階層型CDFGによる非同期コントローラの合成

奥山祐市[†,☆] セッタセリークン ナッタ[††] 齋藤寛[††]
南谷崇[††] 黒田研一[†]

非同期回路は同期回路のクロック信号に関する問題を解決する可能性を持っている．しかし現在の非同期CADツールは，大規模な仕様を合成する際に，分割に関してその粒度を制御することができないため，生成された回路が不適切な場合が多い．本論文では自由度の高い非同期回路の仕様分割を行うために，階層型Control/Data Flow Graph（CDFG）を提案し，合成可能なSignal Transition Graph（STG）が生成されることを示す．この構造では，粒度の小さな非同期回路仕様記述を木構造で表現し，これらのノードの分割，併合を自由に行うことができる．さらに，この構造に対して，大規模なコントローラを合成する際，下位の合成ツールで生成される回路が最適になるような分割手法を提案する．また，提案された構造と分割手法を2つのマイクロプロセッサに適用し，自由な分割が可能であり，かつ効率的な非同期回路を生成することができることを確認した．

Synthesis of Asynchronous Control Circuits Based on Hierarchical CDFG

Yuichi Okuyama,[†,☆] Nattha Sretasereekul,[††] Hiroshi Saito,[††]
Takashi Nanya[††] and Kenichi Kuroda[†]

Asynchronous circuits have the potential to solve the problems related to clock signals of synchronous circuits. However, current CAD tools for large-scale asynchronous circuits partition specification irrelevantly, because these tools cannot control the granularity of circuit decomposition. In this paper, we suggest a hierarchical Control/Data Flow Graph (CDFG) for flexible partition of asynchronous circuits. We also show that a Signal Transition Graph (STG) for circuit synthesis can be generated from this structure. This structure represents asynchronous circuits using fine grain nodes. The nodes have the flexibility to be partitioned or to be merged into other nodes. In addition, we show an algorithm for a hierarchical CDFG to generate suitable STG for low-level CAD tools. We have confirmed that this algorithm can partition asynchronous circuits with flexibility and generate more compact circuits.

1. はじめに

現在のVLSI設計における重要な問題の1つに，クロック信号の分配の問題がある．これは，回路の消費電力の増大と信頼性低下の原因となっている．

グローバルクロックを使用した回路はそのすべてにクロックを供給する必要がある．クロックはCMOS回路の電力消費の大部分を占めている．近年の集積回路では省電力も重要な性能要因となっており，この問題に対する解法を提供することは重要である．

同期式の回路は，クロックによってすべての回路が同期をとることを前提として設計される．しかし，近年のゲート遅延と配線遅延の相対的な変化によって配線遅延が支配的になり，すべてのレジスタ遷移を同時に行うことが困難になっている．この問題はクロックスキューと呼ばれる．

この解法の1つとして，非同期回路の採用がある．非同期回路はグローバルクロックの代わりに局所的な同期を行うハンドシェーク信号を用いる．非同期回路は同期回路に比べ，無用なレジスタ遷移を抑えることができるため，回路全体の消費電力を抑えることができ，かつクロックスキューの問題が生じない．

このような非同期の利点にもかかわらず，現在の非同期CADツールでは大規模なコントローラなどを合成することができないという問題がある．このため，大規模な回路は小規模な回路群に分割せざるをえない．

† 会津大学大学院
Graduate School of the University of Aizu
†† 東京大学先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo
☆ 現在，日本電信電話株式会社
Presently with NTT Corporation

非同期回路の仕様記述法として Control/Data Flow Graph（CDFG）[1] による高位レベルの記述方法と，Signal Transition Graph（STG）[2] による低位レベル記述があるが，高位レベルの記述を入力する非同期 CAD ツールは，仕様を低位レベル記述（STG）に変換するのみである．変換された低位レベル記述は，論理合成ツール Petrify を用いて合成され，ゲート遅延モデルを反映した非同期 Speed Independent（SI）回路が実現される[2]．

本論文では，分割後のコントローラが命令やデータパス回路に制限されない構造を持つ CDFG と，そのレベルでの分割手法を提案する．この構造と分割手法は，従来の研究よりも，分割の自由度に優れており，低位レベルの合成ツールによって回路の生成を行う際に，最適な低位記述を出力することができる．

仕様分割は非同期回路の CDFG 表現から，STG を導く過程で行われる．これらの分割は高位レベル分割と低位レベル分割の 2 つに大別される．高位レベル分割は，分割を CDFG 上で行い，それらの CDFG 群から規模的に合成可能な回路を得る．一方低位レベル分割は，与えられた CDFG に対応する 1 つの STG 上で分割が行われる．

CDFG 分割はさらに，ハードウェア指向の分割とプロセス指向の分割の 2 つに大別できる．ハードウェア指向の分割では，集中型のコントローラが小規模なデータパス回路に対応する制御ノード群に分割される[3),4)]．一方でプロセス指向の分割では，制御ノードは命令に対応している[5)]．これらの方法は体系的であり自動分割を可能にする．しかし，分割の自由度に欠けるため，CAD ツールで扱うことのできない大規模な回路が生成されたり，極端に小規模で制御のオーバヘッドが大きい回路が生成されたりすることがある．

一方，STG の分割は STG の構造（marked graph, state machine）によるものが知られている[6)] が，構造に対する依存性が強いため，実用的な分割を行うことができないことが多い．

本論文の手法では，階層化された CDFG を導入することによって，分割を行う際の基本ノードをできるだけ小さくし，またこれらの基本ノードを木構造で自由に組み合わせることによって，分割後の回路規模を細かく制御することを可能とする．たとえば Petrify [2)] では約 20 信号までの STG しか扱うことができないため，そのような制限を設けた STG を自動的に生成する．また，本手法により大規模 CDFG の階層的な分割が，現実的で最適な STG を生成することを示す．

以降では，まず 2 章で非同期回路および階層型 CDFG を導入し，これらの仕様を基に，3 章で STG の生成について，4 章で CDFG の分割について述べる．さらに，5 章で分割されたコントローラを接続するグルーロジックについて述べ，6 章で実験とその結果について述べる．最後に，7 章でまとめを述べる．

2. 非同期回路の仕様記述

本章では，合成対象となる非同期コントローラの仕様を述べ，本論文で導入される階層型 CDFG についてその定義を行う．

2.1 非同期コントローラ

非同期コントローラは主にデータパス回路に対して制御信号を生成する．コントローラの機能として，データパス回路の制御，分岐制御，他のコントローラとの同期がある．これらの制御のために，それぞれのデータパス回路やコントローラへは要求（req），応答（ack）の 2 つの信号が分配される．この 2 つの信号をハンドシェークペアと呼ぶ．

データパス回路はセレクタ，レジスタ，演算器から実現され，これらの組合せで実現する回路をデータパスと呼ぶ．データパス回路は一般的にあらかじめ用意された最適な回路をライブラリ化し，これを仕様にあわせてマッピングする．したがって，本論文では，データパス回路の合成は議論の対象外とする．

コントローラはデータパス回路に対してハンドシェークペアを介してレジスタのイネーブル信号，セレクタのセレクト信号を送信する．また，演算器に対して演算選択信号が生成される場合もある．コントローラはデータパス回路に対して与えられた仕様に矛盾のないように制御信号を発行する．非同期回路の場合，データパス回路の動作が必要なときにのみコントローラからリクエスト信号を受け取り，動作を行う．

分岐制御はデータフローをセレクタを使って制御し，データパスから送られる条件によるフローの制御を行う．

他のコントローラとの同期は，主に外部モジュールとのハンドシェークと，2 つ以上のコントローラから 1 つのデータパス部品への制御を行う際に必要となる．データパス部品は，同時に複数の制御信号を受け取ることができないため，コントローラどうしが同期をとり，制御信号を送信する必要がある．

2.2 階層型 CDFG

本節では，表現を一部制限した CDFG である階層型 CDFG を定義する．この CDFG は大きな特徴として，CDFG を階層構造で表現する．この階層構造を導入することによって，CDFG の分割が，定義さ

れる基本ノード間で自由になり，STG の処理系に最適な STG を出力することができるようになる．

なお本論文では，与えられる仕様は演算順序のスケジューリングが済んでいる CDFG と仮定する．

2.2.1 CDFG の階層化

本論文で提案する階層型 CDFG は，定義される基本ノードが階層的に構築できるように制限されている．CDFG における基本ノードと非同期回路は 1 対 1 に変換することができる[7] ため，CDFG によって記述された仕様は非同期回路を生成する能力を有するといえる．ただし，CDFG の基本ノードを階層型 CDFG の基本ノードに変換する際に，同じノード群を複製しなければ変換を行うことができない場合がある．このような場合は，実装時にコピーされたノードは生成せずに，同じコントローラに対して複数回の起動を行うことにより（グルーロジックを用い実現．5 章参照），最小限のオーバヘッドで実現することができる．

一般的な CDFG の基本ノードに対して分割を行うと，分割したノードが下位の論理合成ツールの合成条件を満たすことができないことがある．これは，CDFG は fork と join，あるいは選択ノードとその終端が必ずしも 1 対 1 に対応していないためである．

そこで，階層型 CDFG では，開始ノードと終端ノードの対応付けを保証した基本ノードを定義し，これらのノード群の操作によって，非同期回路を合成できるような体系を構築する．基本ノード群は，単独で非同期回路となり，かつ基本ノードが他の基本ノードと自由に結合できるため，粒度に関して自由度の高い非同期回路を生成することができる．

2.2.2 階層型 CDFG 基本ノード

階層型 CDFG の基本ノードは seq，par，choice，loop，main，synchronize，signal の 7 種類からなる（**図 1**）．これらのノードの定義では並列動作や条件分岐の開始ノードと終端ノードを含めて 1 つの基本ノードとしており，それらの子階層にノードを付加することによって，全体的な CDFG を得ることができる．

seq ノードは順序動作を表し，ノード内に現れる事象を順番に生起させることを示す．par ノードは並列動作を表し，ノード内の事象を同時に実行することを示す．このノード内の並列事象がすべて終了すると，このノードの実行が終了したと見なされる．choice ノードは選択を表し，外部からの n ビットの選択信号によって，2^n 種類の動作を選択する．loop ノードは繰返し処理を実現するため，child1 の実行が終わった後に条件判断を行い，繰返しが行われることが判断された場合に child2 の実行が行われる．

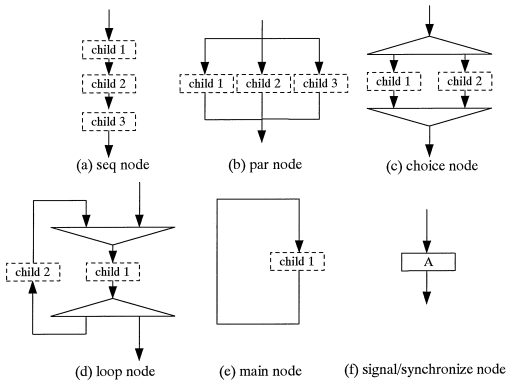

図 1 階層型 CDFG の基本ノード
Fig. 1 Primitive nodes for hierachical CDFG.

これらのノードの開始点として，main ノードが存在する．main ノードは受動型と能動型の 2 種類あり，受動型は他のコントローラから起動されることによって実行を開始し，能動型はシステムのリセットが完了した時点で自発的に実行を開始する．すべての合成可能なコントローラは main ノードをルートとして持つ．

signal ノードと synchronize ノードは定義された階層型 CDFG の外部との同期（通信）に用いられる．1 つの signal ノードには，ハンドシェークペアが 1 つ割り当てられ，1 つの synchronize ノードには，ハンドシェークペアのうちの 1 つの信号，すなわち要求信号か応答信号のどちらかが割り当てられる．ハンドシェークペアを複数のコントローラに分割すると，コントローラ間の制御が複雑化するため，synchronize ノードの合成時には，ハンドシェークペアに対応する信号を 1 つのコントローラに割り当てる．しかし，この割当ては，分割時の自由度を著しく下げるため，外部通信間で同期をとる場合にのみ，synchronize ノードによる同期を用い，データパスや，子階層コントローラへの通信には，signal ノードを用いる．

以上によって定義された CDFG は階層的な構造を持っているため，計算機内で木構造として表現することができる．さらに，定義されたノードはそれぞれの終端を signal ノードとすることによって，独立した形で非同期回路として合成することができる．

3. STG の生成

各基本ノードが STG の合成可能条件を満たすために必要なプロパティについて述べる．階層型 CDFG の基本ノードはここに述べられたプロパティを満たすことにより，分割後も非同期 SI 回路を得ることができるようになる．

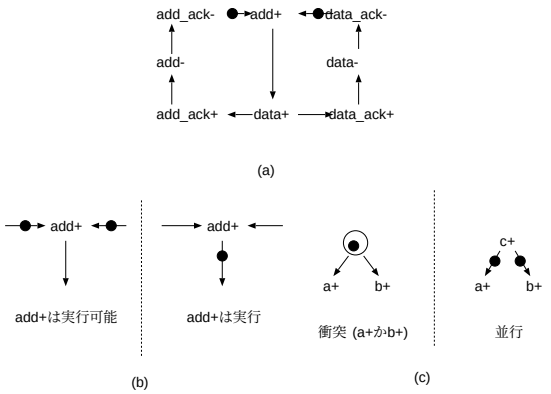

図 2 STG：(a) ある回路の STG，(b) 実行の過程，(c) 衝突と並行
Fig. 2 STGs: (a) an STG, (b) process of execution, (c) conflict and concurrent.

3.1 STG の定義

Signal Transition Graph（STG）は，回路上における信号の遷移を図によってモデル化したものであり，4 つの要素 $\Sigma = \langle P, T, F, M_0 \rangle$ からなる[6]．ここで，P はプレースの集合（STG 上のサークルに対応し，回路上の状態に反映）☆を，T は信号遷移の集合（STG 上のノードに対応）を，F は信号遷移間の依存関係（STG 上のアークに対応）を，そして，M_0 は回路の初期状態を表す（**図 2** (a)）．

すべての信号遷移は $+(0 \to 1)$ と $-(1 \to 0)$ でラベル付けされる．ある信号遷移は，もしそのすべての入力アークでトークンと呼ばれるマーキングを持つならば，実行可能（enabling）となる（図 2 (b 左)）．実行可能となった信号遷移は，実行され，トークンは出力アークへと移動する（図 2 (b 右)）．また，2 つ以上の信号遷移，たとえば $a+$ と $b+$ が両方実行可能で，どちらかの実行によって残りが実行不可能になる関係を，衝突（たとえば，ある状況における選択）とよび（図 2 (c 左)），それ以外の関係，つまり，片方の実行後ももう片方が実行可能である状態を並行と呼ぶ（図 2 (c 右)）．

STG のすべての信号は，入力，出力，内部信号のいずれかに分けられ，出力と内部信号が合成によって回路上に実現される．

3.2 CDFG ノードから合成可能な STG へのマッピング

CDFG の分割によって得られた各基本ノードから，合成可能な非同期 SI 回路を導くには，各基本ノードに対応する STG が以下のプロパティ，有界性（boundedness），可換性（commutativity），一貫性（consistency），持続性（persistency），完全性（complete state coding（CSC））を満たすことが必要である．STG の合成可能性は，STG や信号遷移による状態の到達可能性を解析された state graph（SG）上で行われるが[2]，ここでは，STG（SG）上での考慮を分割後の各 CDFG ノード上に置き換えて説明する．

有界性（Boundedness） 与えられた STG が有界であることは，与えられた STG がアークにトークンを 2 つ以上存在させない構造になっていることで満たされる．これは SG のどの状態においても，実行可能な信号遷移数に限界があるということである．

分割前の CDFG においては，信号遷移の実行を表すトークンの数が main ノードから導出されるため，1 つであった．トークンが 1 つとは，記載されたすべての動作（信号遷移）の実行回数はたかだか 1 度ということを表し，分割後もそれぞれのノードが持つことが可能なトークンの数は 1 つのみなので，それらから得られた STG（SG）も有界である．

可換性（Commutativity） SG が可換であるとは，ある状態において並行に実行可能な信号遷移がそれぞれ実行した後にたどられる状態は同じであるということである（**図 3** (a 左)）．非同期回路は，並行動作内のイベントの動作順序が非決定であるため，並行な信号は起こりうる遷移に対応した中間状態を SG 上に含む．

各基本ノードは，たとえその内部で並行な信号遷移群が存在したとしても，その後に続く別のノードが実行可能となる前に，すべてそのノード内で同一の終了状態に落ち着くので，そのノードから導かれた STG（SG）は可換であるということができる（図 3 (a 右)）．

一貫性（Consistency） STG において，信号に一貫性があるとは，ある信号 $a+$ の後に続く同じ信号の次の遷移は必ず $a-$ であるということである（図 3 (b 左)）．非同期回路では，ある信号における遷移をすべて明確に区別するため，STG 上においても信号の有効遷移を表す稼働相（たとえば $a+$）と，それらの信号のリセット動作を行う休止相（$a-$）が必要となる．

各基本ノードにおいて，対応する STG に変換する際に，ノード上で制御動作に関するすべての信号遷移は稼働相としてマッピングされ，その後それらの信号遷移のリセット状態を表すために休止相を行うという構造で実現すれは，その中で，ある信号 a が立ち上がり遷移（$a+$）であれば，必ず次の遷移は立ち下がり遷移（$a-$）ということになる．したがって，得られる STG は一貫性を満たす（図 3 (b 右)）．

☆ 通常，プレースの入力アークと出力アークが 1 つの場合，そのプレースは省略される．

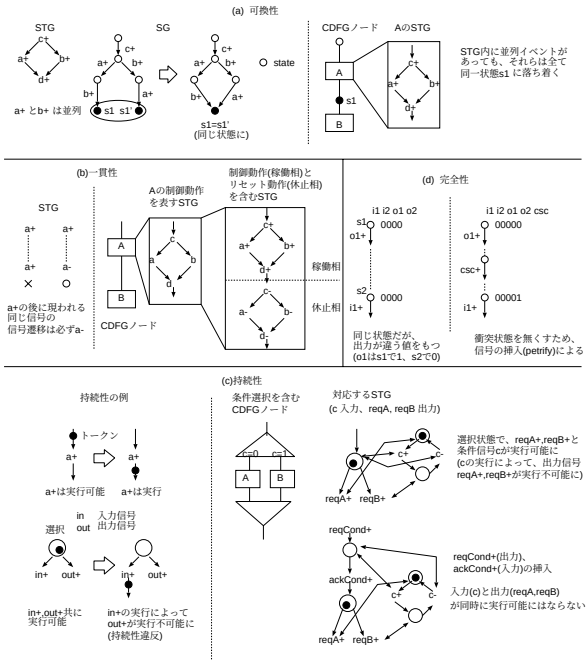

図 3 STG 合成のためのプロパティ
Fig. 3 Properties for STG synthesis.

持続性（Persistency） SG において，信号に持続性があるとは，ある状態 s において信号 a が実行可能となるなら，その信号は必ず実行されるということである．これは STG において考えると，図 3 (c 左) のように表される．

分割されたノード上で条件選択（choice，図 3 (c 右) の，c のように信号のレベルを表すサイクル構造）を除いては，出力信号は入力と同時に実行可能となることはないので（入力 → 出力（内部）の順．またはその逆），持続性が保たれる．条件選択においては，選択後の出力遷移と選択の際の条件を表す条件信号（入力）が同時に，実行可能となる恐れがあり，片方が実行してしまうと，もう片方を実行不可能にしてしまう．したがって，このような状況を避けるために，条件選択の際には条件信号を決定する特別のハンドシェーク信号 *reqCond*（条件信号遷移開始を表す出力信号），*ackCond*（条件信号が安定したことを示す入力信号）を挿入することによって持続性を保証する（つまり，条件選択後の出力信号と条件信号（入力）は同時に実行可能とならない，図 3 (c 右)）．

完全性（Complete State Coding（CSC）） SG において CSC であるとは，すべての状態に 0 と 1 を割り当てる際，割当てが同じ状態は，同じ出力セットを持つということである．仮にもし同じ状態で違う出力を持つ場合は（CSC 衝突，図 3 (d 左)），合成の際 Petrify が新しい信号を挿入することにより，そのような同じ状態を区別する．完全性は非同期 SI 回路を STG から合成する際，Petrify によって解決される（図 3 (d 右)）．

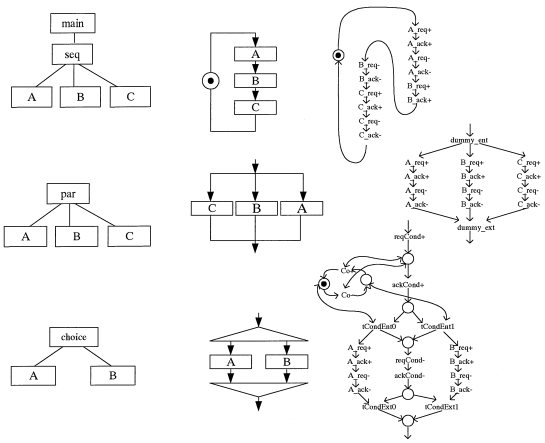

図 4 基本ノードとその STG
Fig. 4 CDFG primitive nodes and corresponding STGs.

4. 階層型 CDFG の分割

本論文で導入された，階層型 CDFG 構造から合成可能な STG を生成できることは，前章で確認された．ところが，既存の STG 合成ツールの規模に対する制約から，実際には，STG 合成が可能な条件として回

路規模の制限を付加しなければならない．本章では，合成系を Petrify とした場合の信号数の制約と CSC 信号の付加を考慮した分割手法を提案する．

STG は非同期 SI コントローラを生成するための強力な記述であるが，扱うことのできる信号数が少ない（20〜30 信号）という問題がある．

STG から合成を行う際に，合成の妨げになる最も大きな問題は，状態爆発問題である．したがって，通常合成可能性を判断する場合，STG に存在する入出力信号数を基準にすることが多い．我々もこの方法に従い，合成可能な STG の十分条件として，入出力信号の最大数を n として議論を進めていく☆．

出力される STG を最大信号数 n に制限するためには，与えられた CDFG を分割し，その分割に従ってコントローラ間のハンドシェーク信号を付加した結果，すべての分割されたコントローラで入出力信号数が最大信号数を下回っている必要がある．

端的な例として，単体の CDFG のノード群をそのまま STG として出力/合成し，それらをグルーロジックで接続することもできる．しかしこの方法では，回路に新たなハンドシェーク信号を付加することになり，回路規模が増大する．

一方で，seq などで表現される実行が順序付けされた STG は，入出力信号が子ノードの処理の開始と終了において必ず一致するため，同じ入出力信号値を持つ状態が複数存在することになり，回路を導く際に CSC 衝突を招く．この衝突は内部信号を付加することによって解決することができるが，これによって回路面積の増大を招く．したがって seq で表現された論理は積極的に分割し，CSC 衝突を回避することによって，分割数が増加してもコントローラ全体としての面積を減らすことができる可能性がある．

4.1 分割の基本

階層型 CDFG は各ノードにおいて任意に分割可能である．分割後はコントローラどうしを接続するためのハンドシェークを挿入する．基本的な分割には，ノード間の接続に関する分割と，ノード自体に関する分割の 2 種類がある，

ノード間の接続に関する分割は，階層型 CDFG の木構造に対して分割を行い，新たなサブツリーを求める．その際サブツリーの親ノードに対しては signal ノードを挿入し，サブツリーに対しては，挿入された signal ノードに対応する受動的な main ノードをルートノードとして付加することによって，分割前の仕様と同じ動作をする．

一方，seq，par，choice ノードは入力信号数が仕様によって変化するため，ノード自体が分割対象となる可能性がある．そのような場合は，それぞれのノードを信号数が制約に収まるまで階層的に分割することで対応することができる．

4.2 分割アルゴリズムの概要

本節では階層型 CDFG の分割アルゴリズムについて述べる．アルゴリズムの詳細は，**図 5**，**図 6** に示す．

まず，選択部分を含むノードとそれ以外のノードをそれぞれ制御ノードと制御を持たない Data Flow Graph（DFG）ノードと呼ぶ．選択動作後に現れる DFG ノード群は，生成された STG もそれ単体でほとんどの場合で合成できるため，それらで構成されるノード群を DFG として基本ノードで表す．また，CDFG と同等な機能を階層型 CDFG で表現すると，同じ機能を持つ DFG が他のノードにコピーされる場

```
program partitionCDFG(S : given CDFG; signals : # of maxmum signals);
var S', result : list of subtree;
begin
  /* synchronize ノードのペアをつなぐパス上に
     現れる基本ノードを分割不可に設定*/
  bindHandshakePair(S);
  /*制御ノードに関する分割*/
  partitionControl(S);
  /*選択ノード以外の DFG に関しての分割*/
  partitionDFG(S);
  /*基本ノードを探索し，併合*/
  while (S' ≠ empty) do
    mergeNodes({s|s ∈ S'});
end

 procedure bindHandshakePair(S : given CDFG);
var r : node; v : 分割不可能フラグが立っている擬似ノード;
begin
  foreach {p|p ∈ S, p は synchronize ノードかつ要求信号 } do
    foreach {q|q ∈ S, q は synchrnize ノードかつ p に対応する応答信号 } do begin
      r := p → q に対応するパス;
      foreach {s|s は r に含まれる基本ノード } do begin
        v := v + s;
        s に接続されているノードを v のノードとして再接続
      end
    end
end

procedure partitionControl(S : given CDFG);
begin
  foreach {c|c ∈ S, c は制御ノード (choice か loop)} do
    if root ノードから c までのパスに他の制御ノードが含まれる then
      createSubtree({c の subtree});
end

procedure partitionDFG(S : given CDFG);
begin
  foreach {D_i|D_i ∈ S, D_i は DFG ノード } do
    if D_i の親ノード ! = conditionanal ノード or D_i has copy flag then
      createSubtree({D_i});
end

function sortChildNode (s : subtree of CDFG, n : node) : list of node;
begin
  n の子ノード n_i に対して totalSignals(s + n_i) をキーとしたソートを行う.
  return result_list;
end

procedure createSubtree(s : subtree of CDFG);
begin
  foreach {r|p ∈ s, q ∉ s, r は p → q となるようなノード間接続 } do begin
    p → q の接続を，singal ノードで置き換える;
    v の親ノードの接続を signal ノードで置き換える;
  result := result + v;
end
```

図 5 階層型 CDFG 分割のためのアルゴリズム（1）
Fig. 5 An algorithm for hierarchical CDFG partition (1).

☆ 経験的にこの n を与える目安として 20 信号前後が良いと思われる．

```
procedure mergeNodes (s : subtree of CDFG)
var n : node; v:=φ , traverse : list of node;
begin
  traverse := root node of s; /* 探索リスト */
  while traverse ≠ empty do begin
    n = getFirst(traverse); remove(traverse, n);
    if totalNodes(v + n) > signals then
      if n 自体が分割可能 then
        if n が choice ノード then
          addFirst(traverse, partitionNode(n));
        else
          addLast(traverse, partitionNode(n));
      else
        break;
    else begin
      v := v + n;
      if n が choice ノード then
        addFirst(traverse, sortChildNode(n));
      else
        addLast(traverse, getChildList(n));
    end
  end
  create_subtree(v);
  S' := S' + v の子ノード
end

function partitionNode ( v : node ) : list of node;
var childList, choiceList : list of node;
      nodeList:=φ : set of list; x, next, c, c' : node;
begin
  childList := getChildList(v)
  x := {n_i|n_i ∈ childList, totalSignals(n_i) is maxmum};
  remove(childList, x);
  while(x ≠ empty) do begin
  next := φ
    while (totalSignals(x + next) ≤ signals) do begin
      remove(childList, next);
      x := x + next;
      next := {n_i|n_i ∈ childList, totalSignals(x + n_i) is minimum};
      if(next == empty) break;
    end
    nodeList := nodeList + x;
    x := {n_i|n_i ∈ childList, totalSignals(n_i) is maxmum};
    remove(childList, x);
  end
  foreach {y|y ∈ nodeList, (y is a list of nodes)}
    c' := y を子ノードに持つ choice ノードを生成;
    choiceList := choiceList + c';
  end
  c_top := choiceList を子ノードに持つ choice ノードを生成;
  return c_top + choiceList;
end
```

図 6 階層型 CDFG 分割のためのアルゴリズム（2）
Fig. 6 An algorithm for hierarchical CDFG partition (2).

合がある．このよう場合は，重複を避けるため，コピーされた DFG を 1 つのノードとして実現し，他のコントローラからの起動にはグルーロジックを用いる．これによって，回路の減少を見込むことができる．さらに seq，par ノードはあらかじめ 2 分岐になるように分割しておく．

次に，synchronize ノードは，要求信号と応答信号としてグラフの任意の位置に置くことができるが，これら信号を分割後に別のコントローラに配置することは，合成を複雑にする．したがって，関数 *bindHandshakePair* によって，あらかじめこの信号をグループ化しておき，分割候補から外しておく．**図 7** (a)，(b) がこの関数の適用前の CDFG，(c)，(d) が適用後の CDFG である．

さらに，選択動作と DFG に関する分割を行う．選択動作を含む choice，loop ノードは生成される STG が複雑になるため，単体で 1 つの STG にマップする（関数 *partitionControl*）．DFG は親に choice ノードを持たない限り分割される（関数 *partitionDFG*）．

最後に，これらの分割で与えられたノード群 S' に

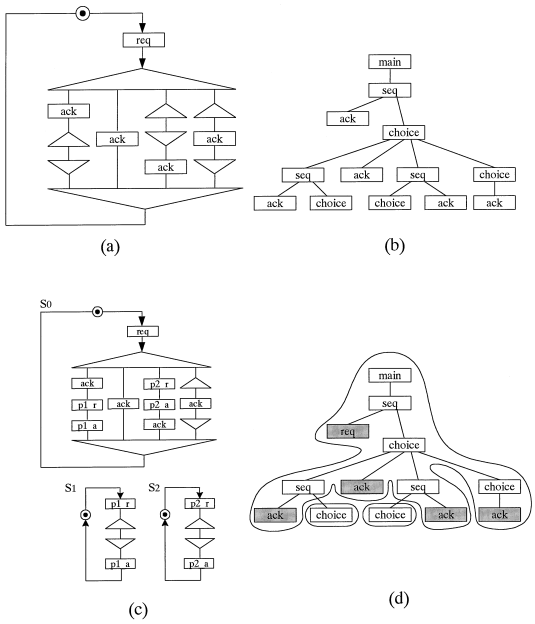

図 7 synchronize ノードに対する分割
(a)，(b)：分割前の CDFG および木構造，(c)，(d)：(a)，(b) を分割したもの
Fig. 7 CDFG partition for synchronize nodes. (a), (b): CDFG and tree structure of CDFG before partition, (c), (d): after partition for (a), (b).

対して，信号数制約 signals を超えない範囲で，S' 内のノードを併合することによって，合成可能なサブグラフを得る（関数 *mergeNodes*）．この際，choice ノード内に現れる基本ノードは複雑な CSC 衝突を引き起こす原因となるためこれらのノードは choice ノード直下の基本ノードを 1 レベルだけ含んだ状態で分割を行う．また，choice ノードに関しては，それ自体の分割を効率化するために，子ノードの信号数を考慮した分割が行われる（関数 *partitionNode*）．

5. コントローラ間のグルーロジック

分割の結果，1 つのデータパス部品が 2 つ以上のコントローラから要求信号を受け取る場合が出る．また，2 つ以上のコントローラがデータパス部品や下位のコントローラを共有すると，コントローラの面積効率が良くなることが多い．これを可能にするため多数のコントローラからの信号を 1 つにまとめ，データパスを制御する．この組合せ論理をグルーロジックと呼ぶ．この章では，グルーロジックについて述べる．

本論文の仮定では，与えられる CDFG はすべてスケジュール済みだと仮定している．そのため，分割されたそれぞれのコントローラは同時にデータパス部品に対してリクエストを発行することはない．したがっ

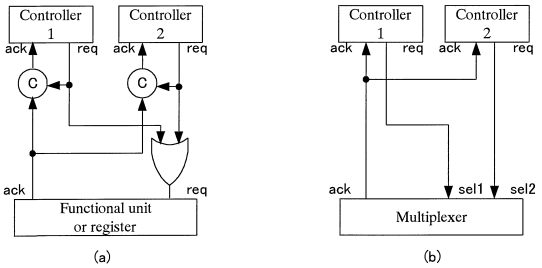

図 8 分割されたコントローラに対するグルーロジック
(a)：排他的な要求，(b)：並列な要求

Fig. 8 Example of glue logics for partitioned asynchronous control circuit. (a): exclusive request, (b): parallel request.

て，挿入される論理は機械的に判断することができる．

たとえば，**図 8** (a) では 2 つのコントローラからの要求に対して，1 つのコントローラ，またはデータパス部品がその要求を受け付ける場合の回路が示されている．この回路では，片方のコントローラから要求が起こった場合，応答を返すコントローラを判断するために，C 素子を用いている．

一方，図 8 (b) では，前述の回路とは異なり，2 つのコントローラに同時に応答を返している．このような状況は，データパス回路でセレクタを含む演算を行っている場合に発生する．

6. 実　　験

この章では本論文で導入したアルゴリズムを用いて，例題として小さな 2 つのプロセッサを非同期回路として合成する．いずれも仕様は CDFG で記述され，階層型 CDFG に変換されている．

実験では，与えられた CDFG から STG を導出し，Petrify によって論理合成する．この際，与えられた CDFG を分割することによって提案したアルゴリズムの評価を行う．実験は 2 つのプロセッサ A，B に関して行う．プロセッサ A は Instruction Fetch（IF）コントローラ，Execution（EXEC）コントローラの 2 つ，プロセッサ B は 1 つの階層型 CDFG によってコントロールされる．3 つの CDFG において以下のような評価をすることができる．

(1) IF コントローラは分割なしで合成できるが，これを提案アルゴリズムによって分割し，分割前のものと比較．
(2) プロセッサ B で分割される CDFG の粒度を変化させ，提案アルゴリズムと比較．

なお，階層型 CDFG は紙面の都合上，DFG ノードのマージを行った後のものを示す．DFG ノードは図中にノード名，そのノードに含まれる信号数の順に記

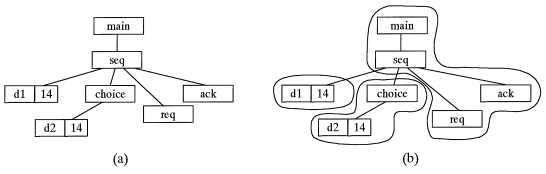

図 9 IF コントローラの分割前後の階層型 CDFG
(a)：分割前，(b)：分割後

Fig. 9 CDFG nodes for IF controller partition. (a): before partition, (b): after partition.

表 1 IF コントローラの合成結果
Table 1 Syntesis result of IF controller.

IF module (# of CDFGs)	partition (3)	non-partition (1)	ratio
CDFG area	150	253	1.68
glue area	26	—	—
total area	176	253	1.44
cycle time(ns)	18.28	18.96	1.03

述してある．また，末尾に c がついている DFG ノードは，階層型 CDFG にマッピングする際に同時に起動されない同等の機能がコピーされたことを示している．このようなノードは合成時に回路が生成されない．また，EXEC コントローラに関しても実験を行ったが紙面の都合上割愛する．

6.1 プロセッサ A の IF コントローラ

プロセッサ A は 2 つの CDFG（IF，EXEC）からなる，アキュムレータを 1 つ持ったアーキテクチャである．命令数は 16，データパスは論理演算器，加算器，シフタから構成されている．

図 9 (a) に IF module の階層型 CDFG を表す．図 9 (b) は分割アルゴリズムに沿って分割されたノード群を表す．それぞれに信号数制約 $n = 22$ として STG を導き，合成した後，NEC 製 CBC10 [8] に対してテクノロジマッピングを行った結果を**表 1** にまとめる．

この IF コントローラは，Petrify によって，分割することなく合成可能である．しかし，提案アルゴリズムによって分割を行うと，約 7 割（1/1.44 倍）の回路規模に抑えることができた．理由は大規模な STG では状態衝突が多くなり，CSC 信号などの内部信号挿入が繁雑になるためである．一方，分割を行うことによって，状態数が減少し CSC 衝突が抑えられ，回路規模が小さくなった．

6.2 プロセッサ B

プロセッサ B は 1 つの CDFG からなる，テンポラリレジスタを 2 つ持ったメモリベースの教育用プロセッサである．命令数は 4 であり，データパスは，マ

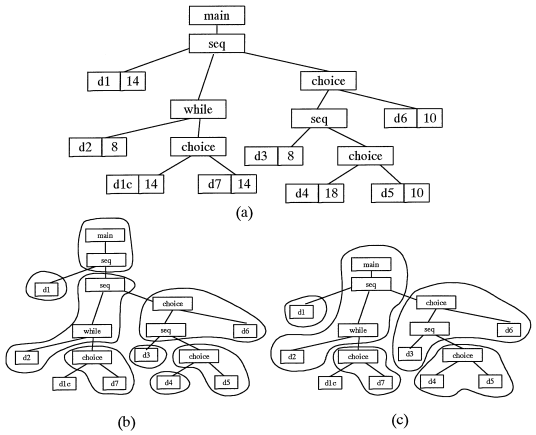

図 10 プロセッサ B の階層型 CDFG とその分割
(a)：分割前，(b)：提案された分割，(c)：粗粒度分割
Fig. 10 CDFG nodes of processor B before/after partition. (a): before partition, (b): suggested partition, (c): coarse grain partition.

表 2 プロセッサ B の合成結果（全回路数）
Table 2 Synthesis resuts of processor B (total area).

processor B	suggested	fine	coarse	ratio
# of CDFGs	8	13	6	(fine/coarse)
non-tm	4184	4336	4443	1.04/ 1.06
CBC10	1099	1130	—	1.03/ —
Petrify	5792	6048	—	1.04/ —

ルチプレクサ，インストラクションレジスタ，プログラムカウンタ，テンポラリレジスタ A，B，ALU，フラグレジスタから構成されている．プロセッサ B の階層型 CDFG を**図 10** (a) に示す．ここでは，信号数制約を $n = 22$ とし，3 つの実験を行って評価する．

(1) 提案アルゴリズムを用いた分割
(2) 出力ノード数が最大になる分割（細粒度分割）
(3) 出力ノード数が最小になる分割（粗粒度分割）

6.2.1 分割アルゴリズムの結果

図 10 (b)，(c) に提案手法と，粗粒度分割によって分割された階層型 CDFG を示す．細粒度分割は単体では意味をなさない main ノードを直下の seq ノードと併合し，それ以外のものは基本ノード単位で分割をした．

これらの分割されたノード群を合成した結果を**表 2** に示す．なお，粗粒度分割時に，テクノロジマッピングが不可能な回路が存在したため，これらはテクノロジマッピングなしの結果に関してのみ記す☆．この結果以下のような考察ができる．

☆ テクノロジマッピングを成功させるためには，マッピングできなかった複合ゲートをカスタムで生成する必要がある．

提案手法と細粒度分割の比較では，提案手法による 4%ほどの面積効率の向上が見られる．この理由は，分割数が増えるほどグルーロジックが増大するためである．したがって，この例から分割数を抑えると，コントローラの構成に必要な論理が減少することが分かる．

一方で提案手法と粗粒度分割の比較では，提案手法による 6% ほどの効率向上が見られた．これは，連続する seq ノードを積極的に分割することによって，状態爆発が抑えられ，合成の際の CSC 信号の挿入が少なかったことを意味している．したがって，分割された各ノードの合成効率を考慮すると，ノードを細粒度にして分割することが良いと考えられる．上の 2 つの評価は明らかに矛盾しており，分割の粒度にトレードオフがあると考えられる．提案手法は特に面積効率において，細粒度分割，粗粒度分割よりも粒度を最適にすることができることが，実験によって示された．

7. ま と め

本論文では，非同期コントローラの分割を行うための階層型 CDFG 記述とその分割アルゴリズムを提案した．さらにこのアルゴリズムに対して評価を行い，分割アルゴリズムによって効率的な回路生成が可能になることを示した．

階層型 CDFG は従来の CDFG の記述を制限し，このノードが非同期回路の合成条件を満たすように定義された構造である．この構造を使用することによって，非同期コントローラを自由に分割ができるようになる．本論文では，与えられた CDFG を最大入出力信号数 n に従って分割し，従来の非同期 CAD ツールで回路を合成できることを示した．

さらに本論文では階層型 CDFG に対して，非同期回路合成ツール Petrify が最適な回路を出力する分割手法を提案した．提案する分割手法は，Petrify による STG への CSC 信号の挿入を考慮し，最適な回路を導くことができるように分割を行う．この手法は実験によって，単純に細粒度，粗粒度分割を行った結果よりも，面積効率が良いことが明らかになった．本論文で提案された階層型 CDFG 構造とその分割手法を用いることによって，効率的な非同期回路の分割と合成の自動化が可能になると期待できる．また，階層型 CDFG はその構造から既存のプログラミング言語との親和性が高いと考えられる．今後はプログラミング言語などの，さらに高位の仕様からの非同期回路への自動変換などを検討していく予定である．

謝辞 本研究は文部科学省科学研究費補助金の助成により行われた．

参 考 文 献

1) De Micheli, G.: *Synthesis and Optimization of Digital Circuits*, McGraw-Hill (1994).
2) 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 (1997).
3) Bachman, B.M., Zheng, H. and Myers, C.J.: Architectural Synthesis of Timed Asynchronous Systems, *IEEE International Conference on Computer Design* (*ICCD*) (Oct. 1999).
4) Theobald, M. and Nowick, S.M.: Transformations for the Synthesis and Optimization of Asynchronous Distributed Control, *Proc. 38th Design Automation Conference* (*DAC*), June 18-22, 2001, pp.263–268 (2001).
5) Kim, E., Lee, J.-G. and Lee, D.: Automatic Process-Oriented Control Circuit Generation for Asynchronous High-Level Synthesis, *Proc. 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems* (*ASYNC2000*), April 2-6, 2000, pp.104–113 (2000).
6) Chu, T.-A.: Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications, Ph.D. Thesis, Massachusetts Institute of Technology (June 1987).
7) Kagotani, H. and Nanya, T.: Synthesis of Two-Phase Quasi-Delay-Insensitive Circuits form Dependency Graphs, 電子情報通信学会論文誌, Vol.J77-D-I, No.8, pp.548–556 (1994).
8) NEC Corporation: CB-C10 Family 0.25-μm CMOS Cell-Based IC (2.5 V) Block Library (July 1999).

(平成 13 年 9 月 21 日受付)
(平成 14 年 3 月 14 日採録)

奥山 祐市

1975 年生．1999 年会津大学大学院博士前期課程修了．2002 年同大学院博士後期課程修了．博士（コンピュータ理工学）取得．同年日本電信電話株式会社入社．コンピュータグラフィックス向けハードウェアの設計，動的再構成可能なハードウェアの設計/応用，非同期回路用 CAD の研究に従事．電気情報通信学会，ACM 各会員．

Nattha Sretasereekul

1993 年タイ国ソンクラ大学電気工学卒業．2000 年東京大学大学院情報工学修士課程修了．現在，同大学院先端学際工学博士課程に在学．

齋藤 寛

1998 年会津大学コンピュータ理工学部コンピュータハードウェア学科卒業．2000 年同大学院修士課程修了．現在，東京大学大学院工学系研究科博士課程に在学．非同期回路合成に興味を持つ．

南谷 崇（正会員）

1946 年生．1969 年東京大学工学部計数工学科卒業．1971 年同大学院修士課程修了．日本電気（株）中央研究所勤務を経て，1981 年東京工業大学情報工学科助教授，1989 年同電気電子工学科教授．1995 年東京大学計数工学科教授．1996 年同先端科学技術研究センター教授．2001 年同評議員・先端科学技術研究センター長．論理システムの物理的実現に関する諸問題に興味を持つ．工学博士．1987 年電子情報通信学会論文賞，1994 年大川出版賞，1998 年 ASP-DAC Best Paper 賞受賞．著書「順序機械」（岩波書店),「フォールトトレラントシステムの構成と設計」（槙書店),「フォールトトレラントコンピュータ」（オーム社）等．IEEE Fellow，ACM，電子情報通信学会，情報通信学会各会員．

黒田 研一（正会員）

1947 年生．1971 年東京大学教養学部基礎科学科卒業．1973 年同大学院理学系大学院相関理化学専門課程修士課程修了．同年日本電信電話公社入社．武蔵野電気通信研究所，厚木 R&D センターにて，弾性表面波デバイス，超伝導集積回路，高温超伝導薄膜，X 線光学等の研究に従事．1995 年 10 月より会津大学コンピュータ理工学部教授．工学博士．主に動的再構成可能ハードウェア，非同期回路等の研究に従事．電子情報通信学会，応用物理学会各会員．