Enumeration of Genaralized Parallel Counters for Multi-Input Adder Synthesis for FPGAs

Mugi Noda
Graduate School of Science and Technology
Kwansei Gakuin University
1 Gakuin Uegahara, Sanda, Hyogo, 669-1330, JAPAN
mugi-noda@kwansei.ac.jp

Nagisa Ishiura
School of Engineering
Kwansei Gakuin University
1 Gakuin Uegahara, Sanda, Hyogo, 669-1330, JAPAN
ishiura@kwansei.ac.jp

***Abstract*—This paper proposes a method for exhaustive search of efficient FPGA implementations of generalized parallel counters (GPCs) that have not yet been discovered. Multi-input addition is the core operation in multiplier circuits and multiply-accumulate circuits. A known method to construct efficient multi-input adders on FPGAs involves building trees of full adders extended into GPCs. To date, three types of GPCs that can be implemented in a single slice of Xilinx 7 series have been identified. This paper attempts to find other GPCs that can be implemented in a single slice exhaustively. We enumerate possible GPC input-output specifications with output of five bits or less and determine the connections from inputs to LUTs and the truth tables of LUTs if they exist. Running the proposed search method resulted in the discovery of five new GPCs: (1,2,6;4), (4,2,5;5), (1,2,4,4;5), (1,3,1,6;5), and (1,3,3,4;5). Furthermore, using these GPCs to construct multiplier circuits and multi-input adder circuits resulted in a reduction in the number of slices for two circuits and the number of stages for two other circuits.**

***Index Terms*—Digital Arithmetic, FPGA, Compressor Tree, Generalized Parallel Counter**

I. Introduction

Multi-input adder circuits have long been used in various arithmetic circuits, such as multipliers and multiply-accumulate units [1] [2]. Recently, they have become increasingly important as core components for hardware acceleration of neural networks [3].

For efficient implementation of multi-input adders, classical methods include carry-save adder trees using 3-input 2-output full adders [1] [2] and trees of redundant binary adders [4]. These methods enable the addition of n integers of n bits with a latency of $O(\log n)$ stages.

These methods assume circuit components are full adders. However, on LUT (Look-Up Table) based FPGAs, circuits using 3-input 2-output full adders may not efficiently utilize 5 to 6-input LUTs.

Thus, for FPGA implementation of multi-input adders, extended adders like 6-input 3-output, or further extended ones allowing inputs with weights of powers of 2, are used. These adders are called generalized parallel counters (GPCs), and trees of GPCs for multi-input addition are known as compressor trees [5] [7] [8] [9] [10]. While the number of stages is still $O(\log n)$, compressor trees generally result in smaller delays and sizes on FPGAs compared to those based on full adders.

Various GPCs have been designed to match FPGA architecture. Particularly for FPGAs with embedded carry look-ahead circuits (carry logic), it is important to utilize these features. Additionally, in FPGAs where a "slice" or "logic block" consisting of several LUTs and carry logic is a design unit. Fitting a GPC into a single slice would enhance overall FPGA efficiency.

For Xilinx 7 series FPGAs, three types of GPCs can be implemented in a single slice [8] [10], and 21 variations from these are used in compressor trees. However, these were designed heuristically, and it was unknown if other GPCs could be implemented in a single slice.

This paper aims to discover more efficient GPCs by exhaustively exploring those that can be implemented in a single slice. We determine whether input connections to LUTs and LUT truth tables exist for all input-output specifications of GPCs with up to 5-bit outputs.

Using the proposed method, we discovered five new GPCs that can be implemented in a single slice. Constructing compressor trees for 8 to 32-bit multipliers and multi-input adders with these GPCs confirmed that they help reduce circuit size and delay.

II. FPGA Implementation of Generalized Parallel Counters

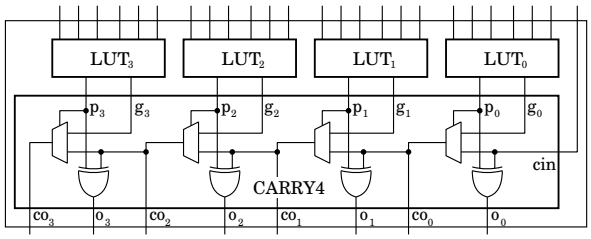
A. LUT-Based FPGA

Look-up Table (LUT) based field-programmable gate arrays (FPGAs) construct logic circuits using LUTs and programmable interconnects. LUTs function as logic gates that implement any logical function by storing truth tables in memory. Additionally, some FPGAs have embedded carry look-ahead logic (carry logic) for efficient addition and subtraction.

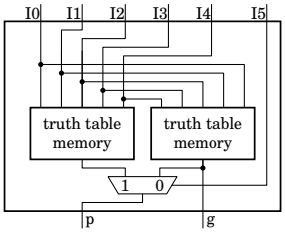
This paper assumes an FPGA model of the Xilinx 7 series [11] shown in Fig. 1. A "slice" in these FPGAs consists of four 6-input, 2-output LUTs and a 4-bit carry logic (CARRY4). Each LUT output is connected to the carry generation signal g_k and propagation signal p_k of the CARRY4.

The LUT consists of two truth table memories and a multiplexer, as shown in Fig. 1b. Each truth table memory can implement any 5-input logical function. When the input to port I5 is set to 1, the LUT computes any 5-input, 2-output function. When I5 is used as a variable, the p output can realize

979-8-3503-7877-1/24/$31.00 ©2024 IEEE



(a) Structure of "slice"



(b) Structure of LUT

Fig. 1: Model of LUT-based FPGA

any 6-input function, and the g output realizes the logical function of p with I5 fixed to 0.

B. Generalized Parallel Counter

Efficient methods for constructing multi-input adder circuits include Wallace trees [1] and Dadda trees [2], which use 3-input 2-output full adders as the basic element. However, for FPGA implementations, trees of full adders do not fit well into 5 to 6 input LUTs and fail to utilize carry logic effectively, resulting in inefficiency.

To address this, adders extended to 6-input, 3-output configurations or those allowing weights of powers of 2 in addition to weight 1 have been proposed. These extended adders are called *generalized parallel counters (GPCs)*, and trees using GPCs are known as *compressor trees*.

The input-output specifications of a GPC are represented as $(p_{q-2}, p_{q-3}, \ldots, p_0; q)$. This circuit outputs the sum of p_k bits with a weight of 2^k as a q-bit value. For example, a full adder (3;2) has three inputs of weight 1 and outputs their sum as a 2-bit value. Similarly, (1,3,5;4) has inputs of weights 1, 2, and 4, with 5, 3, and 1 inputs respectively, and outputs their sum as a 4-bit value.

For the FPGA model in Fig. 1, efforts are being made to find GPCs that can be implemented in one slice. So far, three GPC designs (1,1,7;4) [10], (1,3,5;4) [10], and (1,3,2,5;5) [8] are known. These GPCs, along with reduced input-output versions and their combinations, result in 21 types used in compressor trees [6] [7] [8] [9] [10] [12].

Compressor trees using these GPCs as basic elements are more complex than those based on full adders. To build compressor trees with fewer stages and reduced sizes, various methods have been proposed, which includes formulations for integer linear programming and heuristic algorithms [5] [8] [9] [10].

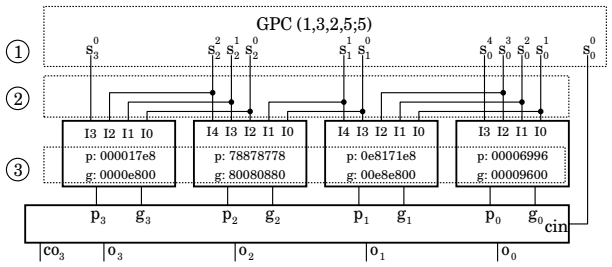


Fig. 2: GPC (1,3,2,5;5) [8]

III. Enumeration of GPCs

A. GPC Enumeration Problem and Algorithm Outline

The GPCs mentioned in the previous section were discovered heuristically, and there is no guarantee that they represent all possible configurations. In this paper, we aim to discover more efficient GPCs by addressing the problem of finding all GPCs that can be implemented within a single slice in the FPGA model discussed in the previous chapter.

We enumerate all GPC specifications with outputs of q bits where $q \leq 5$ (① in Fig. 2) and determine the implementability of each. First, we enumerate all possible ways to connect the inputs to the LUTs (②), and for each configuration, we check whether there exists an assignment of the LUT truth tables that yields the desired GPC output (③).

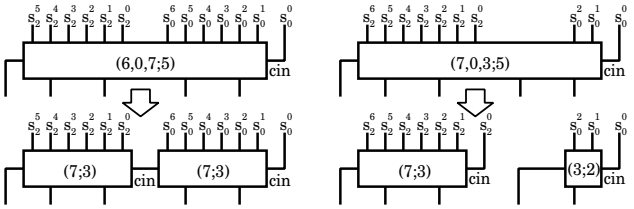
B. Enumeration of GPC Specifications

We generate all GPC specifications of the form $(p_{q-2}, p_{q-3}, \ldots, p_0; q)$ where $q \leq 5$ and $2^{q-1} \leq \sum_{k=0}^{3} 2^k p_k < 2^q$. The generation conducts in lexicographical order, starting from $(0,0,0,0;0)$, $(0,0,0,1;1)$, $\ldots$, $(0,0,0,7;3)$, $(0,0,1,0;2)$, $\ldots$, up to $(3,1,1,1;5)$.

Here, we refer to the k-th LUT from the right as the LUT for the k-th digit, denoted as LUT_k. To reflect the value of the input at the k-th digit in the output of the k-th digit, all inputs for the k-th digit must be connected to LUT_k, as shown in Fig. 2. Since the maximum number of inputs for the LUTs considered in this paper is 6, it is generally sufficient to consider a maximum of 6 inputs for each digit p_k of a GPC. However, as shown by the input s_0^0 in Fig. 2, the "cin" input of the carry chain allows an additional input in the least significant digit. Additionally, by splitting the GPCs, as described in the next section, it is possible to handle up to 7 inputs even in digits other than the least significant digit. Therefore, in this paper, we generate GPC specifications for each digit k within the range $0 \leq p_k \leq 7$.

C. Splitting of GPCs

GPCs can sometimes be split, and determining the feasibility of each can significantly reduce the computational cost of subsequent processes. Additionally, splitting the GPC allows the "cin" input to be used in digits beyond the least significant place, enabling p_k to be set to 7.



(a) Splitting (6,0,7;5) (b) Splitting (7,0,3;5)

Fig. 3: Splitting GPC

When the carry from the k-th to the $(k+1)$-th digit is at most 1, it can propagate using only the carry logic. Such a GPC can be divided into two parts at the k-th digit. For example, the GPC (6,0,7;5) has a maximum carry of 1 from the second to the third digit, so it can be split into two (7;3) GPCs, as shown in Fig. 3a.

If no carry occurs from the k-th to the $(k+1)$-th digit, a split using the unused "cin" input can be made. For example, in the GPC (7,0,3;5) in Fig. 3b, no carry occurs from the second to the third digit. Splitting between these digits allows the unused "cin" to provide an input to the second digit, increasing the maximum inputs for the third digit to 7.

D. Enumeration of Connections from Inputs and LUTs

For the GPC specifications obtained in the previous section, we enumerate all possible connections from inputs to LUTs (② in Fig. 2). In this process, we first treat all input ports of the LUTs as symmetric, and at the end of this section, we discuss the handling of asymmetric ports.

As shown in Fig. 2, using the "cin" input of the carry logic always allows for one additional input in the least significant digit. Therefore, in this section, we exclude this one input and only process the cases where all digits k satisfy $p_k \leq 6$.

Let S_k be the set of inputs for the k-th digit of GPC $(p_3, p_2, p_1, p_0; q)$. In the example shown in Fig. 2, $S_0 = \{s_0^4, s_0^3, s_0^2, s_0^1, s_0^0\}$, $S_1 = \{s_1^1, s_1^0\}$, $S_2 = \{s_2^2, s_2^1, s_2^0\}$, and $S_3 = \{s_3^0\}$. Let L_k be the set of inputs assigned to the k-th digit's LUT (LUT_k). In Fig. 2, $L_0 = \{s_0^4, s_0^3, s_0^2, s_0^1\}$, $L_1 = \{s_1^1, s_1^0, s_0^3, s_0^2, s_0^1\}$, $L_2 = \{s_2^2, s_2^1, s_2^0, s_1^1, s_1^0\}$, and $L_3 = \{s_3^0, s_2^2, s_2^1, s_2^0\}$. We enumerate all possible combinations (L_3, L_2, L_1, L_0) that satisfy the following conditions:

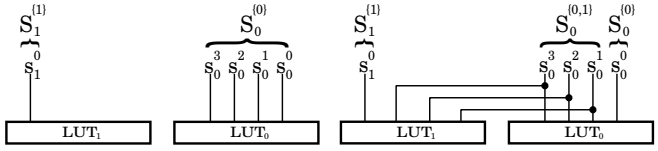
- $|L_k| \leq 6$: The number of LUT inputs is at most 6.
- $S_k \subseteq L_k$: All inputs of the k-th digit of the GPC are connected to the k-th digit's LUT.
- $L_k \subseteq S_k \cup S_{k-1} \cup \ldots \cup S_0$: The inputs to the k-th digit's LUT are only from the inputs of the GPC up to the k-th digit.

To generate the combinations (L_3, L_2, L_1, L_0), we determine L_k in ascending order of k.

When $k = 0$, there are no lower digits, so $L_0 = S_0$.

For $k > 0$, starting with $L_k = S_k$, we try all possible assignments for the remaining ports of LUT_k.

In this process, we reduce the number of combinations by using input symmetry. Let S_k^T denote the set of elements from S_k that are inputs to the set of LUTs T. For example, $S_0^{\{0,1,2\}}$ represents the elements of S_0 that are inputs only to LUT_0, LUT_1, and LUT_2. The inputs contained in S_k^T are symmetric with respect to the outputs of the LUTs, so there is no need to distinguish between them.



(a) Initial state (b) After assignment

Fig. 4: Input group partitioning for assigning inputs to LUTs

An example of input assignment is shown in Fig. 4. In the initial state (Fig. 4a), $S_0^{\{0\}} = S_0 = \{s_0^3, s_0^2, s_0^1, s_0^0\}$ and $S_1^{\{1\}} = S_1 = \{s_1^0\}$. When connecting three inputs from S_0 to LUT_1, there is only one way to choose 3 inputs because the elements within $S_0^{\{0\}}$ are indistinguishable. As a result of this connection, $S_0^{\{0\}}$ is split into $S_0^{\{0\}}$ and $S_0^{\{0,1\}}$ (Fig. 4b).

The inputs connected to LUT_2 are selected from among $S_1^{\{1\}}, S_0^{\{0\}}$, and $S_0^{\{0,1\}}$. Since the inputs within $S_0^{\{0,1\}}$ are symmetric, there is no need to distinguish between them.

When connecting 6 inputs to an LUT (using the I5 port in Fig. 1b), one port is not symmetric. To handle this, we first assign all 6 inputs and then determine which input will be connected to the asymmetric port. The group is then split for the selected input, and the same process is applied.

E. Determination of LUT Truth Tables

Given a GPC input-output specification $(p_3, p_2, p_1, p_0; q)$ and the connections to the LUTs (L_3, L_2, L_1, L_0), we determine whether there exists a truth table for the LUTs that realizes such a GPC (③ in Fig. 2).

This is done by determining the output values p_k and g_k for each LUT_k so that the desired output values are obtained for all possible input values to the GPC. Note that there can be multiple combinations of LUT output values that achieve a given GPC output. This multiplicity is a result of the logic implementation of the carry logic in the FPGA model. Also note that inconsistencies may arise in the outputs of some LUTs for certain inputs. In such cases, some of the combinations may need to be eliminated.

For example, in Fig. 5, when the input is $(0, 0, 0, 1, 1)$ as shown in ①, there are three possible combinations for (p_1, g_1, p_0, g_0) to produce the output $(0, 1, 0)$: $(0, 0, 0, 1)$, $(1, 0, 0, 0)$, and $(1, 1, 0, 0)$.

Some of the candidates may not be feasible due to the constraints of the given LUT connections, and such infeasible candidates are removed. In Fig. 5, ② shows the case when the input is $(0, 0, 0, 1, 0)$. The framed input to LUT_1 is $(0, 0, 0, 1)$ in both ① and ②, so LUT_1 must output the same value for both inputs. Consequently, the outputs $(1, 0)$ and $(1, 1)$ are eliminated from the candidates, leaving only $(0, 0)$ as a valid output for this input.

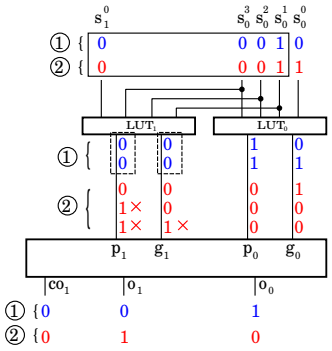


Fig. 5: Determining feasible outputs of LUTs

TABLE I: Set of GPCs used for experiments

(1;1)	(1,3,5;4)	(1,3,4,3;5)	(2,1,3,5;5)	(1,4,0,6;5)	(1,4,1,5;5)
(1,4,2,3;5)	(1,5;3)	(2,1,5;4)	(1,1,7;4)	(2,1,1,6;5)	(1,1,6,3;5)
(3;2)	(2,3;3)	(2,2,3;4)	(2,2,2,3;5)	(7;3)	(2,0,7;4)
(6,0,6;5)	(6,1,5;5)	(6,2,3;5)	(1,3,2,5;5)		
(7,0,3;5)	(1,2,6;5)	(2,1,2,6;5)	(1,2,5,3;5)	(4,3,5;5)	(4,4;4)
(1,2,4,4;5)	(1,3,1,6;5)	(1,3,3,4;5)			

When using the asymmetric input of the LUT (I5 in Fig. 1b), additional constraints are imposed on the LUT truth table. When I5 $=$ 0, the p output must use the same truth table memory as the g output, which requires that $p = q$. Candidates are eliminated based on this constraint.

After these operations, if there is at least one candidate for the output values of the LUT for each input value of the GPC, then a valid truth table can be defined. Otherwise, the GPC cannot be realized with that LUT connection.

IV. Experimental Result

The GPC search program based on the proposed method has been implemented in Rust[1]. The search took about 100 seconds on a single thread of a Ryzen 7 PRO 6850U processor.

As a result, five new GPCs were discovered that can be implemented in one slice of the Xilinx 7 series: (1,2,6;4), (4,2,5;5), (1,2,4,4;5), (1,3,1,6;5), and (1,3,3,4;5). Fig. 6 shows the implementations of these GPCs.

Using the discovered GPCs, we constructed GPC trees for n-bit multipliers and circuits adding n integers of n bits. The GPCs used are listed in Table I. The upper part lists existing GPCs, and the lower part shows the newly discovered GPCs and their weaker variants. Using the formulation by [9], a tree optimization program was run on 24 threads of a Ryzen 9 3900X processor using IBM ILOG CPLEX Optimization Studio 22.1.0 with a time limit of 7,200 seconds.

Table II summarizes the results. The left side shows the multipliers and the right side the squares. The number of slices was reduced in the 13-bit multiplier and the 11×11-bit addition. Additionally, the number of stages was reduced in the 23×23-bit addition and the 26-bit multiplier.

V. Conclusion

This paper has proposed a method for enumerating GPCs implementable with a single slice of an FPGA. By this method,

[1]Implementation: https://github.com/ishiuralab/advgpcgen-rs

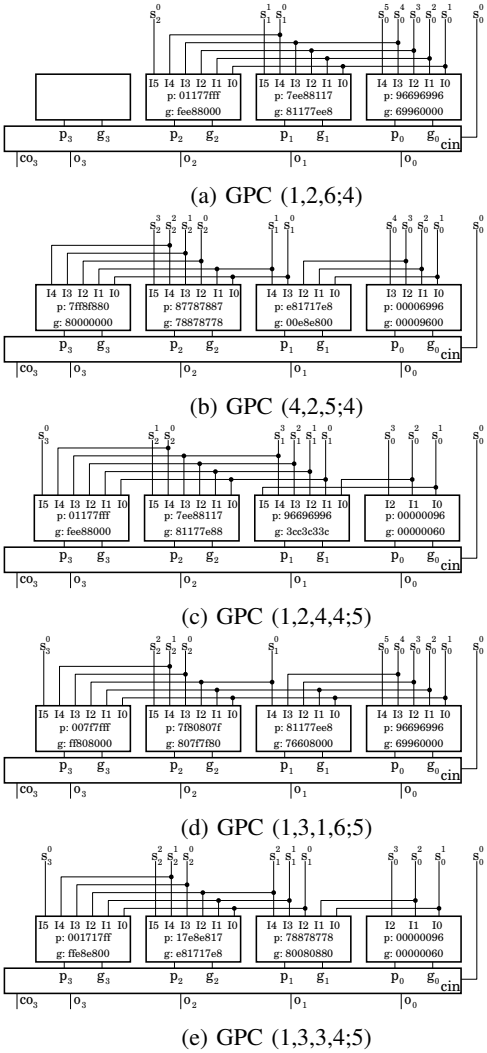


Fig. 6: Implementation of discovered GPCs

TABLE II: Synthesis result of multi-input adders ("✓" represents optimum)

	n-bit multiplier				addition of n integers of n bits			
	conventional		proposed		conventional		proposed	
n	stage	slice	stage	slice	stage	slice	stage	slice
11	✓2	✓14	✓2	✓14	✓3	✓15	✓3	**✓14**
13	✓3	✓20	✓3	**✓19**	✓3	✓21	✓3	✓21
23	✓3	72	✓3	71	4	71	**✓3**	75
26	4	91	**✓3**	94	✓4	94	✓4	93

we have discovered five GPCs that were previously unknown. Constructing multi-input adders with these GPCs confirmed their contribution to reducing circuit size and delay.

A future challenge is to extend the proposed method to FPGA models other than the Xilinx 7 series.

Acknowledgment

Authors would like to express their appreciation to Dr. Hiroyuki Kanbara of ASTEM/RI, Prof. Hiroyuki Tomiyama of Ritsumeikan University, and Mr. Takayuki Nakatani (formerly with Ritsumeikan University) for their discussion and valuable advises. We would also like to thank to the members of Ishiura Lab. of Kwansei Gakuin University.

REFERENCES

[1] S. C. Wallace: "A Suggestion for a Fast Multiplier," in IEEE Trans. on Electronic Computers, vol. EC-13, no. 1, pp. 14–17 (Feb. 1964).

[2] L. Dadda: "Some Schemes for Parallel Multipliers," in Alta Frequenza, vol. 34, pp. 349–356 (May 1965).

[3] T. Tanigawa, M. Noda, and N. Ishiura: "Efficient FPGA Implementation of Binarized Neural Networks Based on Generalized Parallel Counter Tree," in Proc. the Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI 2024), pp. 32–37 (Mar. 2024).

[4] N. Takagi, H. Yasuura, and S. Yajima: "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree," in IEEE Trans. on Computers, vol. C-34, no. 9, pp. 789–796 (Sept. 1985).

[5] T. Matsunaga, S. Kimura, and Y. Matsunaga: "Multi-Operand Adder Synthesis on FPGAs Using Generalized Parallel Counters," in Proc. Asia and South Pacific Design Automation Conference (ASP-DAC 2010), pp. 337-342 (Feb. 2010).

[6] B. Khurshid and R. N. Mir: "High Efficiency Generalized Parallel Counters for Xilinx FPGAs," in Proc. International Conference on High Performance Computing (HiPC 2015), pp. 40–46 (Dec. 2015).

[7] M. Kumm and P. Zipf: "Efficient High Speed Compression Trees on Xilinx FPGAs," in Proc. Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV 2014), pp. 1–12 (Jan. 2014).

[8] M. Kumm and P. Zipf: "Pipelined Compressor Tree Optimization using Integer Linear Programming," in Proc. International Conference on Field Programmable Logic and Applications (FPL 2014), pp. 1–8 (Sept. 2014).

[9] M. Kumm and J. Kappauf: "Advanced Compressor Tree Synthesis for FPGAs," in IEEE Trans. on Computers, vol. 67, no. 8, pp. 1078–1091 (Jan. 2018).

[10] Y. Yuan, L. Tu, K. Huang, X. Zhang, T. Zhang, D. Chen, and Z. Wang: "Area Optimized Synthesis of Compressor Trees on Xilinx FPGAs Using Generalized Parallel Counters," IEEE Access, vol. 7, pp. 134815–134827 (Sept. 2019).

[11] Xilinx, Inc.: 7 Series FPGAs Configurable Logic Block User Guide (UG474) (Sept. 2016), https://docs.amd.com/v/u/en-US/ug474_7Series_CLB (accessed in June 2024).

[12] T. B. Preußer: "Generic and Universal Parallel Matrix Summation with a Flexible Compression Goal for Xilinx FPGAs," in Proc. International Conference on Field Programmable Logic and Application (FPL 2017), pp. 1–7 (Sept. 2017).