設 計 自 動 化 60−26
（1991．12．13）

論理関数処理を用いた分岐時間正則時相論理による順序機械の設計検証

濱口清治† 平石裕実‡ 矢島脩三†
† 京都大学工学部 ‡ 京都産業大学工学部

順序機械の設計の形式的検証の一手法として、近年、Burch らによって提案された記号モデル検査法(symbolic model checking method)は、二分決定グラフを利用した論理関数処理に基づく手法であり、従来手法では検証不可能だった大規模な順序機械を取り扱いうるが、Burch らが仕様記述に用いた計算木論理（CTL）は、繰り返しが書けないなど、表現能力の点で問題がある。これに対して、我々が提案した分岐時間正則時相論理（BRTL）は、CTL より真に強力な表現能力を持っている。本稿では、BRTL に対する論理関数処理に基づく検証アルゴリズムを示すと共に、作成した検証システムの非同期回路およびマイクロプロセッサーに対する適用結果を示す。

Design Verification of Sequential Machines Based on Branching Time Regular Temporal logic Using Logic Function Manipulation

Kiyoharu HAMAGUCHI†, Hiromi HIRAISHI‡, Shuzo YAJIMA†
† Faculty of Engineering, Kyoto University
‡ Faculty of Engineering, Kyoto Sangyo University

Recently, as a formal verification method of sequential machines, Burch et al. developed a new method called symbolic model checking, which is based on logic function manipulation using Binary Decision Diagram, and showed that verification cost can be reduced drastically.

Burch et al. adopted Computation Tree Logic to describe specifications. Its expressive power, however, is known to be rather weak. We have proposed Branching Time Regular Temporal Logic, whose expressiveness is more powerful than that of CTL. In this paper, we show the algorithm for BRTL based on logic function manipulation.

We apply the proposed method to verification of asynchronous circuits and a microprocessor.

1 はじめに

大規模集積回路技術の発達により、多種多様の論理設計が行われるようになってきている。このため、回路が設計者の意図した通り正しく設計されているかどうかを厳密にかつ自動的に検証することは、設計誤りの早期発見／正当性の保証という点から、重要な課題となっている。形式的検証手法は、設計した回路が与えられた仕様記述を満足するかどうかを、完全に保証しようとするものである。

本稿では、順序機械の形式検証のための分岐時間正則時相論理 (Branchiing Time Regular Temporal Logic, 以下 BRTL) を仕様記述言語とする論理関数処理に基づく検証アルゴリズムを示し、これに基づき作成した検証システムによる非同期回路とマイクロプロセッサの検証例を示す。

現在、設計自動化に関連して研究・開発が行われている検証問題には、(1) 論理関数の等価性判定 [1, 2]、(2) 順序回路 (順序機械) の等価性または包含性判定 [3, 4, 5] のほか、(3) 命題時相論理や有限オートマトンを仕様記述言語とした順序機械の検証 [6, 7, 8, 9]、(4) 高階論理などを仕様記述言語とした順序機械よりも複雑な対象 (任意の大きさのメモリをもつマイクロプロセッサーなど) の検証 [10] 等がある。

上記の (1) に関しては、近年、二分決定グラフ [11, 1] が導入され、大規模な論理回路の取り扱いが可能になってきている。(4) では、「(実現を表す論理式) ⇒ (仕様を表す論理式)」を定理の証明系により証明するという手法が用いられ、もっとも複雑な対象を記述できるが、証明の自動化が困難で人手に頼る部分が非常に多く、自動化が可能な場合でも停止性が保証されないなどの問題点がある。(2) と (3) に関しても、定理証明を用いた試みがあるが [12]、対象が有限状態と捉えられる範囲では、上記の問題点を抱えた定理証明系は、表現能力が不必要に強力すぎると考えられる。このため一般には、(2) に関しては、状態枚挙 (state enumeration) の手法が用いられている。これは、設計対象がとり得る全ての状態をたどり、仕様が満足されるどうかを調べる手法である。(3) において時相論理を仕様記述言語として用いる場合にも (2) と同様の手法が用いられるが、これはモデル検査法 (model checking method) と呼ばれている [6]。本稿で述べる分岐時間正則時相論理 BRTL [13] による検証手法もこのモデル検査法によるものである。

命題時相論理 [14] は、通常の命題論理に時間の概念を表すための演算子 (時相演算子と呼ばれる) を加え、拡張したものである。命題時相論理には種々の体系が提唱されており、古典的な時相論理 (Propositional Temporal Logic)[14] や、その拡張である拡張時相論理 (Extended Temporal Logic)[15] などが知られているが、これらを利用した順序機械の検証問題を考えるとその計算複雑度は多項式領域完全またはそれ以上となる [16]。これに対して、計算木論理 (Computation Tree Logic, 以下、CTL)[6] では、有限状態機械の遷移図の一種である Kripke 構造の大きさと、時相論理式によって書かれた記述の大きさの積に対して線形時間のアルゴリズムが知られている。しかし、CTL は繰り返しが書けないなど、その表現能力が限られている [17]。我々が提案した BRTL はこの点の改善を図ったもので、CTL より真に強力な表現能力を持ち、かつその計算量は CTL と同じという特徴を持つ [13]。

一方、これまでモデル検査手法を用いる場合には、状態遷移図や Kripke 構造をグラフとして構成する必要があったため、取り扱うことのできる設計の規模は、せいぜい 10^6 状態程度のとどまっていた。しかし、Burch らは、二分決定グラフを利用し状態遷移図を論理関数で表現することによって、10^{20} 状態を越える極めて大きな順序機械の検証も可能になる場合があることを示した [7]。

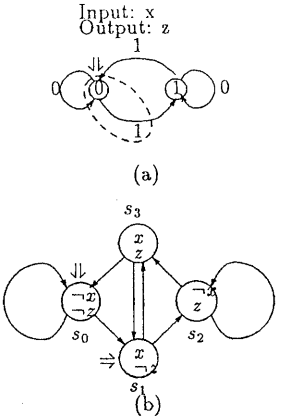


図 1: 順序機械から Kripke 構造への変換

本稿では、我々の提案した BRTL に対して、論理関数処理に基づく検証アルゴリズムを示す。また、これに基づいて作成した検証システムを用い、大規模な順序機械の例として非同期回路およびマイクロプロセッサをとり上げ検証を試みたので、その結果を報告する。

以下、2 章では、BRTL に基づく順序機械の形式的検証手法の概観を述べ、3 章では、論理関数処理に基づくアルゴリズムを例によって示す。3 章のアルゴリズムに基づき作成した検証システムを用いて、4 章と 5 章では、それぞれ非同期回路とマイクロプロセッサの検証例を示す。なお、BRTL の形式的な定義および検証アルゴリズムの詳細は、付録に示している。

2 形式的検証手法の概観

本節では，設計としての順序機械と，仕様としての BRTL の論理式が与えられたとき，形式的検証がどのように行なわれるかを概観する。

入力と出力を対にした入出力対を考えると、順序機械の動作はその入出力対の系列の集合であると捉えることができる。検証アルゴリズム (以下ではモデル検査アルゴリズムをこう呼ぶ) は入出力対の系列の集合が 仕様として与えられた BRTL 式の規定する性質を満足するかどうかを調べるものである。

以下では、順序機械の入出力として，0,1 の 2 値のものを扱うことにする．ブール演算子 $\vee$,$\neg$, $\wedge$, $\equiv$ および $\Rightarrow$ をそれぞれ論理積、否定、論理積、同値、包含の意味で用いるものとし、V_T は恒真式を表すとする。

(1) 順序機械から Kripke 構造への変換

入出力対に着目すると 図 1 (a) の Moore 型の順序機械は 同図 (b) に変換することができる。これが、Kripke 構造である。変換して得られる Kripke 構造は順序機械の動作を反映するものになり，例えば，図 1 (a) の破線で囲まれた部分の動作は (b) の節点 s_1 に変換される。順序機械の初期状態に対応する Kripke 構造上の初期節点は太い矢印で示されている。

(2) 仕様記述

例として，次の仕様を BRTL で記述する。

「入力 x が 1 になれば，次の時刻で出力 z の値は反転するという性質が常に成立する」

$\forall$ Always($x \Rightarrow ((z \Rightarrow \forall$ Next$(\neg z)) \wedge (\neg z \Rightarrow \forall$ Next$(z))))$

ここで，$\forall$ は「(Kripke 構造上の) すべての経路上で」を意味している。Always(f), Next(f) はそれぞれ、「常に f が成立する」，「次の時刻で f が成立する」を意味する。BRTL では、Always と Next は図 2 の有限オートマトン (オートマトン演算子と呼ばれる) によって記述される。ここで、◎ は受理状態、○ は非受理状態を表す。

このオートマトンは枝の遷移条件として命題論理式 (一般には他の BRTL 式) を持つ。使われる原子命題が x, y, z であり、遷移枝に $x \wedge \neg y$ がラベルされている場合は、(x, y, z) が $(1, 0, 0)$ と $(1, 0, 1)$ の場合の両方の遷移を表していると解釈することができる。

BRTL で用いる有限オートマトンは、Kripke 構造上の無限経路、すなわち、0,1 からなる入出力対の無限系列に対して、受理／非受理を決定する。受理条件は、受理状態を無限回通過することと定義する。ただし、次のような制約を満たすものとする (厳密な定義は付録を参照)。

- 唯一の初期状態をもつ。
- 遷移は決定的でかつ完全指定である。
- ある非受理状態 r から始まる経路で、受理状態を経由して r へ戻る経路が存在しない。

有限オートマトンを用いた BRTL 式としては、$\forall A$, $\exists A$, $\forall(A_1 \vee A_2)$, $\forall\neg A$ などがある。

(3) 検証

BRTL の論理式の真偽は，Kripke 構造上の各節点に対して定まる．有限オートマトンを用いた論理式の真偽は，$\forall$ ($\exists$) が先行する場合は，各節点からはじまるすべての (ある) 無限経路が，そのオートマトンによって受理されるかどうかによって定義される．

例えば，図 1(b) においては，$\forall$ Next(z) は節点 s_2 で真になる．これは s_2 から始まるすべての経路上の次の節点 s_2, s_3 で z が真になり，図 2 の Next により受理されるからである．同様に，$g \stackrel{\text{def}}{=} x \Rightarrow ((z \Rightarrow \forall$ Next$(\neg z)) \wedge (\neg z \Rightarrow \forall$ Next$(z)))$ は，節点 s_0, s_1, s_2, s_3 で真になる。

次に g に着目すると、Kripke 構造の任意の初期節点から始まるすべての経路が (g を原始命題とみなしたとき) 図 2 Always によって受理される。すなわち、すべての経路に対して「常に g」が成立し、仕様が検証されたことになる。

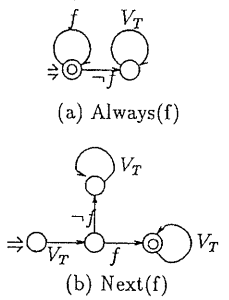
3 論理関数処理を用いた検証アルゴリズム

以下のアルゴリズムは、CTL の場合と同じく、状態枚挙に基づくモデル検査アルゴリズムの一種であるが、本稿では単に検証アルゴリズムと呼ぶことにする。

3.1 準備

次の記法を用いる。ここで、f は論理関数である。$\exists x.f(x, y) \stackrel{\text{def}}{=} f(0, y) \vee f(1, y)$。$\vec{x} = x_1, x_2, \cdots, x_n$ に対して、$\exists \vec{x}.f(\vec{x}, \vec{y}) \stackrel{\text{def}}{=} \exists x_1.(\exists x_2.\cdots(\exists x_n.f(\vec{x}, \vec{y}))\cdots)$

また、$B = \{0, 1\}$ としたとき、次の n 変数論理関数 f_X は $X \subseteq B^n$ を表現するという。



(a) Always(f)

(b) Next(f)

図 2: オートマトン演算子

$$x \in X \Leftrightarrow f_X(x) = 1$$

順序機械 M は各状態に対し符号割当をおこない、遷移関数と出力関数の形で表現したものを入力とする (簡単のため、完全指定を仮定する)。B 上の入力変数と状態変数をそれぞれ $x_i (1 \leq i \leq l)$ および $y_j (1 \leq j \leq m)$ とし、変数のベクトルを $\vec{x} = x_1, x_2, \cdots, x_l$ などと記す。

- 遷移関数: $f_j : B^l \times B^m \rightarrow B$ $(1 \leq j \leq m)$
- 出力関数: $z_k : B^m \rightarrow B$ $(1 \leq k \leq n)$ (Moore 型)
  $z_k : B^l \times B^m \rightarrow B$ $(1 \leq k \leq n)$ (Mealy 型)

また、初期状態の集合を表現する関数を $g_{init}(\vec{y})$ とする。

3.2 検証アルゴリズム

まず、与えられた遷移関数から、Kripke 構造の枝の集合を表現する関数を構成する。

(1) Kripke 構造への変換

この変換では、Kripke 構造の状態の集合は、入力変数の定義域と状態変数の定義域の直積空間 $B^l \times B^m$ となる。$\vec{s} \stackrel{\text{def}}{=} \vec{x}\#\vec{y}$ ($\#$ は $\vec{x}$ と $\vec{y}$ の連接を表す) とする。また $\vec{x}$ に対して、新しい変数 $\vec{x'} = x'_1, x'_2, \cdots, x'_k$ を導入する。同様に $\vec{y'}$ も導入して、$\vec{s'} = \vec{x'}\#\vec{y'}$ とする。このとき、Kripke 構造 S の枝の集合を表現する関数 f_S は次のようになる。

$$f_S(\vec{s}, \vec{s'}) = \wedge_{0 \leq j \leq m}(y'_j \equiv f_j(\vec{x}, \vec{y}))$$

Kripke 構造の初期節点の集合を表現する関数 f_{init} は、$f_{init}(\vec{s}) = g_{init}(\vec{y})$ 、また x_i および z_k が真になる状態の集合表現する関数はそれぞれ、$f_{x_i}(\vec{s}) = x_i$, $f_{z_k}(\vec{s}) = z_k(\vec{y})$ (または $z_k(\vec{x}, \vec{y})$) となる。

(2) 検証アルゴリズム

例によって示す (アルゴリズムの詳細は付録を参照)。

以下では、論理関数を入力とする FB と LFP という操作を用いる。$FB(f_T)$ はある節点集合 T を表現する論理関数 f_T から T の親となっている節点の集合を表現する論理関数 f_U を求める操作であり、次のように定義される。ここで f_S は 前述の Kripke 構造の枝を表現する関数である。

$$FB(f_T) \stackrel{\text{def}}{=} \exists \vec{s'}.(f_S(\vec{s}, \vec{s'}) \wedge f_T(\vec{s'}))$$

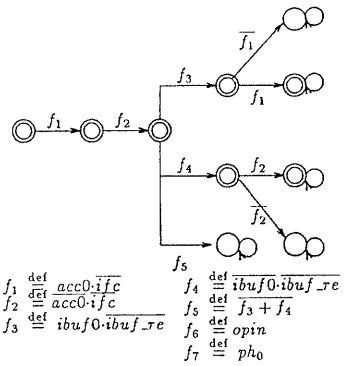


図 3: オートマトン演算子 B

$LFP[f_c](f_T)$ (f_c は論理関数) はある節点集合 f_T によって表現される集合 T を子孫に持つ節点の内、f_c が成立する節点のみを通過して T へ到達する経路がすくなくともひとつ存在するような節点の集合を求める操作である。$f^0 = f_T$ として、次の操作を $f^n = f^{n+1}$ となるまで、$i = 0, 1, \cdots, n$ について繰り返し適用することによって得られた $f^n(\vec{s})$ が解である。

$$f^{i+1} = f^i \vee (FB(f^i) \wedge f_c)$$

これは集合の最小不動点 (least fix point) を求める操作になっている。

例として 図 3 の有限オートマトン B を用いて、

$$g = \forall Always(\forall(Always(op) \Rightarrow B))$$

を検証する場合を考える。順序機械に対する遷移関数は上記の形式で与えられているとする。

(a) $h = \forall\,(Always(op) \Rightarrow B)$ が成立する節点の集合を表現する論理関数 $f_h(\vec{s})$ を構成する。まず、$Always(op) \Rightarrow B$ を一つの有限オートマトンに変更する。これは、$Always(op)$ の受理状態と非受理状態を交換し、B と直積機械をとることによって得られる。こうして得られた有限オートマトンを C とする。次に、$\forall C$ が成立しない状態の集合を表現する論理関数 $f_{\neg h}(\vec{s})$ を構成する。これは、非受理状態に到達する経路を取り出して (図 4(a))、図 4(b) の計算を行うことによって得られる。なお、図では省略しているがどの状態からも $\overline{op}$ を遷移条件とする遷移があって、受理状態へ遷移している。このとき、$f_h = \overline{f_{\neg h}}$ となる。

(b) $f_h(\vec{s})$ を用いて $\forall Always(h)$ が成立しない節点集合を表現する関数 $f_{\neg g}(\vec{s})$ を求める。これは、図 5(b) によって求めることができる。ここで、$LFP[f_c]$ は前述の最小不動点を求める操作を表す。

(c) 初期状態を表現する論理関数 $f_{init}(\vec{s})$ と $f_{\neg g}(\vec{s})$ の論理積を求め、恒偽になれば、初期状態から $\neg g$ の状態に至る経路が存在しないことになり、仕様が設計によって満足されたと判定できる。

なお、$g = \forall A_1 \vee \forall A_2$ の場合は、$\forall A_1$ 、$\forall A_2$ のそれぞれが成立する節点集合を表現する論理関数を求め、論理和を取れば、f_g を求めることができる。他の論理演算ついても同様である。また、グラフの構造がこの例よりも複雑な場合の一般的なアルゴリズムは付録に示す。

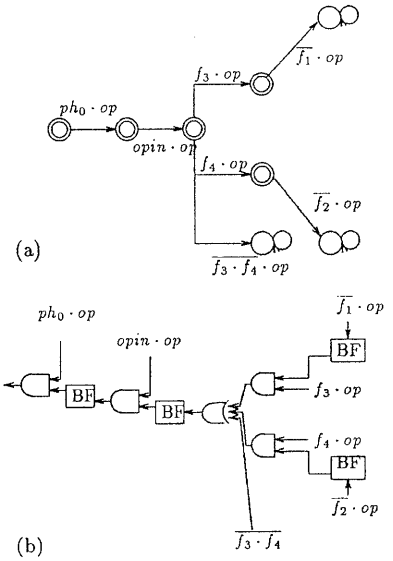


図 4: $f_{\neg h}(\vec{s})$ の計算

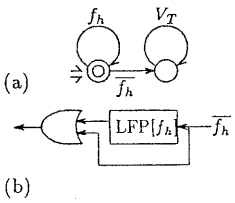


図 5: $f_{\neg g}(\vec{s})$ の計算

3.3 アルゴリズムの実現

以上の検証アルゴリズムを実現する場合、論理関数処理を効率的に行うことが必要である。共有二分決定グラフ (Shared Binary Decision Diagram, 以下 SBDD) [2] は二分決定グラフを改良したもので固定された論理変数の順序によってシャノンの展開形を二分木で表現し、同型な部分グラフをすべて共有させたものである。SBDD は次のような特徴を持つ。

1. 論理関数に対する論理演算、代入演算、等価性判定を高速に実行できる。

2. 大規模な論理関数を効率よく表現できる。

1. の性質は検証の高速化のために、また 2. の性質は状態数のより大きな順序機械を取り扱うために有効である。本稿のアルゴリズムの実現においても SBDD を用いる。

なお、Kripke 構造の枝の集合を表現する f_S は状態遷移関数や出力関数に比べて、SBDD 表現したとき、非常に大きくなる場合がある。以下の実験結果では文献 [9]

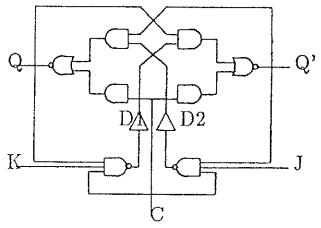


図 6: JK フリップフロップ

の方法によって、f_S の直接生成を避ける方法を用いている。

4 非同期回路の検証

大規模な順序回路の例として、以下のモデル化に基づいて非同期回路の検証を行った。

4.1 回路のモデル

ここでは、離散時間モデルを用いる。各論理素子の遅延にばらつきがあると仮定し、(非決定性の) Moore 型順序機械とみなして、その状態遷移関係を、以下のように論理式を用いて表す。以下、遅延は有限区間で限定されているとする。遅延の値が 3 から 5 の範囲で変化する遅延素子は、入力変数 i, 現状態変数 s_i、次状態変数 s_i' $(1 \leq i \leq 5)$ を用いて、以下のように表現する。

$$((s5' = s4' = s3' = i) \wedge (s1' = s2) \wedge (s2' = s3)) \vee$$
$$((s5' = s4' = i) \wedge (s3' = s4) \wedge (s1' = s2) \wedge (s2' = s3)) \vee$$
$$((s5' = i) \wedge (s4' = s5) \wedge (s3' = s4) \wedge (s1' = s2) \wedge (s2' = s3))$$

前節までの手法は決定性順序機械を対象としているが、この手法は非決定性順序機械 (およびそのネットワーク) に対しても適応できる。具体的には、上記のように素子 C_i を 論理関数 f_{C_i} $(i = 1, 2, \cdots, n)$ で表現した場合、Kripke 構造を表現する関数 f_S は、$f_S(\vec{s}, \vec{s'}) = \wedge_{1 \leq i \leq n} f_{C_i}$ となる。

4.2 BRTL による仕様の記述

図 6 の JK フリップフロップに対する条件として、「クロック C が 4 クロック分の長さのパルスを出した後、立ち下がり、かつ常に $\neg J \wedge K$ が成立するならば、いつか Q は 0 になりそのまま 0 に留まり続ける」かつ「クロック C が 4 クロック分の長さのパルスを出した後、立ち下がり、かつ常に $J \wedge K$ が成立するならば、いつか Q の値は反転する」という仕様は、図 2 の $Always(x)$、図 7 の $Pulse(x)$ および $Fall(x)$ と図 8 の $Change(x)$ を用いて、次のように記述することができる。

$$Spec =$$
$$\forall((Pulse(C) \wedge Always(\neg J \wedge K)) \Rightarrow Fall(Q)) \wedge$$
$$\forall((Pulse(C) \wedge Always(J \wedge K)) \Rightarrow Change(Q)) \quad (1)$$

4.3 実験結果

前節までの手法に基づき、SPARC station 2 上に C 言語を用いてプログラムを実現し、実験を行った。

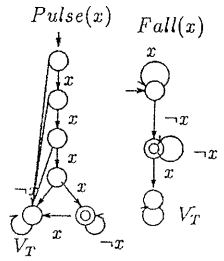


図 7: オートマトン演算子 (1)

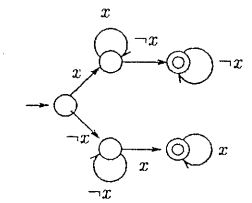


図 8: オートマトン演算子 (2)

図 6の回路に対して遅延の値を変えて、前述の BRTL 式 (1) を検証した結果を表 1に示す。いずれの場合も、$C = 0$ の安定状態を初期状態とし、パルスの長さは適宜調節した。SBDD のための記憶領域はあらかじめ 11 M バイト (SBDD の節点数,50 万ノード) に制限して実験を行った。

検証に要する時間は、遅延値の増大とともに急激に増大することがわかるが、それでもなお、状態遷移図または Kripke 構造をグラフとして陽に生成する手法では取り扱うことのできない規模の回路を扱うことができ、これは SBDD を用いたアルゴリズムが大きな順序回路の検証に有効であることを示唆している。

表 1: 実行結果 (非同期回路)

	# OP	時間	# 変数	遅延 1	遅延 2	OK
djk1	34	1.6	19	[1,5]	[1,1]	×
djk1$^+$	38	1.7	24	[9,10]	[1,1]	○
djk2	42	71.2	43	[11,12]	[1,2]	○
djk3	58	1616.0	59	[15,16]	[2,3]	○

 # OP: BRTL 式中のオートマトン状態数 (仕様の大きさを表す)
時間: CPU 時間 (秒)
 # 変数: 入力数 + 使用された状態変数の数
遅延 1: 遅延素子 $D1$ と $D2$ の遅延
遅延 2: 遅延素子 $D1$ と $D2$ 以外の遅延
OK: 仕様を満足するか否か

Phase 0	Phase 1	Phase 2	Phase 3
(pc) → mar pc++	(mem) → ir	(ibuf) → acc 0 → ibuf_re	0 → ifc

図 9: 命令 IN の動作

5 マイクロプロセッサーの検証

5.1 仕様の記述

Burch らは、パイプライン構造を持つ簡単な計算機の検証を行った [7] が、この順序回路データは、プログラムによってベンチマーク用に生成されたものであった。本稿では、実際に LSI チップとして作成するため設計された回路を取り上げ、本稿の検証手法の適用を試みる。ここで検証実験の対象とした KUE-CHIP 2 は KUE-CHIP (Kyoto University Educational Chip)[18] を改良したものであり、計算機工学教育用に設計された 8 ビットマイクロプロセッサである。同プロセッサは同期式設計された順序回路であり、約 3,500 ゲートの論理素子と、62 個のフリップフロップを含む (内部メモリ (512 語) を除く)。

図 10 のようなバスにたいして、バス競合／フロートのがないという仕様は、次のように書くことができる。これはバスへのデータ出力を制御しているトライステートバッファの制御線が一本だけ 1 になり (競合しない)、かつ少なくとも一本は 1 になる (フロート状態にならない) ということ意味している。

$$\forall Always(a\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}\bar{b}c)$$

命令の検証については、ある命令語が命令レジスタにロードされたときの動作をフェーズごと、ビットごとに仕様として記述する。例えば、図 9 は 命令 IN (外部からのデータの読み込み) の動作を表しているが、3 節の BRTL 式 $g = \forall\ Always(\ \forall\ (Always(op) \Rightarrow B))$ はそのフェーズ 2 以降の部分を記述するためのものである。図 3では ibuf0, acc0 が用いられているが、これはそれぞれ入力データの 0 ビット目、アキュムレータ (ACC) の 0 ビット目を表しており、検証を行う場合は、全てのビットに関して同様の記述を作成し、仕様として用いる。op は CPU が動作中であることを示す信号であり、ph_0 はフェーズ 0 であること表す信号、$opin$ は IN 命令であることを表す原子命題である。

5.2 検証実験

前節と同様、SPARC station 2 上で実験を行った。

順序回路のデータは ES2 (European Silicon Structure) の CAD ツールである SOLO[19] 上で作成された KUE-CHIP 2 の設計データを直接、変換して作成した。この際、フリップフロップ部分は適宜、次状態変数を用いて変換し、また図 10 のようなバスの部分は、次の論理式に変換した。

$$bus = (a \Rightarrow d_a) \wedge (b \Rightarrow d_b) \wedge (c \Rightarrow d_c)$$

また、内部メモリは切り離して回路データを作成した。

SBDD のための記憶領域はあらかじめ 11 Mバイト (SBDD の節点数,50 万ノード) に制限して、バス競合／フロートの有無、およびいくつかの命令の動作の検証を行った。

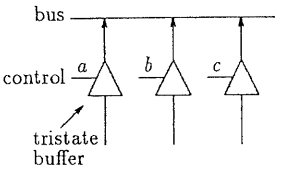


図 10: バス

表 2: 実行結果 (KUE-CHIP 2)

	時間 (秒)	# PH	
bus	13.5	–	バス競合 / フロート
ph0	22.2	–	フェーズ 0
ph1	20.7	–	フェーズ 1
HLT	15.2	3	停止命令
OUT	44.5	4	ACC から出力
IN	24.6	4	ACC へ入力
RCF	15.1	3	キャリーフラグリセット
ST†	742.0	5	ACC から メモリへストア
LD†	748.1	5	メモリから ACC へロード
ADC†	1552.4	5	加算命令 (キャリーつき)

時間: CPU 時間 (秒)
 # PH: 各命令のフェーズ数
†: いずれも修飾アドレスを指定した場合

回路中の全てのトライステートバッファの制御線に対して、上記と同様の仕様を記述し、検証を行ったところ、内部のレジスタ内容の読みだしに利用する オブザーバ・バスがフロート状態になる場合があることが発見され、この部分はデバッグされた。

フェーズ 0 とフェーズ 1 の動作は全ての命令について同じなので、命令語の種類を指定しないで、検証を行った。表 2で ph0 と ph1 はそれぞれフェーズ 0 とフェーズ 1 での動作の検証に対応している。他の命令の検証には、フェーズ 0 とフェーズ 1 の部分は含まれていない。bus は前述のバスに対する仕様を与えたこと意味している。(表中の命令の検証では、フェーズが遷移したとき、内容が変化しないことになっているレジスタ (例えば図 9 でのインデクスレジスタ) については、値が変化していないということは検証していない)

なお、1 命令のみの実行、1 フェーズのみの実行を指定する入力等は OFF に固定するなどして、検証の簡単化を図っている。

ADC はインデクスレジスタによるアドレス修飾を含む命令であり、命令セットの中では、もっとも複雑な動作を指定する命令と考えられることから KUE-CHIP2 の他の命令についても、現実的な時間で検証可能であると考えられる。

6 おわりに

本稿では、分岐時間正則時相論理の論理関数処理のアルゴリズムを示した。また、これに基づいて作成したシステムによって、非同期回路とマイクロプロセッサの動作について検証を行い、従来の状態遷移表または Kripke

構造を用いる手法に比べて、より規模の大きい順序回路の検証に有効であると示した。

非同期回路については、本稿のモデルでは遅延が大きくなるにつれ急激に検証が難しくなる。これは、最小不動点 (LFP) を計算する際、関数列が収束するまで到達した全ての順序回路の状態を記憶しているためであるが、入力タイミングの制約を考慮したり、イベントに注目することにより、必要記憶量の低減化がはかることができないか現在検討中である。

マイクロプロセッサーの検証については、より大規模な回路に適用するためには、ビット幅を 1 ビットに抽象化するといった計算量の低減を図る手法の確立が今後の課題である。また、BRTL ではビットごとの仕様を書かなければならず記述が繁雑になることから、実用的に用いるためには、より記述が容易な言語を構築していく必要がある。

謝辞

KUE-CHIP 2 の設計データの提供に関して御協力頂いた九州大学大学院総合理工学研究科の安浦寛人教授、京都高度技術研究所の神原弘之氏に深謝致します。また、KUE-CHIP 2 について種々御教示頂いた京都大学工学部の越智裕之氏、澤田宏氏、岡田和久氏、立命館大学理工学部の上嶋明氏に感謝致します。

参考文献

[1] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[2] S. Minato, N. Ishiura, and S. Yajima. Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation. In *Proceedings of 27th Design Automation Conference*, pages 52–57, 1990.

[3] O. Coudert, C. Berthet, and J-C. Madre. Verification of Sequential Machines Using Functional Vectors. In *Proceedings of IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, November 1989.

[4] O. Coudert and J-C. Madre. A Unified Framework for the Formal Verification of Sequential Circuits. In *Proc. of ICCAD*, pages 126–129, 1990.

[5] H. J. Touati, H. Savoj, B. Lin, and R. K. Brayton. Implicit State Enumerateion of Finite State Machines Using BDDs. In *Proc. of ICCAD*, pages 130–133, 1990.

[6] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach. In *10th ACM Symposium on Principles of Programming Languages*, pages 117–126, January 1983.

[7] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill. Sequential Circuit Verification Using Symbolic Model Checking. In *Proceedings of 27th Design Automation Conference*, pages 46–51, 1990.

[8] O. Coudert, J.-C. Madre, and C. Berthet. Verifying Temporal Properties of Sequential Machines Without Building their State Diagrams. In *Proceedings of Workshop on Computer-Aided Verification*, 1990.

[9] H. Hiraishi, K. Hamaguchi, H. Ochi, and S. Yajima. Vectorized Symbolic Model Checking of Computation Tree Logic. In *Workshop on Computer-Aided Verification*, pages 279–290, July 1991.

[10] M. Gordon. HOL:A Proof Generating System for Higher-Order Logic. In G. Birtwistle and P. Subrahmanyam, editors, *VLSI Specificaiton, Verificaiton and Synthesis*, pages 73–128. Kluwer Academic Publishers, 1988.

[11] S. B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, June 1978.

[12] W. Mao and G. J. Milne. An Automated Proof Technique for Finite-State Machine Equivalence. In *Proceedings of Workshop on Computer-Aided Verification*, pages 305–316, 1991.

[13] K. Hamaguchi, H. Hiraishi, and S. Yajima. Branching Time Regular Temporal Logic for Model Checking with Linear Time Complexity. In *Workshop on Computer-Aided Verification*, June 1990.

[14] N. Rescher and A. Urquhart. *Temporal Logic*. Springer-Verlag, 1971.

[15] P. Wolper. Temporal Logic Can Be More Expressive. In *Proceedings of 22nd Annual Symposium on Foundations of Computer Science*, pages 340–348, 1981.

[16] A. P. Sistla and E. M. Clarke. The Complexity of Propositional Temporal Logic. *Journal of the ACM*, 32(3):733–749, July 1985.

[17] E. M. Clarke and O. Grümberg and R. P. Kurshan. A Synthesis of Two Approached for Verifying Finite State Concurrent Systems. Technical report, Carnegie Mellon University, 1987. manuscript.

[18] 神原弘之. 教育用マイクロプロセッサ：KUE-CHIP. 情報処理学会技術研究報告、*90-ARC-82*, 1990.

[19] ES2. *SOLO 1400, Release 3.1*.

付録

BRTL の構文と意味

BRTL では時相演算子として次の dfa-1 を用いる。

定義 1 決定性 ω 有限オートマトン 1 型 *(deterministic ω finite automaton type 1,* 以下、*dfa-1)* $A = (Q, P, Br, q_0, F)$ を次のように定義する。

Q は有限個の状態の集合、$P = \{p_1, \cdots, p_n\}$ は命題変数の集合。$Br : Q \times Q \to BF$ (BF は P から構成される命題論理式の集合) は状態の 2 つ組に命題論理式を割り当てる部分関数である。Br は次の条件を満たす。$q \in Q$ に対して、$Br(q, Q) = \{f \mid \exists\, q'.\ Br(q, q') = f\}$ を考えた時、「任意の $f_1 \neq f_2$ なる $f_1, f_2 \in Br(q, Q)$ に対して、$f_1 \wedge f_2$ は恒偽」かつ「$\bigvee_{f \in Br(q,Q)} f$ は恒真」。q_0 は初期状態であり、F は受理状態の集合である。$Q - F$ の要素を非受理状態と呼ぶ。また次の条件を満足する。

「どの非受理状態 q_r からも、受理状態を経由して再び q_r へ遷移させる入力系列が存在しない」

A は $\Delta = 2^P$ の要素の無限系列に対して、受理／非受理を決定する。遷移関数 $\delta : Q \times \Delta \to Q$ は $q, q' \in Q$, $v \in \Delta$ に対して、$\delta(q, v) = q' \Leftrightarrow Br(q, q')(v) = T$ と定める。Δ の要素からなる無限系列 σ を A に入力した時、無限回通過する状態の集合を $Inf(\sigma)$ と表す。A が受理する言語は $\{\sigma \mid Inf(\sigma) \cap F$ が空でない$\}$ と定義する。 □

命題 1 *[13]* dfa-1 $A = (Q, P, Br, \delta, q_0, F)$ に対し、補集合を受理する dfa-1 $\overline{A}$ は、F を $Q - F$ にすることにより得られる。また、dfa-1 A_1 と A_2 の受理する集合の和集合を受理する dfa-1 $(A_1 | A_2)$ も dfa-1 として構成できる。すなわち、dfa-1 はブール演算に関して閉じている。 □

定義 2 構文

BRTL 式：以下では、BRTL 式の集合を BRF と記す。

$p \in AP$、$\psi, \phi \in BRF$ また、A がオートマトン演算子ならば、$p, (\neg\psi), (\psi \vee \phi), (\exists A) \in BRF$ である。

オートマトン演算子

AP を命題変数の集合とする dfa-1 $A_i\ i=0,\cdots,n$ を考える。$\{\psi_1,\psi_2,\cdots,\psi_n\}\subseteq BRF$ とする。このとき、$A(\psi_1,\psi_2,\cdots,\psi_n)$ は、A 中の各命題変数 $p_1,p_2,\cdots p_n\in AP$ を同時に それぞれ $\psi_1,\psi_2,\cdots,\psi_n$ で置き換えたものとする。$Br(q,q')$ も代入後の値を返すものとする。このとき、$A_i(\psi_1,\psi_2,\cdots,\psi_n)$ に対して、$\neg$ (単項演算子) と $\vee$ (二項演算子) を有限回適用して得られるものをオートマトン演算子と定義する。 □

命題 1 より、BRTL のオートマトン演算子に対しても、$\exists\neg A$ と $\exists(A_1\vee A_2)$ をそれぞれ $\exists\overline{A}$ と $\exists(A_1|A_2)$ と表現できることが示せる。

BRTL 式の意味は、Kripke 構造に対して定まる。$AP=\{p_1,p_2,\cdots,p_n\}$ を原始命題の集合とする。

定義 3 $S=\langle\Sigma,I,R,\Sigma_0\rangle$ を Kripke 構造と呼ぶ。ここで、Σ は節点の集合、$I:\Sigma\to 2^{AP}$ は各節点に対し、原始命題の真偽を割り当てる関数、$R\subseteq\Sigma\times\Sigma$ は Σ 上の有向枝の集合 (任意の節点から少なくとも 1 本の枝が出ているとする) であり、Σ_0 は初期節点の集合である。 □

定義 4 意味

$S,s\models\psi$ は BRTL 式 ψ が節点 $s\in\Sigma$ に対して真であることを意味する。$p\in AP$、$\psi,\phi,\psi_1,\psi_2,\cdots,\psi_n$ は BRTL 式、A は dfa-1 とする。

- $S,s\models p\Leftrightarrow p\in I(s)$
- $S,s\models(\psi\vee\phi)\Leftrightarrow S,s\models\psi$ または $S,s\models\phi$
- $S,s\models(\neg\psi)\Leftrightarrow S,s\not\models\psi$
- $S,s\models(\exists A(\psi_1,\psi_2,\cdots,\psi_n))\Leftrightarrow$ Kripke 構造 S 上の節点 s から始まる無限系列 $s_0s_1s_2\cdots$ と A の状態の無限系列 $q_0q_1q_2\cdots$ が存在し、次の条件を満足する。$S,s_i\models Br(q_i,q_{i+1})\ (i=0,1,2,\cdots)$ かつ $q_0,q_1,\cdots$ の内、無限回出現する状態が少なくとも一つ A の受理集合に含まれる。
- $S,s\models(\exists\neg A(\psi_1,\psi_2,\cdots,\psi_n))\Leftrightarrow$ $S,s\models(\exists\overline{A}(\psi_1,\psi_2,\cdots,\psi_n))$

全ての $s\in\Sigma_0$ に対し $S,s\models\psi$ の時、$S\models\psi$ と書く。 □

$\forall A\stackrel{\text{def}}{=}\neg\exists\neg A$ および $\forall\neg A\stackrel{\text{def}}{=}\neg\exists A$ とする。$\forall,\exists$ はそれぞれ Kripke 構造上のすべての経路およびある経路を意味する。

検証アルゴリズム

以下の検証アルゴリズムは次の定理を利用している。

定理 1 $S,s\models\exists A(\psi_1,\psi_2,\cdots,\psi_n)\Leftrightarrow$
Kripke 構造 S の節点と オートマトン演算子 A の状態の組の有限系列 $(s_0,q_0)\ (s_1,q_1)\cdots(s_k,q_k)\ (s_{k+1},q_{k+1})\cdots(s_n,q_n)$ (ここで、$s_0=s$) が存在して、条件 1 および条件 2 を満足する。
条件 1 $(s_i,s_{i+1})\in R\ (i=0,1,\cdots n-1)$ かつ $S,s_i\models Br(q_i,q_{i+1})\ (i=0,1,\cdots n-1)$
条件 2 $s_k=s_n$ かつ $q_k=q_n$ かつ $q_j\in F\ (j=k,k+1,\cdots n)$ □

論理変数として、上記の $\vec{s},\vec{s'}$ の他にオートマトン演算子の状態を表現するための状態変数 $\vec{q}=q_1,\cdots,q_w$ および $\vec{q'}=q'_1,\cdots,q'_w$ を用いる。オートマトン演算子の各状態に 0,1 により符号割当を行い、状態 q に対する符号を $cd(q)$ により表す。また、$\vec{v}=\vec{s}\#\vec{q}$, $\vec{v'}=\vec{s'}\#\vec{q'}$ とする。

アルゴリズム 1 BRTL の検証アルゴリズム

- 入力: BRTL 式 ψ および 上記で構成した Kripke 構造 S に対する f_S, f_{init}, f_{x_i} および f_{z_k}。
- 出力: $S\models\psi$ ならば yes。他の場合は no。
- 方法:

1. ψ 中の各オートマトン演算子 A_j から定理 1により、'$\neg$' を取り除き、A_j に対し、次の論理関数を構成する。
   - A_j の枝 (q,q') に対し $f_{E_j(\psi_i)}(cd(q),cd(q'))=1\Leftrightarrow Br(q,q')=\psi_i$
   - q がオートマトン演算子 A_j の受理状態 $\Leftrightarrow f_{F_j}(cd(q))=1$
2. 以下の (a)-(c) により、ψ の部分式 ϕ_i に関して原始命題からボトムアップに、$S,s\models\phi_i$ なる節点集合を表現する $f_{\phi_i}(\vec{s})$ を求める。f_{h_1},f_{h_2},f_h は、それぞれ、$S,s\models h_1$, $S,s\models h_2$,$S,s\models h$ なる節点 s の集合を表現するとする。
   (a) ϕ_i が原始命題のときはすでに与えられている。
   (b) ϕ_i が $h_1\vee h_2$ または $\neg h$ の場合、$f_{h_1\vee h_2}=f_{h_1}\vee f_{h_2}$、$f_{\neg h}=\neg f_h$
   (c) $\phi_i=\exists A_j$ の場合。
   (a) 定理 1 に基づき、節点 $\Sigma\times Q$ からなる直積グラフ G の枝の集合を表現する関数 $f_E(\vec{s},\vec{q},\vec{s'},\vec{q'})$ を構成する。A_j の枝にラベル付けされている BRTL 式を $\psi_1,\psi_2,\cdots,\psi_k$ とし、$S,s\models\psi_i\ (i=1,2,\cdots,k)$ となる状態の集合を表現する $f_{\psi_i}(\vec{s})$ を求めておくことにより、
   $f_E(\vec{v},\vec{v'})\stackrel{\text{def}}{=}\vee_{i=1,2,\cdots,k}(f_{E_j(\psi_i)}(\vec{q},\vec{q'})\wedge f_{\psi_i}(\vec{s})\wedge f_S(\vec{s},\vec{s'}))$
   (b) 次に G の節点集合を $V'=\{(s,q_F)|q_F\in F_j\}$ に制限した部分グラフ G' の強連結成分および強連結成分へ到達可能な節点の集合 $V_{C'}$ を表現する $f_{C'}$ を求める。(図 1 を参照)
   まず、V' の枝の集合を表現する $f_{E'}$ を求める。
   $$f_{E'}(\vec{v},\vec{v'})\stackrel{\text{def}}{=}f_{F_j}(\vec{q})\wedge f_{F_j}(\vec{q'})\wedge f_E(\vec{v},\vec{v'})$$
   次に、$f_{C'}$ を構成する。$f^0_{C'}(\vec{v})\stackrel{\text{def}}{=}f_{F_j}(\vec{q})$ とする。これは、G' の節点 (s,q) の集合を表現している。$f^{k+1}_{C'}\equiv f^k_{C'}$ になるまで、次の $f^1_{C'},\cdots f^k_{C'}$ を求める。
   $$f^{i+1}_{C'}(\vec{v})\stackrel{\text{def}}{=}(\exists\vec{v'}.(f_{E'}(\vec{v},\vec{v'})\wedge f^i_{C'}(\vec{v'})))\wedge f^i_{C'}(\vec{v})$$
   $$f_{C'}(\vec{v})\stackrel{\text{def}}{=}f^k_{C'}(\vec{v})$$
   (c) G 上で $V_{C'}$ のいずれかの節点に到達可能な節点の集合 V_R を表現する f_R を求める。$f^0_R(\vec{v})\stackrel{\text{def}}{=}f_{C'}(\vec{v})$ とする。$f^{k+1}_R\equiv f^k_R$ になるまで、次の $f^1_R,f^2_R,\cdots f^k_R$ を求める。
   $$f^{i+1}_R(\vec{v})\stackrel{\text{def}}{=}(\exists\vec{v'}.(f_E(\vec{v},\vec{v'})\wedge f^i_R(\vec{v'})))\vee f^i_R(\vec{v})$$
   $$f_R(\vec{v})\stackrel{\text{def}}{=}f^k_R(\vec{v})$$
   (d) $f_{\phi_i}(\vec{s})\stackrel{\text{def}}{=}\exists\vec{q}.f_R(\vec{s},\vec{q})$
3. $(f_{init}(\vec{s})\Rightarrow f_\eta(\vec{s}))=V_T$ ならば yes、他の場合は no を返す。