2005－ＳＬＤＭ－119（７）
2005／3／17

# FSMへの変換に基づくHW/SW協調設計の形式的検証手法に関する研究

西原　佑†　松本　剛史††　小松　聡†††　藤田　昌宏†††

† 東京大学工学部電子工学科　〒113-8656 東京都文京区本郷 7-3-1
†† 東京大学大学院工学系研究科電子工学専攻　〒113-8656 東京都文京区本郷 7-3-1
††† 東京大学大規模集積システム設計教育研究センター　〒113-0032 東京都文京区弥生 2-11-16
E-mail: †{tasuku,matsumoto,komatsu}@cad.t.u-tokyo.ac.jp, ††fujita@ee.t.u-tokyo.ac.jp

**あらまし**　本稿では，ハードウェアがVerilogやVHDLなどによりRTLで，ソフトウェアがC言語などによりプログラムコードで記述された，ハードウェア/ソフトウェア協調設計記述に対して形式的検証を適用する手法を提案する．提案する手法の特徴は，設計におけるハードウェアとソフトウェアの両記述を自動的に抽象化されたFSMへと変換し，既存のプロパティ検証器で一緒に検証を行う点である．本稿で扱う形式的検証は，プロパティ検証と，ハードウェア/ソフトウェア分割前のソフトウェア設計と分割後のハードウェア/ソフトウェア協調設計間の等価性検証である．両者について検証全体の流れと，幾つかの例題に対しての実験結果を示した．

**キーワード**　ハードウェア/ソフトウェア協調検証，形式的検証，C言語，RTL，FSM

# Formal Verification of HW/SW Co-design with FSM Translation

Tasuku NISHIHARA†, Takeshi MATSUMOTO††, Satoshi KOMATSU†††, and Masahiro FUJITA†††

† Department of Electronics Engineering, Faculty of Engineering, University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo, 113-8656 Japan
†† Department of Electronics Engineering, School of Engineering University of Tokyo
Hongo7-3-1, Bunkyo-ku, Tokyo, 113-8656 Japan
††† VLSI Design and Education Center, University of Tokyo
Yayoi 2-11-16, Bunkyo-ku, Tokyo, 113-0032 Japan
E-mail: †{tasuku,matsumoto,komatsu}@cad.t.u-tokyo.ac.jp, ††fujita@ee.t.u-tokyo.ac.jp

**Abstract**　In this paper we present a methodology for formal verification of common hardware/software co-design. Two methodologies are proporsed. One is for property checking of hardware/software co-design in C/RTL, and the other is for equivalence checking between software design in C and hardware/software co-design in C/RTL converted from the software design. We extract caluculation part from them, abstract transactions between hardware and software, translate the description into finite state machines, and verify them with existing property checking tools. We report experimental results with a couple of examples.

**Key words**　HW/SW Co-verification, Formal Verification, C , RTL, FSM

## 1. はじめに

デジタルコンシューマ機器などで利用される組み込みシステムでは，SoC(System on a Chip)やシステムLSIなどが利用されている．それらの設計には，ハードウェアだけでなくソフトウェアも同時協調的に設計開発を行っていくハードウェア/ソフトウェア協調設計（以後HW/SW協調設計）が必須である．現在一般的なHW/SW協調設計では，要求仕様の確定後はハードウェアとソフトウェアを分割し，ハードウェアをRTL(Register Transfer Level)，ソフトウェアをプログラムコードの段階で人手で別々に記述して，以降のプロセスは計算機により自動的に変換を行う．ハードウェアを動作レベルで設計する手法や，ハードウェアとソフトウェアを分割しないシステムレベルで設計する手法はまだ研究段階であり，実用化されている例は少ない．

現在，HW/SW協調設計の検証は主にシミュレーションにより行われているが，設計規模の拡大に伴い全てのテストパタン

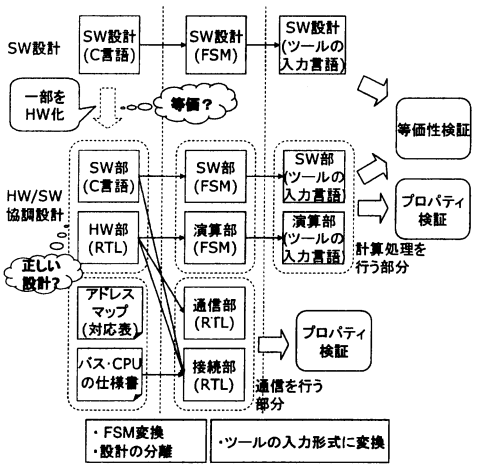


図 1　提案する検証フロー

をシミュレーションすることが事実上不可能な状況になり，シミュレーションでチェックされない場合，いわゆる corner case 問題が発生してきている．そこで，テストパタンによらない形式的検証でシミュレーションの漏れを補完する事が重要になる．形式的検証とは数学的に正しさを証明する検証手法である．現在のところ，先に述べたような現在一般的な HW/SW 協調設計に対して形式的検証を適用する手法は確立されていない．

以上のような背景を踏まえて本稿では，ハードウェア/ソフトウェア協調設計に対して形式的検証を適用する手法を提案する．検証対象はハードウェアが HDL(Hardware Description Language) を用いて RTL で，ソフトウェアが C 言語を用いてプログラムとして記述された HW/SW 協調設計である．

本稿では二種類の形式的検証手法について扱う．一つ目はプロパティ検証で，設計のモデルが決められた仕様 (プロパティ) を満たすかどうかを検証する形式的検証手法である．二つ目は等価性検証で，予め決められた意味で二つの設計が等価かどうかを検証する形式的検証手法である．等価性検証は，上記の HW/SW 協調設計と，その元となった C 言語で記述されたソフトウェアのみの設計とを対象とする．ソフトウェア設計の一部をハードウェア化する事により HW/SW 協調設計を生成する設計手法は一般的に用いられている．

本稿で提案する手法の特徴は，記述から自動的に抽象化されたモデルを生成し，検証を行う点にある．検証全体の流れを図 1 に示す．基本的に次の手順に沿って検証を行う．

（1） ハードウェアとソフトウェア間の通信をモデル化し，設計を計算処理を行う部分と通信を行う部分に分離する

（2） 計算処理を行う部分を，通信を抽象化した有限状態機械 (FSM:Finite State Machine) に変換し，ハードウェアとソフトウェアの表現を統一する

（3） 既存のプロパティ検証器を用いて，計算処理を行う部分と通信を行う部分を別々に検証する

本稿の構成は次の通りである．2 節で上記の手法の詳細を述べる．3 節で実験結果を示し，4 節でまとめと今後の課題について述べる．

## 2. 提案する手法

### 2.1 通信の抽象化

通常の HW/SW 協調設計では，ソフトウェア (SW 部) は CPU 上で動き，バスなどを介してハードウェア (HW 部) と接続されている．検証時にはこの HW 部− SW 部間の接続部分 (以後接続部) の動作を再現する必要がある．シミュレーションでは，シミュレーションモデルを利用して再現している場合が多い．しかし，形式的検証では検証する規模をできるだけ小さくしなくてはならない．そこで本研究では，HW 部と SW 部の通信方式が幾つかのパタンに分類できることを利用して，そのパタンごとの抽象化した動作のみを再現する．

一般的な HW 部− SW 部間の通信方式は Memory-mapped I/O と割り込みの二つに分類できる．Memory-mapped I/O とは，メモリ空間上の特定のアドレスに HW 部の資源 (ポート・レジスタ) が割り当てられており，SW 部がそのアドレスにアクセスする形で，そこに割り当てられた HW 部の資源にアクセスする通信方式である．割り込みとは，HW 部が割り込み信号を送ると，SW 部の実行が一時中断され，特定の処理が実行される通信方式である．本稿では Memory-mapped I/O についてのみ扱い，割り込みについては今後の検討事項とする．

Memory-mapped I/O の具体的な処理は次の様に行われる．

- 読み込み

(1)CPU が，ハードウェア資源が割り当てられたアドレスから値を読み込むソフトウェア命令を実行する (2)CPU がソフトウェアの動作をストップし，バスコントローラにアドレスに対応したバスへのアクセスの可否を問い合わせる (3) バスコントローラがバストラフィックを制御し，そのバスへのアクセスが可能になったら CPU にアクセス許可を出す (4)CPU がアクセス許可を受け取り，そのバスにアドレスと読み込み信号を送る (5) ハードウェア部の通信部分がそのバスからアドレスと読み込み信号を受け取り，アドレスをデコードして対応したハードウェア資源のデータをそのバスに返す (6) そのバスからアドレスに対応したハードウェア資源のデータが返ってくるので，CPU がそのデータを受け取り，ソフトウェアの動作を再開する

- 書き込み

(1)CPU が，ハードウェア資源が割り当てられたアドレスに値を書き込むソフトウェア命令を実行する (2)CPU がソフトウェアの動作をストップし，バスコントローラにアドレスに対応したバスへのアクセスの可否を問い合わせる (3) バスコントローラがバストラフィックを制御し，そのバスへのアクセスが可能になったら CPU にアクセス許可を出す (4)CPU がアクセス許可を受け取り，そのバスにアドレスと書き込む値と書き込み信号を送る (5) ハードウェア部の通信部分がそのバスからアドレスと書き込み信号と書き込む値を受け取り，アドレスをデコードして対応したハードウェア資源に値を書き込み，そのバスに書き込み終了通知信号を返す (6) バスから書き込み終了通知信号が返ってくるので，CPU がそれを受け取り，ソフトウェアの

| アドレス | アドレスへのポインタ | ハードウェア部のレジスタ | 備考 |
|---|---|---|---|
| 0x20000000 | HW_in1 | HW.In1 | 入力1 |
| 0x20000004 | HW_in2 | HW.In2 | 入力2 |
| 0x20000008 | HW_out | HW.Out | 出力 |
| 0x20000012 | HW_str | HW.Str | 演算開始 |
| 0x20000016 | HW_done | HW.Done | 演算終了 |

図 2 アドレスマップ

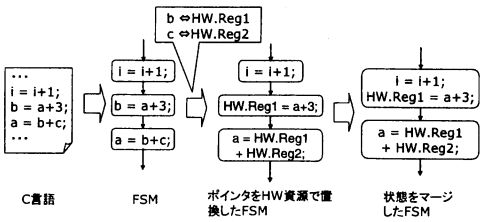


図 3 SW 部の変換

動作を再開する

ここで，設計の各部分を SW 部，接続部，HW 部に分類し，まとめる．SW 部はソフトウェア命令，接続部は CPU とバス関係の部分全て，HW 部はバスに接続されているハードウェアとなる．上記の動作を SW 部，接続部，HW 部のみで表現することで，それぞれの内部で完結した動作を無視することができ，通信を抽象化できる．抽象化された通信は次のようになる．

- 読み込み

(1)SW 部で HW 部の資源が割り当てられたアドレスから値を読み取る命令が実行される (2) 接続部が SW 部の処理を中断して，アドレスと読み込み信号を HW 部に送る (3)HW 部からアドレスに割り当てられた HW 部の資源のデータが返ってくるので，接続部が SW 部にそのデータを送り，SW 部の動作を再開する

- 書き込み

(1)SW 部で HW 部の資源が割り当てられたアドレスに値を書き込む命令が実行される (2) 接続部が SW 部の処理を中断して HW 部にアドレスと書き込む値と書き込み信号を送る (3) 書き込みが終了したら接続部が SW 部の動作を再開する

### 2.2 通信を抽象化した記述の作成

前項で述べた動作のうち接続部についてのものを RTL で記述した HDL を生成し，図 1 真中下の接続部の RTL 記述とする．これは後に述べる通信を行う部分の検証で用いる．その際の検証対象は HW 部の一部分なので，SW 部の制御については省くことができる．したがって，下記の内容を記述すればよい．(1)HW 部の資源が割り当てられたアドレスへのアクセスと特定のレジスタの値を対応付ける (例えば，00:アクセスなし/10:読み込み/11:書き込み) (2) その特定のレジスタの値が読み込み/書き込みの場合，バスの通信プロトコルに従い，HW 部のバスに接続される対応したポートに読み込み/書き込み信号とアドレスを送る

この際に必要な情報は，ハードウェア資源が割り当てられているアドレス，バスの通信プロトコルの仕様，HW 部のポートの対応であるが，通常，それぞれ SW 部のソースコード，CPU やバスの仕様書，HW 部のソースコードから得ることができる．

また，以上の通信が正しく行われると仮定すると接続部で行われるデータの移動を無視できるので，SW 部が直接 HW 部の資源にアクセスする事と等価となる．

そこで，アドレスと割り当てられた HW 部の資源の関係をユーザーが表にまとめ，アドレスマップとする (図 2)．アドレスマップで対応付けられているアドレスへのポインタと HW 部の資源を一つの共有変数として扱うことで，Memory-mapped I/O による通信を抽象化できる．また，アドレスマップには望ましい対応関係が記されているので，後に述べる通信を行う部分の検証でプロパティとして使用することができる．

### 2.3 計算処理を行う部分と通信を行う部分の分離

アドレスマップを用いて Memrory-mapped I/O の通信を抽象化する際には，HW 部内で通信のための処理 (アドレスのデコードなど) を行っている部分 (以後通信部) は必要ないので，それ以外の計算処理を行う部分 (以後演算部) と分離する．一般的な設計では通信部は別々のモジュールとなっている場合が多く，分離はモジュールを切り離す作業となり容易である．演算部と通信部が一体化している場合は演算部の抽出を行うが，この作業は記述の方式によって方法が異なるので，以下の指針に従って手動で演算部の抽出を行い，残りの部分を通信部とする．(1) バスや割り込み信号線などの接続部に接続されているポートを取り除く (2) そのポートに接続されているレジスタとアドレスマップに記されたレジスタ，またはアドレスマップに記されたポートに接続されているレジスタのみからなる状態遷移の記述を取り除く

### 2.4 FSM への変換

HW/SW 強調設計では，ハードウェアとソフトウェアとで記述言語や記述レベルが異なるため，そのまま形式的検証を適用することは難しい．そこで，本研究では設計を FSM に変換し表現を統一する．なぜなら，C 言語記述の FSM への変換は [1] や [2] などの研究で提案されている一文一状態で FSM に変換する手法が利用でき，RTL 記述の FSM への変換は同じ抽象度の変換であり容易だからである．具体的な方法を以下に述べる．

SW 部の変換例を図 3 に示す．変換手順は次の通りである．(1) [1] と同様に for 文の while 文への置換，ポインタから配列・変数への変換，構造体の分解，列挙体から変数への置換，標準出力等の余計な記述の削除など，C 言語の範囲内での変換を行い，if 文による分岐と while 文によるループと変数への代入のみで構成された等価な記述に変換する (2) 変換された記述の代入文を一文一状態で FSM に変換する．if 文や while 文での条件式は，そのまま状態遷移の条件式となる (3) アドレスマップを参照して，アドレスへのポインタを，そのアドレスに割り当てられている HW 部の資源で置換する (4) 同じ変数を用いているなどの因果関係のない状態同士を結合し，状態数を削減する

HW 部の変換は，演算部のレジスタを状態変数と見て，その遷移をそのまま FSM として記述する．なお，その前に，アドレスマップで割り当てられている HW 資源にポートが含まれ

る場合，それをレジスタに置換しておく．

以上で SW 部と HW 部の演算部の FSM を生成できた．この時 Memory-mapped I/O による通信は，HW 部のレジスタを共有変数として，その共有変数へのアクセスに抽象化できたことになる．

### 2.5 既存のプロパティ検証器での検証

前項までの手順を踏むと，この段階で，SW 部と HW 部の演算部は FSM で表現されており，HW 部の通信部と接続部は RTL で表現されている．

本研究では，この二つの部分を別々に既存のプロパティ検証器を用いて検証する．HW 部の通信部と接続部を FSM に変換しないのは，この部分の記述が主に HDL 特有の記述 (ポートやトライステートなど) で構成され，バスの通信方式によってはタイミングが大事になってくるからである．

RTL で表現された部分を検証するプロパティ検証器は，Solidify [3] などの RTL 記述を入力としたものを用いる．これらはハードウェア設計記述に対してプロパティ検証を適用でき，既に商用ツールも存在し実用化されている．

FSM で表現された部分を検証するプロパティ検証器は，Spin [4] などの状態記述を入力としたものを用いる．これらは並列プロセスの検証を目的として作られており，HW 部と SW 部で動作スピードの異なる HW/SW 協調設計の検証に適していると考えられる．入力は独自の状態記述言語を用いているものがほとんどなので，FSM をその入力言語に変換して検証する事になる．また，FSM は RTL と同じ抽象度なので，先に述べた RTL 記述を入力としたものを用いることもできる．クロック指定のプロパティを書きたい時などは有用である．

### 2.6 検証全体の流れ

前項までに述べた手法を用いた検証全体の流れを示す．

**プロパティ検証**

検証対象は図 1 の左中央の，HW/SW 協調設計である．まず，ユーザーが Memory-mapped I/O における対応関係を記したアドレスマップとバス・CPU の仕様書を用意する．

次に，SW 部の記述を 2.4 項で述べた手法を用いて FSM に変換する．また，2.3 項で述べた手法を用いて HW 部から演算部を分離し FSM に変換する．さらに，バス・CPU の仕様書，SW 部記述，HW 部記述から接続部の HDL 記述を生成し，HW 部の通信部と合わせて既存の RTL プロパティ検証器でアドレスマップに記述された対応関係をプロパティとして検証を行う．

最後に，SW 部の FSM と HW 部の演算部の FSM を既存のプロパティ検証器の入力言語に変換し，既存のプロパティ検証器を用いて検証を行う．その際のプロパティは，設計全体の演算について検証したい事をユーザーが記述したものとなる．

**等価性検証**

検証対象は図 1 の左上の C 言語による SW 設計と，その一部を分割しハードウェア化した図 1 の左中央の HW/SW 協調設計の間の等価性である．なお，先にプロパティ検証のフローを用いて HW/SW 協調設計の通信を行う部分の検証を終えておく．

まず，プロパティ検証と同様のフローで HW/SW 協調設計の計算処理を行う部分をプロパティ検証器の入力言語で記述した段階まで変換する．

次に，2.4 項で述べた手法を用いて HW/SW 分割前の SW 設計を FSM に変換し，プロパティ検証器の入力言語で記述する．この段階で，検証は同じ言語で記述された二つの設計間の等価性検証に帰着できたことになる．

最後に，既存のプロパティ検証器を用いて二つの記述の等価性を検証する．ここで，どのような意味での等価性を検証するかが重要であるが，分割前の SW 設計と分割後の HW/SW 協調設計では動作のタイミングが異なってくる．本研究の等価性検証の目的は動作の等価性の検証であるため，両者の入力と出力の関係が等価かどうかを検証すれば十分である．すなわち，二つの記述の動作が等価であるということは，“計算開始時点で両者の入力が等しければ，両者の計算が終了した時点で両者の出力が必ず等しい” というプロパティに置き換えることができる．したがって，上記のプロパティを既存のプロパティ検証器に入力し検証を行えば良い．

なお，SW 部と HW 部の記述から FSM への変換および，FSM から既存のプロパティ検証器の入力言語への変換は自動化できると考えられる．接続部の HDL 記述の生成および HW 部の演算部と通信部への分離は現段階では自動化する手法が定義できていないので手動で変換を行う．後者についてはあらかじめ分離された設計を対象にした場合は手動で変換を行う必要は無い．

## 3. 実験結果

前節で述べた手法を実際の設計に適用し実験を行った．なお，検証システムはまだ未実装であるため今回の実験では全ての変換を手動で行った．既存のプロパティ検証器での検証は次のマシン環境で行った．

OS:Linux(Fedora Core 2)
CPU:Intel Xeon 3.2GHz
Memory:2GB

### 3.1 例　　題

二つの例題に対して実験を行った．いずれの例題も等価な SW 設計と HW/SW 協調設計を用意した．SW 設計は ANSI-C の文法に従った C 言語記述である．HW/SW 協調設計は ATMEL 社 [8] の AT91R40807 という ARM7TDMI コアを内蔵した CPU を用いて実装した．SW 部は CPU で動く C 言語記述で，HW 部はバスに接続されているハードウェアモジュールである．HW 部－ SW 部間は Memory-mapped I/O を用いて通信している．

一つ目の例題は，入力値の階乗を出力値として返す小規模の例題である．設計規模は，SW 設計が 8 行 (C 言語)，HW/SW 協調設計の SW 部が 21 行 (C 言語)，HW 部が 96 行 (Verilog) である．HW 部は演算部 (28 行) と，通信部 (68 行) の二つの

モジュールから成っている．HW/SW 協調設計では，乗算の演算を HW 部で行うようにしている．なお，HW/SW 協調設計の HW 部の入出力データ幅を 16 ビットにしているので，入力値が 8 以下の正の整数でないと正しい結果が出ない．そのため，検証時に入力値は 8 以下の正の整数であるという制限を加える．

二つ目の例題は中規模の例題として，MSSG(MPEG Software Simulation Group) [7] の mpeg2 コーデック内の IDCT(逆離散コサイン変換) を計算するプログラムを使用した．設計規模は，SW 設計が 230 行 (C 言語)，HW/SW 協調設計の SW 部が 107 行 (C 言語)，HW 部が 378 行 (Verilog) である．HW 部は演算部 (123 行，116 行) と通信部 (119 行) の三つのモジュールから成っている．8 × 8 の 16 ビット整数行列に対して IDCT を実行する例題で，HW/SW 協調設計については，それぞれ 8 回ずつ呼び出される 50 行程度の二つの関数 (idctcol と idctrow) をハードウェアで計算を行うようにした．

### 3.2 実験環境

階乗を計算する例題に対しては，

- 計算が最後まで終了する
- 入力値が 8 以下なら出力値は 40320 以下となる
- 入力値が 2 以上なら出力値は 2 の倍数である
- 入力値が 3 以上ならば出力値は 3 の倍数である

の四つのプロパティを検証し，

- 等価な記述
- 不等価な記述 (1):HW/SW 協調設計において，SW 部で同期のためのループが一つ消えている
- 不等価な記述 (2);HW/SW 協調設計において，HW 部で行う積の演算が和の演算になっている

に対して等価性検証を行った．

IDCT を計算する例題に対しては，

- 計算が最後まで終了する，

というプロパティを検証し，

- 等価な記述，

に対して等価性検証を行った．

検証には既存のプロパティ検証器 Solidify [3]，Spin [4] を使用した．

Solidify は米国 Averant 社の商用 RTL 用プロパティ検証器で，入力言語は Verilog/VHDL である．通信を行う部分の検証にでは，そのまま記述を入力した．計算処理を行う部分の検証では，FSM を Verilog で記述し検証を行った．Solidify では有限のステップ数を監視する Bounded Model Checking を採用しているので，プロパティではステップ数を多めに見積もって指定した．

Spin は米国 Bell 研究所で開発されたプロパティ検証器で，入力は状態記述言語 Promela(Process Meta Language) である．計算処理を行う部分の FSM を Promela で記述し，検証を行った．SPIN では未初期化変数が 0 で初期化されてしまうため，非決定的な分岐により，入力値を可能性のある全ての値にすることで演算結果の検証を行った．

### 3.3 通信を行う部分の検証結果

それぞれの例題の HW/SW 協調設計についてアドレスマップを記述し，図 1 のフローに従って，HW/SW 協調設計の通信を行う部分のプロパティ検証を行った．プロパティ検証器は Solidify を用いた．

表 1 階乗を計算する例題の実験結果 (Solidify)

| プロパティ | 検証時間 [s] | 検証結果 | ゲート数 |
|---|---|---|---|
| 計算が最後まで終了する | 5 | Pass | 4301 |
| $input \leqq 8 \rightarrow output \leqq 40320$ | 260 | Pass | 16010 |
| $input \geqq 2 \rightarrow output\%2 = 0$ | 23 | Pass | 16010 |
| $input \geqq 3 \rightarrow output\%3 = 0$ | 295 | Pass | 16010 |
| 等価性検証:等価な記述 | - | - | 26532 |
| 等価性検証:不等価な記述 (1) | - | - | 26532 |
| 等価性検証:不等価な記述 (2) | 362 | 不等価 | 26532 |

表 2 IDCT を計算する例題の実験結果 (Solidify)

| プロパティ | 検証時間 [s] | 検証結果 | ゲート数 |
|---|---|---|---|
| 計算が最後まで終了する:抽象化前 | - | - | 20 万以上 |
| 計算が最後まで終了する:抽象化後 | 603 | Pass | 69932 |
| 等価性検証 (等価な記述) | - | - | 20 万以上 |

表 3 階乗を計算する例題の実験結果 (Spin)

| プロパティ | 検証時間 [s] | 検証結果 |
|---|---|---|
| 計算が最後まで終了する | 1 | Pass |
| $input \leqq 8 \rightarrow output \leqq 40320$ | 1 | Pass |
| $input \geqq 2 \rightarrow output\%2 = 0$ | - | - |
| $input \geqq 3 \rightarrow output\%3 = 0$ | - | - |
| 等価性検証:等価な記述 | 1 | 等価 |
| 等価性検証:不等価な記述 (1) | 1 | 不等価 |
| 等価性検証:不等価な記述 (2) | 1 | 不等価 |

表 4 IDCT を計算する例題の実験結果 (Spin)

| プロパティ | 検証時間 [s] | 検証結果 |
|---|---|---|
| 計算が最後まで終了する:抽象化前 | - | - |
| 計算が最後まで終了する:抽象化後 | 1 | Pass |
| 等価性検証 (等価な記述) | - | - |

階乗を計算する例題では，接続部の Verilog 記述は 65 行であった．アドレスマップで対応しているアドレスとハードウェア資源が同じデータ内容であるというプロパティで検証を行い，全ての組について 1 秒以内で正しいという結果を得た．

IDCT を計算する例題では，接続部の Verilog 記述は 333 行であった．同様のプロパティで検証を行い，全ての組について約 20 秒で正しいという結果を得た．

### 3.4 計算処理を行う部分の検証結果

図 1 のフローに従って，計算を行う部分のプロパティ検証と等価性検証を行った．プロパティ検証器として Solidify と Spin を用いた．Solidify を用いた時の結果を表 1，表 2 に，Spin を用いた時の結果を表 3，表 4 に示す．参考のため表 1，表 2 では Solidify で検証した時に生成されたゲート数を示した．なお，ゲート数はプロパティに関係した部分のみの数となっている．

階乗を計算する例題については，プロパティ検証では全てのプロパティについて，Solidify では数分以内に正しい結果を得ることができた．Spin では三番目と四番目のプロパティはモジュロ演算子を使用できないため検証できなかったが，それ以

外は 1 秒で正しい結果を得ることができた．等価性検証では，Spin を用いた場合は 1 秒で正しい結果を得たが，Solidify を用いた場合は検証に時間がかかり，不等価な記述については判例を一つ見つけた時点で検証が終了するので 6 分程度で終了しているが，等価なものに対しては 6 時間以上かかっても計算は終了しなかった．

なお，Solidify での検証時間が長いのは，内部で Verilog 記述をゲートレベルまで仮合成したうえで検証を行うためだと考えられる．

IDCT を計算する例題については，Solidify，Spin のどちらを用いた場合も，そのままでは設計規模が大きすぎて検証できなかった．そこで，プロパティ検証では，ここで検証するプロパティと演算結果は関係ないことを利用して，人手で設計の抽象化を行った．行列を格納した配列および配列の演算に使用する変数の幅を 1 に抽象化して検証したところ，Solidify では 10 分程度で，Spin では 1 秒程度で正しい検証結果を得ることができた．等価性検証については，演算内容と検証結果が関係してくるのでこれらの抽象化手法は適用できなかった．検証のためには有効な抽象化手法の適用などが必要である．

## 4. 本研究のまとめと今後の課題

### 4.1 本研究のまとめ

本研究では，ハードウェアが RTL で，ソフトウェアが C 言語で記述された HW/SW 協調設計に対してプロパティ検証を適用する手法と，HW/SW 分割前のソフトウェア設計と，分割後にハードウェアを RTL で記述した HW/SW 協調設計間の等価性検証を行う手法を提案した．なお，HW/SW 協調設計は，ハードウェアとソフトウェア間の通信が Memory-mapped I/O によって行われているものを対象とした．その方法として，

（1） ハードウェアとソフトウェア間の通信をモデル化し，設計を，計算処理を行う部分と通信を行う部分に分離する

（2） 計算処理を行う部分を，通信を抽象化した FSM に変換し，ハードウェアとソフトウェアの表現を統一する

（3） 既存のプロパティ検証器を用いて，計算処理を行う部分と通信を行う部分を別々に検証する

というアプローチに基いた具体的な検証フローを示した．さらに二つの例題に対して，二つのプロパティ検証器を用いて実験を行い，本研究で提案したフローにより実際に検証が行えることを示した．

### 4.2 今後の課題

今後は以下の内容について研究を進める．

- 自動検証システムの実装
- 抽象化手法についての調査と研究
- 多くのサイクルを監視するプロパティの分割: 検証可能な規模はプロパティで監視されるサイクル数にも依存するため，多くのサイクルを監視するプロパティを，全体で等価となるような少ないサイクルを監視するプロパティの集合に分割する事を考える．具体的には assume-guarantee 原理を用いる手法などが考えられる．
- 既存のハードウェア IP のシミュレーションモデルからの FSM 変換: HW/SW 協調設計において既存のハードウェア IP が利用される事を考えると，その IP の設計記述を入手できない場合が考えられる．そのような場合でも，その IP のシミュレーションモデルを入手することは通常可能であるので，シミュレーションモデルからの FSM への変換を行うことにより，本研究で提案した検証フローを実現することができる．
- 検証可能な規模を拡大するための等価性検証手法の検討: 本稿で提案した等価性検証手法は大きな例題に対しては適用できなかった．その原因の一つとして，HW/SW 協調設計では二つ以上のプロセスが並列動作しているため，それらの実行順序の組み合わせが膨大な数になってしまっている事が挙げられる．そこで，partial order reduction [5] の考え方を用いてその数を減らす事を考える．すなわち，プロセス間で通信が無い間の実行順序は処理には影響しない事を利用して，代表する一つの実行順序についてのみ検証を行う．現実的な数まで実行順序を減らすことができれば，並列動作記述を現実的な数の逐次動作記述の集合で置き換えることができる．

また，記号シミュレーションを利用した等価性検証手法 [6] を利用することができる．プロパティ検証器を用いた手法では入力値に対する出力値を網羅的にチェックするが，記号シミュレーションを利用した手法では記述を記号による式と見て検証を行うため，検証できる規模が大きい．

- 割り込みを扱うための有効な手法についての検討: 本稿で提案した FSM へ変換する手法では，SW 部の状態遷移のタイミングが文と文との間に限定されてしまうため，割り込みを無制限のまま扱うのは無理だと考えられる．割り込みを扱うためには，割り込みが起こるタイミングや回数に制限を設ける必要がある．その上で，さらに先に述べた partial order reduction を利用して状態削減を行えば，割り込みを用いた設計を検証できると考えられる．

### 文　献


[1] 松本剛史, 齋藤寛, 藤田昌宏, “C 言語動作記述の既存 RTL 用検証ツールを用いた検証の提案,” DA シンポジウム 2004 論文集, pp.241–246, 2004 年.

[2] G.J.Holzmann and M.H.Smith, “Aoutomating Software Feature Verification,” *Bell Labs Technical Journal*, 5(2), pp.72–87, 2000.

[3] Solidify, http://www.averant.com/products.htm

[4] G.J.Holzmann, “The model checker SPIN,” *IEEE Trans. on Software Engineering*, 23(5), pp.279–295, May. 1997.

[5] P.Godefroid ,“*Partial-Order Methods for the Verification of Concurrent Systems, An Approach to the State-Explosion Problem,” Lecture Notes in Computer Science* 1032, Springer-Verlag 1996.

[6] 松本 剛史, 齋藤 寛, 藤田 昌宏, “C 言語を対象とした記述間の差異に基づく効率的な等価性検証手法,” 電子情報通信学会技術研究報告, Vol.103, No.702, pp.61–66, 2004 年 3 月.

[7] MPEG Software Simulation Group, http://www.mpeg.org/MPEG/MSSG/

[8] Atmel Corporation, http://www.atmel.com/