Vol.2011-SLDM-148 No.16
2011/1/17

キャリーチェインを用いたマルチオペランド加算器のFPGA向け低電力合成手法

松永　多苗子[†1]　木村晋二[†2]　松永裕介[†3]

FPGA 上で多入力加算を実行する場合，通常のキャリーチェインを用いた加算木構成よりも LUT 上に桁上げを伝播させない加算を実現してそれを用いる手法の方が高速な回路が得られることが最近になってわかってきた．その一方で，LUT ベースの構成は加算木構成よりも要素数が増大し，消費電力が大きくなる傾向が観測されている．本稿では，一部にキャリーチェインを用いた構成要素を使用することによって，性能の劣化を抑えながら消費電力を削減する手法を提案する．

Low power synthesis of multi-operand adders using carry-chain structures on FPGAs

Taeko Matsunaga,[†1] Shinji Kiura[†2]
and Yusuke Matsunaga[†3]

Recent researches suggest that multi-operand adders can be realized on LUT-based FPGAs efficiently. Faster circuits than adder trees can be obtained by those approaches, while the number of components would increase, which would cause the increase of power dissipation. This paper proposes an approach for power-aware synthesis of multi-operand adders at the higher speed by partially using carry-chain structures on FPGAs.

†1 早稲田大学 IT 研究機構
Information Technology Research Organization, Waseda University
†2 早稲田大学大学院情報生産システム研究科
Graduate School of Information, Production and Systems, Waseda University
†3 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electorical Engineering, Graduate School of Kyushu University

1. はじめに

3 個以上の入力データの総和を計算するマルチオペランド加算は，乗算や積和演算等，より複雑な様々な演算を実現する上で必要とされる基礎的な演算である．高速なマルチオペランド加算を ASIC で実現する場合，まず桁上げを伝播させずにオペランド数を 2 個にまで圧縮し，桁上げの伝播は最後の 1 回だけ実行する方式をとることが一般的に行われている．オペランド数の圧縮回路は全加算器および半加算器を基本要素として，Wallace tree や Dadda tree 等の構成により実現することが可能である[1)–4)]．

一方，対象が FPGA である場合には，上記のような圧縮回路を LUT を用いて実現するよりは，高速桁上げ構造や DSP 等の演算モジュールを使用するなど，対象としている FPGA デバイスに特有の機能や構成を利用する方が高品質な結果が得られると考えられていた．しかし，FPGA を構成する LUT の入力数が増加するにつれ，全加算器よりも入力数の大きなカウンタを用いて圧縮回路を構成することにより，LUT 上により効果的に圧縮回路を実現できる可能性が示唆された[5)]．

Parandeh-Afshar らは,「一般化されたカウンタ」(Generalized Parallel Counter, GPC) を用いて高速な圧縮回路を構成するヒューリスティック[5)]，および，ILP を用いた段数最小化手法[6)] を提案し，FPGA に備わった高速桁上げ機能を用いて加算木で実現するよりも高速な回路の実現が可能であることを提示した．筆者らも 7) において，一定の段数の下で使用する GPC の個数を抑制することによって間接的に配線遅延を削減し高速化を図る手法を提案した．これらの回路を Altera 社の Stratix III 上で評価したところ，遅延時間に関して有意な減少が見られたが，消費電力に関しては増加する傾向が観測された．LUT ベースの手法は加算木で構成する場合に比べて使用する構成要素数が大きくなる傾向があり，それが消費電力の増加にも影響を与えていると考えられる．

本稿は，LUT ベースの圧縮回路実現手法において，特に速度の優位性を保ちつつ低電力化を実現する手法を検討するものである．圧縮回路で用いる要素数が電力に大きな影響を与えることに着目して，キャリーチェインを利用したより入力数の大きなカウンタを部分的に用いることによって、性能の劣化を抑えつつ電力を削減することを目指している．

以下，2 節でいくつかの定義を行った上で，3 節で圧縮回路を GPC を用いて実現する基本アルゴリズムを示す．4 節において，速度の劣化をおさえつつ電力消費を抑える手法について提案する．5 節で実験結果を示し，6 節でまとめを行う．

Vol.2011-SLDM-148 No.16
2011/1/17

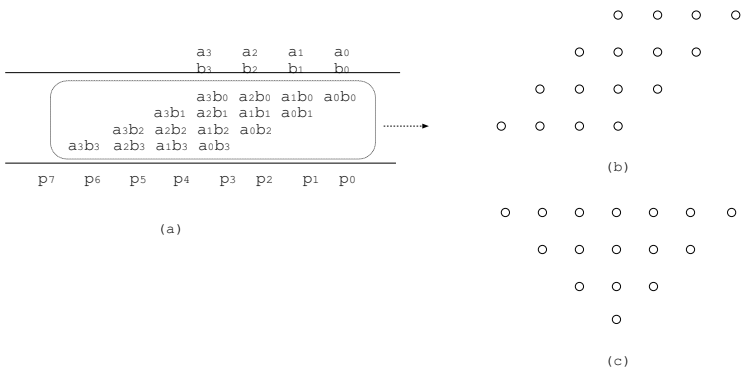

図 1 ドットダイアグラムの例
Fig. 1 Dot diagram

2. 準　　備

以下では，マルチオペランド加算器合成問題の入力となるオペランドは 1 ビットごとにドットで表し，加算すべきすべてのオペランドをドットのマトリックスの形で表現する．ここで，マトリックスの各列は対応するビット位置を表している．このマトリックスをドットダイアグラムと呼ぶ．図 1 は，4 ビット乗算器の部分積加算部分 (a) に対するドットダイアグラム (b) を示している．便宜上，各列は上辺（あるいは下辺）に整列させているものとする (c)．

2.1 並列カウンタ

$A = (a_{n-1}a_{k-2}\dots a_0)$ を n ビット符号なし 2 進整数とする．a の添字 i を，a_i のランクと呼ぶ．

定義 1 単一列に対する並列カウンタ $(m;n)$ とは，m 個のビットを入力し，1 の数を数えて，その個数を n ビットの符号なし整数として出力する組み合わせ回路である．出力は 0 から m の範囲の値であり，m と n の間には関係 $n = \lceil log_2(m+1) \rceil$ が成り立つ．

$(m;n)$ カウンタは，常に同じランク i の入力の加算を行い，出力のランクは $i, i+1, \dots, i+(n-1)$ となる．(3;2) カウンタは全加算器，(2;2) カウンタは半加算器と同等である．

定義 2 一般化された並列カウンタ（Generalized Parallel Counter,GPC）とは，異なるランクのビットを入力し，それらの和を計算して n ビット符号なし整数として出力する組み合わせ回路である．ランク i の入力の個数を m_i とすると，GPC は $(m_{k-1}, m_{k-2}, \dots, m_0; n), m_{k-1} > 0$ で表される．出力の値は 0 から M の範囲の値であり，M と n の間には以下の関係が成り立つ．

$$M = \sum_{i=0}^{k-1} m_i 2^i$$
$$n = \lceil \log_2(M+1) \rceil$$

入力の総数を $m = \sum_{i=0}^{k-1} m_i$ とすると，GPC の総入力数 m の総出力数 n に対する比

$$R = \frac{m}{n}$$

を，この GPC の削減率と呼ぶ．

オペランド数を圧縮するためのカウンタも，ドットを用いて表すことができる．図 2 に入力数 6 としたときの GPC 集合を示す．ここで，横線の上の部分が入力ドット，下の部分が出力ドットを表す．図中 ΔH は，各 GPC を適用した場合の，各ランクにおけるドット数の増減を表している．例えば，(1,5;3)GPC を用いると，ランク 0 ではドット数が 5 個減って出力分 1 個増えるので，トータルで 4 個減少することを表している．

2.2 対象とする問題

マルチオペランド加算における圧縮回路とは，3 個以上の入力 A_i の加算を行い，指定された個数 k 個のオペランド O_j の和として出力する回路である．O_j と A_i の間には以下の関係がある．

$$\sum_{j=0}^{k-1} O_j = \sum_{i=0}^{l-1} A_i.$$

GPC を構成要素とした圧縮回路を GPC ネットワークと呼ぶことにする．

GPC ネットワーク中の各 GPC は，その出力本数分の LUT で実現される．本稿が対象としているのは，このような GPC ネットワークを対象としている FPGA 上に高速，かつ低電力な回路として実現する手法を探索する問題である．

3. GPC ネットワーク合成の基本アルゴリズム

GPC ネットワーク合成は，Wallace 乗算器等と同様のフレームワーク，すなわち，GPC で加算すべきドット数を段階的に削除する操作を，最終的なドット数が指定された個数 k 以下になるまで繰り返すというものである．ASIC の場合 $k = 2$ である．FPGA を対象と

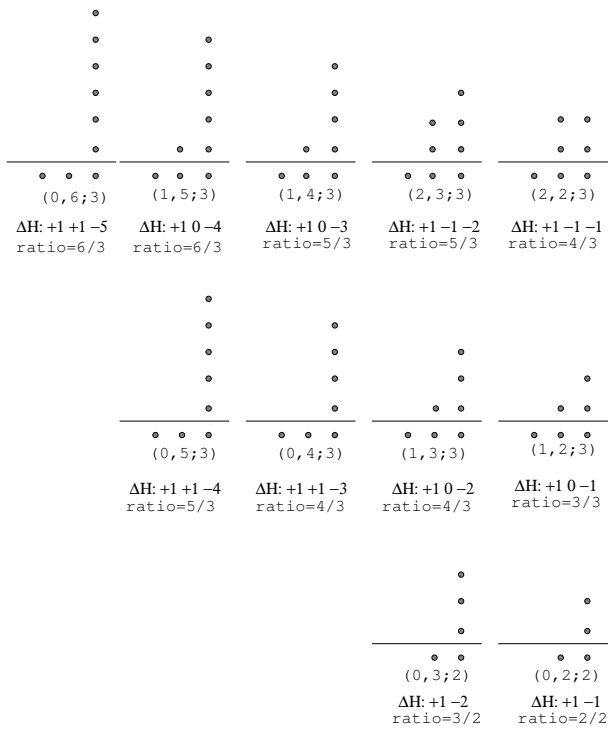

図 2 6 入力 GPC の種類
Fig. 2 6-input GPCs

する場合には，高速な 3 入力加算器構造が実現されることを仮定して，以下では $k = 3$ として扱う．

繰り返しカバーしていく方法としては，7) で示す手法を用いる．このフレームワークは，

乗算器における Dadda の手法の概念を取り込んだもので，段階的に一列あたりのドット数を削減していく中で，各ステージで許される列ごとのドット数を計算しておき，その制限を超えるものだけを GPC でカバーする，というものである．ドットダイアグラム (dot_dg)，GPC の入出力数 m，n，圧縮回路の最終オペランド数 k が与えられたとして，次の手順で GPC ネットワークを合成する：

- 前処理フェイズ:
  - 与えられたドットダイアグラムから，最大の高さ (max_h) と幅 ($width$) を抽出する．
  - 入出力が最大 m，n であるような GPC を列挙し，削減率に基づいて順序付けをおこなう．
  - 各ステージでの目標の高さを計算し，$limit$ に最初の目標値を設定する．
- 圧縮フェイズ: すべての列のドット数が k 以下になるま次のプロセスを繰り返す：
  - 配列 add_dots を初期化する．これは，当該ステージで生成されたドット数を保持するためのものである．
  - 高さ制限を超える最下位列から始めて，制限を超えるドット数を削減でき，かつ最大の削減率をもつ GPC を選択する．ここで制限を超えるドット数には，同じステージの下位ビットの処理において発生したドットも含める．
  - このステージで生成された GPC 構成を構築する．
  - GPC によりカバーされた状況を反映させて，ドットダイアグラムを更新する．
  - 目標の高さを更新して次のステージに進む．

各ステージで目標とする高さは，最も削減率が高い GPC，すなわち，入力数 m の単一列カウンタを用いて計算する．例えば 6 入力の場合，(0,6;3) カウンタの削減率は $6/3 = 2$ であるため，最終的なオペランド数 $k = 3$ とすると，各ステージの高さ目標値は，最終ステージから遡って，$3 \rightarrow 6 \rightarrow 12 \cdots$ となる．

4. 提 案 手 法

上記アルゴリズムは基本的に，対象とする FPGA が何入力関数を 1 個の LUT で実現できるかのみに依存しておりその他の機能は仮定していないが，評価のため，ターゲットを Altera 社の Stratix III として，キャリーチェインを利用した加算木構成との比較を行った．その結果，遅延時間が短縮することが確認できた半面，要素数，および，動的電力が大きくなることが確認された．圧縮回路を構成する要素数と動的電力の間には強い相関があることを確認され，要素数が増大したため電力も増大すると推測された．

低消費電力という観点から考えた場合，高速性が要求されないのであれば，加算木構成を用いることが優位であると考えることができる．しかし，加算木では必要な性能が満たせない場合，あるいは，電力，遅延積として考えた場合には，LUT ベースアプローチで低電力化が可能であれば，有効である可能性が考えられる．GPC ネットワークの要素数削減に関しては，FPGA のもつキャリーチェインの構造を利用して GPC を実現することによって，

そうでない場合に比べて要素数の削減が可能であることが示唆されている[10]．

本稿では，Stratix III のアーキテクチャを対象として，現行のアルゴリズムを改良するとともに，このアーキテクチャのもつキャリー構造を部分的に利用することで，速度低下を抑えつつ低消費電力化を図る手法を提案する．

基本アイデアは，圧縮回路生成のアルゴリズムの改良と，Stratix III のキャリーチェインの構造を利用した 7 入力カウンタの導入，の二点である．

4.1 圧縮アルゴリズムの改良

前節のアルゴリズムでは，対象としている列のドット数を減らすには，この列に対する ΔH が最も大きい単一列カウンタを頻繁に用いることになる．しかし，削減すべきドット数によっては，その列に対する ΔH が最大でなくても必要な個数は変わらない場合が存在する．すると、上位ランクのビットをカバーできる場合があり，それにより，ランクの高い列で必要とされる要素数が削減される可能性も出てくる．

今，目標としている高さを越えたドット数を h とする．単一列カウンタの最大入力数を m とすると，1 個あたりドット数を $m' = m - 1$ 個削減できる．このカウンタを用いてすべての超過分を削減するのに必要な最小の要素数を n' とすると，

$$m' \cdot (n' - 1) < h \leq m' \cdot n'$$

という関係が成り立つ．削減できる要素数が 1 個少ない GPC を用いた場合でも，$(m'-1)\cdot n'$ がこの範囲に入るならば，同じ個数で超過分をカバーできる．ここから，$n' < m'$ の場合には，単一列カウンタ以外を利用することでランクの上位ビットをより多くカバーできる可能性がでてくる．6 入力の GPC を考えた場合，$m' = 5$ であるため，必要な個数が 4 以下の場合，すなわち，超過分が 20 以下の場合に，その可能性がある．例えば，$\Delta H = 12$ の場合，必要な GPC 集合として，$\{(0,6;3),(0,6;3),(2,3;3)\}$, $\{(0,6;3),(1,5;3),(1,4;3)\}$, および $\{(1,5;3),(1,5;3),(1,5;3)\}$ が考えられるが，この場合 (0,6;3) を用いない最後の場合が最も上位ビットを削減できる組み合わせになっている．

本アルゴリズムでは，Stratix III に対象を絞っていることを前提として，入力数は 6（あるいは，7 入力カウンタを用いる場合は 7）とした場合の可能性を列挙し，その中で最適な組み合わせを選択するようにしている．

4.2 7 入力カウンタの利用

本稿が対象としている Altera 社の Stratix III[9] は，アダプティック・ロジック・モジュール (ALM) として知られる基本ブロックを用いて構成される．ALM にはいくつかの動作モードがあり，通常モードでは，最大 6 入力の任意の関数が実現できる．上述の実験においては，この通常モードを用いて GPC を実現している．

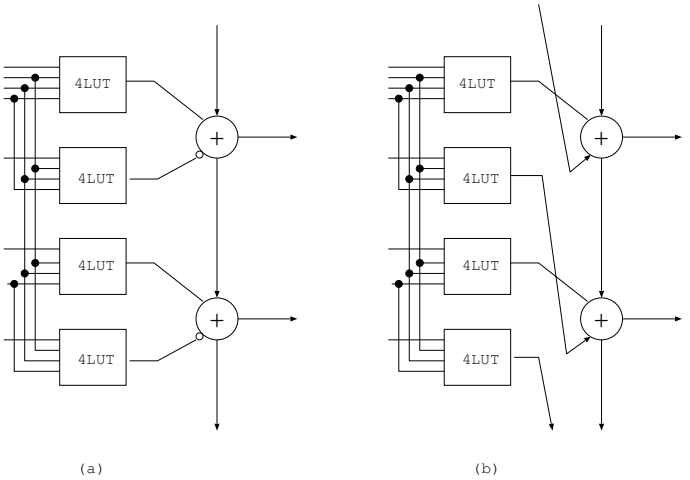

図 3 演算モード (a) と共有演算モード (b)
Fig. 3 Arithmetic mode(a) and shared arithmetic mode(b)

一方，ALM にはキャリーチェインを利用する演算モードと共有演算モードが存在する（図 3）．加算木による構成や，最終段の桁上げ伝播加算器はこれらの構造を用いて実現されている．

通常モードでは 6 入力までの GPC しか実現できないが，共有演算モードを用いると，(0,7;3) カウンタを実現することができる（図 4）．

演算モード，共有演算モードを用いた GPC の実現は 10)，11) で提案されている．通常モードで実現してきたこれまでの GPC を演算モードで実現することも可能である．ただし，演算モードではキャリーチェインを利用するため，通常モードに比べて遅延時間が長くなる．到達時刻の早い信号を遅いパスの入力に接続することによって遅延の増加を軽減できる可能性もあるが，遅延時間は配置配線の影響が大きく，GPC ネットワークのレベルで厳密に見積もることが難しい．一方で，7 入力カウンタを実現できれば，(0,6;3) よりも削減率が高いため，有効に要素数を削減でき，最終的に 3 段にするまでのステージ数も少なくすることが可能な場合がある．その結果，GPC の段数が小さくなれば，遅延時間増加による不利益を軽減できる可能性がある．

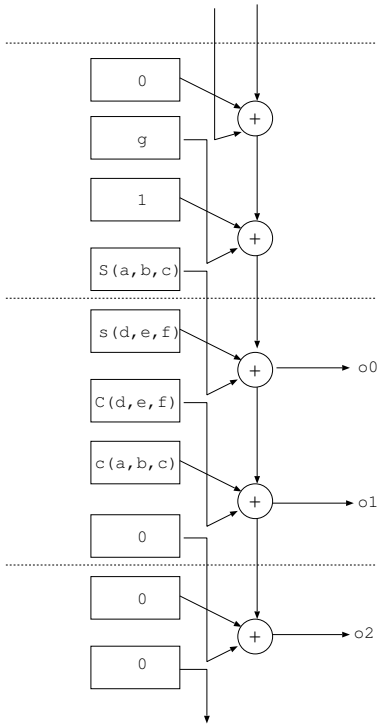

図 4 演算モードを用いた (0,7;3) カウンタの構造

そこで，本手法では，ステージ数に差が出る場合のみに 7 入力カウンタをベースとして適用する方針をとる．例えば，(0,7;3)GPC を基準にした場合の各ステージの高さ制限は，最終ステージから逆にみて，3 → 7 → 15 → 35 ··· となる．したがって，例えば全ドット数が 15 の場合，(0,7;3) を用いるならば 15 → 7 → 3 となり，(0,6;3) を用いた 15 → 12 → 6 → 3 よりもステージ数を減らすことが可能である．

この場合も，上記と同様，7 入力カウンタでも 6 入力カウンタと同じ個数が必要となる場合には，6 入力カウンタを用いることで，段数や要素数の削減に寄与しないと思われる 7 入力カウンタ使用を避けることで，遅延時間の増加を軽減するものとする．

5. 実　　験

提案手法を C++で実装し遅延，ALM 数，動的電力の比較実験を行った．各 GPC を構成する StratixIII のロジックセルを vqm 形式で表し，それらから構成される全体の回路を verilog HDL で記述されたネットリストとして生成する．CPA 部分は，加算演算のまま残しておき自動合成しており，結果的にキャリーチェインを用いた構成として実現されている．これらの入力記述から，Altera 社の Quartus II 10.0sp1[8] を用いて Stratix-III 上にマルチオペランド加算器が実現される．動的消費電力は PowePlay Power Analyzer のレポートから抽出した．

実験においては，以下の５つの手法を比較した．手法４が今回の提案手法に該当する．

(1) ADD: 自動生成された加算木構造
(2) 手法１: 基本アルゴリズム．各 GPC は通常モードの ALM を用いて実現する．
(3) 手法２：手法１に対して，前節で述べた要素数削減の機能を加えた手法.
(4) 手法３：共有演算モードで実現した GPC07 を構成要素として用いる手法．ステージ数削減のチェックはせず，基本的に 7 入力カウンタを基本として削減する．
(5) 手法４：手法３に対して，前節で述べたようにステージ数が削減できる時のみに限定した手法.

表 1 において，3 から 6 列目までは，ADD の結果である．遅延，電力，ALM 数はそれぞれレポートファイルから抽出し，DP は遅延時間と動的電力を単純にかけた値を示している．7 から 10 列，11 から 14 列，15 から 18 列，19 から 22 列はそれぞれ，手法 1 から 4 に対する結果をしめす．ただし，これらの値は ADD の値との比で表しているため，1 以上であれば増加，1 未満であれば減少したことを示している．最後の行は，それぞれの手法における対 ADD の比率の平均値を示している．表から分かるように，圧縮アルゴリズムの改良によって手法２は手法１よりも遅延，要素数，電力ともに小さくなっている．また，単純に 7 入力カウンタを適用すると手法３のように消費電力の増分は削減できるが遅延時間の優位性が減ってしまい，DP 値でみるともとの方法より悪くなるが，適用する範囲を限定することによって，手法４で示すように，遅延時間の増加を軽減でき，DP 積としては ADD も含めた全手法の中で最も良い結果が得られた．

実行時間は Pentium(R) 4 2.4GHz FreeBSD 6.2-RELEASE で，どれも１秒以下であった．

表 1 実験結果
Table 1 Experimental results

		ADD				手法 1(/ADD)				手法 2(/ADD)				手法 3(/ADD)				手法 4(/ADD)			
bit	num	遅延	電力	ALM 数	DP 積	遅延	電力	ALM 数	DP	遅延	電力	ALM 数	DP	遅延	電力	ALM 数	DP	遅延	電力	ALM 数	DP
16	10	3.38	30.00	94	101.37	0.88	1.23	1.23	1.09	0.81	1.20	1.21	0.97	1.01	1.22	1.11	1.23	0.82	1.19	1.22	0.97
16	14	3.59	44.13	137	158.43	0.99	1.27	1.23	1.25	1.05	1.22	1.24	1.28	0.99	1.08	1.20	1.07	0.99	1.08	1.20	1.07
16	15	3.77	47.26	145	177.98	0.89	1.25	1.36	1.12	0.88	1.23	1.34	1.09	0.99	1.03	1.24	1.02	0.99	1.03	1.24	1.02
16	16	4.65	48.76	155	226.78	0.79	1.31	1.35	1.03	0.72	1.23	1.34	0.89	0.89	1.11	1.25	0.98	0.71	1.26	1.34	0.90
16	24	4.51	73.89	241	333.47	0.86	1.28	1.41	1.10	0.86	1.30	1.41	1.12	1.01	1.20	1.24	1.22	0.79	1.26	1.41	1.00
16	25	6.53	87.40	251	570.63	0.66	1.19	1.40	0.79	0.61	1.12	1.39	0.69	0.74	1.05	1.24	0.78	0.74	1.05	1.24	0.78
24	10	3.59	45.27	138	162.47	0.92	1.16	1.28	1.06	0.85	1.12	1.25	0.95	1.23	1.27	1.13	1.57	0.93	1.19	1.24	1.11
24	14	4.23	67.16	201	283.75	0.97	1.23	1.30	1.19	0.91	1.25	1.29	1.14	0.96	1.05	1.23	1.02	0.96	1.05	1.23	1.02
24	15	4.01	70.42	213	282.17	0.91	1.21	1.39	1.10	0.89	1.23	1.38	1.10	1.03	1.02	1.28	1.05	1.03	1.02	1.28	1.05
24	16	5.13	84.15	227	431.86	0.74	1.16	1.40	0.85	0.71	1.12	1.37	0.80	0.83	0.97	1.29	0.80	0.74	1.11	1.37	0.82
						0.86	1.23	1.34	1.06	0.83	1.20	1.32	1.00	0.97	1.10	1.22	1.07	0.87	1.12	1.28	0.97

6. おわりに

本稿では，キャリーチェインを利用した 7 入力カウンタを部分的に利用することにより，速度低下を抑えつつ消費電力を下げる手法を提案し，今回の例題に対しては一定の効果が確認できた．一つ注意点としては，電力や速度に大きな影響を与える配置配線処理を今回の実験では自動で行っていることがあげられる．より正確な評価のためには，配置配線による影響の考慮も必要であると考えられ，今後の検討課題である．

7. 謝　辞

本研究は一部，JST CREST ULP プロジェクトの支援による．

参 考 文 献

1) C.Wallace, “A suggestion for a fast multiplier,” IEE Transactions on Electronic Computers, vol.EC-13, pp.14–17, 1964.
2) L.Dadda, “Some schemes for parallel multipliers,” Alta Frequenza, vol.34, pp.349–356, 1965.
3) V.G. Oklobdzija and D.Villeger, “Improving multiplier design by using improved column compression tree and optimized final adder in cmos technology,” IEEE Transactions on VLSI Systems, vol.3, no.2, 1995.
4) P.Stelling, C.Martel, V.G. Oklobdzija, and R.Ravi, “Optimal circuits for parallel multipliers,” IEEE Transaction on Computers, vol.47, no.3, pp.273–285, 1998.
5) H.Parandeh-Afshar, P.Brisk, and P.Ienne., “Efficient synthesis of compressor trees on fpgas,” ASPDAC, 2008.
6) H. Parandeh-Afshar, P. Brisk, and P. Ienne, “Improving synthesis of compressor trees on FPGAs via integer linear programming,” DATE, 2008.
7) T. Matsunaga, S. Kimura, and Y. Matsunaga, “Multi-operand adder synthesis on fpgas using generalized parallel counters,” Proc. Asia and South Pacific Design Automation Conference, pp.337–342, January2010.
8) Altera Corp., “Quartus II Development Software Handbook.”
9) Altera Corp., “The Stratix III Device Handbook.”
10) H. Parandeh-Afshar, P. Brisk, and P. Ienne, “Exploiting fast carry-chains of fpgas for designing compressor trees,” the 19th International Conference on Field-Programmable Logic and Application, pp.242–249, August2009.
11) H.Parandeh-Afshar, A.Neogy, P.Brisk, and P.Ienne, “Improved Synthesis of Compressor Trees on FPGAs by a Hybrid and Systematic Design Approach,” IWLS 2010, pp.193–200, June2010.