Implementing Parallel Counters Based on Sorting Networks

Yuchen He
Department of Electrical & Computer Engineering
University of California, Santa Barbara
Santa Barbara, CA 93106, USA
yuchenhe@ucsb.edu

Behrooz Parhami
Department of Electrical & Computer Engineering
University of California, Santa Barbara
Santa Barbara, CA 93106, USA
parhami@ece.ucsb.edu

***Abstract*—Parallel counters form a well-studied class of arithmetic circuits with numerous implementations, where conventional designs face irregular wiring and long carry chains, especially for a large number of inputs. We focus on sorting-network-based parallel counters that replace cascaded full-adders with sorting stages and minimal logic. Such sorting networks introduce a regular, highly pipelinable structure, eliminating the irregular wiring and long carry chains of conventional full-adder-based counters. We evaluate our design in terms of delay, hardware cost, and delay-cost product, comparing it against conventional architectures. Experimental results show competitive latency for moderate numbers of inputs (7-bit: 10.43 ns vs. 10.57 ns for FA based). For larger counters (15-bit), by properly tailoring the sorting network, the latency outperforms the conventional one (13.56 ns vs 14.85 ns) with significantly higher logic gates cost. The results cited are for FPGA realizations. We anticipate greater benefits with custom VLSI due to the VLSI-friendliness of our design.**

***Index Terms*—parallel counter, sorting network, custom VLSI, FPGA, pipelining, regularity**

I. Introduction

Parallel counters [1], circuits that count the number of 1s among n binary inputs and present the result as a $(\log_2(n+1))$-bit binary number, constitute useful building blocks for many applications.

The standard way of designing parallel counters is through full-adder trees, with designs requiring $O(n)$ full-adders and $O(\log n)$ delay. However, the resulting structures are not VLSI-friendly, owing to the presence of irregular connections and long wires. Thus, a variety of other designs have been proposed over the years that provide delay-cost-power trade-offs, depending on input size n, nature of application, circuit technology used, and implementation platform.

In this paper, we show that the use of sorting networks provides still more implementation alternatives that the designers can use beneficially in some cases.

II. Background

A. Recursive Construction of Conventional Counters

We represent an n-input parallel counter (actually, its output) as $?/n$. A $?/n$ parallel counting network (PCN) can be systematically synthesized through recursive decomposition. Specifically, as shown in Fig.1, the network can be realized using two $?/\lfloor n/2 \rfloor$ PCNs combined with a binary adder. For odd values of n, the adder's carry-in is driven by the additional input bit x_n; otherwise, the carry-in is set to zero. Iterative application of this construction yields a hierarchical implementation that eventually reduces to the canonical primitives of population counting, namely the $?/2$ half-adder (HA) and the $?/3$ full-adder (FA).

When constructed recursively, an n-input PCN results in $\lfloor \log_2 n \rfloor$ levels of recursion, which corresponds directly to the number of adder stages. The widths of these adders are inherently determined by the input labels and are therefore omitted from our simplified block diagrams. A straightforward derivation of the delay and cost associated with the synthesis scheme illustrated in Fig.1 is presented below.

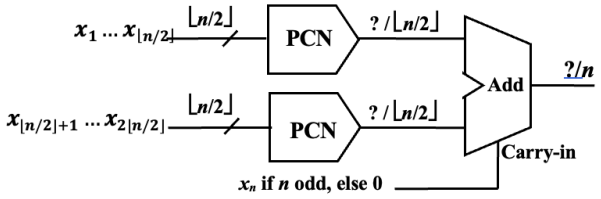

Fig. 1. Recursive synthesis of a $?/n$ parallel counting network by means of a 2-input adder with carry-in and two smaller $?/\lfloor n/2 \rfloor$ parallel counting networks

$$D(?/n) = D(?/\lfloor n/2 \rfloor) + D((\log_2 \lfloor n/2 \rfloor + 1)\text{-bit adder}) \quad (1)$$

$$C(?/n) = 2C(?/\lfloor n/2 \rfloor) + C((\log_2 \lfloor n/2 \rfloor + 1)\text{-bit adder}) \quad (2)$$

B. Saturating Parallel Counters

Saturating parallel counters [2] (SPCs) are a variant of conventional parallel counters designed to impose an upper bound on their output range. Unlike standard counters, which produce an exact population count of the input bits, SPCs cap the count at a pre-defined saturation value. As an example, consider a saturation threshold of 5. In this case, the SPC must generate valid outputs for counts 0 through 5. Any count exceeding 5 yields the saturation value 5.

C. Other Architectures and Optimizations

Newer variants extend or optimize the basic design. Accumulative parallel counters (APCs) [3] integrate an internal accumulator and adder into one stateful network, later refined [4] through hierarchical and pipelined VLSI implementations. Up/Down parallel counters [5], [6] support signed outputs and serve in Hamming-weight comparators. Technology-specific optimizations include: RSFQ/ERSFQ counters achieving 50 GHz operation [7], LUT-based FPGA mappings that remove carry chains [8], symmetric-stacking 6:3 counters with XOR-free critical paths [9], and hand-tuned (15, 4) designs optimized for delay and area [10].

III. Bit Sorting Networks

A sorting network is a circuit that accepts n inputs, and produces n sorted outputs. Later in this paper, we refer to such n-input, n-output sorting network as n-sorters.

A. Basic Block For Bit Sorting Networks

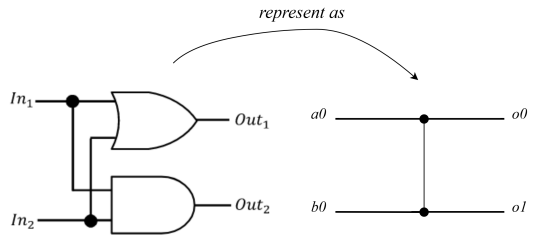

Fig. 2. 2-sorter (comparator).

There are multiple ways to implement sorting networks, all using 2-sorters (see Fig.2) as a building block. The operation of a 2-sorter is simple. For any input pair, the OR gate outputs the greater (or maximum) bit and the AND gate outputs the smaller (minimum) bit, thus yielding the sorted order.

B. Bit Sorter Output Format

A sorting network can be designed to sort numerical values presented in integer or floating-point format. Here, we are interested in sorting networks with single-bit binary inputs (bit-sorters). We assume a descending order for the sorted output. If the n inputs contain k 1s, the output of such a sorting network will be:

$$1_0 \quad 1_1 \quad . \quad . \quad . \quad 1_{k-1} \quad 0_0 \quad 0_1 \quad . \quad . \quad . \quad 0_{n-k-1} \quad (3)$$

C. Batcher's Even-Odd Merge Sorting Network

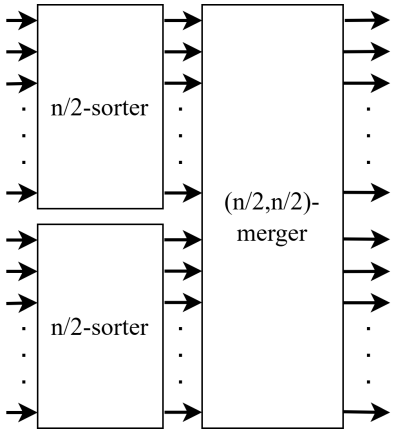

Fig. 3. The recursive structure of Batcher's even–odd merge sorting network.

Batcher proposed an ingenious technique called even-odd merge based on an $(n/2, n/2)$-merger. An $(n/2, n/2)$-merger [11], [12] is a circuit that takes two sorted input sequences of length $n/2$ and merges them into one sorted sequence of length n. Fig. 3 depicts the recursive structure of Batcher's even–odd merge sorting network. In this diagram, an $(n/2, n/2)$-merger is constructed from smaller merger components and 2-sorters. So, by unrolling the recursion of Fig.3, a sorting network consisting of 2-sorters is created.

D. Bitonic Merging Networks

A bitonic sorter exploits the fact that any bitonic sequence (first monotonically increasing, then monotonically decreasing) of length n can be split and recursively sorted. Given a bitonic input, we first apply n 2-sorters between the first half and the second half, as shown in Fig. 4. These 2-sorters partition the sequence into two subsequences—each still bitonic—where all elements in the first subsequence are no greater than those in the second. Each subsequence can then be sorted by recursively building the $n/2$-bitonic sorters.

Finally, concatenating these two sorted subsequences yields the fully sorted output.

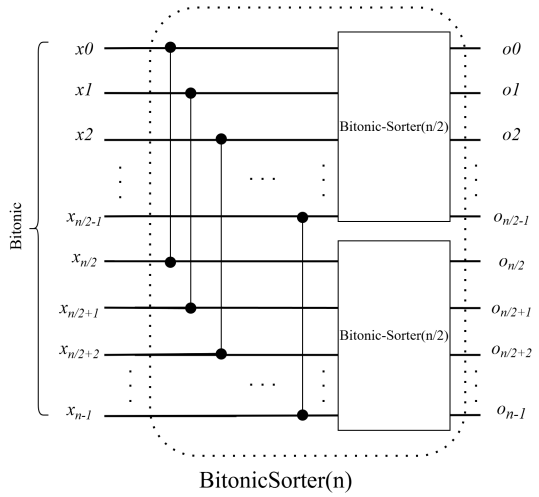

Fig. 4. The recursive structure of Batcher's bitonic sorting network.

E. Complexity of Sorting networks

Table I summarizes both the *exact* and *asymptotic* cost and delay of three canonical n-input sorting networks, as well as their combined cost–delay product $C(n) \times D(n)$. Here:

- **Cost** $C(n)$ is the total number of 2-sorters (comparators) used.
- **Delay** $D(n)$ is the number of 2-sorters on the critical path.
- The product $C \times D$ gives a single metric that balances logic unit consumption (cost) against delay.

Both even–odd Merge and Bitonic Merge networks achieve good scaling by exploiting recursive half-mergers: $C(n) = O(n \log^2 n)$, $D(n) = O(\log^2 n)$, and hence $C \times D = \Theta(n \log^4 n)$. Although their asymptotics match, notice the constant-factor difference in the exact formulas: $C_{\mathrm{EO}}(n) = \frac{n(\log^2 n - \log n + 4)}{4} - 1$, $C_{\mathrm{B}}(n) = \frac{n(\log^2 n + \log n)}{4}$. So in practice the even–odd merge sorter is often preferable for its lower hardware cost at similar depth.

The $C \times D$ metric provides a combined figure of merit for performance, balancing the logic complexity and latency. By highlighting the $C \times D$ metric, Table I neatly captures overall performance in a single measure that combines both cost and delay. In the rest of this paper we focus on the even–odd merge network or other more efficient sorting network variants.

IV. Proposed Design

A. Design Concept

In this paper, we provide an interesting perspective of implementing parallel counters via sorting networks instead of using cascaded FAs. Our key idea is to sort this sequence using a sorting network (so that all 1s move to one end), and then deduce the number of 1s from the sorted output. In effect, the sorting network performs the accumulation of ones without explicit addition. Thus, we eliminate the carry-ripple delay inherent in an FA-based parallel counter and instead incur only the delay of a sorting network plus extraction logic and output generation logic. But the delay will heavily depend on the length of critical path of the sorting network. Guo et al. [14] provide an approach to divide an n-sorter into $\lceil \frac{n}{2} \rceil$- and $\lfloor \frac{n}{2} \rfloor$-sorters. This method can reduce the delay of sorting network slightly for n inputs but increases the complexity of output generation logic.

B. Design Details

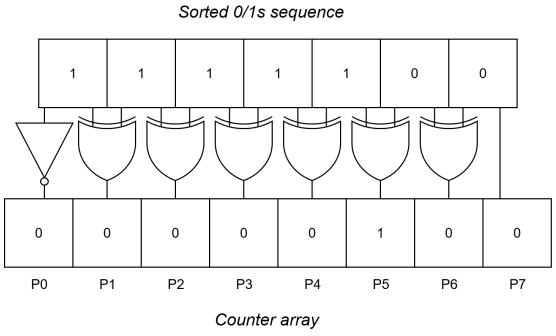

Fig. 5. Counting information extraction circuit for (7;3) counter.

The proposed n-input parallel counter comprises an n-sorter, $(n-1)$ XOR gates, a single NOT gate, and $\lceil \log_2(n+1) \rceil$ OR gates. As Fig.5 illustrates for the 7-input case, six XOR gates form a one-hot encoded *Counter Array*, producing signals P_0 through P_7 that indicate if the count of ones is exactly 0, 1, ..., 7, respectively (e.g., $P_5 = 1$ means exactly five of the seven inputs are 1). Thus, XOR gates and NOT gate constitute the extraction logic. Finally, by driving three OR gates (output generation logic) ($\lceil \log_2(7+1) \rceil = 3$) with the extraction logic output, we obtain the three-bit binary count (C_2, C_1, C_0). For instance, C_2 (the MSB) is 1 if the count of ones is 4 or more (i.e., $4, 5, 6, 7$); C_1 is 1 if the count is in $2, 3, 6, 7$; and C_0 is 1 if the count is odd (1,3,5,7 ones).

$$C_2 = P_4 | P_5 | P_6 | P_7 \tag{4}$$

TABLE I
COST & DELAY OF DIFFERENT SORTING NETWORKS OF SIZE n

	Bubble/Insertion	***Even-odd merge***	***Bitonic merge***
Exact	$C(n) = \frac{n(n-1)}{2}$	$C(n) = \frac{n(\log^2 n - \log n + 4)}{4} - 1$	$C(n) = \frac{n(\log^2 n + \log n)}{4}$
	$D(n) = 2n - 3$	$D(n) = \frac{\log n(\log n + 1)}{2}$	$D(n) = \frac{\log n(\log n + 1)}{2}$
Asymptotic	$C(n) = O(n^2)$	$C(n) = O(n \log^2(n))$	$C(n) = O(n \log^2(n))$
	$D(n) = O(n)$	$D(n) = O(\log^2(n))$	$D(n) = O(\log^2(n))$
	$C \times D = \Theta(n^3)$	$C \times D = \Theta(n \log^4(n))$	$C \times D = \Theta(n \log^4(n))$

$C(n)$: number of 2-sorters, $D(n)$: number of 2-sorters on the critical path, $C \times D$: Cost×Delay [12], [13]

$$C_1 = P_2|P_3|P_6|P_7 \tag{5}$$

$$C_0 = P_1|P_3|P_5|P_7 \tag{6}$$

C. Complexity Analysis

As shown in Table I, the even–odd merge sorting network offers the best asymptotic trade-off in cost, delay, and $C \times D$, so later in this paper we base our counters on that network. Let $D_{\mathrm{SN}}(n)$ and $C_{\mathrm{SN}}(n)$ (C(n) in Table I times 2, since a 2-sorter has 2 gates) be the delay and gate-count of the sorting network , and let $D_{\mathrm{OutputGen}} = 3$, $C_{\mathrm{OutputGen}} = (n-1)XORs + 1(NOT) + \lceil \log_2(n+1) \rceil ORs$ be the constant overhead for final output logic. Note here, $C(n)$ is gate cost and $D(n)$ is gate delay. Then

$$D_{\mathrm{total}}(n) = D_{\mathrm{SN}}(n) + D_{\mathrm{OutputGen}} \tag{7}$$

$$C_{\mathrm{total}}(n) = C_{\mathrm{SN}}(n) + C_{\mathrm{OutputGen}} \tag{8}$$

Because $D_{\mathrm{OutputGen}}$ and $C_{\mathrm{OutputGen}}$ do not grow or grow linearly with n, the overall asymptotic complexity remains $O(\log^2 n)$ depth and $O(n \log^2 n)$ gate-count. The exact formulas also are given below.

$$D(n) = \frac{\log n \, (\log n + 1)}{2} + 3 \tag{9}$$

$$C(n) = \frac{n(\log^2 n - \log n + 4)}{2} + \lceil \log_2(n+1) \rceil + 4n - 5 \tag{10}$$

Our sorting network-based design outperforms the FA tree in delay for moderate n (e.g. 8, 16, 32) in theory, at the expense of higher overall gate count.

D. Benefits of Sorting Networks For Parallel Counters

Based on our previous discussion, a sorting network is built recursively from identical 2-sorter blocks, yielding a highly regular layout. This regular structure is more amenable to VLSI realization than FA-based parallel counters, which suffer from irregular routing. Unlike a ripple-carry adder, there is no long serialized carry chain. The critical path is simply a series of 2-sorters, each with equal delay. Moreover, you can insert registers between comparator stages to pipeline the network, boosting throughput and raising the achievable clock frequency. Because every pipeline stage has nearly the same latency, timing closure is much easier—you avoid the "fast front end, slow back end" imbalance. Taken together, these implementation advantages point to the benefit of sorting network based designs for parallel counters.

V. PERFORMANCE EVALUATION

The following metrics are used for evaluation.

- **Slice LUTs and Slices** denote post-place-and-route FPGA resource utilization, reflecting the overall logic usage.
- **F7/F8 Muxes** represent higher-level multiplexers automatically inferred by the Xilinx toolchain.
- **Max Combinational Delay** (in nanoseconds) corresponds to the critical-path delay reported after implementation.
- **Cost–Delay Product** is computed as the product of the LUT count and the maximum combinational delay, serving as a combined area–speed metric that captures both hardware cost and timing efficiency.

We implemented both the sorting-network-based (SN) and full-adder–based (FA) parallel counters on a Xilinx Artix-7 FPGA (Nexys A7 board) to compare their performance. Table II summarizes the post-place-and-route resource utilization (LUT count, slice count, Mux count) and timing results for $(7;3)$ counter and $(15;4)$ counter. For reference, we also include results from parallel counter based on the single stage sorter design by Kent et al. [15], which represents a highly optimized alternative approach. For the $(7;3)$ counter, our sorting network design uses only 1 more LUT than the FA design (6 vs. 5 LUTs) and achieves a comparable maximum combinational delay (10.43 ns vs. 10.57 ns for FA). This indicates that even for a small counter, the overhead of the sorting network is minimal and it

TABLE II
COMPARISON OF FA- AND SN-BASED PARALLEL COUNTER DESIGNS FOR $(7;3)$ AND $(15;4)$ COUNTERS

Metric	SN(7;3)	FA(7;3)	[15] (7;3)	[15] (15;4)	SN(15;4)	FA(15;4)
Slice LUTs	6	5	6	57	45	13
Slices	2	2	2	16	13	4
F7 Muxes	2	0	2	0	0	0
F8 Muxes	0	0	0	0	0	0
Max Combinational Delay (ns)	10.43	10.57	10.38	13.56	17.67	14.85
Cost-delay product(LUT×delay)	62.59	52.86	62.29	772.69	795.01	193.09

[15] proposed a single stage sorter.

can slightly outperform the ripple-carry adder tree in speed. The Kent et al. [15] design also uses 6 LUTs and obtains a similar delay (10.38 ns). For the larger $(15;4)$ counter, the trends differ. The FA-based design is very LUT-efficient (only 13 LUTs) and has a 14.85 ns delay, whereas our sorting network-based design uses about 3.5× LUTs (45) and incurs a higher delay of 17.67 ns. The single stage sorter from [15] uses even more LUTs (57) but achieves the best delay at 13.56 ns.

It is important to note that FPGA tools such as Vivado map full-adder trees into dedicated LUT-plus-carry-chain primitives, resulting in real-world delays that are lower than our simple gate-delay estimates. With a tailored fast sorting network and straightforward pipelining to break the critical path, our sorting network based design remains attractive for parallel counting despite having a higher cost-delay product (Table II) than the conventional FA tree.

A. Evaluation For Saturating Parallel Counter

We also implemented SN-based and FA-based saturating parallel counters on a Xilinx Artix-7 FPGA. Table III summarizes the post-place-and-route resource utilization (LUT count, slice count, Mux count) and timing results for 7-input saturating counter and 15-input saturating counter of saturation value 5. For 7-input counter, the delay of SN based counter slightly decreased since implementing saturating parallel counter would remove some 2-sorters in last level to make a selection network. The cost reduction can vary depending on the saturation value. For 15-input SN-based counter of saturation value 5, we can see a significant cost decrease compared to a $(15;4)$ SN-based counter. The differences between SN-based and FA-based saturating design are similar with previous parallel counters.

VI. DISCUSSION

The sorting network is the dominant contributor to both cost and delay in the proposed parallel counter. Consequently, future optimizations should focus on designing faster and more resource-efficient sorting networks. One promising strategy is to adopt well-established, high-performance sorting network topologies, as described by Knuth [16](page 231). Although the fastest known networks are typically limited to small n, they remain practical for the $(7;3)$ and $(15;4)$ counters investigated here. Kent's one-stage sorter [15] is also appealing. However, it incurs additional hardware cost and alters the 2-sorter-based structure of the sorting network, which compromises pipelining feasibility.

A. Pipelining

Mueller [12] has shown that sorting networks can be easily pipelined by inserting registers between comparator stages. In contrast, pipelining FA-based counters is more challenging due to their long carry chains. Through pipelining, the effective critical-path delay of the sorting network can be reduced from $D_{\mathrm{SN}}(n)$ to $D_{\mathrm{SN}}(n)/h$ + pipeline stage overhead, where h is the trade-off parameter, thereby greatly increasing throughput. For example, if a sorting network has 12 levels of 2-sorters, we can use h=6 (each pipeline stage includes two 2-sorter delays),h=4,h=3 or h=2. This makes the SN–based counter particularly attractive for applications such as digital neural networks.

B. Generalized Parallel Counters

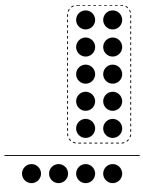

Fig. 6. Dot representation of generalized (5,5;4) counter.

The sorting-network approach can also be extended to generalized parallel counters that process weighted

TABLE III
COMPARISON OF FA- AND SN-BASED SATURATING PARALLEL COUNTER DESIGNS FOR 7-INPUT AND 15-INPUT COUNTERS OF SATURATION VALUE 5

Metric	SN(7)	FA(7)	SN(15)	FA(15)
Slice LUTs	6	4	24	12
Slices	2	2	7	4
F7 Muxes	3	0	0	0
F8 Muxes	0	0	0	0
Max Combinational Delay (ns)	10.12	10.80	16.05	14.75
Cost-delay product(LUT×delay)	60.72	43.20	385.20	177.00

inputs. if there are n_j dots in each of h input columns, $0 \leq j \leq h-1$, the associated generalized parallel counter, denoted as $(n_{h-1}, ..., n_1, n_0; k)$-counter, is feasible only if $\sum(n_j 2^j) \leq 2^k - 1$. $(5, 5; 4)$ counter used here as example (dot representation shown in Fig. 6) that handles five weight-2 and five weight-1 inputs and 4-bit output. In such designs, each group of inputs with the same weight may require its own sorting network and counter array, followed by more complex output generation logic. For instance, weight-1 inputs influence outputs C_0–C_3, while weight-2 inputs affect outputs C_1–C_3. Let us consider the output boolean function of (5,5;4) counter for generalized parallel counter. Counter array indexes are represented as $P_i^1, P_i^2, \quad i \in [0, 5]$, respectively.

$$C_0 = P_1^1 | P_3^1 | P_5^1 \tag{11}$$

Equation 11 is the original boolean function for C_0 of the SN-based generalized parallel counter. The expressions for C_1,C_2 and C_3 can be similarly derived. Optimizations similar to those suggested by Guo et al [14] could be applicable here.

C. Merging two Saturating Parallel Counters

We construct k-input and m-input saturating parallel counters by employing Type I selection networks [13] (page 142). The saturation condition of each counter can be efficiently detected by examining the $(s+1)$-th output element: if either counter reaches the prescribed saturation value s, the final $(k+m)$-input counter directly produces this saturated output. Otherwise, when neither counter reaches saturation, the two partially sorted sequences of length s can be permuted into a bitonic order. This transformation enables the use of a bitonic sorter, which achieves improved efficiency due to the input's prearranged structure. Subsequently, the same synthesis technique described in Fig. 5 can be applied.

Hence, the resulting structure yields a $(k+m)$-input saturating parallel counter with saturation value s, formally expressed as

$$\text{SPC}_{k+m,s} = \text{Merge}(\text{SPC}_{k,s}, \text{SPC}_{m,s})\,. \tag{12}$$

This hierarchical construction provides a scalable method for synthesizing large saturating counters from smaller building blocks.

VII. CONCLUSION & FUTURE WORK

We have shown that synthesizing parallel counters based on sorting networks can be beneficial in terms of delay and/or cost in some cases and leads to more VLSI-friendly designs, providing benefits in delay/cost/power that may not be apparent in gate-level analyses.

Further work can proceed in several directions. First, if we use custom VLSI instead of FPGAs, the trade-offs and cross-over points change, which may render sorting-network based designs more competitive. Second, we can direct our attention to the level of applications of parallel counters, in effect merging the parallel counter design with whatever application logic that uses the counter's outputs. A good example is the design of median filters [17]. Third, there are various repositories of sorting network designs that can be used for implementing larger parallel counters. Optimal sorting networks are known only for n up to 12. For larger values of n, optimal designs are not yet known, but efficient designs exist [18]. In some cases, there are multiple efficient designs for the same value of n that can provide some flexibility to designers.

Finally, generating proposals for modifying the architectures of FPGAs in order to make them more friendly to the embedding of sorting networks can be entertained, in the same way that multipliers, DSP cores, and other specialized units have been added to modern FPGAs.

REFERENCES

[1] E. Swartzlander, "Parallel Counters," *IEEE Trans. Comput.*, vol. C-22, no. 11, pp. 1021–1024, Nov. 1973. [Online]. Available: http://ieeexplore.ieee.org/document/1672232/

[2] P. Balzola, M. Schulte, J. Ruan, J. Glossner, and E. Hokenek, "Design alternatives for parallel saturating multioperand adders," in *Proceedings 2001 IEEE International Conference on Computer Design: VLSI in Computers and Processors. ICCD 2001*, Sep. 2001, pp. 172–177, iSSN: 1063-6404. [Online]. Available: https://ieeexplore.ieee.org/document/955021/

[3] B. Parhami and C.-H. Yeh, "Accumulative parallel counters," in *Conference Record of The Twenty-Ninth Asilomar Conference on Signals, Systems and Computers*, vol. 2. Pacific Grove, CA, USA: IEEE Comput. Soc. Press, 1996, pp. 966–970. [Online]. Available: http://ieeexplore.ieee.org/document/540843/

[4] C.-H. Yeh and B. Parhami, "Efficient designs for multi-input counters," in *Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers (Cat. No.CH37020)*, vol. 2. Pacific Grove, CA, USA: IEEE, 1999, pp. 1340–1344. [Online]. Available: http://ieeexplore.ieee.org/document/831925/

[5] B. Parhami, "Parallel counters for signed binary signals," in *Twenty-Third Asilomar Conference on Signals, Systems and Computers, 1989.* Pacific Grove, California, USA: IEEE, 1989, pp. 513–516. [Online]. Available: http://ieeexplore.ieee.org/document/1200844/

[6] ——, "Efficient Hamming Weight Comparators for Binary Vectors Based on Accumulative and Up/Down Parallel Counters," *IEEE Trans. Circuits Syst. II*, vol. 56, no. 2, pp. 167–171, Feb. 2009. [Online]. Available: http://ieeexplore.ieee.org/document/4781532/

[7] M. Eren Celik, T. V. Filippov, A. Sahu, D. E. Kirichenko, S. M. Sarwana, A. E. Lehmann, and D. Gupta, "Fast RSFQ and ERSFQ Parallel Counters," *IEEE Trans. Appl. Supercond.*, vol. 30, no. 7, pp. 1–4, Oct. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9145805/

[8] B. Khurshid, "LUT Based Generalized Parallel Counters for State - of - art FPGAs," *ELS*, vol. 21, no. 1, p. 3, Jul. 2017. [Online]. Available: http://doisrpska.nub.rs/index.php/electronics/article/view/3551

[9] C. Fritz and A. T. Fam, "Fast Binary Counters Based on Symmetric Stacking," *IEEE Trans. VLSI Syst.*, vol. 25, no. 10, pp. 2971–2975, Oct. 2017, publisher: Institute of Electrical and Electronics Engineers (IEEE). [Online]. Available: http://ieeexplore.ieee.org/document/7981402/

[10] Q. Jiang and S. Li, "A design of manually optimized (15, 4) parallel counter," in *2017 International Conference on Electron Devices and Solid-State Circuits (EDSSC)*. Hsinchu: IEEE, Oct. 2017, pp. 1–2. [Online]. Available: http://ieeexplore.ieee.org/document/8126527/

[11] S. W. Al-Haj Baddar and K. E. Batcher, *Designing Sorting Networks: A New Paradigm*. New York, NY: Springer New York, 2011. [Online]. Available: https://link.springer.com/10.1007/978-1-4614-1851-1

[12] R. Mueller, J. Teubner, and G. Alonso, "Sorting networks on FPGAs," *The VLDB Journal*, vol. 21, no. 1, pp. 1–23, Feb. 2012. [Online]. Available: http://link.springer.com/10.1007/s00778-011-0232-z

[13] B. Parhami, *Introduction to Parallel Processing: Algorithms and Architectures*, ser. Series in Computer Science. Boston, MA: Kluwer Academic Publishers, 2002.

[14] W. Guo and S. Li, "Fast Binary Counters and Compressors Generated by Sorting Network," *IEEE Trans. VLSI Syst.*, vol. 29, no. 6, pp. 1220–1230, Jun. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9388166/

[15] R. B. Kent and M. S. Pattichis, "Design, Implementation, and Analysis of High-Speed Single-Stage N-Sorters and N-Filters," *IEEE Access*, vol. 9, pp. 2576–2591, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9308909/

[16] D. E. Knuth, *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. USA: Addison Wesley Longman Publishing Co., Inc., 1998.

[17] A. Adams, "Fast median filters using separable sorting networks," *ACM Trans. Graph.*, vol. 40, no. 4, pp. 1–11, Aug. 2021. [Online]. Available: https://dl.acm.org/doi/10.1145/3450626.3459773

[18] B. Dobbelaere, "List of Sorting Networks," Online. Available: https://bertdobbelaere.github.io/sorting_networks.html, accessed: Sep. 1, 2025.