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

2008－SLDM－136（11）
2008 / 10 / 7

FPGAを対象とした部分積加算回路の合成について

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 本稿では、FPGAを対象として、並列乗算器の部分積加算回路を、一般化したカウンタを用いて合成する手法について述べる。ライブラリセルを用いて実現する場合、カウンタの規模が大きくなると、その面積や遅延の特性も大きくなり、大規模カウンタを用いる効果は単純には判断できない。しかし、k入力のLUTから構成されるFPGAを対象とした場合、カウンタの入力がk以下であれば、同じコストで実現できるため、適切なカウンタを組み合わせて部分回路を構成することによって高速化、小面積化が期待できる。提案手法は、Dadda Treeの概念を一般化したカウンタに適用したもので、実験結果により、既存手法より10%程度面積が削減できることが確認された。
キーワード 演算器合成、乗算器、部分積加算、generalized parallel counter、FPGA

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 In this paper, an approach for counter tree synthesis targeting FPGA is addressed. In case of FPGA with k-input LUT, any generalized parallel counter with k inputs can be mapped into one LUT per an output. This approach utilizes Dadda's method, and experimental results show its effectiveness against existing approaches.
Key words arithmetic synthesis, multiplier, generalized parallel counter, FPGA

1. はじめに

乗算器は加算器と並んで非常に基本的な演算回路であり、多くの方式の研究がなされてきている[1]。本稿では、並列乗算器の部分積生成回路を、カウンタによる木状の構造で実現する場合を対象とし、テクノロジに依存しないレベルでの概略構造として、カウンタ数が少なく段数も小さい構造を生成する問題に取り組む。

Wallace tree[2]やDadda tree[3]等は(3,2),(2,2)カウンタを要素とした木構造として実現される。このカウンタは(3,2)に限定されない。より大きいカウンタを用いれば、段数は削減される。しかし、これらのカウンタをライブラリセルを用いて実現するとすると、入力数が大きくなるにつれ回路も複雑になり、1個あたりの面積／遅延は大きくなるため、全体として面積、遅延が優位になるかどうかは単純には判断できない。

一方、FPGAを仮定すると状況が変わってくる。(3,2)より大きなカウンタ使っても、1つのLUTで実現できる限りにおいては、各カウンタのコストは一定であると考えると、それだけ優位性が大きくなり得る。一般に、演算回路をFPGAで実現する際にはASICと同様な方法で合成するよりも、それぞれのアーキテクチャに特化された機能を使用した方が性能があがると考えられている。しかし、高性能なFPGAで使用可能な入力数の大きいLUTをもつFPGAの場合、より大きなカウンタを利用した部分積回路を生成することにより、特化された機能を利用するよりも優位な回路を生成できる可能性が示されている[4],[5]。

本稿では、FPGAで実現されることを想定した上で、カウンタ構成レベルでの高速、低面積化を目標とした部分積加算回路の生成手法として、Dadda Treeの概念をGPCに応用した手法を提案する。

以下、一般化されたカウンタ等いくつか定義を述べた上で、部分積加算回路の合成問題、および、FPGAを対象とした既存ヒューリスティックを示す。続いて、daddaの概念を取り入れた合成手法を提案し、実験を通して既存手法との比較を行う。

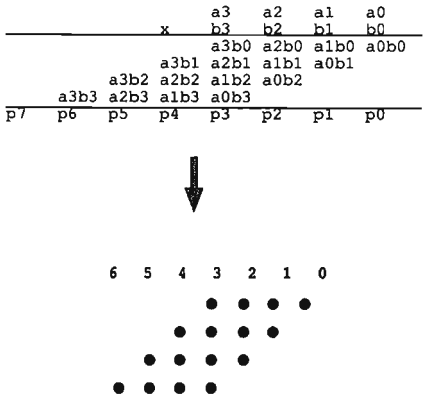

図 1 dot diagram

2. 準　　備

2.1 並列乗算器の構成

$A=(a_{n-1}a_{k-2}\ldots a_0)$ 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。

n ビット乗算回路は、与えられた入力 $A=a_{n-1},\cdots,a_0, B=b_{n-1},\cdots b_0$ に対して、各ビット対の論理積を足し合わせて、2n ビットの積を生成する。

部分積行列の各要素をドットで表したものを、ここでは dot diagram と呼ぶことにする。図 1 に dot diagram の例を示す。

2.2 カウンタの一般化

(m:n) カウンタとは、m 個のビットを入力し、1 の数を数えて、0 から m の範囲の値をもつ n ビットの符号なし整数として出力する。 m, n の間には以下の関係が成り立つ。

$$n=\lceil log_2(m+1)\rceil$$

(m:n) カウンタは常に同じランクのビット同士の加算を行う。出力のランクは、$i, i+1,\ldots,i+(n-1)$ となる。ハーフアダーは 2:2 カウンタ、フルアダーは 3:2 カウンタである。

上記のカウンタを一般化し、異なるランクのビットに対するカウントを可能にするものが、GPC(Generalized parallel counters) である。

[定義 1]　GPC は、各ランクの入力数と出力数によって、以下の記法で定義される。$(m_{k-1}, m_{k-2},\ldots,m_0; n), m_{k-1}>0$ ここで、m_i はランク i の入力数、n は出力数をである。GCP の出力は 0 から R の範囲の符号なし整数で、次の関係を満たしている。

$$R=\sum_{i=0}^{k-1} m_i 2^i$$

$$n=\lceil log_2(R+1)\rceil$$

例えば、$GPC(5,5;4)$ はランク 0 および 1 にそれぞれ 5 個の入力をもち、出力のビット幅は 4 である。すべての入力が 1 であるとき出力が最大になり、その値 R は、$R=5\cdot 2^1+5\cdot 2_0=15$ であるため、4 ビットの出力が要求される。

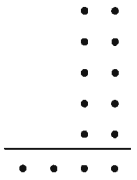

図 2 GPC(5,5;4) の dot diagram

3. GPCを用いた部分積加算回路合成問題

前述したように、大きなGPCを使うほど、部分積加算回路の段数は小さく実現できるが、1個あたりのGPCの実現コストが大きくなり、全体として面積、遅延にどのような影響が起こるかは、使用するライブラリセルやマッピングの手法によって異なってくる。一方、FPGAをターゲットとした場合には、GPCの入力数がLUTの入力数以下である限り、1出力あたり1個のLUTで実現することができ、大きなGPCを使うことによるメリットが高いと考えられる。

本稿では与えられたdot diagramに対して、GPCを用いて、指定した項の和を出力するようなカウンタ木を構成する。カウンタGPC(m,n)のm,nおよび桁上げ伝搬加算により和を求める項の数をkとして、これらは外から与えられるものとする。kは通常2であるが、高性能FPGAには、3項加算を高速に行う機能がついているものがある。例えば、Alteraの Stratix II/IIIでは、3項の加算回路が存在し[6]、それを生かすことを考えると、kは指定できるものとする。それ以外にFPGAの構成等に対する制限は与えていない。

文献[4]ではこの問題が提示されるとともに、1つのヒューリスティックが提案されている。この手法は遅延最小化を第1目標、面積を第2目標としたヒューリスティックで、処理フローは以下のようになっている：

- 与えられた入力数m、出力数nから、m入力n出力のGPCを列挙し、入力数から出力数への削減率を指標として優先順位をつける。
- 各ステージにおいて
  - 高さが最大の列を取り出して、候補GPCで最善のものを割り当てる
  - カバーされたdotを削除

を繰り返す

- 割り当てるものがなくなったら、次ステージ用のdot列生成
- これを、指定した高さになるまで繰り返す。

このヒューリスティックはかなり単純なものであり、遅延（段数）最小性も保証されていないが、FPGA上での評価により、FPGA特有の高速加算機能を使って多入力加算部分を実現した場合に比べて高速になり、本問題に取り組むことの可能性を示している。また、[5]においては、同問題に対していくつかの制約を加えてILPとして定式化して解くアプローチを示している。2つの手法と、多入力加算をternary CPA木で実現する場合比較しており、ILP版の方がFPGA上に合成して評価した結果で、遅延は数%、面積は10数%改善されていることが示されている。

4. 提案手法

前節で述べたように、既存のヒューリスティックには性能の面で改善の余地がある。高速にILP版に近い結果をだせるような手法を実現するために、Dadda treeのアイデアを利用した手法を提案する。

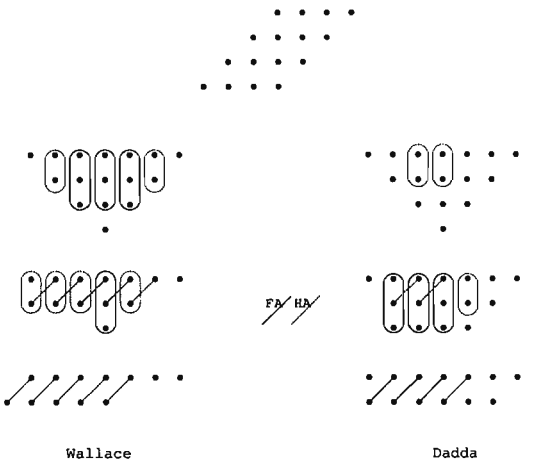

図3 dadda乗算法

4.1 Daddaの手法

Dadda treeによる部分積加算回路の実現方法は、Wallace tree版の変形であり、カウンタを割り当てるステージ数が最小となるように、各ステージの高さをあらかじめ決めておくことを特徴としてもつ。

（1） 終からj段めのステージに対する行列の高さをd_jとして、

$$d_1 = 2, d_{j+1} = \lfloor 1.5 \cdot d_j \rfloor$$

とする。そして、もとの部分積行列の少なくとも一列はd_jビットより多くの要素をもつような、最小のjを求める。

（2） 終からj番目のステージにおいて、(3,2), (2,2)カウンタを用いてどのコラムもd_jビットより多くならないように行列を縮小する。

（3） $j = j - 1$として、すべての列が2行以下となるまで、ステップ2を繰り返す。

図3に、Wallaceの手法とDaddaの手法を適用した違いを示す。小さな例ではあるが、結果としてハードウェア量に違いがでてきている。

4.2 GPCへの応用

提案手法による処理フローを以下に示す。

- 与えられた入力数m、出力数nから、m入力n出力のGPCを列挙し、入力数から出力数への削減率を指標として優先順位をつける。
- 各ステージの段数を計算する：
- 各ステージでは、LSB側から始めて、その制限段数（下位ビットからの追加分も含めて）になるようにGPCを選んで適用する。

与えられた入出力数に対する可能なGPCは、既存手法と同様に前もって計算しておく。優先順位は、減らすべき高さを満たす中で、上位ビットをカバーできる個数によってつける。6入力3出力のGPC集合を図4に示す。

各ステージの段数は、最も高さを減らせる(0,6)GPCを基準にして計算している。(0,6)GPCは、6入力を3出力に減らす

表 1　実験結果 (CPA のオペランドが 2 の場合

		Ours	GD	/Ours
12*12	#GPC	55	48	0.87
	#LUT	144	142	0.99
	LV	3	4	
16*16	#GPC	94	91	0.97
	#LUT	253	270	1.07
	LV	4	5	
24*24	#GPC	208	198	0.95
	#LUT	579	593	1.02
	LV	4	4	
32*32	#GPC	364	363	1.00
	#LUT	1031	1087	1.05
	LV	5	5	
64*64	#GPC	1409	1434	1.02
	#LUT	4102	4299	1.05
	LV	6	6	

表 2　実験結果 (CPA のオペランドが 3 の場合)

		Ours	GD	/Ours
12*12	#GPC	34	37	1.09
	#LUT	102	110	1.08
	LV	2	2	
16*16	#GPC	65	78	1.20
	#LUT	195	233	1.19
	LV	3	3	
24*24	#GPC	163	178	1.09
	#LUT	489	533	1.09
	LV	3	3	
32*32	#GPC	303	337	1.11
	#LUT	909	1010	1.11
	LV	4	4	
64*64	#GPC	1284	1382	1.08
	#LUT	3852	4144	1.08
	LV	5	5	

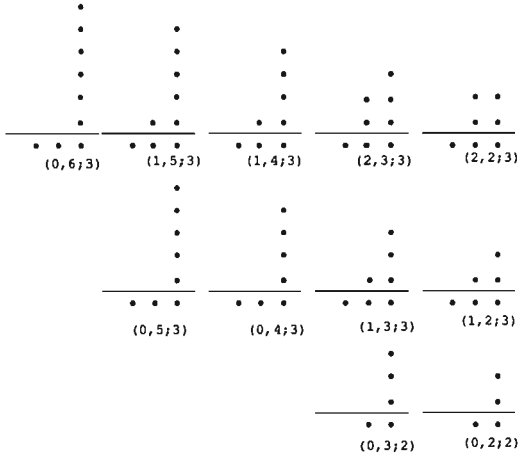

図 4　6 入力 3 出力 GPC

ことができる。したがって、最終的な CPA の入力数が 3 である場合には、

$$3 \rightarrow 6 \rightarrow 12 \rightarrow \ldots$$

というように値を設定する。CPA の入力数が 2 の場合には、2 出力の GPC を使用する必要があるため、(0,3)GPC(フルアダー) あるいは (0,2)GPC（ハーフアダー）を使用して 3 から 2 へ削減するようにしている。

5.　実験および考察

提案手法を C++で実装して数種の乗算器に対して実行し、部分積加算回路を構成する GPC の個数、段数の評価を行った。既存手法については、[4] で述べられていたアルゴリズムに沿って数種のヒューリスティックを実装し、比較に用いた。使用する GPC は、6 入力 3 出力で、最終目標とする CPA のオペランド数は 2, 3 の場合で実行した。結果を表 1,2 に示す。

表 1,2 において、1 列めは乗算器の入力ビット幅を示す。2 列目の#GPC、#LUT、LV はそれぞれ、GPC の総数、LUT 換

算の総数、および、GPC の最大段数を示す。GPC 1 個は、その出力数個分の LUT で実現できる。3 から 5 列目の結果において、Ours は提案手法、GD は比較手法の中の最小値、Ours は、GD の Ours に対する比を示している。

表より、CPA 入力が 3 の場合には、10%前後、多い場合は 20%近く面積が削減できることが可能であることが確認できた。一方、入力数が 2 の場合はの改善に 3 のときほど大きな効果は見られなかったが、LUT 数としては数%の改善が得られた。12,16 ビットのときには面積がむしろ多くなっているが段数は小さくなり、より高速な構成が得られている。3 は 2 への前ステージであり、最後のステージでは FA か HA を使うことになる。削減率が小さいため面積削減の効果が減っていると思われる。ただし、FA と HA は出力が 2 であるため LUT の個数としては同等あるいは数%小さい値が得られている。

実行時間は、64 ビット乗算器の場合でも 1 秒以内 (Pentium 4, 2.40GHz) で結果が出ており、十分実用的であると言える。[4], [5] の結果との直接の比較はできないが、ヒューリスティック版に対しては面積削減の効果、ILP 版に対しては、実行時間での優位性が評価できたと考えられる。

今回の実験では入出力数を 6 と 3 に設定して行ったが、この数値に特価しているものではないので、1LUT で実現できるなら、例えば 6,4 カウンタ等も考えることは可能である。ただしその場合、m, n に応じて、カバーするための GPC を列挙し優先順位を計算することが必要になる。

6. おわりに

本稿では、6-LUT を構成要素としてもつ FPGA を対象として、GPC による木構造を生成する手法について提案し、GPC の個数削減の効果があることを示した。この手法は乗算器に特化されたものではなく、多入力加算、積和演算など、dot diagram で表される加算問題に落とせる演算群に対して同様に効果があると考えられる。

一方、最終的に 2 個に落とす場合には、最後の段階で効果が相殺される可能性もわかり、さらなる検討の余地が残っている。また、今回 GPC の個数に注目して、その傾向が得られたが、実際に FPGA 上で実現されるまでには、まだマッピング等、最終的な性能に影響を与える処理が残っている。今後、特定の FPGA アーキテクチャ上での評価の必要がある。

7. 謝　辞

この研究は一部、早稲田大学グローバル COE プログラム「アンビエント SoC 教育研究の国際拠点」(文部科学省研究拠点形成費補助金)、および、「日本学術振興会科学研究費補助金」により支援された。

文　献

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

[2] C. Wallace, "A suggestion for a faster multiplier," IEEE Trans. Electron. Comput., vol.13, pp.14–17, February 1964.

[3] L. Dadda, "Some schemes for parallel multipliers," ALta Frequenza, vol.34, pp.349–356, May 1965.

[4] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Efficient synthesis of compressor trees on fpgas," Asia and South Pacific Design Automation Conference, pp.138–143, January 2008.

[5] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Improving synthesis of compressor trees on fpgas via integer linear programming," Design, Automation and Test in Europe Conference and Exhibition, pp.1256–1261, March 2008.

[6] C. Altera, "Stratix ii vs. virtex-4 performance comparison," September 2006. Available online from http://www.altera.com/.