Multi-Operand Adder Synthesis on FPGAs using Generalized Parallel Counters

Matsunaga, Taeko

Kimura, Shinji
Graduate School of Information, Production and Systems, Waseda University

Matsunaga, Yusuke
Faculty of Information Science and Electrical Engineering Kyushu University

出版情報：International Workshop on Logic & Synthesis. 18, 2009-08-01
バージョン：
権利関係：

Multi-Operand Adder Synthesis on FPGAs using Generalized Parallel Counters

Taeko Matsunaga
Graduate School of Information, Production and Systems
Waseda University
t_matsunaga@akane.waseda.jp

Shinji Kimura
Graduate School of Information, Production and Systems
Waseda University
shinji_kimura@waseda.jp

Yusuke Matsunaga
Faculty of Information Science and Electrical Engineering
Kyushu University
matsunaga@c.csce.kyushu-u.ac.jp

Abstract

Multi-operand adders, which are also found in parallel multipliers, usually consist of the compression trees which reduce the number of operands per a bit to two, and the carry-propagate adder for the two operands in ASIC implementation. The former part is usually realized using full adders or (3;2) counters like Wallace-trees in ASIC, though adder trees or dedicated hardware are used in FPGA. In this paper, an approach to realize compression trees on FPGAs is proposed. In case of FPGA with m-input LUT, any larger or generalized parallel counters with up to m inputs can be realized with one LUT per an output. Our approach utilizes generalized parallel counters with up to m inputs and synthesizes the compression trees to implement high-performance multi-operand adders by setting some intermediate height limits in the compression process like Dadda multipliers. Several experiments on Altera's Stratix III show its effectiveness against existing approaches.

1. Introduction

Multi-operand addition, which is often found in partial product reduction of multipliers, or some combinations of addition and multiplication, is a fundamental and frequently-used arithmetic operation. Though it can be realized with carry-propagate adder (CPA) trees, fast multi-operand addition usually consists of two phases, where the number of addends is compressed to 2 such as a Wallace tree[1] and a Dadda tree[2], and then the final CPA generates the result of multiplication for ASIC implementation.

Such trees are often constructed using 3-input 2-output counters (also called carry-save adder or full adder) and 2-input 2-output counters (half adder) as basic components[1][2][3][4]. These compression trees can be constructed using not only (3;2) and (2;2) counter, but also larger counters[5][6]. Larger counters can compress the number of addends rapidly, while the costs of each counter such as delay and area become higher. So it is not obvious how fast or small the compression trees become without considering implementation details of the larger counters and cell libraries to be used.

On the other hand, arithmetic operations are usually implemented using dedicated hardware on FPGAs. Many FPGA architectures support fast carry calculation and some DSP modules, and it is considered to be better to utilize such structures to achieve fast operations like addition and multiplication than to construct compression trees on FPGA using LUTs.

However, a larger counter can be implemented as the similar cost as a smaller counter if the numbers of inputs of those counters are within the number of inputs of LUT for each output. As the available number of inputs of LUT, m, becomes larger, compression trees could be implemented more effectively on m-LUTs and high-performance multipliers could be generated using such compression trees. That observation was pointed out in [7], and an heuristic and an approach using Integer Linear Programming (ILP) to construct compression trees using GPCs have been proposed in [7] and [8], respectively.

In this paper, the same problem, that is, synthesis of compression trees targeting m-LUT based FPGAs is addressed. Our objective is to synthesize fast compression trees for practical size of problems in reasonable time by setting some intermediate height limits in the compression process like Dadda-tree. Performance on FPGAs depends not only the maximal level of counters but also the number of counters. Our approach suppresses generation of unnecessary counters for reduction of the maximal level, which results in reduction of delay as well as area.

The rest of this paper is organized as follows. Some definitions and the target problem are shown in the section 2. After related work is reviewed in section 3., the main features of our approach is shown in section 4. Then experimental results for multipliers and multi-operand adders and conclusion are shown in section 5. and 6. respectively.

2. Preliminaries

Assume that $A = a_{n-1}a_{n-2}\ldots a_0$ is an n-bit unsigned binary number. An index of a_i, i, is referred to as the rank of a_i where $0 \leq i \leq n-1$.

Definition 1 *A single column parallel counter, $(m;n)$ is a combinational circuit which inputs m bits, counts the number of bits that are set to 1, and outputs the number as an n-bit unsigned binary number ranging from 0 to m. The following relation is satisfied between m and n:*

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

A single column parallel counter (m;n) accepts inputs of the same rank,i, and the ranks of outputs become $i, i+1, \ldots, i+(n-1)$. A full adder and a half adder is a (3;2) and (2;2) counter respectively.

Definition 2 *A generalized parallel counter, referred to as a GPC, is a combinational circuit which accepts input bits with different ranks, calculates the sum of input bits, and outputs the number as an n-bit unsigned binary number ranging from 0 to M. A GPC is represented as $(m_{k-1}, m_{k-2}, \ldots, m_0; n)$ where $m_{k-1} > 0$. m_i is the number of inputs with the rank i, and n is the number of outputs. The following relation is satisfied among M, m_i, and n:*

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

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

For example, $GPC(4,4;4)$ accepts 4 inputs with rank 0 and 4 inputs with rank 1, and outputs a 4-bit unsigned binary number. The maximum number M occurs when all inputs are set to 1 and the value is $M = 4 \cdot 2^1 + 4 \cdot 2_0 = 12$, which requires 4-bit.

Definition 3 *The ratio of the number of inputs to the number of outputs,*

$$\frac{\sum_{i=0}^{k-1} m_i}{n}$$

is referred to as the reduction ratio of a GPC.

In the followings, only GPCs whose reduction ratios are greater than or equal to 1 are considered.

Definition 4 *A dot diagram represents all operands to be summed aligned by columns. Each bit of each operand corresponds to a dot located in a column corresponding to the bit position.*

Figure 1 shows an example of dot diagrams. (a) and (b) represent a 4*4 multiplier and a GPC(4,4;4), respectively. In Figure 1(b), the upper part above the line corresponds to inputs and the lower part corresponds to outputs.

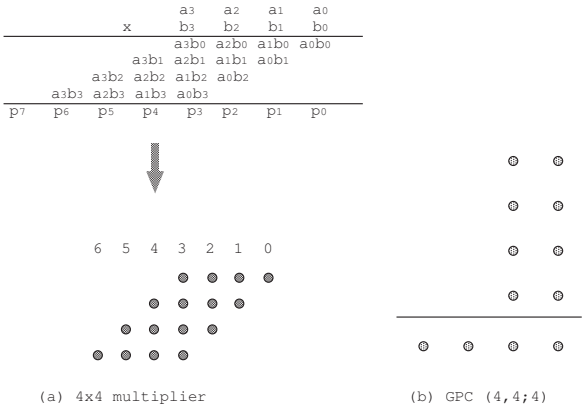

Figure 1. Examples of dot diagrams

Definition 5 *A compression tree accepts more than two inputs A_i to be summed, and outputs the specified number k of operands O_j such that*

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

Outputs of compression trees are connected to inputs of the carry-propagate adder(CPA) whose outputs are the final results of multi-operand additions. The number of operands of CPAs, k is usually 2, but can be more than 2 when the target FPGAs support fast k-ary addition.

Our target problem is as follows:

Problem 1 *Given a dot diagram which represents all inputs to be summed, the maximum number of inputs of GPC, m, the number of outputs of GPC, n, and the number of outputs of a compression tree, k, find a compression tree which consists of m-input n-output GPCs and outputs k operands.*

We call a compression tree as a GPC network where only m, n, and k are given and any other detailed architectures of FPGAs are not assumed.

3. Related work

There have been many approaches for constructing (3;2) and (2;2) counter trees in multipliers[1],[2],[3],[4] for ASIC implementation. Larger counters have been discussed in [5],[6].

As mentioned before, larger GPCs can reduce the maximal level of a compression tree, while each cost of GPC related to area and delay is getting larger and slower depending on library cells and technology mapping algorithms to be used for ASIC. On the other hand, when the number of inputs of LUTs is m, each cost of GPC could be roughly estimated as same as long as the number of inputs of GPC is within m, so remarkable gain could be expected.

This problem has been first stated in [8]. Their heuristic at first lists up possible m-input n-output GPCs based on the given m,n, and gives them orders of priorities based on their reduction ratios. Then the highest column is focused and the GPC which covers dots in the column is selected according to priorities, and covered dots are removed. This process is repeated until there are no more dots to be covered by the GPCs, then new dots generated by the assigned GPCs are added to the corresponding columns. Repeat this until all columns have at most k dots.

Though this simple heuristic does not guarantee the minimum delay nor the minimum number of GPCs, experimental results indicate their advantages to FPGA dedicated implementation. The same authors also proposed an ILP version for the problem with some constraints in [7]. Compared with trees of ternary CPAs which exploits fast carry chains, their heuristic and ILP-based approach produced 27% and 32% faster circuits on Altera Stratix II on average for their experiments.

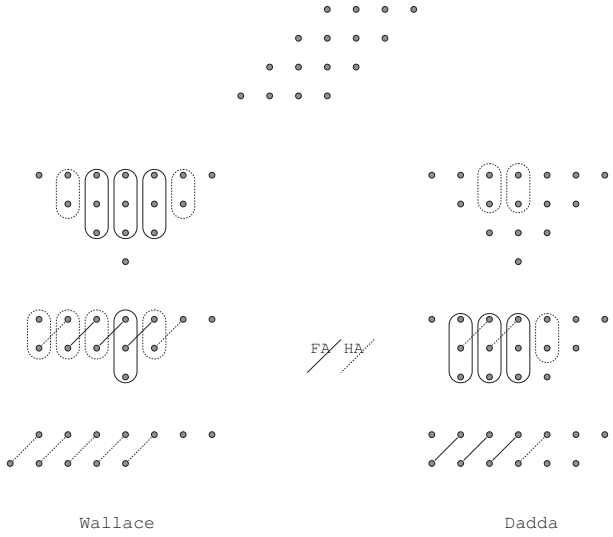

Figure 2. Wallace-tree and Dadda-tree

4. Proposed approach

Our objective is to explore a delay minimization algorithm which outperform the existing heuristic and can be applied to larger problem where ILP approach could not be applied in reasonable time.

There are several factors which affect the speed of compression trees. The most direct measure at GPC network level would be the maximal number of serially connected GPCs, and we call it the maximal level of GPCs. It roughly corresponds to LUT related delay on FPGA. Though wiring delay on FPGA can not be measured accurately at GPC level where the target FPGA device is not specified, we observed experimentally that the number of GPCs would affect the delay because a larger circuit causes wire delay to be expanded. Other than above two factors, arrival times for input signals which correspond to dots in the original dot diagram would also affect the total delay.

We focus on minimization of the maximum level and the number of GPCs as objectives of delay minimization at GPC network, and adopt an idea where some intermediate limits on the column heights are set and the number of dots per a column is reduced to the height using GPCs at each stage of reduction like Dadda tree[2]. As for arrival times, dots in a column would be covered to GPCs in ascending order of arrival times.

In the followings, Dadda's method is first reviewed and then our approach is explained.

4.1. Dadda's method

In Dadda's method using (3;2) and (2;2) counters, the column height is iteratively reduced to the predetermined limits at each stage until all columns have at most 2 dots. The column height for the last j-th stage is d_j, and heights for other stages are calculated as $d_1 = 2, d_{j+1} = \lfloor 1.5 \cdot d_j \rfloor$. So, the sequence of the column heights is

$$2 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow \cdots .$$

At each stage, find the smallest j where at least one column d_j of the dot diagram have more than d_j dots. At each j-th stage from the last, compress the heights of all columns to not more than d_j dots, using (3;2) and (2;2) counters.

Figure 2 shows the difference between Wallace tree and Dadda tree. Since the maximal height is 4 in the first stage, the target height is 3 and only two counters are necessary to reduce all heights to not more than 3. Though this is a small example, resulting compression trees have 4 FAs and 6 HAs in Wallace tree and 3 FAs and 3HAs in Dadda tree. Both maximal levels are same.

4.2. Synthesis of compression trees using GPCs

Given the number of inputs(m) and outputs(n) of GPCs and the target height(k), our approach for synthesizing compression trees is as follows:

- Pre-processing:
  - List up candidate GPCs which have up to m inputs and n outputs, and order them based on their reduction ratios.
  - Calculate the target heights of intermediate stages based on the maximum number of dots among all columns.
- Compression phase: Repeat the following process until all columns have at most k dots.

– Starting from the lowest column in which the number of dots exceeds the limit, select GPCs which can reduce the excess and cover dots in the upper columns as many as possible until the number of dots including ones added from the lower columns get to the limit.

The main difference of our approach to the existing one is to set intermediate limits of heights, which is explained below.

4.3. Setting of the intermediate heights

In Dadda's method, the target heights can be defined based on (3;2) counter since a basic element is a (3;2) counter, while there are many candidates for m-input n-output GPCs and how to decide the target limits is of a problem.

As for transformation of dot diagrams using various possible GPCs, there have been some considerations in ASIC implementation[9][5], especially for use of single column counters and GPCs with input columns of equal height(Figure 3). Considering that our target is to syntheise on k-LUT based FPGA, we have the following observations:

- As the viewpoint of reduction ratio of columns' height, it would be preferable to use a single column counter. It is because m_{j+1} has twice weight of m_j and when the total number of inputs are fixed, the highest reduction per a column, as well as highest ratio, is achieved where all inputs have the lowest rank for the same number of outputs.
- A GPC with input columns of equal height could reduce the height of a column to less than the number of outputs of the GPC. For example, a GPC(5,5;4) in Figure 3 produces four dots at interval of two columns, so only $4/2 = 2$ outputs contribute to one column. Though the reduction ratio is lower than that of the single column counter with the same number of inputs, it could be utilized to reduce column heights to less than the number of outputs of a GPC. However, the number of inputs of GPC are decided according to the number of inputs of an LUT, which means the number will not be so large. As a result, there are not so many GPCs with the above property.

The intermediate heights d_j are decided based on the single column counter $(m;n)$ with the maximum reduction ratio in our approach. The sequence of intermediate heights is calculated as follows:

$$d_1 = k,\ d_j = \lfloor d_{j-1} \cdot \frac{m}{n} \rfloor$$

Only when the height should be reduced to less than the number of outputs of the single column counter, GPCs with the above features or single column counters with fewer outputs are considered.

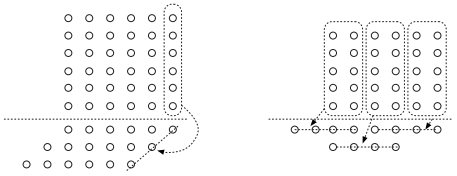

Figure 3. Dot diagram with a flat shape

These intermediate heights are calculated under the assumption that almost all the columns have equal number of dots, so this calculateion can not guarantee the minimum height for dot diagrams with inputs of unequal heights. Nonetheless, we adopted these intermediate heights. Dot diagrams of any shapes are covered by the flat dot diagram whose height is of the highest column and $(m;n)$ counter is an element of candidate GPC set. By reducing as many dots in the upper columns as possible while satisfying the target height, the number of dots would not increase more than those when $(m;n)$ counter are used. So, the intermediate height is guaranteed to be satisfied at each stage without any extra levels.

4.4. An example: 6-input 3-output GPCs

Figure 4 shows 6-input 3-output GPCs as an example of a set of GPCs. ΔH indicates the increased or decreased number of dots of each column. For example, $(0,6;3)$ decreases the number of dots of column j by 5 and increases by 1 for columns $j+1$ and $j+2$.

When the target height, the intermediate height is calculated based on (0,6;3) GPC with the reduction ratio 6/3 = 2. Assume that the number of CPA inputs is 3, the intermediate heights are set to

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

Figure 5 shows the process of reduction. Assume that the first column which exceeds the target intermediate height is j and the excess is two. $(2,3;3)$, which can reduce the column j by two dots and have the highest ratio, is selected. It removes three dots in the column j and two dots the column $j+1$ in ascending order of arrival times, and adds one dot to the columns j, $j+1$, and $j+2$. This process is repeated until the heights of all columns including dots generated from the lower columns get to the limit.

When the number of CPA inputs is 2, (0,6;3) GPC can not be used. In this case, GPC(2,3;3) have the property of GPCs with inputs of equal height as described above, and can be used for that purpose (Figure 6). The number of GPCs can be reduced compared to the case of using (3;2) or (2;2) counters.

5. Experiments

We implemented the above heuristic in C++, and executed for the following examples:

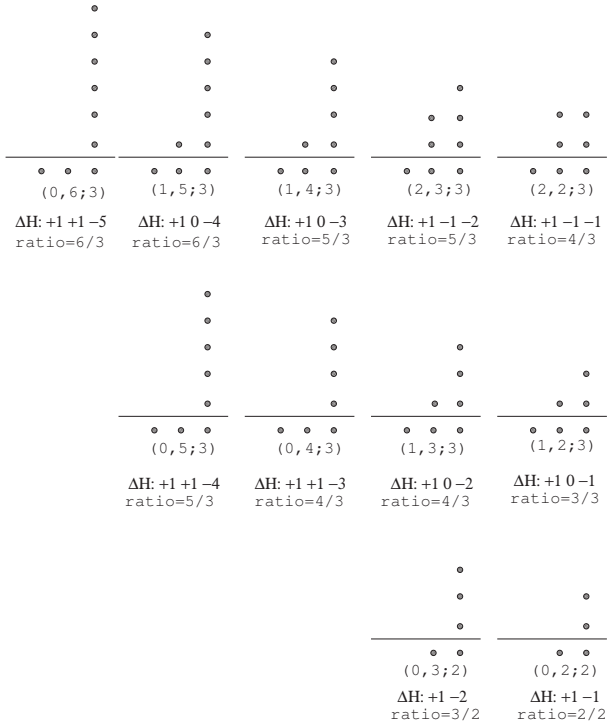

Figure 4. A set of 6-input 3-output GPCs

- n-bit × n-bit multiplications:
  $n = 16, 24, 32, 64, 96, 128$
- 16 operand n-bit additions:
  $n = 16, 24, 32$

The number of operands of CPAs, k is set to 2 and 3. Assumed that the target FPGA is Altera StratixIII[10] which utilizes 6-input LUTs and fast ternary addition, the number of inputs and outputs of GPCs are set to 6 and 3 respectively.

Our algorithm generated overall circuits including compression trees and CPAs described in Verilog HDL, and then they were compiled by Quartus II v.8.0 on Altera Stratix-III. The CPA part was implemented using fast binary or ternary addition of the device according to the target number k.

As a target for comparison, we implemented a greedy heuristic similar to [8].

5.1. Experimental results at GPC network level

Table 1 and 2 show the results at GPC network level for n-bit multipliers and 16-operand n-bit adders, respectively. 'Ours' and 'greedy' in the first column mean our proposed method and a heuristic similar to [8]. The second column shows the bit width of operands where all operands have the same width. The third and forth columns show the number of GPCs and the maximal level in case that k = 3 and the fifth and sixth columns are those in case that k = 2. As shown in these tables, our method produces smaller GPC network under the same or less maximal level. Execution times are within 2 seconds for 128-bit multiplier and within 0.1 second for each multi-operand additions(Pentium 4, 2.40GHz).

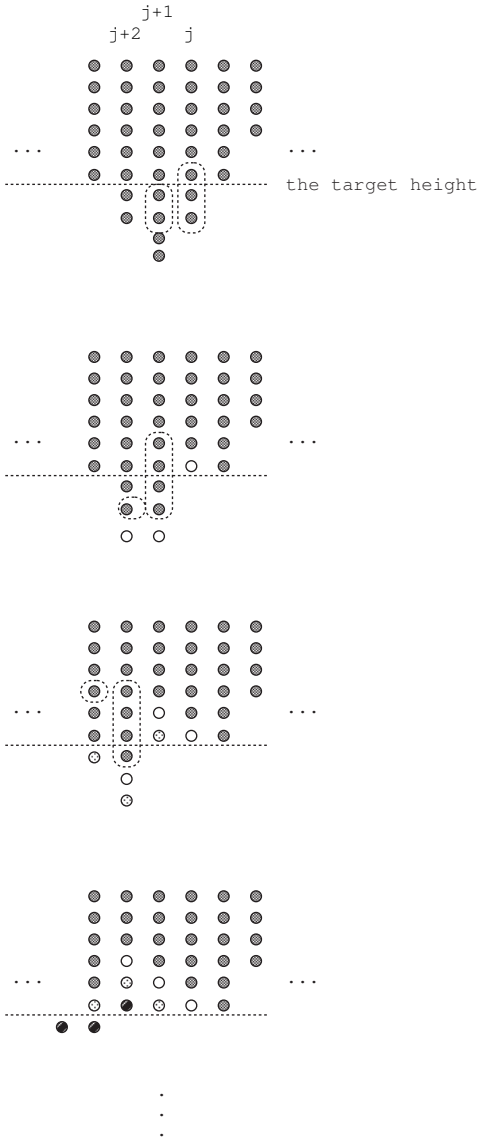

Figure 5. Reduction process

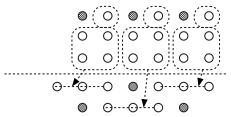

Figure 6. Dot diagram covered by (2,3;3)

Table 1. Experimental results at GPC level(multipliers)

		$k=3$		$k=2$	
type	bit	#GPC	#lv	#GPC	#lv
ours	16	65	3	80	4
greedy	16	78	3	91	5
ours	24	163	3	186	4
greedy	24	178	3	198	4
ours	32	303	4	334	5
greedy	32	337	4	363	5
ours	64	1,284	5	1,347	6
greedy	64	1,382	5	1,434	6
ours	96	2,949	5	3,044	6
greedy	96	3,084	5	3,168	6
ours	128	5,297	6	5,424	7
greedy	128	5,547	6	5,651	7

Table 2. Experimental results at GPC level(16-operand additions)

		$k=3$		$k=2$	
type	bit	#GPC	#lv	#GPC	#lv
ours	16	73	3	82	4
greedy	16	86	4	95	4
ours	24	109	3	122	4
greedy	24	130	4	143	4
ours	32	145	3	162	4
greedy	32	174	4	191	4

5.2. Evaluateion on FPGA

Table 3 and 4 show area and delay on Stratix III for multipliers and 16 operand adders respectively. 'add' in the first column indicates the results where compression networks are implemented as ternary adder trees using FPGA-specific fast carry structure. The third column shows the number of used ALMs which are basic components in StratixIII, and the fourth column shows the maximal delay.

As shown in those tables, our approach produces 7-25% gain for multipliers and 23-47% for multi-operand additions compared to ternary adder trees in speed for our examples.

Compared to a greedy approach, our approach produced faster circuits in most cases. Since the maximal levels of GPC network are often same, reduction of the number of GPCs would lead speed up. However, the greedy approach could achieve faster circuits in 16-operand 32bit adders in spite of the fact that the number of GPCs and the maximal level is smaller. One reason could be that we evaluated the multipliers and adders including not only compression trees but also the last CPA. In our experiments, CPAs were implemented as ripple-carry adders by using fast carry chain automatically. In this case, required times for outputs of GPC network could be bit-wise different, so the objective that the maximal level is minimized could not be adequate for minimization of the maximal delay of the whole circuits. As for CPA implementation, one could apply some approach targeting ASIC such as parallel prefix adder synthesis[11] considering bit-wise timing constraints to FPGA implementation. Future work should include an approach which considers the relation between GPC networks and the last CPA.

Table 3. Experimental results on FPGAs(multipliers)

type	bit	#ALM	dpath
ours	24	716	5.113
greedy	24	745	5.317
add	24	246	6.038
ours	32	1,333	6.134
greedy	32	1,399	6.628
add	32	425	6.989
ours	64	5,728	9.416
greedy	64	5,889	9.656
add	64	1,624	10.085
ours	96	13,177	11.752
greedy	96	13,407	12.061
add	96	3,579	14.093
ours	128	23,728	14.472
greedy	128	24,075	14.940
add	128	6,316	18.142

Table 4. Experimental results on FPGAs(16-operand adders)

type	bit	#ALM	dpath
ours	16	204	3.969
greedy	16	224	4.106
add	16	176	4.900
ours	24	310	3.637
greedy	24	335	4.430
add	24	249	5.374
ours	32	417	4.579
greedy	32	442	4.541
add	32	349	5.918

In the above experiments, the number of inputs GPCs is fixed to 6 according to the number of inputs which a LUT can accept in case of Stratix III. The proposed approach is not specialized into those figures, and can be applied to any m and n which can be implemented in a m-LUT per an output. In those cases, all GPC candidates should be enumerated with their priorities according to those numbers. In general, m and n would be not so large number, and enumeration should be executed only once before the reduction phase, so overhead is expected not so large.

6. Conclusion

In this paper, an approach to generate compression trees based on GPCs are proposed targeting FPGA based on *m*-LUTs. By setting the intermediate heights as in Dadda-tree, and covering more dots in the upper columns satisfying with the limit of height, faster compression trees can be obtained at global structure level and on Altera Stratix III experimentally than another heuristic and ternary adder trees. Though we could not compare our approach with ILP-based one directly, our approach can be applied to larger problems in practical execution times and that could be the advantage to ILP-based approach.

The effectiveness could vary depending to design, so further evaluation is needed for the dot diagrams with various shapes which are shown in practical designs other than multiplications and multi-operand additions. We should also confirm that our intermediate heights are adequate in such cases.

Acknowledgments

This research was supported by Waseda University Global COE Program "International Research and Education Center for Ambient SoC" sponsored by MEXT, Japan, and Grant-In-Aid for Scientific Research from JSPS.

References

[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] W. J. Stenzel and W. J. Kubitz, "A compact high-speed parallel multiplication scheme," *IEEE Transaction on Computers*, vol. C-26, pp. 948–957, 1977.

[6] A. K. Verma and P. Ienne, "Automatic synthesis of compressor trees: Reevaluating large counters," in *DATE*, 2007.

[7] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Improving synthesis of compressor trees on fpgas via integer linear programming," in *DATE*, 2008.

[8] H. Parandeh-Afshar, P. Brisk, and P. Ienne., "Efficient synthesis of compressor trees on fpgas," in *ASPDAC*, 2008.

[9] L. Dadda, "On parallel digital multipliers," *Reprinted from Alta Frequenza*, vol. 45, pp. 547–580, 1976.

[10] Altera Corp., "The Stratix III Device Handbook."

[11] 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.