高並列度仕様からの非同期式回路合成のための信号遷移挿入手法

松本 敦[†] 米田友洋[††]

論理合成法をベースにした制御回路の自動生成手法は，最適に近い回路が得られる一方で並列性の高い記述に対する合成のコストが高いという欠点を持つ．本論文では，出力線ごとに縮退させた信号遷移グラフを用いる論理合成ツールを対象とした，高並列度記述からの信号遷移グラフ生成時における付加処理を提案する．信号遷移グラフ生成のときに，高並列度の原因となる箇所にあらかじめ信号遷移を挿入しておき，合成時の状態数を低減させることが提案手法の狙いである．また，使用する演算器の遅延時間情報を用いることで，回路の動作速度を劣化させることなく合成を実現することが可能となる．本論文では，いくつかの高並列度記述に対し提案手法を組み込んだ論理合成を行い，その結果を合成時間と動作速度について評価し，提案手法の有効性を確認した．

Transition Insertion for Handling Highly Concurrent Asynchronous Specification

Atsushi Matsumoto[†] and Tomohiro Yoneda[††]

In our state based logic synthesis approach for asynchronous circuits, it sometimes occurs that handling highly concurrent specifications costs too much. This paper proposes a method that inserts new signals in order to reduce the synthesis costs. These additional signals are inserted in non-critical paths selectively in order that the synthesized circuit performance is affected as little as possible. The experimental results show that the proposed method can successfully synthesize several circuits that the previous logic synthesis tools could not handle.

1. 背 景

演算動作やレジスタ操作を可能な限り並列に行うことが，回路の動作速度向上に寄与することは自明である．この議論は，非同期式回路に関してもあてはまる．しかし，回路自動合成手法の 1 つである論理合成法をベースにした非同期式回路設計にとっては，高並列化が合成時の使用メモリ量と合成に必要な時間を著しく増大させる可能性がある．

非同期式回路の自動合成手法には構文直接変換法と論理合成法があり，それぞれ利点，欠点を有する．

構文直接変換法[1)~6)] では，仕様に含まれる各構文を対応する回路モジュールに置き換えて動作を実現する．この手法は，合成時間が仕様規模の影響を受けにくく，合成そのものが高速であるという利点と，得られる回路の性能が高くなく，そのグローバルな最適化が難しいという欠点を持つ．この手法で高並列度の記述を扱う場合には，適切な分岐，統合用モジュールを用意することで仕様動作の実現が可能であると考えられる．

もう一方の，論理合成法[7)~9)] では，起こりうる信号遷移の因果関係から，状態空間探索を行うことなどにより最適化された論理関数を求め，それを実際の回路にマッピングする．この手法の利点は前述の構文直接変換法に比べ高性能な回路を得ることができる点であり，欠点は論理関数を求めるための計算コストが大きく，合成に要する時間が仕様規模の影響を受けやすい点である．この欠点により，大規模な仕様記述に対しては，論理合成を現実的な時間で行うことができない．

本研究室で開発している手法を含む最近の研究[10)~12)] では，出力線 1 本ごとに，該当出力線の遷移と因果関係を持たない信号線の遷移を信号遷移グラフから除外した，縮退信号遷移グラフを用いた論理合成を行う．これにより，一度に扱う信号遷移グラフのサイズ，つまりそれから生成される状態グラフのサイズが小さくなるので，既存の論理合成手法で合成できない規模の仕様記述の合成が可能になることがある．しかし，高い並列性を持つ信号遷移グラフを合成する場合，縮退処理が効果的に作用せず，状態グラフ生成時に状態爆発を起こし，有効な時間内での合成が難し

† 東京工業大学
Tokyo Institute of Technology
†† 国立情報学研究所
National Institute of Informatics

い場合がある．

経験から，我々が開発している合成手法において，ある出力信号遷移の直前に起こりうる信号線の集合の要素数と，その出力信号線に関して縮退された信号遷移グラフから生成された状態グラフの大きさ，さらには合成に要する時間に相関があることが分かっている．そこで，本論文では並列性の高い信号遷移グラフからの合成を行う際に，数多くの動作が並列に行われている部分に新たな信号遷移を挿入することで，合成対象である縮退信号遷移グラフの並列度合を減少させ，実用的な時間での制御回路の合成を実現する手法を提案する．さらに，提案手法では，並列動作の中の遅延時間の短い箇所に優先的に信号遷移を追加することで，動作時間への影響を最小限に抑えることが可能である．

本論文は以下のように構成される．2 章では，我々が開発している回路合成手法の概要と信号遷移グラフ生成手法に関して述べる．3 章では，本論文で提案する信号線追加のためのアルゴリズムに関して記す．4 章では，提案手法を適用した信号遷移グラフからの回路合成実験の結果を示し，それを評価する．また，他の論理合成手法との比較実験を行う．最後に，5 章で本論文の結論と将来的な課題を提示する．

2. 論 理 合 成

本章では，我々が開発している高位言語からの非同期式回路自動生成システム（nutas）[13] の，制御回路合成に関する部分について述べる．その後に，本論文で使用する定義と，予備実験の結果を示す．

2.1 合成手法概観

我々が開発している高位合成手法の全体像を**図 1** に示す．

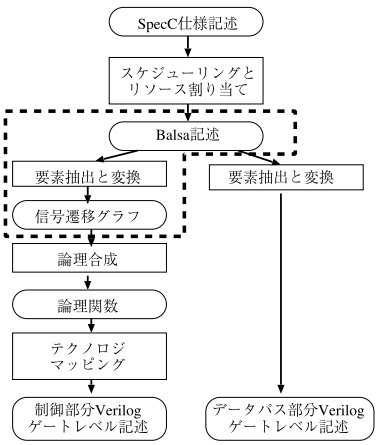

図 1 高位合成手法（nutas）全体図
Fig. 1 Process flow of high-level synthesis (nutas).

高位合成の最初の手順で，本システムが高位仕様記述として想定している SpecC 言語で書かれた仕様記述と演算器やレジスタに関する制約をもとに，演算器スケジューリングとレジスタアロケーションを行う．その結果を，非同期式回路仕様記述言語である Balsa 言語[14] で出力する．中間言語としてこの Balsa 言語を用いる理由は，制御回路を合成する際に必要な性質である完全状態コード（CSC 性）を満足するための制御構文構成を，Balsa 言語が持つためである．

次に，得られた Balsa 記述を構文解析し，その制御動作を示す信号遷移グラフを生成する．本論文での提案手法，すなわち高並列度記述に対する処理は，この図 1 の破線内の Balsa 記述から信号遷移グラフを生成する段階で行われる．本手法で想定する Balsa 構文を以下に示す．

- 並列実行文：$X \equiv Y_1 \parallel Y_2 \parallel \cdots \parallel Y_n$
- 順次実行文：$X \equiv Y_1; Y_2; \cdots; Y_n$
- While 文：$X \equiv$ while $cond$ then Y end
- If-else 文：$X \equiv$ if $cond$ then Y_1 else Y_2 end
- ループ文：$X \equiv$ loop Y end
- ポート入力文：$X \equiv in\_port$ -> reg
- ポート出力文：$X \equiv out\_port$ <- reg
- レジスタ転送文：$X \equiv d\_reg$:= s_reg

これらの構文を，対応する部分信号遷移グラフに変換し適切に接続することで，記述全体に対応する信号遷移グラフを得る．

そして，開発した論理合成ツールに信号遷移グラフを与える．ここでの論理合成とは信号遷移グラフで示された信号遷移の因果関係，つまり変化順序を満足する論理回路を求めることを指し，各信号線の立ち上がり条件と立ち下がり条件が積和形の論理関数として求められる．本研究室で開発している論理合成ツールは，各出力ごとに縮退信号遷移グラフを構成し，そこから得られる状態グラフの探索を行うことで論理関数を得るという方式をとっている[11]．

信号遷移グラフの縮退処理について例示する．**図 2** (a) に示す信号遷移グラフを信号線 c について縮退させると図 2 (b) のような小規模な信号遷移グラフになる．つまり縮退処理とは信号線 c の合成に必要がないと判断された信号遷移をグラフ上から取り除くことを指す．この図 2 (b) に示す縮退信号遷移グラフから，状態グラフを生成し状態探索を行うことで，図 2 (c) に示す制御回路に相当する論理関数を得る．

この縮退状態グラフを用いた手法では，各出力線間における論理関数に含まれる積項の共有を考慮しないことによるゲート数の増加の可能性がある一方で，従

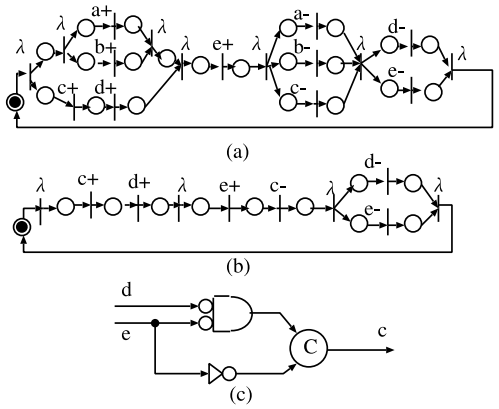

図 2 信号遷移グラフと縮退信号遷移グラフと制御回路
Fig. 2 STG, decomposed STG and control circuit.

来の論理合成ツールで扱えない規模の仕様に対してでも，十分現実的な時間で合成することができる．これは，図 2 (a) の信号遷移グラフから生成された状態グラフに比べて，図 2 (b) の信号遷移グラフから生成された状態グラフの規模が小さくなるからである．

最後に得られた論理関数にテクノロジマッピングを施し，図 2 (c) に示す制御回路の Verilog ゲートレベル記述を得る．

以上の手順によって，高位仕様 SpecC 記述から，適切に動作する制御回路を自動的に合成することができる．

2.2 信号遷移グラフ生成

本節では，Balsa 言語から生成される信号遷移グラフについて示す．なお，本論文では，提案する信号遷移挿入アルゴリズムに関係のある構文の信号遷移グラフへの変換のみを示す．

2.2.1 信号遷移グラフの定義

信号遷移グラフ G は，$G = (P, T, F, \mu^0, l, In, Out)$ で表され，各要素は，プレース集合，トランジション集合，プレースとトランジションの接続関係，初期マーキング，ラベル関数，入力信号線集合，出力信号線集合を表す．ラベル関数 $l(t) : T \rightarrow (In \cup Out) \times \{+, -\}$ はトランジションとそれに関連づけられる信号遷移の関係を表す．なお，トランジション集合の中には，ラベル関数によって入出力信号に関連づけられない，ダミートランジション λ も含まれる．ダミートランジションは，信号遷移グラフ上での分岐や合流を表現するために用いられる．また，$\bullet t$，$t\bullet$ はそれぞれ，トランジション t の入力プレース集合，出力プレース集合を表す．

我々が開発している信号遷移グラフへの変換手法において，Balsa 構文の文 X からは，立ち上がりブロック $\mathsf{N_{X+}}$ と立ち下がりブロック $\mathsf{N_{X-}}$ の 2 つから構成される部分信号遷移グラフが生成される．これらを構文解析時の情報をもとに接続することで，記述全体に対応する信号遷移グラフを得る．

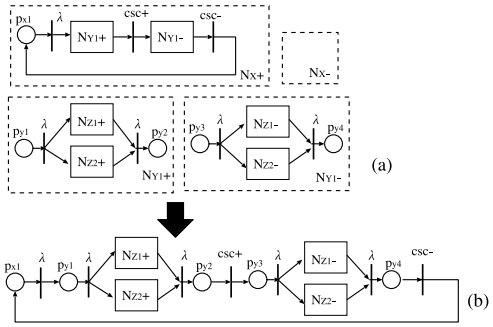

図 3 部分信号遷移グラフと信号遷移グラフ全体
Fig. 3 Partial STG and whole STG.

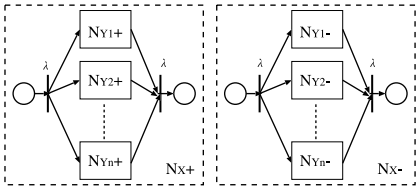

図 4 並列実行文に対する部分信号遷移グラフ
Fig. 4 Partial STG for parallel structure.

例として “*loop* $Z1$ || $Z2$ *end*” という Balsa 記述からの変換を考える．Loop 文と並列実行文から生成される部分信号遷移グラフを，**図 3** (a) に示す（$X \equiv loop\ Y\ end$, $Y \equiv Z1$ || $Z2$）．これを階層的に接続すると図 3 (b) のようになり，記述全体の信号遷移グラフを得ることができる．

また，以降の記述では文 X から生成されるブロック対について，$(\mathsf{N_{X+}})^* = \mathsf{N_{X-}}$，$(\mathsf{N_{X-}})^* = \mathsf{N_{X+}}$ と表現するものとする．この表現はブロック集合についても用いられ，ブロック集合の各要素の対になるブロックからなる集合を表すものとする．

2.2.2 並列実行文（$X \equiv Y_1||Y_2||\cdots||Y_n$）

並列実行文はそれに含まれる各動作を並列に実行するための構文である．並列実行文から生成される信号遷移グラフを**図 4** に示す．

並列実行文において多数の文が実行される場合，その直後の信号遷移の駆動信号遷移候補が増加する．これは，論理合成時の状態グラフの肥大化をもたらす．並列実行文から生成される立ち上がり，立ち下がりの両ブロックを，それに含まれる各文をすべて実行するので *AND_join* ブロックと呼び，後述の *OR_join* ブロックと区別する．なお，駆動信号遷移の形式的定

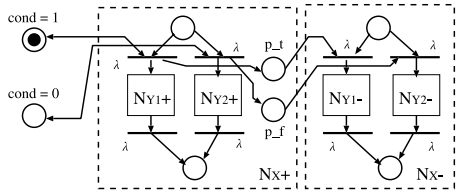

図 5 If-else 文に対する部分信号遷移グラフ
Fig. 5 Partial STG for If-else statement.

義は後に行うが，直観的にはある信号遷移の直前に起こりうる信号遷移のことを指し，信号遷移グラフを縮退させたときに削除されず（図 2 (b) の d や e），その結果として制御回路（図 2 (c)）の入力となる信号遷移のことを指す．

2.2.3 If-else 文 （$X \equiv$ if $cond$ then Y_1 else Y_2 end）

If-else 文は，条件式 $cond$ が真，すなわち 1 であれば文 Y_1 を，そうでなければ文 Y_2 を実行する．If-else 文から生成される信号遷移グラフを**図 5** に示す．If-else 文から生成されるブロックを，それに含まれる各文を排他的に実行するので OR_join ブロックと呼ぶ．

この部分信号遷移グラフの動作を図 5 を用いて簡潔に説明する．条件式が真のとき，すなわち $cond = 1$ のときは立ち上がりブロック $\mathsf{N_{Y1+}}$ の上のトランジションが発火し，ブロック $\mathsf{N_{Y1+}}$ が実行される．このときに左上のトランジションの出力プレース，つまり $cond = 1$ と付記したプレースと図中の中央のプレース p_t にトークンが配置される．立ち上がりブロックが終了した後に立ち下がりブロックが開始される．立ち下がりブロック $\mathsf{N_{X-}}$ の中で発火可能なトランジションは左上のトランジションのみなので，ブロック $\mathsf{N_{Y1-}}$ が実行される．$cond = 0$ の場合も同様の方法で文 Y_2 が実行される．

If-else 文に含まれる 2 つの文は排他的に実行されるが，If-else 文の直後にある信号遷移の合成においては双方の文が直前に起こりうる文であり，駆動信号遷移候補には双方を含めなくてはならない．そのため，並列実行文と同様に論理合成のコストを増加させる要因となりうる．

2.3 定　　義

並列実行文や If-else 文が論理合成における合成コスト増大の要因となるのは，それらが駆動信号遷移の候補を増やすからである．本節では，並列度と以降の説明で用いる表記の定義を行う．

まず，あるトランジション t の駆動信号遷移集合を以下のように定義する．

定義 1

$$\mathsf{trig\_trans}(l(t), G) = \{\, l(u) \mid \langle u, \lambda^*, t\rangle \in trans\_path(G),\ l(u) \neq \lambda\}$$

ただし，λ^* はダミートランジション 0 回以上の繰返しを表す．また，$\langle t_1, t_2, \cdots, t_n\rangle$ は $2 \leq k \leq n$, $t_{k-1} \bullet \cup \bullet t_k \neq \phi$ を満たすとき，トランジションパスであり，$trans\_path(G)$ は信号遷移グラフ G 中のすべてのトランジションパスを表すものとする．

trig_trans を用いて，ある出力信号線 x の駆動信号線集合を以下のように定義する．なお，本論文の論題である並列度とは，各信号線の駆動信号線集合の要素数のことを指す．

定義 2

$$\mathsf{trig\_signal}(x, G) = \{name(u) \mid u \in \bigcup_{\{t \mid name(l(t)) = x\}} \mathsf{trig\_trans}(l(t), G)\}$$

ただし，$name(l(t))$ はトランジション t が関連づけられている信号線名を返す関数であり，たとえば $l(t) = signal\_name+$ であるとき，$name(l(t)) = signal\_name$ となる．

例として，図 2 (a) に示す信号遷移グラフについて考えると，trig_trans，trig_signal は以下のようになる．

- $\mathsf{trig\_trans}(c+, G) = \{d-, e-\}$
- $\mathsf{trig\_trans}(c-, G) = \{e+\}$
- $\mathsf{trig\_signal}(c, G) = \{d, e\}$
- $\mathsf{trig\_trans}(d+, G) = \{c+\}$
- $\mathsf{trig\_trans}(d-, G) = \{a-, b-, c-\}$
- $\mathsf{trig\_signal}(d, G) = \{a, b, c\}$

また，あるブロック B が与えられたときに，そのブロック内で最後の遷移となりうるトランジションの集合を返す関数を以下のように定義する．

定義 3

$$\mathsf{last\_trans}(B) = trans(B) - \bigcup_{u \in trans(B)} \mathsf{trig\_trans}(u, B)$$

ただし，$trans(B)$ はブロック B に含まれる，すべてのダミーではないトランジションからなる集合を表す．

また，last_trans はブロック集合にも適用することができ，ブロック集合 U に対して，$\mathsf{last\_trans}(U) = \bigcup_{u \in U} \mathsf{last\_trans}(u)$ とする．

2.4 予 備 実 験

我々の合成手法において，信号線の並列度と，その合成時間とに関係があることが経験的に分かっている．それを示すための予備実験として，**図 6** に示す 20 個

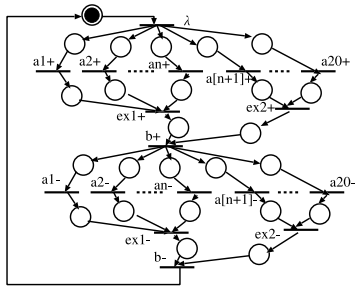

図 6 予備実験に用いた信号遷移グラフ
Fig. 6 STG used in pre-experiment.

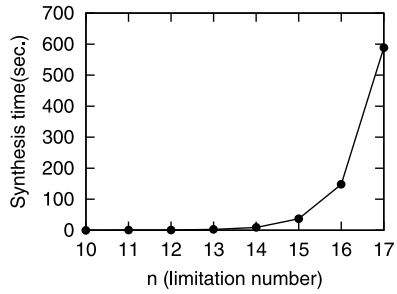

図 7 パラメータ n と合成時間の関係
Fig. 7 Relation between n and its synthesis time.

の信号遷移を並列に並べた信号遷移グラフの合成を行った．この実験ではパラメータ n を 10 から 17 の範囲で変化させて，つまり，信号線 $ex1$ の並列度を変化させ，かつ意図的に信号線 $ex1$ の合成時間が支配的になるようにして論理合成を行った．

パラメータ n と信号遷移グラフ全体の合成時間との関係を**図 7** に示す．実験結果から，図中の $ex1$ の駆動信号線数と，信号遷移グラフ全体の合成時間との間に正の相関があることが分かる．これは，駆動信号線の遷移がその信号の回路を合成するのに必要不可欠なので，縮退処理によって取り除かれることがなく，また，それらは並列に行われるために状態グラフの肥大化を引き起こすことが原因である．なお，状態数はパラメータ n に対し $O(2^n)$ で増加し，n が 18 以上の場合は，今回の実験環境のメモリ 4 GB では不足であったことを追記しておく．

3. 提 案 手 法

予備実験の結果より，限られたメモリ量，時間のもとで合成を行うためには，信号遷移グラフに含まれる各出力信号線の並列度を適切な値に抑えることが有効であることが分かる．本章では，提案する並列度を低下させるための信号遷移グラフ生成方法について述べる．

```
1 :solve_high_concurrency(G, limit) {
2 :    loop{
3 :      find x ∈ Out with
           |trig_signal(x, G)| ≥ limit;
4 :      if(x is not found) return G;
5 :      find B ∈ ⋃_{t|name(t)=x} src_AND(t, G)
           with largest |last_trans(B)|;
6 :      if (|last_trans(B)| > 1)
7 :        G = insert_signal_AND(B, G, limit);
8 :      else
9 :        break;
10:    }
11:    loop{
12:      find x ∈ Out with
           |trig_signal(x, G)| ≥ limit;
13:      if(x is not found) return G;
14:      find B ∈ ⋃_{t|name(t)=x} src_OR(t, G)
           with largest |last_trans(B)|;
15:      G = insert_signal_OR(B, G, limit);
16:    }
17:}
```

図 8 並列度解決アルゴリズムトップレベル
Fig. 8 Top-level algorithm of the proposed method.

提案手法の特徴は以下に示すとおりである．

- 遅延時間情報を用いた，回路の動作時間への影響を抑えた信号線挿入
- 再帰的アルゴリズムによる，複雑な並列構造への対応

3.1 並列度解決アルゴリズム

提案手法を**図 8** に示す．提案手法は引数として信号遷移グラフ G と非負の整数 $limit$ をとる．ただし，すべての信号遷移グラフに適用するためには，$limit$ の値を AND_join ブロック，OR_join ブロック以外の文の組合せによるブロックの最大の $|\mathsf{last\_trans}|$ の 2 倍以上にする必要がある．なぜなら，提案アルゴリズムでは，そのようなブロックの内部の並列度を低下させることができないからである．アルゴリズムの停止条件はすべての出力信号線 $a \in Out$ に対して，$|\mathsf{trig\_signal}(a, G')| \leq limit$ が成立することである．G' はアルゴリズムによって変更された信号遷移グラフであり，初期状態では $G' = G$ である．

まず信号線集合の中から，$|\mathsf{trig\_signal}(x)|$ が $limit$ を超えるような信号線 x を 1 つ任意で選択する（Line 3）．そのような信号線が存在しない場合は，信号線挿入による解決が必要のない信号遷移グラフであると判断され，アルゴリズムは終了する（Line 4）．そうでない場合は，信号線 x の立ち上がり遷移 $x+$ と立ち下がり遷移 $x-$ の直前の AND_join ブロックからなる集合 $\mathsf{src}_{AND}(t, G)$ の中から，$|\mathsf{last\_trans}(B)|$ を最大

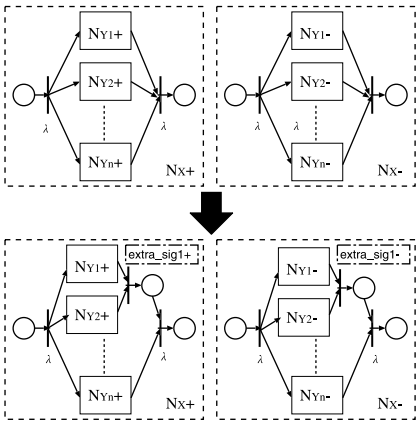

図 9 *AND_join* ブロックへの信号遷移の挿入
Fig. 9 Insertion of extra-transition in *AND_join* block.

にするブロック B を求める（Line 5）．

そのブロック B において，$|\mathsf{last\_trans}(B)| = 1$ である場合には，*AND_join* ブロックへの新たな信号遷移の追加が，並列度の減少をもたらさない．この場合はループ処理を脱する（Line 8–9）．そうでない場合，すなわち $|\mathsf{last\_trans}(B)| > 1$ である場合は，**図 10** に示す信号遷移挿入アルゴリズムによって並列度を減少させることができる（Line 6–7）．

1 つ目のループを 9 行目の *break* で抜けたにもかかわらず，依然として |trig_signal| の値が *limit* よりも大きい信号線が存在する場合は，*OR_join* ブロックが原因であることが考えられるので，それらを *AND_join* に対する解決法と同様の手法を適用することで，すべての出力線の |trig_signal| の値を *limit* 以下にすることが可能となる（Line 11–16）．

3.2 信号遷移挿入アルゴリズム

信号遷移挿入の例を**図 9** に示す．これは，*AND_join* ブロックにおいて，ブロック集合 $\mathsf{N_{Y1}}$，$\mathsf{N_{Y2}}$ の後に信号遷移 *extra_sig*1 を追加する例である．信号遷移 *extra_sig*1 の追加効果は，この *AND_join* ブロックに後続する信号遷移にとって，|trig_trans| の減少，つまり図 9 中の最外サブブロック $\mathsf{N_{Y1+}}$，$\mathsf{N_{Y1-}}$，$\mathsf{N_{Y2+}}$，$\mathsf{N_{Y2-}}$ の最終遷移が *extra_sig*1 1 つに置き換わることで現れる．なお，サブブロックとは，あるブロックに含まれるすべてのブロック群のことであり，最外サブブロックとはその中でも一番上の階層にあるブロック群を指す．

直前に起こりうる信号遷移数の減少のみを目的とする場合，必ずしも図 9 に示すように，ある文の立ち上がりブロックと立ち下がりブロックの双方に遷移を対に挿入する必要はない．信号遷移対を用いる理由は，

```
1 :insert_signal_AND(B, G, limit) {
2 :   U = outer_sub(B);
3 :   S = ∅;
4 :   Y = set of d ∈ U with largest delay(d);
5 :   while (U ≠ ∅) {
6 :     find b ∈ U with smallest delay(b);
7 :     if (|last_trans({b, b*})| > limit) break;
8 :     if (|last_trans(S ∪ S* ∪ {b, b*})| ≤ limit)
          S = S ∪ {b};
9 :     U = U − {b};
10:   }
11:   if (U = ∅) {
      \\ U が空でループを抜けた場合
12:     if (|last_trans(S − Y)| > 1)
          S = S − Y;
13:     G' = insert(S, B, G);
14:   }
15:   else {
      \\Line7 の break でループを抜けた場合
16:     find c ∈ outer_sub_AND(b) with
          largest |last_trans(c)|;
17:     if (c is found)
          G' = insert_signal_AND(c, G, limit);
18:     else{
19:       find c ∈ outer_sub_OR(b) with
            largest |last_trans(c)|;
20:       if(c is found)
            G' = insert_signal_OR(c, G, limit);
21:       else ABORT;
22:     }
23:   }
24:   return G';
25:}
```

図 10 *AND_join* ブロックへの信号遷移挿入アルゴリズム
Fig. 10 Signal insertion algorithm for *AND_join* block.

挿入する信号遷移そのものの一貫性を保持するためである．ここでの一貫性とは，すべての信号遷移の立ち下がりと立ち下がりは交互に現れなくてはならない，という論理合成のために必要な性質である．

図 10 に *AND_join* ブロックへの信号遷移挿入アルゴリズムを示す．信号遷移挿入アルゴリズムは，引数として信号遷移挿入の対象となるブロック B，その時点での信号遷移グラフ G，非負の整数 *limit* をとる．

アルゴリズムの最初でブロック B の最外サブブロック群を U に格納し，その中で最も遅延時間が大きいと見積もられた最外サブブロック群を Y に保存しておく（Line 2–4）．この遅延時間情報は，使用される演算器の遅延時間情報と，図 10 のアルゴリズムによってすでに追加された信号線の情報，ブロックの構成からあらかじめ計算されており，アルゴリズムが呼び出

されるたびに更新されるものとする．

そして，集合 U が空でない間，以下の動作を行う．まず，ブロック集合 U の中で最も遅延時間が小さいブロック b を見つける（Line 6）．そのブロック単体において $|\mathsf{last\_trans}(\{b, b^*\})|$ が $limit$ を超えている場合は，現在のブロック B における信号遷移挿入では解決できない．なぜなら，新たに追加した信号線の $|\mathsf{trig\_signal}|$ が $limit$ を超えてしまうからである．たとえば，図 9 で，$|\mathsf{last\_trans}(\{\mathsf{N_{Y1+}}, \mathsf{N_{Y1-}}\})|$ が $limit$ を超えている場合，信号線 $extra\_sig1$ の追加によって該当 AND_join ブロックの $|\mathsf{last\_trans}|$ を低下させることはできるが，新たに $|\mathsf{trig\_signal}|$ が $limit$ を超える出力信号線 $extra\_sig1$ が作成され，信号遷移グラフを制御回路合成が可能な程度に並列度を低下させるという目的は実現できない．この場合は，ループ処理を脱し，ブロック b に含まれる最外 AND_join ブロックの中で $|\mathsf{last\_trans}|$ の要素数が最も多いものを探し，そのブロックに対して信号遷移挿入アルゴリズムを再帰的に実行する（Line 17）．またブロック b の中に AND_join ブロックが見つからない場合は，対象となる信号遷移グラフ b の中の OR_join ブロックが原因であることが考えられるので，そちらを解決するためのアルゴリズムを呼び出す（Line 20）．それも見つからない場合には，指定した $limit$ の値が小さすぎて解決できない場合なので，アルゴリズムを異常終了する（Line 21）．

$|\mathsf{last\_trans}(\{b, b^*\})|$ が $limit$ 以下である場合は，ブロック集合 S とその対ブロック集合 S^* と $\{b, b^*\}$ の和集合における $|\mathsf{last\_trans}|$ が $limit$ 以下であるか否かを判断し，条件が成立する場合はブロック集合 S にブロック b を追加する．これは，新たに追加した信号線において $|\mathsf{trig\_signal}|$ の値が $limit$ 以下になることを保証する（Line 8）．

以上の動作を，ブロック集合 U が空になるまで繰り返す．その後に信号線を追加するのだが，このままではブロック集合 S の中に最も遅延時間の大きいと判断されたブロック（ブロック集合 Y に含まれるブロック）が含まれる可能性がある．そのような信号遷移の追加は合成された回路の性能の低下につながるため，その信号遷移追加でしか並列度合を減少させることができない場合（$|\mathsf{last\_trans}(S - Y)| \leq 1$，つまり S に挿入しても並列度が減少しない場合と $S = Y$ の場合）を除いて，ブロック集合 Y の各要素をブロック集合 S から除外する（Line 12）．その後に，ブロック集合 S と S^* の直後に図 9 に示すように新たな信号遷移を追加し，これを新たなブロックとする．(Line 13)．

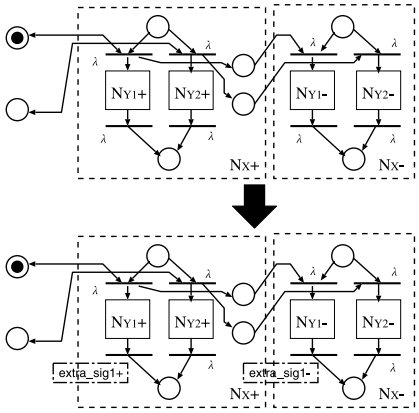

図 11 OR_join ブロックへの信号遷移の挿入
Fig. 11 Insertion of extra-transition in OR_join block.

つまり，図 9 において $\mathsf{N_{Y1+}}$，$\mathsf{N_{Y2+}}$，$extra\_sig1+$ からなる箇所，および $\mathsf{N_{Y1-}}$，$\mathsf{N_{Y2-}}$，$extra\_sig1-$ からなる箇所が，以降の並列度低下処理において単一のブロックとして扱われる．

以上に示すように，AND_join ブロックへの信号遷移挿入はブロックの遅延時間情報をもとに，できるだけ回路性能に影響を与えないような追加処理が行われる．一方で，OR_join ブロックへの信号遷移の追加は**図 11** に示すように行われる．アルゴリズムは図 10 に示すものから遅延時間を考慮している箇所を取り除いたものである．ここでは，ブロック $\mathsf{N_{Y1+}}$，$\mathsf{N_{Y1-}}$ の直後に信号遷移が追加，つまりダミートランジションとの置換が行われた例を示す．If-else 文の意味から，それに含まれるどちらのサブブロックが実行されるかは，環境に依存し，回路合成を行う時点では見積もることができない．つまり，OR_join ブロックへの信号遷移の挿入は性能に悪影響を与える可能性を取り除くことはできない．図 8 に示すアルゴリズムでは，可能な限り AND_join ブロックを優先的に解決するようになっている．

図 8 に示すアルゴリズムの計算コストを，図 10 に示すアルゴリズムが適用される回数で評価すると，記述に含まれる文の総数 M に対して，おおむね $O(M^2)$ と考えられる．以下に導出手順の概略を示す．

まず $l = limit$ とし，すべてのブロックの $|\mathsf{last\_trans}|$ の最大値を n として，$n \leq l^a$ となるような最小の自然数 a を考える．あるブロックを解決するために必要な信号遷移の数，いい換えれば図 10 のアルゴリズムが呼ばれる回数は，追加する信号遷移で l 分木を作るとすると最高で $1 + l + \cdots + l^{a-1}$ 回である．また，そのブロックの最外サブブロックに再帰的な解決を試みる回数は，1 つのブロックに 1 つの信号遷移

が対応していたとしてもブロック数は n 以上にはならないから，多くて l^a 回となるので，このブロックで図 10 のアルゴリズムが呼ばれる回数は，たかだか $1+l+\cdots+l^{a-1}+l^a=\frac{l^{a+1}-1}{l-1}$ となる．

文の総数は M なので，アルゴリズムの呼ばれる回数は多くて $\frac{l^{a+1}-1}{l-1}M$ となる．ここで，$\frac{l^{a+1}-1}{l-1}$ は $O(l^a)$ であり，a は $n \leq l^a$ を満たす最小の自然数であるから $n \leq l^a < nl$ なので，l が定数であることを考えると l^a は $O(n)$ である．また，他のブロックを含まないブロックによる最大の $|\mathsf{last\_trans}|$ を m とすると，$n \leq mM$ なので，n は $O(M)$ であると評価することができる．以上より，$\frac{l^{a+1}-1}{l-1}$ は $O(M)$ であると評価できる．よって，全体では $O(M^2)$ と評価することができる．

また，図 10 に示すアルゴリズムの計算コストは，文の数を M としたとき $O(M)$ であると評価できるので，提案手法全体の計算コストは $O(M^3)$ と評価することができる．

4. 評価実験

4.1 提案手法の評価

我々は提案手法を C 言語によって実装し，開発している高位合成プロセスの中に組み込んだ．

提案手法を適用したことによる，合成プロセスの回路合成時間と得られた回路動作時間への影響を確かめるために，3 種類の仕様記述に対してパラメータ *limit* の値を変化させて生成した信号遷移グラフから論理合成を行い，その回路の動作速度を ModelSIM によるゲートレベルシミュレーションにより評価した．なお，シミュレーションで用いたライブラリは 0.25 μm ルールのものである．

1 つ目の例では，すべての並列動作箇所に意図的に最長遅延時間を有する演算を 1 つずつを配置した．これは，新たな信号遷移を追加したとしてもクリティカルパスに挿入されない限り，回路の動作速度に影響を与えないことを確認することが目的である．

残りの 2 つの例は，それぞれ大きさの異なる有限インパルス応答システム（FIR）である．それぞれ並列度を高めるために**表 1** に示す使用可能な演算器数を想定している．演算遅延時間は加算器が 3 ns，乗算器が 5 ns を仮定した．つまり，乗算演算が各並列部分においてクリティカルパスとなる．

表 1 FIR のパラメータ
Table 1 Parameters of FIRs.

仕様名	次数	段数	加算器数	乗算器数
5x4FIR	5	4	3	3
7x6FIR	7	6	4	4

以上の例について，引数 *limit* を 8 から 17 の範囲で変化させて信号遷移グラフを生成し，論理合成を行った．引数 *limit* と論理合成に要する時間，回路の動作時間の関係を**図 12**，**図 13**，**図 14**，**図 15**，**図 16**，**図 17** に示す．

1 つ目の例（図 12，13）については，信号遷移を挿入することにより大幅な合成時間の短縮が実現されている．その一方で，回路の動作時間がほぼ影響を受けていないことが分かる．これは信号遷移グラフの段階で，クリティカルパスに追加信号遷移が挿入されていないためである．すなわち，少数のクリティカルパスが各並列部分に存在するような場合には，トレードオ

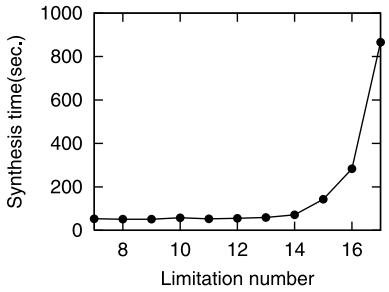

図 12 単純な例の合成時間
Fig. 12 Synthesis time of the simple example.

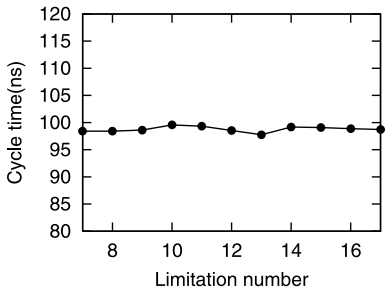

図 13 単純な例の回路動作時間
Fig. 13 Cycle time of the simple example.

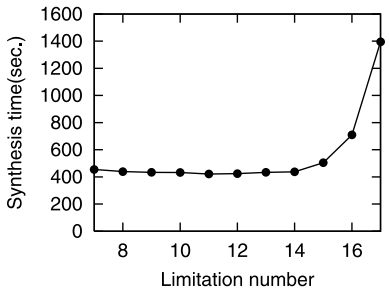

図 14 5x4FIR の合成時間
Fig. 14 Synthesis time of 5x4 FIR.

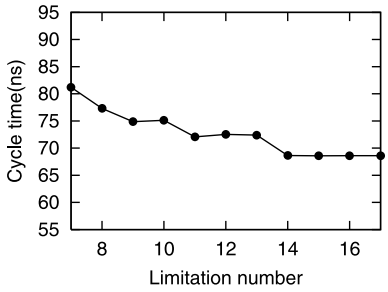

図 15 5x4FIR の動作時間
Fig. 15 Cycle time of 5x4 FIR.

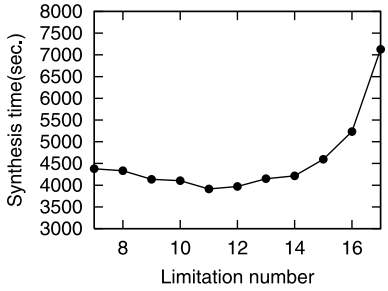

図 16 7x6FIR の合成時間
Fig. 16 Synthesis time of 7x6 FIR.

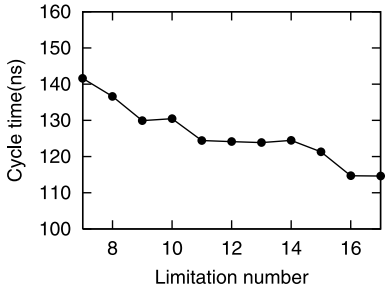

図 17 7x6FIR の動作時間
Fig. 17 Cycle time of 7x6 FIR.

フなしに合成時間の減少をもたらすという提案手法の有用性を，この実験結果は示している．

有限インパルス応答システムの例（図 14–17）に関しても，引数 *limit* を引き下げることによって合成時間を大幅に短縮することが実現されているが，同時に回路の動作速度に多少のオーバヘッドが生じた．これは，乗算演算が各並列部分に複数存在するために，クリティカルパスに追加信号遷移の挿入が行われたことが原因である．しかも，指定した *limit* が小さい場合は信号遷移の挿入が多段で行われるために，性能の低下が顕著に現れたと考えられる．よって，多数のクリティカルパスが各並列部分に存在する場合には，適切な引数 *limit* を選択することが，回路性能低下の回避

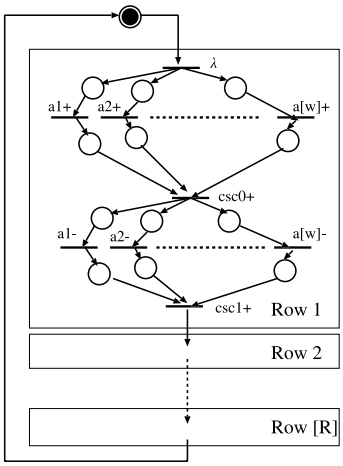

図 18 比較実験に用いた信号遷移グラフ
Fig. 18 STG used in the comparative experiment.

にとって重要になる．

なお，今回の実験環境（4 GB メモリ）での以上の 3 つの例の合成実験は，追加信号遷移なしではメモリ不足により合成不可能であったことを追記しておく．

4.2 他の論理合成法との比較

本節では，提案手法による信号遷移追加処理がなされた信号遷移グラフを，我々が開発している縮退信号遷移グラフからの合成手法（nutas）と，論理合成法の 1 つである petrify[7] とで合成し，提案手法が我々の手法に特に有用であることを示す．

実験で用いる仕様は，**図 18** に示す信号遷移グラフで，以下の 2 つのパラメータにより定義される．

- Row – 信号遷移グラフの段数（図中の R）
- Width – 信号遷移グラフの幅（図中の w）

パラメータの変化と提案手法を適用した後の両手法による合成時間との関係を**表 2** に示す．

実験結果より，nutas に対する本論文での提案手法の効果は，適切な *limit* の指定による合成時間の短縮という形で表れていることが分かる．

一方で提案手法を適用した信号遷移グラフを petrify を用いて合成したところ，nutas とは合成時間の推移の傾向が異なり，$limit = \frac{w}{2}$ の近傍が最も合成時間がかかり，*limit* が大きくなるにつれて合成時間がおおむね小さくなり，信号線を追加しない場合の合成時間がほとんどの例に関して最小であった．petrify では論理関数の計算時に BDD を使っているために，どのような場合に式が単純になるのかの予測が難しいが，少なくとも提案する適切な並列度での合成が有効に働くとはいえない．

合成時間そのものに関する比較では，nutas では，

表 2 我々の合成手法と petrify の回路合成時間に提案手法が与える効果（単位：sec.）
Table 2 Effect of proposed method to our synthesis method and petrify in synthesis time (sec.).

		Row = 1		Row = 2	
w	*limit*	nutas	petrify	nutas	petrify
14	7	0.25	3.61	0.55	94.47
	9	0.26	3.29	0.61	142.99
	11	0.54	4.10	1.19	31.30
	13	3.69	2.43	6.92	38.03
	14	10.13	2.16	19.81	37.70
15	7	0.25	4.23	0.62	159.32
	9	0.28	3.64	0.66	96.84
	11	0.54	5.25	1.24	47.90
	13	3.68	3.48	7.00	40.96
	15	37.75	2.18	78.21	39.27
16	9	0.29	4.82	0.73	142.41
	11	0.56	4.66	1.32	123.93
	13	3.72	4.07	7.06	70.93
	15	37.86	3.74	78.57	49.34
	16	156.53	3.32	337.71	50.98
17	9	0.32	6.20	0.80	372.79
	11	0.58	5.90	1.41	171.89
	13	3.73	5.41	7.12	198.68
	15	38.44	5.62	77.86	185.09
	17	616.80	4.05	1,241.81	65.52

段数 *Row* と合成時間との間に正比例の関係があることが考えられるのに対し，petrify では段数，いい換えれば記述の規模に対する合成時間の増加速度が著しいであろうことが実験結果より予測できる．まとめると，nutas では適切な *limit* を指定することによって，性能への多少のオーバヘッドと引換えに petrify の数十分の 1 の時間での合成が可能である．$Row = 3$ の信号遷移グラフからの合成は petrify の合成が正常終了しなかった．

なお，この実験においてそれぞれの仕様例から得られた論理関数は，nutas と petrify で等価であった．これは，この実験において両手法によって合成された回路の動作時間が等しいことを表す．

5. 結　　論

本論文では，論理合成の際の時間，メモリ双方のコストの低減を実現するための信号遷移グラフへの信号遷移の挿入手法を示した．また，いくつかの並列度の高い仕様記述の合成実験を行い，提案手法が動作時間の微細なオーバヘッドをもたらす可能性があるが，合成時間の大幅な短縮を確実にもたらすことが確認できた．さらに，既存の手法との比較により，提案手法が我々が開発している論理合成手法に特に適した手法であることを示した．

今後の展望としては，さらに実用的な並列度の高い仕様例に対する合成実験を行うことがあげられる．

参 考 文 献

1) Burns, S.M., and Martin, A.J.: Syntax-directed translation of concurrent programs into self-timed circuits, Allen, J. and Leighton, F. (Eds), *Advanced Research in VLSI*, pp.35–50. MIT Press (1988).
2) van Berkel, K., Kessels, J., Roncken, M., Saeijs, R. and Schalij, F.: The VLSI-programming language Tangram and its translation into handshake circuits, *Proc. European Conference on Design Automation* (*EDAC*), pp.384–389 (1991).
3) Kim, E., Lee, J.-G. and Lee, D.-I.: Automatic process-oriented control circuit generation for asynchronous high-level synthesis, *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp.104–113, IEEE Computer Society Press (Apr.2000).
4) Kessels, J. and Peeters, A.: The Tangram framework: Asynchronous circuits for low power, *Proc.Asia and South Pacific Design Automation Conference*, pp.255–260 (Feb. 2001).
5) Edwards, D. and Bardsley, A.: Balsa: An asynchronous hardware synthesis language, *The Computer Journal*, Vol.45, No.1, pp.12–18 (2002).
6) Bystrov, A. and Yakovlev, A.: Asynchronous circuit synthesis by direct mapping: Interfacing to environment, *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp.127–136 (Apr. 2002).
7) Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L. and Yakovlev, A.: Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers, *IEICE Trans. Information and Systems*, Vol.E80-D, No.3, pp.315–325 (Mar. 1997).
8) Beerel, P.A., Myers, C.J. and Meng, T.H.-Y.: Covering conditions and algorithms for the synthesis of speed-independent circuits, *IEEE Trans. Computer-Aided Design*, Vol.17, No.3, pp.205–219 (Mar. 1998).
9) Fuhrer, R.M., Nowick, S.M., Theobald, M., Jha, N.K., Lin, B. and Plana, L.: 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).
10) Carmona, J. and Cortadella, J.: ILP models for the synthesis of asynchronous control circuits, *Proc. IEEE/ACM International Conference on Computer Aided Design*, pp.818–825

(2003).
11) Yoneda, T., Onda, H. and Myers, C.: Synthesis of speed independent circuits based on decomposition, *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp.135–145. IEEE Computer Society Press (Apr. 2004).
12) Khomenko, V., Koutny, M. and Yakovlev, A.: Logic synthesis for asynchronous circuits based on petri net unfoldings and incremental sat, *Proc. ACSD* (2004).
13) Yoneda, T., Matsumoto, A., Kato, M. and Myers, C.: High level synthesis of timed asynchronous circuits, *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp.178–189. IEEE Computer Society Press (2005).
14) Edwards, D. and Bardsley, A. Balsa: An asynchronous hardware synthesis language, *The Computer Journal*, Vol.45, No.1, pp.12–18 (2002).

(平成 17 年 10 月 21 日受付)
(平成 18 年 4 月 4 日採録)

松本　敦

昭和 54 年生．平成 16 年東京工業大学大学院情報理工学研究科計算工学専攻修士課程修了．現在，東京工業大学大学院博士後期課程在学中．非同期式回路の高位合成に関する研究に従事．

米田　友洋（正会員）

昭和 32 年生．昭和 60 年東京工業大学大学院理工学研究科情報工学専攻博士課程修了．同大学助手，講師を経て平成 3 年より同大学助教授．平成 2〜3 年カーネギーメロン大学客員研究員．平成 14 年より国立情報学研究所教授．非同期式回路設計，形式的検証に関する研究に従事．電子情報通信学会，IEEE 各会員．