設計自動化 44－2
（1988. 10. 28）

時間記号論理シミュレーションの結果解析系

高橋 瑞樹、石浦 菜岐佐、矢島 脩三
京都大学工学部

論理回路のタイミング検証のための新しいアプローチとして、我々は時間記号論理シミュレーションを提案している。時間記号論理シミュレーションは、ゲートの遅延や入力の変化時刻を変数で表わしシミュレーション時刻を代数式のまま扱うことにより、遅延のばらつき等をモデル化した精密なシミュレーションを行うことができる。時間記号論理シミュレーションの結果は、遅延に関する条件とイベントの系列からなるイベント木で与えられるが、設計者が期待するイベント系列が得られる条件を求めるという作業は、イベント木が大きい場合や条件が複雑である場合は人手では困難となる。 本論文では、上記の問題を解決するものとして、シミュレーション結果を解析し回路が設計者の期待通りに動作するための条件を求める結果解析系について述べる。

Result-Analysis System for Time-Symbolic Logic Simulation

Mizuki TAKAHASHI, Nagisa ISHIURA and Shuzo YAJIMA
Department of Information Science, Faculty of Engineering, Kyoto University
Kyoto 606, Japan

As a new approach for timing verification of logic circuits, we have proposed a new concept of time-symbolic logic simulation. Time-symbolic logic simulator treats a time as a linear combination of variables representing gate delays or the times of input events, which allows a precise timing analysis of the logic circuits. Since a direct output of the time-symbolic logic simulator is event-trees for circuit outputs, which represents event sequences and the conditions to allow the event sequences, it is often difficult by hands to obtain conditions to allow the expected event sequences when the event-trees are large or the conditions are complex.

In this paper, as a solution of this problem, we describe an result-analysis system for time-symbolic logic simulation which analyzes the simulation result and compute the condition in which the circuit under test will behave expectedly.

1．はじめに

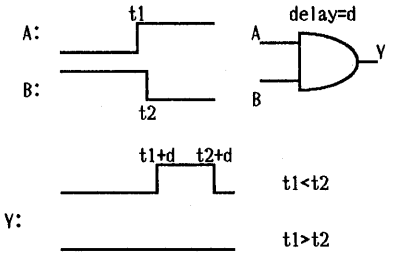
近年の集積回路技術の進歩に伴い、より大規模で高機能な論理システムが実現されるようになっているが、その設計検証をいかに行うかという問題がCAD/DA技術に課せられた大きな課題となっている。論理回路の設計支援ツールとしては、従来から論理シミュレーションが広く用いられており、いまや論理設計支援システムには欠くことのできないものとなっている。設計検証の中でもタイミングに関する検証は複雑で困難な場合が多い。特に非同期回路として設計されるものに対しては、仕様通りの動作をするかどうか、ハザード、発振、競合によるエラーが発生しないかどうかを調べるために精密なタイミング解析が必要になる。このような場合には、微妙なタイミングが問題になり、製造条件や使用環境の違いに起因する素子遅延のばらつきまでをも考慮する必要がある。論理シミュレーションでは、このような遅延のばらつきを最大／最小遅延シミュレーション[1]により解析するが、現実よりも悲観的な結果しか得られないことが知られている。我々は、これらの問題を解決するものとして時間記号論理シミュレーションを提案している[2]。従来の記号シミュレーションが信号線の信号値を記号で表わしシミュレーションを行うのに対し、時間記号論理シミュレーションでは、素子の遅延時間や入力の変化する時刻を、変数を含む式で表現して遅延のばらつきをモデル化し、精密なタイミング解析を行おうとするものである。

以前に作成した時間記号論理シミュレータでは、シミュレーション結果から動作条件を求める作業は人手で行なっていたが、イベント木が大きい場合や条件が複雑な場合には人手では困難であるという問題点があった。この問題を解決するものとして、本論文では、シミュレーション結果と期待値から動作条件を求めるアルゴリズムを示し、実現した結果解析系とその非同期順序回路の検証への適用例について述べる。

2.時間記号論理シミュレーション

時間記号論理シミュレーションでは、ゲート遅延や入力信号値の変化する時刻のばらつきを正確にモデル化するためにゲートの遅延時間、入力の変化時刻を変数で表わしシミュレーションを行う（図2.1参照）。この変数を、時間変数と呼ぶ。時間変数は、取り得る値の範囲が制限されることがあるが（例えば遅延の上下限が与えられる場合）、これを一次不等式の集合（図2.1における、0<d<10,0<t1,0<t2等）で表わし、変数制限条件と呼ぶことにする。

本稿では、信号線の信号値系列をイベント木というデータ構造を用いて表わす。図2.2は図2.1の信号値系列をイベント木で表現したものである。イベント木は、イベント節点（図中長方形で示された節点）、条件節点（楕円で示された節点）と呼ばれる節点からなる有向木である。イベント節点はイベント（時刻と信号値



変数制限条件：0<d<10,0<t1,0<t2

図2.1 時間記号論理シミュレーションの概念

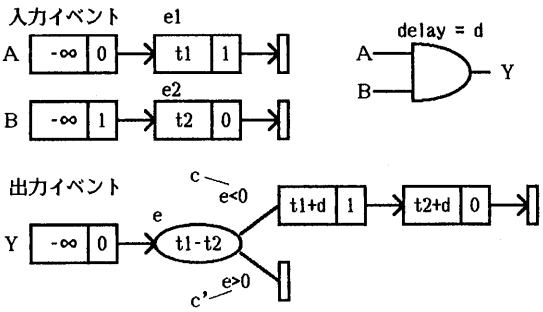


図2.2 イベント木の構造

の組）でラベル付けされた節点であり、イベント節点または条件節点への有向枝が出ている。条件節点は、時間変数の一次式でラベル付けされた節点であり、式ｅでラベル付けされた条件節点からは、ｅ＜０の条件が成り立つ場合の節点への有向枝と、ｅ＞０の条件が成り立つ場合の節点への有向枝が出ている。ｅ＜０，ｅ＞０の条件をそれぞれの有向枝に対する分岐条件という。それぞれの節点から出ている有向枝はイベントの発生順序を表わす。有向木の根には、発生時刻が－∞で初期値を表わす初期イベント、葉には信号値の終了を表わす終了イベントがおかれる。

初期イベントから終了イベントに至るパス上にあるイベント節点の系列がこの信号線に発生し得るイベントの系列を表わす。また、パス上にある分岐条件の集合がそのようなイベント系列が発生する条件であり、これをそのパスのパス条件と呼ぶ。

時刻を変数を含む式として扱う論理シミュレーションは一般には複雑で大変効率の悪いものになるが、回路を組合せ回路に限定すれば、Ｔ－アルゴリズム[3]の考え方により、比較的単純で効率の良いシミュレーションが可能になる[2]。即ち、回路内の各ゲートについてレベル順に、入力のイベント木から出力のイベント木を計算するという操作を繰り返すことにより回路の出力のイベント木を求めるのである。ゲートの出力のイベント木を求める際、イベントの発生時刻が定数ではなく変数であることに起因して、入力イベント

の前後関係によって場合分けが行われ、出力のイベント木上に分岐が生ずる。しかし、変数制限条件とそこまでのパス条件（その時点までの分岐条件の集合）より、注目している２つのイベントの前後関係が一意に決まり分岐が生じない場合もある。例えば図１において、変数制限条件として、0<t1<5,6<t2<10が与えられれば、t1>t2となることは起こり得ないので、出力のイベント木には分岐は生じないことになる。我々は、変数制限条件、パス条件、分岐条件を時間変数に関する連立一次不等式とみなし、分岐の可能性の判定を、線形計画法を用いた連立一次不等式の解の存在判定により行っている。[2]

以上の概念に基づいた時間記号論理シミュレータは、UNIX上でC言語により既に実現している[4]。また、線形計画法のルーチンに実行時間の大半がかかるため、線形計画法の適用法に改良（タブローの再利用等）を加え高速化を図っている[5]。

3. 結果解析系

3.1 結果解析系の概要

時間記号論理シミュレーションでは、出力結果としてイベント木が得られる。イベント木は、信号線に発生し得るイベント系列とその系列が発生する条件を表現している。回路の動作の確認や、正常動作のための条件を求めるためには、シミュレーションによって得られたイベント木と、設計者の期待するイベント系列（以後、期待値と呼ぶ）との照合を行い、両者が一致する条件を求めるという作業が必要となる。この照合は、イベント木が小さいときには人手でも行うことができるが、イベント木が大きい場合、あるいは複数の信号線が期待値と一致する条件を求める場合には、自動的に照合を行う結果解析系が必要となる。

以下、本稿では、この照合処理の手法を次の３段階にわけて考える。

（1）入力パターンp_1に対する出力信号線o_1のイベント系列が期待値に一致する条件を求める。

（2）入力パターンp_1に対する出力信号線o_1～o_nのイベント系列が全てそれぞれの期待値に一致する条件を求める。

（3） 複数の入力パターンp_1～p_mに対する出力信号線o_1～o_nのイベント系列が全てそれぞれの期待値に一致する条件を求める。

3.2 アルゴリズム

3.2.1 イベント木と期待値の照合

イベント木と期待値の照合によって得られる条件は、期待値に一致するイベント系列に対応するパス条件である。ここでは、これらの条件を条件木というデータ構造を用いて表わす。条件木は、不等式条件節点、等式条件節点、TRUE節点、FALSE節点からなる有向木である。不等式条件節点は時間変数の一次式cでラベル付けされた節点であり、c<0の場合とc>0の場合に対応

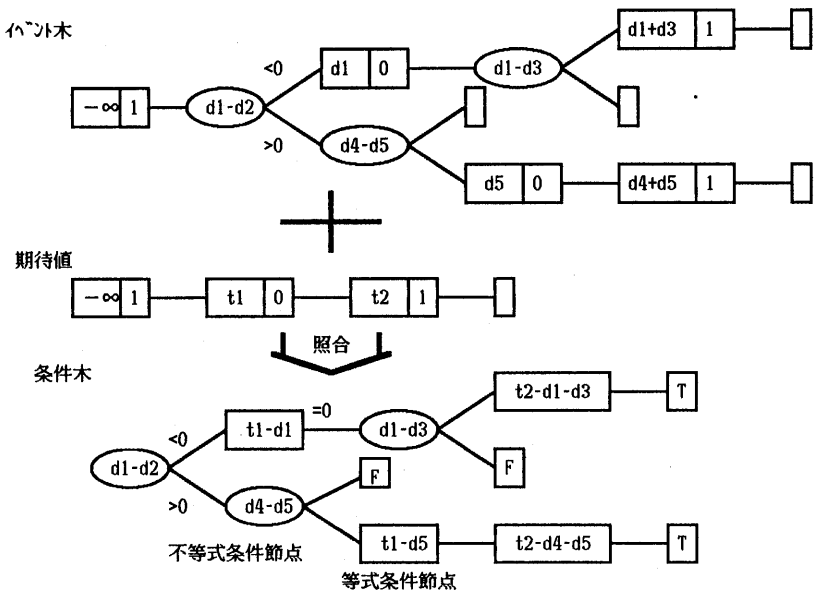


図3.1　イベント木と期待値の照合

```
[照合アルゴリズム]
照合(ev,ex,PC)
{
  if (ev,ex共に終了イベント)
    return(TRUE節点);
  if (evが条件節点)
  {
    /* evは式cでラベル付けされているとする */
    if (PC∪{c<0}とPC∪{c>0}の両方が解を持つ)
    {
      不等式条件節点pを作る;
      pの左の子=照合(evの左の子,ex,PC∪{c<0});
      pの右の子=照合(evの右の子,ex,PC∪{c>0});
      return(p);
    }
    if (PC∪{c<0}だけが解を持つ)
      return(照合(evの左の子,ex,PC));
    if (PC∪{c>0}だけが解を持つ)
      return(照合(evの右の子,ex,PC));
  }
  if (evがイベント節点)
  {
    if (exが終了イベント)
      return(FALSE節点);
    else
    {
      if ((evのイベント値)==(exのイベント値))
      {
        t=(evのイベント時刻)-(exのイベント時刻);
        if (PC∪{t=0}が解を持つ)
        {
          tでﾗﾍﾞﾙ付けされた等式条件節点pを作る;
          pの子=照合(evの子,exの子,PC∪{t=0});
          return(p);
        }
        else
        return(FALSE節点);
      }
      else
        return(FALSE節点);
    }
  }
}
```

図3.2　照合アルゴリズム

する部分木への2本の有向枝が出ている。等式条件節点は時間変数の一次式eでラベル付けされた節点であり、e=0の場合の部分木への有向枝が出ている。上の２つの条件節点において、c<0,c>0,e=0をイベント木のときと同様に分岐条件と呼び、根から葉に至るパスの分岐条件の集合を、そのパスのパス条件と呼ぶ。条件木の葉は、TRUE節点またはFALSE節点であり、根からTRUE節点に至るパスのパス条件が、期待値に一致する条件の一つを表わす。例えば、図3.1において、イベント木が期待値に一致する条件は、

(d1<d2 ∧ t1=d1 ∧ d1<d3 ∧ t2=d1+d3) ∨
(d1>d2 ∧ d4>d5 ∧ t1=d5 ∧ t2=d4+d5)

となる。

図3.2に照合アルゴリズムを示す。入力はシミュレーション結果のイベント木への根ev、期待値（イベント系列であり、これもイベント木）への根ex,変数制限条件PCであり、得られた条件木の根を出力として返す。PCは、不等式条件または等式条件の集合であるが、タプローの形で渡される。PCの初期値は空集合である。

3.2.2 条件木の統合

3.2.1の照合アルゴリズムによって求めた条件は、個々の信号線について独立に求めた条件であり、ある信号線のシミュレーション結果が期待値と一致する条件が、他の信号線の条件と矛盾する場合がある。ここでは、そのような条件を排除し、期待値を設定した信号線に対応する複数の条件木を統合し、一つの条件木を得るアルゴリズムについて述べる。期待値を設定した信号線がn本で、それらのシミュレーション結果を照合し条件木1～条件木nが得られたとき、条件木を統合しn本の信号線が期待値通りになるための条件木を求めるアルゴリズムは次のようになる。

[条件木統合アルゴリズム]

step1 step 2を条件木i(2≦i≦n)について繰り返す。
step2 step3～4を条件木１のすべてのTRUE節点について繰り返す。
step3 TRUE節点を条件木iのコピーで置き換える。
step4 TRUE節点までのパス条件PCのもとで、置き換えた条件木の矛盾をチェックする。

step4の矛盾をチェックする手続きは、PCを空集合、

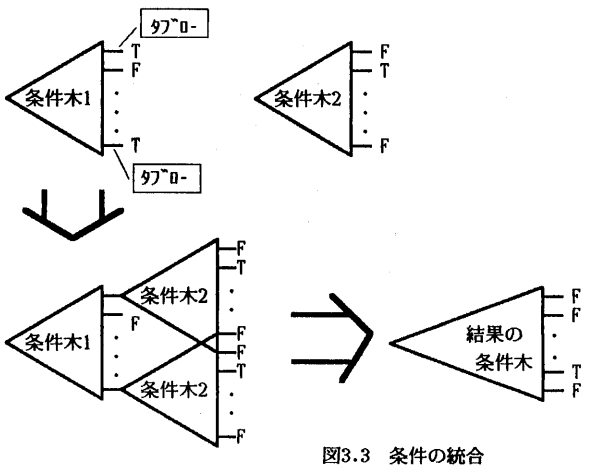


図3.3　条件の統合

[矛盾チェック手続き]

```
矛盾チェック(p,PC)
{
  switch(p)
  {
    case TRUE節点: return;
    case FALSE節点: return;
    case 等式条件節点:
      /* pは式eでラベル付けされているとする */
      if (PC∪{e=0}が解を持つ)
        矛盾チェック(pの子,PC∪{e=0});
      else
        pをFALSE節点で置き換える;
      return;
    case 不等式条件節点:
      /* pは式cでラベル付けされているとする */
      if (PC∪{c<0}とPC∪{c>0}の両方が解を持つ)
      {
        矛盾チェック(pの左の子,PC∪{c<0});
        矛盾チェック(pの右の子,PC∪{c>0});
        return;
      }
      if (PC∪{c<0}だけが解を持つ)
      {
        矛盾チェック(pの左の子,PC∪{c<0});
        pの右の子をFALSE節点で置き換える;
        return;
      }
    if (PC∪{c>0}だけが解を持つ)
      {
        矛盾チェック(pの右の子,PC∪{c<0});
        pの左の子をFALSE節点で置き換える;
        return;
      }
  }
};
```

図3.4　条件木矛盾チェック手続き

置き換えた条件木の根をpとするとき、図3.4に示す通りである。

3.2.3 条件木の下でのシミュレーション

時間記号論理シミュレーションは、１つの入力パターンに対してイベント木を求めるため、複数の入力パターンに対する回路の正常動作の条件を求めるには、そのための手段が必要となる。これには、次に示す２つの方法が考えられる。

(1) 入力パターンの数だけ独立にシミュレーションを行い、それぞれ照合、統合を行って入力パターンに対する条件木を得る。得られた条件木を、さらに統合し最終的に一つの条件木を得る。

(2) (1)と同様にシミュレーションは入力パターンの数だけ行うが、前回のシミュレーション、照合及び統合で得られた条件木も入力し、その条件木の下でシミュレーションを行う。これにより、最後に行われたシミュレーションによる条件木が最終的な条件となる。

(1)と(2)を比較した場合、(2)のシミュレーションは条件木も入力されるためイベント木を小さくでき、適用回路規模も大きくできると考えられる。そこで、本論文では(2)の方法を用いる。

条件木の下でのシミュレーションは次に示す手順により行える。

step1 step2～step4を条件木の全てのTRUE節点について繰り返す。

step2 TRUE節点のパス条件PCを変数制限条件としてシミュレーションを行う。

step3 step2の結果と期待値とを照合し、統合を行って条件木を求める。

step4 TRUE節点をstep3で求めた条件木で置き換える。

3.3 結果解析系の実現

3.3.1 システムの概要

3.2で示したアルゴリズムをSONY NWS-830上にＣ言語を用いて実現し、以前に作成した時間記号論理シミュレータに組み込み新たなシミュレータを作成した。このシミュレータの入力は回路Ｃ、外部入力パターンＩ、条件木ＣＴ、期待値Ｅであり、出力は回路が期待値通りに動作するための条件木ＣＴ'である。

3.3.2 条件の出力

シミュレータの出力は条件木であるが、木の形のままで出力しただけでは利用者には理解しにくい。そこで本シミュレータでは、条件木の出力のほかに、条件木をパス条件に展開して表示している。例えば図3.1の条件木は

```
t1=d1 t2=d1+d3
d1<d2 d1<d3
－－－－－－－－－－－
t1=d5 t2=d4+d5
d1>d2 d4>d5
```

のように表示される。また、この際、冗長な条件の削除も行う。例えば、パス条件が

```
d2+d3+d6<d4 d4+d5<d1+d3 d2<d1
```

と与えられたとする（全ての時間変数は正）。このとき、d2+d3+d6<d4とd4+d5<d1+d3よりd2+d6<d1が導け、d2<d1は冗長な条件であり必要ない。パス条件の集合をPC、不等式条件cの否定(不等号を逆にした条件)をc'で表わすと、PCからcを取り除いてc'を加えたものが解を持たない場合には条件cは冗長であることがわかる。これを、PC中の全ての不等式条件に対して行うことにより、冗長な条件を全て削除している。現在、条件式は、上の例のように等式及び不等式の積和の形で表示しているが、論理式簡単化プログラムや記号シミュレータ[7][8]を用いれば更に簡潔な表示が得られる

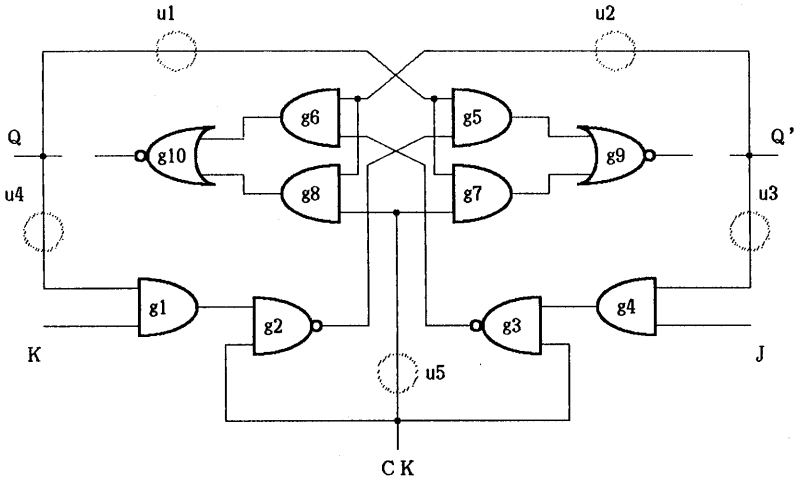


図4.2　ＪＫフリップフロップ

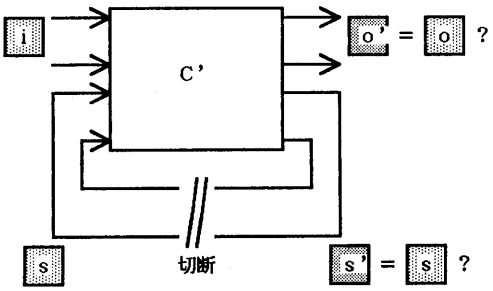


図4.1　非同期順序回路の検証

入力 J	入力 K	Q(t+1)	Q'(t+1)	
0	0	Q(t)	¬Q(t)	記憶
0	1	0	1	リセット
1	0	1	0	セット
1	1	¬Q(t)	Q(t)	ＴＦＦ

図4.3　ＪＫ－ＦＦの機能表

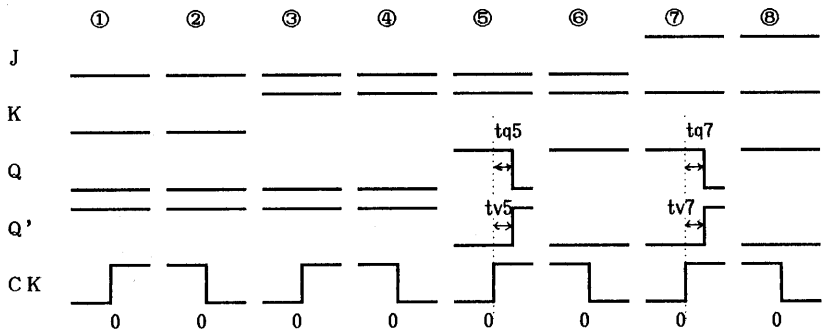


図4.4　入力パターン

可能性がある。特に、条件木は二分決定図表現とも見ることができるので、記号シミュレータとの連けいを検討中である。

4．応用例

4.1 非同期順序回路の検証

時間記号論理シミュレータは組合せ回路を対象としているが、非同期順序回路についても次に示す手続きによって、その組合せ回路部分のシミュレーションにより検証を行うことができる[8]。

Step 1　状態変数に対応するフィードバックループを切断することにより組合せ回路C'を得る。

Step 2　状態遷移表の各遷移について、遷移を引き起こす入力イベント系列iと、それに対する状態変数、出力のイベント系列s,oを求める。

Step 3　外部入力とフィードバック線に、それぞれStep 2で得たi,sを与えて、組合せ回路C'の時間記号シミュレーションを行い、回路の外部出力とフィードバック線上のイベント系列o',s'を得る。

Step 4　システム結果o',s'と、期待値o,sとを照合し条件木を求める。

上記の手続きにより、外部出力線およびフィードバック線上に期待したイベント系列が現れるための条件が得られる。

4.2 JK-FFの検証例

次に、図4.2のJK-FFの検証について説明する。この場合は、切断点がすべて出力線であるため、図4.3の機能表から全状態遷移に対応する入力パターンを容易に求めることができる。回路の対称性を考慮して、全部で８通りのパターン（図4.4）を得、そのそれぞれについてシミュレーションを行い動作を確認した。

はじめに、図4.7においてu1～u4の４箇所に遅延を設定しシミュレーションを行った結果、出力として正常動作しているものは得られなかった。次に、u5にも遅延を設定し同様にシミュレーションを行ったところ、ポジティブエッジ型のJK-FFとして正常に動作している場合が得られた（実際にTTLを用いた実験でも、この場所に遅延を入れないとpositive-edgeのT-FFとしては動作しない）。これにより、u5に遅延を入れることにより図4.2のJK-FFが正常に動作するということを確認することができた。以下、u1～u5に遅延を設定した場合について、4.1のstep1～step4に従って説明する。

・Step 1

図１のゲートg9とg10の出力の部分でフィードバックループを切断する。

・Step 2,3

外部入力は図4.9の入力パターンであり、切断された信号線上に発生するイベントの系列の期待値は、g9がQ'、g10がQと同じである。

・Step 4

入力パターン①～⑧について、前回のシミュレーションで得られた条件木も入力しシミュレーションを行う。以下では、条件木は展開した形で示す。また、全ての時間変数は正であるとする。

パターン①～④…g9とg10の出力には一つのイベントの系列しか発生せず、無条件で正常動作

パターン⑤

得られた条件木

```
tq5=u2+d2+d5+d6+d9+d10          (a)
tv5=d2+d5+d9                    (b)
u5>u1+u2+d2+d5+d6+d9+d10
u4+d2>d1
```

パターン⑥

パターン⑤で得られた条件木を入力してシミュレーション。

得られた条件木

```
tq5=u2+d2+d5+d6+d9+d10
tv5=d2+d5+d9
u5>u1+u2+d2+d5+d6+d9+d10
u4+d2>d1
```

パターン⑦

パターン⑥で得られた条件木を入力してシミュレーション。

得られた条件木

```
tq5=u2+d2+d5+d6+d9+d10
tv5=d2+d5+d9
tq7=u2+d2+d5+d6+d9+d10          (c)
tv7=d2+d5+d9                    (d)
u5>u1+u2+d2+d5+d6+d9+d10
u4+d2>u1
u5+d8<u3+d2+d3+d5+d6+d9
u3+d3>u2
```

パターン⑧

パターン⑥で得られた条件木を入力してシミュレーション。

得られた条件

```
tq5=u2+d2+d5+d6+d9+d10
tv5=d2+d5+d9
tq7=u2+d2+d5+d6+d9+d10
tv7=d2+d5+d9
u5>u1+u2+d2+d5+d6+d9+d10        (1)
u4+d2>u1                        (2)
u5+d8<u3+d2+d3+d5+d6+d9         (3)
u3+d3>u2                        (4)
```

この(1)～(4)の条件が、①～⑧の全てのパターンについて正常に動作するための条件となる。また、回路の対称性を考えると、(1)(3)よりそれぞれ、

```
u5>u1+u2+d3+d5+d6+d9+d10        (5)
u5+d7<u4+d2+d3+d5+d6+d10        (6)
```

が得られ、(1)から(6)が全ての１状態遷移に対して正常に動作するための条件となる。また、(a)～(d)の等式条件より、各入力パターンに対する信号線の値の変

化時刻も得ることができる。

一般には、フィードバック線に期待値を設定して得られた条件は、入出力関係を満たすための十分条件であるが、この例の場合は、外部出力についてのみ期待値を設定し条件を求めているので、上の(1)～(6)の条件はこのJK-FFが正常動作するための必要十分条件となる。8個の入力パターンに対するシミュレーション及び結果解析に要したCPU時間の合計は6.4秒であった。

5．おわりに

これまで、人手で行っていた時間記号論理シミュレーションの結果解析を、結果解析系の実現により自動化することができた。これにより、異常動作の有無だけではなく、正常動作あるいは異常動作をする時の遅延の分布に関する条件も容易に求めることができるようになった。

参考文献

[1] M. A. Breuer, A. D. Friedman: Diagnosis & Reliable Design of Digital Systems, p.308, ComputerScience Press (1976)

[2] 石浦菜岐佐，矢島脩三，時間記号シミュレーションについて，信学技報 VLD87-112 pp.39～46 (Dec. 1987)

[3] 石浦菜岐佐，安浦寛人，矢島脩三：時間優先評価アルゴリズムによる論理シミュレーションの高速化，情処論文，Vol.26，No.3，pp.459～466 (May 1985)

[4] 高橋瑞樹,石浦菜岐佐,矢島脩三：時間記号論理シミュレータについて,第36回情処全大3x-2,pp.1923～1924(Mar. 1988).

[5] 高橋瑞樹,石浦菜岐佐,矢島脩三：時間記号論理シミュレータの高速化と性能評価,第37回情処全大3U-3,pp.1579～1760(Sep. 1988).

[6] 北嶋雅哉,髙木直史,矢島脩三：論理関数のグラフ表現を用いた記号シミュレーション,情報処理学会研究報告,87-DA-40,pp.111～116(Dec. 1987).

[7] 湊真一,石浦菜岐佐,矢島脩三：真理値表表現を用いた記号シミュレータ,第37回情処全大3U-2,pp.1757～1756(Sep. 1988).

[8] 木村晋二,羽根田博正：系列集合論理シミュレーション手法に基づく非同期式順序回路の検証,信学技報VLD87-118,pp.15～22(Feb. 1988).