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

2008－SLDM－135（6）
2008／5／8

スイッチング確率を考慮した prefix graph 合成手法の改良について

松永多苗子† 木村 晋二† 松永 裕介††

† 早稲田大学大学院情報生産システム研究科 〒808–0135 北九州市若松区ひびきの 2–7
†† 九州大学大学院システム情報科学研究院 〒819–0395 福岡市西区元岡 744
E-mail: †t_matsunaga@akane.waseda.jp

あらまし Prefix graph は parallel prefix adder の概略構造を表現するもので、加算器の、テクノロジに依存しないレベルでの構造を探索するために用いることができる。Prefix graph に対しては、これまでにタイミング制約下でのノード数最小化問題を対象として、動的計画法による面積最小化、および、再構築による面積削減の２つのプロセスからなる合成手法が提案されている。また、消費電力の一つの要因であるスイッチング確率に対する適用も提案されているが、ノード数に比べて効果があがっていなかった。本稿では、スイッチング確率を考慮した prefix graph 合成手法として、動的計画法プロセスにおけるコストに、再構築プロセスの効果を見積り補正を加える手法を提案する。特に入力タイミング制約や入力が１である確率がビットごとに一様ではない場合を例に実験結果を示し、アプローチの効果と課題を考察する。

キーワード 演算器合成, parallel prefix adder, 低消費電力, スイッチング確率

improvement of switching activity aware algorithm for prefix graph synthesis

Taeko MATSUNAGA†, Shinji KIMURA†, and Yusuke MATSUNAGA††

† Graduate School of Information, Production and Systems Waseda University 2–7 Hibikino, Wakamatsu-ku, Kitakyushu-shi, Fukuoka 808-0135, JAPAN
†† Faculty of Information Science and Electorical Engineering, Graduate School of Kyushu University 744 Motooka, Nishi-ku, Fukuoka 819–0395, JAPAN
E-mail: †t_matsunaga@akane.waseda.jp

Abstract A prefix graph represents a global structure of a parallel prefix adder, and can be utilized to search various adder structures at technology independent level. An approach for timing-driven area minimization has been proposed which consists of two phases, dynamic programming based area minimization and area reduction with restructuring. This approach is also applied to minimize the total switching activity which is one factor which affects power consumption, though it is not so effective as area minimization. In this paper, an approach is proposed which integrates the effect of the restructuring phase into dynamic programming phase to improve ability of switching cost minimization. Effects and issues of our method are discussed through experimental results.

Key words arithmetic synthesis, parallel prefix adder, low power, switching activitiy

1. はじめに

加算器は最も基本的な算術演算回路であり、これまでに数多くの構成方法が研究されている [1]。その中で、parallel prefix adder は桁上げ先見の概念を一般化した手法により桁上げの伝搬を高速化するもので、様々な特徴を持った構造が提案されている [2]〜[4] だけでなく、指定された制約に応じた適切な構造を自動生成するアルゴリズムも提案されている [5]〜[7]。自動生成の場合、例えば個々の演算で、ビットごとにタイミング制約が異なるような場合等に、各々の状況に適合させて柔軟に構成を決めることにより、より品質の高い加算器として実現できる可能性がある。

筆者らはこれまでに、parallel prefix adder の概略構造を prefix graph として表現し、タイミング制約下で prefix graph の面積最小化を行う手法を提案してきた [7]。この手法は、ある特徴をもった prefix graph の部分集合に探索範囲を絞って動

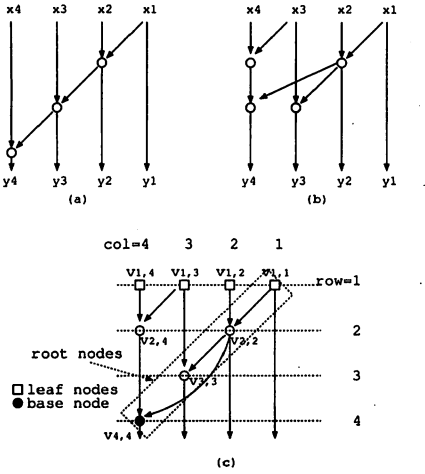

図 1 Prefix graph: (a)(b) 幅 4 の prefix graph の例, (c) ノードを入力範囲の幅で整列した prefix graph
Fig. 1 Examples of prefix graph

的計画法により最適解を厳密に求めた上で、制限を取り去って解をさらに改善するものである。また、最小化目標はノード数ではなくても、各ノードのコストの総和として計算できるものであれば同様の手法が適用できることから、消費電力の要因の一つとしてスイッチング確率を取り上げ、同手法を適用することによりスイッチング確率の総和を最小化する手法も提案した [8]。スイッチング確率の計算に OBDD [9] を用いることで、面積最小化の場合と同様に効率良く適用することが可能である。しかし、結果として得られた prefix graph の中には、既存の pararrel prefix adder の構造と比較すると prefix graph レベルのスイッチングコストは削減できたが、面積最小化を意図して生成された prefix graph のスイッチングコストと比べると、優位性は小さい、あるいは、逆にコストが大きくなってしまう場合が見られた。

本稿では、動的計画法を適用する最初のフェイズで、後半の再構築処理の影響を見積りコストを補正することにより、より最終的なコストに近い指標を対象とするアルゴリズムを提案する。また、実験を通して、提案手法の効果や、入力変数が 1 になる確率のパターンの種類による結果への影響を考察する。

以下、準備としてまず prefix graph に関する概念や用語を定義した上で、prefix graph に対する面積最小化問題とその解法、消費電力を対象とした拡張とその問題点を示す。この問題に対して補正コストを導入した提案手法を示し、実験を通してアプローチの効果、および課題について考察する。

2. 準　　備

2.1 Prefix computation と 2 進加算

Prefix computation とは、N 個の入力 $x_N, x_{N-1}, \cdots, x_1$ と結合則を満たす任意の演算 $\circ$ が与えられたとき、N 個の出力 y_i $(1 \leqq i \leqq N)$ を、$y_i = x_i \circ x_{i-1} \circ \cdots \circ x_1$ によって計算するものである。これは、i 番目の出力 y_i は，$j \leqq i$ となる入力 x_j のみに依存することを意味する。

n ビットの 2 進加算の入力を $A = a_n, \cdots, a_1, B = b_n, \cdots, b_1$、出力を和 $S = s_n, \cdots, s_1$、及び、キャリー出力 c_n とすると、各ビットの和 s_i とキャリー出力 c_i は、$s_i = a_i \oplus b_i \oplus c_{i-1}$, $c_i = a_i b_i + a_i c_{i-1} + b_i c_{i-1}$ で計算される。この n ビット加算は，個々のビットに対する桁上げ生成/伝搬関数 (g, p)、それを拡張した複数ビットのグループに対する桁上げ生成/伝搬関数 (G, P) を用いると以下のように計算できる。

- g,p の生成: $g_i = a_i \cdot b_i$, $p_i = a_i \oplus b_i$
- prefix 処理： G, P を用いて $c_i = G_{[i:1]}$ を計算

$$G_{[i:j]} = \begin{cases} g_i & \text{if } i = j \\ G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]} & \text{otherwise} \end{cases}$$

$$P_{[i:j]} = \begin{cases} p_i & \text{if } i = j \\ P_{[i:k]} \cdot P_{[k-1:j]} & \text{otherwise} \end{cases}$$

- s_i の生成: $c_i = G_{[i:1]}, s_i = p_i \oplus c_{i-1}$

Parallel prefix adder は上記の 3 つの処理を行う要素から構成される加算器である。このうち、prefix 処理の部分は、演算 $\circ$ を以下のように定義することによって，prefix computation とみなせ，その演算順序は以下に示す prefix graph によって表現することができる。

$$\begin{aligned} (G,P)_{[i:j]} &= (G,P)_{[i:k]} \circ (G,P)_{[k-1:j]} \\ &= (G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]}, \\ &\quad P_{[i:k]} \cdot P_{[k-1:j]}) \end{aligned}$$

g_i, p_i の生成、及び、s_i の計算部分に自由度はないが、prefix graph の構成には自由度があり、この部分をどのように構築するかが parallel prefix adder の特徴を決める。

2.2 Prefix graph

[定義 1] N 入力 prefix graph は、N 個の入力に対する prefix computation における各演算 $\circ$ をノードとした非巡回有向グラフ (directed ascyclic graph: DAG) であり、各出力の値を計算するための演算の実行順序を表現したものである。ノードのファンインは 2 項演算 $\circ$ の 2 つのオペランドに対応し、ノード v_1 が v_2 のオペランドであるとき、v_1 から v_2 へのエッジが存在する。v_1 を v_2 のファンインノード、v_2 を v_1 のファンアウトノードと呼ぶ。Prefix graph の入力数 (および出力数)N を、prefix graph の幅と呼ぶ。

図 1 に幅 4 の prefix graph の例を示す。(a) の y_4 は、$y_4 = x_4 \circ (x_3 \circ (x_2 \circ x_1))$ という演算順で計算され、(b) では、$y_4 = (x_4 \circ x_3) \circ (x_2 \circ x_1)$ で計算される。Prefix computation は結合則が成り立つため、どちらの順序で計算しても結果は変わらない。

図 1(c) は，図 1(b) の各ノードを入力範囲の幅で整列したものである。行 i、列 j のノード $v_{i,j}$ $(i \leqq j)$ は、i 個の入力、$x_{j-i+1}, \cdots, x_j$ に対する prefix computation の中間結果を表わしている。この入力範囲を $[j : j-i+1]$ と記す。演算の 2 つのオペランドは、連続した入力範囲に対する中間結果となっている。例えば、図 1(c) の $v_{2,4}$ は 2 つのファンイン $v_{1,4}, v_{1,3}$ を

持ち、$v_{2,4} = v_{1,4} \circ v_{1,3}$ という演算結果を表している。$v_{2,4}$ は入力 3 から 4 に対する演算結果、$v_{2,2}$ は入力 1 から 2 に対する演算結果であり、それらに演算を施した結果 $v_{4,4}$ は入力 1 から 4 に対する演算結果を表わしている。幅 N の prefix graph において、行 1 上のノード $v_{1,j}$ をリーフノードと呼び、$v_{j,j}, 1 \leqq j$ をルートノードと呼ぶ。リーフノードは入力、ルートノードは出力に対応している。ルートノードの中で最大の幅をもつノードをベースノードと呼ぶ。

2.3 Prefix graph における面積、遅延の指標

Parallel prefix adder の概略構造は prefix 処理部の構成に依存し、その構成は prefix graph によって表現される。加算器の面積は、prefix graph のリーフ以外のノード数で測るものとする。遅延に関しては、prefix graph の各ノード v に対して、入力からの遅延時間に相当する到達レベル $AL(v)$ と、そのノードの到達レベルが満たすべき要求レベル $RL(v)$ を以下のように定義する。

$$AL(v) = max\{AL(v'),\ v' \in FI(v)\} + 1$$

$$RL(v) = min\{min\{RL(v'), v' \in FO(v)\} - 1,\ RL'(v)\}$$

ここで、$FI(v)$ は v のファンインノード、$FO(v)$ は v のファンアウトノード、$RL'(v)$ は自分自身に与えられた要求レベル (未定の場合は ∞ とする) を示す。

入力ノードに対する到達レベル、出力ノードに対する要求レベルは、遅延制約として外部から与えられるものとする。ある prefix graph が制約を満たすとは、そのすべてのノード v において，$RL(v) \geqq AL(v)$ が成立することである。

3. Prefix graph 合成手法

3.1 タイミング制約の下での面積最小化

本手法は、ビットごとに与えられた入力到達レベル、出力要求レベル制約の下での prefix graph ノード数最小化問題を対象としている。もともと prefix graph は DAG であり、実用的なサイズの prefix graph に対してタイミング制約下で面積最小化の厳密解を効率よく得るのは困難な問題である。そこで、本手法では最適化の過程を 2 段階に分け、まずいったん prefix graph の部分集合に集中してその中で動的計画法を用いて厳密最適解を求め (DP フェイズ)、その解を再構築することでさらに面積削減を行う (RS フェイズ)。この部分集合は、制限付 prefix graph と呼ばれ、以下のように定義される：

[定義 2] 幅 n の prefix graph のベースノード $v_{n,n}$ の 2 つのファンインノードを、$v_{n-k,n}$, $v_{k,k}$ とする。制限付 prefix graph $R(n,n)$ とは、その構造が次の特徴をもつ prefix graph の集合である (図 2):

- 各ルートノード $v_{n-j,n-j}$ $(0 < j < n-k)$ のファンインノードは、$v_{k,k}$、$v_{n-k-j,n-j}$ である。
- $v_{n-k,n}$、$v_{k,k}$ をそれぞれベースノードとする部分グラフもまた、同様の特徴を再帰的にもつ。

部分グラフ $R2, R3$ に含まれないノード $v_{n-j,n-j}$, $0 \leqq j \leqq n-k$ を $R(n,n)$ の固有ノードと呼ぶ。

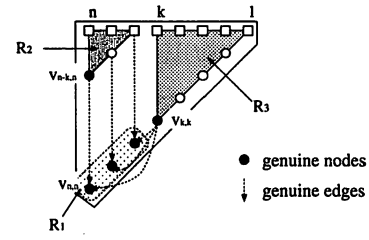

図 2 制限付 prefix graph
Fig. 2 Restricted prefix graph

制限付 prefix graph (図 2) において R_2, R_3 の部分は排他的であり、全体のコストはそれぞれのより小さい部分問題の最適解のコストの和に固有ノード分を加えれば計算できる。したがって、動的計画法を適用し、サイズの小さい部分問題から順に最適解をもとめ、それらを組み合わせて全体の解を得ることができる。ただし、制約として与えられる要求レベルは外部出力に対してであり、部分問題が満たすべき要求レベルは、その部分問題を解いている時点では確定しない。そのため、部分問題の各出力に対して可能な要求レベルの全組み合わせを想定し、各々に対して、部分問題を解いておくことが考えられる。この組み合わせは膨大な量になってしまうが、制限付 prefix graph を用いると厳密性を失うことなく、場合の数を削減することができ、結果的に、実用的なサイズの問題を現実的な時間で解くことが可能となっている。

制限付 prefix graph に対して得られた最適解は、構造に関する制限が付いている。この制限をはずして、よりノード数を削減する過程が後半のフェイズである。制限付 prefix graph の固有ノード (R_1) はどれも部分グラフ R_3 のベースノード $v_{k,k}$ をファンインとして用いることになっているが、RS フェイズではこの制約を外して、ファンインを付け替え、その結果として使用されなくなったノードを削除することでコストを削減する処理を行う。

3.2 タイミング制約下でのスイッチングコスト最小化

面積最小化は、各ノードのコストを 1 とした場合の、ノードコスト総和の最小化である。しかし、ノードのコストにばらつきがあったとしても、総和を小さくするのが目標であれば、同じ枠組が適用できる。その一つの応用例がスイッチングコストである。

回路の動的な消費電力は、ゲートを駆動する容量 C, 電源電圧 V, スイッチング周波数 f とすると $\Sigma CV^2 f$ に比例する。この中で、f を削減することのみに着目して、prefix graph 上のノードに対してスイッチングコスト SW を定義する。

まず、大前提として加算器の入力信号に対して、その信号が 1 になる確率 p_i が与えられ、かつ、ビットごとの依存性も、時間的依存性もないと仮定する。

ある信号 f が 1 から 0, あるいは、0 から 1 に遷移する確率 (以下、スイッチング確率とよぶ) SW(f) は、その f が 1 になる確率を p(f) とすると以下の式で計算される：

$$SW(f) = p(f)\cdot(1-p(f)) + (1-p(f))\cdot p(f)$$
$$= 2\cdot p(f)\cdot(1-p(f))$$

prefix graph ノードは、2.1 節で定義した式で与えられた論理を表す G, P の 2 種類の出力を持っているため、prefix graph のノード v のスイッチングコスト $SW(v)$ を以下のように定義する：

$$SW(v) = \alpha\cdot SW(G) + \beta\cdot SW(P)$$
$$SW(G) = 2p(G)\cdot(1-p(G))$$
$$SW(P) = 2p(P)\cdot(1-p(P))$$

ここで、α, β はコスト調整のためのパラメタである。

遷移確率の計算には、BDD [9] を用いたアプローチを採用する。$f = x_{j'} f_{x_{j'}} + \bar{x_{j'}} f_{\bar{x_{j'}}}$ であるので、

$$p(f) = p(x_{j'})p(f_{x_{j'}}) + p(\bar{x_{j'}})p(f_{\bar{x_{j'}}})$$
$$= p(x_{j'})p(f_{x_{j'}}) + (1-p(x_{j'}))p(f_{\bar{x_{j'}}})$$

ここで $f_{x_{j'}} = f(x_1,\cdots,1,\cdots,x_j), 1 \leqq j' \leqq j$

$f_{\bar{x_{j'}}} = f(x_1,\cdots,0,\cdots,x_j), 1 \leqq j' \leqq j$

となる。したがって、ノードの各出力のグローバル関数を表現する BDD を生成し、各変数の 1 になる確率を用いて、ノード数に比例した手間で $p(f)$ を計算できる

各ノードのグローバル関数は各行ごとに前もって生成でき、各変数の 1 になる確率が与えられた時点で、prefix graph のすべての可能なノードに対するスイッチングコストを、動的計画法を適用する前にあらかじめ計算しておくことができ、ノード数最小化の場合と同様に効率良く処理を実現することができる。

3.3 手法の評価

これまでの実験により、面積最小化は、同問題に対する既存のヒューリスティック [5], [6] や、Brent-Kung [2] や Sklansky [4]、Kogge-Stone [3] 型のような規則的な parallel prefix graph と比べて良い結果が得られている [7]。一方スイッチングコストに関しては、規則的な parallel prefix graph に比べればコストは有意に削減されるが、実は面積最小化と比べて大きくなる場合も起こり、必ずしも優位とはいえない場合が確認されている [8]。

4. 補正コストを用いたスイッチングコスト最小化

スイッチングコスト最小化の結果が、面積最小化の結果よりもスイッチングコストが大きくなってしまう原因の一つとして、DP フェイズの最小化目標は制限付 prefix graph の最小化であり、最終的な最小化目標とは異なるということがある。RS フェイズでは、DP フェイズで生成された prefix graph のノード間の接続をつなぎかえて、削除できるノードを生じさせることによりコストの削減を図る。この際、必ずしも初期解のコストが制限付 prefix graph として最小でなくても、減らせるノードが多ければ最終的なコストは削減できる。

この問題に対して、本稿では DP フェイズにおいて、より RS 処理の結果を反映させるための補正コストを導入したコスト計算方法を提案する。

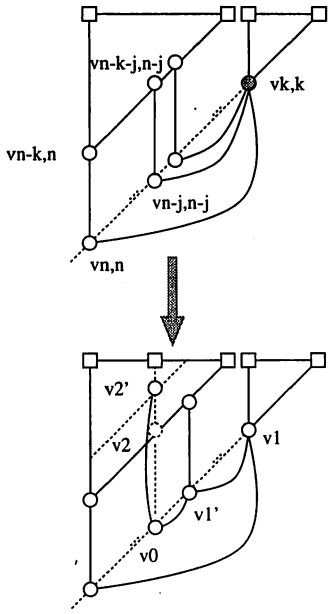

図 3 RS フェイズの処理

Fig. 3 Restructuring of a restricted prefix graph

4.1 後半フェイズの処理の詳細

図 3 に、制限付 prefix graph を出発点とした再構築の概要を示す。制限付 prefix graph では、固有ノードは必ず片方の部分グラフのベースノードをファンインとして持つ構造になっている。再構築フェイズでは、この制限を取り去り、要求レベルを満たす中で、他の固有ノードにファンインを変更する処理を行う。図 3 下の制限付 prefix graph において v_0 のファンインを v_1 から $v_{1'}$ に変更したいとすると、もう一方のファンインは v_2 から $v_{2'}$ に変更される。このとき、$v_{2'}$ がすでに存在し、かつ元のファンインノード v_2 のファンアウト数が 0 になるとすると、v_2 は削除することができる。

実際の RS フェイズにおいては、要求レベルの制約下でファンインを付け変えた場合のゲイン（削除できるノードのコストの総和）を計算し、ゲインが最も大きい付け替えを行う、という処理をゲインが得られなくなるまで繰り返した上で、部分グラフに対して同様の処理を再帰的に実行している。

4.2 提案手法

DP フェイズにおいて各部分問題を解いている際に、RS フェイズの処理を完全に模倣することは困難であるため、簡略化された再構築処理に基づいてゲインの見積りを行い、DP フェイズのコストを補正する、というアプローチをとる。

簡略化された再構築処理としては、ノード $v(n-j, n-j)$ のファンインを、$v(n-j-1, n-j-1)$、すなわち行、列ともに 1 小さいノードに変更する処理を一定順序で実行する場合のみを考える。この場合、もう一方のファンインノードはリーフノードになり必ず存在するため、要求レベルを満たした上で、

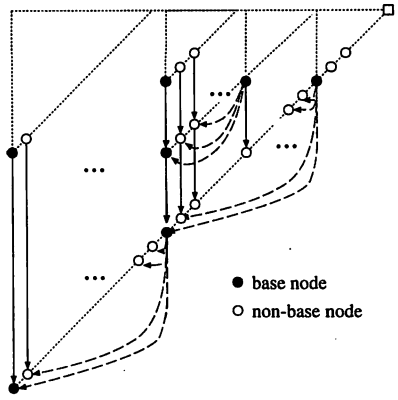

図 4　fanout の種類
Fig. 4　Types of fanouts

削減できるノードがあるかどうかをチェックすればよい。

制限付 prefix graph の場合、各ノードのファンアウトには図 4 に示すような特徴がある。すなわち、あるノードが何らかの制限付 prefix graph のベースノードになっていないならば、すなわち非ベースノードであるならば、そのファンアウトは高々1 個でそのノードも非ベースノードになる。何らかの制限付 prefix graph のベースノードになっている場合には、ファンアウトが 2 個以上存在し得るので、上記の再構築処理でファンアウトが減ったとしても 0 にはならず、削除できない。削除できるのは、同じ列上の非ベースノードのコストの和のみということになる。

なお、この指標はあくまで最終コストそのものではないので、実際には再構築の処理をここでは行わず、RS フェイズにおいて実行する。

5. 実　　験

補正コストを用いたスイッチングコスト最小化手法の効果を評価するために、上述の手法を実装し、タイミング制約あるいは入力変数の 1 になる確率にビットごとのバラツキがあるものを対象として、評価実験を行った。

- 64 ビット加算器。入力タイミング制約は凸型（図 5）。入力が 1 になる確率として、以下の 4 種類を与える

（1） uniform: すべてのビットに一様に 0.5

（2） random: ビットごとにランダム

（3） hi_00: 上位ビット群（全体の幅の 1/4）は 0 で、残りは 0.5

（4） hi_01: 上位ビット群（全体の幅の 1/4）は低確率 (0.1) で、残りは 0.5

- 16*16, 24*24, 32*32 Wallace tree 型乗算器の最終段加算器（それぞれ加算器としては、24、39、55 ビット）。入力タイミング制約は乗算器の Wallace tree 部分の出力到達レベルを計算して用いる。入力変数の 1 になる確率は、乗算器の入力各ビットにランダムに値を発生させたときの 1 になる確率を見積もっている。

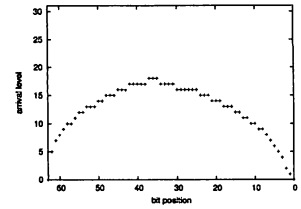

図 5　64 ビット加算器に対する入力タイミング制約
Fig. 5　Input profile for 64bit adder

表 1　実験結果 1: 凸型の入力タイミング制約をもった 64bit 加算器
Table 1　Experimental result 1: 64-bit adder with a convex input timing profile

prob	max-lv	A-min	AC-min	S-min	SC-min
uniform	22	137	135	136	136
		135.77	134.76	134.94	134.14
random	22	137	135	139	136
		114.97	114.04	113.28	111.61
hi_00	22	137	135	189	189
		24.82	24.82	22.29	22.29
hi_01	22	137	135	138	148
		44.21	44.21	41.62	42.20

表 1 は、64 ビット加算器に対して上記 4 種類の 1 になる確率を与えたときの結果の prefix graph のスイッチングコストを示している。A-min/S-min はノード数/スイッチングコスト最小化手法の結果、AC-min/SC-min は、ノード数/スイッチングコストに補正コストを加えた場合の結果を示している。

uniform と random のケースでは、補正コストを用いた処理により最終的なスイッチングコストが削減できている。また、ノード数最小化に対しても、効果が見られている。hi_00 と hi_01 の場合、ノード数最小化の結果のスイッチングコストに比べて、スイッチングコスト最小化の結果の方が 5%から 10%小さい解が得られている。ただし、補正コストを用いた場合、用いない場合よりも悪くなるケースも見られた。

表 2 は、Wallace 型乗算器の最終段加算器に対する prefix graph 合成の結果を表している。第 1 列はビット幅、2 列は最大段数制約である。3 から 5 列目は、それぞれノード数最小化、スイッチングコスト最小化、補正つきスイッチングコスト最小化の結果生成された prefix graph のスイッチングコストを表している。これらについては、それぞれ乗算器の構造から計算したタイミング制約、1 になる確率を用いている。補正コストを用いることにより、多くの場合で補正コストを用いない場合よりも結果がよくなっていたが、中には逆の場合も見られた。

一方、表 3 は、同じ対象に対して、上位ビット群の確率を低くした場合 (hi_00, hi_01) の評価を行った。これらのケースでも、表 1 の場合と同じく、面積最小化とスイッチングコスト最小化の場合で 10%前後の差が見られた。

6. おわりに

本稿では、スイッチングコスト最小化を目的とした prefix

表 2 実験結果: Wallace-tree 型乗算器における加算器（出力制約条件の違い）

Table 2 Experimental result: Adders in Wallace-tree multipliers with various output timing constraints

幅	max-lv	A-min	S-min	SC-min
24	19	31.22	31.16	30.84
	20	28.98	28.90	29.02
	21	28.97	28.10	28.21
39	21	57.26	58.06	56.51
	22	51.64	52.40	52.04
	23	50.98	50.70	50.58
55	22	84.89	85.84	84.90
	23	79.43	82.06	78.96
	24	77.06	78.11	76.69

表 3 実験結果：Wallace-tree 型乗算器における加算器 (確率の違い)

Table 3 Experimental result: Adders in Wallace-tree multipliers under various 1-probabilities

幅	max-lv	A-min	S-min	SC-min
24	19	10.68	9.35	9.35
39	21	16.88	15.92	15.92
55	22	24.45	22.45	22.45

graph 合成手法に、後処理の効果を見積もる補正コストを導入し、最初のフェイズから後半での削減効果を組み込んだコストを用いる手法を提案した。

補正コストを用いることによって結果が改善される場合が見られたが、必ずしもすべての場合ではなく、かつ、補正コスト無し、あるいは、面積最小化の結果と比べて、それほど大きい差にはならなかった。そもそも今回の補正コストの計算では、かなり簡略化された見積を行っていることも原因と思われるので、さらなる検討の余地が残っている。

また、入力が 1 になる確率の傾向によっては、スイッチングコスト最小化を対象とすることで、面積最小化の場合よりも有意に効果があがることも確認できた。より意味のある比較を行うためには、実際のデータから抽出した確率を用いる等、現実的な環境での評価が必要と思われる。

7. 謝　　辞

本研究は一部、文部科学省の早稲田大学アンビエント SoC グローバル COE プログラム、JST CREST ULP プロジェクトによる。

文　　献

[1] I. Koren, Computer Arithmetic Algorithms, A K Peters, Ltd., 2002.

[2] R.P. Brent and H.T. Kung, "A regular layout for parallel adders," IEEE Trans. Computers, vol.31, no.3, pp.260–264, March 1982.

[3] P.M. Kogge and H.S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," IEEE Trans. Computers, vol.22, no.8, pp.786–793, August 1973.

[4] J. Sklansky, "Conditional sum addition logic," IRE Trans. Electron. Comput., vol.9, no.6, pp.226–231, 1960.

[5] J. Liu, S. Zhou, H. Zhu, and C.K. Cheng, "An algorithmic approach for generic parallel adders," ICCAD, pp.734–730, November 2003.

[6] R. Zimmermann, "Non-heuristic optimization and synthesis of parallel-prefix adders," International Workshop on Logic and Architecture Synthesis, pp.123–132, December 1996.

[7] T. Matsunaga and Y. Matsunaga, "Timing-constrained area minimization algorithm for parallel prefix adders," IEICE Transactions on Fundamentals, vol.E90-A, no.12, pp.2770–2777, December 2007.

[8] T. Matsunaga, S. Kimura, and Y. Matsunaga, "Power-conscious synthess of parallel prefix adders under bit-wise timing constraints," Proc. the Workshop on Synthesis And System Integration of Mixed Information technologies(SASIMI), Sapporo, Japan, pp.7–14, October 2007.

[9] R. Bryant, "Graph-based algorithms for boolean function manipulation," IEEE Trans. Computers, vol.35, no.8, pp.677–691, August 1986.