Recursive Implementation of Voting Networks

Behrooz Parhami
Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106-9560
parhami@ece.ucsb.edu

***Abstract*—Recursive synthesis of digital circuits leads to systematic design methods, reuse of building blocks, and clean mathematical models for circuit cost and delay. Recursive integer and matrix multiplications, and Fourier transform, are familiar examples. In this paper, we show that threshold voters, including the important special case of majority voters, can be synthesized from smaller networks of the same kind in a simple and easily analyzable way. At the end of the recursion, we get to readily-available AND & OR gates, 3-input counters (or full-adders), and majority circuits, which are realizable in a variety of platforms, including emerging atomic-scale digital technologies.**

Keywords— Majority gate, nanoelectronics, parallel counter, threshold circuit, VLSI-friendly design, voter, weight checker

I. Introduction

This paper relates to a previous study of building large majority and plurality voting circuits [1]. At the time of the previous study, interest in voters with large numbers of inputs was mostly theoretical, as practical voters had 3-5 inputs. Today, we need larger voters, owing to imperfection, and thus extreme error rates, in nanoelectronic circuits. Redundancy factors of several-dozen have already been proposed and even higher redundancy may be needed with further circuit shrinkage and the accompanying variabilities and uncertainties [2] [3]. Reliability problems are also front-and-center in quantum computing [4] [5], but it is unclear at this time whether they will be eventually overcome by massive replication or through domain-specific methods of encoded computation.

At the same time, the emergence of majority-friendly technologies motivates the majority-gate-based synthesis of logic functions, thus rekindling interest in the field of threshold logic [6], [7]. Any symmetric function of n logical variables can be implemented by first counting the number of 1s among the inputs and then using the count to form the output.

Recursive design methods allow testing and verification to be pursued even in the early stages of design, a boon for safety-critical and high-stakes applications such as medical monitoring and aerospace [8]. Furthermore, recursive designs, for both hardware and algorithms, tend to be simple and clean, easy to reason about or prove correct, and subject to major overall enhancement with a small improvement in each recursion stage. Examples include recursive realizations of integer & matrix multiplication [9] [10] and Fourier transform [11], that have led, to Karatsuba-Ofman integer multiplication [12], Strassen's matrix multiplication [13], and Cooley-Tuckey FFT [14]. Recursive designs also facilitate the derivation of latency and circuit-complexity formulas and simplify comparison and trade-off studies via rigorous mathematical analysis.

The main contributions of this paper relative to existing designs and published research are as follows:

- We propose a recursive method for counting-based realization of symmetric functions, a class of digital circuits that includes majority and minority voters.
- We investigate the characteristics of different gate structures for use in implementing our designs in order to transfer the design benefits to actual circuit parameters.

This paper discusses preliminary ideas that are being refined and extended in several different directions, about which we will report in multiple planned papers.

II. Prior Work and Motivation

A. History and Significance

Voting networks, or voters for short, which determine the majority value, if it exists, among n inputs, have a long history. Because of their practical significance, voters have been the subjects of many research studies and implementations. Triple-modular redundancy with three-way voting has been used for many decades in various applications [15] [16]. Safety-critical systems are just beginning to use CMOS voters with 5 or more inputs [17] [18]. Voters with larger numbers of inputs have also been proposed for quantum-dot cellular automata [19].

Voters are key components in today's advanced circuit designs. Examples of such applications can be found in storage devices, where RAID algorithms are used alongside voter circuits for data recovery after a disk failure [20]. Moreover, the need for weight-checkers arises in authentication circuits for certain error-detecting and error-correcting codes and in network communications where packets are verified using signature approval [21]. Use of replication and voting to recover from single-event upsets and other disruptions due to device-level physical damage caused by high-energy particles is another area of application [22].

Critical applications in domains such as aerospace, nuclear power generation, electrical grids, banking & financial systems, and industrial control & automation usually incorporate physical redundancy to ensure a specific degree of fault tolerance for successfully overcoming faults or malfunctions in various function units. While the degree of redundancy has

979-8-3503-6013-4/24/$31.00 ©2024 IEEE

been fairly limited in the past, it is on the rise, as additional critical applications are automated and less-reliable nanoelectronic devices are deployed [23]. Recursive designs, discussed later in this paper are well-suited to such systems, owing to their high testability and reliability.

We have found only two prior attempts at recursive synthesis of voters. One goes only as far as a single level of recursion (it should thus be called unrolling, not recursion) to design voters with more than 3 inputs [24]. In the other [1], designs referred to as mux-based are shown to be superior to a number of other designs for majority voting with the then-practical smallish values of n. Here, we extend the design methods and assessments to an arbitrary number of inputs.

B. Pros and Cons of Recursive Design

There is some tendency to equate recursive algorithms with inefficiency. This is a remanent from early days of digital computers when computing power was expensive and thus in short supply. Programmers would spend much time converting a recursive algorithm to a non-recursive one through unrolling. The same would apply to replacing procedure calls with in-line code to save on a few call- and return-instructions. Things have changed since then. Computer architects now provide substantial support for efficient procedure calls and recursion.

For hardware, a similar pre-judgment exists, except that the concerns are even less relevant here. Recursively-defined hardware is always implemented with partial or even full unrolling. For example, an n-input FFT network is built as a combinational system for small-to-moderate n. For large n, partial unrolling may be utilized for reducing the cost through hardware time-sharing. Even in the latter case, the overhead will be limited to the cost and delay of storage elements, of the same kind routinely used between pipeline stages.

VLSI design considerations sometimes work against asymptotically-optimal designs, favoring instead designs with regularity, short interconnects, and balanced delay paths. After decades of research, it was proven in 2019 that the multiplication of n-digit numbers is an $O(n \log n)$ operation in terms of circuit cost [25], but in many practical settings, naive $O(n^2)$ designs or Karatsuba-Ofman's $O(n^{1.58})$ realization still prevail over the impractical $O(n \log n)$ scheme, or its $O(n \log n \log(\log n))$ predecessors.

Furthermore, whereas in mass-produced circuits speed and power consumption play outsize roles, thus making domain-specific designs highly desirable, in the case of limited-production circuits, the latter may take a back seat to the attributes of regularity, testability, and rapid realizability (viz., short time-to-market). In these domains, recursive design algorithms, by virtue of simplifying the design and analysis processes, provide useful tools in the designers' repertoire.

III. Recursive Threshold Counters

In this section, we discuss the design of ordinary $\geq l/n$ and inverse $< m/n$ threshold counters. The notation $\geq l/n$ represents the fact that the circuit determines whether at least l of its n inputs are 1s.

A. Ordinary Threshold Counters

A majority voting network realizes the h-out-of$-(2h-1)$ function, which assumes the value 1 when at least h of the $2h-1$ inputs are 1s. Let's generalize a tad and focus on realizing l-out-of-n threshold networks, that produce a 1 output when at least l of the n inputs are 1s, where n isn't necessarily $2l-1$ or another odd number.

Our method begins by isolating the last input x_n. If $x_n = 0$, then to satisfy the $\geq l/n$ output condition, the number of 1s among the remaining $n-1$ inputs must be at least l. If $x_n = 1$, then the number of 1s among the remaining inputs should be at least $l-1$. Unrolling the recursion, we will eventually arrive at the familiar building blocks of $\geq 1/h$ (OR), $\geq h/h$ (AND), and $\geq 2/3$ (basic majority). Circuit implementation of an example $\geq 3/5$ voting network, derived from two levels of recursion, is depicted in Fig. 1.

Clearly, we do not need to restrict ourselves to the use of 2-input multiplexers. For example, the two mux levels in Fig. 1 can be replaced with a 4-to-1 mux. The use of an 8-input mux for building a $\geq 4/7$ voting network is illustrated in Fig. 2. The use of larger muxes just discussed is also applicable to the designs that follow in the rest of the paper.

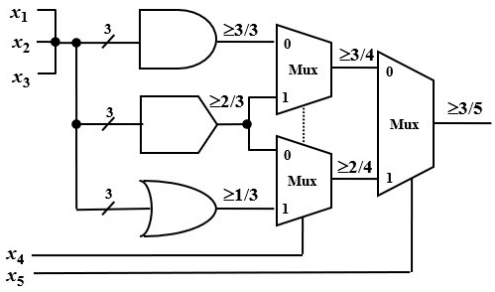

Fig. 1. Recursive synthesis result for a 3-out-of-5 threshold circuit, using two levels of unrolling that leads to two levels of multiplexers. An h-level binary tree of 2-input muxes can be replaced by a 2^h-input mux, if desired. In this diagram, we can use a 4-input mux in the last two circuit levels.

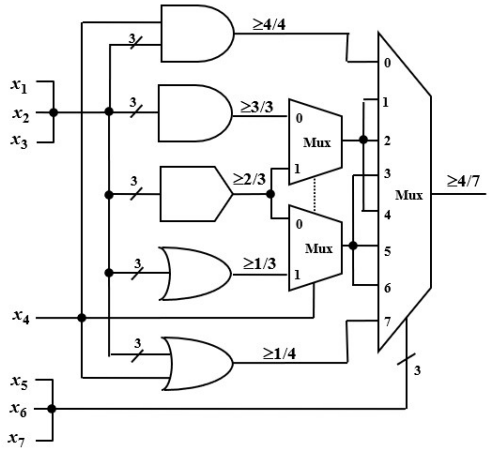

Fig. 2. Using an 8-input & two 2-input muxes to synthesize a 4-out-of-7 voter. The 5-sided box in the middle of the column of gates is a 2-out-of-3 majority gate, which can be synthesized from ordinary gates if not directly available.

B. Cost and Delay Analysis

Focusing on the recursive implementation with 2-way muxes, we can readily derive cost and delay formulas, assuming unit-cost and unit-delay 2-way muxes.

$$B(\geq l/n) = 1 + B(\geq l/(n-1)) + B(\geq (l-1)/(n-1)) - B(\geq (k-1)/(n-1))$$

$$\begin{aligned} B(\geq l/n) &= 1 + 2 + \cdots + (n-l-1) + (n-l) \quad [n-l \text{ terms}] \\ &\quad + (n-l) + \cdots + (n-l) \quad [2l-n-1 \text{ terms}] \\ &\quad + (n-l-1) + \cdots + 3 + 2 \quad [n-l-2 \text{ terms}] \\ &= (n-l)(l-1) - 1 \end{aligned}$$

Here are two numerical examples to test the formula for the number of muxes in the design:

$$\begin{aligned} B(\geq 5/9) &= 1 + B(\geq 5/8) + B(\geq 4/8) - B(\geq 4/7) \\ &= 1 + 11 + 11 - 8 = 15 \end{aligned}$$

$$\begin{aligned} B(\geq 6/11) &= B(\geq 6/10) + B(\geq 5/10) - B(\geq 5/9) \\ &= 1 + 19 + 19 - 15 = 24 \end{aligned}$$

To the result $B(\geq l/n)$ above, we must add the cost of AND and OR gates at the periphery to find the total cost *C*:

$$C(\geq l/n) = B(\geq l/n) + \Sigma_{i=3,l} C_{AND}(i) + \Sigma_{j=3,n-l-1} C_{OR}(j) + C_{Maj}$$

For delay, we have:

$$D(\geq l/n) = 1 + max[D(\geq l/(n-1)), D(\geq (l-1)/(n-1))]$$

$$D(\geq l/n) = n - 3 + \max(D_{AND}(3), D_{OR}(3), D_{Maj})$$

The example network of Fig. 3 is a $\geq 3/6$ threshold circuit, with 5 multiplexers and 3 mux levels. The $\geq 6/8$ threshold circuit example of Fig. 4 uses 9 muxes in 5 levels.

Were we not using majority elements as basic building blocks, the unrolling would continue for one more step:

$$D(\geq 2/3) = 1 + \max(D_{AND}(2), D_{OR}(2))$$

Similarly, for cost:

$$C(\geq 2/3) = 1 + C_{AND}(2) + C_{OR}(2)$$

The net result is the addition of 1 mux and two 2-input gates to network cost and 1 unit to network delay. But with our target implementation technologies [8], majority elements tend to be both faster than 3 gate-delays (counting 2 for a mux) and simpler than 5 gate-costs (counting 3 for a mux).

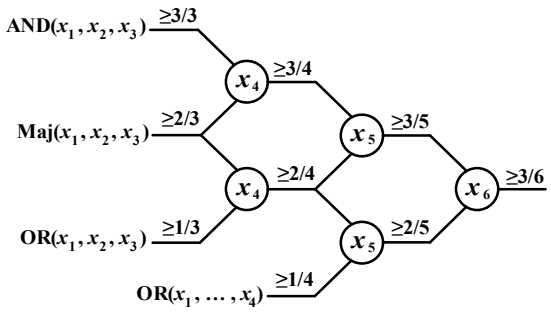

Fig. 3. An example threshold circuit, represented with our simplified notation.

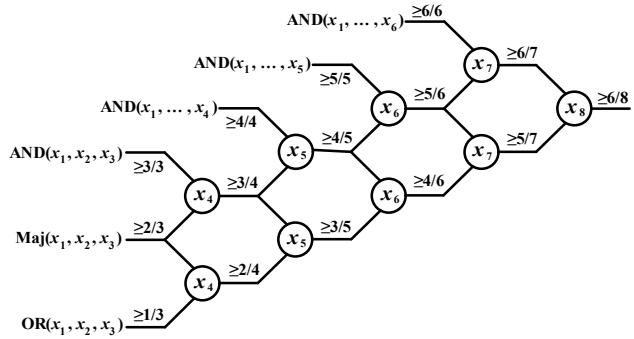

Fig. 4. Another example threshold counting network. This one determines whether at least 6 of its 8 inputs are 1s.

C. Inverse Threshold Counters

An ordinary $\geq l/n$ threshold counter has its output asserted whenever at least *l* of its *n* inputs are 1s. We can also define an inverse threshold circuit, or $< m/n$ counter, whose output is asserted if fewer than *m* of its *n* inputs are 1s. Clearly, a $< m/n$ inverse threshold circuit can be built by simply inverting the output of a $\geq m/n$ circuit. However, as the example of Fig. 5 confirms, when compared to Fig. 4, direct realization of an inverse threshold circuit using the techniques previously outlined can lead to a simpler design.

To confirm the observation above analytically, we write the expression for the number of muxes in a $< m/n$ circuit for values of m in the range $2 \leq m \leq n/2$:

$$\begin{aligned} B(< m/n) &= 1 + 2 + \cdots + (m-2) + (m-1) \quad [m-1 \text{ terms}] \\ &\quad + (m-1) + \cdots + (m-1) \quad [n-2m \text{ terms}] \\ &\quad + (m-1) + \cdots + 3 + 2 \quad [m-2 \text{ terms}] \\ &= (n-m)(m-1) - 1 \end{aligned}$$

Comparing the latter expression with our expression for $B(\geq l/n) = (n-l)(l-1)-1$, after replacing l with $n-m$ to get $m(n-m-1)-1$, we see that $B(< m/n) < B(\geq (n-m)/n)$ for all $m \leq n/2$. The number of edge gates is comparable for both designs. So, the direct realization of a $< m/n$ counter is beneficial for $2 \leq m \leq n/2$. Both designs require $n-3$ mux levels, plus one level of gates on the left edge.

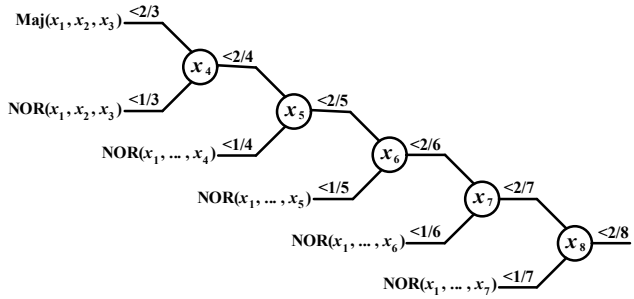

Fig. 5. Example of inverse threshold circuit that determines whether fewer than 2 of the 8 inputs are 1s. This circuit's output is the complement of the output of the threshold circuit $\geq 2/8$.

D. The Important Special Case of Voting

A simple majority voter, or simply voter, is a $\geq \lceil n/2 \rceil / n$ counter with an odd number n of inputs. This special case is of immense importance in the design of fault-tolerant systems relying on multi-channel computation and majority-based decisions to choose highly-reliable outputs. To convey the complexity & latency of an 11-input majority voter, we provide a $\geq 6/11$ example in Fig. 6. Simple voter layouts have vertical symmetry and, with the exception of the rightmost mux, horizontal symmetry, leading to more efficient layouts.

Majority voting need not be limited to an odd number of inputs. For example, a $\geq 4/6$ majority voter may be used to build a multi-channel computation scheme with greater reliability than one based on $\geq 3/5$ voting, without incurring the added cost of going to the higher redundancy level of a $\geq 4/7$ scheme. A simple majority voter with an even number n of inputs is $a \geq (n/2+1)/n$ counter.

Also, we do not have to limit voters to a simple majority. When more than a majority of inputs are required to be 1s for the output to be 1, we have a super-majority scheme, although super-majority is sometimes reserved for $2/3$ agreement among the inputs. Either way, using super-majority, which requires broader agreement among the input values, is appropriate for certain safety-critical systems, in which the output of 1 may be unsafe whereas 0 is safe. A $\geq 6/9$ two-thirds majority circuit is depicted in Fig. 7.

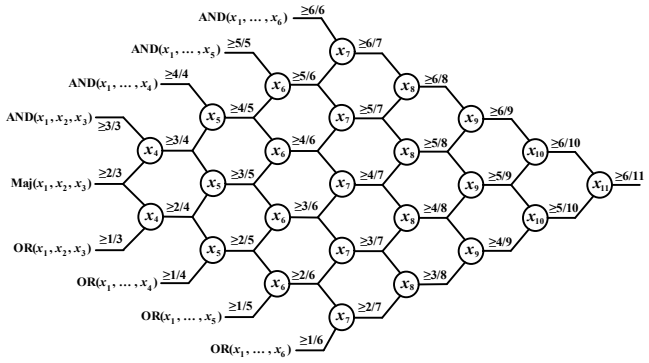

Fig. 6. An 11-input simple majority voter.

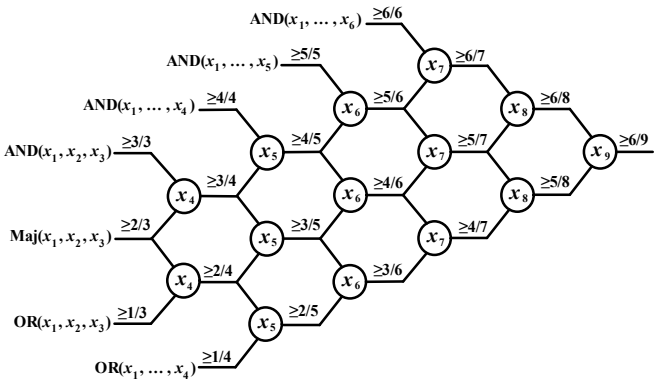

Fig. 7. Super-majority voter with 9 inputs.

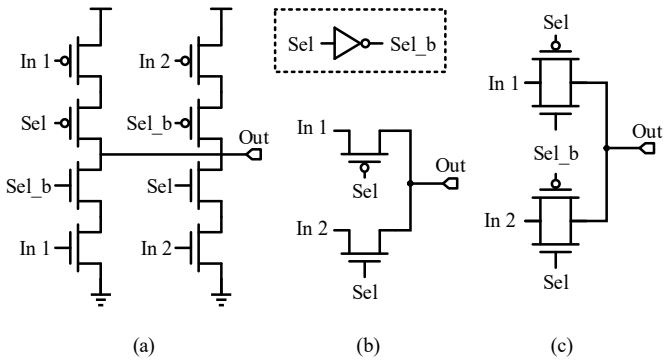

Fig. 8. Three possible structures for a 2-input multiplexer: (a) Ordinary CMOS; (b) Bypass transistor; (c) Transmission gate.

IV. Implementations and Comparisons

In this section, we present preliminary circuit realizations and compare them to alternate designs, after discussing the technology platform used and associated components. More detailed implementation studies are underway.

A. Technology and Components

Considering our recursive design approach, it can be seen that a major portion of the circuitry consists of muxes. Therefore, selecting a suitable mux structure can greatly impact the speed, cost, and power requirements of our final designs. The muxes in our recursive designs are faster than muxes in typical designs, because the selection input to each mux is always ready before the data inputs arrive.

Three possible mux structures, based on ordinary CMOS, bypass transistors, and transmission gates, are given in Fig. 8. These are simulated in our evaluations, using the FinFET 7nm technology file [26].

Our simulation and timing studies have shown that using the bypass transistor structure is best for our recursive designs due to lower preset delay & power consumption, and smaller area overhead. The only shortcomings of the bypass structure are non-full-swing output voltage and low drive strength; problems that can be mitigated by using buffers after every few serial stages of bypass transistors [27]. The last stage of the circuit must also be followed by a buffer or it can be designed using an ordinary CMOS structure to resolve voltage-swing and drive-strength issues.

B. Comparative Analysis

To evaluate our proposed method and compare it with alternative designs, we synthesized five different circuits using our proposed recursive method and three existing alternatives: Two-stage schemes beginning with parallel-counter or parallel-compressor [9], followed by a comparator, and a third scheme with input-capacitances, which uses analog summing instead of digital counting (thus, limiting the design's scaling potential). The simulated circuits are a <2/8 inverse threshold counter (deciding whether there is at most a single 1 among the 8 inputs), an $= 4/8$ weight checker (e.g., for a 4-out-of-8

constant-weight code), $\geq 5/9$ and $\geq 6/11$ simple majority voters, and a $\geq 6/9$ super-majority voter. In our evaluations, we considered room-temperature environment, operation frequency of 100MHz, and supply voltage of 0.7V. FinFET parameters are obtained from [26]. The input set of all circuits is the same and includes all possible combinations of entries in order to have a fair and thorough comparison.

Gate-level realization of the <2/8 inverse threshold circuit based on our design strategy is given in Fig. 9. Note that the NOR networks of Fig. 5 are simply implemented by successive use of NOR results from previous stages. This observation greatly reduces the area and power overhead of the design by eliminating the need for large NOR gates. Multiplexers M1 to M4 are based on the bypass transistor structure (Fig. 8b), while M5 is designed based on the typical CMOS method (Fig. 8a). These choices allow a compact low-power design, while preserving the output's full swing and drive strength.

C. *Assessment of Performance Boost*

Intuitively, one reason for the observed improvements in speed and circuit cost is the use of low-complexity and highly-optimized mux units as the basic building blocks. Additionally, an examination of Fig. 9 reveals that our design is quite regular and has mostly-local connections. To confirm these intuitive advantages, we evaluated the performance boost that can be achieved by using the recursive design technique. To this end, we formulated the improvement achieved by our method in comparison to the best competitor, that is, the parallel-counter/comparator design.

Generally speaking, the exact delay of a circuit depends on many factors, including the number of inputs and the circuit's function. However, Fig. 10 can provide an overall perspective on the benefits of using recursive designs. We see that the delay reduction continually increases with n, although the reduction is sublinear in n (it appears to be even sublogarithmic). The jumps in the blue curve in Fig. 10, which produce jumps in the purple curve, arise because of increases in the number of levels in the parallel-counter for certain values of n. The smoothed green curve provides a good model for predicting delay improvements, before synthesizing actual circuits.

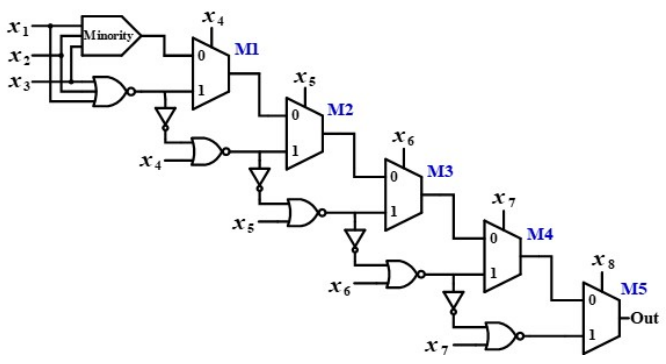

Fig. 9. Design of <2/8 inverse threshold counter using our recursive construction method (a physical realization of the design in Fig. 5).

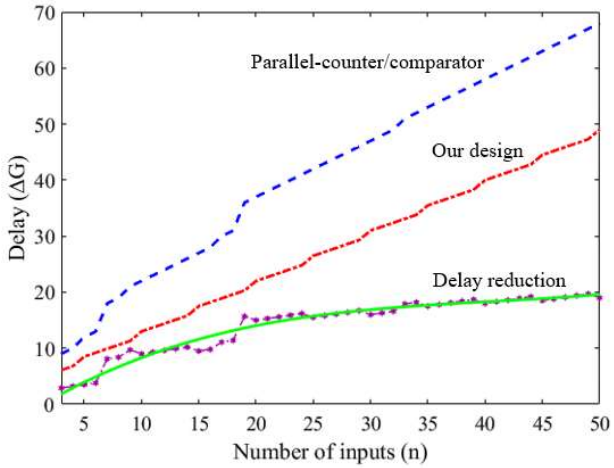

Fig. 10. Delay reduction of our design as a function of the number of inputs.

V. Conclusion

We have presented a recursive method for synthesizing threshold networks, including special cases of majority bit-voters, that offers the benefits of regularity, circuit reuse, ease of analysis, and formal verifiability.

To assess the practical advantages of the proposed method for real implementations, five structures were simulated, with the results compared with those of three alternative design strategies. The simulated examples included a $< 2/8$ recursive threshold counter (determining whether at most 1 of the 8 input bits is 1), an $= 4/8$ equality checker (deciding whether exactly 4 of the 8 input bits are 1s), $\geq 5/9$ and $\geq 6/11$ conventional threshold counters or simple majority voters (determining whether at least 5 of the 9 input bits or 6 of the 11 input bits are 1s), and a $\geq 6/9$ super-majority circuit (determining if there is a 2/3 agreement amongst the 9 inputs).

Our results show that the recursive construction method offers speed, cost, and power-consumption benefits. For the five diverse examples considered, an average reduction of 18% in delay and reduction to less than half in power dissipation and circuit complexity (54% power savings; 51% fewer transistors) were demonstrated, compared with three alternative implementations. When power savings are achieved at the same time as speed improvements, they lead to even greater savings in energy consumption.

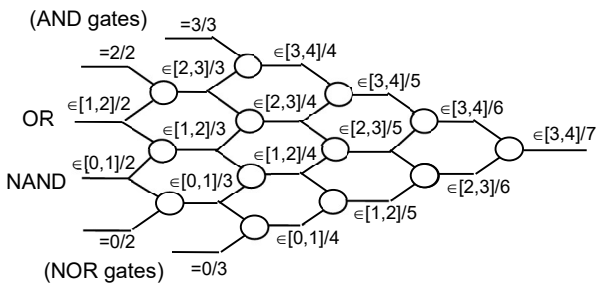

Fig. 11. Simple example of a between-limits threshold circuit.

We know that in the case of bit-voting, circuits based on selection (median-finding) can be beneficial [1]. How do our designs compare with such circuits? Are there alternative design strategies that we have overlooked? Other plans for generalizations and extensions include applying the recursive design method to word-voters, possibly with the ability to signal a lack of majority [28].

Among the many possible generalizations, combination threshold/inverse-threshold circuits can be contemplated as checkers for some weighted codes. An example of such a circuit is depicted in Fig. 11, where the circuit determines whether 3 or 4 of its 7 inputs are 1s (combining the functionalities of $< 5/7$ and $\geq 3/7$ threshold counters). Finally, modular, approximate, and saturating counting networks are worth considering.

ACKNOWLEDGMENT

The author is indebted to Mr. Behzad Davoodnia for several implementation ideas and to Dr. Ghassem Jaberipur & Mr. Sina Bakhtavari for supplying the implementations and comparative evaluations discussed in Section IV.

REFERENCES

[1] B. Parhami, "Voting networks," *IEEE Trans. Reliability*, vol. 40, no. 3, pp. 380-394, Aug. 1991.

[2] K. Nikolic, A. Sadek, and M. Forshaw, "Architectures for reliable computing with unreliable nanodevices," *Proc. 1st IEEE Conf. Nanotechnology*, 2001.

[3] M. Stanisavljević, A. Schmid, and Y. Leblebici, *Reliability of Nanoscale Circuits and Systems: Methodologies and Circuit Architectures*. Springer-Verlag, 2011.

[4] S. Pakin and P. Coles, "The problem with quantum computers," *Scientific American*, 2019.

[5] M. A. Thornton, "Introduction to quantum computation reliability," *Proc. Int'l Test Conf.*, pp. 1-10, 2020.

[6] S. Muroga, *Threshold Logic and Its Applications*, Wiley, 1971.

[7] F. Sabetzadeh, M. H. Moaiyeri, and M. Ahmadinejad, "A majority-based imprecise multiplier for ultra-efficient approximate image multiplication," *IEEE Trans. Circuits and Systems I*, vol. 66, no. 11, pp. 4200-4208, Nov. 2019.

[8] S. Bakhtavari Mamaghani, M. H. Moaiyeri, and G. Jaberipur, "Design of an efficient fully nonvolatile and radiation-hardened majority-based magnetic full adder using FinFET/MTJ," *Microelectronics J.*, vol. 103, p. 104864, Sep. 2020.

[9] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford, 2nd ed., 2010, sec. 12.1 on the design of divide-and-conquer or recursive multipliers.

[10] M. Blaser, "Fast matrix multiplication," *Theory of Computing*, pp. 1-60, 2013.

[11] W. L. Briggs, *The DFT: An Owners' Manual for the Discrete Fourier Transform*, SIAM, 1995.

[12] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Soviet Physics-Doklady* (English translation), vol. 7, no. 7, pp. 595-596, 1963.

[13] V. Strassen, "The asymptotic spectrum of tensors and the exponent of matrix multiplication, *Proc. 27th Symp. Foundations of Computer Science*, pp. 49-54, 1986.

[14] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series." *Mathematics of Computation*, vol. 19, pp. 297-301, 1965.

[15] R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM J. Research & Development*, vol. 6, no. 2, pp. 200-209, 1962.

[16] V. Elamaran, V. S. Balaji, G. Rajkumar, M. Chandrasekar, and H. N. Upadhyay, "Survey on hardware redundancy and CMOS majority voting circuits," *Int'l J. Pure and Applied Mathematics*, vol. 119, no. 15, pp. 1081-1091, 2018.

[17] P. Balasubramanian, D. Maskell, and N. Mastorakis, "Majority and minority voted redundancy scheme for safety-critical applications with error/no-error signaling logic," *Electronics*, vol. 7, no. 11, p. 272, 2018.

[18] E. Abdulhay, V. Elamaran, N. Arunkumar, and V. Venkataraman, "Fault-tolerant medical imaging system with quintuple modular redundancy (QMR) configurations," *J. Ambient Intelligence and Humanized Computing*, pp. 1-13, 2018.

[19] R. Jayalakshmi and R. Amutha, "An optimized high input majority gate design in quantum-dot cellular automata," *Int'l. J. Engineering and Manufacturing Science*, vol. 8, no. 1, pp. 63-75, 2018.

[20] S. Pashazadeh, L. N. Tazehkand, and R. Soltani, "RSS_RAID a novel replicated storage schema for RAID system," in *Data Science: From Research to Application,* Springer, pp. 36-43, 2020.

[21] J. Xu, T. Zhang, and Z. Dong, "On forward error correction with Hamming code for multi-path communications," *Proc. Int'l Conf. Wireless Communication & Signal Processing*, Oct. 2012.

[22] S. Sayil, "A survey of circuit-level soft error mitigation methodologies," *Analog Integrated Circuits and Signal Processing*, vol. 99, no. 1, pp. 63-70, 2019.

[23] E. Dubrova, *Fault-Tolerant Design*. Springer, 2013.

[24] P. Balasubramanian and D. L. Maskell, "A distributed minority and majority voting based redundancy scheme," *Microelectronics Reliability*, vol. 55, nos. 9-10, pp. 1373-1378, Aug. 2015.

[25] D. Harvey and J. Van Der Hoeven, "Integer multiplication in time $O(n \log n)$," *Annals of Mathematics*, vol. 193, no. 2, pp. 563-617, 2021.

[26] L. T. Clark et al., "ASAP7: A 7-nm FinFET predictive process design kit," *Microelectronics J.*, vol. 53, pp. 105-115, Jul. 2016.

[27] J. M. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuits: A Design Perspective*, 2nd ed., Pearson, 2016.

[28] B. Parhami, *Dependable Computing: A Multilevel Approach*, online book available at: https://web.ece.ucsb.edu/~parhami/text_dep_comp.htm