A 2D Array of Multiplexers Can Efficiently Realize a Variety of Counting Networks

Behrooz Parhami
University of California
Santa Barbara, USA
parhami@ucsb.edu

***Abstract*—A counting network accepts *n* bits as inputs and produces *m* output bits, collectively representing a function of the number *w* of 1s among the *n* inputs. Examples include parallel counters (Hamming-weight determiners), parallel compressors, majority bit-voters, comparators, and various threshold functions. We show that a recursive formulation of such networks allows us to efficiently map them onto a 2D reconfigurable array of 2-input multiplexes, offering low-cost realizations and high-throughput, pipelined computation due to the availability of highly-optimized multiplexer circuits and VLSI regularity of the multiplexer array.**

Keywords—Hamming weight, majority, multiplexer array fabric, parallel counter, reconfiguration, threshold function, voter

I. Introduction

Symmetric logical functions of n Boolean variables are those whose values depend only on the number m of the inputs $x_0, x_1, \ldots, x_{n-1}$ that are 1s. Using the Hamming weight notation H, a symmetric logical function $f(x_0, x_1, \ldots, x_{n-1})$ can be expressed as $g(H(x_0, x_1, \ldots, x_{n-1}) = g(m)$. Many functions of interest are symmetric. Examples include parallel counters (Hamming-weight determiners), parallel compressors, majority bit-voters, some comparators, and various threshold functions.

Any arbitrary function of $x_0, x_1, \ldots, x_{n-1}$ can be realized with multiplexers (muxes) using the well-known Shannon expansion, which defines f in terms of the value of one of the variables, say x_1, and residuals of f for $x_1 = 0$ and $x_1 = 1$.

$$f(x_1, x_2, \ldots, x_n) = x_1' f(0, x_2, \ldots, x_n) \vee x_1 f(1, x_2, \ldots, x_n)$$

This is very appealing, except that, in general, direct expansion of the formula above can lead to exponential hardware complexity, given that the exponentially-many branches of the resulting tree do not share any hardware. Even a limited sharing of hardware, say, among polynomially-many branches, will leave the circuit complexity exponential. It is necessary to have an exponential number of shared circuits if the complexity is to drop below exponential.

Fortunately, this exponential level of sharing of circuitry does occur for symmetric functions. Let us unroll the formula above for one more variable, x_2:

$$f(0, x_2, \ldots, x_n) = x_2' f(0, 0, x_3, \ldots, x_n) \vee x_2 f(0, 1, x_3, \ldots, x_n)$$

$$f(1, x_2, \ldots, x_n) = x_2' f(1, 0, x_3, \ldots, x_n) \vee x_2 f(1, 1, x_3, \ldots, x_n)$$

For a symmetric function f, we must have

$$f(0, 1, x_3, \ldots, x_n) = f(1, 0, x_3, \ldots, x_n),$$

which implies the possibility of circuit-sharing.

The reduction of 4 residuals to 3 after two unrollings still does not imply subexponential circuit complexity, as we know from examples such as Strassen matrix multiplication [1] and Karatsuba integer multiplication [2]. However, we note that subsequent unrollings do not increase the number of residual functions to 9, 27, 81, … , but to 5, 7, 9, … , as we will see in examples throughout the paper. This pattern of increase leads to quadratic or $O(n^2)$ complexity.

Of course, one might question whether quadratic complexity is the best we can do. In fact, it isn't, but to do better, we must sacrifice VLSI friendliness in terms of short, regular interconnects among circuit elements. We already use certain quadratic-complexity circuits (e.g., array multipliers [3]), which offer VLSI layout advantages and high throughput via pipelining, especially for moderate, practical values of n which prohibit asymptotic advantages from fully kicking in.

II. Counting Networks

Counting networks are quite varied. In this section, we review the functions, structures, realizations, and applications of three commonly-used classes of counting networks, viz., parallel counters, majority/supermajority voters, and code-error-checkers, in order to provide motivation and a source of examples for the rest of this study.

A. Parallel Counters

Parallel counters are perhaps the best-known counting networks. An n-input parallel counter produces a $\lceil \log_2(n+1) \rceil$-bit binary number at output which represents the count of 1s among the n inputs. Such a counter is known as an $(n; \lceil \log_2(n+1) \rceil)$-counter for short. Parallel counters have been studied extensively in computer arithmetic [4], where they find applications in column-compression and tree multipliers [5]. The designs often use full adders, aka (3;2)-counters, (4;2)-counters, and other kinds of compressors [6]. Figure 1 depicts a (7;3)-counter built of four (3;2) counters or full-adders.

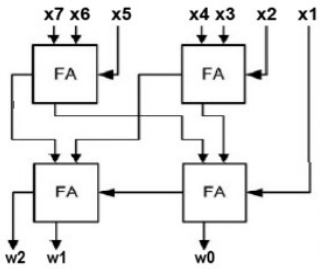

Fig. 1. A (7;3)-counter realized by four (3;2)-counters or full-adders.

The design in Fig. 1 can be best understood as consisting of two 3-input parallel counters (the top 2 boxes), each producing a 2-bit output, and a 2-bit ripple-carry adder (the bottom 2 boxes), which adds the two 2-bit outputs of the top boxes and accommodates the final input, x_1, as its carry-in. This represents a kind of recursive design that realizes a 7-input counter with two $\lfloor 7/2 \rfloor$-input counters and an adder. In our recursive design method, each bit of a parallel counter's output is synthesized as a separate Boolean function, as we will see later.

B. Majority and Supermajority Bit-Voters

Majority voters have been used since the 1950s to build functioning computers from marginal components [7], to protect against errors from radiation-caused bit-flips [8], and to design long-life and safety-critical computers based on replication [9]. The redundancy factor, limited to 3-6 in the past, is moving up, owing to variability & imperfections, and thus high error rates, in nanoelectronics circuits. Redundancy factors of several-dozen have already been proposed and even higher redundancy may be needed with further circuit shrinkage and the accompanying rise in variabilities and uncertainties [10].

In our prior work, we have discussed several methods for designing arbitrarily large voting networks [11] and recursive designs for majority bit-voters [12]. Higher reliability can be achieved by supermajority voters, which can be designed through the same design strategies as simple majority voters. Figure 2 depicts a 6-out-of-9 supermajority voter built of 2-input muxes (drawn as circles) and simple peripheral circuitry, all of which can be replaced by muxes, if desired. The graphical notation in Fig. 2 will be further explained later.

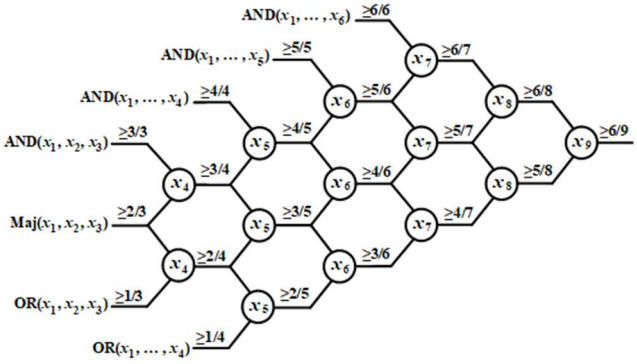

Fig. 2. A recursively-designed 6-out-of-9 supermajority voter.

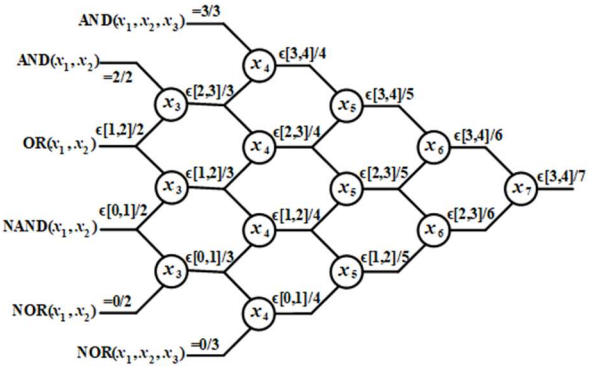

Fig. 3. Error checker for a 7-bit code, with codewords of weight 3 or 4.

C. Error Checkers for Certain Codes

Constant-weight codes [13] and, more generally, codes whose codewords have a defined set of Hamming weights can be designed by using a parallel counter to find the Hamming weight, followed by one or more comparators. However, deriving a code-valid flag directly from the input codeword can lead to substantial savings. For example, consider a 7-bit code whose codewords have weights 3 or 4. The code-valid flag will be raised to 1 whenever the input's Hamming weight is in {3, 4} and it will be lowered to 0 otherwise.

The error checkers just discussed are related to another class of useful circuits: Hamming-weight comparators [14]. Given two binary input vectors, we might be interested to learn whether the two inputs have the same Hamming weight. We can use two parallel counters and a comparator to decide this question, but a single signed parallel counter, which considers one input's bits as positive and the other's as negative will allow us to perform an equality comparison as well as greater-than or less-than comparisons. Such circuits can be viewed as code checkers, with the weight we check for provided by a second bit-vector.

III. A Recursive Design Strategy

In this section, we define our recursive design strategy and show how it can be applied to example circuits in the three application domains defined in Section II.

A. An Introductory Example

Let's begin with an example. To design a 6-out-of-9, or ≥6/9, supermajority voter with inputs x_1-x_9, we rewrite the Shannon expansion about x_9 in either of the following two forms:

$$(\geq 6/9) = x_9'(\geq 6/8) \vee x_9(\geq 5/8)$$

$$(\{6,7,8,9\}/9) = x_9'(\{6,7,8\}/8) \vee x_9(\{5,6,7,8\}/8)$$

Where the number of input variables that remain to be handled is understood, we simplify the latter equation to:

$$\{6,7,8,9\} = x_9'\{6,7,8\} \vee x_9\{5,6,7,8\}$$

We see that a ≥6/9 voter is recursively defined in terms of smaller ≥6/8 and ≥5/8 voters, along with a multiplexer that combines the outputs of the latter. In the second and third of the equations above, acceptable weights of the remaining inputs are listed explicitly. We will discuss in the next subsection where and how the recursion ends, as the unrolling expands the circuit upward, downward, and to the left.

B. General Formulation

A counting network can be characterized by the set of Hamming weights at input that produce an output of 1. For example, a 6-out-of-9 supermajority voter's characteristic set is {6,7,8,9} and the middle output bit of a 7-input parallel counter has the characteristic set {2,3,6,7}. Shannon expansions of the two symmetric functions just defined can be expressed as:

$$\{6,7,8,9\} = x_9'\{6,7,8\} \vee x_9\{5,6,7,8\}$$

$$\{2,3,6,7\} = x_7'\{2,3,6\} \vee x_7\{1,2,5,6\}$$

In the first equation above, we are saying that to have 6, 7, 8, or 9 inputs of 1 among 9 Boolean inputs, we need 6, 7, or 8 inputs of 1 among the remaining 8 inputs if $x_9 = 0$ and 5, 6, 7, or 8

inputs of 1 among those inputs if $x_9 = 1$. Of course, there is nothing magical about the input x_9; we could have expanded around x_2, say. For consistency, we adopt the convention of expanding about input variables in order from highest- to lowest-indexed.

Unrolling the first recurrence leads to the circuit of Fig. 2, but with line labels replaced by the characteristic sets {6,7,8,9}, {6,7,8}, {5,6,7,8}, {6,7}, {5,6,7}, {4,5,6,7}, … , reading from right to left and top to bottom.

As we unroll the recurrences above, we eventually get to terms such as ({5}/5), or {5} in the simplified notation, which defines a 5-input AND gate, or ({1,2,3}/3) = {1,2,3}, realizable by a 3-input OR gate. A ({2,3}/3) = {2,3} element can be realized by a majority gate, if available, or further expanded in a final recursive step as

$$\{2,3\} = x_3'\{2\} \vee x_3\{1,2\},$$

whose circuit realization entails an AND gate for {2}, an OR gate for {1,2}, and a 2-input mux.

Any large AND or OR gate at the upper/lower edge of the circuit can be replaced by a 2-input circuit, one of whose inputs comes from a smaller AND/OR gate in the circuit stage immediately to its left. This simplification does not add to the circuit delay, given that all signals experience one mux delay from one stage to the next.

C. Assessment of the Recursive Design Method

We have done cost-performance comparisons for our recursive designs relative to existing state-of-the art designs in a previous publication [12]. Figure 4 shows delay improvements in gate levels compared with conventional designs using parallel counters and comparators.

We have used detailed circuit simulations to assess implementation parameters for a number of different designs. In all cases, delay, area, and power dissipation improved, often significantly. The results for simple ≥5/9 voter, ≥6/9 supermajority voter, and =4/8 weight checker are shown in Table I. Other example circuits synthesized show delay/power/area improvements in the same ballpark.

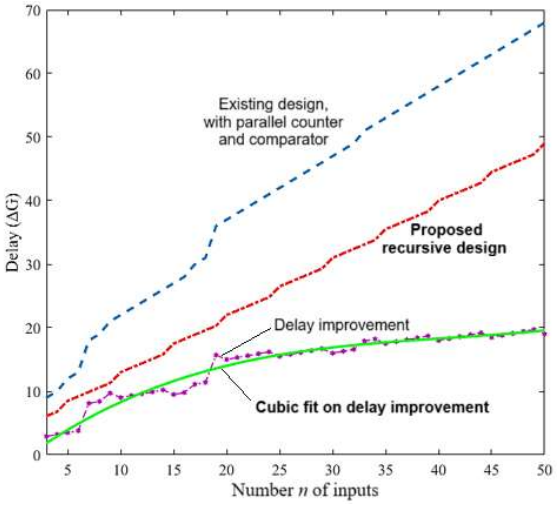

Fig. 4. Delay advantage of recursive design for simple n-voters [12].

TABLE I. SIMULATION RESULTS FOR A FEW EXAMPLE CIRCUITS

Circuit/Design	Delay (ps)	Power (nW)	Transistors
=4/8, Prior → Ours	59.0 → 49.9	180.7 → 94.3	188 → 86
≥5/9, Prior → Ours	52.1 → 44.1	197.6 → 67.5	206 → 94
≥6/9, Prior → Ours	45.4 → 43.0	178.3 → 62.7	196 → 92
Average reduction	12.1 %	59.5 %	53.9 %

Both Fig. 4 and Table I exhibit a trend that we anticipated. The delay reduction is less significant than the savings in power and the number of transistors, given our circuits' linear delay and quadratic cost. We see that the delay reduction becomes less significant as we move to larger values of n, but even for fairly large values of n, we still achieve some speed advantage. The average improvement in the power-delay product is an impressive 64.6%.

Using multiplexers that have been highly optimized with respect to delay, power, and number of transistors plays a big part in the observed improvements.

IV. A RECONFIGURABLE FABRIC

A. Mapping Designs onto a 2D Mux Array

We see from the examples in Figs. 2 & 3 that recursively synthesized counting networks use a 2D array of muxes whose design is universal, except at the edges where recursion ends. Given that AND, OR, and NOT gates can be realized with muxes, these edge elements can also be replaced with muxes, leading to a uniform 2D mux fabric capable of realizing all counting networks of interest discussed so far (Fig. 5). For example, an edge NAND(x_1, x_2, x_3) gate is replaceable by two muxes, as follows

$$\text{NAND}(x_1, x_2, x_3) = x_3' \vee x_3(x_2' \vee x_2x_1'),$$

with x_2 controlling Mux1 and x_3 controlling Mux2. If needed, a third mux can be used to derive x_1'.

We can map any n-input counting network onto an $\sim n/2 \times \sim n/2$ mux array. Reconfiguration is needed only at the edges, with the rest of the array being fixed. Software tools can be developed to provide the settings for the edge multiplexers for any given counting network of interest.

B. An Additional Example

Consider the design of a counting network for verifying that the Hamming weight of a bit-vector of length 8 is in [2,6], representing a closed interval of integers.

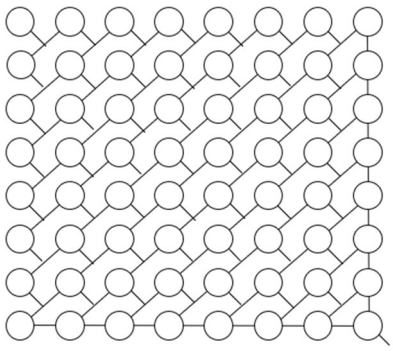

Fig. 5. An 8×8 segment of a larger array of 2-input multiplexers.

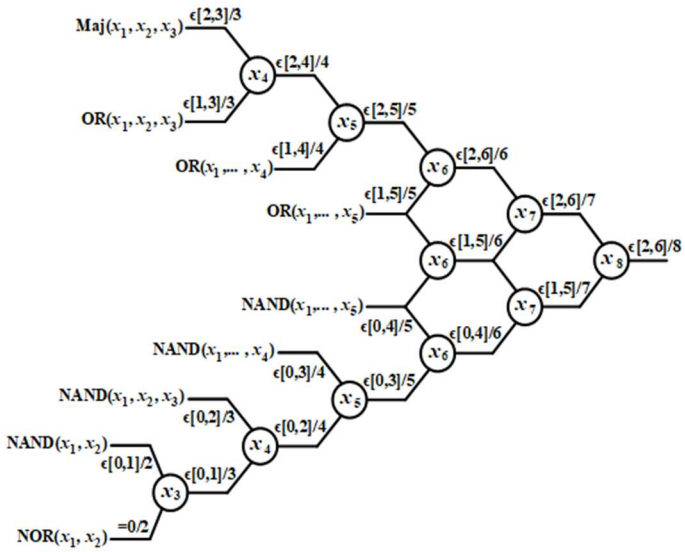

Fig. 6. An irregular and asymmetric between-limits counting network.

This is known as a between-limits counting network, because it checks that the Hamming weight of the input is between a lower and an upper limit; 2 and 6, in this case. The recurrence starts as:

$$\{2,4,5,6\} = x_8' \{2,4,5,6\} \vee x_8 \{1,3,4,5\}$$

The fully-unrolled circuit is shown in Fig. 6, where two features are immediately noticeable: The blank/empty area in the middle and asymmetry of the edge gates at the top and bottom.

We have derived formulas for the delay and cost of nearly all counting networks [12], expressed in terms of the number of muxes and mux levels, thus allowing us to predict accurate figures of merit for custom VLSI realizations. For example, threshold counters or generalized voters are characterized by:

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

$$C(\geq l/n) = (n - l)(l - 1) + \text{lower-order terms} = O(nl)$$

There are a handful of exceptions, where closed-form delay and cost formulas are not yet available. Figure 6 represents one such example. Fortunately, such circuits can also be mapped onto our 2D mux fabric. The OR and NAND gates at the interior edges of the circuit of Fig. 6 can be replaced with muxes using techniques that are identical to those of Subsection IV.A. Any remaining blank area in the middle of the diagram of Fig. 6 will be mapped to inactive muxes generating 0 outputs.

C. Performance Attributes of the Reconfigurable Fabric

The mux array does not introduce any performance overhead compared with the custom designs of the preceding sections. The length of the critical path through an implementation with the reconfigurable array remains the same as that of the custom design. This is because the same number of muxes appear on the critical path in either case. We have not run simulations to assess actual changes in latency or power consumption, leaving these evaluations to future work. Also left to future work are the evaluation of implementation alternatives for muxes.

V. Conclusion

We have presented a recursive method for synthesizing counting networks, including threshold circuits of various kinds and special cases of majority bit-voters, that offers the benefits of regularity, circuit reuse, ease of analysis, and formal verifiability. Our designs yield moderate reductions of around 12% in latency and significant savings of 50-60% in both transistor count and power consumption. That an O(n2)-cost method beats competing methods of $O(n \log n)$ theoretical circuit complexity isn't surprising, given the specific attributes of VLSI technology.

Mapping the unrolled versions of these recursive designs onto a 2D array of multiplexers allows us to achieve additional cost savings through volume production, without suffering from the common inefficiencies associated with reconfigurability. Our designs are successful for the same reasons that FPGAs yields superior cost-performance in many application domains.

Future work can proceed in several directions, from further theoretical studies of the delay and cost of recursively-designed counting networks, to more extensive and more accurate experimental evaluations. An early study of large voting networks [11] concluded that in the case of bit-voting, circuits based on selection (median-finding) can be beneficial. How do our designs compare with such circuits?

References

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

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

[3] T. Sharma, K. G. Sharma, and B. P. Singh, *Low Power and High Performance Array Multiplier: Design and Analysis*, Lambert, 2011.

[4] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford, 2nd ed., 2010.

[5] Z. Wang, G. A. Jullien, and W. C. Miller, "A new design technique for column compression multipliers," *IEEE Trans. Computers*, vol. 44, no. 8, pp. 962-970, Aug. 1995.

[6] Dormido and M. A. Canto, "Synthesis of generalized parallel counters," *IEEE Trans. Computers*, vol. 30, no. 9, pp. 699-703, Sep. 1981.

[7] A. Svoboda, "From mechanical linkages to electronic computers: recollections from Czechoslovakia," In: *A History of Computing in the Twentieth Century*, Academic Press, pp. 579-586, 1980.

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

[9] B. Parhami, *Dependable Computing: A Multilevel Approach*, draft of unpublished book, available online (accessed Mar. 30, 2025) https://web.ece.ucsb.edu/~parhami/text_dep_comp.htm

[10] M. Stanisavljevic, A. Schmid, and Y. Leblebici, *Reliability of Nanoscale Circuits and Systems*, Springer, 2010

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

[12] B. Parhami, "Recursive implementation of voting networks," *Proc. IEEE 14th Annual Computing and Communication Workshop & Conf.*, 2024.

[13] P. R. Ostergard, "Classification of binary constant weight codes," *IEEE Trans. Information Theory*, vol. 56, no. 8, pp. 3779-3785, 2010.

[14] B. Parhami, , "Efficient Hamming weight comparators for binary vectors based on accumulative and up/down parallel counters," *IEEE Trans. Circuits and Systems II*, vol. 56, no. 2, pp. 167-171, Feb. 2009.