

動作合成前後の設計記述に対する記号シミュレーションによる 形式的等価性検証の検討

松本 剛史[†] 小松 聡^{††} 藤田 昌宏^{††}

[†] 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: matsumoto@cad.t.u-tokyo.ac.jp, komatsu@cad.t.u-tokyo.ac.jp, fujita@ee.t.u-tokyo.ac.jp

あらまし システムレベル設計によって得られた動作記述と制約から、動作合成によって RTL 設計を導出する設計フローにおいては、動作合成前後の動作記述と RTL 設計の動作の等価性が保証されていることが重要である。しかし、そのための等価性検証手法、特に形式的手法は、実用化されているものはほとんどないのが現状である。そこで、本稿では、動作合成前後の設計の等価性を形式的に検証する一手法を提案し、その評価を行う。提案する手法では、RTL 設計を解析して抽出された、1 サイクルで行われる動作と、合成前の動作記述を記号的に実行して、形式的に等価性を検証する。RTL 設計から動作を抽出する作業では、いくつかの変換規則を適用することによって、RTL 設計中のビット結合などのビットレベル処理をワードレベルの処理に置き換え、個々の演算を解釈せずに検証が可能になるようにしている。また、いくつかの設計例に対して実験を行い、指定された適当な等価性を検証できることを確認した。
キーワード 等価性検証、形式的検証、動作合成、記号シミュレーション

An Approach to Equivalence Checking by Symbolic Simulation between Behavioral and RTL Designs

Takeshi MATSUMOTO[†], Satoshi KOMATSU^{††}, and Masahiro FUJITA^{††}

[†] Dept. of Electronics Engineering, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656 Japan

^{††} VLSI Design and Education Center, University of Tokyo
2-11-16 Yayoi, Bunkyo-ku, Tokyo, 113-0032 Japan

E-mail: matsumoto@cad.t.u-tokyo.ac.jp, komatsu@cad.t.u-tokyo.ac.jp, fujita@ee.t.u-tokyo.ac.jp

Abstract In system-level design, RTL designs are generated by behavioral synthesizers from behavior descriptions, and the equivalence between designs before and after behavioral synthesis should be guaranteed. However, few verification methods, especially formal ones, are available for this purpose. In this paper, we propose a method to formally verify the equivalence. In the method, behaviors executed in a clock cycle in RTL designs are extracted, and symbolically simulated with the original behavior descriptions. When extracting behaviors from RTL designs, several rules are applied to translate bit-level operations in RTL into word-level ones so that the equivalence can be verified uninterpretedly. Also, we show the experimental results on some design examples.

Key words Equivalence checking, Formal verification, Behavioral synthesis, Symbolic simulation

1. はじめに

SoC(System on a Chip) のような大規模な LSI 製品の設計においては、ソフトウェアとハードウェアの双方を考慮した最適なアーキテクチャ探索や高速なシミュレーションなどが可能なシステムレベル設計が採り入れられることが多くなってきてい

る。このシステムレベル設計の結果として得られる設計は動作記述であり、実現されるべき動作の実行順序が規定されている。この動作記述を与えられた制約の元で RTL(Register Transfer Level) 設計に変換する技術が動作合成である。RTL 設計では、どのレジスタの値を用いてどのような処理を行い、その結果をどのレジスタに保持するか、といったクロックサイクルごとの

処理が記述されている。

システムレベル設計から RTL 設計を自動的に短時間で導出するために、動作合成は必要不可欠であり、盛んに研究が行われており、また、現在では、いくつかの商用ツールとして利用可能になっている。一方、動作合成の結果として得られる RTL 設計が、合成された動作記述と等価であるかどうかを検証することも重要である。これは、合成前に十分に検証がされている動作記述との等価性を検証することによって、合成後の RTL 設計の正しさを保証する必要があるからである。しかし、動作合成前後の設計を対象にした等価性検証に関しては、まだ多くが研究段階であり、利用可能なものはほとんどないのが現状である。

本稿では、文献 [3] などでは我々が提案した C 言語記述に対する等価性検証手法を応用して、動作合成前後の設計に対する検証の一手法を提案し、いくつかの設計例に対する実験結果を示す。提案手法では、合成結果の RTL 設計の 1 サイクルで行われる動作を解釈して C 言語記述として表し、同じく C 言語記述として表された動作記述との間の等価性検証を行っている。本研究の目的は、動作合成ツールによって自動的に生成された RTL 設計記述に対する、動作合成ツールに依存しない検証手法を検討することである。提案する手法では、RTL 設計から抽出した動作を C 言語で記述する際に、いくつかの規則を適用することによって、できるだけビットレベルの処理を取り除き、ワードレベルで動作の等価性を検証する。これによって、演算を解釈せずに (uninterpreted に) 等価性を検証できるため、大きな計算量を要する決定手続き (decision procedure) を実行する回数を減少させ、効率的に検証を行うことができるようになっている。

本稿の構成は以下のようなものである。まず、第 2 節で、既存の研究と記号シミュレーションによる等価性検証を紹介する。次に、第 3 節では、提案する検証フローを述べる。ここでは、主に、RTL 設計中の動作をどのようにして C 言語記述に変換していくかを述べる。第 4 節で、いくつかの例題に対する実験結果を示し、第 5 節において、今後解決する必要がある問題点を考察する。最後に、第 6 節において、まとめを述べる。

2. 関連研究

2.1 動作合成前後の等価性検証手法

動作合成前後の動作記述と RTL 設計記述を対象とした等価性検証や応用可能な検証手法としては、いくつかの手法が提案されている。

文献 [1] では、動作記述と RTL 設計の EFSM (Extended Finite State Machine) モデルを作成し、その 2 つの EFSM 上の状態遷移列ごとの等価性を検証する手法を提案している。この手法では、2 つの EFSM の状態間に対応を取るために、動作合成ツールの情報を使っており、合成過程から独立した検証手法ではなくなっている。しかし、本稿では、合成前の動作記述と合成後の RTL 設計記述を入力として、合成環境からの情報に依らない検証を実現することを目的としている。

一方、文献 [2] では、並列化前後などのように、異なるスケ

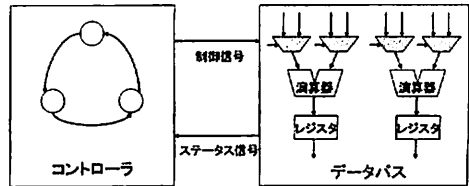


図 1 動作合成によって得られる回路

ジューリングがなされた動作記述の間の等価性検証手法が提案されている。この手法では、実行される動作自体に変更が加えられることがないという前提において、検証を行っているため、動作合成前後の等価性検証に適用することを考えると、動作合成の過程で行われ得る最適化などに対応できないと考えられる。本稿で提案する手法では、対象設計記述に対して記号シミュレーションを行い、必要に応じて解釈やより詳細な等価性判定手続きを適用するため、そのような最適化が行われた場合であっても等価性を検証することができる。

2.2 記号シミュレーションによる等価性検証

記号シミュレーションとは、設計記述を記号的に実行する手法であり、ある特定のテストボタンについて行う通常のシミュレーションとは異なり、あり得る全ての入力ボタンについてのシミュレーションに相当する検証が可能である。記号シミュレーションによる等価性検証では、記号シミュレーションの結果、等価であると証明されたものを Equivalence Class (EqvClass) にまとめることによって可能となっている [4]。この手法は、前述した文献 [1] においても利用されている。

なお、純粋な記号シミュレーションにおいては、各変数・演算を単に記号として扱うため、置換と uninterpreted function によって類推できない等価性は証明することができない。そこで、本研究では、必要に応じて、決定手続きを用いたより詳細な等価性判定を CVC [5] を用いて行うことによって、算術的な変更などがある場合でも正しい検証結果を得られるようにしている。

3. RTL 設計記述からの動作抽出による等価性検証手法

3.1 動作合成後の RTL 設計記述

本稿では、動作合成前後の動作記述と RTL 設計記述との間の等価性検証手法を検討する。そこで、本節では、動作合成の結果として、どのような RTL 設計が得られるかを概説する。

動作合成では、あらかじめ与えられた演算器の集合に対して、合成する動作に必要な演算を演算器資源の制約の範囲内でマッピングしていくことによって行われる (アロケーション)。このとき、制約を満たすように、どのクロックサイクルでどの演算が実行されるかも決定される (スケジューリング)。結果として、与えられた演算器に加えて、合成した動作が実行できるようにレジスタやマルチプレクサを結合したデータベースと、データベースを制御するためのコントローラが得られる (図 1)。

合成されたデータベースは、各サイクルで使用される演算器に対して、どのレジスタからデータを読み込み、演算結果をどの

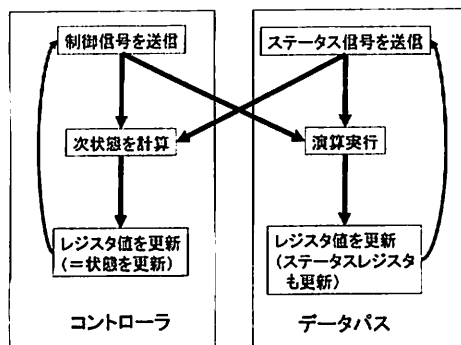


図2 図1のRTL設計で実行される動作

レジスタに書き込むかをコントローラから制御することが可能になっている。これは、必要に応じて、1つの演算器を共有するためである。そのため、演算器やレジスタの前には、マルチプレクサが必要となり、それらはコントローラが生成する制御信号によって制御される。なお、データベース中の演算器・マルチプレクサは、事前にネットリストとしてライブラリに登録されたものを使うため、論理合成されることはない。

一方、コントローラは、有限状態機械 (Finite State Machine : FSM) として実現されており、データベースを制御するための制御信号を生成する。このコントローラでは、現在の状態とデータベース中の比較器などが返すステータス信号から、次状態を計算し、新たな制御信号を生成する。

以上を踏まえると、動作合成されたRTL設計の1サイクルでの基本的な動作は、図2のように考えることができる。まず、直前のサイクルで更新されたレジスタ値によって、コントローラFSMにおいて現状態が決定される。これによって、対応する制御信号が生成され、データベースへと送られる。そして、この制御信号に従って、データベース内で演算が行われ、必要があれば、比較器などからステータス信号がコントローラに送られる。コントローラでは、現状態とデータベースから送られたステータス信号に基づいて、次状態が計算される。最後に、この次状態とデータベースの必要な演算結果は、レジスタ値を更新することによって保持される。実際に、シャープ社の開発した動作合成ツールBACHで合成した結果は、前述した図2のような動作をするRTL設計となっている。

3.2 動作の抽出とC言語記述への変換

RTL設計の動作を抽出し、C言語のようなプログラミング言語を用いて記述することができれば、動作合成前の動作記述との等価性検証は、動作記述どうしの検証として行うことができる。提案する変換手法を適用することによって、動作合成ツールに依存することなく、合成前後の設計記述のみを入力として検証を行うことができる。本稿では、合成前の動作記述はC言語で、合成後のRTL設計記述は完全同期回路であり、Verilog-HDLで記述されているものとして、以降の説明を進める。本稿では、非同期回路や多相クロックを含む設計は対象としない。合成前の動作記述に関しては、動作合成可能なものであるため、ポインタや再帰呼び出しなどが制限された記述と

なっている。

同期回路の基本的な動作は、

(1) 現在のレジスタ値に対して、組合せ回路において処理を行う

(2) 処理結果をレジスタに書き込む

の繰り返しである。そのため、それぞれの動作をHDLの文法に従って、動作記述に変換して、1サイクルで行われる動作を求めることができる。RTL設計記述では、「どのレジスタの値を用いて、どのような処理を行い、どのレジスタに結果を書き込むか」が記述されているため、この変換作業は以下のような手順で行うことが可能である。

(1) 現在のサイクルのレジスタ値の決定

現在のレジスタ値は、基本的には直前のサイクルで書き込まれたものと同じ値である。しかし、リセット信号が入力された場合などは、対応する特定の値を用いる必要がある。

(2) 入力信号の決定

各サイクルで入力される値が決まっている場合はその値を用いる。不定である場合は、記号値として検証を行う。サイクルの違いについては、サイクルを表す変数を導入することによって、区別が可能である。

(3) 組合せ回路の実行

現在のレジスタ値と入力値を用いて、組合せ回路で行われる処理を計算する。このとき、出力信号も更新される。

(4) レジスタ値の更新

組合せ回路の処理結果のうち、必要なものをレジスタに書き込む。

これらをwhileループで繰り返すことによって、RTL回路の動作を表現することができる。

図3は、簡単なRTL回路での変換例を示したものである。図の(a)がRTL回路を、(b)が対応するHDLをそれぞれ表している。このRTL回路に対応する動作をC言語で表現したものが(c)である。このとき、組合せ回路部分について(この例ではwire文で記述されている)、(b)のHDL記述の表記順に変換すると誤った動作を得ることに注意が必要である。RTL設計記述では、回路構成要素とその結線が記述されており、表記順には意味がないためである。そのため、設計記述から組合せ回路を再現してから、動作記述に変換する必要がある。

3.3 ビット演算に関する変換規則の導入

本稿で検証する等価性は、ビット幅を考慮しない動作の等価性である。合成前の動作記述においては、Cベース設計言語や合成用のプラグマによって、変数のビット幅が指定されているが、動作自体はワードレベルの演算によって記述されている。一方、合成後のRTL設計では、合成前の動作記述にはないビット演算を含んでいる可能性がある。加えて、RTL設計における演算は、全てビット幅を考慮して行われる。これらのRTL設計中のビットレベルの演算を、意味を解釈して、ワードレベルの処理へと適切に変換することができれば、等価性検証をワードレベルで高速に行うことができる。以降では、データベース中の演算器とビット演算に対する変換手法を紹介する。

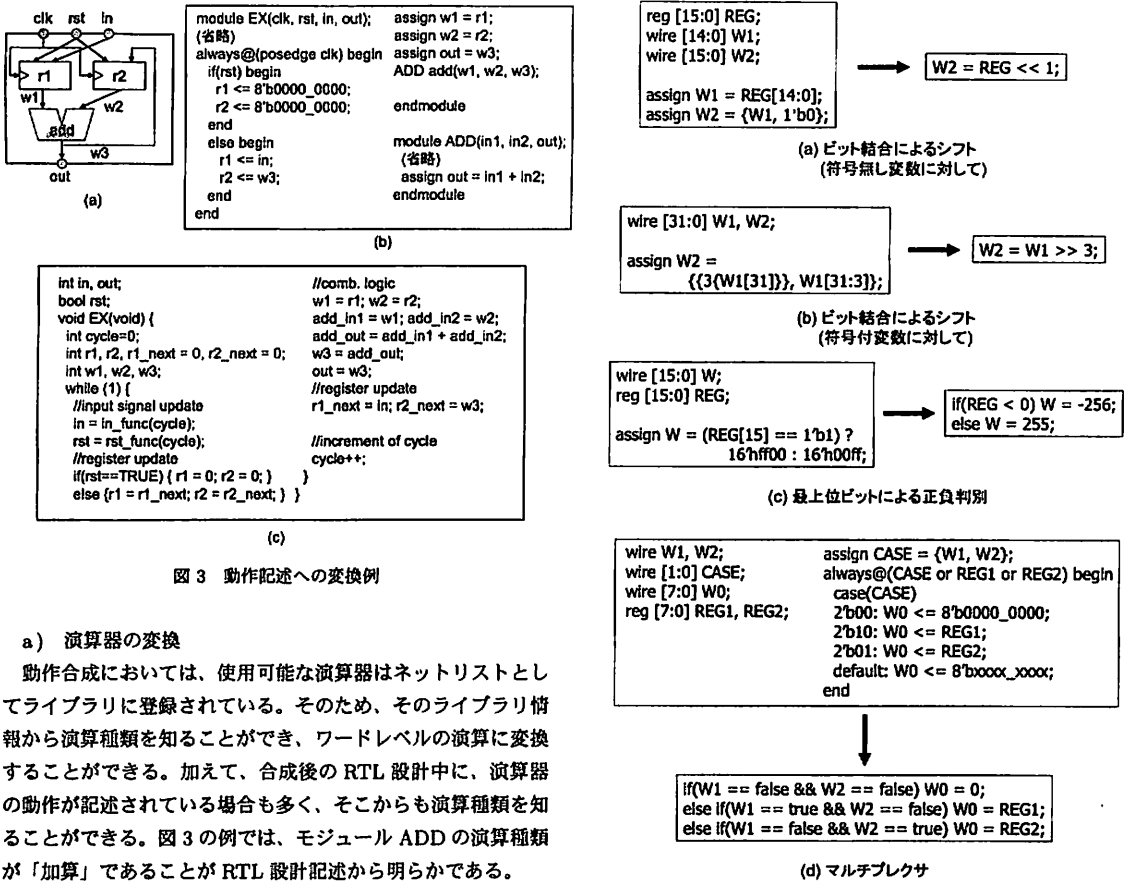


図 3 動作記述への変換例

a) 演算器の変換

動作合成においては、使用可能な演算器はネットリストとしてライブラリに登録されている。そのため、そのライブラリ情報から演算種類を知ることができ、ワードレベルの演算に変換することができる。加えて、合成後のRTL設計中に、演算器の動作が記述されている場合も多く、そこからも演算種類を知ることができる。図3の例では、モジュールADDの演算種類が「加算」であることがRTL設計記述から明らかである。

b) ビット演算の変換

動作合成後のRTL設計では、ビット結合・分割やマルチプレクサによる制御を含むことがほとんどである。それらに対しては、その意味を解釈することによって、対応するワードレベルの処理へと変換する。いくつかの具体的な例を図4に示す。実際の動作合成においては、使用する動作合成ツールに対して、このような変換規則をあらかじめ定義しておくことによって、ほとんどの場合、変換が可能であると考えられる。

3.4 等価性と制約の指定

動作記述とRTL設計の等価性を検証するためには、どのタイミングで、動作記述中のどの変数とRTL回路中のレジスタ・ポートが等価であるべきか、を指定する必要がある。動作記述とRTL設計での間の等価性の指定法については、文献[6]などで議論がなされているが、十分に実用的な手法については研究段階であると言える。

また、一般的に、動作記述ではリセット信号やインターフェースプロトコルが考慮されていないが、合成後のRTL設計では、それらが設計中に含まれている場合が多い。そこで、RTL設計が、与えられた動作記述と等価な動作をするための制約も指定する必要がある。この制約としては、「リセット信号が1サイクル目で1になり、2サイクル目以降は0である」「リクエスト信号が1になってから2サイクル後に入力データが利用可能になる」などが挙げられる。

本稿では、「入力したデータに対して、あらかじめ決められたサイクル数で何らかの処理を行い、最後に結果を出力する」ようなデータ演算モジュールの検証を対象として、以下に示す指定方法を用いる。その他の設計に対応できるような、より一般的な等価性や制約の指定方法については、今後の課題とする。

• 入力データの等価性 $\text{Eqv_Input}(n, in_{beh}, t, in_{rtl})$

動作記述における入力変数 in_{beh} の n 個目の値とRTL設計における t サイクル後の入力信号 in_{rtl} が等価である

• 出力データの等価性 $\text{Eqv_Output}(n, out_{beh}, t, out_{rtl})$

動作記述における出力変数 out_{beh} の n 個目の値とRTL設計における t サイクル後の出力信号 out_{rtl} が等価である

• 等価であるためのRTL設計に対する制約条件 $\text{Constraint}(t, sig, val)$

動作記述と等価な動作をするためには、RTL設計の入力信号 sig が t サイクル後に val でなければならない

これらの指定法を用いることによって、合成後のRTL設計のモジュールのレイテンシやスループット、入出力が与えられるタイミング、などを考慮した等価性検証が可能である。

3.5 記号シミュレーションによる検証

最後に、与えられた2つの動作記述、合成前のC言語記述と

合成後の RTL 設計から変換した C 言語記述、に対して、指定された制約の下で記号シミュレーションを行い、指定された等価性が満たされるかどうかを判定する。

記号シミュレーションは、設計記述を記号的にシミュレーションするため、有限長のパスにしか適用することができない。そのため、ループ構造がある場合、その繰り返し回数があらかじめ決定されている必要がある。データ演算モジュールを検証する場合には、多くの場合、ループの繰り返し回数は定数であり、検証することが可能である。また、変換された RTL 設計は、while ループで記述されており、その繰り返し 1 回に 1 クロックサイクルが対応している。そのため、記号シミュレーションは、指定された等価性を検証するために必要なサイクル数まで行うことになる。

4. 実 験

実験として、逆離散コサイン変換 (IDCT) と楕円フィルタ (ELLIP) の例題を用いた等価性検証を行った。それぞれの例題の詳細は、表 1 に示す通りである。表中の入力数・出力数は、与えられた動作記述を 1 回実行するために必要な入力・出力のデータ数を表している。また、動作記述は C 言語で、RTL 設計は HDL で記述したものの行数を記述量をして記した。なお、IDCT 例題では Verilog-HDL を、ELLIP 例題では VHDL を用いている。RTL 設計の FF 数と回路規模 (NAND ゲート換算) は、Synopsys 社の論理合成ツール Design Compiler によって、0.18 μ m プロセスのライブラリを用いて、100MHz で動作するように合成した結果に基づいたものである。

IDCT 例題は、ともに人手で動作合成を行った例題であり、行計算と列計算の間で演算器の共有を行わない場合 (IDCT1) と行った場合 (IDCT2) の 2 通りの例に対して検証を行った。動作記述では、入出力データは一度に与えられるが、RTL 設計では、モジュール外のメモリから 1 つずつ送信・受信されるため、適当なタイミングで入力データが与えられるように、また、適当なタイミングでの出力データと等価性を比較するように指定する必要がある。

一方、ELLIP 例題は、シャープ社の動作合成ツール BACH を用いて、自動合成した例題である。この例題では、それまでの実行結果と新たな 1 つの入力に対して、1 つの出力を出す動作が繰り返し実行される。そのため、 n 回目の繰り返しを実行するときの実質的な入力数・出力数は n となる。ELLIP 例題においては、動作記述中にプラグマを記述することによって、以下の 3 通りの合成を行った。

- (ELLIP1)
ハンドシェーク通信あり・パイプライン実行なし
- (ELLIP2)
ハンドシェーク通信なし・パイプライン実行なし
- (ELLIP3)
ハンドシェーク通信なし・パイプライン実行あり

4.1 RTL 設計の変換

第 3 節で述べた手法に従って、人手によって変換を行った。ビットレベルの処理をワードレベルの対応する処理に変換する

表 1 実験に用いた例題

例題	入力数	出力数	動作記述 記述量 (行)	RTL 設計		
				記述量 (行)	FF 数	回路規模 (ゲート数)
IDCT1	64	64	160	984	197	2971
IDCT2	64	64	160	654	197	2957
ELLIP1	1	1	86	846	215	3310
ELLIP2	1	1	86	832	1027	20220
ELLIP3	1	1	86	896	539	11543

表 2 RTL 設計の動作を抽出した C 言語記述の規模

例題	HDL 記述	変換後の C 言語記述
IDCT1	984	1115
IDCT2	654	734
ELLIP1	846	603
ELLIP2	832	594
ELLIP3	896	615

ために、以下の変換規則が必要であった。

- ビット結合 $\Rightarrow$ 左シフト (図 4(a) に相当)
- ビット結合 $\Rightarrow$ 右シフト (図 4(b) に相当)
- ビット幅拡張 $\Rightarrow$ 変化なし
- ビット幅縮小 $\Rightarrow$ 変化なし
- 最上位ビットが true $\Rightarrow$ 負数
- 最上位ビットが false $\Rightarrow$ 非負数
- 下位 3 ビット $\Rightarrow$ 8 で割った余り
- マルチプレクサの case 文記述 $\Rightarrow$ 制御信号での条件分岐 (図 4(d) に相当)

変換後の動作は C 言語で記述した。その記述量を表 2 に示す。これより、記述量は概ね HDL 記述と同程度になることが分かる。

4.2 記号シミュレーションによる検証

IDCT 例題

IDCT 例題では、どのタイミングで入力・出力が与えられるかが分かっているため、それに基づいて等価性の指定を行った。例えば、in32 という入力は、RTL 設計では開始から 50 サイクル目で入力されるため、Eqv_Input(1,in32,50,in32) となる。動作記述は繰り返し実行されないため、動作記述における全ての入出力値は 1 つ目のものを扱うことになる。入力の等価性が 64 個、出力の等価性が 64 個必要であった。制約としては、1 サイクル目でリセットをする (Constraint(1,rst,true)) ことと、それ以降、動作が終わる 181 サイクル目までリセットされない (Constraint(t,rst,false) for $2 \leq t \leq 181$) ことを与えた。

はじめに、2 つの C 言語で記述された動作記述を記号シミュレーションしたところ、検証が 24 時間経っても終了しなかった。これは、IDCT 計算の最後に、出力データの丸め込みを行う部分で、 2^{64} もの実行パスがあるためであった。そこで、この丸め込みを両記述から除いて検証を行ったところ、IDCT1 で 86 秒、IDCT2 で 70 秒で等価性を示すことができた。また、記述を等価でないように変更した場合にも、ほぼ同じ検証時間で等価でないことを示すことができた。

ELLIP 例題

ELLIP1 例題では、入力データがハンドシェーク通信に従って与えられる。そのため、RTL 設計が正しく動作するように、プロトコル信号を制約として与えて検証を行った。また、入出力の等価性を適切に指定して、1 回の繰り返しの検証を行った。

与えられた 2 つの C 言語記述に対して、記号シミュレーションを行ったところ、等価性を示すことができなかった。これは、合成の際に、 $a * 3 = (a < 1) + a$ という変換が行われていることが原因であった。そこで、記号シミュレータで、この変換の等価性を認識できるようにして、再度、検証したところ、全ての例題で 1 秒以内に等価であることを示すことができた。続いて、ELLIP2 と ELLIP3 に対して、等価性の指定を繰り返し 2 回分として検証を行った。このとき、パイプライン化によってスループットが変化しているため、2 回目の繰り返し実行の出力に対して、ELLIP2 では 12 サイクル目で、ELLIP3 では 11 サイクル目で等価性を指定した。この場合も、1 秒以内で等価であると示すことができた。

5. 考 察

本節では、実験から得られた解決すべき課題とそれに対する現時点で考えられる解決方法について考察を与える。

5.1 帰納的な手法の導入

ELLIP 例題のように、以前の実行結果に依存されながら、繰り返し実行をするようなモジュールでは、提案手法では有限の繰り返し回数までしか等価性を証明することができない。そこで、そのような場合に、帰納的に等価性を証明する手法が必要であると考えられる。つまり、いくつかのレジスタと全入力を等価と仮定して、繰り返し 1 回分の記号シミュレーションを行い、指定したレジスタと全出力が等価になることを確認することによって、任意の繰り返し回数に対しても等価性を証明することができる。

5.2 部分的な検証の必要性

IDCT 例題では、条件分岐による実行パス数の増大のために、現実的な時間で検証することができなかった。そのため、部分的に記号シミュレーションを行って検証をする必要があると考えられる。ただし、合成環境に依存しない検証では、動作記述と RTL 設計の間で対応を取ることが難しい。そこで、今後の予定として、ランダムシミュレーションをベースにした内部等価点候補 (potential internal equivalent point) の特定を考えている。

5.3 リセットや通信プロトコルを考慮した検証

今回の実験では、データ演算モジュールを対象として検証を行い、入力データ以外の入力信号については、制約という形で完全な定数として与えた。そのため、どのようなタイミングでリセット信号や通信プロトコルに関する信号が与えられたとしても等価である、という証明には至っておらず、正しいリセット信号・プロトコル信号の与え方の一例に対してのみ等価性を検証したことになる。これを解決するためには、通信プロトコルとデータ演算部分を分けて検証する必要があると考えられる。

6. ま と め

本稿では、動作合成前後での動作の等価性検証を行うための手法を提案して、評価を行った。提案手法では、合成後の RTL 設計記述から動作を抽出してワードレベルの動作へと変換し、合成前の動作記述と検証するため、高速な検証が可能である。実験を通して、一定の条件内で、モジュール単位で実際に動作合成前後の設計記述の等価性が証明できることを確認した。今後の課題としては、動作記述と RTL 設計間で対応を取ることによって、部分的な検証を導入し、より高速に検証を行うことがあり、その手法を検討していく予定である。

謝 辞

本研究で手法の評価に用いた動作合成前後の設計記述は、株式会社 半導体理工学研究センターの援助により提供されたものである。また、本研究は東京大学大規模集積システム設計教育研究センターを通し、シノプシス株式会社の協力で行われたものである。

文 献

- [1] 竹中, 向山, 若林, 中田, 前川, 山際, “動作合成前後の動作記述と RTL 記述の論理等価性検証,” 第 17 回路とシステム軽井沢ワークショップ論文集, pp.555-560, 2004 年 4 月.
- [2] S. Abdi and D. Gajski, “Functional Validation of System Level Static Scheduling,” *Proc. of Design, Automation and Test in Europe*, pp.542-547, Mar. 2005.
- [3] 松本, 齋藤, 藤田, “C ベース高位設計における等価性検証フレームワークと反例解析手法の提案,” 第 18 回路とシステム軽井沢ワークショップ論文集, pp.557-562, 2005 年 5 月.
- [4] G. Ritter, *Formal Sequential Equivalence Checking of Digital Systems by Symbolic Simulation*, PhD thesis, Darmstadt University of Technology and Universite Joseph Fourier, 2000.
- [5] A. Stump, C. Barret, and D. Dill, “CVC: a Cooperating Validity Checker,” *Proc. of 14th International Conference on Computer-Aided Verification*, pp.500-504, 2002.
- [6] M. Fujita, “Equivalence checking between behavioral and RTL descriptions with virtual controllers and datapaths,” *ACM Trans. on Design Automation of Electronic Systems*, Vol.10, Issue 4, pp.610-626, Oct. 2005.