一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
VLD2024-84,RECONF2024-114(2025-01)

一般化並列カウンタのチェーン状接続による コンプレッサツリーの効率的 FPGA 実装

野田　　麦†　石浦菜岐佐†

† 関西学院大学　〒669–1330 兵庫県三田市学園上ケ原 1
E-mail: †{mugi-noda,ishiura}@kwansei.ac.jp

あらまし　本稿では, 乗算器や積和演算器のコアとなる多入力加算器の効率的 FPGA 実装法として, 一般化並列カウンタ (Generalized Parallel Counter; GPC) のチェーン状接続による回路構成法を提案する. GPC は全加算器を拡張したものであり, 従来法では GPC による桁上げ保存加算器の木 (コンプレッサツリー) の最適構成を整数線型計画法で求めている. また, 回路規模削減効果の大きい GPC (6,0,7;5) をチェーン状に接続することにより, 限られた条件でさらに回路規模を削減する手法が提案されている. 本稿ではこれを一般化し, コンプレッサツリーに GPC のチェーン状接続を導入することにより回路規模を削減する手法を提案する. 本手法を用いて 8 ~ 32 ビットの乗算回路および多入力加算器を構成した結果, 従来の最適なコンプレッサツリーを構成する方法と比較して, 同程度のクリティカルパス遅延を維持しつつ, 回路規模を乗算回路では平均 13.06%, 多入力加算器では平均 11.08% 削減した.
キーワード　多入力加算器, 一般化並列カウンタ, FPGA, コンプレッサツリー

Efficient FPGA Implementation of Compressor Trees Based on Generalized Paralell Counter Chains

Mugi NODA† and Nagisa ISHIURA†

† Kwansei Gakuin University,　1 Uegahara, Gakuin, Sanda, Hyogo, 669-1330, Japan
E-mail: †{mugi-noda,ishiura}@kwansei.ac.jp

Abstract　This paper proposes an efficient FPGA implementation of multi-input adders, serving as the core of multipliers and multiply-accumulators, through the chain connection of generalized parallel counters (GPCs). GPCs are an extension of full adders, and previous methods determine the optimal configuration of carry-save adder trees (compressor trees) for multi-input adders using GPCs via integer linear programming. On the other hand, a method has been proposed to further reduce circuit size under limited conditions by connecting GPCs (6,0,7;5), which have a significant effect on circuit size reduction, in a chain structure. In this paper, we generalize this approach to reduce circuit size by incorporating chained GPCs into compressor trees. Using this method, we designed multiplication circuits and multi-input adders for bit widths ranging from 8 to 32. As a result, we achieved an average circuit size reduction of 13.06% for multiplication circuits and 11.08% for multi-input adders, compared to conventional optimal compressor tree configuration methods, while maintaining a comparable critical path delay.
Key words　multi-input adder, generalized parallel counter, FPGA, compressor tree

1. はじめに

複数の 2 進数の和を計算する多入力加算は, 乗算や積和演算等の様々な算術演算回路のコアとなる. 近年ではニューラルネットワークのハードウェアアクセラレーションにおいてもその重要度が高まっている.

多入力加算器の効率的な実装手法としては, 3 入力 2 出力の全加算器を基本素子として桁上げ保存加算器の木を構成する手法 [1] [2] が古くから知られている. しかし, LUT (Look-up Table) 型の FPGA での実装を考えた場合, 全加算器に基づく回路は必ずしも 5 ~ 6 入力の LUT を効率的に活用できるとは限らない.

このため, 全加算器を 6 入力 3 出力に拡張した加算器 [3] や, 入力に 1 だけでなく 2 の冪乗の重みを許した, 一般化並列カウンタ (Generalized Parallel Counter; GPC) を基本素子に用いる手法が提案されている [4] [5] [6] [7]. また, FPGA 中に実装されているキャリーロジックを活用した GPC の構成も考案されており,

Xilinx 7 シリーズの FPGA で効率的に実装可能な GPC が種々提案されている [8] [9] [10].

GPC を基本素子とした桁上げ保存加算器の木をコンプレッサツリー (Compressor Tree) と呼ぶ. コンプレッサツリーは全加算器の木よりも構造が複雑なため, 最小の段数と回路規模を得るためのヒューリスティックアルゴリズムや整数線型計画問題への定式化が提案されている [3] [5] [6] [7] [8] [9].

コンプレッサツリーにおいて, GPC はビット数を削減する役割を果たすため, ビット数の削減効率の高い GPC の使用が回路規模の削減につながる. 1 スライス実装可能な GPC のうち最もビット数削減効率が高いのは 13 入力 5 出力, 即ち 13 ビットを 5 ビットに削減する GPC (6,0,7;5) である. しかし, この GPC が 1 スライスで実装できるのは, この GPC の最下位の入力が別の GPC のキャリー出力から供給されている場合に限られ, それ以外の場合には 2 スライスを消費してしまう.

文献 [11] ではこの点に着目し, GPC (6,0,7;5) のキャリーロジックをチェーン状に接続することにより 6 つの 2 進数を 2 つにまで加算する 6-2 adder を作り, それを樹状に接続することにより多入力加算器を構成する手法を提案している. この手法では, 文献 [7] [8] の最適コンプレッサツリー構成法よりもさらに回路規模と遅延の小さな多入力加算器を得ることができる. しかし, この手法で最適な回路を構成できるのは, 複数の値の加算のような単純な入力形状に限られる.

本稿では, 文献 [11] の手法を一般化し, GPC のチェーン状接続をコンプレッサツリーに導入することにより, 一般の入力形状で最適な回路を得られるようにする. 本手法では GPC (6,0,7;5) のみをチェーン接続するのではなく, それ以外の GPC をチェーン状に接続することを許した上で, コンプレッサツリーの場合と同様に整数線型計画法により最適な回路構成を求める.

提案手法に基づき 8 ~ 32 ビットの乗算回路と多入力加算回路のコンプレッサツリーを構成した結果, いずれの場合においても文献 [7] [8] の最適コンプレッサツリー構成法に比べて回路規模が平均で約 10 % 小さく, クリティカルパス遅延が同程度の回路を得ることができた.

2. 一般化並列カウンタによる多入力加算器

2.1 LUT 型 FPGA のアーキテクチャ

LUT (Look-up Table) 型 FPGA (Field-Programmable Gate Array) は LUT とプログラム可能な配線によって論理回路を構成する. LUT はメモリに真理値表を記憶することにより, 任意の論理を実現する論理ゲートとして利用できる. また, 加減算を効率よく実装するために専用のキャリーロジックが組み込まれているものもある. FPGA は, いくつかの LUT やキャリーロジックをまとめた "スライス" または "ロジックブロック" と呼ばれる回路単位で構成される.

本稿では図 1 に示すように LUT とキャリーロジックにより構成されるスライスのモデルを想定する. このような構成の FPGA としては Xilinx 7 シリーズが挙げられる [12]. 1 つのスライスには 4 つの 6 入力 2 出力 LUT と 4 ビットのキャリーロジック (CARRY4) が含まれるものとする. LUT の各出力は CARRY4 のキャリー生成信号 g_i と伝播信号 p_i に入力している. また, CARRY4 の co_3 ポートを隣のスライスの cin ポートに接続することにより長いキャリーチェーンを構成することができる.

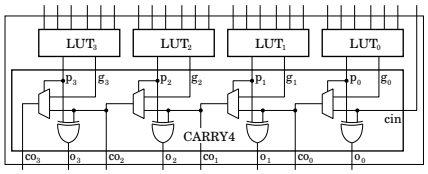


図 1: スライスのモデル

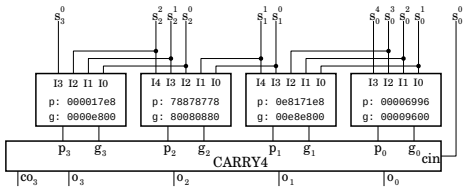


図 2: GPC (1,3,2,5;5) の 1 スライス実装 [8]

2.2 一般化並列カウンタ

多入力加算の効率的な回路構成法としては, 3 入力 2 出力の全加算器を基本素子として桁上げ保存加算器の木を構築する Wallce Tree [1] や Dadda Tree [2] が知られている. しかし, これらをそのまま FPGA 実装に適用しようとすると入力数が 3 より大きい LUT やキャリーロジックを有効活用できないため必ずしも効率的ではない. そこで, 入力数を増やし, さらに入力に 2 の冪乗の重みを許した拡張加算器である一般化並列カウンタ (Generalized Parallel Counter; GPC) を基本素子とする回路構成法が提案されている [3] [4] [5] [6] [7] [8] [9] [10].

GPC の入出力仕様は $(p_{k-1}, p_{k-2}, \ldots, p_0; q)$ のように表される. この回路は 2^i の重みを持つ p_i 個のビットの総和を q ビットで出力する. 例えば, 全加算器は重み 1 の入力 3 つの和を 2 ビットで出力するので, GPC (3;2) と表現できる. GPC (1,3,2,5;5) は重み 8, 4, 2, 1 の入力をそれぞれ 1, 3, 2, 5 個持ち, その重み和を 5 ビットで出力する.

これまでに, 図 1 のモデルを対象に, 1 スライスで実装可能な GPC が提案されている. スライス内部では, 信号が直接配線されているため遅延が小さく, また, GPC 構成を 1 スライスで固定することにより, 特定の GPC が遅延のボトルネックになることを防げる. 図 2 に GPC (1,3,2,5;5) の 1 スライス実装例を示す. 1 スライスで実装できる GPC の基本型は 8 種類知られており [8] [9] [10], そこから入出力を省略/結合した約 70 種類が多入力加算器の基本素子として使用される.

2.3 コンプレッサツリー

GPC を基本素子とした多入力加算器の構成例を図 3 に示す. ここでは n 個の m ビット 2 進数の加算を行っており, 丸は 1 つのビットを表わす. コンプレッサツリー (Compressor Tree) は GPC により構成される桁上げ保存加算器の木であり, 各段のビットは GPC に入力されて加算され, GPC は破線で区切られた次の段に部分和を出力する. コンプレッサツリーによって入力を 2 つの値になるまで加算した後, キャリーチェーンを用いた行加算器 (Row Adder) で足し合わせるのが一般的である [7] [8] [9].

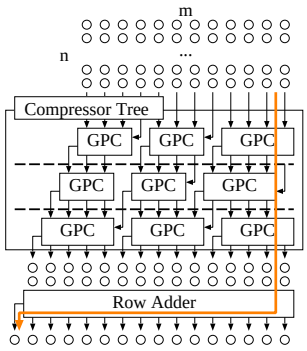


図 3: コンプレッサツリーの構造

表 1: 従来手法 [7] の定式化における変数 (上) と定数 (下)

$k_{s,c,e} \in \mathbb{N}_0$	段 s, 列 c で使用される GPC e の数
$N_{s,c} \in \mathbb{N}_0$	段 s, 列 c のビット数
$S \in \mathbb{N}_0$	コンプレッサツリーの段数
$C \in \mathbb{N}_0$	コンプレッサツリーの列数
E	使用可能な GPC の集合
$c_e \in \mathbb{R}$	GPC e のコスト
$M_{e,c} \in \mathbb{N}_0$	GPC e の下から c 列目の入力ビット数
$q_e \in \mathbb{N}_0$	GPC e の出力ビット幅
$K_{e,c} \in \mathbb{N}_0$	GPC e の下から c 列目の出力ビット数
$I_c \in \mathbb{N}_0$	コンプレッサツリーの入力の列 c のビット数
$L \in \mathbb{N}_0$	コンプレッサツリーの出力の最大行数

回路のクリティカルパスを 図 3 においてオレンジ色の矢印で表す. コンプレッサツリーによる 2 数までの加算には $O(\log n)$ 段の GPC が必要である. 行加算器の遅延は出力のビット数 $m + \log_2 n$ に比例する. 従ってこの回路のクリティカルパス遅延は $O(m + \log n)$ となる[1].

2.4 最適コンプレッサツリー構成問題

コンプレッサツリーの構造は, 全加算器の木に比べて複雑である. より小さい段数と回路規模のコンプレッサツリーを構成するために, ヒューリスティックアルゴリズムや整数線型計画問題への定式化が提案されている [7] [8] [9].

文献 [7] [9] では GPC の段数 S を固定して回路規模最小のコンプレッサツリー構成を求める最適化問題を定式化している. $S = 0, 1, 2, \ldots$ と段数を増やしながら求解することにより, 段数最小であって回路規模最小の回路構成を求めることができる.

この手法の定式化における変数と定数を表 1 に, 目的関数と制約を表 2 に示す. 目的関数は使用する GPC のリソースコストの総和を表し, コストの最小化を目指す. また制約は, 回路全体の入力 (C1) と出力 (C2) の制約, 各段の GPC の入力 (C3) と出力 (C4) の制約で構成される.

C1 はコンプレッサツリーの初段 ($s = 0$) の各列 c におけるビット数が入力の各列のビット数 I_c と等しいことを表す.

C2 はコンプレッサツリーの最終段 ($s = S - 1$) における各列 c のビット数が, 出力の最大行数 L 以下であることを保証する

(注1): 行加算器に桁上げ先見加算器を用いれば $O(\log m + \log n)$ にできるが, キャリーチェーンが高速であるため, $m = 256$ 程度までは桁上げ伝播加算器を用いるほうが高速になる.

表 2: 従来手法 [7] の定式化

目的関数	minimize $\sum_{s=0}^{S-1} \sum_{c=0}^{C-1} \sum_{e \in E} k_{s,c,e} \cdot c_e$
制約 C1	$N_{0,c} = I_c$ $\forall c \in \{0, 1, \ldots, C-1\}$
制約 C2	$N_{S-1,c} \leq L$ $\forall c \in \{0, 1, \ldots, C-1\}$
制約 C3	$\sum_{e \in E} \sum_{c'=0}^{q_e - 1} k_{s,c-c',e} \cdot M_{e,c'} \geq N_{s,c}$ $\forall s \in \{0, 1, \ldots, S-1\}, \forall c \in \{0, 1, \ldots, C-1\}$
制約 C4	$\sum_{e \in E} \sum_{c'=0}^{q_e - 1} k_{s,c-c',e} \cdot K_{e,c'} = N_{s+1,c}$ $\forall s \in \{0, 1, \ldots, S-2\}, \forall c \in \{0, 1, \ldots, C-1\}$

(2. 3 節で述べた通り, $L = 2$ とするのが一般的である).

C3 は各段の全てのビットが GPC に入力されて次段に伝播されることを制約する.

C4 は各段の GPC の出力ビット数が次段のビット数と等しいことを表し, 全ての GPC の出力ビットが次段に引き継がれることを保証する.

2.5 圧縮率の大きな GPC の活用

コンプレッサツリーにおいて, GPC はビット数を削減する役割を果たす. 例えば, 全加算器である GPC (3;2) は 3 ビットを入力として 2 ビットを出力する. 3 ビットが 2 ビットに「圧縮」されるので「圧縮率」は 2/3 と考えることができる. コンプレッサツリーの入力に対する出力の総ビット数は決まっているので, 圧縮率の大きな GPC の使用が GPC の総数, 即ち回路規模の削減につながる.

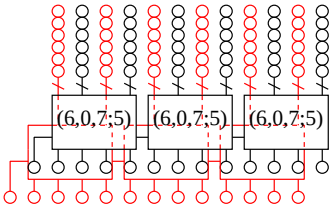
表 3 に示す 1 スライス実装可能な GPC の中で圧縮率が最も大きいのは, 13 入力 5 出力の GPC (6,0,7;5) であり, その次は 12 入力 5 出力となる. この GPC (6,0,7;5) を活用することが望ましいのだが, GPC (6,0,7;5) も含めて最下位が 7 入力の GPC は, 最下位の LUT と CARRY4 の cin ポートで合計 7 入力を実現する構造である. このため, これらの GPC が 1 スライス実装可能なのは cin ポートが隣のスライスの co_3 ポートから接続されている場合に限られ, それ以外の場合は 2 スライスを消費してしまう.

文献 [11] はこの点に着目し, GPC (6,0,7;5) をキャリーチェーンで接続した回路構成により, 文献 [7] の最適解よりもさらに回路規模の小さい多入力加算器を構成する手法を提案している. GPC (6,0,7;5) のチェーン状接続によって, 6 つの 2 進数を 2 つまで加算する回路である 6-2 adder を構成し, それを用いて加算木を構築している. 6-2 adder は図 4a に示すとおり, 偶数桁目と奇数桁目を加算できるように GPC (6,0,7;5) のチェーンを 2 つ配置した構造を持つ. さらに, 図 4b のように, 6-2 adder を多段化することにより, 任意の大きさの多入力加算器を構築できる.

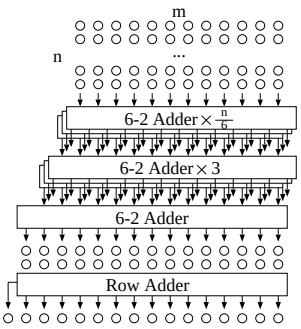
この手法では前節のコンプレッサツリーよりも回路規模の小さな多入力加算器を得ることができる. また, 回路全体の段数は増加するが, キャリーロジックのチェーンに LUT が含まれないため, 遅延は却って減少する. さらに, 6-2 adder の木は GPC の

表 3: Xilinx 7 シリーズで 1 スライス実装可能な GPC [8] [9] [10]

(3;2)	(7;3)	(1,5;3)	(2,3;3)	(3,1;3)	(4,4;4)	(6,3;4)	(7,1;4)
(1,1,7;4)	(1,2,6;4)	(1,3,5;4)	(1,4,3;4)	(1,5,1;4)	(2,0,7;4)	(2,1,5;4)	(2,2,3;4)
(2,3,1;4)	(3,0,3;4)	(3,1,1;4)	(4,2,5;5)	(4,3,3;5)	(4,4,1;5)	(6,0,6;5)	(6,0,7;5)
(6,1,5;5)	(6,2,3;5)	(6,3,1;5)	(7,0,3;5)	(7,1,1;5)	(1,1,6,3;5)	(1,1,7,1;5)	(1,2,4,4;5)
(1,2,5,3;5)	(1,2,6,1;5)	(1,3,1,6;5)	(1,3,2,5;5)	(1,3,3,4;5)	(1,3,4,3;5)	(1,3,5,1;5)	(1,4,0,6;5)
(1,4,0,7;5)	(1,4,1,5;5)	(1,4,2,3;5)	(1,4,3,1;5)	(1,5,0,3;5)	(1,5,1,1;5)	(2,0,4,4;5)	(2,0,6,3;5)
(2,0,7,1;5)	(2,1,1,6;5)	(2,1,1,7;5)	(2,1,2,6;5)	(2,1,3,5;5)	(2,1,4,3;5)	(2,1,5,1;5)	(2,2,0,6;5)
(2,2,0,7;5)	(2,2,1,5;5)	(2,2,2,3;5)	(2,2,3,1;5)	(2,3,0,3;5)	(2,3,1,1;5)	(3,0,0,6;5)	(3,0,0,7;5)
(3,0,1,5;5)	(3,0,2,3;5)	(3,0,3,1;5)	(3,1,0,3;5)	(3,1,1,1;5)			



(a) 6-2 adder の構造



(b) 6-2 adder の木の構造

図 4: 6-2 adder による多入力加算器

木よりも単純なため, 加算器の構築にあたって整数線型計画問題を解く必要がない.

反面, 最適な回路を得られるのは複数の値の加算のような単純な入力形状に限られ, 乗算器のような複雑な形状で最適な回路を得るのは難しい.

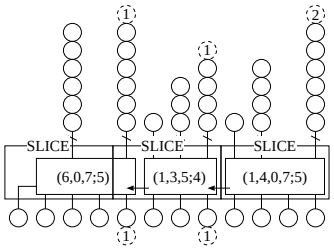
3. 一般化並列カウンタのチェーン状接続による多入力加算器

3.1 一般化並列カウンタのチェーン状接続

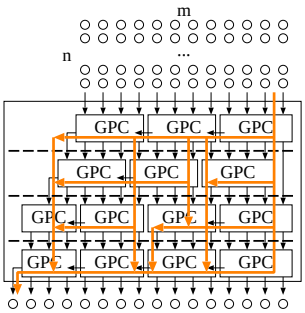
本稿では, 6-2 adder を一般化し, 複数種類の GPC をチェーン状接続したものをコンプレッサツリーに適用する. これにより, (6,0,7;5) 等最下位が 7 入力の GPC の効率的利用が可能になり, 回路規模の削減が期待できる. また, 乗算器を含む一般的な入力の形状に対しても最適な回路構成を求めることができる.

本手法において図 5a のように GPC をキャリーチェーン同士で接続したものを GPC チェーンと呼ぶ. GPC チェーンを用いた多入力加算器の構成を図 5b に示す. 本手法では, 従来のコンプレッサツリーにおける各段の GPC をキャリーチェーンで接続する. また, 各段は最上位から最下位までが一つの GPC チェーンである必要はなく, 途中で切れていることも許容する.

従来手法は図 3 に示すとおり GPC の桁上げ保存加算器の木で 2 数になるまで加算し, それを行加算器で加算して最終結果を得る. しかし, 本手法では, 行加算器を GPC (3;2) をキャリーチェーンで連結した GPC チェーンとみなし, これを含めた回路全体を最適化の対象に含める.



(a) GPC のチェーン状接続



(b) 多入力加算器の構造

図 5: GPC のチェーン状接続による多入力加算器

4 節で詳説する通り, 回路規模の減少は圧縮率の大きな GPC の活用, およびコンプレッサツリーの段数制約の緩和による. また, GPC チェーンによる加算木の遅延のオーダーは従来手法と同じく $O(m + \log n)$ である.

3.2 整数線型計画問題への定式化

従来のコンプレッサツリーと同様に, 本手法も整数線型計画問題として定式化する.

本手法の加算器は, 従来のコンプレッサツリーの各段[2]の GPC をチェーン状に接続した構造である. 従って, 従来のコンプレッサツリーから変数と定式化を引き継ぎ, チェーン状接続に関わる部分を追加, 変更する.

本手法では, 図 5a において破線で示すビットが使用できなくなるため, これらのビットの扱いを定式化に追加する. 1 で示すビットは GPC のチェーン状接続によって使用できなくなるビットであり, 次節 3.2.1 で扱う. また, GPC チェーンの最下位が 7 入力だった場合, 図 5a の 2 で示すように 1 入力削除することにより, 1 スライスで実装できるようにする. このビットに関する定式化は 3.2.2 節で述べる.

3.2.1 GPC チェーンの定式化

ある 2 つの GPC がチェーン状接続されているとき, 下位側の GPC の最上位ビットと, 上位側の最下位ビットの一つが接続されるため使用できなくなる. 図 5a の 1 で示すビットは GPC のチェーン状接続によって使用できなくなるものを表す.

このようなビットを扱うために追加, 変更する変数と制約を表 4 と 表 5 に示す. GPC の入出力に関する制約 C3, C4 を C3′, C4′ に変更する. また, GPC のチェーン接続数の上限の制約 C5, C6 を追加する.

(注2): ここで言う「段」は, 図 3 及び図 5b の破線で区切られたものを表す. 従来法では段の数と GPC の段数は一致するが, 提案手法では一致しない.

表 4: GPC チェーンを扱うために追加する変数

$w_{s,c} \in \mathbb{N}_0$	段 s, 列 c で次段にそのまま伝播する入力ビットの数
$t_{s,c} \in \mathbb{N}_0$	段 s, 列 c の GPC 下位からのチェーン接続を行う数

表 5: GPC チェーンを扱うために追加, 変更する制約

制約 C3′	$w_{s,c} - t_{s,c} + \sum_{e \in E} \sum_{c'=0}^{q_e-1} k_{s,c-c',e} \cdot M_{e,c'} \geq N_{s,c}$ $\forall s \in \{0,1,\ldots,S-1\}, \forall c \in \{0,1,\ldots,C-1\}$
制約 C4′	$w_{s,c} - t_{s,c} + \sum_{e \in E} \sum_{c'=0}^{q_e-1} k_{s,c-c',e} \cdot K_{e,c'} = N_{s+1,c}$ $\forall s \in \{0,1,\ldots,S-2\}, \forall c \in \{0,1,\ldots,C-1\}$
制約 C5	$\sum_{e \in E} k_{s,c,e} \geq t_{s,c}$ $\forall s \in \{0,1,\ldots,S-1\}, \forall c \in \{0,1,\ldots,C-1\}$
制約 C6	$\sum_{e \in E} k_{s,c-q_e,e} \geq t_{s,c}$ $\forall s \in \{0,1,\ldots,S-1\}, \forall c \in \{0,1,\ldots,C-1\}$

表 6: 最下位が 7 入力の GPC を扱うために追加する変数 (上) と定数 (下)

$r_{s,c} \in \mathbb{N}_0$	段 s, 列 c で GPC から削減するビット数
$E_7 = \{e \in E \mid M_{e,0} = 7 \wedge q_e = 5\}$	1 スライス実装不可能な GPC の集合

$w_{s,c}$ は次段にそのまま伝播するビット数を表す. 従来手法では, このようなビットを GPC (1;1) によって表現していたが, GPC (1;1) はチェーンにすることができないため, 本手法では専用の変数を使用する. また, 段 s 列 c で接続される GPC の数を $t_{s,c}$ で表す. チェーン状接続によって各段の入出力から削除されるビット数は $t_{s,c}$ と等しい.

C3′, C4′ は従来の制約 C3, C4 に, そのまま伝播するビットの数 $w_{s,c}$ と, チェーン状接続によって削除されるビット数 $t_{s,c}$ の項を追加したものである.

C5, C6 は段 s 列 c で接続される GPC の数 $t_{s,c}$ の上限を表す制約である. 上位と下位に GPC が存在しなければチェーン接続は行えないため, チェーン接続の数は, 上位側 (C5), 下位側 (C6) の GPC の数を超えない.

3.2.2 GPC チェーンの最下位が 7 入力の場合の扱い

前節までの定式化では, GPC チェーンの最下位が 7 入力である場合は 2 スライスを消費してしまう. そこで, 最下位が 7 入力の GPC が, GPC チェーンの最下位に配置される場合, 即ち下位側の GPC からキャリーを受け取らない場合は 1 入力削減して 6 入力として扱うことにより, この問題を解決する.

なお, 最下位が 7 入力でも 1 スライスで実装可能な GPC はそのまま 7 入力で使用する. 例えば, GPC (7;3) は, 1 スライス実装可能な (7,1;4) の最下位桁の入出力を削除することにより同じコストで実現できるため, 入力を削減する必要はない.

表 6 に追加する変数と定数, 表 7 に追加, 変更する制約を示す. 本手法では, GPC のチェーン接続と, 最下位 7 入力の GPC により GPC の入力数が削減される. これら 2 つによって削減されるビット数をまとめて $r_{s,c}$ とする. C3′ の削減するビットの項 $t_{s,c}$ を $r_{s,c}$ に変更した制約が C3″ であり, $r_{s,c}$ に対する制約が C7, C8 である.

表 7: 最下位が 7 入力の GPC を扱うために追加, 変更する制約

制約 C3″	$w_{s,c} - r_{s,c} + \sum_{e \in E} \sum_{c'=0}^{q_e-1} k_{s,c-c',e} \cdot M_{e,c'} \geq N_{s,c}$ $\forall s \in \{0,1,\ldots,S-1\}, \forall c \in \{0,1,\ldots,C-1\}$
制約 C7	$t_{s,c} \leq r_{s,c}$ $\forall s \in \{0,1,\ldots,S-1\}, \forall c \in \{0,1,\ldots,C-1\}$
制約 C8	$\sum_{e \in E_7} k_{s,c,e} \leq r_{s,c}$ $\forall s \in \{0,1,\ldots,S-1\}, \forall c \in \{0,1,\ldots,C-1\}$

制約 C7, C8 は, 最下位が 7 入力の GPC を優先的にチェーン接続し, 接続できないものから 1 入力ずつ削減するための制約である. 本来, 削減数は $r_{s,c} = max(t_{s,c}, \sum_{e \in E_7} k_{s,c,e})$ と表されるが, この制約は線型ではないため, C7 と C8 に分割することにより, $r_{s,c}$ がこれを満たす最小値に制約される.

本手法は制約として C1, C2, C3″, C4′, C5, C6, C7, C8 を使用する.

4. 実験結果

提案手法に基づき, 多入力加算器を実装する実験を行った. 実験では $n = 8 \sim 32$ について, n ビットの乗算器と, n 個の n ビットの 2 進整数を加算する回路を構築した[3].

表 3 に使用した GPC の集合を示す [8] [9] [10][4]. 提案手法においては全ての GPC のコストは 1 スライスとした. 従来手法においては, (6,0,7;5), (1,4,0,7;5), (2,1,1,7;5), (2,2,0,7;5), (3,0,0,7;5) は 2 スライスのコストとし, それ以外を 1 スライスとした. また, 提案手法においてこれらは 1 スライスで実装できるので, これらの最下位から 1 入力減らした GPC は提案手法では使用しない.

整数線型計画ソルバーには CPLEX 22.1.0.0 を用い, Ryzen 9 3900X 上で 7200 秒を制限時間とした. 解に基づく多入力加算器を Verilog HDL で設計し, Vivado 2023.2 で Xilinx Artix-7 (xc7a100tcsg324-3) をターゲットに論理合成と配置配線を行った.

図 6a と図 6b に n ビットの乗算器と n 個の n ビットの 2 進整数の加算器の回路規模を示す. いずれも, 横軸が入力ビット数の合計 ($= n^2$), 縦軸がスライス数 (回路規模) である. 従来手法は, 文献 [7] の定式化で求められる最適コンプレッサツリーに基づく図 3 の構成の回路である. 提案手法により, 乗算器と多入力加算器の回路規模をそれぞれ平均 13.06%, 11.08% 削減できた.

図 6c と図 6d に n ビットの乗算器と n 個の n ビットの 2 進整数の加算器の遅延を示す. 横軸が入力ビット数の合計 ($= n^2$), 縦軸が遅延時間 (ns) である. 提案手法に基づく加算器の遅延時間は, 従来手法とほぼ同じである.

使用した GPC 集合の中では 13 入力 5 出力の GPC (6,0,7;5) が最大の圧縮率を持つ. 従来手法において 1 スライスに実装で

(注3)：求解プログラム: https://github.com/ishiuralab/pycmpgen

(注4)：GPC 生成プログラム: https://github.com/ishiuralab/advgpcgen-rs

きる圧縮率最大の GPC は 12 入力 5 出力の (6,0,6;5), (6,1,5;5) であり, GPC (6,0,7;5) の圧縮率は約 8.3% 大きい. 回路規模の削減は主にこの圧縮率向上の効果であると考えられる.

また, 回路規模削減は, 従来手法において GPC の段数制限を緩和して規模のより小さい回路を求めていることにもよると考えられる. GPC チェーンで接続された GPC は従来手法では別の段として扱われる. 即ち, 提案手法は従来手法よりも大きな段数で規模が最小となる回路構成を探索しているために, 従来手法より回路規模の小さい解を発見できている可能性がある.

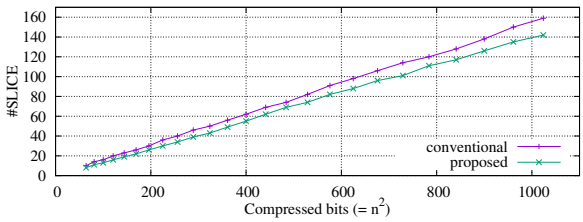
ただし, 提案手法では, GPC チェーン内の各 GPC の最上位ビットは常にキャリー入力に接続されるため, クリティカルパス上の GPC の段数が増えても LUT の段数は増加しない. 従来法と同程度の遅延となっているのはこれが原因と考えられる.

提案手法において, m ビットの 2 進整数を n 個加算する回路の遅延時間のオーダーは, 従来のコンプレッサツリーと同等である. 回路全体における最長の信号パスは図 5b においてオレンジ色の矢印で示している. 従来のコンプレッサツリーと異なり, 信号は GPC チェーン内で段の中を伝播しうる. しかし, GPC チェーンを使用することにより前段や下位の列に伝播することはないため, 右上から左下へ至るパスは同じ長さである. これにより, 遅延時間のオーダーは従来のコンプレッサツリーと同じく $O(m + \log n)$ である.

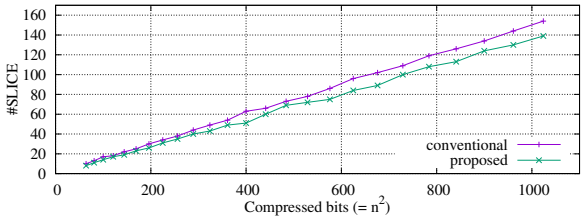
5. むすび

本稿では, チェーン状に接続した GPC による多入力加算器の効率的な FPGA 実装手法を提案した. GPC をキャリーチェーンを用いて接続し, それを多段化することにより, 任意の入力形状の加算器に対応可能とし, それを整数線型計画問題に定式化した. 提案手法により, 従来のコンプレッサツリーと同程度の遅延時間を維持しつつ, 乗算器と多入力加算器において回路規模をそれぞれ平均 13.06 %, 11.08 % 削減した回路が得られた.

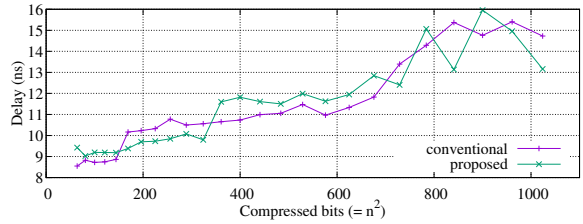
今後の課題としては, 整数線型計画ソルバーを使用しない加算器の構築手法の開発が挙げられる.



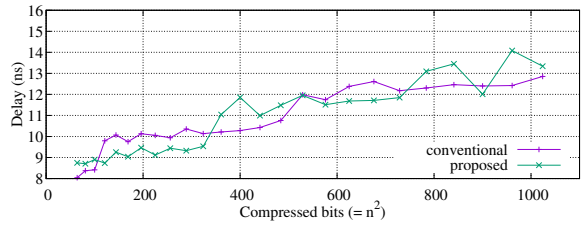
(a) n ビット乗算器のスライス数



(b) n 個の n ビットの 2 進整数の加算器のスライス数



(c) n ビット乗算器の遅延時間



(d) n 個の n ビットの 2 進整数の加算器の遅延時間

図 6: 実験結果

謝　辞

本研究に関して有益な御助言を頂いた京都高度技術研究所の神原弘之氏, 立命館大学の冨山宏之教授, 元立命館大学の中谷嵩之氏に感謝いたします. また, 本研究に関してご協力, ご討議頂いた関西学院大学の叶亮氏はじめ石浦研究室の諸氏に感謝いたします.

文　献

[1] S. C. Wallace: “A Suggestion for a Fast Multiplier,” in *IEEE Trans. on Electronic Computers*, vol. EC-13, no. 1, pp. 14–17 (Feb. 1964).

[2] L. Dadda: “Some Schemes for Parallel Multipliers,” in *Alta Frequenza*, vol. 34, pp. 349–356 (May 1965).

[3] T. Matsunaga, S. Kimura, and Y. Matsunaga: “Multi-Operand Adder Synthesis on FPGAs Using Generalized Parallel Counters,” in *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC 2010)*, pp. 337–342 (Feb. 2010).

[4] B. Khurshid and R. N. Mir: “High Efficiency Generalized Parallel Counters for Xilinx FPGAs,” in *Proc. International Conference on High Performance Computing (HiPC 2015)*, pp. 40–46 (Dec. 2015).

[5] M. Kumm and P. Zipf: “Efficient High Speed Compression Trees on Xilinx FPGAs,” in *Proc. Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV 2014)*, pp. 1–12 (Jan. 2014).

[6] T. B. Preußer: “Generic and Universal Parallel Matrix Summation with a Flexible Compression Goal for Xilinx FPGAs,” in *Proc. International Conference on Field Programmable Logic and Application (FPL 2017)*, pp. 1–7 (Sept. 2017).

[7] M. Kumm and J. Kappauf: “Advanced Compressor Tree Synthesis for FPGAs,” in *IEEE Trans. on Computers*, vol. 67, no. 8, pp. 1078–1091 (Jan. 2018).

[8] M. Kumm and P. Zipf: “Pipelined Compressor Tree Optimization using Integer Linear Programming,” in *Proc. International Conference on Field Programmable Logic and Applications (FPL 2014)*, pp. 1–8 (Sept. 2014).

[9] Y. Yuan, L. Tu, K. Huang, X. Zhang, T. Zhang, D. Chen, and Z. Wang: “Area Optimized Synthesis of Compressor Trees on Xilinx FPGAs Using Generalized Parallel Counters,” *IEEE Access*, vol. 7, pp. 134815–134827 (Sept. 2019).

[10] M. Noda, and N. Ishiura: “Enumeration of Generalized Parallel Counters for Multi-Input Adder Synthesis for FPGAs,” in *Proc. Asia Pacific Conference on Circuits and Systems (APCCAS 2024)*, pp. 64–68 (Nov. 2024).

[11] 野田麦, 叶亮, 石浦菜岐佐: “一般化並列カウンタ (6,0,7;5) による多入力加算器の効率的 FPGA 実装,” 電子情報通信学会ソサイエティ大会, A-6-2 (Sept. 2024).

[12] Xilinx, Inc.: 7 Series FPGAs Configurable Logic Block User Guide (UG474) (Sept. 2016), https://docs.amd.com/v/u/en-US/ug474_7Series_CLB (accessed in Dec. 2024).