LUT Based Generalized Parallel Counters for State-of-art FPGAs

Burhan Khurshid

***Abstract*- Generalized Parallel Counters (GPCs) are frequently used in constructing high speed compressor trees. Previous work has focused on achieving efficient mapping of GPCs on FPGAs by using a combination of general Look-up table (LUT) fabric and specialized fast carry chains. The resulting structures are purely combinational and cannot be efficiently pipelined to achieve the potential FPGA performance. In this paper, we take an alternate approach and try to eliminate the fast carry chain from the GPC structure. We present a heuristic that maps GPCs on FPGAS using only general LUT fabric. The resultant GPCs are then easily re-timed by placing registers at the fan-out nodes of each LUT. We have used our heuristic on various GPCs reported in prior work. Our heuristic successfully eliminates the carry chain from the GPC structure with the same LUT count in most of the cases. Experimental results using Xilinx Kintex-7 FPGAs show a considerable reduction in critical path and dynamic power dissipation with same area utilization in most of the cases.**

Index Terms* - *Look-up table, Compressor trees, Technology mapping, Retiming

Original Research Paper
DOI: 10.7251/ELS1721003K

I. INTRODUCTION

Multi-operand addition is an important operation in many arithmetic circuits. It is frequently used in many applications like filtering [1], motion estimation [2], array multiplication [3, 4, 5, 6, 7] etc. Compressor trees form the basic elements in multi-operand additions. Compressor trees based on carry save adders (CSA) typically provide higher speeds due to the avoidance of long carry chains. Wallace [3] and Dadda [7] trees are CSA based compressor trees which are frequently used in application specific integrated circuit (ASIC) design. However, the introduction of fast carry chains in FPGAs has made ripple carry addition faster than the carry save addition. Evidently CSA based compressor trees are not well suited for implementation involving FPGAs [8].

Prior work on compressor tree synthesis using FPGAs has used GPCs as basic constituent element. It has been demonstrated that the usage of GPCs can lead to a considerable reduction in the critical path delay with comparable resource utilization [8, 9, 10, 11, 12, 13, 14]. Initial attempts in this regard were made by Parandeh-Afshar et al. [8, 9, 10, 11]. In [9] they claim to report the first method that synthesizes compressor trees on FPGAs. The proposed heuristic constructs compressor trees from a library of GPCs that can be efficiently implemented on FPGAs. Their latter work [11] focuses on further reducing the combinational delay and any increase in area by formulating the mapping of GPCs as an integer linear programming (ILP) problem. They reported an average reduction in delay by 32% and area by 3% when compared to an adder tree. In [10] they focus on reducing the combinational delay by using embedded fast carry chains. This concept was further extended in [8] and a delay reduction of 33% and 45% was achieved in Xilinx Virtex-5 and Altera Stratix-III FPGAs respectively.

Matsunaga et al. [12, 14] also formulated the mapping of GPCs as an ILP with speed and power as optimization goals. Their results show a 28% reduction in GPC count when compared to [9]. A reduction in GPC count results in reduction of compression stages thereby reducing the delay and power consumption.

Recent attempts from Kumm and Zipf [15, 16] focus on exploiting the low-level structure of Xilinx FPGAs to develop novel GPCs with high compression ratios and efficient resource utilization. Both general purpose LUT fabric and specialized carry chains have been used for synthesizing resource-efficient delay-optimal GPCs.

All the above mentioned approaches (except [9]) focus on exploiting the fast carry chain embedded in modern FPGAs. The idea is to use the fast carry chain to connect the adjacent logic cells and by pass the programmable routing network to reduce delay [10]. In this paper, however, we try to avoid the usage of embedded carry chains and propose a heuristic that tries to implement GPCs using only the general LUT fabric. The heuristic tries to minimize the number of LUTs in a GPC. The area-optimized GPCs are then easily retimed by inserting registers that are available in each logic cell. Thus instead of using an LUT-carry chain combination we use an LUT-register combination to map the GPCs. The motivation for our approach is backed by following reasons:

i. GPCs based on LUTs and carry chains are purely combinational in nature. FPGAs are synchronous devices and it is better to adhere to synchronous practices while using them as implementation platforms. Our approach provides this synchronous description by including registers in the synthesis process.

Manuscript received 15 June 2015. Received in revised form 5 May 2017. Accepted for publication 2 June 2017.

Burhan Khurshid is with the Department of Computer Science and Engineering, National Institute of Technology Srinagar, Jammu and Kashmir, India, 190006 (phone: +91-9797875163; e-mail: burhan_07phd12@nitsri.net).

ii. Usually specialized FPGA resources are fixed in position. Routing data to and from the fixed blocks sometimes creates problems in the placement and routing (PAR) phase of the FPGA design flow. Thus instead of using fixed specialized resources it is desirable to use general LUT resources as their placement can be altered during PAR.

iii. Finally, retiming GPC structures by placing registers at the input of nodes with large capacitances reduces the switching activities at these nodes [17]. This results in reduced dynamic power dissipation.

The rest of the paper is organized as follows. Section II presents the basic preliminaries about the GPCs and the terminology used in this paper. Section III discusses the heuristic that is used to synthesize different GPCs. Synthesis and implementation is carried out in section IV. Conclusions are drawn in section V and references are listed at the end.

II. Preliminaries and Terminology

A compressor tree is a circuit that takes *k*, *n*-bit unsigned operands: A_{k-1}, A_{k-2}... A_1, A_0 and generates two output values, Sum (*S*) and Carry (*C*), such that:

$$\sum_{i=0}^{k-1} A_i = S + C \qquad (1)$$

A generalized parallel counter computes the sum of bits having different weights. A GPC is traditionally represented as a tuple (K_{i-1}, K_{i-2}...K_1, K_0; *n*), where K_i denotes the number of input bits of weight *i*, and *n* is the number of output bits. The upper limit on the value of GPC is given by:

$$M = K_0 2^0 + K_1 2^1 + \cdots + K_{i-1} 2^{i-1} \qquad (2)$$

$$M = \sum_{i=0}^{K-1} K_i 2^i \qquad (3)$$

$$n = \lceil log_2(M+1) \rceil \qquad (4)$$

As an example, a (1, 4, 1, 5; 5) GPC has five input bits of weight 0; one input bit of weight 1; four input bits of weight 2 and one input bit of weight 3. The upper limit on the output value is 31 and five output bits are required to represent the output.

Logic synthesis is concerned with hardware realization of a desired functionality with minimum possible cost. The *cost* of a circuit is a measure of its speed, resource utilization, power consumption or any combination of these. A *Boolean network* is a directed acyclic graph (DAG) that represents a combinational function. Logic gates, primary inputs (PIs) and primary outputs (POs) within this network are represented by *nodes*. Each node implements a *local function*. A *global function* is implemented by connecting the logic implemented by individual nodes. The transformation of a Boolean network into targeted logic elements gives the *circuit-netlist*. For FPGAs the targeted element is a *k*-LUT.

A *cone* of node *v*, C_v, is a sub-network that includes the node *v* and some of its non-PI predecessor nodes. Any node *u* within this cone has a path to the *root* node *v*, *u*→*v*, which lies entirely in C_v. The *level* of the node *v* is the length of the longest path from any PI node to *v*. Network *depth* is defined as the largest level of a node in the network. The critical path delay and area of a circuit is measured by the depth and number of LUTs respectively. A node may have zero or more predecessor nodes known as *fan-in* nodes. Similarly a node may drive zero or more successor nodes known as *fan-out* nodes. A network is said to be *k bounded* if the fan-in of every node does not exceed *k*.

III. GPC Mapping Heuristic

This section describes the heuristic for efficiently mapping the GPCs onto LUTs. The primary goal of the heuristic is to eliminate the fast carry chain and map the GPCs onto minimum possible LUTs. Eliminating the carry chain makes the GPCs feasible to pipelining. The resulting structures are easily pipelined by placing the registers along the feed-forward cut-sets. We explain the different steps involved in the heuristic by considering the mapping of GPC (1, 4, 1, 5; 5). Conventional implementation requires four LUTs and a CARRY4 primitive, with a total delay of T_L+4T_{CC}, where T_L is the delay associated with a single LUT and T_{CC} is the single carry delay. Figure 1 shows the Boolean network for (1, 4, 1, 5; 5) GPC. The network has eleven inputs and five outputs. All the primary inputs, primary outputs and intermediate signals have been labeled.

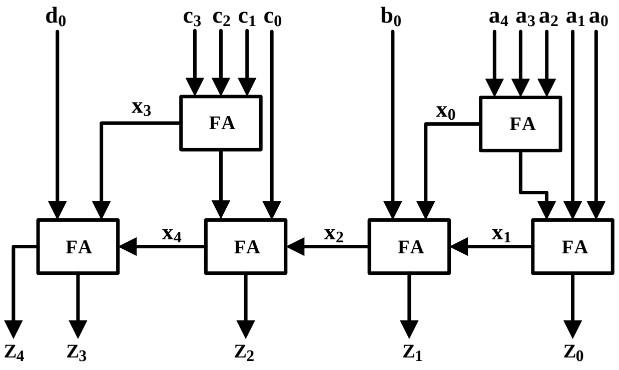

Fig. 1. Boolean network for (1, 4, 1, 5; 5) GPC

<u>Construction:</u> The first step constructs multiple networks from the original network. This is done by traversing the parent network and dividing it at the output nodes. Thus Boolean networks corresponding to each output node are constructed in this step. For the parent network of figure 1 there are five output nodes resulting in five different Boolean networks. The individual networks are named as per their outputs Z_0, Z_1, Z_2, Z_3 and Z_4. This is shown in figure 2.

<u>Recognition and Prioritization:</u> After the individual networks have been obtained, the heuristic searches for redundant nodes in each of the networks. Redundant nodes are the nodes which exist in more than one network. These are shown as shaded portions in figure 2. The network for redundant nodes is then drawn separately as shown in figure 3. Each redundant network is assigned a priority based on the number of appearances in the original networks of figure 2. For example, the network in figure 3(a) is assigned a priority

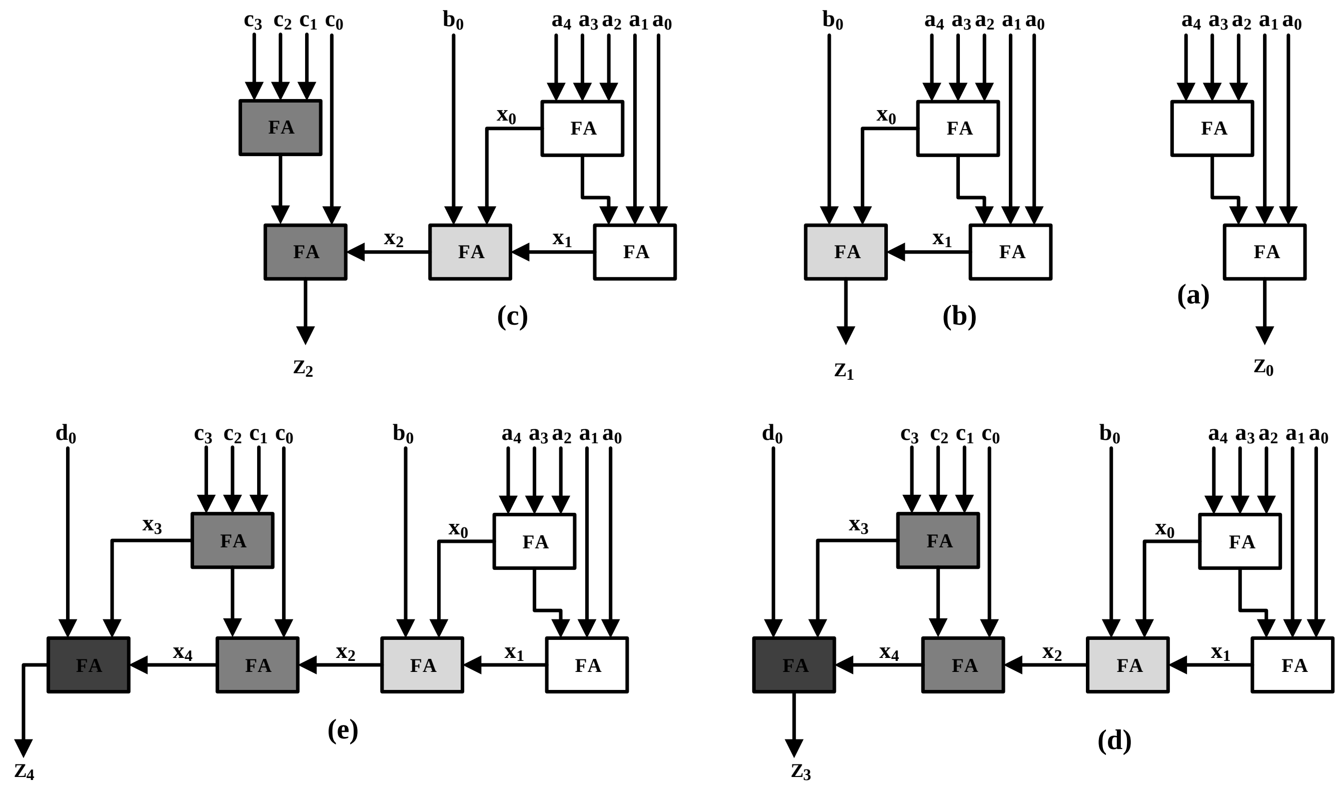

Fig. 2. Boolean networks for individual outputs. Redundant nodes are shown in same shades.

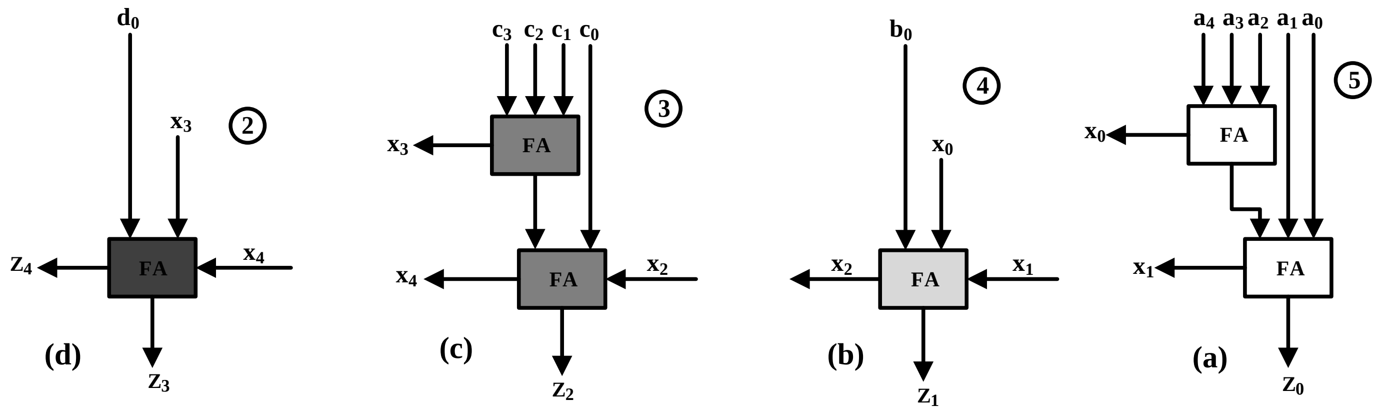

Fig. 3. Boolean networks for redundant nodes. The number in the circle represents the priority of each network.

of 5 because it appears in five different networks. Similarly 3(b) is assigned a priority of 4 because it appears in four different networks and so on. Note that the entire parent network can be constructed by interconnecting these redundant networks.

Covering and Re-structuring: Next the heuristic tries to optimally map these redundant networks onto LUTs. Mapping is done as per the priority, as it results in the maximum logic density. For example the network in figure 3(a) has a priority of 5 and, if mapped optimally will result in an improved logic density in all the networks it is a part of. In this paper, we have targeted FPGAs with 6-input LUTs as basic logic elements. Thus the mapping should ensure a proper utilization of this basic element. For efficient mapping each network in figure 3 is divided into sub-networks. This is again done by traversing through the network and dividing it at output nodes. Thus the network of figure 3(a) is divided into three sub-networks corresponding to outputs X_0, X_1 and Z_0. Similarly networks in 3(b), 3(c) and 3(d) are divided into different sub-network as per their fan-out. This is shown in figure 4. A straight forward approach to mapping would be to assign the logic implemented by each sub-network to a separate LUT. This, however, leads to under utilization of the resources. For efficient mapping, therefore, the entire assembly of sub-networks is re-structured. This requires transferring some sub-networks from their original networks to sub-networks that belong to different networks. For example sub-network X_0 that originally belonged to 4(a) is now transferred to 4(b) and included with sub-networks X_2 and Z_1. This re-structuring of sub-networks ensures a proper utilization of the LUT fabric. Note that the 6-input LUTs in Xilinx FPGAs can implement a single 6-input function or two 5-input functions with shared inputs. The re-structured sub-networks are shown in figure 5. The re-structured sub-networks are then efficiently mapped onto 6-input LUTs by directly mapping their functionalities onto these target elements.

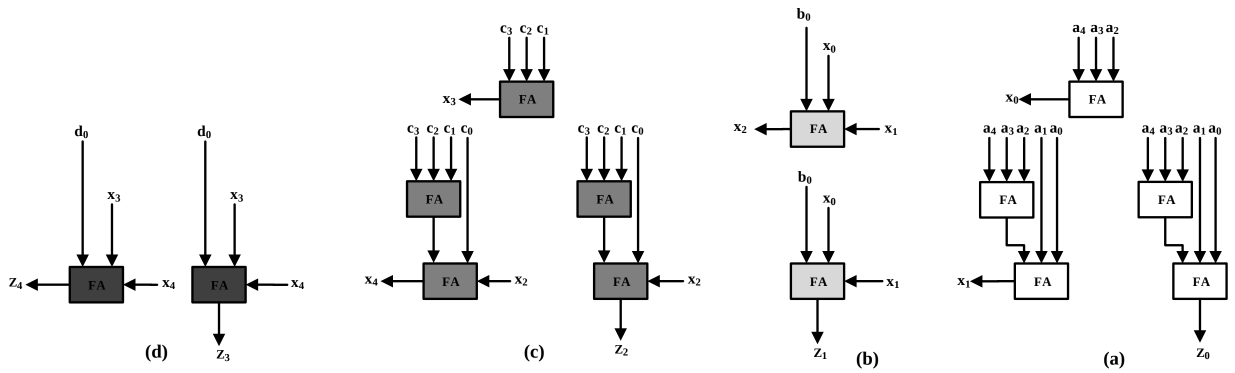

Fig. 4. Sub-networks for different networks

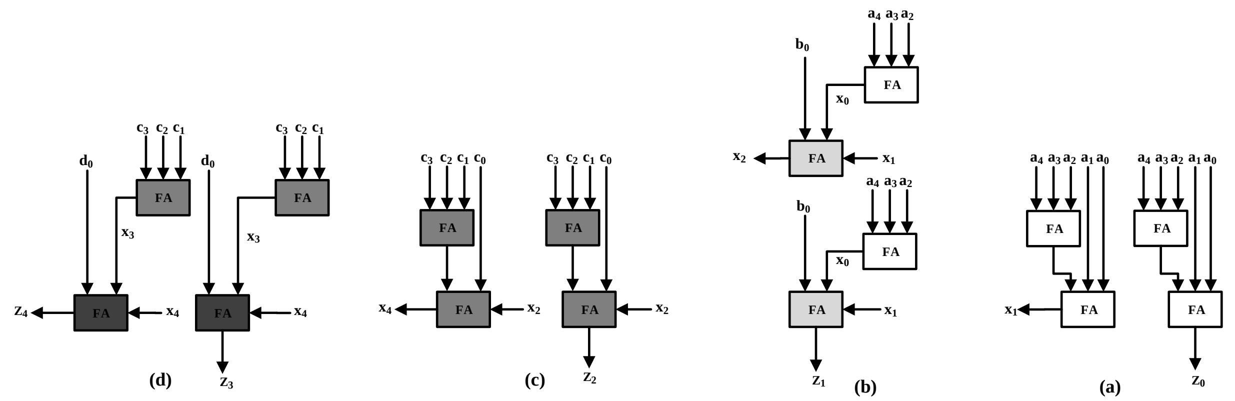

Fig. 5. Re-structuring of networks for efficient utilization of LUTs

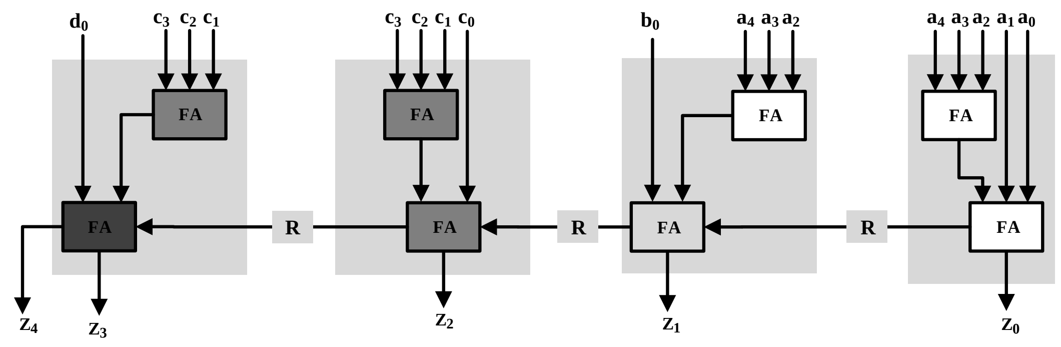

Fig. 6. Re-timed optimal circuit for (1, 4, 1, 5; 5) GPC

Re-construction and Re-timing: The parent network is then constructed by connecting the mapped networks from step III. The overall structure is a simple feed-forward structure having a unidirectional dataflow. This feed-forward nature lends itself for efficient pipelining by simply placing the registers along the feed-forward cut-sets. The final mapped and re-timed structure is shown in figure 6.

The circuit implementation of figure 6 requires four LUTs and three registers and has a critical path that includes only the delay of a single LUT (T_L). The carry chain has been eliminated and there is no increase in the delay associated with the GPC. Different GPCs proposed in prior work were implemented using this heuristic. The carry chain was successfully eliminated in all of the GPCs with no extra hardware cost, except in few cases where the column length of the GPCs exceeded five. The circuits for different GPCs are shown in figures 7, 8, 9 and 10. A theoretical evaluation of different GPCs is listed in table 1. With respect to table 1 it should be noted that previous implementations using carry chains consider only LUTs as the hardware resource. However, for each bit in a carry chain there is a carry multiplexer (MUXCY) and a dedicated XOR gate for adding/subtracting the operands with a selected carry bit. Thus an increase in LUT count that is observed in some GPCs using our heuristic may be compensated by the elimination of the resources included in the carry chain.

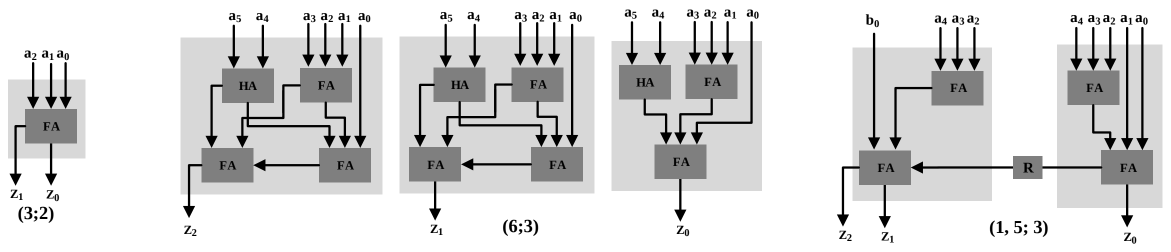

Fig. 7. LUT based GPCs from [9]

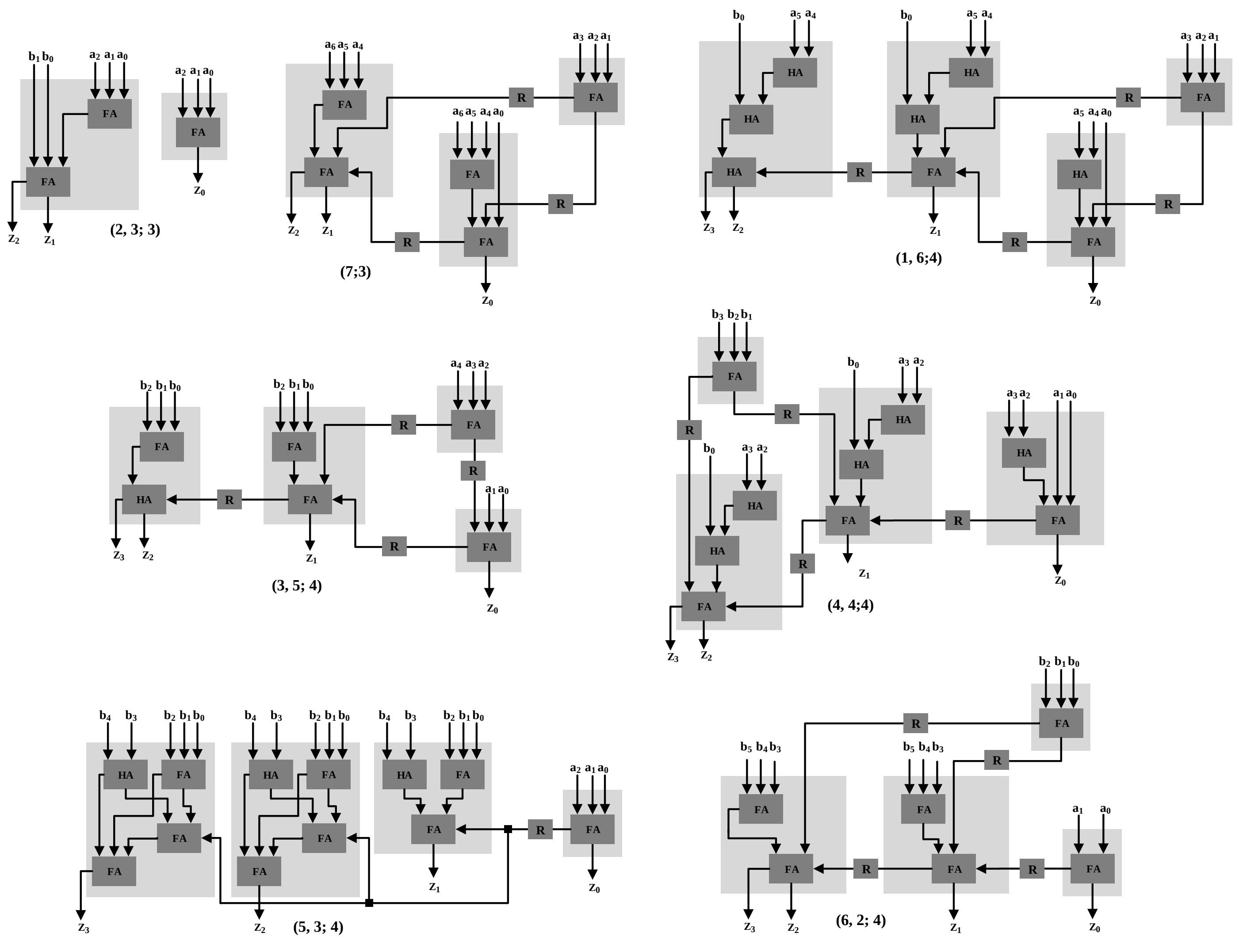

Fig. 8. LUT based GPCs from [8]

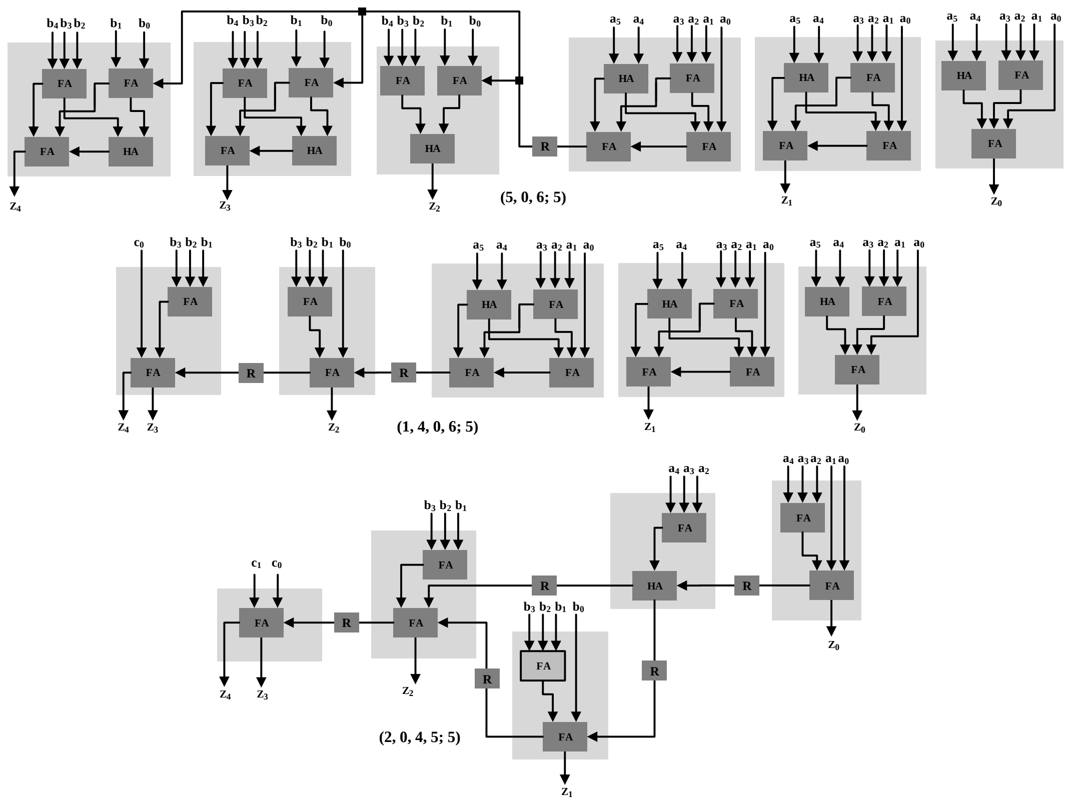

Fig.9. LUT based GPCs from [15]

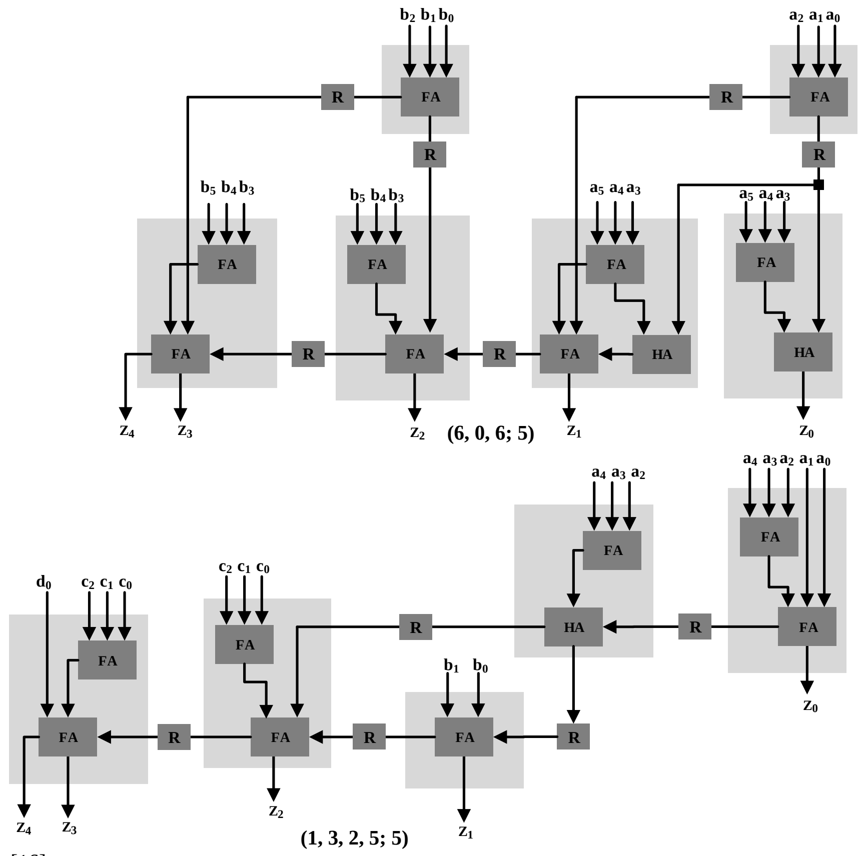

Fig. 10. LUT based GPCs from [16]

TABLE I
COMPARISON OF DIFFERENT GPCS

GPCs	Previous Mappings		Mappings based on proposed heuristic	
	LUTs	Delay	LUTs	Delay
GPCs from [9]				
(3;2)	1	T_L[1]	1	T_L
(6;3)	3	T_L	3	T_L
(1,5;3)	3	T_L	2	T_L
GPCs from [8]				
(6;3)	4	$2T_L+T_R$[2]$+4T_{CC}$[3]	3	T_L
(1,5;3)	3	T_L+3T_{CC}	2	T_L
(2,3;3)	3	T_L+3T_{CC}	2	T_L
(7;3)	4	$2T_L+T_R+4T_{CC}$	3	T_L
(1,6;4)	4	$2T_L+T_R+4T_{CC}$	4	T_L
(3,5;4)	4	$2T_L+T_R+4T_{CC}$	4	T_L
(4,4;4)	4	$2T_L+T_R+4T_{CC}$	4	T_L
(5,3;4)	4	$2T_L+T_R+4T_{CC}$	4	T_L
(6,2;4)	4	$2T_L+T_R+4T_{CC}$	4	T_L
GPCs from [15]				
(6;3)	3	$2T_L+T_R+3T_{CC}$	3	T_L
(1,5;3)	2	T_L+2T_{CC}	2	T_L
(2,3;3)	2	T_L+2T_{CC}	2	T_L
(7;3)	3	$2T_L+T_R+3T_{CC}$	3	T_L
(5,3;4)	3	$2T_L+T_R+3T_{CC}$	4	T_L
(6,2;4)	3	$2T_L+T_R+3T_{CC}$	4	T_L
(5,0,6;5)	4	T_L+4T_{CC}	6	T_L
(1,4,1,5;5)	4	T_L+4T_{CC}	4	T_L
(1,4,0,6;5)	4	T_L+4T_{CC}	5	T_L
(2,0,4,5;5)	4	$2T_L+T_R+4T_{CC}$	5	T_L
GPCs from [16]				
(6,0,6;5)	4	T_L+4T_{CC}	6	T_L
(1,3,2,5;5)	4	T_L	5	T_L

[1]delay associated with LUT.
[2]delay associated with routing.
[3]delay associated with carry chain

IV. SYNTHESIS, IMPLEMENTATION AND RESULTS

Synthesis and implementation is done using xc7k70t-2fbg676 device from Xilinx Kintex-7 family. The parameters considered are resources utilized, critical path delay and dynamic power dissipation. Constraints relating to synthesis and implementation are duly provided and a complete timing closure is ensured in each case. Synthesis and implementation is carried out in Xilinx Vivado 2016.3 [18] with speed as the optimization goal. Power analysis is done using the Xpower analyzer tool. For power analysis switching activity is captured in the value change dump (VCD) file by applying test vectors and checking for correct output. Similar test benches have been used to ensure a fair comparison. Table 2 provides a comparison of different performance metrics for different GPCs.

From table 2 it is observed that the GPC mappings based on the proposed heuristic show an average increase in speed by almost 65% and an average reduction in dynamic power dissipation by 10%. The carry chain is eliminated in each GPC with an overhead of pipelining registers and LUTs (in few cases). Each slice in Kintex-7 supports four registers which normally remain unutilized. Our experimentation with different arithmetic circuits on Kintex-7 devices reveal that each carry chain utilizes resources that are equivalent to 1 to 1.5 6-input LUTs. Thus any increase in LUT count is justified by the elimination of carry chain.

TABLE II
PERFORMANCE COMPARISON OF DIFFERENT GPCS ON XC7K70T-2FBG676

GPCs	Previous Mappings			Mappings based on proposed heuristic		
	LUTs	**Critical path(nS)**	**Power (mW)**	**LUTs**	**Critical path(nS)**	**Power(mW)**
GPCs from [9]						
(3;2)	2	0.318	0.0287	1	0.091	0.0213
(6;3)	3	0.642	0.0300	3	0.641	0.0240
(1,5;3)	3	0.642	0.0301	2	0.108	0.0214
GPCs from [8]						
(6;3)	4	0.872	0.0321	3	0.641	0.0300
(1,5;3)	3	0.772	0.0301	2	0.108	0.0300
(2,3;3)	3	0.602	0.0301	2	0.318	0.0300
(7;3)	4	0.877	0.0321	3	0.108	0.0300
(1,6;4)	4	1.101	0.0321	4	0.108	0.0300
(3,5;4)	4	1.002	0.0321	4	0.108	0.0300
(4,4;4)	4	0.988	0.0321	4	0.108	0.0300
(5,3;4)	4	0.887	0.0321	4	0.602	0.0300
(6,2;4)	4	1.007	0.0321	4	0.108	0.0300
GPCs from [15]						
(6;3)	3	0.802	0.0311	3	0.641	0.0300
(1,5;3)	2	0.796	0.0300	2	0.108	0.0300
(2,3;3)	2	0.598	0.0300	2	0.318	0.0300
(7;3)	3	0.885	0.0311	3	0.108	0.0300
(5,3;4)	3	0.839	0.0311	4	0.602	0.0300
(6,2;4)	3	0.991	0.0311	4	0.108	0.0300
(5,0,6;5)	4	0.989	0.0500	6	0.641	0.0426
(1,4,1,5;5)	4	0.839	0.0500	4	0.091	0.0371
(1,4,0,6;5)	4	0.989	0.0500	5	0.641	0.0377
(2,0,4,5;5)	4	1.101	0.0500	5	0.091	0.0377
GPCs from [16]						
(6,0,6;5)	4	1.006	0.0500	6	0.091	0.0452
(1,3,2,5;5)	4	0.989	0.0500	5	0.091	0.0377

V. CONCLUSIONS

In this paper we took an alternate approach to GPC synthesis on FPGAs. Unlike prior work on GPC synthesis that used a combination of LUTs and carry chains, we used a combination of LUTs and registers and eliminated the carry chain completely from the GPC structure. Our approach works in two steps: first a heuristic is used to eliminate the carry chain and map the GPC logic efficiently onto the underlying LUT fabric. The mapped GPC is then retimed by placing the registers along the feed-forward cut-sets. Retiming breaks the critical path resulting in higher operating frequencies. Our implementation targeting Xilinx FPGAs show an increase in speed and reduction in power dissipation for almost same resources utilized.

REFERENCES

[1] S. Mirzaei, A. Hosangadi, and R. Kastner, “High speed FIR filter implementation using add and shift method,” International Conference on Computer Design, San Jose, CA, USA, Oct. 1-4, 2006.

[2] C.Y. Chen, S.Y. Chien, Y.W. Huang, T.C. Chen, T.C. Wang, and L.G. Chen, “Analysis and architecture design of variable block-size motion estimation for H.264/AVC,” IEEE Transactions on Circuits and Systems-I, Vol. 53, No. 2, pp. 578-593, Feb. 2006.

[3] L. Dadda, “Some schemes for parallel multipliers,” Alta Frequenza, Vol. 34, pp. 349-356, May, 1965.

[4] O. Kwon, K. Nowka, and Jr. Swartzlander, “A 16-bit by 16-bit MAC design using fast 5:3 compressor cells,” Journal of VLSI Signal Procsesing, Vol. 31, No. 2, pp. 77-89, June, 2002.

[5] H. Mora Mora, J. Mora Pascual, J. L. Sánchez Romero, and F. Pujol López, “Partial production reduction based on lookup tables,” International Conference on VLSI Design, Hyderabad, India, pp. 399-404, January 3-7, 2006.

[6] J. Poldre and K. Tammemae, “Reconfigurable multiplier for Virtex FPGA family,” International Workshop on Field-Programmable Logic and Applications, Glasgow, UK, pp. 359-364, Aug. 30 – Sept. 1, 1999.

[7] C. S. Wallace, “A suggestion for a fast multiplier,” IEEE Transactions on Electronic Computers, Vol. 13, pp. 14-17, Feb., 1964.

[8] H. Parandeh-Afshar, A. Neogy, P. Brisk and P. Ienne, “Compressor Tree Synthesis on Commercial High-Performance FPGAs,” ACM Transactions on Reconfigurable Technology and Systems, Vol. 4, No. 4, Article 39, December 2011.

[9] H. Parandeh-Afshar, P. Brisk, and P. Ienne, “Efficient Synthesis of Compressor Trees on FPGAs,” in Asia and South Pacific Design Automation Conference (ASPDAC). IEEE, 2008, pp. 138–143.

[10] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Exploiting Fast Carry-Chains of FPGAs for Designing Compressor Trees," Proceedings of the 19th International Conference on Field Programmable Logic and Applications. 242-249.

[11] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Improving Synthesis of Compressor Trees on FPGAs via Integer Linear Programming," in Design, Automation and Test in Europe (DATE). IEEE, 2008, pp. 1256–1261.

[12] T. Matsunaga, S. Kimura, and Y. Matsunaga, "Power and Delay Aware Synthesis of Multi-Operand Adders Targeting LUT-Based FPGAs," International Symposium on Low Power Electronics and Design (ISLPED), pp. 217–222, 2011.

[13] T. Matsunaga, S. Kimura, and Y. Matsunaga, "Multi-Operand Adder Synthesis Targeting FPGAs," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E94-A, no. 12, pp. 2579–2586, Dec. 2011.

[14] T. Matsunaga, S. Kimura, and Y. Matsunaga, "An Exact Approach for GPC-Based Compressor Tree Synthesis," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E96-A, no. 12, pp. 2553–2560, Dec. 2013.

[15] M. Kumm and P. Zipf, "Efficient High Speed Compression Trees on Xilinx FPGAs," in Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, 2014.

[16] M. Kumm and P. Zipf, "Pipelined Compressor Tree Optimization using Integer Linear Programming," in 24th International Conference on Field Programmable Logic and Applications, 2014.

[17] K. K. Parhi, "VLSI Digital Signal Processing Systems Design and Implementation," Wiley, 1999.

[18] http://www.xilinx.com