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

2006－ＳＬＤＭ－126（20）
2006／10／27

動的計画法を用いた parallel prefix adder 合成アルゴリズムについて

松永多苗子†　　松永　裕介††

† 福岡県産業・科学技術振興財団 福岡知的クラスター研究所　〒814-0001 福岡市早良区百道浜 3-8-33
†† 九州大学システム LSI 研究センター　〒814-0001 福岡市早良区百道浜 3-8-33
E-mail: †t_matsunaga@fleets.jp, ††matsunaga@c.csce.kyushu-u.ac.jp

あらまし　本論文では，遅延制約下で面積最小化を対象とした parallel prefix adder 合成問題を，遅延制約の下での prefix graph のノード数最小化問題と捉え，動的計画法を用いて解く手法を提案する．新規性は，prefix graph の構造を制限し，その制限の特徴を利用することで，動的計画法の効率的な適用を可能にしたこと，及び，後処理として構造の制限を解除したヒューリスティックを実行することにより，解を改善することである．既存の手法に対して prefix graph のノード数が 10%程度，論理合成後の回路においては，同手法に対して 10%程度，市販ツールに比べて 35%以上小さな回路が生成される場合があることが確認された．

キーワード　演算器合成，parallel prefix adder，動的計画法，論理合成

On synthesis algorithm for parallel prefix adders using dynamic programming

Taeko MATSUNAGA† and Yusuke MATSUNAGA††

† FLEETS　3-8-33 Momochihama, Sawara-ku, Fukuoka, 814-0001 JAPAN
†† System LSI Research Center, Kyushu University　3-8-33 Momochihama, Sawara-ku, Fukuoka, 814-0001 JAPAN
E-mail: †t_matsunaga@fleets.jp, ††matsunaga@c.csce.kyushu-u.ac.jp

Abstract　This paper addresses parallel prefix adder synthesis which targets area minimization under given timing constraints. This problem is treated as synthesis of prefix graphs, which represent global structures of parallel prefix adders, and an algorithm using dynamic programming is proposed. Contributions are to enable dynamic programming to be applied efficiently by introducing and utilizing an appropriate constraints for prefix graphs, and to apply heuristics which reduce the number of nodes by relaxing the constraints. Experimental results show that the number of nodes of generated prefix graphs can be reduced by about 10%, and area after logic synthesis can be reduced by about 35% than existing methods.

Key words　arithmetic synthesis, parallel prefix adder, dynamic programming, logic synthesis

1. はじめに

加算，乗算等の算術演算を実現する演算器は，設計において回路全体の品質に影響を与える重要な要素である．中でも加算は，最も基本的な演算で用途も多いため，古くより多くの研究がなされており，様々な特徴をもつ構成が知られている [1], [2]．また，固定的な構成ではなく，個々の加算器が使用される際の制約条件や目的に応じた parallel prefix adder を自動生成する手法も現れてきている [4], [5]．論理合成ツールを用いた設計フローを用いる場合，演算器を含む回路全体の設計過程において，個別の演算に対する制約が定まる場合も多いため，その場の状況に応じて適切な構成を自動生成できることは有用である．

本論文では，この parallel prefix adder の合成問題を対象とする．ビットごとの入力/出力遅延制約が与えられた場合の面積最小化問題を，prefix graph におけるノードのレベルの形で与えられた遅延制約の下で，prefix graph のノード数を最小化する問題として捉える．Prefix graph の構造に制限を与えることによって，動的計画法の適用を可能にし，制限された構造内での厳密解を実用的な時間で求める．また，得られた厳密解に対して，構造の制限を解除するヒューリスティックを加えることにより，解を改善する．

以下では，まず，prefix graph，および，prefix adder 合成

問題を定義した上で，本手法における探索空間の削減方法について述べ，実験によりノード数最小化の効果を評価する．さらに，生成した概略構成から演算器を合成して既存の加算器合成手法との比較を行う．

2. 準　　備

2.1 Prefix computation

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，以下のように計算するものである．

$$y_i = x_i \circ x_{i-1} \circ \cdots \circ x_1$$

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．

Prefix computation は結合則が成り立つため，必ずしも逐次的に計算する必要はない．例えば，以下のどの順序で計算しても結果は変わらない．

$$\begin{aligned} y_4 &= x_4 \circ (x_3 \circ (x_2 \circ x_1)) \\ &= (x_4 \circ x_3) \circ (x_2 \circ x_1) \\ &= (((x_4 \circ x_3) \circ x_2) \circ x_1) \end{aligned}$$

2.2 Prefix graph

Prefix graph $G(n)$ 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 DAG(Directed Ascyclic Graph) 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ている．Prefix graph の入力数 (および出力数)n を，prefix graph のサイズと呼ぶ．

図 1(a)，(b) は，サイズ 4 の prefix graph の例である．(a) は，

$$\begin{aligned} y_4 &= x_4 \circ (x_3 \circ (x_2 \circ x_1)) \\ y_3 &= x_3 \circ (x_2 \circ x_1) \\ y_2 &= x_2 \circ x_1 \\ y_1 &= x_1 \end{aligned}$$

という演算順を表わしており，(b) は，

$$\begin{aligned} y_4 &= (x_4 \circ x_3) \circ (x_2 \circ x_1) \\ y_3 &= x_3 \circ (x_2 \circ x_1) \\ y_2 &= x_2 \circ x_1 \\ y_1 &= x_1 \end{aligned}$$

を表している．

図 2 は，図 1(b) の各ノードを，それが関与する入力の個数ごとに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 2 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．

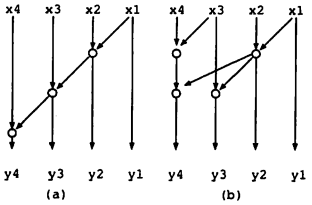

図 1　サイズ 4 の prefix graph の例
Fig. 1　Examples of prefix graphs (size = 4)

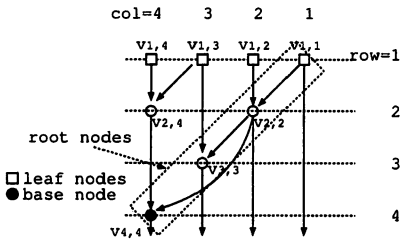

図 2　ノードを入力範囲の幅で整列した prefix graph
Fig. 2　An aligned prefix graph

$G(n)$ において，行 1 上のノード $v_{1,j}$ をリーフノードと呼び，$v_{j,j}$ 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 j をもつノードをベースノードと呼ぶ．$G(n)$ における ベースノードは，$v_{n,n}$ である．

2.3 Prefix computation としての 2 進加算

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，各ビットの和およびキャリー出力 s_i, c_i は以下のように定義される．

$$s_i = a_i \oplus b_i \oplus c_{i-1}$$

$$c_i = a_i b_i + a_i c_{i-1} + b_i c_{i-1}$$

n ビット加算は，個々のビットに対する generate 関数 (g), propagate 関数 (p)，および，それを複数ビットのグループに拡張した，グループ generate 関数 (G), グループ propagate 関数 (P) を用いて，以下のように計算できる．

- ビットごとの (g, p) の生成 (pre-processing)

$$g_i = a_i \cdot b_i$$

$$p_i = a_i \oplus b_i$$

- prefix 処理：(G,P) を用いて $c_i = G_{[i:1]}$ を計算する．

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

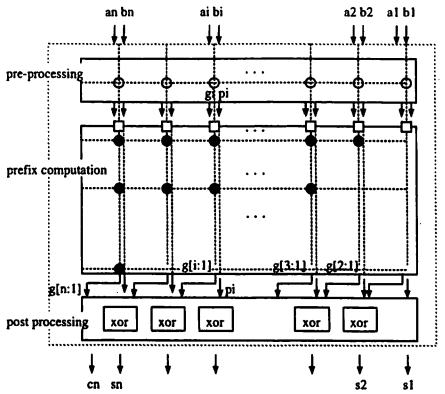

図 3 Parallel prefix adder の構成
Fig. 3 Structure of parallel prefix adder

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

この部分は，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，prefix graph で表現することができる．

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

- 各ビットの和 s_i の生成 (post-processing)

$$c_i = G_{[i:1]}$$

$$s_i = p_i \oplus c_{i-1}$$

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 (図 3)．ビットごとの g,p の計算，および，各ビットの和の計算部分の構成は一意に決まるが，prefix 処理の部分には自由度があり，この部分が parallel prefix adder の特徴を決める．

2.4 Prefix graph における遅延，面積の指標

Parallel prefix adder の概略構造は，prefix 処理部の構成に依存し，それは 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 の各ノードに対して，入力からの遅延時間に相当する到達レベルと，そのノードの到達レベルが満たすべき，要求レベルを定義する．

- ノード v の到達レベル $AL(v)$：

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

$FI(v)$ は v のファンインノードを示す．

- ノード v の要求レベル $RL(v)$：

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

$FO(v)$ 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，$RL(v) \geq AL(v)$ が成立することである．

2.5 Parallel prefix adder 合成問題

本論文では，遅延制約下で面積最小化を目指した parallel prefix adder 合成問題を，下記の制約/目的関数をもつ prefix graph 生成問題と捉える．

- 制約条件
  - ビットごとの入力到達レベル
  - ビットごとの出力要求レベル
- 目的関数：Prefix graph のノード数最小化

3. 提案手法

対象とする prefix graph 生成問題は，DAG に対する遅延制約下でのノード数最小化を目的としている．遅延最適解を求めることだけを考えるならば，動的計画法を用いて計算できる ([5])．しかし，ノード数は DAG の場合単純な足し算にならない．したがって，最適性の原理が保証されないため動的計画法を適用することができず，効率よく探索することが難しい．

そこで，以下の 2 つの観点から探索空間を減少させる：

（1） prefix graph の構造に対する制限

探索対象をある一定の構成をもつ prefix graph に限定することによって，動的計画法を適用して部分問題を組合せて全体の最適解を生成することを可能にする．

（2） 部分問題に対して考慮すべき制約の数の削減

制限された prefix graph 構造の特徴を利用して，部分問題を解く際に考慮すべき要求レベルの組合せを大幅に減少させる

(1) により元の探索空間の部分空間しか探索しないので，最適解が見落とされる可能性がある．これに対しては，部分空間での最適解に後処理を加えて，解の改善を行う．

以下，構造の対する制限，部分問題に対する制約，及び、後処理について説明する．

3.1 構造に対する制限

Prefix graph $G(n)$ の構造を，以下のように限定する（図 4）：

- ノード $v_{n,n}$ のファンインを $v_{n-k,n}, v_{k,k}$ とする
- ノード $v_{n-j,n-j}\ 1 \leqq j < k$ の片側のファンインを，$v_{k,k}$ とする．このとき、もう一方のファンインは $v_{n-k-j,n-j}$ となる．
- ノード $v_{n-k,n}$, $v_{k,k}$ をベースノードとする部分グラフにおいても，再帰的に同様の構造が成り立つ．

ノード $v_{n',p}$ をベースノードとして上記の制限を満たした prefix graph を $G'(n',p)$ と記す．

G' は以下の特徴をもつ：

- $G'(n,n)$ は，入力数のより少ない 2 つのグラフ，$G'(n-k,n)$，$G'(k,k)$ と $\{v_{add}\},\{e_{add}\}$ から構成される．
- $G'(n-k,n)$ のルートノードに対する要求レベルは，$G(n)$ のルートノードに対する要求レベル RL_o から唯一に定まる．入力到達レベルは同一．
- $G'(n,n)$ のノード数は，部分グラフ $G'(n-k,n)$,

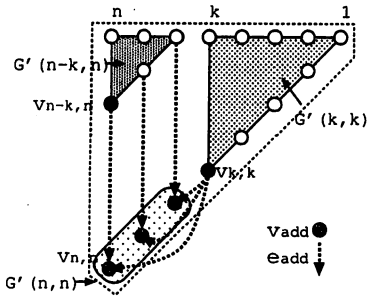

図 4 限定された構造の prefix graph
Fig. 4 Restricted prefix graph

$G'(k,k)$ それぞれのノード数と $|\{v_{add}\}|$ の総和で求まる．

- $G'(n,n)$ が制約の下でのノード数最小解であるならば，$G'(n-k,n)$，$G'(k,k)$ は 2 つの部分問題，
  - RL_o から生成される制約 RL'_o の下での $G'(n-k,n)$ のノード数最小化問題
  - RL_o の下での $G'(k,k)$ のノード数最小化問題

に対するそれぞれ最小解である．これは，もし最小でないとするならば，それぞれの最小解に置き換えることで $G'(n,n)$ に対するより小さい解が生成されることから証明される．

以上により，$G'(n,n)$ のノード数最小化問題は最適性の原理が成り立ち，動的計画法により，部分問題の最適解から全体の最適解を導くことが可能となる．

3.2 部分問題に対する制約

サイズ $n' < n$ の部分グラフ $G'(n',p)$ に対する部分問題を解くためには，そのグラフの n' 個のルートノードに対する要求レベルが必要となる．要求レベルは，そのノードの推移的ファンアウトが決まらないと計算できないため，サイズの小さい順に部分問題を解いている段階では定まらない．そこで，遅延を考慮したテクノロジマッピングで用いられている手法 ([3]) と同様，そのノードがとりうる要求レベルの範囲を想定して，それぞれの場合について最適解を求めておき，ファンアウトの構成が決まった段階で，定まった要求レベルに対する最適解を選択していくことで，解を生成する．

部分グラフの各ルートノードが，r_j 通りの値を要求レベルとしてとりうるとすると，サイズ n' のグラフに対する部分問題としては，それらすべての組合せ，すなわち，$\Pi_{j=1}^{n'} r_j$ 通りの制約集合を考慮する必要があることになり，すべての部分問題において，その制約全てに対する解を求めるのは現実的ではない．そこで，G' の構造の特徴を利用して，考慮すべき制約の個数を削減する．

3.2.1 保持すべき要求レベルの個数の削減

G' における各ノード $v_{i,j}$ のファンアウトは，次の 2 種類に分類される：

- $v_{i',j}$ $(i < i')$：垂直成分
- $v_{i+k,j+k}$ $(1 \leq k)$：斜め成分

$v_{i,j}$ と垂直成分のノードを結ぶエッジを垂直エッジ，斜め成分ノードを結ぶエッジを斜めエッジと呼ぶことにすると，$v_{i,j}$ は

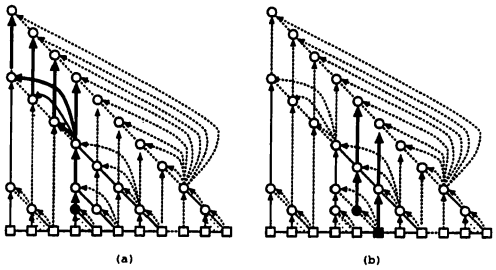

図 5 ノードのファンアウト

- 唯一の垂直エッジをもつ
- k 個 $(k \geq 1)$ の斜めエッジと高々1 個の垂直エッジをもつ

のどちらかであり，かつ，斜めエッジをもちうるのは，そのノードが部分グラフにおけるベースノードになる場合のみである (図 5(a))．ベースノードにならないノードは，垂直エッジしかもたず，その推移的ファンアウトノードがベースノードになることはない．したがって ベースノード以外のノードから出力へのパスは一通りであり，出力へのパスの長さは垂直エッジの個数 m で定まる (図 5(b))．また，m は同じ部分グラフのルートノードでは同一であるため，部分グラフに対して一意に定まり，ベースノード以外のルートノードの要求レベルは，以下の式で計算できる：

$$RL(v_{i,j}) = RL(v_{j,j}) - m$$

以上により，各部分問題に対して，すべてのルートノードに対する要求レベルを用意する必要はなく，ベースノードの要求レベルと，それ以外のノード集合用の垂直エッジの個数という 2 つの要素のみに着目すれば，すべてのノードの要求レベルを計算できる．

3.2.2 垂直エッジの個数 m のとりうる範囲

- m の最小値 m_{min}：

$$m_{min} = \begin{cases} 0 & \text{if } pos = n' \\ 1 & \text{otherwise} \end{cases}$$

- $G'(n',pos)$ における m の最大値 m_{max} は，n'(部分グラフのサイズ) と pos(ベースノードの位置) から決まる (図 6)：

$$m_{max} = pos - n' \qquad 1 \leq n' \leq pos \leq n$$

3.2.3 ベースノードの要求レベルのとりうる範囲

- 下限

構造の制約がない場合の最小到達レベルを AL_{min} とすると，$RL(v_{i,j}) \geq AL_{min}(v_{i,j})$

- 上限

あるグラフのベースノードから出力へのパスは複数存在するが，それぞれにおける垂直エッジの数は，同じ部分グラフのベースノード以外のノードから出力への垂直エッジの数 m に等しい．要求レベルは最小値で効いてくるため，斜めエッジの存在は要求レベルを厳しくする方向にしか働かない．したがって，$RL(v_{i,j}) \leq RL(v_{j,j}) - m$ であり，要求レベルの範囲の上限として使える．

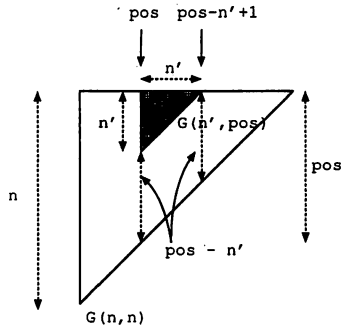

図 6 $G'(n', pos)$ の垂直エッジ数の範囲

Fig. 6 Possible range of the number of vertical edges

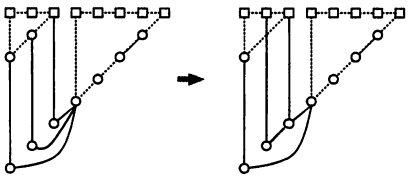

図 7 後処理によるノード数の削減

Fig. 7 Post processing for reduction of the number of nodes

3.3 後 処 理

本手法で加えた構造への制限により，探索空間から除外される prefix graph が存在することになるため，元の問題に対する最適性が失われる．これを改善するために，$G'(n,n)$ に対して得られた最小解に対して，下記のヒューリスティックを用いてノード数を削減する（図 7）．

- ベースでないノード $v_{i,j}$ のうち，片方のファンイン $v_{i-k,j-k}$ を $v_{i-1,j-1}$ に変更しても要求レベルを満たす場合，ファンインをつけ変える．この時，もう片方のファンインは同じ列のリーフノードになる．もとのファンインノードはファンアウトがなくなり，出力へ到達可能でなくなるため削除できる．

4. 関 連 研 究

ビットごとに与えられた入力到達レベル，出力要求レベルを満たす範囲で，prefix graph のノード数最小化を行う手法としては，文献 [4], [5] が知られている．文献 [4] は，prefix graph に対して局所変換を順次適用するもので，実行時間が短く，実験的にノード数最小，あるいは，最小に近い解が得られるという報告があるが，最適性の保証はない．文献 [5] の手法は，動的計画法を用いて遅延最小解を求めることが基本となっている．面積に関しては，各ノードの最小レベルを計算したのち，出力から逆向きに走査してノードに対する要求レベルを計算し，要求レベルが満たされる範囲で，できるだけシリアルに近い形を選ぶ，というヒューリスティックによって，面積の削減を行っている．制約を満たすことは保証されるが，面積に関しては最適性の保証はない．

本手法は，構造を限定することによって，その中で厳密最小解を求めるもので，制限したことによるデメリットは，後処理のヒューリスティックで回復する．後処理のヒューリスティックは，Liu の手法で「シリアルに近い形を選ぶ」のと類似の処理であるが，ヒューリスティックを適用する以前に，制限された中ではあるが，まずグローバルに最適な構造を得るところが異なっている．

表 1 入力レベルが均一な場合（左表 32 ビット，右表 64 ビット）

Table 1 Results with uniform input timing constraints

lv	DP	liu	min
5	74	74	-
6	59	66	-
7	55	58	55
8	54	57	54
9	53	59	53
10	52	53	52

lv	DP	liu	min
6	169	170	-
7	142	152	-
8	125	137	-
9	120	130	117
10	117	124	116
11	115	123	115

5. 評　　価

5.1 Prefix graph のノード数の比較

提案手法の効果を評価するために，下記のタイミング制約の下で prefix graph を生成し，文献 [5] の手法とノード数の比較を行った．

- 入力到達レベル（0），出力要求レベル（表中の lv）ともばらつきがない場合．（表 1）
- ランダムに入力にばらつきを与えた場合（表 2 左）
- 凸型のばらつきを与えた場合（表 2 右）

表中，DP は本手法，liu は文献 [5] の手法を実装した結果である．入力/出力にばらつきのない n ビットの prefix graph においては，そのノード数 S と深さ D（ノードの到達レベルの最大値）の間には，$D + S \geq 2n - 2$ という関係が成り立ち，特に深さが $(2\ log\ n - 3) \leq D \leq (n-1)$ の範囲にあるときは，等号が成り立つ解が存在することが知られている [6]．表 1 における min 欄は，この式から求めたノード数最小値である．

上記 3 つのどのケースにおいても，Liu の手法より少ないノード数が得られた．また，入力にばらつきがない場合で，レベル数制約が比較的緩い場合には，厳密最小解が得られている．ノード数削減の割合は，与えた制約によって変わってくるが，ランダムに与えた場合に関しては，平均で 10%程であった．

処理時間に関しては，ビット幅と制約条件の緩さに依存するが，64 ビット，深さ 11 の場合で 10 秒程度，128 ビット，深さ 7 で 100 秒程度であり，実用上，適用可能であると考えられる（CPU: Pentium 4 3.0GHz, OS: FreeBSD 5.3）．

参考までに，文献 [4] に示されていたデータに対して，本手法を実行した結果を表 3 に示す．ビット幅は 32 で，表に示したような特徴をもつ制約が与えられている．[4] の項の値は，文献に書かれていたものである．ほぼ同等の結果が得られているが，本手法の方がノード数が大きくなる場合も見られた．この他，表 1 左 (32bit) の例に関しては，同じ結果が得られたが，それ以外のデータについては比較ができていない．この手法はヒューリスティックなので，結果の傾向を予測することはで

表 2 タイミング制約が一様で無い場合 (32 ビット)
Table 2 Results with non-uniform input timing constraints

	DP	liu
R0	61	66
R1	58	61
R2	57	63
R3	58	66
R4	58	65
R5	53	57
R6	54	56
R7	53	59
R8	58	70
R9	58	67
平均の比	1.00	1.11

	DP	liu
C	56	65
D	64	72

表 3 文献 [4] の手法との比較
Table 3 Comparison with [4]

	dp	[4]	制約の特徴
A	49	50	上位 16bit の入力到達レベルが遅い場合
B	61	61	下位 16bit の入力到達レベルが遅い場合
C	56	56	凸型
D	64	63	緩やかな凸型
E	55	55	上位 16bit の出力要求レベルが遅い場合
F	73	73	下位 16bit の出力要求レベルが遅い場合
G	59	59	出力のうち 1 ビットのみ要求レベルが早い場合
H	80	78	入力のうち 1 ビットのみ到達レベルが遅い場合
I	70	68	入力のうち 1 ビットのみ到達レベルが遅い場合

きないため，より詳細な比較/評価を行うためには，実際に文献 [4] の手法を実装する必要がある．

5.2 合成後の回路面積の比較

ランダム生成の場合と，凸型の場合について，レベル 1 段分を遅延時間に換算して入力到達時刻を設定し，市販の論理合成ツールでテクノロジマッピング (TSMC 0.13 artisan library) を行った結果を示す．表 4，5 において，max-delay は最大遅延制約の値 (ns) である．bk，sk，ks はそれぞれ既存の parallel prefix adder (Brent-Kung 型，Sklansky 型，Kogge-Stone 型) を合成した結果である．市販ツール (表中 LS) 以外は，prefix graph に相当する verilog 記述に対して論理合成を行った結果であり，市販ツールに関しては，算術式の形で verilog 記述を与え，自動的に演算器合成を行った結果である．合成オプションとしては，概略構造を変更しない範囲で論理最適化/マッピングを行っている．表の例に対しては，本手法を用いて prefix graph レベルで入力ばらつきを考慮した概略構造を生成することにより，固定の構成を使用する場合，論理合成で実現する場合，既存の手法を用いる場合と比較して，最終的な回路面積が小さくなることが確認された．削減率は，既存手法に対して，10%程度，市販ツールに対しては，35%以上削減できる場合があることが示されている．

6. おわりに

本論文では，parallel prefix adder 問題を，その概略構造を

表 4 論理合成後の面積の比較 (凸型)
Table 4 Area comparison after logic synthesis(convex)

max-delay	DP	liu	LS	bk	sk	ks
2.50	1994	2478	2702	2823	2976	5326
3.00	1568	1725	1906	1721	1938	2984
3.50	1499	1662	1726	1670	1937	2987

表 5 論理合成後の面積の比較 (ランダム)
Table 5 Area comparison after logic synthesis(random)

max-delay	DP	liu	LS	bk	sk	ks
2.00	1657	1820	2315	1947	2050	3113
2.50	1599	1719	1871	1711	1923	2987
3.00	1604	1728	1750	1677	1937	2987

表わす prefix graph の探索問題として捉え，構造に対する制限と，考慮すべき制約の数の削減により探索空間を削減する手法について提案した．構造の制限により元の問題に対する最適性は保証されないが，後処理によって解を改善することにより，結果的に既存手法より 10%程度ノード数の小さい prefix graph を生成できる場合があることを確認した．また，論理合成後の回路の面積については，与えられた制約によって，既存のツールに比べて，35%以上小さくなる場合があることを示した．与える制約によっては，効果の度合いが変わると思われるが，本手法による概略構造レベルでの考慮が，最終的な回路の質に大きく影響を与える場合があることが確認できた．

今後の課題としては，まず後処理に関する体系的な考察が考えられる．また，より制約/目的に即した回路を生成するために，ノード数/要求レベルより精度の高い面積/遅延の指標を扱うことや，テクノロジマッピング技術との融合があげられる．

7. 謝　辞

本研究は，福岡地域の文部科学省知的クラスター創成事業の支援による．

文　献

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

[2] Neil H.E. Weste, David Harris, Chapter 10. Datapath Subsystems, in CMOS VLSI Design: A Circuits and Systems Perspective, pp.637-711, Addison Wesley

[3] H. Touati, C. Moon, R. K. Brayton, and A.Wang, "Performance-oriented technology mapping", MIT VLSI Conference, 1990.

[4] R.Zimmermann, "Non-heuristic optimization and synthesis of parallel-prefix adders", in Proceedings of International Workshop on Logic and Architecture Synthesis, Dec.1996, pp.123-132.

[5] Jianhua Liu, Shuo Zhou, Haikun Zhu, and Chung-Kuan Cheng, "An Algorithmic Approach for Generic Parallel Adders", in Proceedings of ICCAD'03, pp.734-740, Nov.2003.

[6] Marc Snir, "Depth-Size Trade-offs for Parallel Prefix Computation", in Journal of Algorithms 7, 185-201, 1986.