システムＬＳＩ設計技術　106－7
（２００２．５．２３）

回路遅延を考慮した最小カット法に基づく回路分割アルゴリズム

稲木　雅人†　　畔上　謙吾††　　高橋　篤司†

† 東京工業大学 理工学研究科 集積システム専攻　〒152–8552 東京都目黒区大岡山 2–12–1
†† 富士通研究所 システム LSI 開発研究所　〒211–8588 神奈川県川崎市中原区上小田中 4–1–1
E-mail: †{inagi,atushi}@lab.ss.titech.ac.jp, ††azegami@flab.fujitsu.co.jp

あらまし　回路の分割実装に伴って生じる部分回路間配線遅延による回路動作速度の低下を抑えた回路分割手法を提案する．各ネットに対して，遅延増加に対する余裕度を計算し，余裕度の小さなネットを避けて分割することにより回路遅延の増大を防ぐ．本手法は回路をフローグラフに変換し最小カットで回路を分割する手法に基づいており，余裕度の小さなネットを，大きなフローを流す構造に変換することによって最小カット上から取り除く．また，ネットに接続する入出力ゲート間の遅延余裕度をより柔軟にゲート間のフロー容量に反映させる方法についても検討する．
キーワード　回路分割，遅延，ネットワークフロー，最小カット

Network-Flow Based Delay-Aware Circuit Partitioning Algorithm

Masato INAGI†, Kengo R. AZEGAMI††, and Atsushi TAKAHASHI†

† Communications and Integrated Systems, Tokyo Institute of Technology
2–12–1 Ookayama, Meguro-ku, Tokyo, 152–8552 Japan
†† System LSI Development Laboratory of Fujitsu Laboratories LTD.
4–1–1 Kamikodanaka, Nakahara-ku, Kawasaki-shi, 211–8588 Japan
E-mail: †{inagi,atushi}@lab.ss.titech.ac.jp, ††azegami@flab.fujitsu.co.jp

Abstract　We propose a network-flow based delay-aware circuit partitioning algorithm which takes into account the propagation delays introduced by partitioning operation. The idea is in first calculating the timing slacks of the nets and then refrain from cutting the slack-tight nets. It is done by taking into account the timing slack to determine the flow network edge capacity, i.e., the tighter the slack is, the larger the capacity of the flow network edge will be. We also describe a way to flexibly define the capacities of the flow network edges by the timing slacks between the input of a net and the outputs of the net.
Key words　circuit partitioning, delay, network-flow, mincut

1. はじめに

近年，VLSI の設計規模は非常に増大しており，回路のレイアウト，MCM 実装，マルチ FPGA 実装のためなど，様々な局面で回路分割が必要とされている．例えば一つの FPGA に収まらない規模の回路を複数の FPGA で実装する場合，回路の実現に必要な FPGA 数が多いとコストの増大につながるため，できるだけ少ない FPGA 数で回路を実現しなくてはならない．各 FPGA に実装する部分回路のサイズは FPGA の許容回路サイズ以下である必要があり，FPGA の I/O ピン数を超える数の入出力を持つことができないため，入出力数は FPGA の I/O ピン数以下でなければならない．また，FPGA 間配線は FPGA 内部の配線に比べて遅延が大きなため，分割により回路の動作速度が低下することがある．回路シミュレーションなどにおいては，回路の動作速度の低下は好ましくないため，回路の動作速度を低下させないような分割が求められている．

本稿ではこのようなマルチ FPGA 実装のための分割問題を扱う．一般にピン数制約とサイズ制約では，ピン数制約の方が厳しい場合が多い．このため，ピン数制約を満たす部分回路を列挙し，その中からサイズ制約を満たす部分回路を選択する，回路の最小カットに着目した方法が提案されている [2], [4] が，動作速度については考慮されていない．

そこで，我々は, [4] の方法を拡張し，回路の動作速度に影響を与える配線を避けて分割する方法を提案する. [2], [4] の方法では，回路をフローグラフに変換し，フローグラフ上で最小カットを探索することによって，I/O 制約，サイズ制約を満足する部分回路を切り出す．回路をフローグラフに変換する際，各配線はフローが 1 流れるグラフ構造に変換される．回路には遅延

が増大すると回路動作速度の低下に繋がる配線と，ある程度まで遅延が増大しても回路動作速度には影響を与えない配線が存在する．そこで我々は各配線を，配線の遅延余裕度に応じたフローが流れるグラフ構造に変換することによって，遅延余裕度の小さな配線の FPGA 間配線化を避け，分割回路の動作速度を向上させる．

以下，第 2 章で提案手法のための準備を行ない，第 3 章で遅延余裕度の定義，提案手法の説明を行なう．第 4 章で比較実験を行ない，第 5 章でまとめを述べる．

2. 準　　備

2.1 回路モデル

回路ネットリストを，ハイパーグラフである回路グラフ $G(V,E)$ でモデル化する．

ゲートを表す頂点の集合を V_{gate}，Flip-Flop の入力を表す頂点の集合を V_{FFin}，Flip-Flop の出力を表す頂点の集合を V_{FFout}，回路の外部入力ピンを表す頂点の集合を V_{PI}，外部出力ピンを表す頂点の集合を V_{PO} とし，$V = V_{gate} \cup V_{FFin} \cup V_{FFout} \cup V_{PI} \cup V_{PO}$ とする．また，クロックが入力される毎に出力値が変化する回路要素 $V_{PI} \cup V_{FFout}$ を V_{in}，入力値を取り込む回路要素 $V_{PO} \cup V_{FFin}$ を V_{out} とする．

E は，ネットを表すハイパー枝の集合であり，各ネット $e(\in E)$ が接続する頂点集合を $V(e)$ で表す．外部入出力ピンに接続するネット (I/O ネット) の集合を $E_{io} = \{e \in E | V(e) \cap (V_{PI} \cup V_{PO}) \neq \emptyset\}$，それ以外のネット (内部信号ネット) の集合を E_{sig} とし，$E = E_{io} \cup E_{sig}$ とする．

ネット e の入力頂点を $v_{fi}(e)$，出力頂点集合を $V_{fo}(e)$ と表す $(V(e) = \{v_{fi}(e)\} \cup V_{fo}(e))$．同様に，頂点 $v(\in V)$ の入力頂点集合を $V_{fi}(v)$，出力頂点集合を $V_{fo}(v)$，$e(\in E_{io})$ が接続する外部入出力端子を $v_{io}(e)$ で表す．

各頂点 $v(\in V)$ が対応する回路要素の遅延を $d(v)$，サイズを $\mathrm{size}(v)$ とする．また頂点集合 $V_a(\subset V)$ のサイズを $\mathrm{size}(V_a) = \sum_{v \in V_a} \mathrm{size}(v)$ とする．ネット e の入力 $v_i = v_{fi}(e)$ から出力 $v_o \in V_{fo}(e)$ の遅延を $d(v_i, v_o)$ とする．

回路の動作周期 T は

$$T = \max_{v \in V_{out}} (\mathrm{ASAP}(v))$$

と表すことができる．

ただし，$\mathrm{ASAP}(v)$ は

$$\mathrm{ASAP}(v) = \begin{cases} \max\limits_{v' \in V_{fi}(v)} (\mathrm{ASAP}(v') + d(v', v) + d(v)) & (v \notin V_{in}) \\ 0 & (v \in V_{in}) \end{cases}$$

で再帰的に定義される．$\mathrm{ASAP}(v)$ はクロックが入力されてから，回路要素 v の出力信号が決定するまでにかかる時間である．

また，$\mathrm{ALAP}(v)$ は

$$\mathrm{ALAP}(v) = \begin{cases} \min\limits_{v' \in V_{fo}(v)} (\mathrm{ALAP}(v') - d(v') - d(v, v')) & (v \notin V_{out}) \\ T & (v \in V_{out}) \end{cases}$$

で再帰的に定義される．$\mathrm{ALAP}(v)$ はいつまでに回路要素 v の出力が決定されれば回路動作周期 T を増大させないかを表す．

回路グラフ $G(V,E)$ を分割するとは，頂点集合 V を複数の部分集合 $V_1, V_2, \cdots, V_m$ と $V_{PI} \cup V_{PO}$ に分けることである．ただし $V_1 \cup V_2 \cup \cdots \cup V_m = V_{gate} \cup V_{FFin} \cup V_{FFout}$ かつ $V_a \cap V_b = \emptyset (1 \leqq a, b \leqq m$ かつ $a \neq b)$ である．また同一の Flip-Flop の入出力を表す頂点は同じ部分集合 $V_a (1 \leqq a \leqq m)$ に属さなければならない．

頂点集合 V_a に対応する部分回路に必要な I/O 数 $\mathrm{io}(V_a)$ は，V_a 内の頂点と V_a 外の頂点を結ぶ枝の数であり，

$$\mathrm{io}(V_a) = |\{e \in E | (V(e) \cap V_a \neq \emptyset) \wedge (V(e) \cap V \setminus V_a \neq \emptyset)\}|$$

と定義できる．サイズ制約を lim_{size}，I/O 数制約を $lim_{i/o}$ としたとき，V_a は $\mathrm{size}(V_a) \leqq lim_{size}$，$\mathrm{io}(V_a) \leqq lim_{i/o}$ を満たさなければならない．

ネット e の入力 $v_i = v_{fi}(e)$ から出力 $v_o \in V_{fo}(e)$ までの遅延 $d(v_i, v_o)$ は，v_i と v_o が同じ部分集合に属している場合に比べ，異なる部分集合に属している場合，大きくなる．したがって，回路の動作周期 T が増大しないように分割しなければならない．

2.2 最小カット分割手法

提案手法のもととなる最小カット分割手法 *PART* [4] の概略を述べる．*PART* は部分回路の最大サイズ，最大 I/O 数を制約として持ち，分割数の最小化を目標とする分割手法である．*PART* は与えられた回路から，I/O 制約を満たし，サイズ制約を満足するできるかぎり大きな部分回路を一つずつ逐次的に切り出すことで回路全体を制約を満たす複数の部分回路に分割する．以下では回路の未分割部分から一つの部分回路を切り出す *PART* の処理について述べる．

PART は Hyper-Flow 変換 [4] により回路グラフから複数のフローグラフを生成する．生成した各フローグラフでは，ソース s とシンク t を分割するカットの容量が有限であるとき，カットのソース側に対応する部分回路の I/O 数がカットの容量に一致する．I/O 数が小さい部分回路を切り出すために，*PART* はまず，各フローグラフにおいて *s-t* 最小カットのソース側に対応する部分回路でサイズ制約を満足するできる限り大きい部分回路を探索する．*s-t* 最小カットの容量が I/O 数制約の値を超える場合，制約を満たす部分回路を発見できないため，部分回路の探索は行なわない．次に，発見した部分回路がサイズ制約，I/O 数制約に対して余裕がある場合，制約を満たす範囲内で部分回路を拡大する操作を行ない切り出す部分回路の候補とする．各フローグラフから得られた部分回路候補のうち，サイズが大きく，サイズが等しいならば I/O 数が少なく，I/O 数が等しいならば内部信号でなくなるネットが少ない部分回路候補を部分回路として切り出す．

2.3 内部信号ネットに対する Yang-Wong 変換

回路グラフからフローグラフへの変換において，内部信号ネット $e(\in E_{sig})$ から部分フローグラフ構造への変換を Yang-Wong 変換 [1] という．変換例を図 1 に示す．

$V(e)$ の任意の部分集合を $V_s(\neq \emptyset)$ としたとき，e を変換して

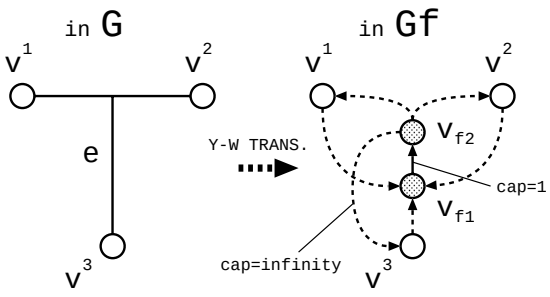

図 1　Yang-Wong 変換

得たグラフ構造上で頂点集合 V_s から $V(e) \setminus V_s$ へ最大フローを流すことを考える．このとき，V_s から流れるフローは必ず枝 $(v_{f1}(e), v_{f2}(e))$ を通り，$V(e) \setminus V_s$ に流れこむ．$(v_{f1}(e), v_{f2}(e))$ のフロー容量は 1 であり，他の枝のフロー容量は ∞ であるから，V_s から $V(e) \setminus V_s$ 間に流れる最大フローは 1 となる．すなわち，e に接続する頂点集合間に最大フローが 1 となる部分フローグラフ構造を生成する．

2.4　回路グラフに対する Hyper-Flow 変換

回路グラフ $G(V, E)$ からフローグラフ $G_f(V_f, E_f)$ への Hyper-Flow 変換 [4] を以下で定義する．以下では，フロー容量 c のフロー枝 (v_a, v_b) を $(v_a, v_b) : c$ と表す．

Hyper-Flow 変換

入力: 回路グラフ $G(V, E)$

出力: フローグラフ $G_f(V_f, E_f)$

（1）頂点集合 V_f は，

（a）ソース s，シンク t

（b）頂点集合 $V_f^v = V_{gate} \cup V_{FFin} \cup V_{FFout} \cup V_{PI} \cup V_{PO}$

（c）ダミー頂点集合 $V_f^d = \{v_{f1}(e), v_{f2}(e) | e \in E_{sig}\}$

からなる．

（2）枝集合 E_f は，

（a）回路の内部信号ネットに対応する枝集合

$$\begin{aligned} E_f^e \quad = \quad & \{(v_{f1}(e), v_{f2}(e)) : 1 | e \in E_{sig}\} \\ & \cup \{(v, v_{f1}(e)) : \infty | e \in E_{sig} \wedge v \in V(e)\} \\ & \cup \{(v_{f2}(e), v) : \infty | e \in E_{sig} \wedge v \in V(e)\} \end{aligned}$$

(Yang-Wong 変換)

（b）回路の I/O ネットに対応する枝集合

$E_f^{io} =$
$\{(v_a, v_{io}(e)) : \infty | e \in E_{io} \wedge v_a \in V(e) \setminus \{v_{io}(e)\}\}$

（c）各 Flip-Flop の各入出力頂点 v_i，v_o を結ぶ枝 $(v_i, v_o) : \infty$，$(v_o, v_i) : \infty$ の集合 E_f^{FF}

（d）シンク t に入力する枝集合

$E_f^t = \{(v_f, t) : 1 | v_f \in V_{PI} \cup V_{PO}\}$

（e）ソース s から出力する枝集合 E_f^s

からなる．

Hyper-Flow 変換では，E_f^s を様々に与えることで，回路グラフから様々なフローグラフを生成する．各内部信号ネット e をフローが高々1 流れる構造に変換するから，G_f の有限カットのフロー容量と，対応する G のカット上の枝数が一致する．G のカット上の枝数と，そのカットで回路を分割した場合に必要となる I/O 数が等しいので，フローグラフ G_f の最小カットを求めることによって I/O 数の小さな部分回路を得ることができる．

回路グラフからフローグラフへの変換例を図 2 に示す．この図において，E_f^s は空集合であるが, [4] では，I/O ネットに接続する頂点集合を選択し，ソースから容量無限大のフロー枝を加える．

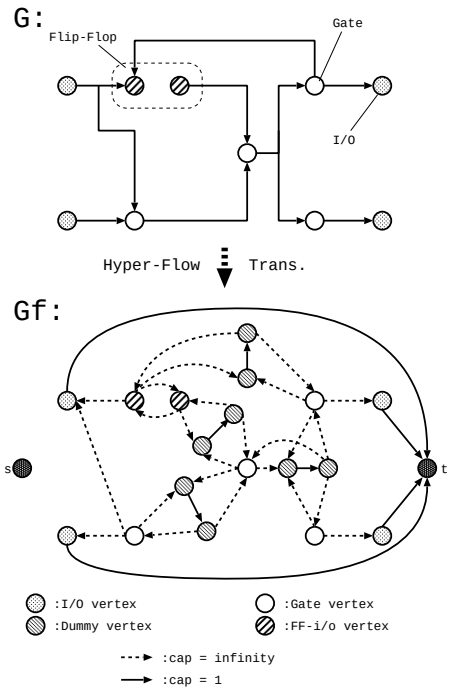

図 2　Hyper-Flow 変換

3. 提案手法

従来手法 *PART* の最適化目標は分割数の最小化のみであった．我々は分割数を増大させずに，より高速に動作する分割を得るため，*PART* を改良した手法を提案する．

提案手法では，ネットに遅延に対する余裕度 slack を定義し，slack の小さなネットができるだけ部分回路間の配線にならない分割を求める．フローグラフ上では頂点間のフロー容量を大きくすればその頂点間に最小カットが存在しにくくなるから，余裕度の小さなネットをフローがより多く流れる部分グラフ構造に変換することにより，そのネットに対応する部分が最小カットに選ばれにくくする．

以下では各ネットの入出力頂点の ASAP/ALAP 値からネットに slack 値を定義する．さらに slack をフロー容量に対応付ける weight 関数を定義し，その値を反映した変換を定義し提案手法とする．

3.1　slack

各ネットに対して遅延に対する余裕度である slack を定義

する．

$v_o(\in V_{fo}(e))$ に対応するゲートは時刻 ALAP(v_o) までに出力信号が確定すれば回路の動作周期に影響を与えない．v_i に対応するゲートの出力信号が確定するのは時刻 ASAP(v_i) であるから，v_i,v_o 間では (ALAP(v_o) − $d(v_o)$) − ASAP(v_i) までの遅延が許容できる．v_i,v_o 間には配線によって $d(v_i, v_o)$ の遅延が生じるから，さらに生じても良い遅延は

$$\mathrm{slack}_{io}(v_i, v_o) = \mathrm{ALAP}(v_o) - \mathrm{ASAP}(v_i) - d(v_o) - d(v_i, v_o)$$

までである．全ての $v^o(\in V_{fo}(e))$ に関して考えると，このネット上で生じても良い遅延は

$$\min_{v_o \in V_{fo}(e)} (\mathrm{slack}_{io}(v_i, v_o))$$

となる．したがってネット e の遅延余裕度 $\mathrm{slack}_{sig}(e)$ を

$$\mathrm{slack}_{sig}(e) = \min_{v_o \in V_{fo}(e)} (\mathrm{slack}_{io}(v_i(e), v_o))$$

と定義する．

またこのとき

$$\begin{aligned}
&\mathrm{slack}_{sig}(e) \\
&= \min_{v_o \in V_{fo}(e)} (\mathrm{ALAP}(v_o) - \mathrm{ASAP}(v_i(e)) - d(v_o) - d(v_i(e), v_o)) \\
&= \min_{v_o \in V_{fo}(e)} (\mathrm{ALAP}(v_o) - d(v_o) - d(v_i(e), v_o)) - \mathrm{ASAP}(v_i(e)) \\
&= \mathrm{ALAP}(v_i(e)) - \mathrm{ASAP}(v_i(e))
\end{aligned}$$

である．

3.2 weight 関数

slack 値に応じて，slack 値が小さい配線ほど大きなフロー容量のフロー枝を割り当てるために，slack 値に対してフロー容量を決定する weight 関数を次のように定義する．

$$\mathrm{weight}(x) = \begin{cases} \alpha - x + 1 & (x < \alpha) \\ 1 & (x \geq \alpha) \end{cases}$$

ただし α は定数として与える．

3.3 提案手法 1

PART では，$G(V, E)$ 上の枝 e は，Yang-Wong 変換によって，フローが最大で 1 流れる部分グラフ構造に変換される．ここで最大フローを決めているのは枝 $(v_{f1}(e), v_{f2}(e))$ の容量であるから，この枝の容量を γ とすれば G_f 上の頂点間には最大で γ のフローを流すことができるようになる．

従来手法 *PART* の Yang-Wong 変換のうち，枝 $(v_{f1}(e), v_{f2}(e))$: $1(e \in E_{sig})$ のフロー容量を weight(slack(e)) に置き換えた変換を Yang-Wong* 変換とする．提案手法 1 では Hyper-Flow 変換を行なうために，Yang-Wong 変換の代わりに Yang-Wong* 変換を用いる．

この変更にともない，最大フロー値と，最小カットのソース側を部分回路として切り出した場合の I/O 数が一致しなくなる．そのため，最小カット探索時には各カットで回路を分割した場合に必要な I/O 数を計数し，I/O 数制約を満たす部分回路のみを評価の対象にする．

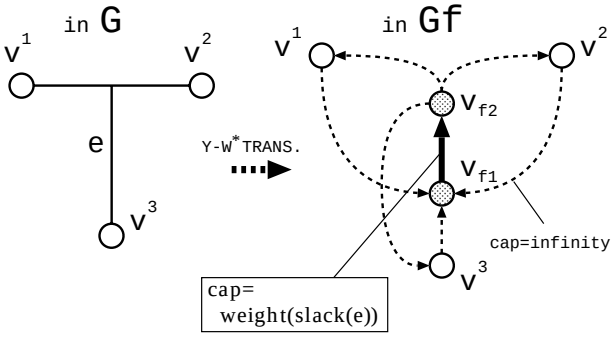

図 3　Yang-Wong* 変換

提案手法ではソースから I/O に隣接する頂点への枝を次々に付加することで複数のフローグラフを得る．

PART では，最初にソース s から I/O ネット $e \in E_{io}$ に隣接する外部入出力ピン以外を表す頂点 (I/O 隣接頂点) $v \in V(e) \setminus v_{io}(e)$ のうち次数最大の頂点へのフロー容量無限大の枝を追加し，以降はすでにソース s からのフロー容量無限大枝が追加されている頂点からの距離が最小の I/O 隣接頂点に，ソース s からフロー容量無限大の枝を追加する．ここで，s とフロー無限大枝で結ばれた頂点は，必ず s-t 最小カットのソース側部分に入り，切り出される部分回路の候補となる．そこで，提案手法 1 では slack の厳しい部分を優先して部分回路化するため，s からのフロー無限大枝を追加する頂点候補のうち，評価が等しい頂点が複数ある場合，そのうち slack が最も小さな頂点を選択するように変更する．

3.4 提案手法 2

提案手法 1 の Hyper-Flow 変換に使用した Yang-Wong* 変換では，各配線が接続する入出力ゲートペアのうち，最も遅延余裕度の小さなペアの遅延余裕度をその配線の slack とし，slack の小さな配線が部分回路間配線になりにくくすることを狙っていた．しかし，提案手法 1 では遅延余裕が最も小さなペアの値だけを反映するため，同一配線上でも遅延余裕が大きなゲートペアを別々の部分回路に入れたときに動作速度が悪化しないことが考慮されない．

ここでは提案手法 2 として，一つの配線に接続する複数の入出力ゲートペアの余裕度それぞれを最大フロー容量に反映することのできる梯子変換を提案する．

ハイパー枝 e の変換後のグラフ構造の持つべき性質について考察する．e に接続する頂点をソース/シンク側に分断する最小カットについて考える．$\mathrm{weight}(\mathrm{slack}_{io}(v_i, v_o))(v_i = v_{fi}(e), v_o \in V_{fo}(e))$ 全てを e の変換後のグラフ構造に反映させるのが提案手法 2 の梯子変換の目的である．v_i とカットの同じ側にある $v_o(\in V_{fo}(e))$ 間は新たな遅延は生じないから考慮する必要はない．v_i とカットの異なる側にある $v_o(\in V_{fo}(e))$ 間では新たな遅延が生じる．このとき回路の動作速度を決定するのは v_i，v_o 間で最も slack が小さな組み合わせであるから，その slack の値を x としたとき，枝 e を梯子変換してできた部分に $\mathrm{weight}(x)$ のフローが流れるようにしたい．

この条件を満足するグラフ構造を得る以下の変換を提案する．

梯子変換

ネット e の fanin である $v_i(=v_i(e))$ と fanout である $v_o(\in V_{fo}(e))$ 間の weight($\text{slack}_{io}(v_i,v_o)$) 値を昇順に列挙したものを $w_1,w_2,\cdots,w_n(w_a<w_{a+1})$ とし，w_a の値を与えた頂点を $v_a^1,v_a^2,\cdots,v_a^{m_a}\in V_{fo}(e)$ とする．

(1) $V_f^d(e)=\{v_{f1}^a,v_{f2}^a|1\leqq a\leqq n+1\}$

(2) $E_f^e(e)=$
$\{(v_{f1}^a,v_{f1}^{a+1}):w_a,(v_{f2}^{a+1},v_{f2}^a):w_a|1\leqq a\leqq n\}$
$\cup\{(v_a^b,v_{f1}^a):\infty|1\leqq a\leqq n\wedge 1\leqq b\leqq m_a\}$
$\cup\{(v_{f2}^a,v_a^b):\infty|1\leqq a\leqq n\wedge 1\leqq b\leqq m_a\}$
$\cup\{(v_{f1}^{n+1},v_i),(v_i,v_{f2}^{n+1})\}$

Hyper-Flow 変換において，V_f^d および E_f^e は以下のように変更される．

(1) $V_f^d=\bigcup_{e\in E\setminus E_{io}}V_f^d(e)$

(2) $E_f^e=\bigcup_{e\in E\setminus E_{io}}E_f^e(e)$

梯子変換例を図 4 に示す．

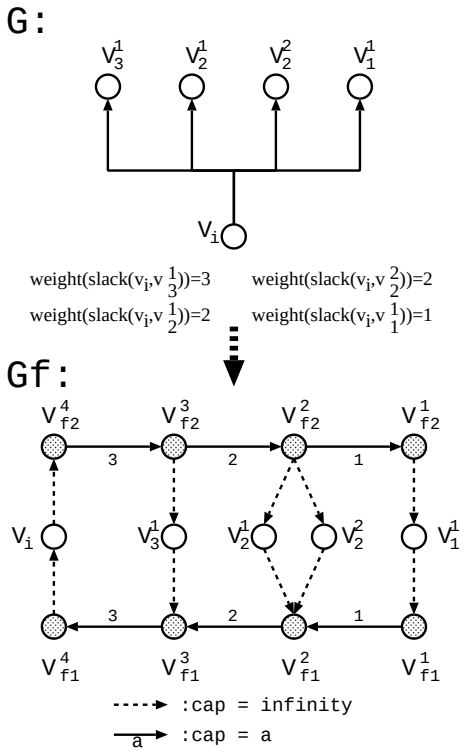

図 4 梯子変換

4. 実　　験

ISCAS85 ベンチマーク回路 (表 1) について実験環境 CPU:AthlonXP 1800+，MEM:768MB の Linux 機のもとで実験を行なった．制約，定数はそれぞれサイズ制約 $lim_{size}=200$，ピン数制約 $\text{lim}_{io}=40$，ゲート遅延 $d(v)=1$，部分回路内配線遅延 $d(v,v')=0$，部分回路間配線遅延 $d(v,v')=5$ と与えた．また，各ゲートのサイズ $\text{size}(v)$ はすべて 1 とした．

実験結果を表 2 に示す．Revisit 修正手法の再分割アルゴリズム [3]，従来手法 (*PART* [4])，提案手法 1(Proposal1) について実験を行なった．提案手法 2 については実験を行なっていない．#clu がクラスタ数，del が分割後の回路遅延，time が計算時間 (秒) を表す．また，各アルゴリズム毎に，Revisit 手法を基準とした回路遅延の平均値 (%)，従来手法を基準とした回路分割数の平均値 (%) を計算した．

実験の結果，従来手法と比べ，回路遅延では 90%，分割数では 96%を実現し，Revisit 手法と比べ，回路遅延では 106%，分割数では 73%の分割を得ることができた．

回路遅延の平均値では Revisit 手法に及ばなかったが，八個の回路中，一つの回路で同等，三つの回路で Revisit 手法よりも回路遅延の小さな分割を得ることができた．

PART に比べ，提案手法では遅延を考慮している分，分割数が増大してしまう可能性があったが，分割数にはあまり差がでなかった．提案手法では I/O に接続しないネットは Hyper-Flow 変換後のフロー容量が増大するのに対して，I/O ネットのフローは増大しないため，I/O ネットが最小カットに属しやすくなる．そのため，部分回路切り出し時に，切り出されず残る部分に必要な I/O 数の増大を防ぐことができ，I/O 数制約による分割数の増大が発生しにくくなる．その結果，遅延を考慮したことで生じる分割数の増分が抑えられていると考えられる．

表 1 実 験 回 路

回路名	ゲート数	配線数	I/O 数	分割前遅延
c499	202	243	73	21
c880	383	443	86	34
c1355	546	587	73	34
c1908	880	913	58	50
c3540	1669	1719	72	57
c5315	2307	2485	301	59
c6288	2416	2448	64	134
c7552	3512	3718	313	53

5. ま と め

最小カット法に基づく，分割により発生する回路遅延を考慮した回路分割アルゴリズムを提案し，計算機実験により，分割数は最小カットに基づく分割数最小化手法 *PART* と同等，回路遅延は Revisit 手法の再分割アルゴリズムと同等の分割が得られることを確認した．提案手法 2 は複数の入出力ゲートペアの余裕度を最大フロー容量に反映する．提案手法 2 に関する実験は行なっていないが，より良い分割が得られることを実験により確かめることを今後の課題とする．

謝　　辞

本研究を進めるにあたり，御助言くださった北九州市立大学梶谷洋司教授に感謝する．本研究は，CAD21 プロジェクトの一部である．

表 2 実 験 結 果

-	Revisit			*PART*			Proposal1		
回路名	#clu	del	sec	#clu	del	sec	#clu	del	sec
c499	5	31	0.4	5	36	9.3	5	36	6.5
c880	5	44	0.7	6	59	26	4	44	40
c1355	6	49	1.3	5	59	155	5	54	121
c1908	10	69	2.6	7	82	233	6	64	161
c3540	22	80	23	15	84	1026	15	72	1124
c5315	29	74	86	22	89	1606	25	98	1995
c6288	25	164	22	13	184	3981	13	186	3811
c7552	35	72	48	28	82	2468	28	70	3500
ave.	132%	100%		100%	118%		96%	106%	

文　　献

[1] H. H. Yang, and D. F. Wong, “Circuit Clustering for Delay Minimization Under Area and Pin Constraints,” IEEE Trans. on Computer-Aided Design, Vol.16, No.9, pp.976–986, Sep., 1997.

[2] H. Liu and D.F. Wong, “Network-Flow-Based Multiway Partitioning with Area and Pin Constraints,” IEEE Trans. on Computer-Aided Design, Vol. 17, No. 1, pp.50–59, Jan., 1998.

[3] H. Shitara, K. R. Azegami, K. Sakanushi, and Y. Kajitani, “Repartition of a Circuit for Delay Reduction Driven by Revisit of the Critical Path” Technical Report of IEICE, VLD99-124 (ICD99-281),Vol.99, No.659 (No.661), pp. 55-62, Mar., 2000

[4] K. R. Azegami, M. Inagi, A. Takahashi, and Y. Kajitani, “An Improvement of Network-Flow Based Multi-Way Circuit Partitioning Algorithm” IEICE TRANS. FUNDAMENTALS, VOL.E85-A, NO.3, pp.655–663, MARCH 2002.