ワードレベル論理式の充足可能性判定問題を利用したシステムレベル設計デバッグ支援手法

原田　裕基[†]　　西原　佑[‡]　　松本剛史[§]　　藤田昌宏[§,◇]
[†]東京大学大学院工学系研究科電気系工学専攻　　[‡]東京大学大学院工学系研究科電子工学専攻
[§]東京大学大規模集積システム設計教育研究センター
[◇]科学技術振興機構 戦略的創造研究推進事業 CREST

1　はじめに

シミュレーションやモデル検査によるハードウェアの検証では、設計中にバグが存在する場合、反例を得ることができる。しかし、設計デバッグにおいては、得られた反例を解析してバグの位置を特定する作業に特に時間が費されている。本研究では、与えられた仕様を満たさない値を出力する入力パタンの情報を元に、設計におけるバグの位置を特定する手法を提案する。

2　関連研究：ゲートレベルにおけるSATソルバを用いたバグ位置特定手法

文献[1]では、ゲートレベル回路において、シミュレーションやモデル検査で得られた反例の入力パタンからバグの位置を特定する手法が提案されている。この手法では、デバッグ対象の回路に、バグが存在する可能性のある信号線にマルチプレクサを挿入する。このマルチプレクサは、セレクト信号の値が0の場合は元の設計の信号線の値を出力し、1の場合には自由変数を出力する。マルチプレクサ挿入後の回路では、各マルチプレクサに入力される自由変数は回路の外部入力となる。次に、このマルチプレクサを挿入した回路における各信号線が満たすべき制約式を導出する。この制約式と、与えられた反例の入力パタン、仕様から求められる正しい出力パタンの論理積を求め、その充足可能性を判定する。充足可能な解(各信号線の値)が得られた場合、セレクタ信号値が1となったマルチプレクサが挿入された信号線を修正することにより、与えられた反例を正すことができる。マルチプレクサを挿入したどの信号線を修正しても、与えられた反例を正すことができない場合には、判定結果が充足不可能となる。

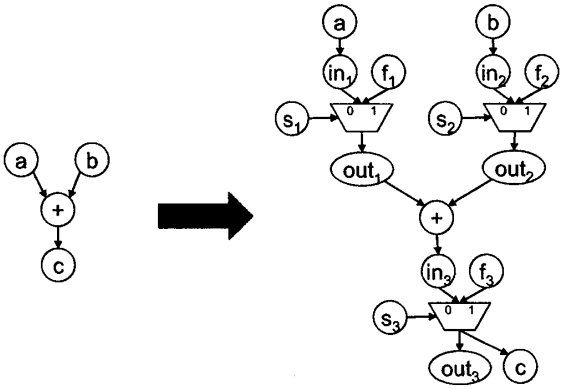

図1: DFGへのマルチプレクサの挿入

3　DFGを用いた高位設計記述におけるバグ位置特定手法の検討

3.1　概要

本節では、前節で述べた手法をシステムレベル設計記述に拡張する形でシステムレベル設計記述におけるバグ位置特定手法を検討する。ゲートレベルでは信号値をビットレベルで扱っていたが、これをそのままシステムレベル設計に適用しようとすると制約式が大きくなりすぎてしまう。この問題を解決するため、本研究では信号値(変数値)をワードレベルで扱う。システムレベル設計において、変数値をワードレベルで扱うための設計の表現方法として、Data Flow Graph(DFG)を用いる。DFGの各エッジ上に擬似的にマルチプレクサを挿入し、そこから制約式及び入力パタンによる制約式の積を求め、これをワードレベル変数を含む論理式の充足可能性判定を行うツールに解かせることによりバグの箇所を特定する。

3.2　提案手法のフロー

まず、ダイナミックスライシング[2]により、設計記述のうち仕様を満たさない出力が得られる入力パタンを用いた場合に実行される代入文列からDFGを生成する。この入力パタンはシミュレーションやモデル検査によって得られたものを想定する。DFGによって、反例となる入力パタンを実行した際の変数間の依存関係を表現することができる。

次に、このDFGに対して図1のように各エッジ上

A Debugging Support Method for System-Level Designs by Using Word-Level Satisfiability Problem
Hiroki HARADA[†]　Tasuku NISHIHARA[‡]
Takeshi MATSUMOTO[§]　Masahiro FUJITA[§,◇]
[†]Department of Electrical Engineering and Information Systems, The University of Tokyo
[‡]Deparment of Electronic Engineering, The University of Tokyo
[§]VLSI Design and Education Center, The University of Tokyo
[◇]CREST, JST

Copyright © 2010 Information Processing Society of Japan.
All Rights Reserved.

にマルチプレクサを挿入する。このマルチプレクサは、セレクト信号 (s_i) の値が0の場合は元のエッジの値を、1の場合は外部入力として追加された自由変数値を出力する。

続いて、このマルチプレクサ付きのDFGに対し、各変数が満たすべき制約を導出する。この制約、与えられた入力パタン、仕様から求められる出力による制約の3つ条件の論理積を求め、その充足可能性を判定する。充足可能な解 (条件を満たす変数値) が得られた場合、その解においてセレクト信号値が1となったエッジに相当する変数へ代入される値を変更することにより、与えれた反例を仕様に従った出力に正すことができる。

3.3 実験結果

例題としてSpecC言語によって記述された標準偏差を求めるシステムレベル設計を用いた。この例題に対し、エッジの接続関係の誤りに関するバグを1箇所挿入し、提案手法を適用した。

実験の結果、合計で6つの解が得られた。これは、バグの可能性のあるエッジが6つあるという事を表現している。元々マルチプレクサは10箇所に挿入されており、バグの位置の候補を10箇所から6箇所に減らすことができたと言える。

3.4 ゲートレベルにおける手法を単純拡張した手法の問題点

提案手法の問題点として、バグ修正方法について有用な情報が得られないという点が挙げられる。この手法では、どのエッジをどの値に修正すれば仕様に従った正しい出力が得られるか、を得ることができる。一方で、この情報だけでは具体的に設計のどの部分をどう直せばいいのか分からないので、実際にデバッグを行う事は難しい。

この問題は、バグの修正方法が多数存在することが原因と考えられる。この問題を解決するため、本稿では本節で提案した手法にバグのモデルを導入した手法を提案する。典型的な記述誤りをバグのモデルとして定義することによって、修正候補の数を限定することにより、修正箇所とともに修正方法も明確にすることが可能になり、デバッグに有用な情報が得られると考えられる。

4 バグモデル及びそれを用いたバグ位置特定手法の検討

バグモデルを定義することは、コード上でどのような記述誤りを想定するのかを決定することに対応する。本節では、二つの記述誤りに対するバグ位置特定手法を検討する。

4.1 演算における引数の誤り

これは、演算に用いる変数を間違えるという誤りを示す。例えば、$c = a + b$ と書くべきところを $c = a + d$ と書いてしまったという例が挙げられる。

このバグモデルに対する実験を、SpecCによる設計例の例題に対して行った。この例題に対して、演算における引数の誤りによるバグを4種類用意し、それぞれに対して提案手法を適用した。実験の結果、修正候補総数70個に対し、最大で修正候補を2個にまで絞る事に成功した。最も多く修正候補が残ってしまったものでも6個にまで絞れており、バグの候補の数を大きく減らすことに成功したと言える。また、求められた解において、それぞれのバグを修正するためには記述のどの部分をどう直せば仕様を満たす出力を得られるかが明確に示されており、この点もバグモデルを導入した利点と言える。

4.2 演算子の誤り

これは例えば、$c = a + b$ と書くべきところを $c = a * b$ としまったという例が挙げられる。今回は演算子として四則演算のみを想定した。

このバグモデルを用いて4.1と同様の実験を行った。その結果、修正候補総数48個に対し、全ての例題で1個にまで候補を絞ることに成功した。これは、演算内容が大きく変わるため各変数の値に与える影響が大きい事が理由であると考えられる。

5 まとめ

本稿では、システムレベル設計におけるバグ位置特定手法を提案した。これは、ゲートレベルにおけるバグ位置特定手法をシステムレベル設計に拡張したものであるが、バグの修正方法の候補が多すぎるためにデバッグに有用な情報が得られなかった。これに対しては、バグモデルを定義する手法を提案した。バグモデルとしては、分岐条件の誤り、演算の引数誤りの2つを提案し、これらに関して実験により評価を行い、バグモデルとして定義されているバグが設計中に存在する場合には、提案手法によりバグの修正候補の数を絞り、デバッグを行う事が可能であることを示した。

参考文献

[1] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, "Fault Diagnosis and Logic Debugging Using Boolean Satisfiability," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 24, No. 10, pp. 1606–1621, Oct. 2005.

[2] X. Zhang, H. He, N. Gupta, and R. Gupta, "Experimental evaluation of using dynamic slices for fault location," In *Proc. international symposium on Automated analysis-driven debugging*, Vol. 6, pp. 33–42, Sep. 2005.

Copyright © 2010 Information Processing Society of Japan.
All Rights Reserved.