Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2015, Article ID 518272, 16 pages
http://dx.doi.org/10.1155/2015/518272

Research Article

High Efficiency Generalized Parallel Counters for Look-Up Table Based FPGAs

Burhan Khurshid and Roohie Naaz Mir

Department of Computer Science & Engineering, National Institute of Technology Srinagar, Jammu and Kashmir 190006, India

Correspondence should be addressed to Burhan Khurshid; burhan_07phd12@nitsri.net

Received 29 June 2015; Revised 1 September 2015; Accepted 20 September 2015

Academic Editor: Martin Margala

Copyright © 2015 B. Khurshid and R. N. Mir. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Generalized parallel counters (GPCs) are used in constructing high speed compressor trees. Prior work has focused on utilizing the fast carry chain and mapping the logic onto Look-Up Tables (LUTs). This mapping is not optimal in the sense that the LUT fabric is not fully utilized. This results in low efficiency GPCs. In this work, we present a heuristic that efficiently maps the GPC logic onto the LUT fabric. We have used our heuristic on various GPCs and have achieved an improvement in efficiency ranging from 33% to 100% in most of the cases. Experimental results using Xilinx 5th-, 6th-, and 7th-generation FPGAs and Stratix IV and V devices from Altera show a considerable reduction in resources utilization and dynamic power dissipation, for almost the same critical path delay. We have also implemented GPC-based FIR filters on 7th-generation Xilinx FPGAs using our proposed heuristic and compared their performance against conventional implementations. Implementations based on our heuristic show improved performance. Comparisons are also made against filters based on integrated DSP blocks and inherent IP cores from Xilinx. The results show that the proposed heuristic provides performance that is comparable to the structures based on these specialized resources.

1. Introduction

Multioperand addition is an important operation in many arithmetic circuits. It is frequently used in many applications like filtering [1], motion estimation [2], array multiplication [3–7], and so forth. Compressor trees form the basic elements in multioperand additions. Compressor trees based on carry save adders (CSA) typically provide higher speeds due to the avoidance of long carry chains. Dadda [3] and Wallace [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 [8–14]. Initial attempts in this regard were made by Parandeh-Afshar et al. [8–11]. 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. The authors reported an average reduction in delay by 32% and area by 3% when compared to an adder tree. In [10] focus is 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.

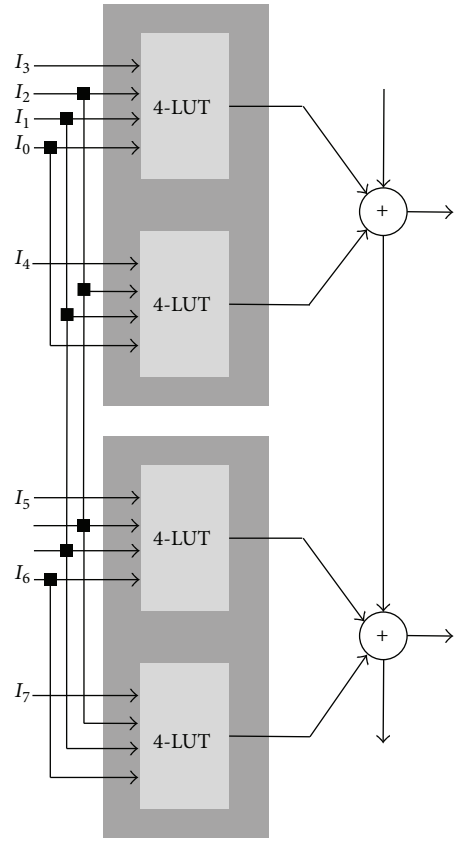


Figure 1: Stratix IV, V ALM when used in arithmetic mode.

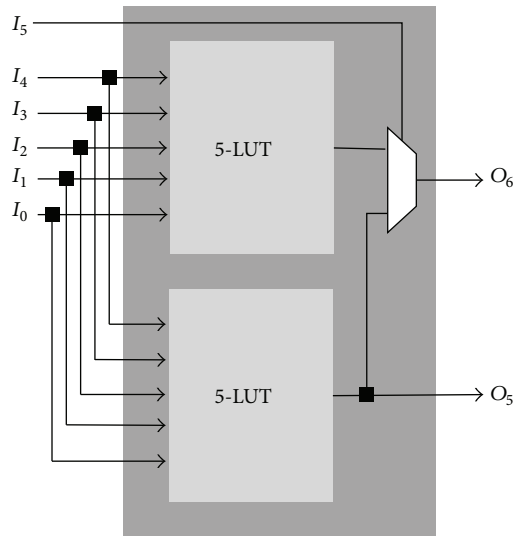


Figure 2: Xilinx 6-input dual LUT for 5th-, 6th-, and 7th-generation FPGAs.

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 abovementioned 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 bypass the programmable routing network to reduce delay [10]. This results in reduced critical path delay and hence increased speed. The mapped GPCs, however,

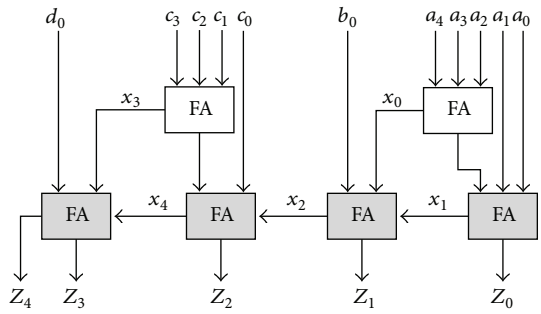


Figure 3: Boolean network for (1, 4, 1, 5; 5) GPC.

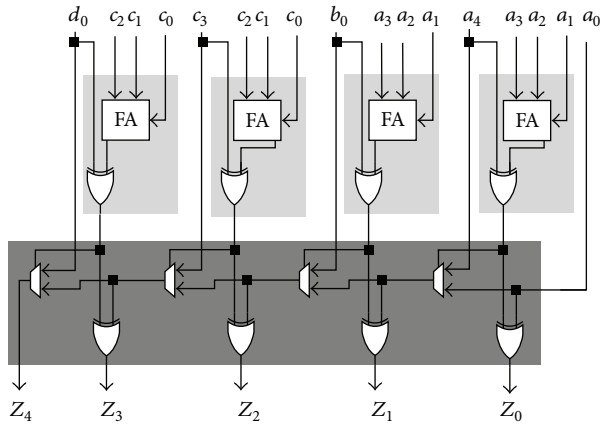


Figure 4: FPGA realization of (1, 4, 1, 5; 5) GPC using fast carry chain.

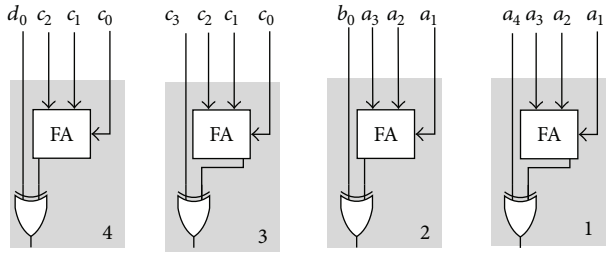


Figure 5: Boolean network for (1, 4, 1, 5; 5) GPC with no carry logic.

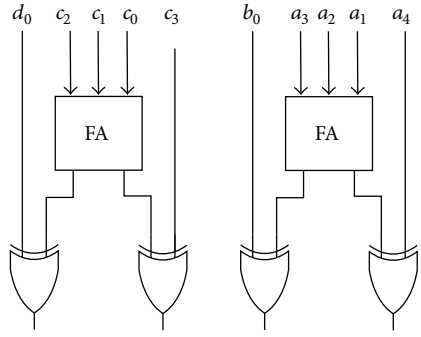


Figure 6: Boolean network for (1, 4, 1, 5; 5) GPC after node combination.

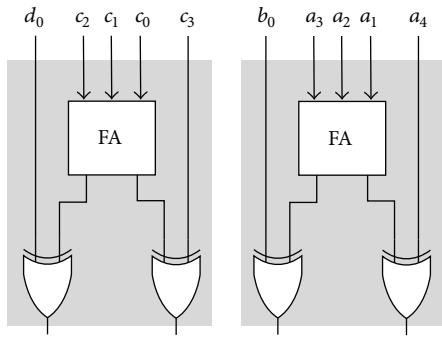


Figure 7: Covering of the combined network using 6-input dual LUTs.

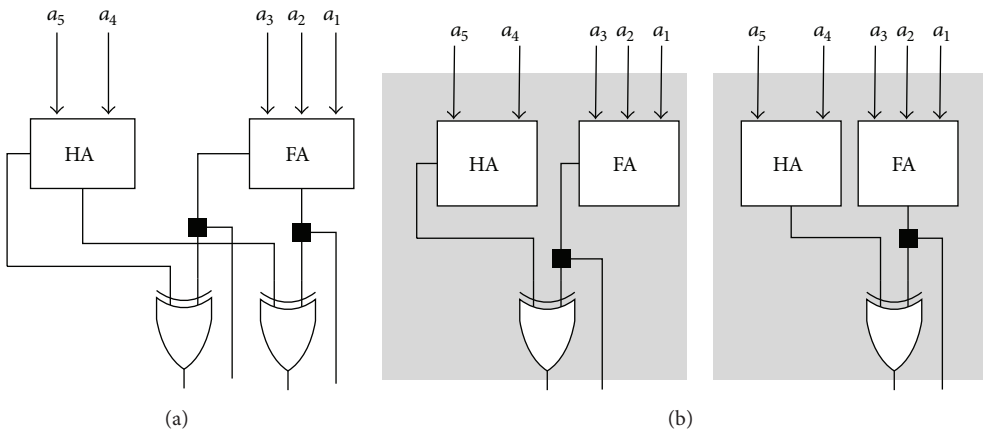


Figure 8: Covering for (0, 6; 3) GPC. (a) Combined network. (b) Restructuring for optimal covering.

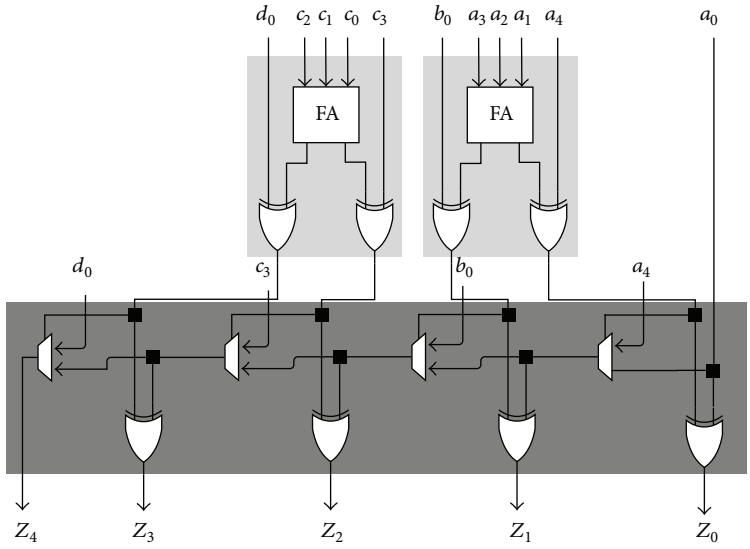


Figure 9: Area optimal mapping for (1, 4, 1, 5; 5) GPC.

suffer from poor efficiency. In this paper, we use a heuristic that improves the efficiency of the mapped GPCs by reducing the number of LUTs required to map the GPC logic. Our heuristic mainly targets 6-input LUTs from Xilinx FPGAs that can implement a single 6-input function or two 5-input functions with shared inputs. However, the same heuristic can be used for Altera FPGAs that support decomposition of 6-input LUTs into dual 4-input LUTs, when used in the arithmetic or shared-arithmetic mode. Additionally in both devices the fast carry chain is used to handle the carry rippling so that there is no increase in the critical path delay.

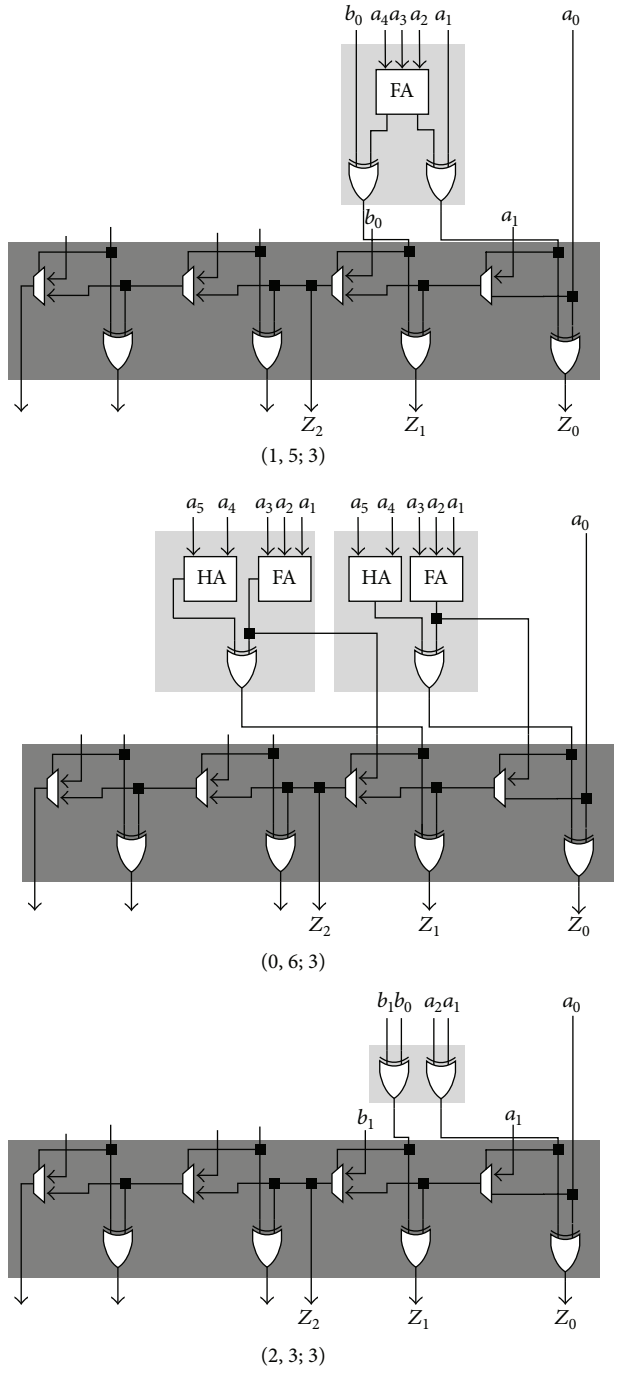


Figure 10: Area optimal mapping for different GPCs using proposed heuristic on Xilinx FPGAs.

The rest of the paper is organized as follows: Section 2 briefly introduces the basic preliminaries about the GPCs, the Xilinx and Altera LUT architecture, and the terminology used in this paper. Section 3 discusses the heuristic used to synthesize different GPCs. Synthesis and implementation are carried out in Section 4. Conclusions are drawn in Section 5 and references are listed at the end.

2. Preliminaries and Terminology

A compressor tree is a circuit that takes k, n-bit unsigned operands $A_{k-1}, A_{k-2} \cdots A_1, A_0$ and generates two output values, sum (S) and carry (C), such that

$$\sum_{i=0}^{k-1} A_i = S + C. \tag{1}$$

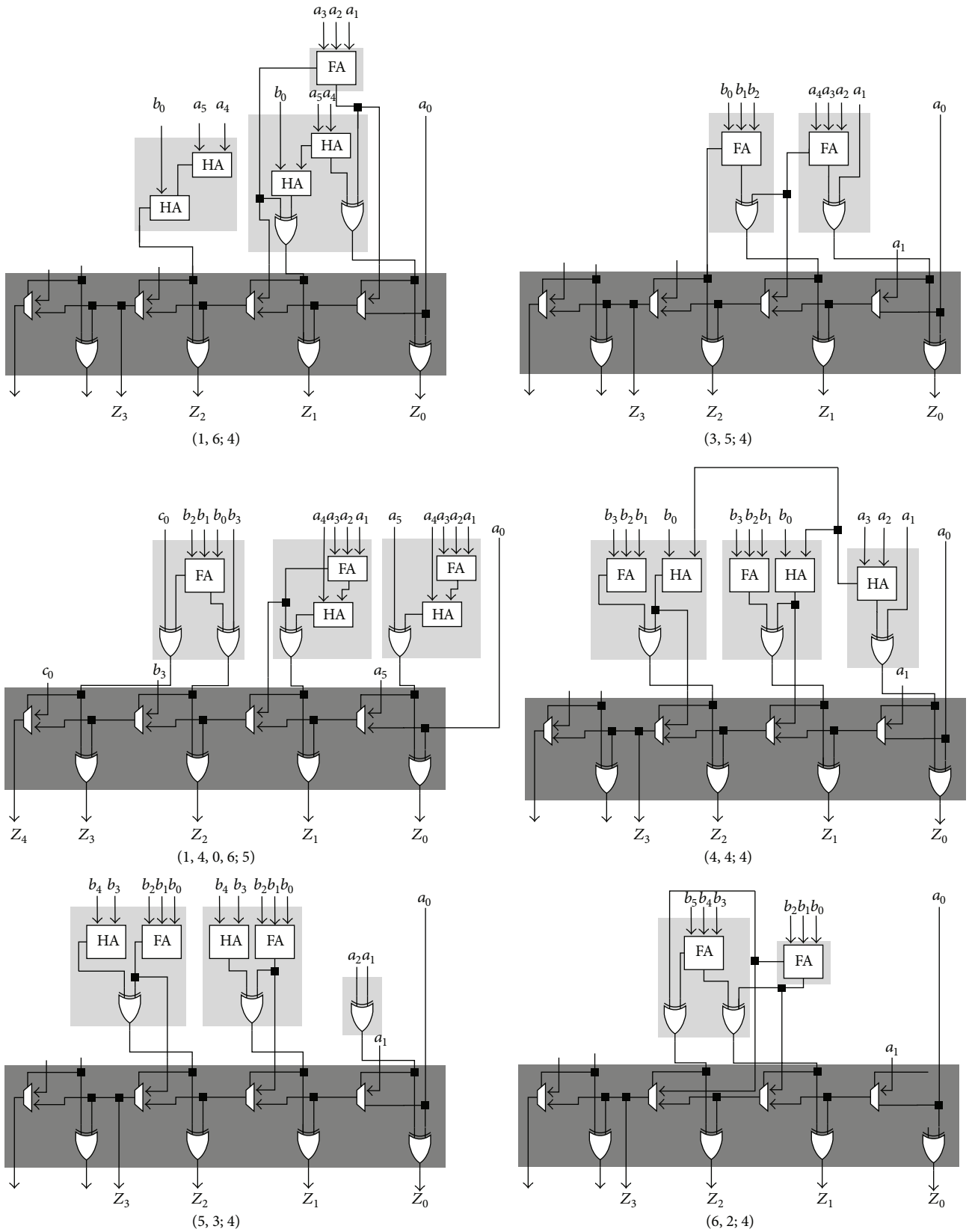


Figure 11: Area optimal mapping for different GPCs using proposed heuristic on Xilinx FPGAs.

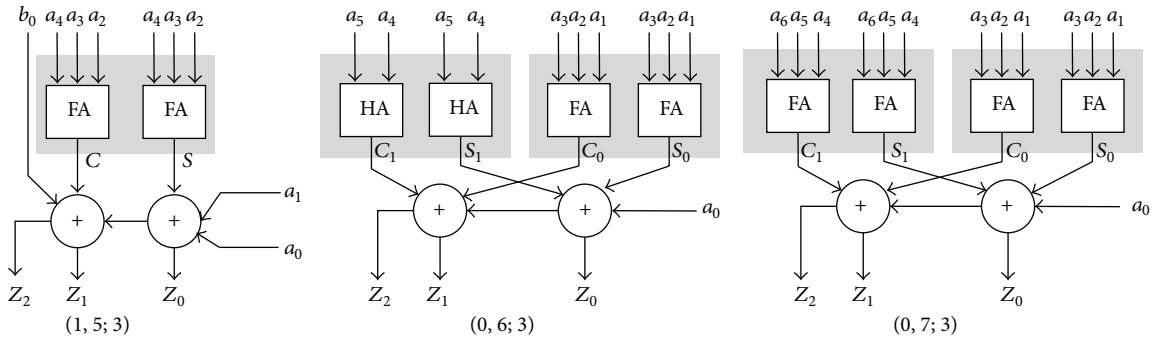


Figure 12: Area optimal mapping for different GPCs using proposed heuristic on Altera FPGAs.

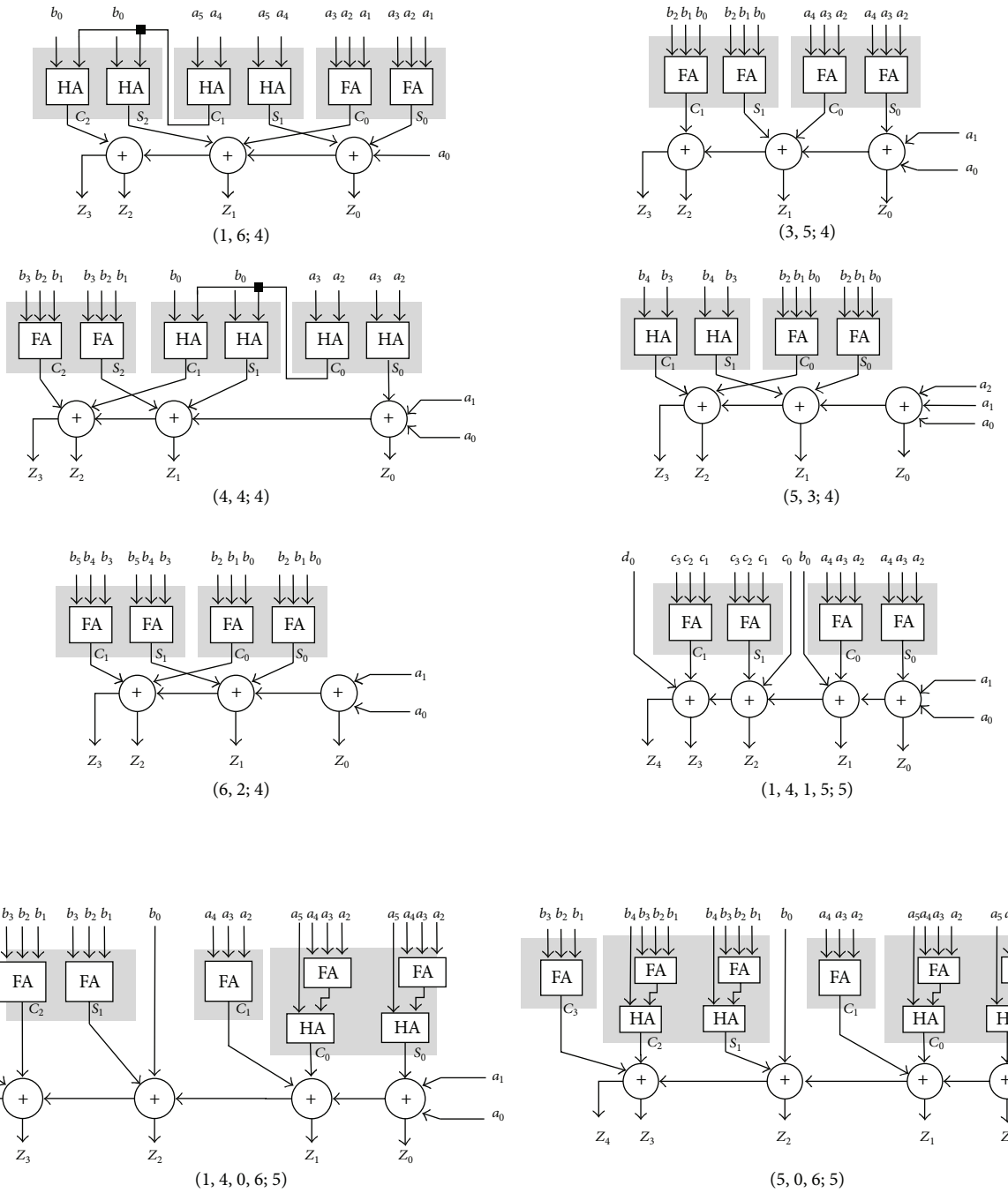


Figure 13: Area optimal mapping for different GPCs using proposed heuristic on Altera FPGAs.

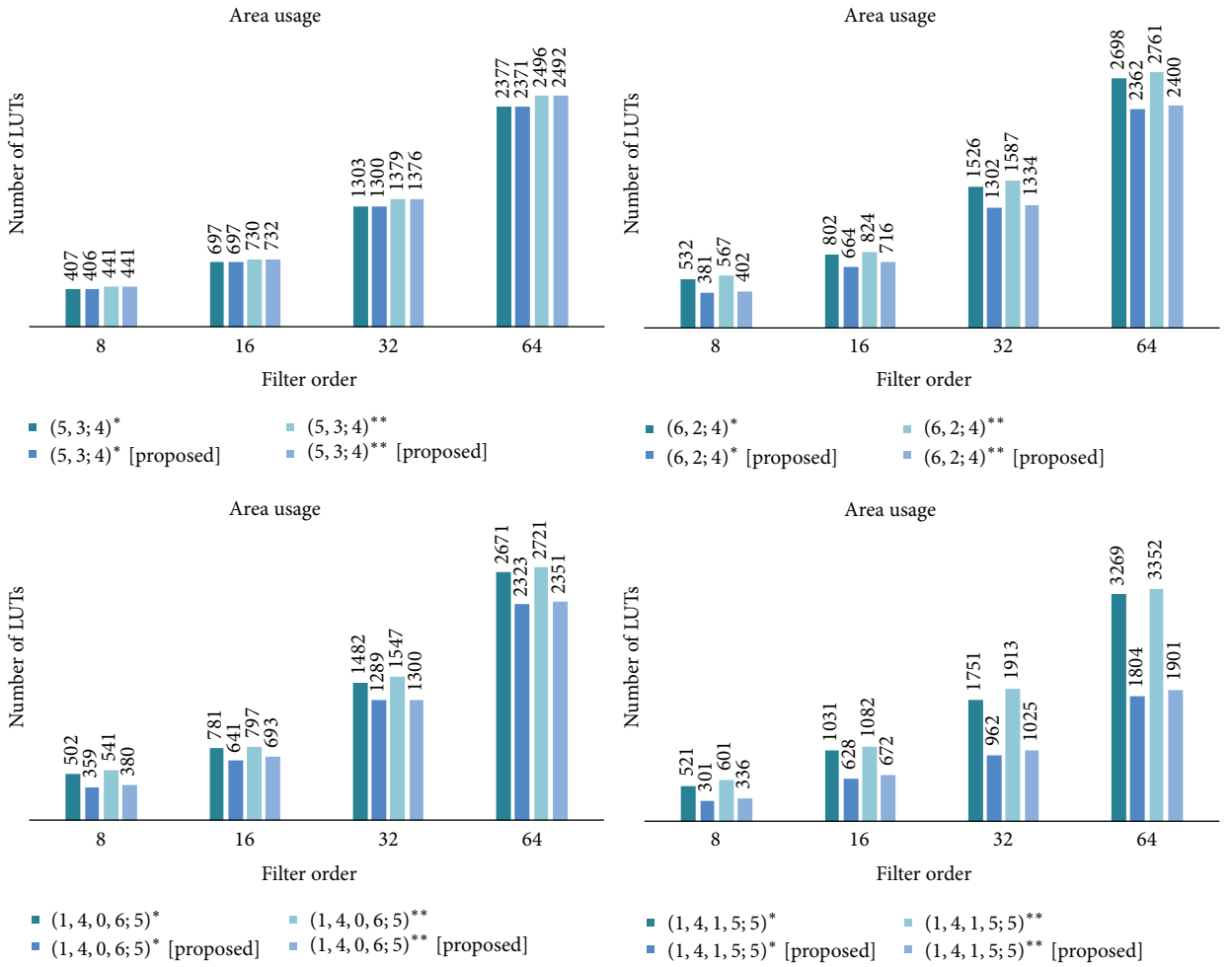


FIGURE 14: Area usage for filters based on different GPCs on Kintex-7 FPGA. $*$ with area as optimization goal. $**$ with speed as optimization goal.

A generalized parallel counter computes the sum of bits having different weights. A GPC is traditionally represented as a tuple $(K_{i-1}, K_{i-2} \cdots K_1, K_0; n)$, 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

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

$$M = \sum_{i=0}^{K-1} K_i 2^i \tag{2}$$

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

The efficiency of a GPC is measured by the number of reduced bits in relation to the hardware resources and is given by

$$E = \frac{(b_i - b_o)}{k}, \tag{3}$$

where b_i is the number of input bits; b_o is the number of output bits; and k is the number of LUTs used. 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 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*. 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. Each node implements a *local function*. A *global function* is implemented by connecting the logic implemented by individual nodes. 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 are measured by the depth and number of LUTs, respectively. The transformation of a Boolean network into targeted logic

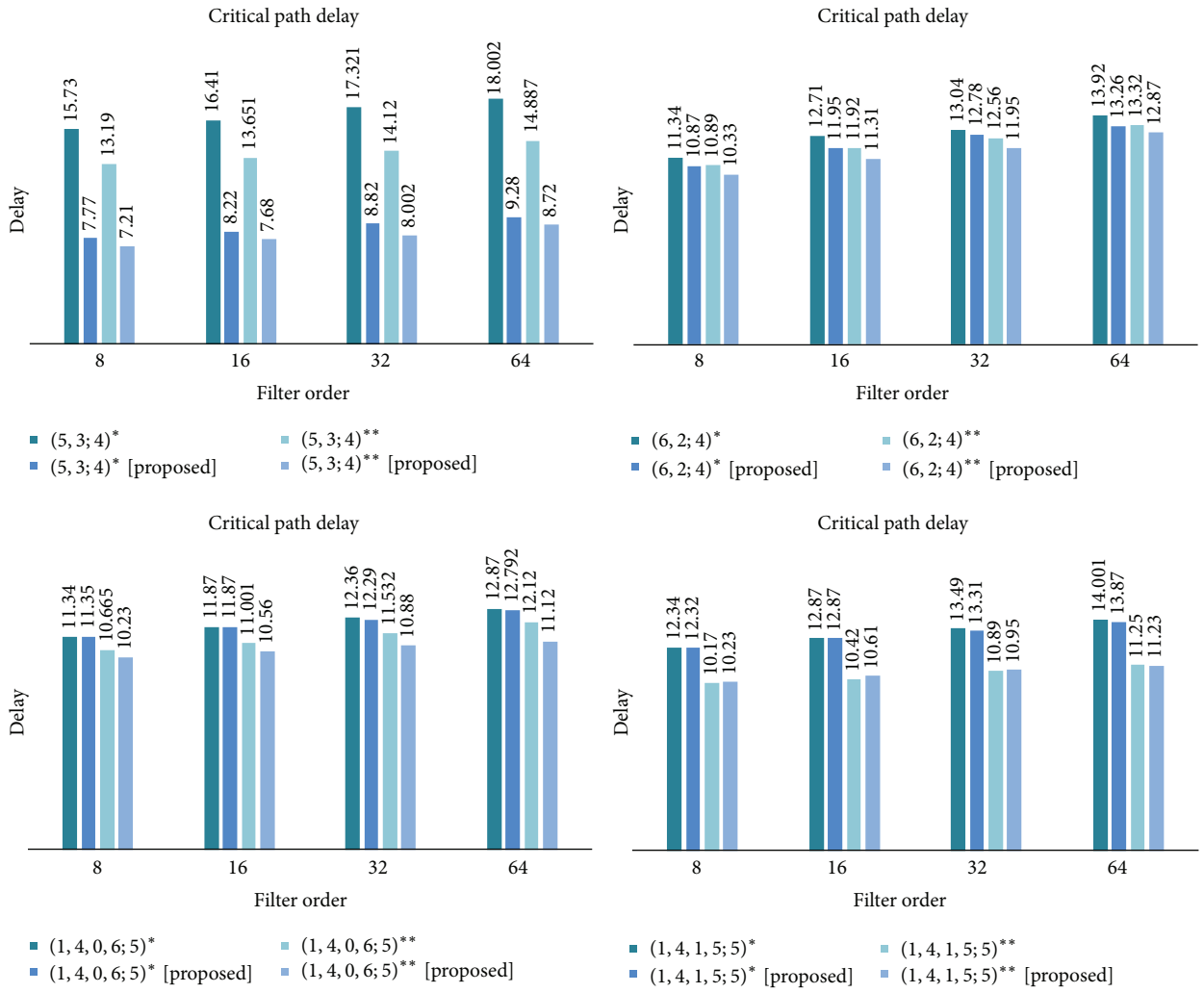


Figure 15: Critical path delay (in nS) for filters based on different GPCs on Kintex-7 FPGA.

elements gives the *circuit-netlist*. For FPGAs the targeted element is a k-LUT.

Altera Stratix IV and V FPGAs have Adaptive Logic Module (ALM) as the basic logic cell. The LUT resources within each ALM are divided into two adaptive LUTs (ALUTs). Normal operating mode uses a combination of these ALUTs within an ALM to implement functions with up to eight different inputs [17]. However, Altera supports specialized arithmetic and shared-arithmetic modes for each Stratix ALM for arithmetic extensive applications. In these modes each individual LUT can implement two 4-input functions with shared inputs. The arithmetic and shared-arithmetic modes also enable the use of fast carry chains that result in efficient implementation of different arithmetic functions. A typical Stratix IV ALM in arithmetic mode driving the carry chain is shown in the schematic of Figure 1.

Xilinx 5th-, 6th-, and 7th-generation FPGAs have 6-input LUTs as basic logic elements. Each logic slice provides combinatorial and synchronous resources, supporting 6-input LUTs, storage elements, function generators, arithmetic logic gates, and a fast carry chain in the form of CARRY4 primitive [18, 19]. The 6-input LUTs can be used in dual mode to implement two 5-input Boolean functions that share inputs as shown in Figure 2. The carry chain along with the logic gates performs fast arithmetic addition based operations in a slice. Each carry chain supports four-bit operand. The absence of a special arithmetic mode in Xilinx FPGAs sometimes results in inefficient arithmetic circuits in these devices. Also the use of carry chain primitive requires an additional XOR gate to be included at each CARRY4 input.

3. GPC Mapping Heuristic

This section describes the heuristic for efficiently mapping the GPC logic onto LUTs. The primary goal of the heuristic is to map the GPCs onto minimum possible LUTs. The heuristic involves *elimination* of the carry logic; *combination* of the redundant nodes; *covering* and *restructuring* of the Boolean network; and finally *reinsertion* of the carry logic. We explain the different steps involved in the heuristic by considering the mapping of GPC (1, 4, 1, 5; 5) on Xilinx 6-input LUTs. Conventional implementation has an efficiency of 1.5 and

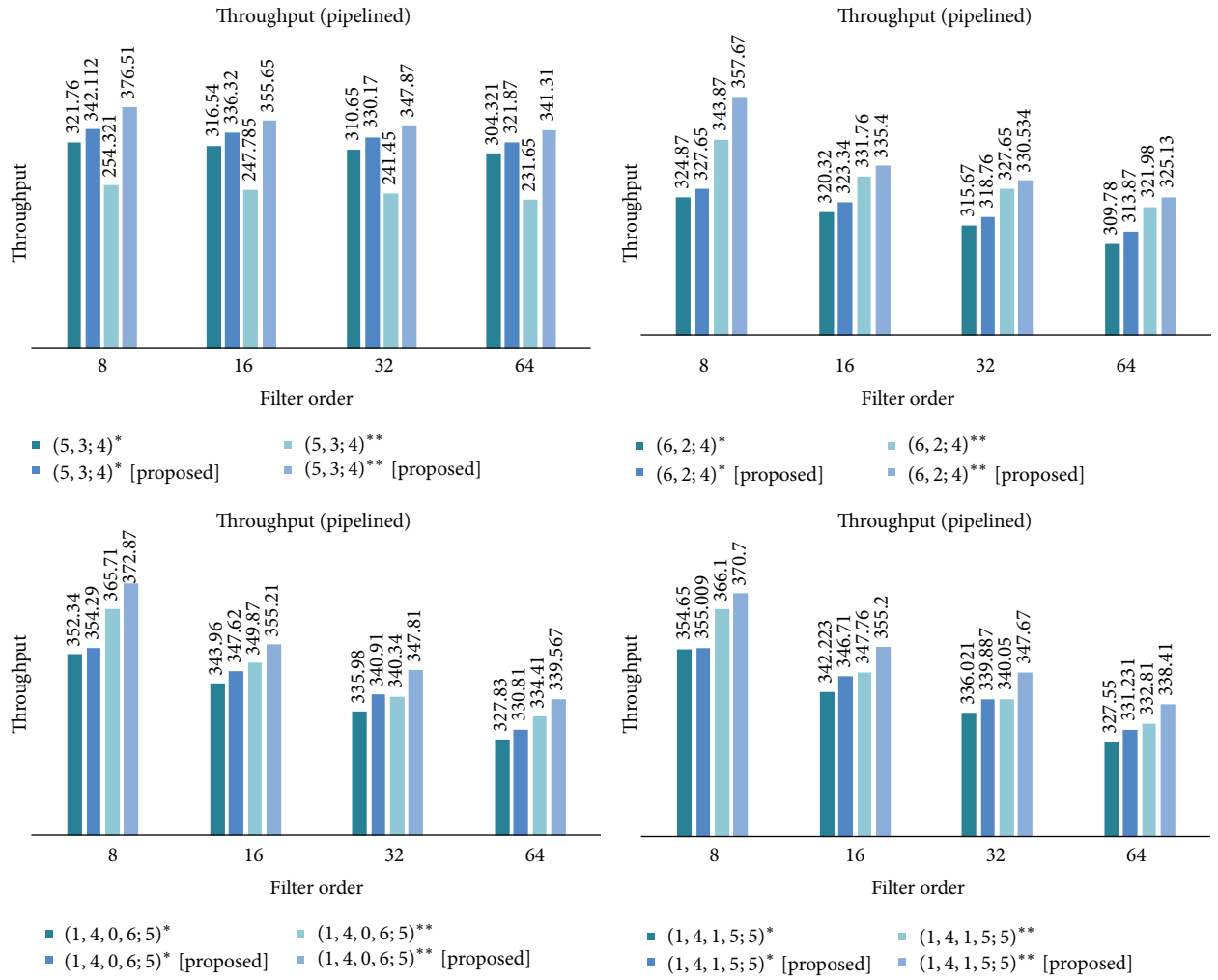


Figure 16: Throughput (in MHz) for filters based on different GPCs on Kintex-7 FPGA.

requires four LUTs and a CARRY4 primitive. The same steps can be used for Altera FPGAs; however, the availability of the special arithmetic mode makes the mapping process highly efficient and the combination step of the heuristic is only used in some complex cases.

Figure 3 shows the Boolean network for (1, 4, 1, 5; 5) GPC. The carry logic is shown by the shaded portion. All the primary inputs, primary outputs, and intermediate signals have been labeled. Figure 4 shows the conventional implementation using the fast carry chain [15].

3.1. Elimination. In the elimination step the logic implemented by the carry chain is eliminated from the GPC. For a (1, 4, 1, 5; 5) GPC the Boolean network after the elimination of the carry chain is shown in Figure 5. The GPC consists of four separate networks and each of these networks is mapped onto a separate LUT. Note that for Altera FPGAs the XOR gates in each LUT will be eliminated.

3.2. Combination. In this step the heuristic looks for redundant nodes for possible combination. The feasibility for combination is determined by the total number of outputs of the networks whose nodes are being combined. If the total number of outputs does not exceed two, then the nodes can be combined to eliminate the redundancy. For example, in the GPC network of Figure 5, network 1 and network 2 have a common full adder node with the same inputs. Each network has only one output so that the total number of outputs does not exceed 2. The redundant nodes from these two networks are thus combined, such that the two networks now share a common node. Similarly the full adder nodes in network 3 and network 4 share common inputs and thus can be combined into a single node. The Boolean network for a (1, 4, 1, 5; 5) GPC after node combination is shown in Figure 6.

3.3. Covering and Restructuring. The Boolean network after combination is traversed in a post-order depth-first fashion and the individual nodes are covered with suitable LUTs. Since we are targeting 6-input LUTs with dual output capability, the aim is to completely utilize the LUTs. For the combined network of Figure 6, the covering process is straightforward. Each of the combined networks has five

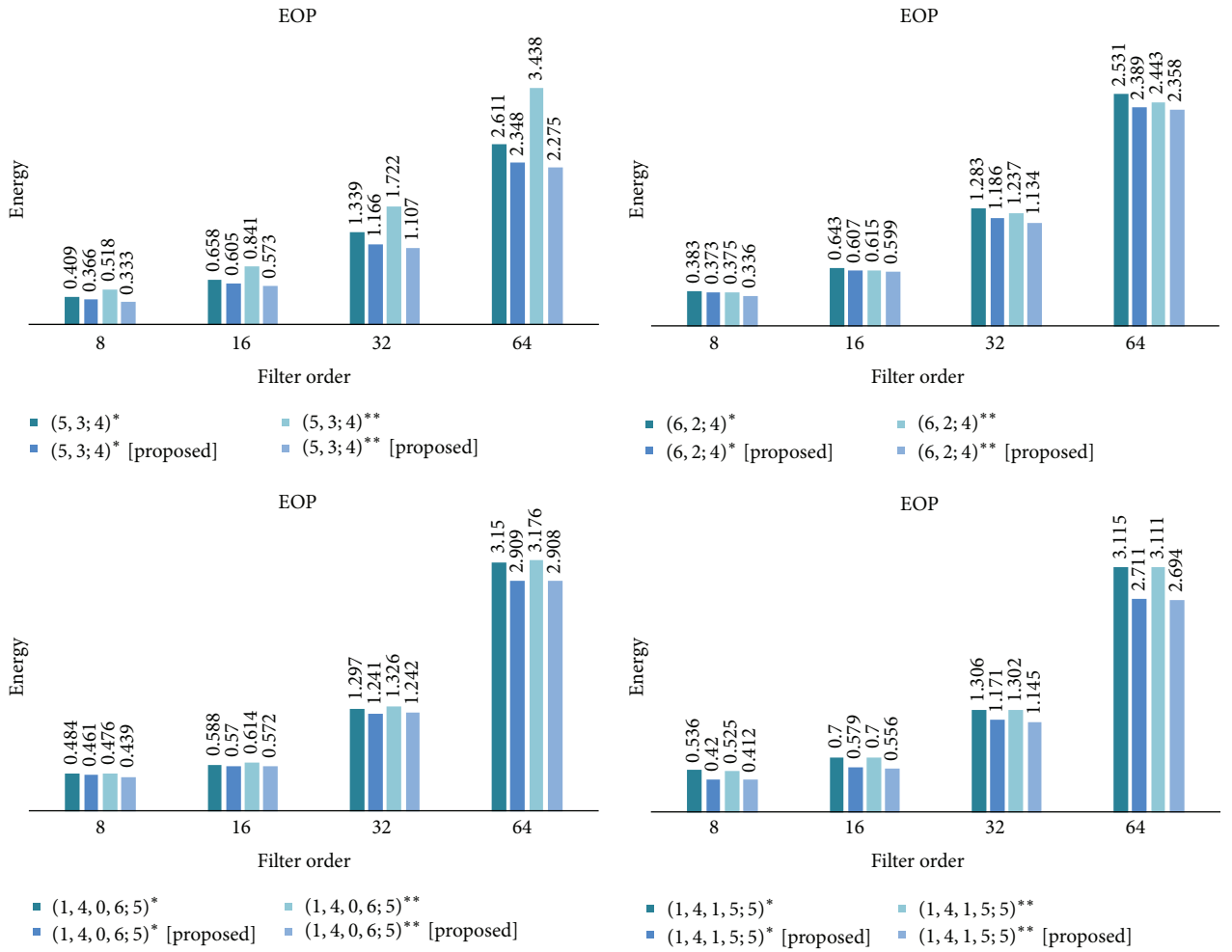


Figure 17: Energy per operation (in nJ) for filters based on different GPCs on Kintex-7 FPGA.

inputs and two outputs and will map optimally onto 6-input LUTs when operated in dual output mode as shown in Figure 7. However, in some cases the number of outputs per combined network may exceed two. In such case the heuristic restructures the combined network by either moving the nodes from one network to the adjacent one or duplicating the nodes to create additional networks. For example, the covering of (0, 6; 3) GPC is shown in Figure 8. The Boolean network obtained after carry chain elimination has five inputs and four outputs and would require two 6-input LUTs. This is possible if the individual nodes are duplicated to form two networks which are then mapped onto two separate LUTs. For Altera FPGAs the covering process is straightforward. Each of the full adders can be mapped onto single LUT operating in the arithmetic mode to obtain both sum (S) and carry (C) outputs.

3.4. Reinsertion. After the covering and restructuring, the carry logic is inserted back into the GPC structure by simply including the fast carry chain in the GPC network. For (1, 4, 1, 5; 5) GPC this is shown in Figure 9. The LUT count is reduced to two and there is a 100% increase in the efficiency.

Different GPCs from prior work were implemented using the proposed heuristic. An increase in efficiency was observed in most of the cases. The implemented circuits for different GPCs are shown in Figures 10 and 11 for Xilinx FPGAs and in Figures 12 and 13 for Altera FPGAs. A theoretical evaluation of different GPCs is listed in Table 1.

4. Implementation and Results

The main aim of the proposed heuristic is to improve the efficiency of the GPCs as defined in Section 2, since this is the performance metric used in the prior literature. From an implementation point of view this means that the mapped GPCs will utilize the underlying FPGA fabric more efficiently. Since dynamic power dissipation in FPGAs is a function of the amount of logic utilized, a reduction in the mapped logic will result in reduced dynamic power dissipation. Another useful side effect of area reduction is the reduction in the critical path which leads to increased speed. However, this occurs only in GPCs where the reduced logic is the part of the critical path. In our implementation, therefore, area and power are the primary metrics of concern.

Table 1: Comparison of different GPCs.

GPCs	Previous mappings			Mappings based on proposed heuristic (Xilinx)			Mappings based on proposed heuristic (Altera)		
	LUTs	Delay	Efficiency	LUTs	Delay	Efficiency	LUTs	Delay	Efficiency
GPCs from [9]									
(3; 2)	1	T_L[1]	1	1	T_L	1	1	T_L	1
(6; 3)	3	T_L	1	**2**	$T_L + 2T_{CC}$	**1.5**	**2**	$T_L + 2T_{CC}$	**1.5**
(1, 5; 3)	3	T_L	1	**1**	$T_L + 2T_{CC}$	**3**	**1**	$T_L + 2T_{CC}$	**3**
GPCs from [8]									
(6; 3)	4	$2T_L + T_R$[2] $+ 4T_{CC}$[3]	0.75	**2**	$\mathbf{T_L + 2T_{CC}}$	**1.5**	**2**	$\mathbf{T_L + 2T_{CC}}$	**1.5**
(1, 5; 3)	3	$T_L + 3T_{CC}$	1	**1**	$\mathbf{T_L + 2T_{CC}}$	**3**	**1**	$\mathbf{T_L + 2T_{CC}}$	**3**
(2, 3; 3)	3	$T_L + 3T_{CC}$	0.67	**1**	$\mathbf{T_L + 2T_{CC}}$	**2**	**0**	$\mathbf{2T_{CC}}$	—
(7; 3)	4	$2T_L + T_R + 4T_{CC}$	1	**2**	$\mathbf{2T_L + T_R + 2T_{CC}}$	**2**	**2**	$\mathbf{T_L + 2T_{CC}}$	**2**
(1, 6; 4)	4	$2T_L + T_R + 4T_{CC}$	0.75	**3**	$\mathbf{2T_L + T_R + 3T_{CC}}$	**1**	**3**	$\mathbf{2T_L + T_R + 3T_{CC}}$	**1**
(3, 5; 4)	4	$2T_L + T_R + 4T_{CC}$	1	**2**	$\mathbf{2T_L + T_R + 3T_{CC}}$	**2**	**2**	$\mathbf{T_L + 3T_{CC}}$	**2**
(4, 4; 4)	4	$2T_L + T_R + 4T_{CC}$	1	**3**	$\mathbf{2T_L + T_R + 3T_{CC}}$	**1.33**	**3**	$\mathbf{2T_L + T_R + 3T_{CC}}$	**1.33**
(5, 3; 4)	4	$2T_L + T_R + 4T_{CC}$	1	**3**	$\mathbf{T_L + 3T_{CC}}$	**1.33**	**2**	$\mathbf{T_L + 2T_{CC}}$	**2**
(6, 2; 4)	4	$2T_L + T_R + 4T_{CC}$	1	**2**	$\mathbf{2T_L + T_R + 3T_{CC}}$	**2**	**2**	$\mathbf{T_L + 2T_{CC}}$	**2**
GPCs from [15]									
(6; 3)	3	$2T_L + T_R + 3T_{CC}$	1	**2**	$\mathbf{T_L + 2T_{CC}}$	**1.5**	**2**	$\mathbf{T_L + 2T_{CC}}$	**1.5**
(1, 5; 3)	2	$T_L + 2T_{CC}$	1.5	**1**	$T_L + 2T_{CC}$	**3**	**1**	$\mathbf{T_L + 2T_{CC}}$	**3**
(2, 3; 3)	2	$T_L + 2T_{CC}$	1	**1**	$T_L + 2T_{CC}$	**2**	**0**	$\mathbf{2T_{CC}}$	—
(7; 3)	3	$2T_L + T_R + 3T_{CC}$	1.33	**2**	$\mathbf{2T_L + T_R + 2T_{CC}}$	**2**	**2**	$\mathbf{T_L + 2T_{CC}}$	**2**
(5, 3; 4)	3	$2T_L + T_R + 3T_{CC}$	1.33	3	$\mathbf{T_L + 3T_{CC}}$	1.33	**2**	$\mathbf{T_L + 2T_{CC}}$	**2**
(6, 2; 4)	3	$2T_L + T_R + 3T_{CC}$	1.33	**2**	$2T_L + T_R + 3T_{CC}$	**2**	**2**	$\mathbf{T_L + 2T_{CC}}$	**2**
(5, 0, 6; 5)	4	$T_L + 4T_{CC}$	1.5	4	$T_L + 4T_{CC}$	1.5	4	$T_L + 4T_{CC}$	1.5
(1, 4, 1, 5; 5)	4	$T_L + 4T_{CC}$	1.5	**2**	$T_L + 4T_{CC}$	**3**	**2**	$T_L + 4T_{CC}$	**3**
(1, 4, 0, 6; 5)	4	$T_L + 4T_{CC}$	1.5	**3**	$T_L + 4T_{CC}$	**2**	**3**	$T_L + 4T_{CC}$	**2**
(2, 0, 4, 5; 5)	4	$2T_L + T_R + 4T_{CC}$	1.5	4	$2T_L + T_R + 4T_{CC}$	1.5	4	$2T_L + T_R + 4T_{CC}$	1.5

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

Synthesis and implementation are done using XC5VLX30-2FF324 device from Virtex-5; XC6VLX75T-2FF484 device from Virtex-6; and XC7K70T-2FBG676 device from Kintex-7 FPGAs. The parameters considered are resources utilization (in terms of LUTs) and power delay product (PDP). Constraints relating to synthesis and implementation are duly provided and a complete timing closure is ensured in each case. Design entry is done using VHDL. However, instead of writing inferential codes, we have adopted an instantiation based coding strategy. This complicates the design entry but a better control over mapping is achieved. Dynamic timing analysis is done for each GPC to verify the functionality after Placement and Routing (PAR). This is done by applying different test vectors and checking for correct output vectors. Dynamic timing analysis gives information about the switching activity of the design, which is captured in the value charge dump (VCD) file. Apart from post-PAR timing analysis the functionality of the design is also verified by dumping the design on the Virtex-5 platform. Synthesis and implementation are carried out in Xilinx ISE 14.2 [20] with speed as the optimization goal. Power analysis is done using the Xpower analyzer tool. For power analysis switching activity is provided by the VCD file obtained during dynamic timing analysis. Similar test benches have been used to ensure a fair comparison.

For initial comparison we have implemented all the GPCs reported in prior work and compared their performance against the implementation based on the proposed heuristic. Table 2 provides a comparison of performance metrics for different GPCs. From Table 2 it is observed that most of the GPC mappings based on the proposed heuristic show an increase in area efficiency ranging from 33% to 100%. Since the underlying LUT fabric is utilized efficiently, there is a reduction in resources utilized. This results in reduced power

Table 2: Performance comparison of different GPCs on XC5VLX30-2FF324.

GPCs	XC5VLX30-2FF324				XC6VLX75T-2FF484				XC7K70T-2FBG676			
	Previous		Proposed		Previous		Proposed		Previous		Proposed	
	LUTs	PDP	LUTs	PDP	LUTs	PDP	LUTs	PDP	LUTs	PDP	LUTs	PDP
GPCs from [9]												
(3; 2)	1	0.008	1	0.008	1	0.021	1	0.021	1	0.007	1	0.007
(6; 3)	3	0.021	2	0.019	3	0.054	2	0.05	3	0.018	2	0.017
(1, 5; 3)	3	0.021	1	0.015	3	0.054	1	0.039	3	0.018	1	0.013
GPCs from [8]												
(6; 3)	4	0.043	2	0.019	4	0.114	2	0.05	4	0.039	2	0.017
(1, 5; 3)	3	0.024	1	0.015	3	0.061	1	0.039	3	0.021	1	0.013
(2, 3; 3)	3	0.024	1	0.015	3	0.061	1	0.039	3	0.021	1	0.013
(7; 3)	4	0.041	2	0.03	4	0.118	2	0.085	4	0.039	2	0.028
(1, 6; 4)	4	0.041	3	0.035	4	0.118	3	0.098	4	0.039	3	0.032
(3, 5; 4)	4	0.041	2	0.032	4	0.118	2	0.091	4	0.039	2	0.03
(4, 4; 4)	4	0.041	3	0.035	4	0.118	3	0.098	4	0.039	3	0.032
(5, 3; 4)	4	0.041	3	0.022	4	0.118	3	0.057	4	0.039	3	0.019
(6, 2; 4)	4	0.041	2	0.032	4	0.118	2	0.091	4	0.039	2	0.03
GPCs from [15]												
(6; 3)	3	0.041	2	0.019	3	0.105	3	0.05	3	0.036	3	0.017
(1, 5; 3)	2	0.021	1	0.015	2	0.053	2	0.039	2	0.018	2	0.013
(2, 3; 3)	2	0.021	1	0.015	2	0.053	2	0.039	2	0.018	2	0.013
(7; 3)	3	0.036	2	0.03	3	0.102	3	0.085	3	0.034	3	0.028
(5, 3; 4)	3	0.036	3	0.022	3	0.102	3	0.057	3	0.034	3	0.019
(6, 2; 4)	3	0.036	2	0.032	3	0.102	3	0.091	3	0.034	3	0.03
(5, 0, 6; 5)	4	0.026	4	0.026	4	0.063	4	0.063	4	0.022	4	0.021
(1, 4, 1, 5; 5)	4	0.026	2	0.021	4	0.063	4	0.051	4	0.022	4	0.017
(1, 4, 0, 6; 5)	4	0.033	3	0.03	4	0.066	4	0.061	4	0.023	4	0.02
(2, 0, 4, 5; 5)	4	0.043	4	0.043	4	0.079	4	0.079	4	0.027	4	0.027

Table 3: Performance comparison for proposed GPC-based filters and DSP, IP based filters.

Filter design	LUTs	Registers (pipelined)	DSP cores	Critical path (nS)	Throughput (pipelined) (MHz)	EOP (nJ)	ET (nJ/bit)
IP based	747	1883	0	14.701	303.366	0.765	0.003
DSP based	128	240	31	18.398	363.086	0.581	0.0022
Proposed (5, 3; 4)	732	1683	0	7.68	355.65	0.573	0.0022
Proposed (6, 2; 4)	716	1683	0	11.31	335.4	0.599	0.0023
Proposed (1, 4, 0, 6; 5)	693	1683	0	12.69	355.21	0.571	0.0022
Proposed (1, 4, 1, 5; 5)	672	1683	0	12.34	355.2	0.555	0.0021

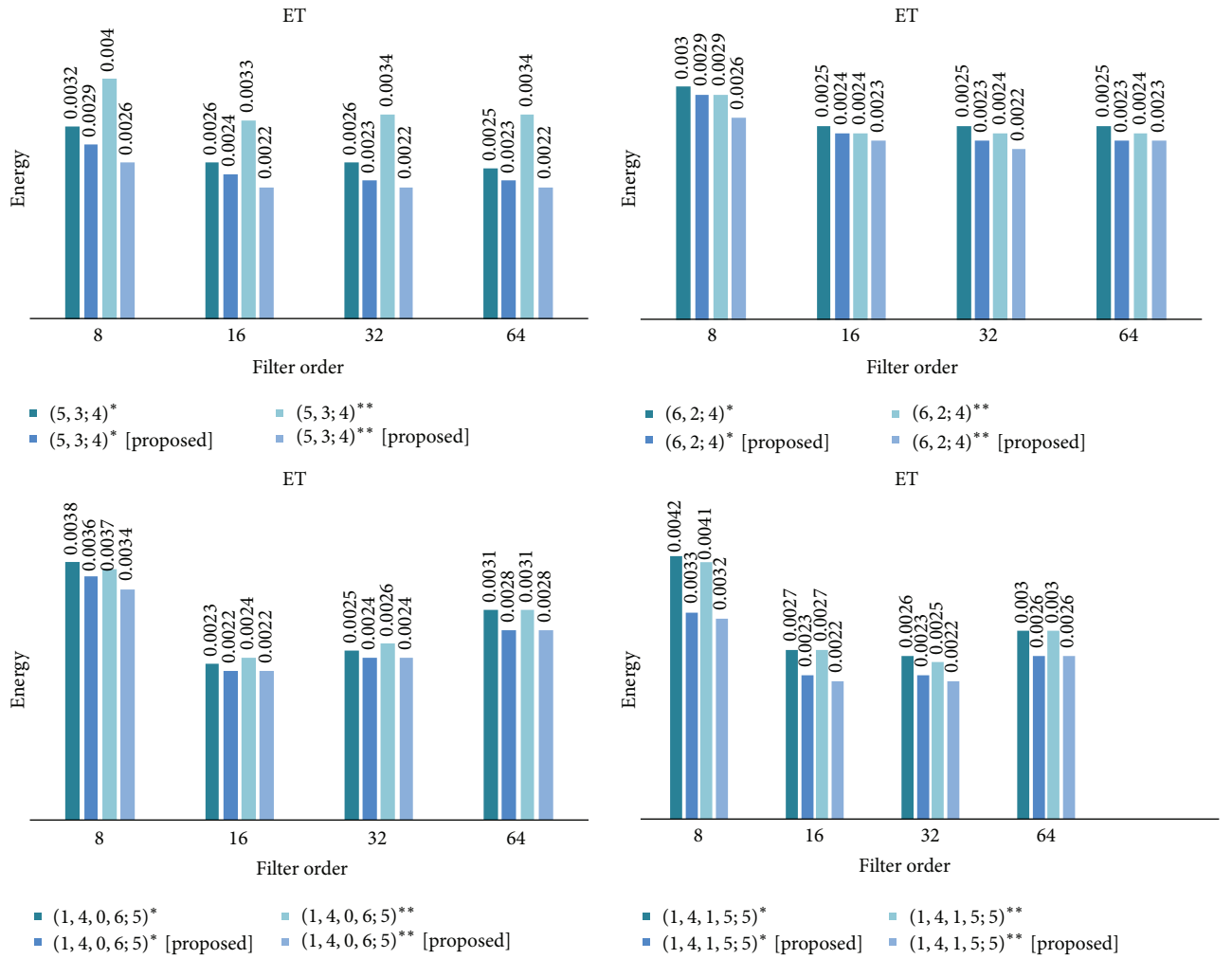


Figure 18: Energy throughput (in nJ/bit) for filters based on different GPCs on Kintex-7 FPGA.

dissipation. The critical path delay is also reduced in GPCs where the reduced logic is a part of the critical path. Our implementation shows an average reduction in power delay product by more than 20%.

We have also implemented FIR filters using different GPCs. The implementation is carried for different filter orders and for an operand word-length of 16 bits. The filter structures are based on fixed-point array multipliers and multioperand adders. Each of these units is constructed using the GPCs. Since FPGAs provide a high potential for pipelining we have used both combinational and pipelined versions of these individual units. For pure combinational structures critical path delay is used as a metric for speed and for pipelined designs throughput gives an idea about the speed of the structure. For high throughput DSP systems it is more appropriate to quantify the power efficiency through energy analysis. In our implementation we have used three energy related parameters for FIR systems. These include energy per operation (EOP), which is the average amount of energy required to compute one operation; energy throughput (ET) which is the energy dissipated for every output bit processed; and energy density (ED) which is the energy dissipated per LUT. We have used GPCs with maximum efficiencies from [8, 9, 15] and compared their performance against those based on the proposed heuristic. Figures 14, 15, 16, 17, 18, and 19 provide the performance comparison of the filters based on different GPCs implemented using the conventional methods and using our proposed heuristic. Note that the Xilinx synthesizer uses its own optimization strategies during the mapping process. In our implementation we have done separate analysis for area and speed. This is done by selecting the desired optimization goal prior to synthesis and implementation. The optimization effort in each case is selected to be high.

Finally, we have compared our filter implementation against that based on integrated DSP blocks and IP cores. DSP based filters have adders and multipliers constructed using DSP macros. For IP based filters the adder unit is the LogiCORE IP adder/Subtractor v 11.0 and the multiplier unit is the LogiCORE IP Multiplier v 11.2. Although these specialized inbuilt resources are highly optimized they do suffer from some disadvantages like fixed bit-width, limited number, and so forth [9, 21]. However, their biggest drawback is that they remain fixed in the FPGA fabric. This limits

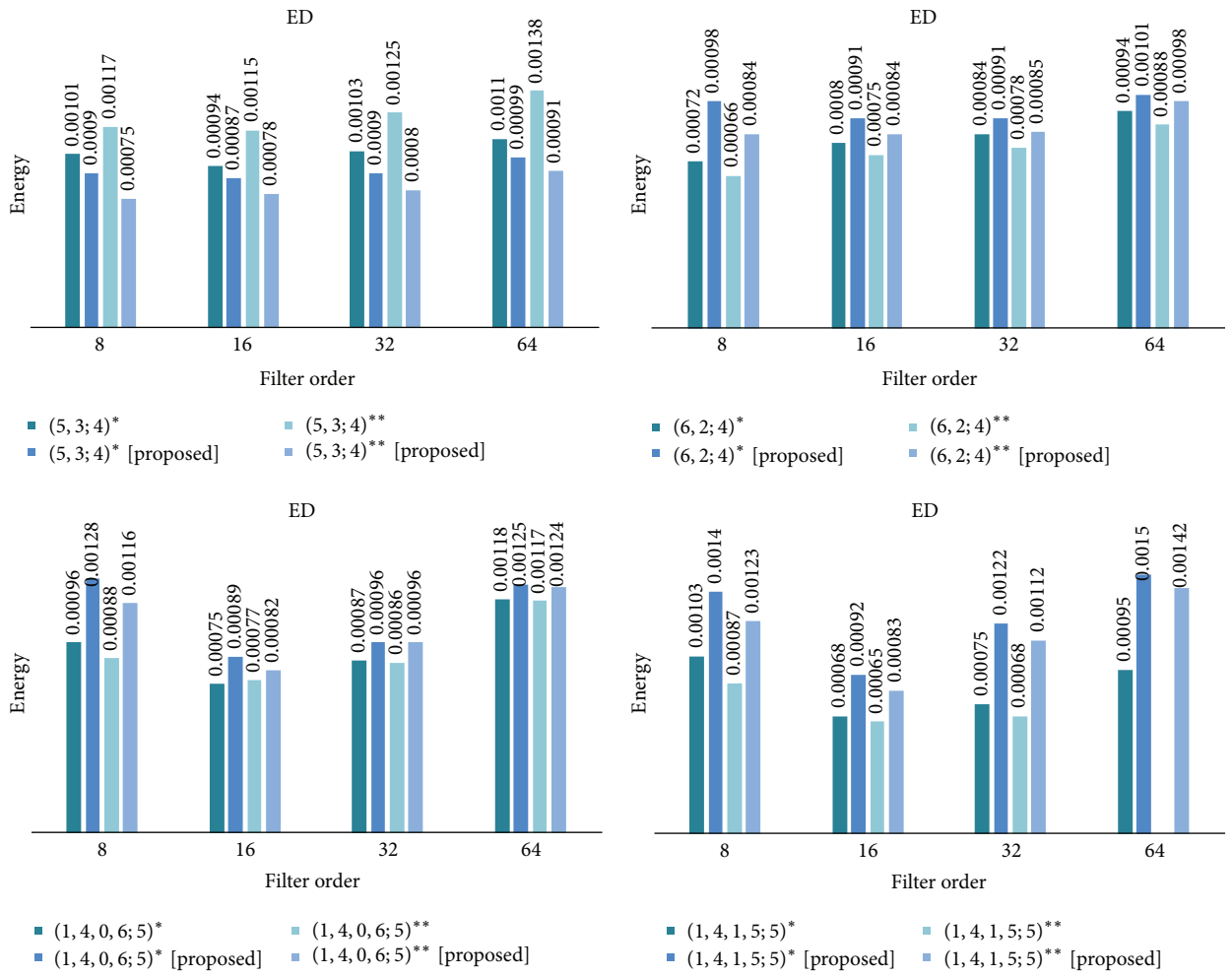


Figure 19: Energy density (in nJ/LUT) for filters based on different GPCs on Kintex-7 FPGA.

the ability of the synthesizer to alter their position during the PAR phase of the design cycle and sometimes the post-PAR performance may be highly degraded. In our analysis we have synthesized 16-tap direct form filters with input bit-width of 16 bits. The target platform is from Kintex-7 and the optimization goal is speed. The results provided in Table 3 show that the performance of our design is comparable to filters based on these specialized resources.

5. Conclusions

GPCs form an inherent part of high speed compressors. In this work we proposed a heuristic that mapped GPCs onto minimum possible LUTs by exploiting the improved logic handling capability of modern day FPGAs. A comparative analysis of our implementation against prior work showed a reduction in LUT count and the average power dissipated. This resulted in an increased compressing efficiency in most of the GPCs. Filter structures based on our modified GPCs show enhanced performance when compared to the conventional GPC-based filters. We also compared our results against filters based on specialized resources like DSP macros and IP cores. The results indicated that the performance of our design is comparable with these specialized resources. Our future work aims at efficiently pipelining the GPCs by eliminating the carry chain and using only a combination of LUTs and registers to implement the GPCs.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] S. Mirzaei, A. Hosangadi, and R. Kastner, "High speed FIR filter implementation using add and shift method," in *Proceedings of the International Conference on Computer Design*, pp. 1–4, San Jose, Calif, USA, October 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. 3, pp. 578–593, 2006.

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

[4] O. Kwon, K. Nowka, and E. E. Swartzlander Jr., "A 16-bit by 16-bit MAC design using fast 5:3 compressor cells," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 31, no. 2, pp. 77–89, 2002.

[5] H. Mora Mora, J. Mora Pascual, J. L. Sánchez Romero, and F. Pujol López, "Partial production reduction based on lookup tables," in *Proceedings of the International Conference on VLSI Design*, pp. 399–404, Hyderabad, India, January 2006.

[6] J. Põldre and K. Tammemäe, "Reconfigurable multiplier for Virtex FPGA family," in *Field Programmable Logic and Applications: International Workshop on Field-Programmable Logic and Applications, Glasgow, UK*, vol. 1673 of *Lecture Notes in Computer Science*, pp. 359–364, Springer, Berlin, Germany, 1999.

[7] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, vol. 13, no. 1, pp. 14–17, 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, 2011.

[9] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Efficient synthesis of compressor trees on FPGAs," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '08)*, pp. 138–143, IEEE, Seoul, Republic of Korea, March 2008.

[10] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Exploiting fast carry-chains of FPGAs for designing compressor trees," in *Proceedings of the 19th International Conference on Field Programmable Logic and Applications*, pp. 242–249, Prague, Czech Republic, August 2009.

[11] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Improving synthesis of compressor trees on FPGAs via integer linear programming," in *Proceedings of the Design, Automation and Test in Europe (DATE '08)*, pp. 1256–1261, IEEE, Munich, Germany, March 2008.

[12] T. Matsunaga, S. Kimura, and Y. Matsunaga, "Power and delay aware synthesis of multi-operand adders targeting LUT-based FPGAs," in *Proceedings of the 17th IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED '11)*, pp. 217–222, Fukuoka, Japan, August 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. 94, no. 12, pp. 2579–2586, 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, 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 (MBMV '14)*, Böblingen, Germany, March 2014.

[16] M. Kumm and P. Zipf, "Pipelined compressor tree optimization using integer linear programming," in *Proceedings of the 24th International Conference on Field Programmable Logic and Applications*, pp. 1–8, IEEE, Munich, Germany, September 2014.

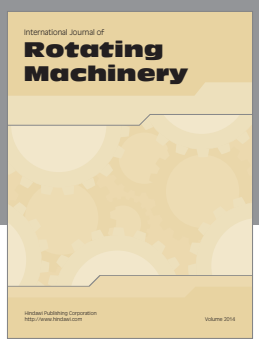
[17] Altera Corporation, *Stratix-IV Device Handbook*, vol. 1, Altera Corporation, 2015.

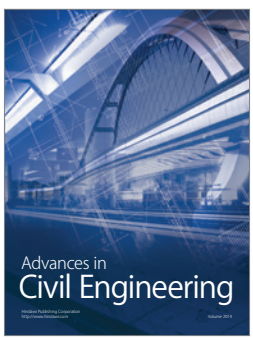
[18] Xilinx Corporation, *Virtex-5 Family Overview LX, LXT, and SXT Platforms*, Xilinx Inc, San Jose, Calif, USA, 2010.

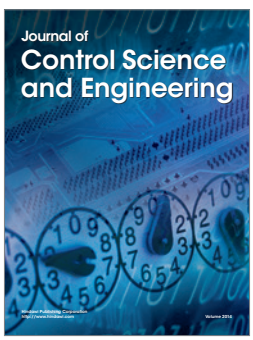
[19] Xilinx Corporation, *Virtex-6 FPGA Data Sheet*, Xilinx Inc, San Jose, Calif, USA, 2010.

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

[21] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.







Hindawi
Submit your manuscripts at
http://www.hindawi.com