社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2003－ＳＬＤＭ－112 (12)
2003／11／28

非同期式回路自動合成の高速化について

米田 友洋† Chris MYERS††

† 国立情報学研究所 情報基盤研究系 〒101-8430 千代田区一ツ橋 2-1-2
†† Electrical and Computer Engineering, University of Utah Salt Lake City, Utah 84112 U.S.A
E-mail: †yoneda@nii.ac.jp, ††myers@ece.utah.edu

あらまし 本報告では，非同期式回路の仕様を出力線１本毎に分割し，その各々に対して論理合成を行うことにより，回路全体を高速に合成する手法について考察する．仕様の規模が大きくなると，論理合成に要するコストは急激に増加するため，分割により仕様を小規模に抑えることにより，本手法は従来合成不可能であった規模の回路合成を可能とする．本手法では，まず，着目する出力とそのトリガ信号のみを持つように，与えられた仕様を縮退する．もし，トリガ信号だけでは入力として不十分であり，合成に失敗する場合には，この縮退仕様の状態空間を調べ，最低限必要な入力線を決定する．本稿では，入力線決定アルゴリズムの詳細を述べるとともに，プロトタイプによる実験結果も示す．
キーワード 仕様分割，論理合成，スピードインデペンデント回路，STG

On the Acceleration of Asynchronous Circuit Synthesis

Tomohiro YONEDA† and Chris MYERS††

† Infrastructure Systems Research Division, National Institute of Informatics
2-1-2 Hitotsubashi Chiyoda-ku, Tokyo, 101-8430 Japan
†† Electrical and Computer Engineering, University of Utah Salt Lake City, Utah 84112 U.S.A
E-mail: †yoneda@nii.ac.jp, ††myers@ece.utah.edu

Abstract This paper presents a decomposition-based logic synthesis method for speed-independent circuit design such that each output is synthesized individually. Since the cost of logic synthesis increases rapidly as the specification becomes larger, this approach that keeps the specification small by decomposition can potentially be applied to synthesize circuits for which the conventional methods cannot be successfully applied. The proposed method begins by reducing the specification to include only the output of interest and its trigger signals. Next, if the synthesis process fails due to the lack of sufficient input signals, the reachable state space for this reduced specification is analyzed to determine a minimal number of additional input signals. This paper presents the details of this input decision algorithm, and also shows some experimental results.
Key words Decomposition, Logic Synthesis, Speed-Independent Circuits, STG

1. はじめに

論理合成に基づく非同期式回路の合成手法[1]～[3]は，非同期式回路合成の主要なアプローチである．この手法は，もう一つのアプローチである構文直接変換法[4]～[7]に比べ，高速で最適化された回路を合成できるという利点を持つが，通常，論理合成は仕様の状態空間を探索するため，コストが大きく，状態爆発問題を伴う．本研究は，特に信号遷移グラフ (Signal Transition Graph: STG) で表された仕様からのスピードインデペンデント回路の論理合成において，仕様を出力線１本毎に分割し，その分割された仕様毎に論理合成を行うことにより，論理合成のコストを大幅に削減することを目的とする．

出力線１本毎に論理合成を行うというアイデアは，Chuにより最初に提案された[8]．この方式では，まず１本の出力を選び，その出力に影響を与えない信号線を内部信号線と定義し直す．内部信号線は，その変化が外部からは観測できない信号線である．このような信号線は，外部から見た動作を変更することなしにSTGから除去することができる場合がある．そのような信号線は除去しても外部から見た動作は変わらないため，縮退したSTGに通常の論理合成手続きを適用することで，小さいコストで正しい回路を得ることができる．しかし，[8]では，２つの未解決問題があった．一つは，STGの縮退手続きが形式化されていない点である．マークグラフと呼ばれるような非常に簡単な構造のSTGに対しては，縮退手続きはほぼ自明であるが，一般のSTGに対してはそうではない．２番目に，着目する出力に対し，どのような信号線がその出力に影響を与えないかを決定する手続きが不明であった．最初の問題に対しては，最近 Vogler と Wollowski が[9]において双模倣関係に基づくSTGの縮退アルゴリズムを形式化し，また，Zheng と Myers は[10]において，時間付きSTGにおける縮退アルゴリズムを発表している．一方，２番目の問題に対しては，Puri と Gu が[11]において議論している．彼らは，ある信号を内部信号線とすることで完全状態符号 (Complete State Coding: CSC) 違反の数が増えないような信号は，合成手続きには不要な信号であることに着目し，もとのSTGの状態グラフを用いてそのような信号を見つけるグリーディアルゴリズムを開発した．この方法は，小規模なSTGに対しては，確かに有効であるが，もとのSTGの状態グラフを必要とするため，論理合成のコストを下げるという我々の目的には適さない．

本稿では，もとのSTGの状態グラフではなく，縮退STGの

状態グラフのみを用いて，与えられた出力に必要な信号線を決定するアルゴリズムを提案する．通常，縮退STGの信号線はもとのSTGの信号線のごく一部であるため，状態生成のコストを大幅に削減できる．また，提案するアルゴリズムはグリーディではなく，多少の発見的手法を用いるがほぼ決定的であるため，効率がよい．

なお，本研究では，CSCを持つSTGのみを対象とする．これは，特に本研究が対象としているような比較的大規模なSTGの場合，CSCソルバによる自動CSC変数挿入は効率よく働かないことが多く，むしろ，最初からCSCを持つようにSTGを構成するほうが現実的であることから，大きな制約とは考えていない．特に，高位仕様記述言語から[12]の方式で変換されたSTGは大規模であるため，従来の論理合成ツールでは合成が不可能であることが多いが，CSCを持つことは保証されている．このようなSTGが，最も適した対象である．

2. 定　　義

STGは7項組 $G=(P,T,F,\mu^0,l,In,Out)$ である．ここで，P はプレースの集合，T はトランジションの集合 $(P\cap T=\emptyset)$，$F\subseteq(P\times T)\cup(T\times P)$ はプレースとトランジションの接続関係，μ^0 は初期マーキング，$l:T\to In\cup Out\cup\{\lambda\}$ はラベリング関数，そして，In と Out は入出力信号線集合である．$\mathsf{sig}(G)$ は $In\cup Out$ を表すものとする．$l(t)=\lambda$ なるトランジション t を内部トランジションと呼ぶ．また，w-トランジションで，$l(t)=w$ なるトランジションを意味する．各トランジション t に対し，$\bullet t=\{p\in P\mid(p,t)\in F\}$，および，$t\bullet=\{p\in P\mid(t,p)\in F\}$ とする．トランジション t と t' は，$\bullet t\cap\bullet t'\neq\emptyset$ であるとき，競合するという．以下では，G,G_1 等を考えるとき，それらのそれぞれの要素を $P,T,\ldots,P_1,T_1,\ldots$ 等と表示する．

G のマーキング μ は P の任意の部分集合である．トランジション t は，$\bullet t\subseteq\mu$ を満足するとき発火可能である．発火可能なトランジション t が発火することにより新しいマーキング $\mu'=(\mu-\bullet t)\ \cup\ t\bullet$ が生成される．これを，$\mu\xrightarrow{t}\mu'$ と記す．トランジションの系列 $v=t_1t_2\cdots$ に対しても，同様に $\mu\xrightarrow{v}\mu'$ と記す．ここで，v が空なら μ は μ' に等しい．$\mu^0\xrightarrow{v}\mu'$ なる μ' が存在するとき，v をトレースと呼ぶ．G のすべてのトレースの集合を $\mathsf{trace}(G)$ と表す．同じトランジションが複数回発火する可能性があるが，トレースでは適当な方法（例えば，発火回数を属性として付加する）を用いて区別しているものとする．

各マーキングには，入出力線の信号値を表す状態ベクトルが対応する．異なるマーキングが等しい状態ベクトルを持つこともある．状態ベクトルの各要素は，通常0と1の値を取るが，出力に対応する要素はそれ以外に R と F の値を取る．R は，値としては0であるが，その出力が立ち上がり可能であることを表す．同様に，F は値としては1であるが，立ち下がり可能であることを表す．例えば，2つのマーキング μ と μ' が状態ベクトル(1010)と(101R)を持つ（4番目の要素が出力に対応するとする）とき，値としてはこれらは同じであるが，出力の振る舞いは異なる．このような状況をCSC違反であるといい，また，このような2つのマーキングをCSC違反ペアと呼ぶ．もし，STGがCSC違反ペアを持つとき，そのSTGはCSCを持たないという．STGがCSCを持たないと，そのSTGから回路は合成できない．

STGは，出力トランジションが他のトランジションを発火不可にしたり，他のトランジションに発火不可にされたりすることがない場合，出力セミモジュラであるという．これも，STGから回路合成を可能とするために必要な性質である．

STG G_1 と G_2 が以下の関係を満足するとき，G_1 から G_2 への模倣関係 S が存在するという．

- $(\mu_1^0,\mu_2^0)\in S$，
- 任意の $(\mu_1,\mu_2)\in S$ と $\mu_1\xrightarrow{t}\mu_1'$ に対し，$\mu_2\xrightarrow{v}\mu_2'$ かつ $(\mu_1',\mu_2')\in S$ なる v と μ_2' が存在する．ただし，$l_1(t)\neq\lambda$ であるなら $v=u_1u_2\cdots u_nt'$ $(n\geqq 0)$，$1\leqq i\leqq n$ に対し $l_2(u_i)=\lambda$，$l_1(t)=l_2(t')$ であり，また，そうでないなら，$v=u_1u_2\cdots u_n$ $(n\geqq 0)$，$1\leqq i\leqq n$ に対し $l_2(u_i)=\lambda$ である．

もし，B が G_1 から G_2 への模倣関係であり，B^{-1} が G_2 から G_1 への模倣関係であるとき，B は G_1 と G_2 の間の双模倣関係であるという．G_1 と G_2 は，そららの間に双模倣関係が存在するとき，双模倣であるという．

STG G，$x\in Out$，そして $x\in V$ なる $V\subset\mathsf{sig}(G)$ に対し，$\mathsf{abs}(G,V,x)$ は，$\mathsf{sig}(G)-V$ に対応する G のトランジションを内部トランジションで置き換え，入力信号集合 $V-\{x\}$ および出力信号集合 $\{x\}$ を持つ G と双模倣なSTGを表すものとする．$\mathsf{abs}(G,V,x)$ は，[9]等のネット縮約アルゴリズムを用いて求めることができ，通常，その状態空間は G の状態空間に比べて非常に小さい．

形式的な証明は省略するが，以下の定理が成り立つ．

[定理1] STG G はCSCを持ち，かつ，出力セミモジュラであるとする．ある出力 $x\in Out$ と，$x\in V$ なるある信号線集合 $V\subset\mathsf{sig}(G)$ に対し，$\mathsf{abs}(G,V,x)$ がCSCを持てば，$\mathsf{abs}(G,V,x)$ から合成した x に対するスピードインデペンデント回路（厳密には，複合ゲート実現，または，一般C素子実現[13]）は，もとの G に対して正しい．　□

これより，$\mathsf{abs}(G,V,x)$ がCSCを持つような信号集合 V を求めることが，本アプローチのキーとなる．以下，G の状態グラフを作ることなく，そのような信号集合を求める決定的に近いアルゴリズムを中心に，本合成手法を述べる．

```
decomposition_based_synthesis(G) {
  forall x ∈ Out {
    G_abs = obtain_synthesizable_abs(G,x)
    if (G_abs == ''impossible'') then abort
    C_x = logic_synthesis(G_abs)
  }
}
```

図1　最上位アルゴリズム

```
obtain_synthesizable_abs(G,x) {
  V = initial_input_set(G,x)
  loop {
    G_abs = obtain_abs(G,V,x)
    if (G_abs has CSC) then return G_abs
    CSCV =
        obtain_CSC_violation_trace_set(G_abs)
    forall g ∈ CSCV {
      candidate =
          analyze_CSCV_trace(g,G,V,x,μ0,ε)
      if (candidate == ''impossible'') then
          return ''impossible''
      add_constraints_matrix(candidate)
    }
    newV = solve_covering_problem()
    V = V ∪ newV
  }
}
```

図2　合成可能な縮退STGを求めるアルゴリズム

3. 合成アルゴリズムの概要

本手法の最上位アルゴリズムを図1に示す．このアルゴリズムでは，まず G_{abs} を求めるが，これは，CSCを持つ $\mathsf{abs}(G,V,x)$ である．もし，それを求めることが不可能な場合は，もとのSTGからも回路合成が不可能である（定理3参照）．このような場合は，処理を中断する．求めることができた場合には，Petrifyや ATACS等の通常の論理合成ツールを用いて，G_{abs} から x に対する回路を合成する．

G_{abs} を求めるアルゴリズムを図2に示す．このアルゴリズムでは，まず x のトリガ信号（その遷移により x を立ち上がり

あるいは立ち下がり可とした信号）を求め，それを初期の V とする．これは，トリガ信号はその出力の合成に必ず必要であるからである．

この初期信号集合 V に対し，次に $\mathsf{abs}(G,V,x)$ を求め，それがCSCを持つかどうか調べる．もし，CSCを持てば，$\mathsf{abs}(G,V,x)$ を返す．そうでないなら，$\mathsf{abs}(G,V,x)$ の状態グラフを調べることにより，CSC違反を引き起こす $\mathsf{abs}(G,V,x)$ のトレース集合（の一部）を取り出す．アルゴリズムは，その各トレース g を解析し，信号集合に加えるべき候補を求める．これらの候補に関する条件は，後述するように和積形式で蓄積され，最後に被覆問題を解くことにより，最適な候補 $newV$ を見つける．それを V に加え，新しい V を用いて再び，$\mathsf{abs}(G,V,x)$ を求める．

4. 縮退仕様の解析

この節では，`analyze_CSCV_trace` の詳細を述べる．これは，CSC違反を引き起こす $\mathsf{abs}(G,V,x)$ のトレース g を入力とするが，出力 x に対する信号集合の候補を見つけるためには，それに対応するもとのSTG G のトレースを求める必要がある．これを，G の状態グラフを構築せずに行うために，ガイドシミュレーションと呼ぶ手法を用いる．本節では，まずガイドシミュレーションの詳細を述べた後，それを解析する方法について述べる．

なお，以下では，$\mathsf{abs}(G,V,x)$ で使われている信号（すなわち V 中の信号）をインターフェース信号，それ以外の信号（$\mathsf{sig}(G)-V$ 中の信号）を非インターフェース信号と呼ぶ．対応するトランジションについても同様である．

4.1 ガイドシミュレーション

ガイドシミュレーションでは，縮退STGのトレース g に対し，それに対応する G のトレース f を求める．より正確には，f は，それから非インターフェース信号を除去したものが g と等しくなるようなものである．

互いに並行関係にあるインターフェーストランジションと非インタフェーストランジションはどのような順番で発火させてもよいが，ここでは，すべての発火可能な非インターフェーストランジションを発火させてからインターフェーストランジションを発火させることにする．これは，後述するように，アルゴリズムの簡単化に役立つ．このようなトレースの性質をレギュラー性と呼ぶ．

ガイドシミュレーションのアルゴリズムを図3に示す．g が非空なら，アルゴリズムはその最初のトランジション g_1 を選択する．もし，g_1 が発火可能で，それと並行な発火可能非インターフェーストランジションが存在するなら，レギュラー性を満足するため，それらの非インタフェーストランジションを発火させる．もし，g_1 が発火可能でないか，または，そのような非インターフェーストランジションが存在しないなら，g_1 のnecessary集合を求める．g_1 が発火可能なら，`necessary` は，g_1 およびそれと競合する発火可能な非インタフェース信号の集合を返す．g_1 が発火不能なら，`necessary` は，発火可能な非インターフェース信号の集合のうち，それが含むどのトランジションも発火させないなら g_1 は決して発火可能とはならない，という性質を持つもの返す．例えば，図4で，g_1 はインターフェーストランジション，それ以外はすべて非インターフェーストランジションであるとする．図4 (a) では，g_1 のnecessary集合は $\{g_1,t_1\}$ となる．これは，g_1 が発火可能で，t_1 は g_1 と競合しているからである．図4 (b) の場合，necessary集合は $\{t_1\}$ または $\{t_2\}$ となる．これは，t_1 が発火しないと g_1 は決して発火可能とならないし，t_2 に対しても同じことがいえるからである．一方，図4 (c) の場合，t_1 が発火しなくても，t_2 の発火により g_1 は発火可能となるかもしれないし，また，その逆が生じるかもしれない．ここで言えることは，t_1 と t_2 のどちらも発火しないなら，g_1 は発火可能とはならないということだけである．よって，この場合の g_1 のnecessary集合は $\{t_1,t_2\}$ となる．

```
analyze_CSCV_trace(g,G,V,x,μ,f) {
  if (g is empty) {
    candidate = find_inputs(f,G,V,x)
    return candidate
  }
  g1 = pick the first transitin of g
  if (g1 ∈ enabled(μ) and ∃t ∈ enabled(μ)
      s.t. t ∈ NonIF ∩ concur(g1)) {
    μ' = fire(μ,t)
    result =
      analyze_CSCV_trace(g,G,V,x,μ',f·t)
  }
  else {
    N = necessary(μ,g1)
    if (N is empty) return ''backtrack''
    forall t ∈ N {
      μ' = fire(μ,t)
      g' = g
      if (t == g1)
          remove first transition of g'
      result =
          analyze_CSCV_trace(g',G,V,x,μ',f·t)
      if (result ≠ ''backtrack'')
          exit forall
    }
  }
  return result
}
```

図3 ガイドシミュレーションアルゴリズム

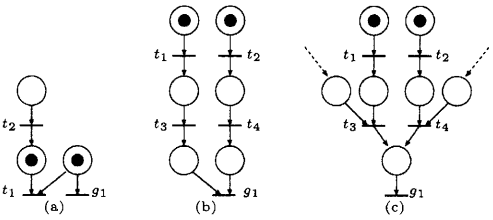


図4 Necessary集合の構築

このようにして求めたnecessary集合が二つ以上のトランジションを含む場合，それらを一つずつ選び，再帰的に発火させていく．これは，g に対応する f を求めたいのであるが，着目しているマーキングでは発火すべきトランジションが正確に決められないからである．これは，アルゴリズムはバックトラックに依存することを意味する．すなわち，もし不適切なトランジションを発火させた場合には，やがて g のいずれかのトランジションを発火させることができなくなり，空のnecessary集合が返されることになる．この場合には，アルゴリズムは“backtrack”を返すことにより，バックトラックを行う．

このバックトラック機構により，g に対するもとのSTG G のトレース f を求めることは常に可能である．しかし，G が多くの競合関係を含むとき，バックトラックが頻発し，効率が低下する．より現実的な解として，初期縮退STGを求める際に，いくつかの競合トランジションを除去せずに，保持しておくという方法がある．このような競合トランジションの情報は，設計者ならよく知っているはずであるから，設計者の情報によりこのように保持すべきトランジションを決定するのが有効な方法であると考える．そこで，我々のツールでは，STGの入力ファイル中に，コメントを利用してこの情報を指示できるようになっている．

上記のようにして発火させられたトランジションは，各再帰呼び出しにおいて f に追加され，g が空になった段階で，G のトレース f が完成する．このような f は次に，`find_inputs` により解析される．

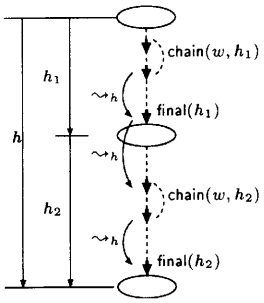


図 5　Confinement 関係

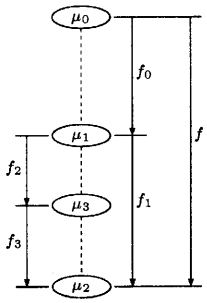


図 6　f のラベル付け

4.2　追加入力信号の決定

obtain_CSC_violation_trace_set で求められる abs(G,V,x) の各 CSC 違反トレース g は，$g=\langle g_0,g_1\rangle$，$\mu_0' \xrightarrow{g_0} \mu_1'$ (μ_0' は abs(G,V,x) の初期マーキング)，$\mu_1' \xrightarrow{g_1} \mu_2'$ の形式であるものとする．ただし，μ_1' と μ_2' は CSC 違反ペアであり，μ_1' と μ_2' の間には CSC 違反ペアの状態ベクトルと同じ状態ベクトルを持つマーキングは存在しないものとする．さらに，そのような g からガイドシミュレーションにより生成された f も $\langle g_0,g_1\rangle$ に対応した $\langle f_0,f_1\rangle$ の形式を持ち，f_0 と f_1 はそれぞれインターフェーストランジションで終わっているものとする．f_0 および f_1 により得られるマーキングを μ_1 および μ_2 とする（図 6 参照）．

f 中のインターフェーストランジション a と b において，もし a が b より先に発火するなら，$(a,b)\in R^f_{order}$ と記す．f 中の 2 つの（インターフェースでも非インターフェースでもよい）トランジション t_1 と t_2 において，もし t_1 の結果として t_2 が生じる場合には，$(t_1,t_2)\in R^f_{cause}$ と記す．t_1 と t_2 が R^f_{order} と R^f_{cause} の和集合の推移的閉包の要素，すなわち，$(t_1,t_2)\in (R^f_{order}\cup R^f_{cause})^*$ なら，t_1 を t_2 の祖先といい，$[t_1 \rightsquigarrow_f t_2]$ と記す．この祖先関係は，着目している g に関する因果関係を表すと考える．祖先関係に R^f_{order} も含めるのは，並行なインターフェーストランジション，すなわち，g 中の並行トランジションの発火順序が変わった場合は，異なる CSC 違反ペアによる異なる CSC 違反トレースとなり，そのような異なる CSC 違反トレースは obtain_synthesizable_abs 中の forall ループ中で別に処理されるからである．この祖先関係は，我々のアルゴリズムで非常に重要な役割を果たすので，find_input の最初に求めている．これは，実際には occurrence net [14] とほぼ同等のデータ構造を用いて実現している．

マーキング μ_1 と μ_2 に関する CSC 違反を解消するには，f_1 で奇数回の遷移を起こす非インターフェース信号 w を探す必要がある．もし，w-トランジションが f_1 で確実に奇数回発火するなら，信号 w が μ_1 と μ_2 で異なる値を取るため，その CSC 違反は解消できる．しかし，並行トランジションが存在すると，確実に奇数回発火することを保証できなくなる．そこで，次の用語を定義する（図 5 参照）．

final(h) は，発火シーケンス h の最後のトランジションを表し，chain(w,h) は，h でその順に発火する w-トランジションのシーケンス $(t_0t_1t_2\cdots t_{n-1})$ を表す（すなわち，各 i に対し，$l(t_i)=w$ が成り立つ）．h_2 がインターフェース信号で終わるようなトレース $h=\langle h_1,h_2\rangle$ と信号 w において，以下が成り立つとき w は h において h_2 に confine されるという．

- w-トランジションが h_1 で発火するなら，chain(w,h_1) の最後のトランジション t_{e_1} に対し，$[t_{e_1}\rightsquigarrow_h \text{final}(h_1)]$ が成り立ち，かつ，
- w-トランジションが h_2 で発火するなら，chain(w,h_2) の最初と最後のトランジション t_{s_2} と t_{e_2} に対して，$[\text{final}(h_1)\rightsquigarrow_h t_{s_2}]$ かつ $[t_{e_2}\rightsquigarrow_h \text{final}(h_2)]$ が成り立つ．

トレースのレギュラー性と h_2 がインターフェース信号で終わるという仮定から，final(h_2) と並行な非インターフェーストランジションはどれも final(h_2) より前に発火する．よって，final(h_2) は，h_2 の後に発火する w-トランジションの祖先となる．このため，上記の条件では，h_2 の後の w-トランジションについては考慮する必要がない．$f=\langle f_0,f_1\rangle$ においては，f_0 もインターフェース信号で終了する．よって，この場合は上記の $[\text{final}(h_1)\rightsquigarrow_h t_{s_2}]$ も不要となる．しかし，他の場合では，h_1 が必ずしもインターフェース信号で終わるとは限らないため，この条件は必要である．

w が h において h_2 に confine され，chain(w,h_2) が奇数個のトランジションを含むとき，w は h において h_2 に odd-confine されるという．even-confine も同様に定義できる．

f_1 において，最初に出現したインターフェーストランジションに着目し，そのトランジションにより f_1 を分割し，f_2 と f_3 を定義する．すなわち，$f_1=\langle f_2,f_3\rangle$ であり，f_2 は f_1 における最初のインターフェース信号で終わることになる．図 6 に，f_0 から f_3 の関係を示す．このとき，以下の補題が成立する．

[補題 1]　もし，w-トランジションが f において f_1 に odd-confine され，かつ，f_2 が w-トランジションを含まないとき，f における CSC 違反は V に w を加えることで解消できる．□

f_2 が非インターフェーストランジションを含む場合には，それにより得られるマーキングは V の信号線に関してすべて同じ状態ベクトルを持つ．これは，μ_3 は f_1 の最初のインターフェーストランジションにより得られたマーキングだからである．よって，その信号を V に加えても，μ_1 と μ_2 の CSC 違反は解消できるが，その非インターフェーストランジションが生成したマーキングと μ_2 が CSC 違反を起こす．例えば，図 7 において，非インターフェース信号 w を V に加えるとする．これにより，μ_1 の状態ベクトルは $(010R)$，μ_2 の状態ベクトルは (1100) となるが，最初の $w+$ により得られる状態ベクトルは $(110R)$ となり，μ_2 と CSC 違反を起こす．

しかし，もし，非インターフェース信号 u が，f において f_4 に odd-confine され，かつ，f_4 における最初の u-トランジションが f_3 内で発火するなら，上記のような CSC 違反は，w に加えて u も V に追加することで解消できる．ここで，f_4 は f_2 における最初の w-トランジション以降，f_1 の最後までの発火シーケンスである．図 7 の最後の列に，このような w と u を加えることにより CSC 違反が解消されるようすを示す．このような信号 u を f において w に対する必須信号と呼ぶ．必須信号のより正確な定義は以下のようになる．

w は f において f_1 に odd-confine されており，chain$(w,f_1)=t_1t_2\cdots t_n$ であり，f_2 で発火する最後の w-トランジションを t_k とする．各奇数整数 $i\leq k$ に対し，非インターフェース信号

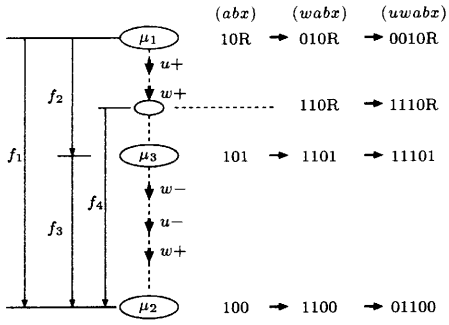


図 7　必須信号の追加による CSC 違反の解消

$u_i(\neq w)$ は，以下を満足するとき w に対して必須である．

（1）　u_i は f において $h_i(t_i$ 以降 f_1 の最後まで) に odd-confine され，かつ，

（2）　h_i の最初の u_i-トランジション t_{u_i} に対して，(i) t_{i+1} が存在しないか，または，$[\mathsf{final}(f_2) \rightsquigarrow_f t_{i+1}]$ が成り立つなら，$[\mathsf{final}(f_2) \rightsquigarrow_f t_{u_i}]$ 成り立ち，(ii) そうでない場合は，$[t_{i+1} \rightsquigarrow_f t_{u_i}]$ が成り立つ．

各奇数整数 $i \leqq k$ に対し，いずれかの必須信号 u_i を含むような必須信号の集合を w に対する必須信号組集合と呼ぶ．

図 8 には，w の必須信号組集合のひとつ $\{u_1, u_3\}$ を示している．必須信号組集合中の t_i に関する必須信号は，t_i と t_{i+1} の間のマーキングが取る共通の状態ベクトルと μ_2 の状態ベクトルを区別するためのものである．そのため，最初の u-トランジションである t_{u_i} は，確実に t_{i+1} の後に発火しなくてはならない．さらに，f_2 の最後の部分では，t_{i+1} に関する条件は $\mathsf{final}(f_2)$ に対して課せられることに注意する（上記，必須信号に対する条件の 2 番目参照）．以上の議論より，補題 1 を一般化した以下の定理が成り立つ．

[定理 2]　f における CSC 違反は，f において f_1 に odd-confine される非インターフェース信号 w，および，その必須信号組集合を V に加えることで解消できる．　□

定理 2 は CSC 違反を解消するための十分条件であることに注意する．すなわち，たとえその条件が満足されなくても，CSC 違反が解消されることがある．例えば，図 9 の STG G とその出力 c を考える．そのトリガ信号は a と b であるので，まず $\mathsf{abs}(G, \{a, b, c\}, c)$ を考えるが，これは，CSC 違反トレース $g = c+, a+, b+, b-$ を持ち，$g_0 = c+, a+$ および $g_1 = b+, b-$ となる．ガイドシミュレーションは $f_0 = c+, x1+, x2+, a+$ と $f_1 = b+, x1-, b-$ を生成する．ここで，$x1$-トランジション

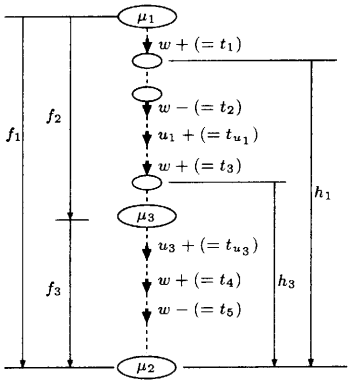
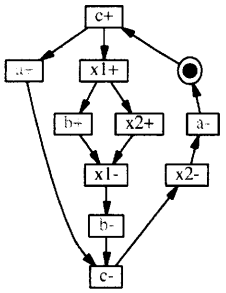


図 8　複数の必須信号が必要な例



INPUTS: a,b
OUTPUTS: c,x1,x2

図 9　定理 2 の条件では解決できない例

は f_1 に奇数回現れるが，$x1+$ と $a+$ は並行関係にあるため，$[x1+ \not\rightsquigarrow_f a+]$ となり，信号 $x1$ は f において f_1 に confine されない．他に，f_1 に confine される信号は存在しない．よって，どの非インターフェース信号も定理 2 の条件を満足しない．しかし，G 自体は CSC を持つ．すなわち，$x1$ と $x2$ は定理 2 の条件を満足しないにも関わらず，それら双方を加えることで CSC 違反を解消できることになる．

一方，与えられた CSC 違反が解消されないための十分条件も求めることができる．誌面の都合で証明は省略するが，以下の定理が成り立つ．

[定理 3]　もし，f_2 で $[t \rightsquigarrow_f \mathsf{final}(f_2)]$ なるトランジション t（インターフェースでも非インターフェースでもよい）が発火しており，任意の非インターフェース信号 u について，u が f において f_1 の t 以降に even-confine されるなら，その CSC 違反は決して解消できない（図 10 参照）．　□

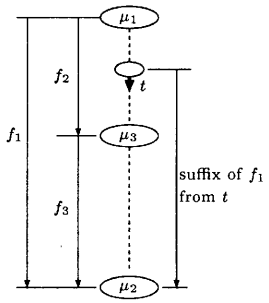


図 10　解消できない CSC 違反の例

定理 3 の条件が満足されると，`find_input` は “impossible” を返す．そうでなければ，CSC 違反を解消する非インターフェース信号を探す．通常，多くの信号の組が候補となりうる．例えば，a, b, c, d が w となり得，c の必須信号組集合として $\{e\}$ が，d の必須信号組集合として，$\{f\}$ と $\{g\}$ が存在するとする．これら全体の条件は，

$$a \vee b \vee (c \wedge e) \vee (d \wedge (f \vee g))$$

と表現できる．被覆問題として解くため，これを和積形に直すと以下のようになる．

$$(a \vee b \vee c \vee d)(a \vee b \vee c \vee f \vee g)(a \vee b \vee e \vee d)(a \vee b \vee e \vee f \vee g)$$

着目している CSC 違反を解消するには，各 AND 項が満足される必要がある．すべての CSC 違反ペアに対して上記のような条件を求め，`obtain_synthesizable_abs` にて被覆問題を解

表 1 実験結果 (1)

Circuit	(#I,#O)	Petrify			Proposed method		
		CPU (s)	Mem(MB)	area	CPU(s) (Petrify+other)	Max(MB)	area
cb	(10,10)	9.6	4.6	82	3.3 = (2.9+0.3)	3.4	82
cachem	(11,16)	219.5	7.7	122	39.3 = (38.4+0.9)	3.9	123
lf6	(21,41)	† (≥59272.4)	(≥742)	-	106.3 = (101.7+4.6)	4.5	200

(†: BDD manager overflow: ≥ 30000000 nodes)

表 2 実験結果 (2)

Circuit	(#I,#O)	Petrify			Proposed method		
		CPU (s)	Mem(MB)	area	CPU(s) (Petrify+other)	Max(MB)	area
FIR5_2mul_csc	(7,19)	78.8	7.8	151	32.2 = (31.3+0.9)	4.3	150
IIR2_2mul_d_csc	(7,19)	240.2	12.3	150	184.1 = (182.3+1.8)	5.5	152
LMS4_pr12_csc	(9,18)	354.6	18.6	177	26.3 = (25.2+1.1)	4.3	177

くことにより，すべての AND 項を満足させる最適な信号の集合が求められる．

定理 3 の条件が成立しない限り，定理 2 の条件を満足する非インターフェース信号が存在しなくても，図 10 の例からもわかるように，何らかの非インターフェース信号を加えることは意味がある．そこで，`find_input` はそのような場合には単純な発見的手法を用いて，加えるべき信号を決定している．

5. 実験結果

提案手法を C 言語と Perl を用いて実装し，UNIX ワークステーション (Pentium 4, 2.8GHz, 4GB メモリ) 上で動作を確認した．ここでは，いくつかの例について，従来手法との比較を行う．

表 1 は，非同期式プロセッサ TITAC2 [15] の命令キャッシュ制御回路の一部について，従来手法である Petrify と提案手法を用いた場合の合成時間，合成された回路のエリアサイズ等を比較したものである．また，表 2 は [16] で作成されたベンチマーク回路の一部について同様な比較を行ったものである．

以上の結果より，提案手法では，従来手法が合成不可能な規模の回路も扱うことができ，また，提案手法により合成された回路の質も，エリアサイズを見る限り大幅に悪くなっていることはないといえる．

6. 結　論

本稿では，回路分割に基づくスピードインデペンデント回路の合成方式について，信号線決定アルゴリズムを中心に検討した．実験結果からは，特に大規模な仕様に対しては大幅な合成時間の短縮が可能となることがわかった．今後，時間付き STG への対応，および，CSC ソルバへの応用等を行いたい．

謝　辞

ベンチマーク回路を提供いただいた東京大学斉藤氏，および，実装の一部を行ってくれた東工大音田氏に感謝したい．

文　献

[1] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, March 1997.

[2] P. A. Beerel, C. J. Myers, and T. H.-Y. Meng. Covering conditions and algorithms for the synthesis of speed-independent circuits. *IEEE Transactions on Computer-Aided Design*, March 1998.

[3] R. M. Fuhrer, S. M. Nowick, M. Theobald, N. K. Jha, B. Lin, and L. Plana. Minimalist: An environment for the synthesis, verification and testability of burst-mode asynchronous machines. Technical Report TR CUCS-020-99, Columbia University, NY, July 1999.

[4] Steven M. Burns and Alain J. Martin. Syntax-directed translation of concurrent programs into self-timed circuits. In J. Allen and F. Leighton, editors, *Advanced Research in VLSI*, pages 35–50. MIT Press, 1988.

[5] Kees van Berkel, Joep Kessels, Marly Roncken, Ronald Saeijs, and Frits Schalij. The VLSI-programming language Tangram and its translation into handshake circuits. In *Proc. European Conference on Design Automation (EDAC)*, pages 384–389, 1991.

[6] Joep Kessels and Ad Peeters. The Tangram framework: Asynchronous circuits for low power. In *Proc. of Asia and South Pacific Design Automation Conference*, pages 255–260, February 2001.

[7] Doug Edwards and Andrew Bardsley. Balsa: An asynchronous hardware synthesis language. *The Computer Journal*, 45(1):12–18, 2002.

[8] Tam-Anh Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*. PhD thesis, MIT Laboratory for Computer Science, June 1987.

[9] Walter Vogler and Ralf Wollowski. Decomposition in asynchronous circuit design. In J. Cortadella, A. Yakovlev, and G. Rozenberg, editors, *Concurrency and Hardware Design*, volume 2549 of *Lecture Notes in Computer Science*, pages 152–190. Springer-Verlag, 2002.

[10] H. Zheng, E. Mercer, and C. J. Myers. Modular verification of timed circuits using automatic abstraction. *IEEE Transactions on Computer-Aided Design*, 22(9), September 2003.

[11] Ruchir Puri and Jun Gu. A modular partitioning approach for asynchronous circuit synthesis. In *Proc. ACM/IEEE Design Automation Conference*, pages 63–69, June 1994.

[12] Tomohiro Yoneda and Chris Myers. Synthesizing timed circuits from high level specification languages. *NII Technical Report*, NII-2003-003E, 2003.

[13] Myers(著) 米田 (訳). 非同期式回路の設計 *(Asynchronous Circuit Design)*. 共立出版, 2003.

[14] Kenneth McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In G. v. Bochman and D. K. Probst, editors, *Proc. International Workshop on Computer Aided Verification*, volume 663 of *Lecture Notes in Computer Science*, pages 164–177. Springer-Verlag, 1992.

[15] Akihiro Takamura, Masashi Kuwako, Masashi Imai, Taro Fujii, Motokazu Ozawa, Izumi Fukasaku, Yoichiro Ueno, and Takashi Nanya. TITAC-2: An asynchronous 32-bit microprocessor based on scalable-delay-insensitive model. In *Proc. International Conf. Computer Design (ICCD)*, pages 288–294, October 1997.

[16] H. Saito. *Synthesis of Globally Delay Insensitive Locally Timed Asynchronous Circuits from Register Transfer Level Descriptions*. PhD thesis, University of Tokyo, 2003.