Vol.2018-ARC-229 No.27
Vol.2018-SLDM-182 No.27
2018/1/19

多クロックサイクルのワードレベルの関係を含むハードウェアアサーションの自動抽出

Automated Assertion Mining for Word-Level Relations over Multiple Clock Cycles

宮本 真実 [1], 濵口 清治 [2]
Mami Miyamoto, Kiyoharu Hamaguchi

島根大学大学院総合理工学研究科
Interdisciplinary Graduate School of Science and Engineering
Shimane University, Matsue, Shimane, 690-8504 JAPAN

[1]E-mail: s169513@matsu.shimane-u.ac.jp
[2]E-mail: hama@cis.shimane-u.ac.jp

Abstract

In this paper, we focus on the mining method for automatically generation of temporal assertions from execution traces of hardware designs. The existing methods can handle assertions based on LTL formulas or PSL, and many of them can represent word-level relations such as inequalities, additions, and so on as atomic propositions. However, such relations are searched only within a clock cycle. They cannot extract a property such that two values at inputs are added, and its result appears two clock cycles later at an output. We propose a method to extract relations over multiple clock cycles between variables as atomic propositions by analyzing execution traces and to generate assertions including the relations. Our method can also efficiently generate assertions by extracting frequent relations between atomic propositions over multiple clock cycles as propositions, that is, conjunctives of atomic propositions. The experimental results demonstrate the feasibility of the proposed method.

1. はじめに

ハードウェア設計検証におけるアサーションは，設計が満たすべき論理条件のことである．これらは主に，アサーションベース検証に使用される．アサーションベース検証は，機能検証のための一般的な手法である．アサーションは，シミュレーションやフォーマル検証のチェッカーとして使われる．通常，アサーションは手動で定義されるが，アサーション定義は多くの時間と高い専門性を必要とするプロセスである．したがって，アサーションの手動定義に対する補完的な方法として，アサーション自動抽出のための多くの手法が提案されている．抽出されたアサーションは，設計の変更の確認や，エラーの発見，文書化などに使用できる．さらに，設計の入出力信号間の関係をアサーションとして抽出することで，他の設計に再利用することができる．

アサーションマイニングの手法には，静的手法と動的手法の 2 種類がある．[1][2]のような静的手法は，設計の形式的解析に依存しているため，大規模設計で使用するのは難しい．これに対して，近年，シミュレーショントレースからアサーションをマイニングする動的手法が研究されており，よりコンパクトで理解が容易な，エラー検出能力の優れたアサーションを抽出することに成功している．第 2 節で示すように，ハードウェア設計の実行トレースからのテンポラルアサーションの自動抽出のための様々なマイニング手法が提案されている．しかし，既存の手法では「あるクロックサイクルで入力xとyの値が加算され，その結果が 2 クロックサイクル後に出力zに現れる」というようなプロパティを抽出することはできない．このようなプロパティは，SVA (System Verilog Assertion) における内部変数を使用して記述でき，SVA では上記のプロパティは「$(1, v_x = x, v_y = y)$ |-> ##2$(z = v_x + v_y)$」となる．しかし，既存のアサーションマイニングの手法では LTL (Liner Temporal Logic) や PSL (Property Specification Language) を使用しているため，このようなプロパティを記述することは容易でない．

この問題を解決するために，本報告では，実行トレースを解析することで変数間の多クロックサイクルにまたがるワードレベルの関係を原子命題として抽出し，このような関係を含むアサーションを生成する手法を提案する．また，提案手法は，多クロックサイクルにまたがる原子命題間の頻出する関係を，命題すなわち原子命題を結合子 “∧” で接続した連言として抽出することにより，アサーションを効率的に生成することができる．このプロセスによって，過剰に制約された条件部を持つアサーションのマイニングを防ぐことができる．提案手法の特徴を以下に示す：

- 多クロックサイクルにまたがるワードレベルの変数間の関係を原子命題として抽出し，そのような関係を含むアサーションを生成する．
- 頻出パターンマイニングを利用し，多クロックサイクルにまたがる原子命題間の頻出する関係を命題として抽出することで，過剰に制約された条件部を持つアサーションのマイニングを防ぐ．
- 多クロックサイクルにわたって値が保持されることの多いディジタル回路の特性を考慮し，値の変化点に着目して原子命題抽出とアサーションマイニングを行う．

2. 関連研究

テンポラルプロパティを得るためのハードウェアアサーションマイニングについて，多くの動的手法が提案されている．[3]では，req-ack 信号の関係やステートマシンのプロトコルのような典型的なパターンをデータマイニング技術によってマイニングしている．[4]では，繰り返しパターンを検出する手法が提案されている．[5][6]は，インタフェースプロトコルのパターンをマイニングするための手法を示している．[7]は，2 つの変数を含むテンポラルパターンのテンプレートを仮定し，信号値の変化に注目した高速なマイニングアルゴリズムを提案し，得られたアサーション集合にフォール

ト解析を適用している．[8][9]では，決定木アルゴリズムと静的解析に基づく手法を用い，「$always(antecedent \rightarrow consequent)$」と表現される，より一般的なテンポラルパターンが抽出された．$antecedent$と$consequent$は，0 回以上$next$演算子が適用された原子命題の連言である．この研究は，[10]では，トランザクションレベルモデルを対象とし，結論部に不確実な長さの遅延を持つアサーションをカバーするように拡張された．

上記の研究では，原子命題としてブール変数またはバイナリ変数が考慮されている．2 ビット以上のビット幅を持つ変数間の関係，すなわちワードレベルの特徴を扱う方法は[11][12]で提案されている．[11]では，「$x = constant$」という形の原子命題を抽出し，アサーションを構成している．[12]では，Daikon[13]を使用し，「$z = a + b$」や「$x > y$」のようなワードレベルの算術関係を抽出している．[12]の研究は，[14][15][16]で改良され，与えられた一般的な LTL テンプレートと一致するアサーションをマイニングする方法が提案されている．

しかし，既存の手法では，算術関係を含む原子命題は，単一クロックサイクル内のみで探索され，多クロックサイクルにまたがるワードレベルの関係を原子命題として扱うことができる方法は知られていない．本報告では，この問題を改善する手法を提案する．

3. 準備

この節では，提案手法に必要な定義およびマイニングされるアサーションについて説明する．

以下では，モデルMを RTL 設計と仮定する．VはMの変数の集合であり，アサーションはV上でマイニングされる．Vの変数は手動で与えられ，本報告ではこれらをMの任意の信号とする．V中の変数のビット幅は 1 ビット以上で，主入出力信号，内部信号のどちらでも良い．また，$V_a \subseteq V$を，アサーションの結論部で値が割り当てられる変数の集合として与える．例えば，$z = a + b$という関係の変数zはV_aに含まれる．

定義 1.（実行トレース） *シミュレーションインスタントの有限シーケンス$t_0 t_1 t_2 \dots t_{n-1}$と，変数 V 上のモデル M が与えられたとき，M の実行トレースは有限シーケンス$T = V_0 V_1 V_2 \dots V_{n-1}$である．ここで，$V_i$はシミュレーションインスタント$t_i$における変数の値の集合であり，$V_i(v)$は$t_i$での$v \in V$の値である．*

定義 2.（原子命題） *原子命題は，論理結合子を含まない論理式である．*

本研究では，先行研究で扱われる$a = True, a = 5, a > 5, a > b, z = a + b$のような 1 クロックサイクルで成り立つ原子命題に加え，$z[2] = a[0] + b[0]$のような，変数間の多クロックサイクルにまたがる関係を原子命題として扱う．このような原子命題に含まれる変数は，多クロックサイクルにわたって値が参照される．その中で最も早いクロックサイクルで参照される変数にクロックサイクル 0 を割り当てる．$x[i]$はiクロックサイクル後のxの値である．つまり，$z[2] = a[0] + b[0]$は「あるクロックサイクルで入力aとbの値が加算され，その結果が 2 クロックサイクル後に出力zに現れる」という意味である．本報告では，以下のような原子命題を考える：**(i)** 値の割り当て（例：$a = True, a = 5$），**(ii)** 2 変数間のビットシフト演算を表す関係$z[i] = a[j]\ op\ n$ $(op = \ll \text{ or } \gg)$ $(i \geq j, z \in V_a, n \in \mathbb{N})$，**(iii)** 3 変数間の関係$z[i] = a[j]\ op\ b[k]$ $(op = +, -, \times, /, bitwize\text{-}and, bitwize\text{-}or)$ $(i \geq j, j \geq k, z \in V_a)$．これらの原子命題テンプレートは，ユーザによって変更することができる．

定義 3. *上記の原子命題(ii) $z[i] = a[j]\ op\ n$，原子命題(iii) $z[i] = a[j]\ op\ b[j]$は，クロックサイクル$min(i,j,k)$で成立すると定義する．*

命題についても同様に定義できる．例えば，表 1 において，$(p[0] = True) \wedge (q[1] = True)$は$t_0$，$z[2] = x[0] + y[0]$は$t_2$で成立する．

定義 4.（命題） *命題は，論理結合子による原子命題の結合である．原子命題も命題である．*

命題トレースは，実行トレースと同様に定義される．さらに，提案手法では，[15]と同様の以下のタイムウィンドウを利用する．

定義 5.（タイムウィンドウ） *実行トレースまたは命題トレース$\tau = A_0 A_1 A_2 \dots A_{n-1}$と，$0 \leq t_i \leq t_j \leq n-1$である 2 つのシミュレーションインスタント$t_i$と$t_j$が与えられたとき，タイムウィンドウ$TW[i,j]$は，$t_i$と$t_j$の間の連続したサブシーケンス$A_i A_{i+1} \dots A_j$である．*

表 1. 実行トレースの例

time	p	q	x	y	z
t_0	True	False	2	2	1
t_1	False	True	1	1	2
t_2	False	False	2	3	3
t_3	False	True	0	4	2
t_4	True	False	0	4	5

##2

(p[0]=True)∧(q[1]=True)

z[2]=x[0]+y[0]

実行トレースで成立する，多サイクルにまたがるワードレベルの変数間の関係を原子命題として抽出し，2 つの命題間の含意の関係をアサーションとしてマイニングする．例えば，「$p = True$が成立し，次のサイクルで$q = True$が成り立ったならば，その次のサイクルのxとyの値が加算された結果がさらに 2 サイクルあとにzと等しくなる」というプロパティを含む実行トレースを表 1 に示す．これは，SVA において「$(p = True)$ ##1 $(q = True)$ |−> ##1 $(1, v_x = x, v_y = y)$ ##2 $(z = v_x + v_y)$」と記述できる．SVA では，「always」は暗黙的に仮定されている．以下では，上記のプロパティを簡単に「$(p[0] = True) \wedge (q[1] = True) \rightarrow$ ##2 $(z[2] = x[0] + y[0])$」と記述する．「##n」は，n クロックサイクルの時間の経過を表す．「−>」は，通常の論理包含であり，$(p[0] = True) \wedge (q[1] = True)$と##2 $(z[2] = x[0] + y[0])$が同じクロックサイクルt_0で成り立つことを意味する．x, yが参照されるクロックサイクルで，$z[2] = x[0] + y[0]$が真とみなされることに注意．上記の式は，$(p[0] = True) \wedge (q[1] = True)$が成立すると，2 クロックサイクル後に$(z[2] = x[0] + y[0])$が成立することを意味する．本報告では，「##$n$」を結論部のオフセットと呼ぶ．

本研究では，アサーションは事前に設定された長さの各タイムウィンドウ内でマイニングされる．このように，アサーションのクロックサイクルの最大数を決定することで，時間的に離れすぎている命題間の関係をマイニングするのを避けることができる．また，命題も各タイミングウィンドウ内で抽出する．

さらに，多クロックサイクルにわたって値が保持されることの多いディジタル回路の特性を考慮し，変数に割り当てられた値の変化点に注目して，命題抽出とアサーション生成を行う方法を提案する．これにより，多クロックサイクルにわたって変化しない値は 1 つの値とみなすことができ，すべての値を考慮する既存手法では得られないアサーションをマイニングすることができるようになる．

表 2 は，マルチサイクル MIPS プロセッサの実行トレースの一部を示す．MIPS プロセッサでは，実行する命令の種類を表す信号Opと$Funct$の値によって，行う演算が決まる．マルチサイクルプロセッサでは，多サイクルで 1 つの命令を実行する．たとえば，デコードステージで$Op = 0$，$Funct = 32$を読み込んだ場合，指定されたレジスタの値$RD1$と$RD2$が加算された結果が，2 クロックサイクル後に指定されたレジスタに書き込まれる．このようなプロパティは$(Op[0] = 0) \wedge (Funct[0] = 32) \text{ -> } \#\#1\ (WD3[1] = RD1[0] + RD2[0])$と記述できそうに見える．しかし，次の命令が読まれるまでOpと$Funct$には同じ値が保持されるため，このアサーションは実行トレースで成立しない（表 2 中央）．

表 2. マルチサイクル MIPS プロセッサの実行トレース

Op	Funct	RD1	RD2	WD3
0	32	0	0	76
0	32	5	10	4
0	32	5	10	15
0	32	5	10	6
4	5	5	10	76
4	5	49	1	76
4	5	49	1	48

(Op[0]=0)∧(Funct[0]=32)
–> ##1(WD3[1]=RD1[0]+RD2[0])

Op	Funct	RD1	RD2	WD3
0	32	0	0	76
0	32	5	10	4
0	32	5	10	15
0	32	5	10	6
4	5	5	10	76
4	5	49	1	76
4	5	49	1	48

(Op[0]=0∧(Funct[0]=32)
–> ##1 (WD3[1]=RD1[0]+RD2[0])

Op	Funct	RD1	RD2	WD3
0	32	0	0	76
0	32	5	10	4
0	32	5	10	15
0	32	5	10	6
4	5	5	10	76
4	5	49	1	76
4	5	49	1	48

各変数について 値の
変化点に焦点を当てた場合

そこで，値の変化点のみに注目して命題抽出とアサーションマイニングを行う．表 2（右）は，1 つ前のクロックサイクルと比べて値が変化している点が太字で示されている．この部分の値のみを考えると，$(Op[0] = 0) \wedge (Funct[0] = 32) \text{ -> } \#\#1\ (WD3[1] = RD1[0] + RD2[0])$というプロパティを実行トレースから得ることができる．このようなプロパティは，$((\ Op[-1] != 0\ ) \vee (\ Funct[-1] != 32\ )) \wedge (\ Op[0] = 0\ ) \wedge (Funct[0] = 32) \text{ -> } \#\#1\ (WD3[1] = RD1[0] + RD2[0])$と解釈することで，実行トレースで成り立つアサーションとなる．一方で，アサーションマイニングフェーズにおいて，アサーションの結論部の値の変化点は考慮しない．表 2（左）において，最初の行を 0, 32,0, 0, 76 ではなく 0, 32, 5, 10, 4 であるとする．このとき値の変化点のみに注目すると，2 行目の 5, 10, 4 が無視され，アサーションをマイニングすることができなくなるためである．

4. 提案手法

提案手法は 4 つのフェーズからなる：

1) **原子命題抽出**：実行トレースを分析することにより，値の割り当てや変数間の関係を表すワードレベルの頻出原子命題を抽出する．
2) **命題マイニング**：アサーションの条件部候補を得るために，原子命題トレースから多クロックサイクルにまたがる頻出命題をマイニングする．
3) **アサーションマイニング**：フェーズ 1, 2 で得られた頻出命題のトレースから，アサーションをマイニングする．
4) **アサーションの枝刈りと結合**：マイニングされたアサーション集合から不要なアサーションを削除し，同じ条件部を持つアサーションの結論部を“∧”でつなぎ合わせる．

実行トレースにおいて頻繁に成立する命題は，DUV の動作をよく表現すると考えることができる．つまり，頻出する命題間の関係をマイニングすることで，DUV の動作においてカバレッジの高いアサーションを生成できることが期待できる．

値の変化点に注目するか，トレースのすべての値を考慮するかは，ユーザが選択できる．値が多クロックサイクルにわたって保持されるかどうかは，実行トレースやドキュメントから判断できる場合がある．上記のフェーズ 1 - 4 はこれらの 2 つの手法から独立して実行することができる．以下では，トレースのすべての値を考慮する場合について説明する．

4.1. 頻出原子命題の抽出

提案手法の第 1 フェーズでは，DUV の実行トレースを解析し，頻繁に成立するワードレベルの原子命題を抽出する．

アサーションマイニングをより効率的に行い，品質の良いアサーション集合を得るために，頻出原子命題を以下のように分類する：

(A) 条件部の候補：1 クロックサイクルで値の割り当てを行う原子命題の集合 (例：$a[0] = constant$)

(B) 結論部の候補：V_aの変数に値を割り当てる原子命題の集合 （例：$z[2] = a[0] + b[0]$, $z[1] = a[0] << 3$, $z[0] = constant\ (z \in V_a)$)

アルゴリズム 1. 頻出原子命題の抽出

```
1: Function GetAtomicProps(T, max_len, ant_th, con_th)
2:   ant_a_props = {}
3:   con_a_props = {}
4:   candidates = {}
5:   t_i = 0
6:   while t_i ≤ (length(T) − max_len) do
7:     ap_list = extractionAP(TW[t_i, t_i + max_len − 1])
8:     candidates.extend(ap_list)
9:     t_i = t_i + 1
10:  counted_ap = Counter(candidates)
11:  for all < ap, m >∈ counted_ap do
12:    if ((ap is candidate_of_antecedents)
                    and (m ≥ ant_th)) then
13:      ant_a_props = ant_a_props ∪ {ap}
14:    if ((ap is candidate_of_consequents)
                    and (m ≥ con_th)) then
15:      con_a_props = con_a_props ∪ {ap}
16:  return ant_a_props, con_a_props
```

頻出原子命題抽出の手順をアルゴリズム 1 に示す．関数$getAtomicProps$は，実行トレースT，アサーションをマイニングするタイムウィンドウの長さmax_len，2 つの閾値ant_th, con_thを引数として取る．ant_thは原子命題(A)の頻度の閾値，con_thは原子命題(B)の頻度の閾値である．関数$GetAtimicProps$は，まず，長さmax_lenの各タイムウィンドウ上で成り立つすべての原子命題を抽出する (6 - 9 行目)．各反復において，関数$extractionAP$によって各タイムウィンドウが分析され，タイムウィンドウ上で成り立つ原子命題のリストap_listが抽出される (7 行目)．同じ時刻で成り立つ同じ

原子命題を重複して抽出しないために，関数$extractionAP$は，時刻t_iの変数，すなわちタイムウィンドウの開始クロックサイクルで値を参照する変数を含む原子命題のみを抽出する．その後，ap_list内の原子命題が$candidates$に追加される (8 行目).

次に，関数$Counter$によってap_list内の各原子命題の頻度をカウントし，組<原子命題p, pの頻度>の集合を得る (11 行目)．その後，各原子命題は(A)または(B)に分類され，閾値を超える頻度を持つ命題のみが選択される (11 - 15 行目)．12 行目の「$ap\ is\ candidate\_of\_antecedents$」は，「原子命題$ap$は(A)に分類される」という意味であり，14 行目の「$ap\ is\ candidate\_of\_consequents$」は，「原子命題$ap$は(B)に分類される」という意味である．最終的に得られるant_a_propsは，アサーションの条件部候補の頻出原子命題の集合，con_a_propsはアサーションの結論部候補の頻出現し命題の集合である．

表 3. 頻出原子命題の例

実行トレース

time	p	q	r	x	y	z
t_0	0	32	False	5	1	0
t_1	0	34	True	10	5	0
t_2	43	5	False	20	8	6
t_3	0	34	False	3	30	5
t_4	0	32	True	6	7	35
t_5	4	2	False	4	4	-27
t_6	8	32	False	1	2	13

頻出原子命題

(A)
p[0] = 0
q[0] = 32
q[0] = 34
r[0] = True
z[0] = 0

(B)
z[2] = x[0] + y[0]
z[2] = x[0] - y[0]
z[0] = 0

例として，表 3（左）に示す実行トレースを考える．$\{z\} = V_a$かつant_th, con_thが 2 のとき，表 3（右）に示す原子命題が抽出される．ただし，この例ではブール変数の値“$False$”は考慮しない．

頻出原子命題の集合は，ユーザによって編集可能である．また，ユーザによって指定された他の形の原子命題を抽出することもできる．

条件部の頻出原子命題は，条件部の候補命題として用いられる他，フェーズ 2 において，より複雑な命題を生成するためにも用いられる．結論部の頻出原子命題は，結論部の候補命題としてそのまま使用する．

4.2. 頻出命題マイニング

このフェーズの目的は，アサーションの条件部の頻出原子命題から，多クロックサイクルにまたがる頻出命題をマイニングすることである．原子命題のすべての組み合わせを条件部の候補命題として考慮しないことで，過度に制約された条件部を持つアサーションをマイニングするのを防ぐことができる．つまり，実行トレース中で偶然成り立つアサーションを抽出するのを避けることができる．

既存手法[15]では，頻出原子命題間の頻出する関係は，アサーションマイニングにおいて考慮されない．[16]では，各クロックサイクルで保持される頻出原子命題間の頻出する関係が命題として考慮されるが，アサーションマイニングにおいて，多クロックサイクルにまたがる原子命題または命題間の頻出する関係は考慮されない．

頻出命題マイニングは以下のように行う：

i) 実行トレースTと前フェーズで得られた条件部候補の頻出原子命題の集合から，原子命題トレースωを得る．

ii) ωに対して長さmax_lenの各タイムウィンドウ$TW[ti, ti + max_len - 1]$を考慮し，各タイムウィンドウ内で成り立つ頻出原子命題を列挙したアイテムリストを生成する．

iii) 頻出パターンマイニングによって，アイテムリストから最小支持度min_sup以上の頻度を持つ頻出アイテムセットを抽出し，各ラージアイテムセット内の原子命題を“∧”で接続することにより，頻出命題を生成する．ここで，命題の重複を避けるために，時刻t_iで成り立つ原子命題を含む命題のみをマイニングする．

表 3 の例を考えると，$max_len = 3$のときに得られるアイテムリストは表 4（上）のようになる．

表 4. アイテムリストと頻出命題の例

アイテムリスト

TW[t_0, t_2]	p[0]=0, q[0]=32, z[0]=0, p[1]=0, q[1]=34, r[1]=True, z[1]=0
TW[t_1, t_3]	p[0]=0, q[0]=34, r[0]=True, z[0]=0, p[2]=0, q[2]=34
TW[t_2, t_4]	p[1]=0, q[1]=34, p[2]=0, q[2]=32, r[2]=True
TW[t_3, t_5]	p[0]=0, q[0]=34, p[1]=0, q[1]=32,r[1]=True
TW[t_4, t_6]	p[0]=0, q[0]=32,r[0]=True, q[2]=32

頻出命題	支持度	頻出命題	支持度
p[0]=0	0.8	(p[0]=0)∧(q[0]=34)	0.4
q[0]=32	0.4	(p[0]=0)∧(z[0]=0)	0.4
q[0]=34	0.4	(p[0]=0)∧(r[0]=True)	0.4
r[0]=True	0.4	(p[0]=0)∧(r[1]=True)	0.4
z[0]=0	0.4	(p[0]=0)∧(p[1]=0)	0.4
(p[0]=0)∧(q[0]=32)	0.4	(p[0]=0)∧(p[1]=0)∧(r[1]=True)	0.4

$min_sup = 0.4$で頻出パターンマイニングを実行し，アイテムリストから得られた命題は，表 4（下）のようになる．

提案手法におけるアサーションは，フェーズ 1, 2 で得られた条件部候補の頻出命題と，フェーズ 1 で得られた結論部候補の頻出命題の組からなる．

4.3. アサーションマイニング

このフェーズの目的は，実行トレースTに保持される，長さmax_len以下のアサーションを抽出することである．ここでのアサーションとは，条件部候補の命題から結論部候補の命題への含意の関係である．アサーションの長さは，アサーションが成立するのに必要なクロックサイクル数である．

アサーションマイニングの手順をアルゴリズム 2 に示す．関数$GetAssert$は，実行トレースT，アサーションをマイニングするタイムウィンドウの長さmax_len，2 つの集合$Ants$, $Cons$を引数として取る．$Ants$はフェーズ 1, 2 で得られたアサーションの条件部候補の頻出命題の集合，$Cons$はフェーズ 1 で得られたアサーションの結論部候補の頻出命題の集合である．関数$GetAssert$は，まず，関数$GetPropTrace$によって長さmax_lenの命題トレースAT, CTを得る (6 - 7 行目)．次に，各命題トレースで成り立つ命題$< p, offset >$の集合を得る(8 - 9 行目)．pは$Ants$または$Cons$の命題であり，$offset$は，t_iを 0 としたときに命題pが成立する時刻である．その後，各反復において，関数$GetAssert$は，$Assert$と$FailedAssert$という 2 つの集合を得る (10 - 22 行目)．このと

き，アサーションの重複を避けるために，条件部については$offset = 0$の命題のみを考慮する (10 行目)．$Assert$はt_i以前の実行トレースで成り立つアサーション$< a, c, offset >$の集合であり，$FailedAssert$はt_i以前の実行トレースで成り立たないアサーション$< a, c, offset >$の集合である．16 行目では，「$(p[0] = True) \wedge (q[1] = True) \text{ -> } \#\#0\ (z[0] = 1)$」のような時間的に矛盾したアサーションを生成しないために，命題aと命題cの長さが考慮される．$< a, c, offset >$は「$a \text{ -> } \#\#offset\ c$」を意味する．上記の処理は，タイムウィンドウごとに実行される．最後に，実行トレースTで成り立つアサーションの集合$Assert$を得る．

アルゴリズム 2. アサーションマイニング

```
 1: Function GetAssert(T, max_len, Ants, Cons)
 2:   Assert = {}
 3:   FailedAssert = {}
 4:   t_i = 0
 5:   while t_i ≤ (length(T) − max_len) do
 6:     AT = GetPropTrace(TW[t_i, t_i + max_len − 1], Ants)
 7:     CT = GetPropTrace(TW[t_i, t_i + max_len − 1], Cons)
 8:     a_set = GetProps(AT, Ants)
 9:     c_set = GetProps(CT, Cons)
10:     for all < a, 0 > ∈ a_set do
11:       for all c ∈ Cons do
12:         for each offset in max_len do
13:           if < a, c, offset > in FailedAsert do
14:             continue
15:           if < c, offset > in c_set do
16:             if lenth(a) ≤ offset + length(c) do
17:               Assert = Assert ∪ {< a, c, offset >}
18:               continue
19:           if < c, offset > not in c_set do
20:             if < a, c, offset > in Assert do
21:               Assert = Assert\{< a, c, offset >}
22:             FailedAssert
                    = FailedAssert ∪ {< a, c, offset >}
23:     t_i = t_i + 1
24:   return Assert
```

4.4. アサーションの枝刈りと結合

このフェーズの目的は，アサーションの枝刈りと結合によってアサーション集合の品質を向上させることである．以下のように枝刈りを行ったあと，同じ条件部を持つアサーションの結論部を“∧”で結合し，1 つのアサーションとする．まず，以下のようなアサーションαがあれば，αを枝刈りする．

(i) αは，条件部と結論部で同じ変数$v[i]$に値を割り当てる．

次に，以下のようなアサーション$\alpha 1$と$\alpha 2$がある場合，$\alpha 2$を枝刈りする．枝刈り(iii)は，1 つの動作に対して複数のアサーションが抽出された場合に，最も単純なアサーションを採用することで，アサーション集合の可読性を向上させるために行う．

(ii) $\alpha 1$と$\alpha 2$は同じ結論部を持ち，$\alpha 2$の条件部は$\alpha 1$の条件部を包含する．

(iii) $\alpha 1$と$\alpha 2$は同じ条件部を持ち，結論部で同じ変数$z[i]$ $(z \in V_a)$に値を割り当てる．このとき，$\alpha 1$の結論部は値の割り当て，$\alpha 2$の結論部は変数間の関係を表す．

5. 実験結果

実験は，24GB RAM を搭載した Intel Corei5 4590@3.30GHz 上で，4 つの設計を使用して行った．単一サイクル MIPS プロセッサ，マルチサイクル MIPS プロセッサ，パイプライン MIPS プロセッサの 3 つの設計は[17]，CORDIC の設計は OpenCores[18] のものを使用した．CORDIC (Coordinate Rotation Digital Computer)は，sinやcosなどの超越関数を計算するためのアルゴリズムである．

5.1. アサーションマイニング

表 5 は，実験に使用したV，V_aの変数の数 ($|V|, |V_a|$)，V中の変数の平均ビット数 ($size(var)$)，アサーションマイニングにおける最大クロックサイクル数 (max_len)，頻出条件部命題，結論部命題の数 ($\#ant, \#con$)，マイニングされたアサーションの数 ($\#assert$)，マイニングされたアサーションに含まれる条件部原子命題，結論部原子命題の平均数 ($size(ant)$, $size(con)$)，アサーションマイニングフェーズと枝刈り・結合フェーズの合計実行時間 ($time$) を示す．max_lenは，実行トレースから決定した．MIPS プロセッサでは 10000 行，CORDIC の設計では 3000 行の実行トレースを使用した．また，MIPS プロセッサでは値の変化点に着目する手法，CORDIC の設計ではトレースのすべての値を考慮する手法を選択した．さらに，CORDIC の設計に対する実験では，「$theta < 0$」,「$theta \geq 0$」を原子命題として手動で追加した．提案手法における各閾値は，明らかに頻度が低い命題をフィルタリングするように設定している．

表 5. アサーションマイニングの実験結果

DUV	|V|	|Va|	size(var)	max_len	#ant	#con	#assert	size(ant)	size(con)	time
単一サイクル MIPSプロセッサ	6	1	23.33	3	50	58	7	1.57	1.00	4.39s
マルチサイクル MIPSプロセッサ	6	1	23.33	5	55	34	14	1.29	1.21	3.08s
パイプライン MIPSプロセッサ	6	1	23.33	5	84	58	8	1.63	1.00	3.33s
CORDIC	6	2	14.33	2	20	29	9	1.10	2.60	1.24s

表 5 は，提案手法が，与えられたワードレベルの変数の集合に対して，数秒でアサーションを抽出できることを示している．パイプライン MIPS プロセッサと CORDIC の設計に対する実験で抽出されたアサーションの例を以下に示す．

(i) Pipelined MIPS processor
$(OpD[0] = 0) \wedge (FunctD[0] = 32)$
$\text{ -> } \#\#1\ (ResultW[2] = SrcAE[0] + SrcBE[0])$

(ii) CORDIC
$(init[0] = 0) \wedge (theta[0] \geq 0)$
$\text{ -> } \#\#0\ (y[1] = y[0] + x\_shifted[0]) \wedge (x[1] = x[0] - y\_shifted[0])$

アサーション(i)は，与えられた命令$(OpD[0] = 0) \wedge (FunctD[0] = 32)$にしたがってレジスタの値が加算される動作を捉えている．アサーション(ii)は，$\theta \geq 0$かつリセット信号が 0 のときの，$cos\theta$と$sin\theta$を得る計算を捉えている．

5.2. 頻出命題マイニングの効果

表 6 は，多クロックサイクルにまたがる頻出命題の抽出(フェーズ 2)を行わず，アサーションを抽出した結果を示す．この実験では，1 クロックサイクルで成り立つ頻出命題のすべてのテンポラルパターンが，アサーションの条件部候補として考慮される．表 5 と表 6 を比較すると，多クロックサイクルにまたがる頻出命題マイニングが，アサーションマイニ

ングの実行時間を改善することがわかる．さらに，アサーション数および条件部の平均サイズが，表 5 に示すものより大きくなっている．これは，実行トレースで偶然成り立つ，条件部が過度に制約されたアサーションが抽出されているためである．この実験により，多クロックサイクルにまたがる頻出命題のマイニングが有効であることが確認できた．

表 6.多クロックサイクルにまたがる頻出命題マイニングを行わない場合の実験結果

DUV	#assert	size(ant)	size(con)	time
単一サイクルMIPSプロセッサ	324	3.17	1.43	68.56s
マルチサイクルMIPSプロセッサ	137	2.96	2.38	19.15s
パイプラインMIPSプロセッサ	3925	3.73	2.91	251.97s
CORDIC	31	1.71	5.23	1.78s

5.3. ミュータント解析

表 5 の実験でマイニングされたアサーション集合の品質を測定するために，ミュータント解析を行った．ミュータントは DUV の人為的なエラーであり，ミュータントカバレッジは，生成されたミュータントとカバーされたミュータントの間の比率である．ミュータントを含む DUV 上で，あるアサーションが不成立であった場合，そのミュータントはアサーション集合によってカバーされる．本報告におけるミュータント解析では，MIPS プロセッサにおける制御ユニットの各出力信号と，CORDIC の設計における$V \backslash V_a$中の信号の各値をワードレベルで固定し，ミュータントとした．実験結果を表 7 に示す．$\#mutant$は 1 つのミュータントを含む DUV の数，$\#covered$はカバーされたミュータントの数，Avgは各アサーションによってカバーされたミュータントの平均数である．提案手法において，CORDIC の設計で達成されたミュータントカバレッジは 100%であった．プロセッサにおいては，Vの変数に影響を与えないミュータントが存在するため，カバレッジは 100%でない．表 7 は，提案手法でマイニングされたアサーション集合が，Vに影響するほとんどすべてのミュータントをカバーできることを示している．

表 7. ミュータント解析の結果

DUV	#mutant	#covered	Avg
単一サイクルMIPSプロセッサ	24	16	9.00
マルチサイクルMIPSプロセッサ	24	21	4.93
パイプラインMIPSプロセッサ	24	18	8.63
CORDIC	7	7	2.00

6. おわりに

本報告では，実行トレースを解析することにより，ワードレベルの変数間の多クロックサイクルにまたがる関係を原子命題として抽出し，それらの関係を含むアサーションをマイニングする手法を提案した．また，提案手法は，多クロックサイクルにまたがる原子命題の頻出する関係を命題として抽出し，アサーションの条件部候補とすることにより，アサーションを効率的に生成することができる．実験結果は，提案手法が多クロックサイクルにまたがるワードレベルの関係を含むアサーションを効率的に抽出できることを示している．また，ミュータント解析の結果は，提案手法によって抽出されたアサーションが DUV の動作を捉えることができることを示す．

今後は，抽出可能なアサーションの形を増やし，より大きな設計で実験を行うことが必要である．

参考文献

[1] L. -C. Wang, M. S. Abadir, and N. Krishnamurthy, "Automatic generation of assertions for formal verification of powerpc microprocessor arrays using symbolic trajectory evaluation". Design Automation Conference (1998) 534-537.

[2] G. Ammons, R. Bodík, and J. R. Larus, “Mining specifications,” ACM Sigplan Notices, vol. 37, no. 1 (2002) 4-16.

[3] S. Hangal, S. Narayanan, N. Chandra and S. Chakravorty: "IODINE: a tool to automatically infer dynamic invariants for hardware designs," Design Automation Conference (2005) 775-778.

[4] G. Fey and R. Drechsler, "Improving simulation-based verification by means of formal methods”.Asia South-Pacific Design Conference (2004) 640-643.

[5] B. Isaksen and V. Bertacco, "Verification through the principle of least astonishment". ICCAD (2006) 860-867.

[6] P. H. Chang and L. C. Wang, "Automatic assertion extraction via sequential data mining of simulation traces," Asia and South Pacific Design Automation Conference (2010) 607-612.

[7] W. Li, A. Forin and S. A. Seshia, "Scalable specification mining for verification and diagnosis," Design Automation Conference (2010) 755-760.

[8] S. Vasudevan, D. Sheridan, D. Tcheng, S. Patel, W. Tuohy, and D. Johnson, "GoldMine: Automatic assertion generation using data mining and static analysis”, DATE (2010) 626-629.

[9] S. Hertz, D. Sheridan, S. Vasudevan, "Mining Hardware Assertions With Guidance From Static Analysis”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,Volume 32 Issue 6 (2013) 952-965.

[10] L. Liu and S. Vasudevan. Automatic Generation of System Level Assertions from Transaction Level Models. J. Electronic Testing 29, 5 (2013), 669-684.

[11] L. Liu, C. H. Lin and S. Vasudevan, "Word level feature discovery to enhance quality of assertion mining," IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2012) 210-217.

[12] M. Bonato, G. D. Guglielmo, M. Fujita, F. Fummi, G. Pravadelli, "Dynamic Property Mining for Embedded Software," IEEE/ACM/IFIP international conference onf Hardare/softwere codesign and system synthesis (2012) 87-196.

[13] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, C. Xiao, "The daikon system for dynamic detection of likely invariants”, Sci. Comput. Program., 69(1-3) (2007) 35-45.

[14] A. Danese, T. Ghasempouri, G. Pravadelli, “Automatic extraction of assertions from execution traces of behavioral models”, Design, Automation & Test in Europe Conference & Exhibition (2015) 67-72.

[15] A. Danese, F. Filini and G. Pravadelli, "A time-window based approach for dynamic assertions mining on control signals," IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC) (2015) 246-251.

[16] A. Danese, N. D. Riva, G. Pravadelli, "A-TEAM: Automatic template-based assertion miner," Design Automation Conference (2017), Article No. 37.

[17] D. M. Harris, S. L. Harris, “Digital Design and Computer Architecture,” 2nd ed., Waltham, MA: Morgam Kaufmann (2012).

[18] OpenCores benchmarks. www.opencores.org.