DL-Sort: A Hybrid Approach to Scalable Hardware-Accelerated Fully-Streaming Sorting

Hyun Woo Oh, Joungmin Park, and Seung Eun Lee*
Department of Electronic Engineering
Seoul National University of Science and Technology, 01811, Seoul, Republic of Korea
Email: {ohhyunwoo, parkjoungmin, seung.lee}@seoultech.ac.kr

***Abstract*—Designing high-performance hardware sorter for resource-constrained systems is challenging due to physical limitations and the need to balance streaming bandwidth with memory throughput. This paper introduces a novel, scalable hardware sorter architecture with fully-streaming support and an accompanying RTL generator to provide versatile, energy-efficient hardware acceleration. Our solution employs a dual-layer architecture consisting of a parallel one-way linear insertion sorter (OLIS) for bandwidth optimization and a cyclic bitonic merge network (CBMN) for a compact, high-throughput implementation. Furthermore, we developed the RTL generator written in Chisel to provide the agile implementation of the scalable architecture. Experimental results targeting the Xilinx XVU37P-FSVH2892-2L-E FPGA show that our design achieves up to 126.26% increase in throughput and 68.46% decrease in latency, with an area increment of no more than 132.94% for LUTs, and a decrement of up to 79.84% for flip-flops, compared to state-of-the-art streaming sorter.**

***Index Terms*—sorting network, hardware acceleration, scalable architecture, bitonic sort, energy-efficient computing**

I. Introduction

Sorting is integral to numerous computing applications today. The demand for quick and resource-efficient sorting has spurred extensive research into enhancing the speed of sorting operations while minimizing resource consumption. Hardware acceleration has emerged as a promising avenue for achieving high-performance, energy-efficient sorting operations [1].

Bitonic sort, with its high throughput derived from an algorithm optimized for fully parallel streaming execution, stands as a key method in hardware sorting [2]. However, its practical application is hampered by significant design challenges. The implementation demands extensive area due to the compare-and-swap (CAS) operations and memory scaling with $O(n \log^2 n)$, resulting in a substantial resource usage [1], [3]. Moreover, these variants require equal input and output bandwidths, corresponding to the count of sorted elements, for maximal throughput. This requirement leads to the need for multiple parallel access of memory blocks, a configuration that standard host processors cannot easily accommodate, often causing severe communication bottlenecks and reducing the effectiveness of expansive sorting networks. Despite emerging advances such as high-bandwidth memory (HBM) [4] or in-memory computing [5], [6], these limitations persist in the bitonic variant sorter designs.

In response, numerous studies have sought to diminish area usage while preserving the high throughput characteristics of bitonic sort variants [3], [7]–[9]. A notable direction is the development of parallel merge tree (PMT) variants [7]–[9], which utilize smaller bitonic networks for the merge sorting of pre-sorted data bundles, achieving area efficiency alongside high throughput. Despite these advances, limitations remain, such as the lack of fully-streaming support and the prerequisite preprocessing of data bundles.

Conversely, linear insertion sorters, with their $O(n)$ area and $O(1)$ throughput scaling, represent one of the most resource-efficient sorter designs [10]–[12]. However, this efficiency comes at the cost of reduced throughput for large datasets and suboptimal performance in pipelining or parallel processing scenarios. Still, in specific contexts, such as preprocessing for the PMT variants, linear sorters offer considerable value [13].

This paper introduces DL-Sort, a scalable dual-layer hardware sorter that synergizes the strengths of linear insertion sorters and bitonic sorters while addressing their respective limitations. Our approach integrates a parallel one-way linear insertion sorter (OLIS) optimized for streaming and a cyclic bitonic merge network (CBMN) tailored for compactness, high throughput, and reduced latency. DL-Sort is designed to support fully-streaming operations under certain configurations, providing an area-efficient and high-performance sorting solution. We have developed an RTL generator in Chisel [14] for flexible deployment across various settings. The performance and efficacy of our design are confirmed through FPGA implementations with multiple configurations.

II. Sorter Microarchitecture

A. *One-way Linear Insertion Sorter*

Fig. 1 depicts the OLIS architecture, which compares incoming data with all previously stored data in a single cycle. The OLIS leverages a series of daisy-chained blocks, enabling the positioning of a new entry into its correct location. Sorting a bundle completes instantaneously as each entry is processed in one cycle when the write enable (*wen*) signal is set.

The design of the compare block is illustrated in Fig. 2. Taking current input entry and previously stored data as inputs to the comparator, the compare block determines if the new data should be inserted in its position, generating a comparison result indicator (*cmp_result*) for subsequent blocks. When the *cmp_result* is set by any preceding block, the entry is shifted.

The OLIS distinguishes itself from earlier linear sorter designs through several streaming-optimized features. Each

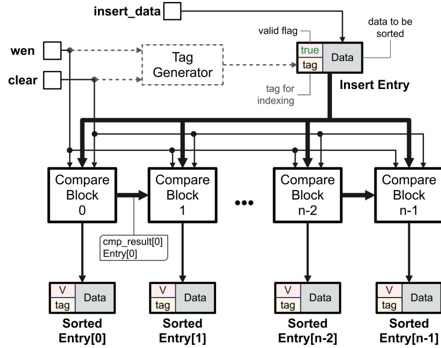

Fig. 1: The architecture of the one-way linear insertion sorter.

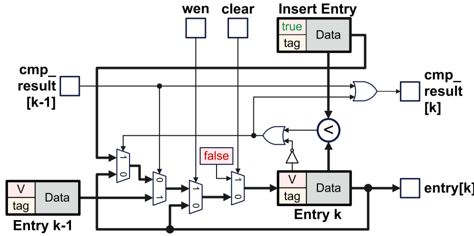

Fig. 2: Compare block component design.

entry within the sorter is marked with a valid flag to signify whether the entry is occupied. Furthermore, entries are cleared in a single cycle by resetting these flags upon a global clear signal, facilitating the continuous processing of data stream.

Additionally, the OLIS includes a configurable function for incremental tag generation, particularly useful in situations where only indexes are needed. This feature enhances the throughput and minimizes latency during the output stage, as it allows the transmission of multiple tags over a single streaming bus due to their reduced bitwidth.

B. Cyclic Bitonic Merge Network

Fig. 3 illustrates the of the CAS unit design, the core component of the CBMN. Our design enhances the previous CAS unit by adding a validity check, allowing the prioritization of valid entries. This enables the CBMN to process partially filled sorting entries without reconfiguration, which is advantageous for not only the FPGA but also the ASIC implementations.

The CBMN architecture, depicted in Fig. 4, represents an approach to hardware generation for sorting networks. This design notably decreases the number of CAS units required by utilizing them in a recursive manner. By integrating MUXes at each input and output of the CAS units, the area scales $O(n)$ rather than $O(n \log^2 n)$.

The initial phase in configuring the CBMN includes creating the bitonic network as presented in Fig. 4(a). This network is transformed into separate input and output permutation matrices (Fig. 4(b)), using a methodology referenced in [15]. These matrices are then combined using matrix transposing and multiplication, optimizing the pathway each data element takes through the network. The resulting combined matrix is translated into a series of MUX configurations (Fig. 4(c)). This

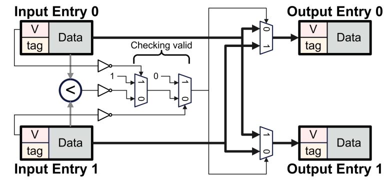

Fig. 3: The compare-and-swap unit design.

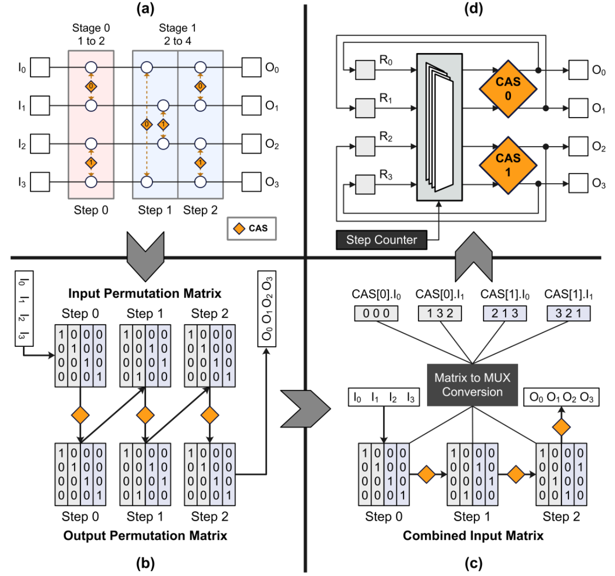

Fig. 4: The overview of the cyclic bitonic merge network generation process. **(a)** Network generation. **(b)** Permutation matrix generation. **(c)** Permutation optimization. **(d)** Automated hardware generation.

strategy markedly diminishes the area footprint of the CBMN by replacing additional CAS units with simpler MUXes, which are inherently more area and power-efficient.

Fig. 4(d) showcases the finalized CBMN microarchitecture, which comprises n/2 CAS units and $n \cdot w_E$ input MUXes, where w_E denotes the bitwidth of an entry. The area of MUXes, which are calculated by the MUX type and count, scales to $O(n \log n)$, reflecting the number of merging steps required. For instance, a merging phase consolidating data from two sources with two sorted data into four (a 2 to 4 merge) would utilize four 2-to-1 MUXes. As MUXes require fewer resources than CAS units, the architecture ensures optimized resource usage.

III. Proposed Sorter Architecture

The sorter is composed of two distinct sorting layers, each configured according to the parallel streaming width (P) and the total element count (E) parameters, which are constrained to be powers of two. Fig. 5 showcases the DL-Sort architecture for configurations with $E = 8$ and $P = 4$.

The first layer consists of P OLIS units that sort (E/P) data entries, with each OLIS receiving data via its own streaming bus. This arrangement enables the continuous processing of

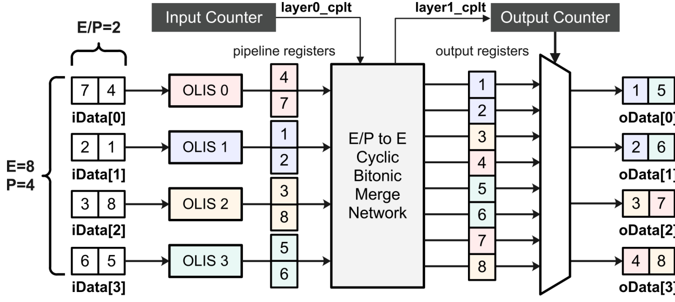

Fig. 5: Proposed scalable dual-layer sorter architecture.

incoming data in chunks, each containing E data entries. The OLIS units are optimized to replace the early stages of the bitonic sort algorithm, thus achieving lower latency and constant throughput. The additional area required for the OLIS units is counterbalanced by a reduction in the number of multiplexers (MUX), resulting from the fewer stages in the CBMN. After all data are received, the layer transfers the P sorted bundles of E/P entries each to pipeline registers and resets the OLIS units to process subsequent data chunks.

The second layer is comprised of the E/P to E CBMN. This layer takes in the sorted bundles from the pipeline registers and merges them through several iterations at each sorting step. The number of required steps (N_s) for merging from E/P to E is calculated as follows:

$$N_s = \frac{2\log_2 E - \log_2 P + 1}{2} \cdot \log_2 P \qquad (1)$$

For a fully-streaming operation, where the system processes input data continuously without interruption, it is attainable when $N_s \leq E/P$. This condition is met because the output from the OLIS layer is generated every E/P cycle. In scenarios where $N_s > E/P$ and data continue to be received, the CBMN layer cannot complete its merging operation before the next OLIS output is ready. Nevertheless, the N_s value generally scales as $O(\log E \cdot \log P)$, suggesting that numerous configurations can support fully-streaming operations, particularly when P is substantially less than E.

Finally, the CBMN layer transfers the fully sorted data to the output registers. These data are then sequentially dispatched using an output counter and P multiplexers, each configured as an E/P-to-1 MUX.

IV. Implementation & Evaluation

A. Implementation

The DL-Sort architecture was initially verified through register-transfer level (RTL) simulation using Verilator version 5.009. The RTL generator, crafted in Chisel, allowed the creation of flexible sorting architectures tailored to various E configurations while maintaining other parameters as $P = 4, w_D = 32, w_T = 0$, where w_D and w_T denote the bitwidth of the data and tag field, respectively.

B. Evaluation Methodology

The evaluation was framed as a comparative analysis against state-of-the-art scalable sorter featuring a complete sorting mechanism [13]. This referenced architecture, incorporating a linear insertion sorter alongside the PMT variant named FLiMS [8], demonstrated substantial throughput of up to 49× compared to the cortex-A53 core on Zynq Ultrascale+ ZU3EG device, with $E = 256$, $P = 4$ configuration. The RTL source of [13] was obtained from the generator provided in [16]. Notably, we excluded the buffering component of the [13] design to ensure an equitable comparison on area analysis.

The FPGA implementation results are shown in Table I. Each result is obtained by running 512 iterations of sorting E amount of data and is organized with both physical traits such as look-up tables (LUTs), flip-flops (FFs), maximum frequency (f_{MAX}), and power, and performances such as latency and throughput for both functional level and actual level. The result of the benchmark sorter is split into minimum and maximum cases due to the lack of fully-streaming support, showing different results reflecting the dataset composition.

C. Analysis

The performance analysis of the DL-Sort in Fig. 6 indicates several key outcomes. In terms of area usage, the DL-Sort exhibits the compact design for smaller E configurations ($E = 8, 16$). Especially on $E = 32$, the DL-Sort requires 28.77% more LUTs and 48.59% less FFs while achieving the 104.84% enhanced throughput. This is because the required cycles for the OLIS ($E/P = 8$) and the CBMN ($N_s = 9$) layers are similar, minimizing their idle state. This trend presents the area efficiency, which is crucial to meet resource constraints.

In terms of latency, our sorter has better performance compared to the previous work, showing the latency reduction up to 68.46% ($E = 8$, worst case of [13]) and least 4.61% ($E = 512$, best case of [13]). On average, the DL-Sort achieved 52.52% and 30.01% reduced latency compared to the worst case and best case of [13], respectively. These results highlight the potential for latency-critical applications.

In terms of throughput, with higher maximum frequency (f_{MAX}), our sorter achieved better throughput for most conditions except $E = 8$ configuration. Considering the worst case of the previous work, the DL-Sort achieves better performance for larger E values, showing the suitability for applications demanding high performance and consistent sorting. The highest throughput enhancement results from $E = 256$ configuration, showing 126.26% enhancement compared to the worst case of [13]. The average enhancements were 27.38% and 81.83% for the best and worst cases, respectively.

In terms of power efficiency, our design shows less performance for most configurations compared to the previous work. As the E parameter increases, the power consumption of DL-Sort also rises, surpassing that of [13]. Nevertheless, the $E = 32$ configuration, also mentioned before, shows better power efficiency as our sorter almost doubles the throughput while consuming only 16.85% more power. This suggests that minimizing the idle state will result in better power efficiency.

TABLE I: Overall performance breakdowns of DL-Sort and [13], implemented on XVU37P-FSVH2892-2L-E FPGA

E	Sorter	Physical traits				Function level performance					Actual performance			
		LUTs (1,303,680)	FFs (2,607,360)	f_{MAX} (MHz)	Power (W)	Latency (cycles) Min	Max	Entry/cycle Min	Max	Fully Streaming	Latency (nS) Min	Max	Perf. (GB/s) Min	Max
8	[13]	2,827 (0.22%)	3,959 (0.15%)	288.85	3.214	12	16	3.95	3.96	no	41.54	55.39	4.57	4.57
	DL-Sort	1,134 (0.09%)	798 (0.03%)	515.20	3.269	9	9	1.99	1.99	no	17.47	17.47	4.10	4.10
16	[13]	3,211 (0.25%)	4,823 (0.18%)	267.95	3.236	16	20	2.89	3.97	no	59.71	74.64	3.10	4.26
	DL-Sort	2,386 (0.18%)	1,690 (0.06%)	457.67	3.425	15	15	2.65	2.65	no	32.78	32.78	4.86	4.86
32	[13]	4,839 (0.37%)	6,555 (0.25%)	238.78	3.264	24	32	3.35	3.98	no	100.51	134.02	3.20	3.80
	DL-Sort	6,231 (0.48%)	3,370 (0.13%)	412.54	3.814	25	25	3.98	3.98	no	60.60	60.60	6.56	6.56
64	[13]	7,516 (0.58%)	10,027 (0.38%)	236.74	3.332	40	56	3.64	3.99	no	168.96	236.54	3.45	3.78
	DL-Sort	13,953 (1.07%)	6,532 (0.25%)	340.37	4.648	43	43	3.98	3.98	yes	126.33	126.33	5.42	5.42
128	[13]	13,430 (1.03%)	17,070 (0.65%)	198.33	3.383	72	111	2.63	3.99	no	363.02	559.66	2.09	3.17
	DL-Sort	26,317 (2.02%)	13,055 (0.50%)	292.74	5.830	77	77	3.98	3.98	yes	263.03	263.03	4.66	4.66
256	[13]	24,652 (1.89%)	31,212 (1.20%)	184.91	3.495	136	223	2.05	3.99	no	735.49	1205.98	1.52	2.95
	DL-Sort	53,558 (4.11%)	25,382 (0.97%)	215.84	8.028	143	143	3.98	3.98	yes	662.52	662.52	3.44	3.44
512	[13]	46,635 (3.58%)	59,801 (2.29%)	128.63	3.655	264	447	2.01	3.99	no	1913.47	3239.86	1.11	2.20
	DL-Sort	108,631 (8.33%)	50,552 (1.94%)	149.57	10.105	273	273	3.98	3.98	yes	1825.28	1825.28	2.38	2.38

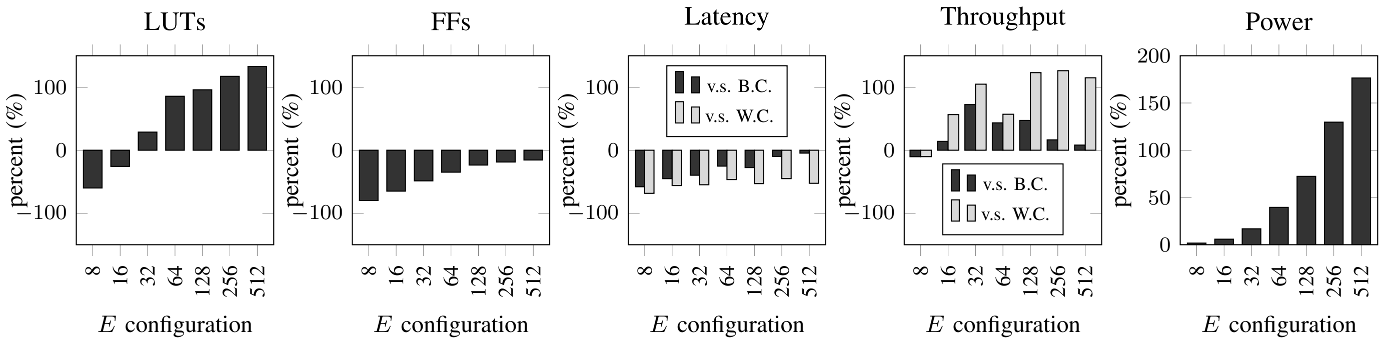

Fig. 6: Comparative analysis on the DL-Sort and the baseline sorter [13].

D. Discussion

The DL-Sort architecture has demonstrated robustness and efficiency across various configurations. The area efficiency, as evidenced by the reduced use of LUTs and FFs, particularly in smaller E configurations, is a testament to the architecture's compact design. It showcases the potential for applications where area footprint is prioritized.

The latency improvements are significant, showing the suitability for latency-critical applications. The reduction in latency without an increase in power efficiency for the $E = 32$ is particularly noteworthy and suggests that the DL-Sort can achieve high throughput without sacrificing power efficiency.

While the DL-Sort generally consumes more power than the previous work across most configurations, the increase in throughput for the $E = 32$, coupled with only a modest increase in power, indicates that there are sweet spots where the performance-to-power ratio is optimized. This balance is crucial for sustainable, high-performance computing and warrants further investigation into the scalability of power efficiency across other configurations.

The throughput improvements in larger configurations position DL-Sort as a compelling solution for high-performance applications. The architecture's ability to maintain and even enhance throughput as the complexity of the sorting task increases shows promise for use in data-intensive domains.

V. Conclusion

We introduced DL-Sort, a scalable hardware sorter architecture designed for high-speed, efficient, and fully-streaming sorting. Our sorter has proven to be a competitive and viable solution, offering high-speed, efficient sorting coupled with consistent performance. With enhanced throughput, reduced latency, and acceptable area and power efficiency, our sorter is an attractive alternative to traditional hardware sorters. The scalable architecture with hardware generator makes the DL-Sort a suitable option for a wide range of applications. The DL-Sort architecture exemplifies a stride in hardware sorting acceleration, adeptly meeting the evolving demands of modern computational challenges.

In future work, we will focus on enhancing power efficiency, seeking architectural modifications to reduce energy dissipation without diminishing performance. Additionally, the potential integration of the DL-Sort architecture into system-on-chip platforms using various interfacing concepts will be explored, aiming to leverage hardware sorting in a wider spectrum of computing environments. These forthcoming endeavors will propel our studies forward, solidifying the role as a cornerstone in the hardware sorting acceleration optimized for resource-constrained systems.

References

[1] M. Zuluaga, P. Milder, and M. Püschel, “Streaming Sorting Networks,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 21, no. 4, pp. 1–30, Sep. 2016.

[2] R. Chen, S. Siriyal, and V. Prasanna, “Energy and Memory Efficient Mapping of Bitonic Sorting on FPGA,” in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2015, pp. 240–249.

[3] A. Norollah, D. Derafshi, H. Beitollahi, and M. Fazeli, “RTHS: A Low-Cost High-Performance Real-Time Hardware Sorter, Using a Multidimensional Sorting Algorithm,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 7, pp. 1601–1613, Jul. 2019.

[4] W. Qiao, L. Guo, Z. Fang, M.-C. F. Chang, and J. Cong, “TopSort: A High-Performance Two-Phase Sorting Accelerator Optimized on HBM-Based FPGAs,” *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 2, pp. 404–419, 2023.

[5] Z. Li, N. Challapalle, A. K. Ramanathan, and V. Narayanan, “IMC-Sort: In-Memory Parallel Sorting Architecture using Hybrid Memory Cube,” in *Proceedings of the 2020 on Great Lakes Symposium on VLSI (GLSVLSI)*, Virtual Event, China, Sep. 2020, pp. 45–50.

[6] N. Samardzic, W. Qiao, V. Aggarwal, M.-C. F. Chang, and J. Cong, “Bonsai: High-Performance Adaptive Merge Tree Sorting,” in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, Valencia, Spain, May 2020, pp. 282–294.

[7] W. Song, D. Koch, M. Lujan, and J. Garside, “Parallel Hardware Merge Sorter,” in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Washington, D.C., USA, May 2016, pp. 95–102.

[8] P. Papaphilippou, W. Luk, and C. Brooks, “FLiMS: A Fast Lightweight 2-Way Merger for Sorting,” *IEEE Transactions on Computers*, vol. 71, no. 12, pp. 3215–3226, 2022.

[9] S. Mashimo, T. Van Chu, and K. Kise, “High-Performance Hardware Merge Sorter,” in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, USA, Apr. 2017, pp. 1–8.

[10] R. Perez-Andrade, R. Cumplido, C. Feregrino-Uribe, and F. Martin Del Campo, “A versatile linear insertion sorter based on an fifo scheme,” *Microelectronics Journal*, vol. 40, no. 12, pp. 1705–1713, Dec. 2009.

[11] J. Ortiz and D. Andrews, “A configurable high-throughput linear sorter system,” in *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, Atlanta, GA, USA, Apr. 2010, pp. 1–8.

[12] C.-Y. Lee and J.-M. Tsai, “A shift register architecture for high-speed data sorting,” *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 11, no. 3, pp. 273–280, 1995.

[13] P. Papaphilippou, C. Brooks, and W. Luk, “An Adaptable High-Throughput FPGA Merge Sorter for Accelerating Database Analytics,” in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, Gothenburg, Sweden, Aug. 2020, pp. 65–72.

[14] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, “Chisel: Constructing Hardware in a Scala Embedded Language,” in *Proceedings of the 49th Annual Design Automation Conference (DAC)*, San Francisco, CA, USA, 2012, p. 1216–1225.

[15] M. Püschel, P. A. Milder, and J. C. Hoe, “Permuting streaming data using RAMs,” *Journal of the ACM*, vol. 56, no. 2, pp. 1–34, Apr. 2009.

[16] P. Papaphilippou. [Online]. Available: https://philippos.info/sorter/