Vol.2016-SLDM-176 No.5
2016/5/11

特定の状態遷移経路に注目した論理装置のシステム検証

藤田 智久，高橋 隆一

広島市立大学　情報科学研究科　情報工学専攻

ソフトウェアとハードウェアの協調設計が行われる専用プロセッサにおいては特にシステムレベルでの検証が市場を逃さないために重要になっている．アサーションベース検証（ABV）が普及しつつあるが手法は必ずしも確立されていない．設計作業の正しさを設計者自身が確認するための検証は上位レベルで行う方が効率が良い．ソフトウエアとハードウエアの両者によって実現される性質に対して ABV を適用することでシステム検証が行える．本研究ではソフトウェアとハードウェアから構成されるダイクストラ法専用プロセッサの状態遷移経路を追うことで問題個所の特定を試みた．ダイクストラ法の状態遷移系を抽象化することで経路を組織的に見出すことを試みた．

System verification of digital systems by tracking particular path of state transitions

Tomohisa Fujita，Ryuichi Takahashi

Faculty of Information Sciences，Hiroshima City University

System level verification is one of the most important subjects especially for application specific instruction set processors (ASIPs) design, where hardware / software codesign is used for catching the market window. The way for assertion based verification (ABV) has not been established well. For checking the correctness of the design, higher level verification is fast and easy. System level verification could be done by applying ABV to the property implemented by using both software and hardware We investigated the way to find bugs by tracking the state transition path of an ASIP for Dijkstra's algorithm. Abstraction for Dijkstra's algorithm could be a way to identify the paths formally.

1 はじめに

設計の大規模化，複雑化に伴い ASIP に対するソフトウェア/ハードウェア協調設計が注目されている．ASIP は特定の用途向けの専用プロセッサである．協調設計の課題の 1 つに検証がある．検証にはアサーションベース検証（ABV）と形式的検証がある．ABV が普及しつつあるが手法は必ずしも確立されていない．論理装置はソフトウェアとハードウェアによって実現されている．ソフトウェアとハードウェアの両者によって実現される性質を検証することでシステム検証が行える．何らかの方法で問題個所を特定する必要がある．本研究ではシステムの状態遷移経路を追うことで問題個所を特定することを試みた．論理装置の状態はプログラムカウンタ PC，プログラムで用いている変数，中央処理装置内部のレジスタやフラグがとる値の直積によって実現される．

ダイクストラ法を実行可能な 3 段パイプラインコンピュータを設計した．システムの状態遷移の追跡に ABV を適用した．システムの動作に問題を見出し，当該部分のソフトウェアとハードウェアを調べることで，ソフトウエアのバグを見出すことができた．

形式的検証には定理証明型とモデル検査がある [1]．設計はクリプキ構造としてモデル化される．検証しようとする性質は時相論理で記述される．時相論理としては CTL と LTL が代表的である．モデル検査では状態爆発がひとつの課題となっている．抽象化 [2] は LTL において対象とする設計がとる状態の爆発を抑えるひとつの手法である．ダイクストラ法の状態遷移系を抽象化することで経路を組織的に見出す手法を模索した．

2 設計検証

設計が仕様を満たしているかを検証することを設計検証という．検証にはシミュレーションによって検証を行うアサーションベース検証 [3][4] と，形式的仕様記述やプロパティなどを用いてシステムの妥当性を数学的に証明する形式的検証 [1] がある．

2.1 アサーションベース検証

アサーションベース検証 (ABV) では，アサーションで記述した性質をシミュレーションによって検証する．アサーションとは正しい設計として満たされるべき，または起こってはならない性質の記述である．設計者自身がアサーションを使用することによって別の視点から設計を見直すことができる．

2.2 形式的検証

形式的検証では，仕様を形式的に記述し，設計が仕様を満たしているかを調べる．検証に成功すれば設計が仕様を完全に満たしているという結論が得られる．形式的検証には定理証明型とモデル検査がある．

定理証明型では，検証したい性質を定理として記述し，設計が満たしている性質を公理として記述する．定理証明システムを用いて公理から定理を導くことで設計の検証を行う．

モデル検査には様相論理 [5] が用いられる．述語論理では常時成り立つ性質しか扱えない．可能世界を考えてどの世界では何が成り立つかを扱うのが様相論理である．コンピュータの状態を扱う様相論理は特に時相論理と言われる．検証対象のモデルは状態集合，遷移関係，初期状態の集合，各状態での原子命題への真偽の割り当てからなるクリプキ構造 S で表される．モデル検査では時相論理式 P がモデル S で成り立つかを判定する．時相論理としては CTL と LTL が代表的である．CTL は与えられた状態から始まる実行経路の可能性を扱う．LTL は個々の経路に対して成り立つ性質を扱う．

3 システム検証

3.1 想定する開発環境

本研究では設計チームとは別に検証チームが存在する開発体制を想定する．設計チームは仕様書をもとに設計を行う．検証チームは仕様書をもとに設計が満たすべき性質を確認する項目を網羅的に用意して設計の完了を待つ．検証

チームでの検証は網羅的な ABV によって行われる．検証チームに渡す設計の品質を高めておくために，設計チームも ABV を用いることが考えられる．本研究で研究対象としているシステム検証は設計チームが設計品質を高めるために用いる ABV である．上位レベルで効率よく実施できることが期待される．従来はソフトウェアまたはハードウェアが別個に検証されていた．本研究では，ソフトウェアとハードウェアを同時に検証するシステム検証を試みた．

3.2 システム検証

ソフトウエアとハードウエアの両者によって実現される性質を対象とする ABV によってシステム検証が行える．拡張ユークリッドの互除法 [6] に頻出する計算式を直接実行できる命令を備えた専用プロセッサが設計された．余りが $ax + by$ の形であり続けるという性質は当該命令と機械語のソフトウエアの両者によって実現されていた．当該性質を表現するアサーションによる ABV によってシステム検証が行えた [7]．他にユークリッド整域であるというアサーションを用いることも考えられる．

ソフトウエアとハードウエアの両者によって実現される性質を対象としても，実際にはどの部分にバグが存在するのかを特定する手段が必要になる．本研究ではソフトウエアとハードウエアの両者によって実現されるシステムの状態遷移系列に注目することで問題個所を特定することを試みた．

4 設計したプロセッサ

本研究ではダイクストラ法 [8] を実行できる 3 段パイプラインコンピュータを設計した．

4.1 ダイクストラ法

ダイクストラ法は辺の重みがすべて非負の場合に，重みつき有向グラフの単一始点の最短路を求めるアルゴリズムである．V を節点の集合，E を枝の集合とする．グラフ G(V,E) の各枝 $e \in E$ に重み $l(e)$ がつけられている．始点を s，終点を t，各節点 v には $\lambda(v)$ という仮の距離をラベルとして割り当てるものとする．X をすでに距離が求められている節点の集合，Y を各時点で注目する節点の集合とする．ダイクストラ法のアルゴリズムは以下の Step1〜6 のアルゴリズムで実行される．

Step1 $\lambda(s)$ を 0, s を除く各節点 v の $\lambda(v)$ を ∞ とおく．

Step2 $X \leftarrow \emptyset, Y \leftarrow V$ とおく．

Step3 $u \in Y$ で $\lambda(u)$ の値が最小の u を探す．

Step4 $u = t$ ならば終了．

Step5 u から枝 e があるすべての v について，$v \in Y$ であって，
$\lambda(v) > \lambda(u) + l(e)$ なら
$\lambda(v) \leftarrow \lambda(u) + l(e)$.

Step6 $X \leftarrow X \cup u, Y \leftarrow Y - u$ として $Step3$ へ

4.2 データ構造

本研究では，節点と枝をリンクリストを用いて実現した．図 1 に節点のデータ構造を示す．

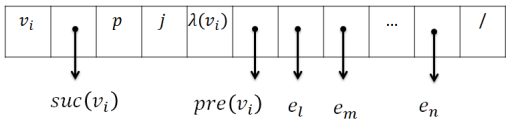

図 1: 節点のデータ構造

図 1 の v_i は節点番号．$suc(v_i)$ は v_i の次の節点番号へのポインタ，p は始点か終点かそれ以外かを表す，j は選ばれているかいないかを表す，$\lambda(v_i)$ はラベル，$pre(v_i)$ は直前の節点へのポインタ，e_l〜e_n は v ら出ている枝へのポインタ, そして/はストッパーを表している．

図 2 に枝のデータ構造を示す．

図 2 の e_p は枝番号．$suc(e_p)$ は e_p の次の枝番号へのポインタ，v_q は e_p の始点へのポインタ，v_r は e_p の終点へのポインタ, $l(e_p)$ は枝 e_p の重みを表している．

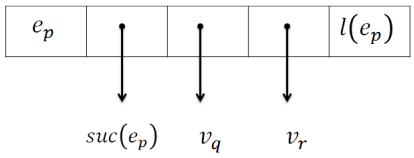

図 2: 枝のデータ構造

4.3 マイクロアーキテクチャ

図 3 に本研究で設計したコンピュータのブロック図を示す．

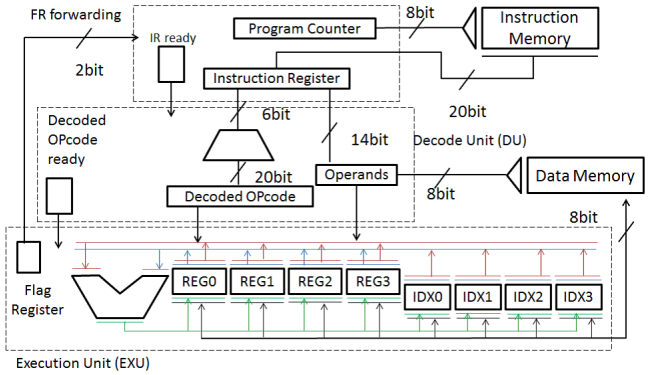

図 3: 本研究で設計したパイプラインコンピュータのブロック図

プリフェッチユニット，デコードユニット，実行ユニットからなる 3 段パイプラインとなっている．クロックは単相である．設計したコンピュータはハーバードアーキテクチャである．ハーバードアーキテクチャでは，命令とデータでビット幅が異なるメモリを用いる．節点と枝はリンクリストを用いて実現されるため，アドレスを指示するレジスタとしてインデックスレジスタを持っている．

5 状態遷移系

5.1 クリプキ構造

Q を状態集合，R を状態の遷移関係，I を初期状態，α を各状態での原子命題への真偽の割り当てとして，$S = \{Q, R, I, \alpha\}$ をクリプキ構造という [2]．原子命題とは，各状態において真偽が一意に定まる命題である．設計対象をクリプキ構造で表し，時相論理式で書かれた仕様を満たすかを調べる検証をモデル検査という．

5.2 CTL と LTL

検証すべき性質は時相論理で記述される．CTL は与えられた状態から始まる実行経路の可能性を扱う．LTL は個々の経路について成り立つ性質を扱う．経路を追跡するには LTL が適していると考えられる．

LTL でのモデル検査は全ての経路において性質が成り立つとき真を返し，それ以外では偽を返す．

5.3 LTL 式

LTL 式で使われる時相記号は X, G, F, U がある．X は次で，G は常に，F はいつか，U がまでという意味である．p,q を原子命題とすると，pXq は p が成り立った次の状態で q が成り立つことを意味する．Gp は常に p が成り立つという意味である．Fp はいつか q が成り立つという意味である．pUq は q が成り立つまで p が成り立っているという意味である．LTL は個々の状態遷移経路での性質を記述する．

6 特定の経路に注目したシステム検証

図 4 に本研究が検証対象としたダイクストラ法のフローチャートを示す．当該フローチャートはソフトウエアとハードウエアの両者によって実現されている．図の L1，L2，L3…はフローチャートのどの部分を実行しているかを表し，プログラムカウンタ PC がとる値に対応している．状態遷移はプログラムで用いている変数，中央処理装置内部のレジスタやフラグがとる値の組合せの変化だが，PC の値をたどることで代表される．

期待通り動作しなかったため，L2 の節点の選択と最小であるラベルの計算結果が一致するか調べた．一致しなかったためラベルの選択かラベルの計算にバグがあることが分かった．ラベルの計算は L5，L6，L7 で行われているので L5 → L6 → L7 → L5 と L5 → L6 → L5 の経路に ABV

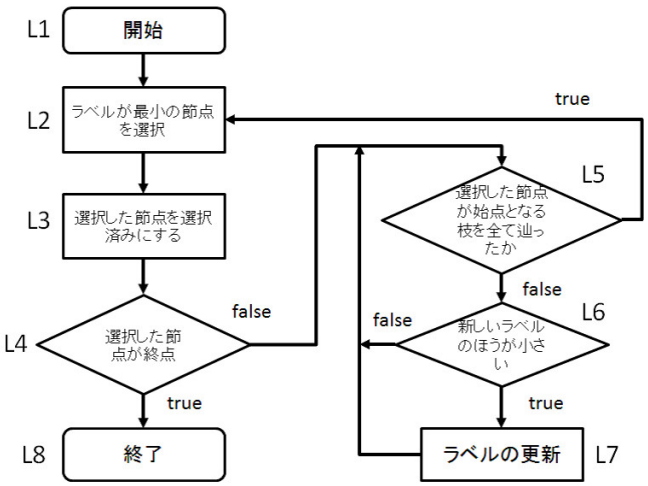

図 4: フローチャート

```
label_compare: assert property
   (
     @(posedge CLOCK1)
     PC==L5 |-> $stable(MEM[IDX1+4]) ||
           $past(MEM[IDX1+4])>MEM[IDX1+4] || $past(PC==L4)
   );
```

図 5: アサーション 1

```
label_renewal: assert property
   (
     @(posedge CLOCK1)
     PC==L7 |-> MEM[IDX1+4] > MEM[IDX0+4]+MEM[IDX2+4]
   );
```

図 6: アサーション 2

を適用した．図 5 に ABV に用いたアサーションを示す．図 5 のアサーションは L4 → L5 か L7 → L5 か L6 → L5 かの経路で正しく動作しているかを記述している．図 5 のアサーションでは 3 つの経路のいずれかという性質を or でまとめることにより，同時に調べる工夫を行った．図 5 のアサーションは満たされなかった．L5 → L6 の経路に問題は見出せなかった．L6 → L7 の経路に問題見つかった．アサーションを図 6 に示す．当該部分のソフトウェアとハードウェアを調べた結果，ソフトウェアのバグが見つかった．

7 抽象化による経路の特定

LTL には状態爆発を抑える工夫として抽象化がある [2]．

7.1 LTL における抽象化の手法

抽象化とはある性質のもとに複数の状態を 1 つの状態にまとめる手法である．抽象化することで検証するモデルの状態数を減らし，状態爆発を抑えることができる．LTL における抽象化にはデータマッピング法と述語抽象化法がある．データマッピング法は，プログラムに現れる各変数ごとに抽象化する方法である．例として，変数が整数型としたとき正の数，負の数，0 という 3 種類に抽象化することができる．述語抽象化法は，複数の変数で表される述語を真偽で分けることによって抽象化する方法である．例として，整数型の変数 x,y があるとき，述語 $x > y$ が真か偽の 2 種類に抽象化することができる．

7.2 抽象化写像

設計したシステムのクリプキ構造を $\mathcal{C} = \{C, R, I, \alpha\}$，抽象化したモデルのクリプキ構造を $\mathcal{A} = \{A, \bar{R}, \bar{I}, \bar{\alpha}\}$ とする．$\beta : C \to A$ が以下を満たすとき抽象化写像という．

$$\beta(I) \subseteq \bar{I}$$

$c, \acute{c} \in C$ に対し，$cR\acute{c} \Longrightarrow \beta(c)\bar{R}\beta(\acute{c})$

$c \in C$ に対し，$\alpha(c) = \bar{\alpha}(\beta(c))$

抽象化写像 β で写像されたクリプキ構造 $\mathcal{A}$ で成り立つ性質はもとのクリプキ構造 $\mathcal{C}$ でも成り立つ．

7.3 ダイクストラ法に対する抽象化

抽象化は通常は設計に対して用いられる．本研究ではダイクストラ法を抽象化し，設計したシステムの状態遷移経路を組織的に見出すことを試みた．述語抽象化を用いた．枝の重みを不等式で表した．図 7 に本研究で用いた述語抽象化の例示に用いるグラフを示す．s は始点であり，t が終点である．枝の重みは節点間の距離を表している．

図 8 に述語抽象化に用いる枝の重みについての制約を例示する．この制約は s → b → a → t

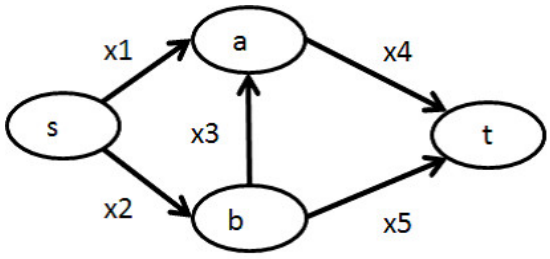

図 7: 本研究に用いたグラフ

$$x2 < x2+x3 < x1 < x2+x3+x4 < x1+x4 < x2+x5$$

図 8: 枝の重みの制約例

という最短路に対応している．図 4 に示したフローチャートでは L1 → L2 → L3 → L4 → L5 → L6 → L7 → L5 → L6 →…L8 という遷移に対応している．L5 → L6 → L7 → L5 というループを 5 回巡っている．述語抽象化に用いた制約を満たす枝の重みを具体的に与えて ABV を用いることで，実際にループを 5 回巡るかなどを調べることで，システムの検証が行える．抽象化写像の像は図 7 のグラフに対するダイクストラ法での可能な分岐の仕方の数だけ存在する．

8　まとめと今後の課題

ソフトウエアとハードウエアの両者によって実現される性質に注目することでシステム検証が行えることを示した．システムとして実現されている動作の状態遷移の特定部分を追跡するアサーションベース検証で問題個所を特定できることを示した．実現しようとしているアルゴリズムに対して抽象化を行うことで状態遷移系列を組織的に見出す手法を提案した．問題個所特定のための抽象化手法を見出すこと，形式的検証への応用などが今後の課題である．

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ケイデンス株式会社の協力で行われたものである．

参考文献

[1] 米田友洋，梶原誠司，土屋達弘：ディペンダブルシステム-高信頼システム実現のための耐故障・検証・テストの技術，共立出版（2005）．

[2] 田辺良則，高井利憲，高橋孝一：抽象化を用いた検証ツール，コンピュータソフトウエア，Vol.22，No.1，pp.2-44（2005）

[3] Haryy D. Foster, Adam C. Krolnik, David J. Lacey 著，東野輝夫，岡野浩三，中田明夫　訳：アサーションベース設計　原書 2 版，丸善 (2004)

[4] Ben Cohen，Srinivasan Venkataramanan，Ajeetha Kumari 著，三橋明城男，朽木順一，茂木幸夫，小笠原敦，明石貴昭　訳：SystemVerilog アサーション・ハンドブック，丸善 (2006)

[5] 萩谷昌巳，西崎真也：論理と計算のしくみ，岩波書店 (2007)

[6] 高橋隆一：電子機器設計に活かすディジタル代数学，科学情報出版 (2013)

[7] 垣鍔千幸 :” 拡張ユークリッド互除法命令対応のコンピュータにおける代数的性質に注目したアサーションベース検証，” 平成 25 年度広島市立大学卒業論文 (2014)

[8] 平田富夫：C によるアルゴリズムとデータ構造，科学技術出版 (2002)