占めている．提案手法では RTL シミュレーションによる内部信号の観測と他の検証技術による効率化を組み合わせることで，システムの設計・検証全体にかかる時間を短縮する．

以下，本論文では，提案手法について 2章で概略を示し，3章で詳細に述べる．4章では MPEG4 デコーダの設計事例により提案手法の有効性を評価する．

2 設計・検証フロー概略

図 1に我々の提案する協調設計・検証手法の全体像を示す．本手法の目的はシステム記述からソフトウェア，ハードウェアの設計・検証までを効率的に行うことにある．以下では記述の作成と検証を合わせて 1 つの段階とし，本手法の各段階について説明する．

本手法では，まずシステム全体を C 言語で設計する．この段階ではシステムを計算機上のソフトウェアとして動作させ，要求仕様を満たしているかどうかを検証する．また，システム内で高速化が必要な部分の分析もこの段階で行う．この段階を図 1では「C-System」と「機能検証・分析 C-System」として表している．

次の段階では，C 言語で記述したシステムを実際のシステムにおいてソフトウェアとして動作させる部分とハードウェアとして動作させる部分とに分割する．ここで分割とは，ソフトウェア部分とハードウェア部分をそれぞれ独立した C 言語記述へと分けることを言う．分割点については前段階での分析結果から決定する．分割したソフトウェア部分とハードウェア部分とをホスト計算機上で並行に動作させ，両者が分割前と同様の機能を満たせばこの段階は完了となる．さらに，後の段階でハードウェアを検証するためのテストベクタの採取もここで行う．この段階は図 1では「C-HW」「C-SW」および「機能検証 C-HW & C-SW」の部分に相当する．

ここまでの段階でシステムが要求仕様を満たすことを確認できる．本手法ではソフトウェア部分はここで検証を終えたものとし，ここからはハードウェア部分の詳細な設計へと移る．本手法が想定する動作合成ツールは C 言語記述を入力とし，レジスタ転送レベルの記述（RTL 記述）を出力する．この動作合成ツールを使用して，さきほど作成したハードウェア部分の C 記述を RTL 記述へと合成する．動作合成により生成した RTL 記述は，前段階で採取したテストベクタを用い，RTL シミュレータを使用して検証を行う．図 1では「動作合成」,「RTL-HW」,「HW 検証 RTL-HW」および「テストベクタ」に対応する．

以上により RTL 記述へと変換されたハードウェア部分を，ソフトウェア部分とのコシミュレーションにより検証する．ただし，ハードウェアの規模が大きい場合にはコシミュレーションにかかる時間が非実用的に長くなってしまうので，ここでは主に動作の一部，特に通信についての検証を行う．図 1では「コシミュレーション RTL-HW & C-SW」に対応する．

本手法では FPGA エミュレータによる検証を行う．このとき使用する FPGA は計算機との通信機構を持ち，ソフトウェア部分と通信させた協調検証を可能にする．FPGA を利用することでハードウェア部分は高速に動作し，大規模なハードウェアとの協調検証であっても短時間で完了する．この段階は図 1の「FPGA-HW」および「協調検証 FPGA-HW & C-SW」に対応する．

FPGA を使用した検証によりハードウェアの動作および有効性を判断し，問題があれば HW/SW 分割段階や動作合成段階へと戻り設計をやり直す．

今回，我々は通信インタフェースを定義し,「機能検証 C-HW & C-SW」,「コシミュレーション RTL-HW & C-SW」,「協調検証 FPGA-HW & C-SW」の各検証時に使用する通信ライブラリを作成した．これにより上記の各検証段階においてソフトウェア部分の通信用記述を変更する必要はない．また，ハー

ドウェア部分では動作合成により生成したRTL記述を変更することなく，通信ライブラリによって「コシミュレーションRTL-HW & C-SW」および「協調検証FPGA-HW & C-SW」に対応させることができる．現在通信ライブラリはハンドシェイク通信のみに対応している．

3 設計・検証手法詳細

本章では2章で概略を述べた設計・検証手法の詳細について説明する．

3.1 C言語によるシステム記述とHW/SW分割

協調設計・検証においてソフトウェア設計とハードウェア設計は並列に進行することが不可欠である．本手法ではC言語を入力とする動作合成ツールを用いることでソフトウェア設計だけでなくハードウェア設計でもC言語を使用することができ，ハードウェア・ソフトウェア双方を同時に設計することができる．

システム記述（C-System）は，システム全体を計算機上で動作するアプリケーションプログラムとして作成する．記述したシステムの機能検証や分析は，記述を実際にコンパイルし動作させることで行う（機能検証・分析C-System）．そのため，この段階ではデバッグや分析のためのツールとしてソフトウェア開発と同様のものを使用することができる．

機能検証と分析の完了後，システムをハードウェアとして実現する部分（C-HW）とソフトウェアとして実現する部分（C-SW）へと分割する．分割によるハードウェア化はシステム記述から一部を抜きだす形で行う．抜きだしたハードウェア部分のC記述は，独立したC言語プログラムとする．ここで，単に記述を2つの独立なプログラムに分けるだけではこれらが協調して動作することを検証できない．そこで分割した後，ソフトウェア部分とハードウェア部分のそれぞれに通信用入出力の記述を加える．

以上により分割され，独立したプログラムとなったソフトウェア部分とハードウェア部分を並行に実行し通信させ，システムが分割前と同様の機能を持つことを検証する（機能検証C-HW & C-SW）．この段階ではソフトウェア・ハードウェアの機能を検証することが目的であるため，通信の実装には計算機上での実現が容易な仕組みを用いればよい．また検証と同時に，後のハードウェア検証用テストベクタとして用いるための通信履歴を採取しておく．

本研究では通信インタフェースとして，後の段階で使用する動作合成ツールeXCite[5]により提供される通信用関数を採用した．これにより後の段階でもソフトウェア部分の通信記述を変更する必要がなくなる．我々が作成した通信ライブラリはハンドシェイク通信をサポートしているため，この段階での検証にはeXCiteが提供するライブラリのハンドシェイク通信機能を用いる．提供されるライブラリにより，通信インタフェースを使用したC言語記述をホスト計算機上でコンパイルするとソケット通信によるソフトウェア・ハードウェア間通信が可能になる．また，通信と同時にテストベクタの採取を行うこともできる．

3.2 C記述ハードウェアの動作合成

動作合成により，3.1節で作成したハードウェアのC記述をRTL記述（RTL-HW）へと変換する．得られたRTL記述は「機能検証C-HW & C-SW」時に採取したテストベクタを用いてシミュレーションを行う（HW検証RTL-HW）．

本研究では，動作合成ツールとしてeXCiteを使用した．eXCiteはC言語を入力とし，Verilog-HDLおよびVHDLによるRTL記述を生成する動作合成ツールである．動作合成を行う際，検証用にテストベンチも作成する．このテストベンチを用い，RTL記述の検証を行う．

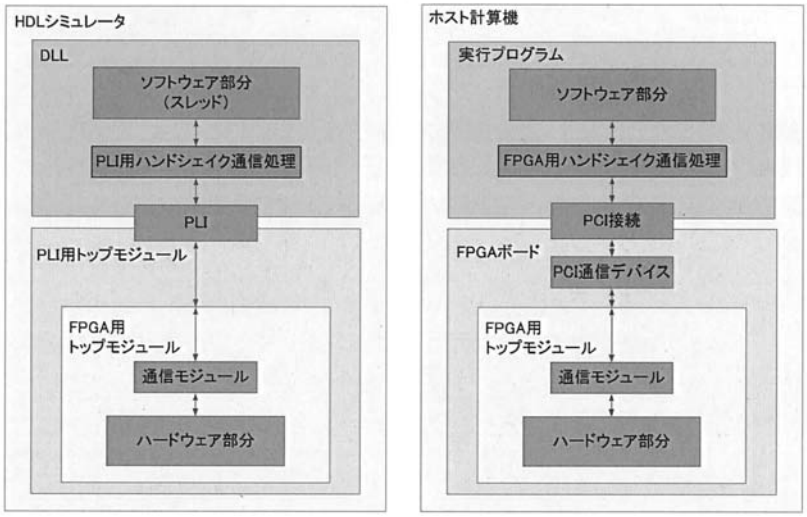
eXCiteにより提供される通信用関数は，動作合成時にはハードウェア部分の入出力定義として用いられる．これにより「機能検証C-HW & C-SW」時と動作合成時で記述の変更が不要となる．

3.3 RTL記述ハードウェアとC記述ソフトウェアのコシミュレーション

ここでは図1における「コシミュレーションRTL-HW & C-SW」について述べる．

コシミュレーションは，C言語プログラムとして動作する「C-SW」とシミュレータ上で実行する「RTL-HW」とを通信させて行う．このため，C記述ソフトウェアおよびRTL記述ハードウェアの通信処理を通信ライブラリにより実現する．通信処理を追加されたシステムの協調検証にはVerilog-HDLのPLIを用いる．PLIはVerilog-HDLとC言語との接続を可能にする仕組みであり，Verilog-HDLシミュレータ内でC言語プログラムを動作させることができる．PLIを用いると，C言語プログラムからシミュレーション対象のRTL記述（Verilog-HDL記述）に対する入出力を制御することができる．PLIを用いた検証の仕組みを図2（a）に示す．HDLシミュレータ内の仕組みは大きく分けてC言語実行部（図内上側）とRTLシミュレーション部（図内下側）とで構成される．C言語実行部内においてソフトウェア部分はスレッドとして実行し，RTLシミュレーション部内でハードウェア部分を動作させ，これらの通信をPLIを用いたC言語プログラムが仲介する．PLIを使用した通信にはC言語を用いるので，柔軟な記述が可能である．このため，後の段階で使用するFPGAとホスト計算機との通信方法に合わせることができる．これによりPLIシミュレーション時とFPGAエミュレーション時のRTL記述を統一し，検証環境の変化に伴う不具合の発生を抑えることが出来る．

ソフトウェア部分の検証は「機能検証C-HW & C-



(a)PLIシミュレーション環境　　(b)FPGAエミュレーション環境

図 2: **PLI** および **FPGA** を用いた検証環境

SW」時に終えており，ハードウェア部分の検証は「HW 検証 RTL-HW」時に完了しているため，PLI を使用した検証により通信用記述の検証が可能となる．

本研究では通信ライブラリとして，ハンドシェイク通信を行う PLI 用 C 言語記述と通信用 RTL 記述を作成した．このライブラリにより，ソフトウェア部分の C 言語記述やハードウェア部分の RTL 記述に変更を加えることなくコシミュレーションを行うことが出来る．

3.4 FPGA エミュレーションと計算機上 C ソフトウェアによる協調検証

ハードウェアが大規模である場合，実行速度の遅い RTL シミュレータによるシステム全体の検証は現実的ではない．システム全体の検証は FPGA エミュレータを用いて高速に行う．

FPGA を使用した協調検証の仕組みを図 2（b）に示す．ハードウェア部分は計算機との通信機構を持つ FPGA によりエミュレーションし，ソフトウェア部分を計算機上で動作させ，両者を通信させて協調検証を行う．

FPGA エミュレーションには PLI シミュレーション時と同様の RTL 記述を使用し，作成したハードウェア記述をそのまま論理合成して FPGA に書き込む（FPGA-HW）．ソフトウェア部分の FPGA 用通信処理は通信ライブラリにより実現する．この変更の後，ソフトウェアをコンパイルし実行することで計算機上ソフトウェアと FPGA エミュレータによる協調検証をすることができる（協調検証 FPGA-HW & C-SW）．この検証は RTL シミュレーションとは違い，ハードウェア内部の信号を観測することはできない．しかし高速に実行を終え，分割前の C 記述システムによる結果と比較することでハードウェアの動作を確認することが可能である．

本手法を用いると高速に検証できるため，設計したシステムを短時間で評価ができ，何度も設計をやり直すことができる．こうすることでシステムの最適な HW/SW 分割点を探索することが可能になる．

今回我々はホスト計算機との通信を行う FPGA として，共有通信レジスタを持つ FPGA[4] を使用した．共有通信レジスタは計算機側から観測・操作することができ，動作合成により生成された RTL 記述ハードウェアの入出力信号と接続することでホスト計算機とハードウェア部分の通信を行うことが出来る．ホスト計算機との接続には PCI バスを使用する．

通信ライブラリとしては，ハードウェア部分については PLI によるコシミュレーション時と同じものを使用した（図 2）．一方，ソフトウェア部分には FPGA との通信機構を使用するものを用意した．通信インタフェースは分割・検証時に追加したものと一致させてあるため，記述を変更する必要はなかった．

4 実験

ここでは提案手法の効果を確かめるために行った実験について述べる．実験では設計対象として MPEG4 デコーダを選択した．

本手法では，C 言語コンパイラ，動作合成ツール，RTL シミュレータ，および計算機と通信可能な FPGA を用いる．それぞれのツールに求める機能は 2章で述べた．今回我々が本手法の実行環境を構築するた

めに使用したツールを以下に示す．

- C 言語コンパイラ：Visual C++
- 動作合成ツール：eXCite
- RTL シミュレータ：ModelSim SE
- FPGA：Spartan3-4000
  動作周波数 15MHz

実験は，プロセッサが Intel Core 2 Duo（2.66GHz）の，メモリを 2GB 搭載した計算機を使用して行った．計算機の OS は Windows XP である．

4.1 設計対象：MPEG4 デコーダ

MPEG4 デコーダは入力として動画データを受け取り，1 フレームごとに復号して静止画像とし，フレーム数分の静止画像データを出力する．1 フレームに対する復号処理の流れを図3に示す．MPEG4 デコーダは入力データを可変長復号し，逆量子化，IDCT（逆離散コサイン変換）の順で復号して画像を得る．これと同時に動きベクトルにより過去のフレームから動き補償を行い，フレームの予測画像を作成する．これら 2 つの画像を加算したものを復号結果画像とし，出力する．

実験ではデコーダシステムの C 記述として EEMBC の提供する MPEG4 デコーダを使用した．この MPEG4 デコーダはビデオコーデックとして Xvid を採用したベンチマークソフトであり，実行時間評価用の拡張がなされている．また，EEMBC により浮動小数点演算を極力使用しない仕様となっており動作合成しやすいという利点がある．デコード対象としてフレーム数 49，サイズ 192×192 の動画データを用いた．

4.2 ソフトウェア・ハードウェア分割

実験として，MPEG4 デコーダの IDCT 処理のハードウェア化と，逆量子化と IDCT を合わせたハードウェア化の 2 種類を行った．

まず，IDCT 処理のハードウェア化について述べる．本論文で提案する手法の最初の段階であるシステムの C 記述は，既に完成したものを使用するため省略することが出来た．完成している MPEG4 デコーダを計算機上アプリケーションプログラムとしてコンパイル・実行し，正常動作を確認した．この MPEG4 デコーダシステムに対しプロファイリングプログラム gprof を使用して，デコーダ内部で使用する各関数に要する実行時間を解析した．解析結果から IDCT が最も処理に時間がかかることがわかったため，IDCT 処理をハードウェア化対象とした．提案手法に則り動作合成，テストベンチシミュレーション，PLI によるコシミュレーション，および FPGA エミュレーションを用いた協調検証を行い，IDCT 処理のハードウェア化に成功した．

以上の実験により，IDCT のみのハードウェア化が可能であることを確認した．しかし IDCT だけではまだ回路規模として FPGA に余裕があった．IDCT の直前に必ず行う逆量子化処理も時間のかかる処理であり，ハードウェア化の効果が見込めると判断した．そこで次に逆量子化と IDCT の処理をまとめてハードウェア化することとした．

準備段階として，提案手法に則り逆量子化のみをハードウェア化した．これにより逆量子化ハードウェアの RTL 記述を検証し終え，その後，IDCT の C 記述と逆量子化の C 記述をひとつの C 記述にまとめた．C 記述をまとめた際の変更は小さく，通信処理にも大きな変更は起きなかったため，まとめたハードウェア記述は動作合成後に論理合成して FPGA に書き込み，計算機上ソフトウェアと協調検証を行うことで検証を終えることができた．逆量子化のハードウェア化をはじめてから検証完了までの作業にかかった時間は 1 時間弱である．

4.3 シミュレーション時間の評価

提案手法では FPGA エミュレータの導入により，高速な検証を可能としている．このことを示すため，設計した MPEG4 デコーダを各検証方法について 10 回ずつ実行し，要した時間の平均を表1にまとめた．表中「RTL-HW テストベンチ」および「C-SW & RTL-HW（PLI）」で示した値は RTL シミュレーションを行った時間であるが，この値は FPGA に内部信号の観測機能がないことを考慮し，波形などの情報を一切取得しない条件設定で計測した．実験結果から，「C 記述システム（分割前）」の検証時間が「RTL-HW テストベンチ」の検証時間の約 4000 分の 1 以下であり，効率面での RTL 記述に対する動作合成の優位性が見て取れる．また，PLI を使用したコシミュレーションでは逆量子化と IDCT を合わせたハードウェアについて 5 分以上を要したのに対し FPGA エミュレータによる協調検証では 3 秒未満で終えており，FPGA エミュレータ導入による設計時間の短縮効果が確認できた．

IDCT のみをハードウェア化した場合と逆量子化・IDCT をまとめてハードウェア化した場合とを比べると，PLI シミュレーションでは実行時間が 1.39 倍になっているのに対し，FPGA エミュレーションでは 1.05 倍に留まっている．逆量子化を含めた C 記述は，IDCT のみの C 記述に対して 1.2 倍程度記述量が増加した．RTL シミュレーションの実行時間は記述増加量以上に実行時間が伸び，FPGA エミュレーションは記述増加量の影響が小さいことが示された．より大規模なハードウェアを設計する場合には PLI シミュレーションと FPGA エミュレーションの実行時間の差が大きくなることが予想される．

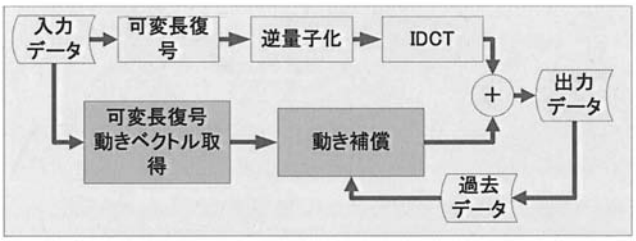


図 3: MPEG4 デコーダ処理の流れ

表 1: 各段階の検証時間比較

検証段階	IDCT	逆量子化-IDCT
C 記述システム（分割前）	0.049 秒	0.049 秒
C 記述システム（分割後）	0.568 秒	0.755 秒
RTL-HW テストベンチ	218 秒	313 秒
C-SW & RTL-HW（PLI）	242 秒	336 秒
C-SW & FPGA-HW	2.59 秒	2.74 秒

4.4 通信ライブラリの効果

本研究ではソフトウェアハードウェア間通信を整備し検証段階ごとの記述変更を軽減することで設計・検証を効率化することを試みた．行った実験では，HW/SW 分割時に行う通信関数の追加以後，ソフトウェア部分の記述変更はリンク先ライブラリの変更のみである．ライブラリの C 言語記述量は，RTL シミュレータ用が 1084 行，FPGA エミュレータ用が 324 行であるが，通信データ量の変化に対応するにはそのうち 10 行以下の変更で済んだ．また，ハードウェア記述の変更は，動作合成後の RTL 記述に対する通信用モジュールの追加である．この通信用モジュールの RTL 記述量は 496 行であり，通信データ量変更への対応はそのうちの 20 行以下の変更で完了した．

以上のことから，システム設計・検証における本手法の有効性を確認できた．

5 まとめと今後の課題

本研究では動作合成と FPGA エミュレータを利用した協調設計・検証の手法を提案した．MPEG4 デコーダを使用した実験から，本手法により協調設計・検証に要する時間を短縮できることを確認した．

今後の課題としては，通信ライブラリの実装を充実させることがあげられる．現在はハンドシェイク通信にのみ対応しており，設計対象に対して適切な通信をシミュレーションすることができない．複数の通信方法を実装したライブラリを用意することで，通信方法の設計を含めた設計・検証環境にできると考えられる．

謝辞

FPGA ボードをご提供頂いた NEC メディア情報研究所の諸氏に感謝致します．

参考文献

[1] K. Wakabayashi, "CyberWorkBench: Integrated Design Environment Based on C-Based Behaviour Synthesis and Verification," *VLSI Design, Automation and Test*, 2005.

[2] Y. Nakamura, K. Hosokawa, "Fast FPGA-Emulation-Based Simulation Environment for Custom Processors," *IEICE Trans. on Fundamentals ECCS, vol.E89-A no.12*, 2006.

[3] Young-Il Kim, Ki-Yong Ahn, Heejun Shim, Wooseung Yang, Young-Su Kwon, Ando Ki, Chong-Min Kyung, "Automatic Generation of Software/Hardware Co-Emulation Interface for Transaction-Level Communication," *VLSI Design, Automation and Test*, 2005.

[4] Y. Nakamura, K. Hosokawa, I. Kuroda, K. Yoshikawa, T. Yoshimura, "A fast hardware/software co-verification method for system-on-a-chip by using a C/C++ simulator and FPGA emulator with shared register communication," *Design Automation Conference*, 2004.

[5] Y Explorations 社, http://www.yxi.com/.