Efficitent FPGA Implementation of Multiple-Input Adders Using Generalized Parallel Counter (6,0,7;5)

Mugi NODA † Ryo Kanai † Nagisa ISHIURA ††

† Graduate School of Science and Technology †† School of Engineering
Kwansei Gakuin University
1 Gakuen Uegahara, Sanda, Hyogo, 669-1330, JAPAN

Abstract—This paper proposes an efficient method for implementing multi-input adders on FPGAs, which are essential components in multipliers and neural networks, by hierarchically connecting 6-input 2-output adders. One major approach for FPGA implementation of multi-input adders involves constructing a tree of carry-save adders using Generalized Parallel Counters (GPCs) optimized through integer linear programming. However, many GPCs with a lowest-level input of 7 often consume two units of FPGA slices (basic FPGA components) resulting in reduced efficiency. In this research, we utilize the GPC (6,0,7;5), which achieves the highest bit reduction rate and can be implemented in a single slice only when the carry outputs are chain-connected. By cascading this GPC, we construct a 6-input, 2-output adder. These adders are then arranged into a carry-save tree structure to perform multi-input addition. Based on this method, we designed circuits to add m binary numbers of n bits for $n = 16, 32, 64$ and $m = 16, 32, ..., 512$, targeting the Xilinx 7 Series FPGA. The results demonstrates that, on average, the proposed method reduced the circuit area by 8.9%, the critical path delay by 6.7%. The time required for circuit construction to less than 0.001% compared to conventional methods.

I. Introduction

Multi-input addition, which computes the sum of multiple binary numbers, is a core operation for constructing various arithmetic circuits such as multipliers and multiply-accumulate (MAC) units. In recent years, its significance has grown particularly in the context of hardware acceleration for neural networks.

The implementation of multi-input addition as a combinational circuit has long been studied. A representative method is to construct a tree of carry-save adders using 3-input 2-output full adders as the basic building blocks [1][2]. However, when considering implementation on LUT-based FPGAs, circuits based on full adders do not necessarily map well onto 5- or 6-input LUTs.

For this reason, methods have been proposed that use extended full-adders with 6-inputs and 3-outputs [3], or adders called *generalized Parallel Counters* (abbreviated as *GPC* hereafter), which allow input weights not only of 1 but also of powers of 2, as the basic components. FPGAs are equipped with carry-lookahead circuits (carry logic) to efficiently handle addition and subtraction, and by utilizing this feature, GPCs can be implemented efficiently. Especially for Xilinx 7 Series, research has been conducted to implement a GPC in a single "slice", which consists of multiple LUTs and the carry logic [4][5][6][7].

A tree of carry-save adders based on GPCs is referred to as a *compressor tree*. Because a compressor tree is a complex structure composed of multiple GPCs with varying input/output specifications, heuristic algorithms and formulations based on integer linear programming have been proposed to obtain circuit configurations with minimal depth and area [4][5][8].

However, the previously proposed methods are considered to have two issues. First, the most efficient GPCs are not necessarily utilized. Among the GPCs that can be implemented within a single slice of the Xilinx 7 Series, the one with the greatest circuit size reduction effect is the 13-input, 5-output GPC (6,0,7;5). However, this GPC can only be implemented in a single slice when its carry outputs are connected in a chain, and thus it has not been employed in conventional compressor tree construction methods. Second, obtaining the optimal circuit configuration requires solving an optimization problem, which incurs a high computational cost.

In this paper, we propose a method for constructing multi-input adders that addresses these two issues. The proposed method ensures the implementation of the GPC (6,0,7;5) within a single slice while determining the circuit configuration with low computational overhead. First, by chaining GPC (6,0,7;5) units, a "6-2 adder" is constructed, which outputs the sum of six binary numbers as two binary numbers. A multi-input adder is then built by hierarchically connecting these 6-2 adders in a tree structure. Owing to the intensive use of highly efficient GPCs in terms of circuit area reduction, the resulting circuit requires fewer slices than conventional methods. Moreover, the circuit structure is regular and does not require solving complex optimization problems, resulting in negligible

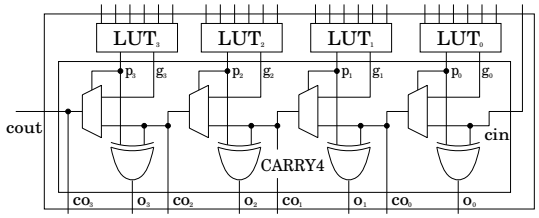

Fig. 1. FPGA slice model assumed in this paper [9]

computation time for generating the circuit configuration.

We implemented a multi-input adder based on the proposed method and compared it with conventional methods. As a result, the circuit area was reduced by 8.9%, the critical path delay was reduced by 6.7%, and the time required to generate the circuit configuration was reduced to less than 0.001%.

The rest of this paper is structured as follows. Section 2 introduces the target FPGA model, the GPC-based multi-input adder architecture, and the properties of the GPC (6,0,7;5). Section 3 describes the implementation of a 6-2 adder using the GPC (6,0,7;5) and a method for constructing multi-input adders. Section 4 evaluates the proposed circuits in terms of area and critical path delay, comparing them with conventional methods. Section 5 concludes the paper and outlines directions for future work.

II. FPGA Implementation of Multi-Input Adders

A. LUT-based FPGA

In this paper, following prior works [4][5][8], we assume an FPGA model composed of logic blocks (slices) made up of LUTs and carry logic, as illustrated in Fig. 1. A typical example of such an FPGA is the Xilinx 7 Series FPGA [9]. A "slice" consists of four 6-input 2-output LUTs and one 4-bit carry chain. LUT_0 to LUT_3 each generate a carry generate signal g_i and a carry propagate signal p_i, which are input into the CARRY4 unit.

Since the carry logic is implemented as embedded circuitry, its delay is smaller than that of circuits constructed using LUTs. A long carry chain can be formed by connecting the `cout` port of one slice to the `cin` port of another. Even when the chain is long, if the slices are placed adjacently, the delay can be kept low.

B. Multi-input adders and generalized parallel counters

Efficient methods for implementing multi-input adders include constructing carry-save adder trees using 3-input, 2-output full adders as basic building blocks, such as the Wallace tree [1] and the Dadda tree [2]. However, when targeting FPGA implementation, 3-input full adders are not well suited to 5- or 6-input LUT architectures. Furthermore, effectively utilizing fast carry chains presents an additional challenge.

TABLE I
GPCs implementable with a single slice

GPC					Ref.
(6,0,6;5)	(1,3,2,5;5)				[4]
(1,3,5;4)	(6,0,7;5)	(2,1,1,7;5)			[5]
(7;3)	(1,5;3)	(1,4,0,6;5)	(1,4,1,5;5)		[6]
(1,2,6;4)	(4,2,5;5)	(1,2,4,4;5)	(1,3,1,6;5)	(1,3,3,4;5)	[7]

To address this, FPGA-oriented implementation methods have been proposed that use extended adders, such as 6-input, 3-output adders, or *generalized parallel counters (GPCs)*, which allow inputs with binary-weighted values, as basic building blocks. The input/output configuration of a GPC is typically denoted in the form GPC (2,1,5;4), which indicates that the GPC takes two inputs with a weight of 4, one input with a weight of 2, and five inputs with a weight of 1, and produces a 4-bit binary output representing their weighted sum.

To make effective use of the carry logic and direct wiring within a slice, many studies have been conducted on implementing GPCs within a single slice based on the model shown in Fig. 1. Table. I lists GPCs that are known to be implementable in a single slice.

C. FPGA implementation of compressor trees

A carry-save addition tree using GPCs is referred to as a *compressor tree*.

In FPGA implementations of multi-input adders, a common approach is to reduce multiple binary numbers all the way down to two using a compressor tree, and then compute the final sum using a row adder that leverages the carry chain [4][5][8]. Figure 2 illustrates the circuit structure for summing m binary numbers of n bits each. The multiple GPCs in the center of the figure form the compressor tree which reduces the input to two rows. The row adder then adds these two rows to produce the final output.

Since the row adder is a carry-propagate adder (a ripple carry adder) as shown in Fig. 3, the delay is proportional to the bit width. Therefore, under the assumption of using such a row adder, the critical path delay of the multi-input adder implemented with a compressor tree becomes the sum of the compressor tree depth and the carry chain length of the row adder, resulting in an order of $O(\log m + n)$.

Using a carry look-ahead adder for the final addition can reduce the delay order to $O(\log m + \log n)$. However, since the delay of the carry chain is smaller than that of LUTs, a ripple-carry adder achieves lower actual delay for up to around $n = 256$ bits.

Compressor trees are composed of multiple types of GPCs with different input/output configurations, making their structure complex. Therefore, heuristic algorithms and formulations as integer linear programming (ILP) problems have been proposed to construct compressor trees with minimal depth and minimal circuit area [4][5][8]. However, methods based on ILP require long

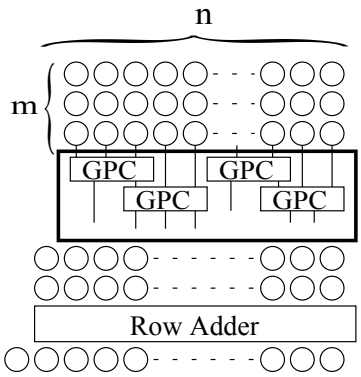

Fig. 2. Compressor tree

computation times, and when the values of n or m become large, it becomes infeasible to obtain the optimal solution.

D. Generalized parallel counter (6,0,7;5)

In a compressor tree, GPCs serve to reduce the number of bits in the inputs and intermediate results. For example, GPC (1,3,2,5;5) takes 11 input bits and produces 5 output bits, achieving a reduction ratio of 5/11. Similarly, GPC (6,0,7;5) takes 13 input bits and produces 5 output bits, achieving a reduction ratio of 5/13. Therefore, using GPCs with high bit-reduction efficiency that can be implemented in a single slice leads to smaller circuit size, as well as fewer stages.

Among the known GPCs implementable in a single slice, the one with the highest bit-reduction efficiency is GPC (6,0,7;5), with 13 inputs and 5 outputs. Its implementation is shown in Fig. 4. GPC (6,0,7;5) inputs one of the seven least significant bits to the `cin` port, while the remaining six least significant bits and the six most significant bits are each input to two LUTs as shown in the figure.

However, this GPC can be implemented in a single slice only if the carry output from another GPC's `cout` port is connected to its `cin` port. Otherwise, it consumes two slices.

In compressor trees, the carry output from a lower GPC is not always connected to the carry input of a higher GPC. In such cases, GPC (6,0,7;5) will require two slices for implementation, meaning its bit-reduction efficiency cannot be fully exploited.

III. Multi-Input Adder Based on Tree Connection of 6-Input 2-Output Adders

In this paper, we propose an efficient FPGA implementation method for multi-input adders that fully exploits the GPC (6,0,7;5). By chaining instances of GPC (6,0,7;5), which provides the highest bit reduction efficiency, we construct a 6-input, 2-output adder, while ensuring that each GPC can be implemented within a single slice. These adders are then connected in a carry-save structure to build a multi-input adder capable of summing multiple binary numbers.

Compared to conventional methods, the proposed approach improves bit reduction efficiency per slice, which is expected to reduce the overall circuit area of the multi-input adder. In addition, since the circuit structure is regular, there is no need to solve complex optimization problems, allowing the circuit configuration to be determined in a short computation time.

A. Construction of a 6-2 adder using the generalized parallel counter (6,0,7;5)

As shown in Fig. 5, by connecting the `cout` output of a GPC (6,0,7;5) to the `cin` input of the next GPC (6,0,7;5), a circuit can be constructed that receives six input bits every other digit and outputs their sum as a single binary number. Here, since the carry from the lower GPC is always input to the least significant digit of the next GPC, each GPC (6,0,7;5) can be implemented within a single slice.

A 6-input 2-output adder is constructed by placing two adders of the type shown in Fig. 5 to handle odd and even digit positions, respectively. The circuit structure is illustrated in Fig. 6. The black and red portions each represent the adder shown in Fig. 5. By shifting them by one bit and aligning them, a circuit that adds six binary numbers and outputs two binary numbers can be constructed.

The example in Fig. 6 assumes that the number of input digits is even. When the number of digits is odd, GPC (7;3) is used instead of GPC (6,0,7;5) for the most significant digit. While this does not change the number of slices used, it optimizes the number of LUTs required.

The 6-input 2-output adder constructed in this manner will be referred to as the **6-2 adder** hereafter.

B. Multi-input adder using 6-2 adders

A multi-input adder is constructed by connecting the 6-2 adders described in the previous section in a tree structure.

The overall structure of a multi-input adder that sums m binary numbers of n bits each into a single binary number is shown in Fig. 7. The thick-framed section in the center of the figure shows the tree-structured connection of 6-2 adders, which reduces the m inputs down to two rows. The "Row Adder" adds these two binary numbers using a method similar to the final stage of the compressor tree version and outputs the result.

The structure of the tree connection of 6-2 adders is illustrated in Fig. 8. In the first stage, each 6-2 adder outputs two values, and in the second stage, three of those outputs are grouped as inputs to another 6-2 adder. By repeating this process, m binary numbers of n bits can be reduced to two binary numbers.

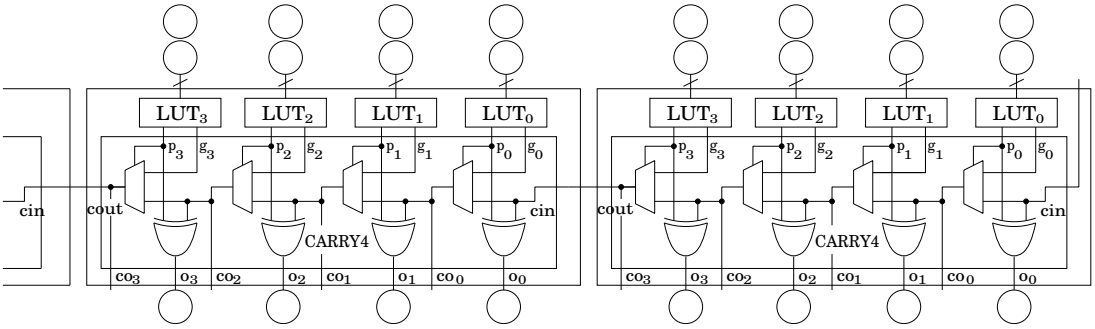

Fig. 3. Implementation of row adder

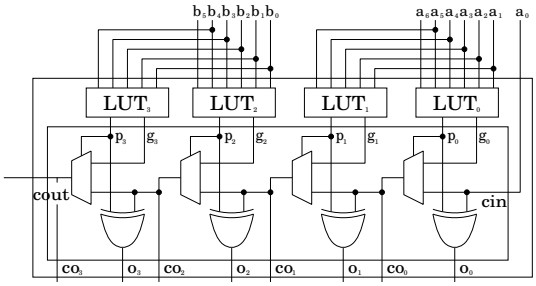

Fig. 4. FPGA implementation of GPC (6,0,7;5) [5]

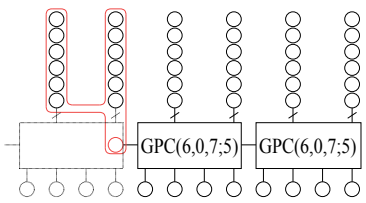

Fig. 5. Chained connection of GPC (6,0,7;5)

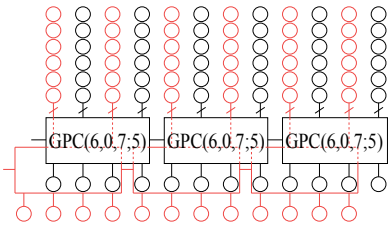

Fig. 6. Structure of the 6-2 adder

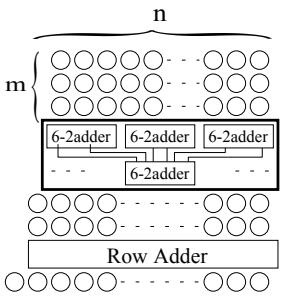

Fig. 7. Multi-input adder using 6-2 adders

When the number of inputs is exactly 2×3^h, a 6-2 adder tree of h stages can be constructed. If the number of inputs does not match this value, a tree of 2-1 adders (i.e., row adders) is used to reduce the number of binary inputs to 2×3^h, which are then provided as inputs to the h-stage 6-2 adder tree.

The order of the critical path delay (i.e., the number of GPC stages) in each level is $O(n)$. However, since the i-th bit output of one stage is input to the i-th bit of the next stage (not the least significant bit), the overall delay order for the 6-2 adder tree is $O(\log m + n)$. As a result, the delay order of the entire multi-input adder, including the row adder, is equivalent to that of a compressor tree.

The computational cost of constructing the tree using the algorithm in this section is proportional to the number of inputs m, and each 6-2 adder depends on the bit width n. Thus, the total complexity of building a multi-input adder for m n-bit numbers is $O(mn)$, which is significantly lower than that of solving an integer linear programming problem for compressor tree construction.

IV. Experimental Result

Based on the proposed method, we have implemented multi-input adders in Verilog HDL which outputs the sum of m binary numbers of n bits as a binary number. The resulting Verilog HDL descriptions were synthesized and placed/routed using Xilinx Vivado 2023.2 targeting Artix-7 (xc7a100tcsg324-3). Logic synthesis was performed using the default settings ("Vivado synthesis defaults" and "Vivado implementation defaults"), and the default value was used for the target clock frequency.

For comparison, we designed circuits using the conventional compressor tree construction method described in [8]. We also generated circuits through logic synthesis from a simple Verilog HDL description using addition operators. The structure of the compressor tree was obtained by solving an ILP problem using IBM ILOG CPLEX Studio 22.1 with the GPC set shown in Table II. The solver was executed on an AMD Ryzen 9 3900X with a time limit of 3600 seconds. In the logic synthesis-based method, logic circuits are generated from Verilog HDL

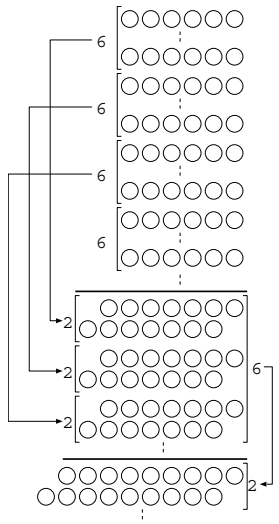

Fig. 8. Tree connection of 6-2 adders

description such as `dst = src1 + src2 +...+ src`m , without incorporating any external modules.

The results of the logic synthesis are summarized in Table III. Here, "#Slice" indicates the number of slices used, "Delay" represents the critical path delay, and "Time" denotes the computation time required to determine the circuit configuration.

In most cases, the proposed method outperformed the conventional compressor tree-based method in both the number of slices and delay. On average, the number of slices was reduced by approximately 8.9%, and the delay by about 6.7%. Moreover, the computation time for determining the circuit structure was reduced to less than 0.001% compared to the conventional method, as it does not require solving an integer linear programming problem. Compared to the method using the '+' operator, the number of slices was reduced by 10.4% on average, and the delay was reduced by approximately 16.8%.

The reduction in circuit size achieved by the proposed method is primarily attributed to the implementation of the GPC (6,0,7;5)—which offers the highest bit reduction rate among existing GPCs—within a single slice, as well as to the fact that the majority of the multi-input adder is constructed using this GPC. As for delay, although the number of GPC stages on the critical path is nearly the same in both the proposed and conventional methods, the key factor contributing to the improvement is that, in the proposed method, a larger portion of the critical path passes exclusively through the carry chain without traversing any LUTs.

Furthermore, when we replaced the final-stage adder in the multi-input adder from a carry-propagation adder to a carry look-ahead adder, the delay increased in all cases: on average, the delay increased by 27.1% with the proposed method and by 30.6% with the conventional method.

TABLE II
GPC SET USED FOR CONVENTIONAL METHOD [8]

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

V. Conclusion

In this paper, we have proposed a construction method for multi-input adders based on 6-2 adders. By connecting GPC (6,0,7;5) in a chained manner, we ensured its implementation within a single slice and constructed a 6-2 adder. These adders were then connected in a tree structure to realize a multi-input adder.

As a result of constructing the multi-input adder based on the proposed method, the circuit size was reduced by an average of 8.9% and the critical path delay by an average of 6.7% compared to conventional methods. The time required for circuit construction was suppressed to less than 0.001% of that of the conventional method.

The proposed method is considered to have practical value in that it enables efficient FPGA implementation of multi-input adders without solving optimization problems.

The proposed method can be applied to simple summation of multiple values, but it cannot be directly applied to cases such as summation of partially shifted binary numbers in multiplication circuits. Furthermore, since the method assumes that all numbers are added using combinational circuits (in a single clock cycle), it is not compatible with sequential inputs or pipelined processing. Extending the proposed method to accommodate various application forms remains a subject for future work.

Acknowledgements

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 suggenstion for a fast multiplier," in *IEEE Trans. Electronic Computer*, vol. EC-13, issue 1 (Feb. 1964).

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

[3] 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).

TABLE III
SYNTHESIS RESULTS FOR CIRCUITS ADDING m BINARY NUMBERS OF n BITS

		Proposed			Compressor Tree [8]			+ Operators	
n	m	#Slice	Delay (ns)	Time (s)	#Slice	Delay (ns)	Time (s)	#Slice	Delay (ns)
16	16	39	9.8	0.02	39	10.2	3626	46	11.8
	32	75	11.5	0.01	76	11.8	3628	88	12.4
	54	118	11.8	0.01	130	11.7	3628	145	15.5
	64	146	13.3	0.02	150	13.5	3622	169	15.4
	128	285	14.2	0.02	297	15.8	3623	335	17.3
	162	354	14.9	0.02	377	15.5	3622	422	17.2
	256	564	15.8	0.01	599	16.5	3619	661	18.1
	486	1063	17.0	0.02	1127	19.5	3618	1252	21.3
	512	1128	17.8	0.02	1190	20.6	3615	1319	20.3
32	16	75	10.3	0.02	80	10.9	3621	84	13.1
	32	143	12.4	0.01	156	12.4	3612	158	14.1
	54	226	12.6	0.01	259	14.7	3615	259	16.0
	64	278	14.0	0.01	303	14.8	3610	301	16.7
	128	545	16.0	0.02	598	16.3	3615	597	19.2
	162	678	15.6	0.01	759	16.1	3619	751	18.9
	256	1080	17.1	0.01	1189	19.0	3618	1180	19.6
	486	2035	18.3	0.02	2265	21.7	3610	2228	23.5
	512	2156	19.2	0.02	2366	21.0	3609	2350	22.8
64	16	147	11.9	0.02	168	13.6	3603	163	16.0
	32	279	13.7	0.01	313	14.7	3605	298	15.3
	54	442	15.7	0.01	523	16.0	3608	483	19.5
	64	542	16.8	0.01	615	16.7	3604	578	21.3
	128	1065	16.8	0.01	1217	18.9	3606	1120	21.1
	162	1326	17.4	0.01	1524	20.2	3606	1410	22.6
	256	2112	19.0	0.01	2392	22.4	3605	2220	21.5
	486	3979	21.7	0.01	4556	22.9	3605	4187	25.9
	512	4212	24.3	0.02	4825	23.1	3604	4408	26.3

[4] 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).

[5] 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," in *IEEE Access*, vol. 7, pp. 134815–134827 (Sept. 2019).

[6] 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).

[7] M. Noda, and N. Ishiura: "Enumeration of Generalized Parallel Counters for Multi-Input Adder Synthesis for FPGAs," in *Proc. Asia and Pacific Conference on Circuit and Systems (APCCAS 2024)*, pp. 64–68 (Nov. 2024).

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

[9] Xilinx, Inc.: 7 Series FPGAs Configurable Logic Block User Guide (UG474) (Sept. 2016), `https://docs.amd.com/r/en-US/ug474_7Series_CLB` (accessed in Aug. 2025).