Late Breaking Results: A Fast and Low-Cost Comparison-Free Sorting Engine with Unary Computing

Amir Hossein Jalilvand*, Seyedeh Newsha Estiri*, Samaneh Naderi+, M. Hassan Najafi*, and Mohsen Imani†
*University of Louisiana at Lafayette, +Iran University of Science and Technology, †University of California Irvine
Corresponding Author: najafi@louisiana.edu

ABSTRACT

Hardware-efficient implementation of sorting operation is crucial for numerous applications, particularly when fast and energy-efficient sorting of data is desired. Unary computing has been used for low-cost hardware sorting. This work proposes a comparison-free unary sorting engine by iteratively finding maximum values. Synthesis results show up to 81% reduction in hardware area compared to the state-of-the-art unary sorting design. By processing right-aligned unary bit-streams, our unary sorter is able to sort many inputs in fewer clock cycles.

1 INTRODUCTION

Sorting is essential for numerous applications, including image processing, artificial intelligence, task scheduling, scientific computing, etc. For high-performance sorting, sorting is performed in hardware with application-specified integrated circuits or field-programmable gate arrays. Hardware-based sorting is fundamentally different from software-based sorting such as QuickSort, MergeSort, BubbleSort, etc. In software sorting, the order of comparisons depends on data. But, in hardware sorting, this order is fixed and is independent of data. The number of sorting operations can vary significantly from application to application. For example, in image processing applications, thousands of inputs may need to be sorted. Therefore, an optimal hardware implementation of sorting operation is of great importance.

There is a relatively large body of work for hardware-based sorting [3]. The ultimate goal is to sort data with minimum latency and hardware cost. One of the most popular approaches is Batcher's sorting [3]. Batcher wires up a network of compare-and-swap (CAS) units, which can be pipelined easily. The hardware cost and the power consumption of Batcher's network depend on the number of CAS blocks and the cost of each CAS block. Each CAS block compares two input values and swaps the values at the output if needed. The total number of CAS blocks in an N-input Batcher's sorting is $N \times log_2(N) \times (log_2(N)+1)/4$. Thus, 8-, 16-, 32-, and 256-input Batcher networks require 24, 80, 240, and 4,608 CAS blocks, respectively [2].

Batcher's sorting is conventionally implemented based on the weighted binary representation. Binary representation is compact; however, computation on this representation is relatively complex. The complexity increases by increasing the data-width. Increasing the complexity affects the cost of hardware implementation, latency, power, and hence, energy consumption. Najafi *et al.* proposed an alternative low-cost hardware design for Batcher's networks using *unary computing* [5]. In unary computing, numbers are encoded uniformly by a sequence of one value (say 1) followed by a sequence of the other value (say 0) with the data value determined by the fraction of 1's in the sequence. For example, 11000 is a left-aligned unary sequence (i.e., *bit-stream*) representing 0.4. The minimum and the maximum value functions, the essential functions in building Batcher sorting networks, can be realized efficiently in the unary domain using simple bit-wise AND and OR operations. An area and power saving of up to 92% is reported in [5] for the unary Batcher sorting design compared to the conventional binary counterpart.

The hardware design of a comparison-free sorting engine is proposed in [4]. Their design sorts N data elements in nearly N clock cycles while recognizing the maximum number in the 1st clock cycle. Their sorting engine is constructed by employing N symmetric cascaded blocks, and sorting operations are performed in a pipelined fashion. A comparison-free sorting algorithm is also introduced in [1]. This design can be applied to any data distribution with no significant adjustment. The number of required cycles falls in the range of $2N$ to $2N + 2K$ 1, where K is the bit-width of data and N is the number of input data.

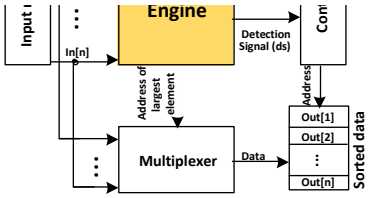

Fig. 1: High-level Architecture of Comparison-Free Unary Sorter.

This work proposes a fast and low-cost comparison-free sorting architecture based on unary computing. We iteratively find the index of the maximum value by converting data to left-aligned unary bit-streams and finding the first "1" in the generated bit-streams. Our synthesis results show a significant area reduction, up to 81%, compared to the state-of-the-art unary sorting design of [5] and up to 45% compared to the comparison-free design of [4]. The proposed sorter sorts many inputs in fewer clock cycles compared to the unary design of [5].

2 COMPARISON-FREE UNARY SORTER

Here we describe our proposed comparison-free unary sorting design. The high-level architecture is shown in Fig. 1. The architecture includes a sorting engine, a controller, and a multiplexer. The design reads unsorted data from the input registers and performs sorting by finding the address of the maximum number at each step. Fig. 2 shows the proposed sorting engine. In the first step, the sorting engine converts data to right-alighted unary bit-streams and returns the index of the bit-stream corresponding to the maximum value. This is done by finding the bit-stream that produces the first 1. Consider a set of inputs, $p_1 = 0.4$, $p_2 = 0.2$, $p_3 = 0.8$, $p_4 = 0.6$, $p_5 = 0$ 2, and $p_6 = 0.8$. A right-aligned unary representation for these numbers is $p_1 = 00011$, $p_2 = 00001$, $p_3 = 01111$, $p_4 = 00111$, $p_5 = 00001$ and $p_6 = 01111$. In the first cycle, the *shared down counter* starts counting down and a zero bit is generated for all inputs. In the second cycle, a one is generated for the third (p_3) and the last (p_6) input. This enables the flip-flops corresponding to the third and last inputs. When these flip-flops are activated, the detection signal (ds), which is the output of an *addition* unit, will have a value of two. $ds = 2$ means that the next maximum value is not a single number but two numbers with the same value. We utilize a *priority encoder* to obtain the memory address of one of the maximum values in the second cycle. Next, ds is passed to the *controller*. The controller's finite state machine is shown in Fig. 3. When $ds = 2$, the state changes from "Find the index" to "Put the results." The state does not change until the two numbers are in

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530615

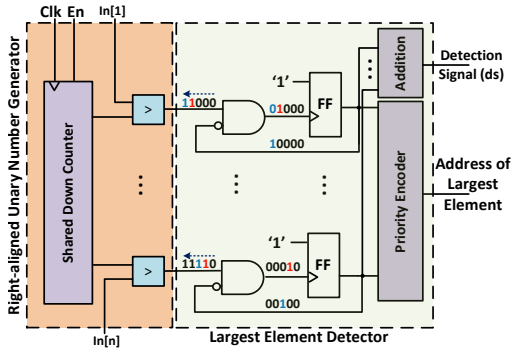

Fig. 2: Proposed Unary Sorting Engine.

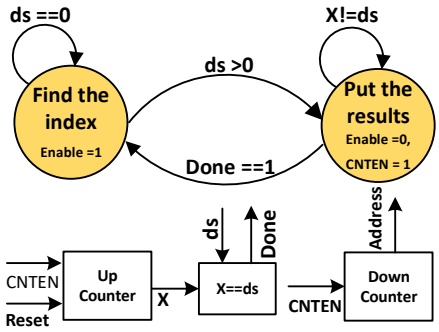

Fig. 3: Proposed Controller.

the output. Then, the controller enables a *down counter* that gives the address of the numbers found in the "Put the result" state. A *multiplexer* is also used to get the address of the maximum values from the sorting engine and put the weighted-binary value of the maximum on the output registers.

3 DESIGN EVALUATION

We developed a VHDL description of the proposed comparison-free sorter and synthesized it using the Synopsys Design Compiler O-2018.06-SP2 with a 45-nm standard cell library. We compare our unary sorting design with the unary design of the Batcher networks proposed in [5] and also the comparison-free sorter developed in [4]. Similar to our method, the design of [4] finds the index of the maximum value. We report the hardware cost for three different data widths of 8, 16, and 32 bits. Table 1 reports the synthesis results for all implemented designs in terms of hardware area footprint, critical path latency, and power consumption at maximum working frequency. As the results show, the proposed design achieves up to 81% and 46% area saving compared to the designs of [5] and [4], respectively.

The downside of unary computing designs is long computation time, which translates to high energy consumption. Like Bubble-sort, our proposed design finds the maximum values and sends them out iteratively. Contrary to prior unary designs that process K-bit data in 2^K clock cycles, our unary sorter finds the maximum value as soon as it sees the first 1. This can reduce the number of clock cycles to sort many inputs. To find the latency and energy saving of our approach in finding the maximum and minimum values, we developed a MATLAB program to find the average number of clock cycles for finding the maximum and minimum values in 1000 sets of random data. Table 2 compares the average number of clock cycles for our method and the method of [5] when processing 8-bit precision data. As can be seen, our proposed architecture can find the maximum value, on average, after three cycles for the 256-input design. This provides a significant energy saving in finding the maximum value. The amount of energy consumption can be found by finding the product of the critical path latency, the number of clock cycles, and the power consumption of the design.

Table 1: Synthesis Results of the Implemented Designs.

Input	M	Area (μm^2)			Critical Path (ns)			Power (mW) @ Max freq.		
		[4]	[5]	**Prop.**	[4]	[5]	**Prop.**	[4]	[5]	**Prop.**
8	8	1,752	2,659	**1,536**	1.33	0.39	**0.22**	0.23	3.29	**4.40**
	16	2,463	5,024	**2,113**	2.42	0.42	**0.22**	0.14	5.59	**5.91**
	32	4,458	9,916	**4,243**	4.28	0.49	**0.22**	0.10	10.1	**8.29**
16	8	3,155	5,834	**2,282**	1.63	0.4	**0.22**	0.35	5.3	**5.14**
	16	5,169	10,323	**3,939**	2.98	0.44	**0.22**	0.22	8.94	**6.44**
	32	9,630	18,065	**7,357**	5.22	0.5	**0.22**	0.18	15.9	**9.86**
32	8	5,546	13,095	**4,079**	2.02	0.41	**0.22**	0.51	8.4	**5.68**
	16	9,262	17,029	**7,155**	3.92	0.46	**0.23**	0.31	13.8	**7.54**
	32	19,282	29,682	**12,268**	6.07	0.5	**0.23**	0.29	25.4	**12.24**
64	8	10,093	25,248	**7,498**	2.42	0.44	**0.24**	0.80	13.4	**6.06**
	16	19,003	37,726	**11,350**	4.02	0.47	**0.24**	0.58	22.5	**9.13**
	32	37,031	63,144	**21,210**	6.92	0.5	**0.25**	0.50	41.2	**15.34**
128	8	19,417	59,579	**13,039**	2.73	0.47	**0.24**	1.34	21.4	**10.41**
	16	37,128	84,646	**22,615**	4.73	0.5	**0.24**	0.98	37.1	**16.10**
	32	84,657	134,746	**45,866**	8.04	0.52	**0.25**	1.07	69.1	**29.31**
256	8	39,155	140,006	**25,614**	2.93	0.49	**0.24**	2.48	36.5	**16.40**
	16	74,170	189,903	**42,616**	5.62	0.51	**0.24**	1.68	62.1	**27.01**
	32	103,622	289,723	**82,694**	10.02	0.54	**0.25**	1.14	113	**51.76**

Table 2: Latency and Energy Consumption Comparison of the Proposed and Prior Unary Design [5].

Input	Required Cycle			Energy (pj)		
	Prior min/max	Prop. min	Prop. max	Prior min/max	Prop. min	Prop. max
8	256	228.8	29.1	328.47	**221.61**	**28.25**
6	256	241.5	16.0	542.72	**273.18**	**18.16**
32	256	248.3	9.06	881.66	**310.31**	**11.32**
64	256	252.4	5.21	1,509.38	**367.09**	**7.58**
128	256	254.3	3.55	2,575.85	**635.54**	**8.87**
256	256	255.4	2.56	4,578.56	**1,005.33**	**10.08**

4 CONCLUSION

This work proposed a fast and low-cost comparison-free sorting design based on unary computing. The proposed design generates right-aligned unary bit-streams to speed up finding the bit-stream with maximum value. The sorter finds duplicate numbers by using a detection signal. A controller returns the memory address of the found maximum values. Our proposed design can decrease the number of clock cycles for sorting many input data. The hardware cost is further significantly reduced compared to prior unary and comparison-free binary sorting designs.

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation (NSF) grants #2127780 and #2019511, Semiconductor Research Corporation (SRC) Task #2988.001, Office of Naval Research, grants #N00014-21-1-2225 and #N00014-22-1-2067, Air Force Office of Scientific Research, the Louisiana Board of Regents Support Fund #LEQSF(2020-23)-RD-A-26, and a generous gift from Cisco.

REFERENCES

[1] S. Abdel-Hafeez and A. Gordon-Ross. An Efficient O(N) Comparison-Free Sorting Algorithm. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(6):1930–1942, jun 2017.

[2] F. Amin, H. J. Duwe III, M. J. Schulte, and K. Compton. Modular design of high-throughput, low-latency sorting units. *IEEE Transactions on Computers*, 62(7):1389–1402, 2013.

[3] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference on - AFIPS '68 (Spring)*, page 307, New York, New York, USA, 1968. ACM Press.

[4] S. Ghosh, S. Dasgupta, and S. Saha Ray. A Comparison-free Hardware Sorting Engine. *Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI*, 2019-July:586–591, 2019.

[5] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan. Low-Cost Sorting Network Circuits Using Unary Processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(8):1471–1480, aug 2018.