Voting Networks

Behrooz Parhami, Senior Member IEEE
University of California, Santa Barbara

Key Words — **Hardware voter, *m*-out-of-*n* voter, Majority circuit, median voter, Plurality voter, Threshold voter.**

Reader Aids —
Purpose: Present and analyze several models
Special math needed for derivations: Elementary probability, switching theory
Special math needed to use results: Same
Results useful to: Designers of fault-tolerant hardware systems

Summary & Conclusions — **Voting is a fundamental operation in the realization of ultrareliable systems that are based on multichannel computations. When data to be voted on are generated at a high rate, the voter must be able to keep up with the processing speed. The actual voting delay might not be critical but the voter throughput must match or exceed the input data rate. Designs of hardware voters are presented that can be easily pipelined to accommodate extremely high data rates. Design strategies for bit-voters and word-voters are described. Examples of resultant designs are given and each design is evaluated with respect to cost and performance. Both ordinary and generalized *m*-out-of-*n* voting, with arbitrary votes assigned to the inputs, are considered. Majority and many other types of threshold voters are simply special cases of generalized *m*-out-of-*n* voters. Median voters can be synthesized by simple variants of our design methods. Using currently available technology, these designs can operate at speeds of many millions of votes per second. For majority bit-voters with small values of *n*, a multiplexer-based design method generally yields the best realizations while for larger values of *n*, designs based on selection networks tend to be most efficient. For word-voters, cost-effective designs based on modified or augmented sorting networks are feasible. In either case, the rich theory developed for the analysis and synthesis of parallel/pipelined sorting networks directly benefits the design process.**

1. INTRODUCTION

Voting is an important operation in the realization of ultrareliable systems that are based on the multi-channel computation paradigm. Voting is required whether the multiple computation channels consist of redundant hardware units, diverse program modules executed on the same basic hardware, identical hardware and software with diverse data, or any other combination of hardware/program/data redundancy and/or diversity. Depending on the data volume and the frequency of voting, hardware or software voting schemes are appropriate. Low-level voting with high frequency necessitates the use of hardware voters whereas high-level voting on the results of fairly complex computations can be performed in software without serious performance degradation or overhead.

The use of voting for obtaining highly reliable data from multiple unreliable versions was first suggested by von Neumann in the mid 1950s [28]. Since then, the concept has been used in fault-tolerant computer systems and has been extended and refined in many ways. Reliability modeling of voting schemes by considering compensating errors [22], handling of imprecise or approximate data [5], combination with standby or active redundancy [17], voting on digital 'signatures' obtained from computation states to reduce the amount of information to be voted on [25], and dynamic modification of vote weights based on *a priori* reliability data [21] constitute some of these extensions and refinements. More recently, generalized voting with unequal vote weights has been proposed for maintaining the reliability and consistency of data stored with replication in distributed computer systems [12]. This has become a very active research area [11, 27].

Replicated systems operating synchronously can achieve extremely high reliabilities if each computation result is voted upon as it is produced. Such frequent voting involves some delay which lengthens the system cycle time and degrades the performance. However, since networks can be pipelined, voter throughput is a function of pipelining period which is unrelated to latency. Thus, 2 performance parameters need to be specified for the voters. When voting is performed to obtain a majority value in order to detect and isolate disagreeing units, delayed detection of disagreement can be acceptable, provided that the delay is not so large as to make the probability of fault accumulation intolerably high. In this situation, voter throughput, rather than its delay, is of primary importance. Similarly, when the voter is part of a pipelined special-purpose computation scheme, voter delay is much less important than its throughput. In either case a cellular pipelined voter consisting of multiple stages, with each stage having a fairly small delay, is desirable.

This paper considers the design of such cellular pipelined voters whose general structure is depicted in figure 1. The n hardware computation units produce a continuous stream of values that must be voted upon in order to pass a dependable value to the next phase of the computation or to detect and disable the unit(s) producing non-conforming output(s). The D stages are designed to have small, roughly equal latencies in order to maximize the throughput. The design of m-out-of-n or threshold bit-voters and word-voters is discussed in sections 2 and 3. Section 4 covers some variations and extensions as well as directions for future work. The appendix provides proofs of all theorems and lemmas.

Hardware voter designs in the literature are essentially 'bit-voters' that compute a majority function on n input bits [14, 24]. Combined bit voting and disagreement detection has also been discussed [6]. Hardware voting on words and higher-level data objects has traditionally been handled by using independent parallel bit-voters or feeding the data sequentially through a single unit. Such independent bit-voting yields results that are optimistic, particularly when correlated errors are likely (see

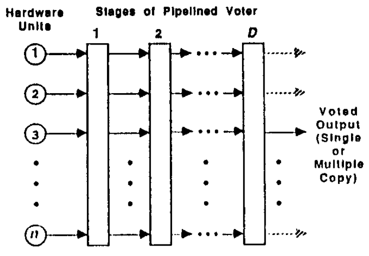
Hardware Units
Stages of Pipelined Voter
1
2
D
1
2
3
n
Voted Output (Single or Multiple Copy)

Figure 1. General Structure of a Pipelined Cellular Voter.

section 3.1). Hardware voters in actual systems include *3*-way voters in MIT's FTMP design [13], CMU's C.vmp system [23], and August Systems' industrial control computers [30]. The JPL STAR computer [4] used a special 2-out-of-5 voter that was symmetrically connected to 3 active modules and 2 standby spares for its critical Test and Repair Processor (aerospace systems used hardware voters even before STAR). To the best of my knowledge, no hardware voter with adjustable or variable vote weights has been described in the literature. Similarly, approximate or other context-dependent voting algorithms have not been implemented in hardware. The effect of pipelining between voting stages on the performance of triple-modular redundant systems has been recently discussed [9] but this is different from my use of parallelism and pipelining within each voting circuit.

2. BIT-VOTING NETWORKS

Define an m-out-of-n hardware bit-voter as a circuit with n bit-inputs x_i, $i=1,2,\ldots,n$, and 1 bit-output y, such that $y=\mathbf{1}$ iff at least m of the n input bits have the value **1**. This definition is asymmetric with respect to **0** and **1** values in that an output of **0** does not imply and is not implied by the presence of at least m **0**'s at the inputs. Such asymmetry should not be worrisome and is useful in the design of fail-safe systems where one of the two output values is considered 'safe' (more on this below). Simple majority bit-voting corresponds to the special case of $m = \text{gilb}(n/2)+1 = \text{liub}((n+1)/2)$. This special case has **0/1** symmetry for odd values of n. Section 2.2 generalizes this definition to weighted bit-voters.

2.1. *m-out-of-n Bit-Voters*

An m-out-of-n hardware bit-voter can be constructed as a 2-level AND-OR (equivalently NAND-NAND) logic circuit with $g \equiv n!/[m!(n-m)!]$ m-input AND gates and 1 g-input OR gate ($g+1$ NAND gates) for small values of the parameters m and n. Somewhat less obvious is a *2*-level OR-AND realization requiring $g' \equiv n![(m-1)!\ (n-m+1)!]$ $(n-m+1)$-input OR gates and 1 g'-input AND gate. In the first realization, all possible subsets of m inputs are ANDed together and the voter output is **1** if at least one of the AND results is **1**. In the second realization, all possible subsets of $n-m+1$ inputs are ORed together and the voter output is **0** if at least one of the OR results is **0**.

Theorem 1: The *2*-level AND-OR realization of an m-out-of-n bit-voter is 'simpler' than its 2-level OR-AND realization iff $m>(n+1)/2$. The complexities are equal for odd values of n if $m=(n+1)/2$. □

Example 1: Consider a 3-out-of-5 bit-voter which is the same as a simple *5*-input majority bit-voter. Both the AND-OR and the OR-AND designs require 11 gates (10 *3*-input gates at level-1 and 1 *10*-input gate at level-2, as shown in figure 2), with a total of 40 gate inputs. □

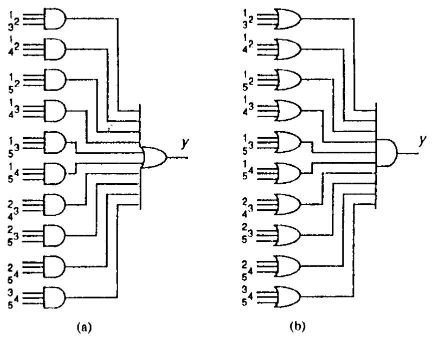

Figure 2. Two Realizations for a 2-Level, 3-out-of-5 Bit-Voter (the digit i on a line represents the input x_i)

Example 2: Consider a 2-out-of-5 bit-voter. Such a voter is suitable for use in a system with fail-safe modules such that the probability of producing an incorrect **1** output is very small compared to the probability of an incorrect **0** output. The AND-OR design requires 11 gates (10 *2*-input ANDs and 1 *10*-input OR, as shown in figure 3a), with a total of 30 gate inputs. The OR-AND design requires 6 gates (5 *4*-input ORs and 1 *5*-input AND, as shown in figure 3b), with a total of 25 gate inputs. □

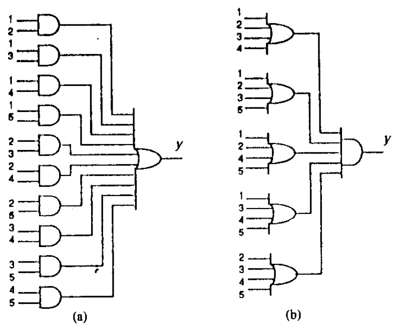

Figure 3. Two Realizations for a 2-Level, 2-out-of-5 Bit-Voter (the digit i on a line represents the input x_i)

For large values of n, it is impractical to realize bit voters as 2-level logic circuits. Assuming the use of f-input gates and ignoring the possibility of gate sharing for now, the total number of gates in the 2-phase AND-OR and OR-AND realizations changes from $g+1$ and $g'+1$ to:

$$G = g\ \text{liub}((m-1)/(f-1)) + \text{liub}((g-1)/(f-1))$$

$$G' = g'\text{liub}((n-m)/(f-1)) + \text{liub}((g'-1)/(f-1)),$$

respectively. These expressions are obtained by simply substituting, for each gate, an equivalent tree of f-input gates of the same type. The number of gate inputs in the two designs increases from $g(m+1)$ and $g'(n-m+2)$ to approximately Gf and $G'f$. Finally, the delays increase from 2 logic levels to $\text{liub}(\log_f m) + \text{liub}(\log_f g)$ and $\text{liub}(\log_f(n-m+1)) + \text{liub}(\log_f g')$ logic levels, respectively. It is quite straightforward to make these multilevel designs pipelined by including 2 to 4 logic-gate levels per pipeline stage [29].

With gate sharing, an exact general analysis for the gate count becomes difficult. However bounds for the number of gates can be obtained that are close to actual values in most practical cases. The following approximate analysis establishes one such bound for the AND-OR design assuming unlimited fan-out (the OR-AND case is similar). We can use $n!/[f!(n-f)!]$ f-input gates in logic level-1 to generate all possible logical products of f variables. Each m-variable product can then be computed by a tree of $\text{liub}((\text{liub}(m/f)-1)/(f-1))$ additional gates. Since there are g such products which must then be ORed together, the total number of required gates is:

$$G'' = n!/[f!\ (n-f)!] + g\cdot\text{liub}((\text{liub}(m/f)-1)/(f-1)) + \text{liub}((g-1)/(f-1))$$

Example 3: Consider the design of a 4-out-of-8 bit-voter using 2-input gates ($n=8$, $m=4, f=2$). We have $g=70$, $G=70\times3+69=279$, and $G''=28+70\times1+69=167$. Thus for this example, hardware savings of at least 40% is achieved with gate sharing. □

Example 4: Reconsider the design of the 3-out-of-5 bit-voter of example 1 using 2-input gates ($n=5$, $m=3, f=2$). Then $g=10$, $G=10\times2+9=29$, $G''=10+10\times1+9=29$. Thus, no hardware savings is indicated by the upper bound G''. However, the design shown in figure 4 suggests that only 4 of the 10 level-1 gates are necessary and a 20% savings is possible. Generally, the upper bound G'' is not tight when n is small or m is close to n. □

Example 5: Reconsider the design of the 2-out-of-5 bit-voter of example 2 using 2-input gates ($n=5$, $m=2, f=2$). Then $g=10$, $G=G'=G''=19$. The value of G'' in this example correctly indicates that no gate sharing is possible for the AND-OR design. Figure 5 shows that the OR-AND design might be preferable due to the possibility of gate sharing. □

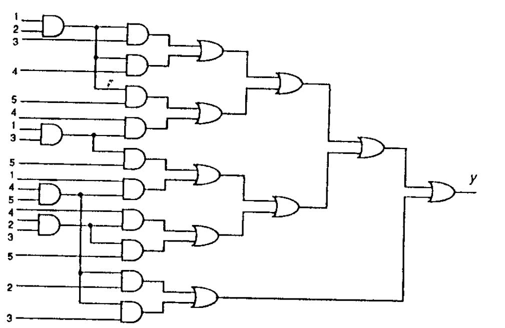

Figure 4. Limited Fan-In Realization for a 3-out-of-5 Bit-Voter with $f=2$ and Gate Sharing (the digit i on a line represent the input x_i)

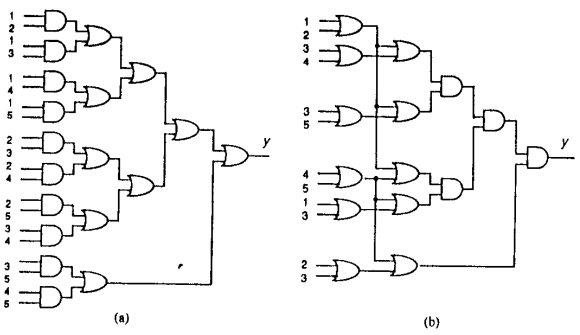

Figure 5. Limited Fan-In Realizations for the Circuits of Figure 3 with $f=2$

2.2. *Weighted Bit-Voters*

If each of the n bit-inputs x_i, $i=1,2,\ldots, n$, has an associated vote v_i such that the output **1** is produced iff the sum of votes associated with the inputs having the value **1** is no less than a threshold t, then a generalized t-out-of-Σv_i voter is obtained. Except where noted otherwise, the remainder of section 2 considers only such weighted bit-voters; designs for non-weighted voters can be obtained as their special case.

This section 2.2 discusses gate-level direct logic realizations for weighted bit-voters. A multiplexer-based approach and an 'arithmetic' approach are discussed in sections 2.3 and 2.4. Designs based on decomposition (divide and conquer) and selection are treated in sections 2.5 and 2.6.

When the input votes are small integers, it is tempting simply to replicate (fan-out) each input by its corresponding weight and use an m-out-of-n bit voter with $m=t$ and $n=\Sigma v_i$.

Example 6: Consider 3 inputs with the associated votes 2, 2, 1 and the voting threshold of 3. Then, a 3-out-of-5 bit-voter can be used, with 2 of its input lines connected to each of the inputs carrying a vote of 2. The input votes are not optimally assigned in that equal votes (a simple 2-out-of-3 voter) would accomplish the same thing. □

More — on the outside.

Example 7: Consider 6 inputs with the associated votes 2, 2, 2, 1, 1, 1 and the voting threshold of 5. Then, a 5-out-of-9 bit-voter can be used, with 2 of its input lines connected to each of the inputs carrying a vote of 2. Unlike example 6, here the assignment of votes is optimal in that at least two different input vote values are needed to achieve the desired effect. □

However, this input-replication method seldom results in an optimal voting network, even when the votes are assigned in an optimal way as in example 7. Intuitively, the reason is that the asymmetry of votes must actually simplify the voting process whereas the input-replication approach complicates it. The direct logic approach simply enumerates all minimal subsets of **1** inputs that should result in a **1** output. Reconsider example 7 with this approach.

Example 8: With 6 inputs having the associated votes 2, 2, 2, 1, 1, 1 and the voting threshold of 5, the minimal input subsets correspond to the product terms in the following sum-of-products expression for the output function —

$$123+124+125+126+134+135+136+234+235+236$$

$$+1456+2456+3456$$

where each digit i represents the input x_i. The required 13 AND gates and 1 OR gate with a total of 55 input lines is considerably less than that required for a 5-out-of-9 simple bit-voter. With a fan-in limit of $f=4$, a *3*-level *17*-gate circuit realizes the desired function. □

2.3. *Design with Multiplexers*

A variation of the direct logic approach discussed in section 2.2 involves the use of multiplexers. Interest in this approach is triggered by the availability of multiplexers as off-the-shelf components and by their universality. The design strategy is to start with the highest-vote inputs and take them as control inputs to a multiplexer and then repeat the process for each multiplexer input function until we reach simple residual functions that can be efficiently implemented (the extreme case being *1*-gate functions such as AND and OR). The reason for proceeding in descending order of votes is that when high-vote inputs are fixed at **1** or **0**, relatively simpler residual logic functions of the remaining input variables tend to result.

The multiplexer-based approach is essentially a special case of the decomposition (divide and conquer) strategy (section 2.5) whereby an n-input t-out-of-Σv_i bit-voter is built from a multiplexer and two or more smaller voters. In particular if a *2*-input multiplexer is used in decomposition stage-1, then 2 $(n-1)$-input bit-voters, 1 t-out-of-$(\Sigma v_i - v_{\max})$ and the other $(t-v_{\max})$-out-of-$(\Sigma v_i - v_{\max})$, are required where $v_{\max}$ is the largest input vote. In the special case of equal votes, an m-out-of-n bit-voter is constructed from an m-out-of-$(n-1)$ voter, an $(m-1)$-out-of-$(n-1)$ voter, and a *2*-input multiplexer.

Even though the worst-case complexity of such multiplexer-based voter designs appears to be exponential in n, pruning and circuit-sharing is possible in practice. Thus the complexity can be acceptable for particular values of n and m with specific vote assignments. Numerous worked-out examples suggest that the size of the multiplexers used can be just as important as the ordering of the decomposition variables in obtaining efficient designs. Clearly there is ample room for further research in this area.

Example 9: With 6 inputs having the associated votes 2, 2, 2, 1, 1, 1 and the voting threshold of 5, we can use 2-input multiplexers in the following way. Take x_1 as the first control variable. The residual functions corresponding to $x_1=\mathbf{0}$ and $x_1=\mathbf{1}$ are:

$$x_2x_3x_4 + x_2x_3x_5 + x_2x_3x_6 + x_2x_4x_5x_6 + x_3x_4x_5x_6$$

$$x_2x_3 + x_2x_4 + x_2x_5 + x_2x_6 + x_3x_4 + x_3x_5 + x_3x_6 + x_4x_5x_6,$$

respectively. Continuing in this manner, we arrive at the expansion:

$$[x'_1[x'_2(x_3x_4x_5x_6) + x_2h] + x_1[x'_2h + x_2(x_3+x_4+x_5+x_6)]]$$

$h \equiv x_3x_4+x_3x_5+x_3x_6+x_4x_5x_6$ has the multiplexer realization:

$$h = [x'_3(x_4x_5x_6) + x_3(x_4+x_5+x_6)]$$

Thus a total of 4 multiplexers corresponding to the square brackets in the above expressions and 4 gates corresponding to the parentheses in the above expressions are needed. The required circuit is shown in figure 6a. The last 2 levels of multiplexers in figure 6a can be replaced by 1 *4*-input multiplexer. With *8*-input multiplexers, the relevant expression becomes:

$$[x'_1x'_2x_3h_1+x'_1x_2x'_3h_1+x'_1x_2x_3h_2+x_1x'_2x'_3h_1$$

$$+ x_1x'_2x_3h_2+x_1x_2x'_3h_2+x_1x_2x_3]$$

$$h_1 \equiv (x_4x_5x_6),\ h_2 \equiv (x_4+x_5+x_6).$$

Thus, 1 *8*-input multiplexer and 2 gates are sufficient in this case, as depicted in figure 6b. □

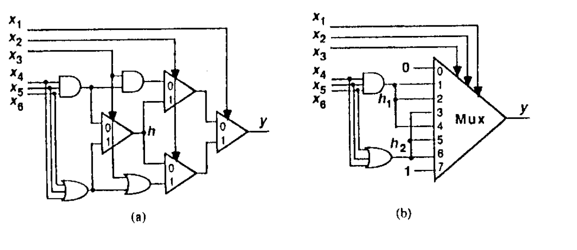

Figure 6. Multiplexer-Based Realizations for the Voter of Example 9

2.4. *Arithmetic-Based Design*

In the 'arithmetic' approach for implementing a threshold voter, the sign of the arithmetic expression $-t+\Sigma(x_i v_i)$ is computed. Since a non-negative result implies an output of **1**, the output is the complement of the computed sign. The products $x_i v_i$ can be computed by AND gates and then added by standard carry-save technique to yield the final result. If the v_i's are constants, the hardware realization can be optimized in each case by compressing the constant **0**'s in the binary numbers to be added. Standard arithmetic pipelining techniques [29] apply to the resulting design.

Example 10: With 6 inputs having fixed associated votes of 2, 2, 2, 1, 1, 1 and the voting threshold of 5, the arithmetic expression to be evaluated is:

$$-5+2x_1+2x_2+2x_3+x_4+x_5+x_6.$$

The reduction steps in our multiple-operand addition using full-adder and half-adder cells are shown in figure 7 (1011 is the *4*-bit 2's-complement representation of the constant -5). Each vertical box encloses the inputs to a full or half adder and the associated horizontal box encloses the outputs. Referring to figure 7, a *2*-bit adder can be used to reduce the remaining y and z bits in the middle 2 columns, with the carry-out of this adder providing the final output. The leftmost **1** can be ignored since it causes only a complementation that cancels the complementation needed for obtaining the resultant output from the sign bit. This implementation requires 4 *1*-bit full adders and 2 half adders in a *4*-level circuit; actually, further optimization is possible. □

Instead of using full adders and half adders, one can use larger building blocks known as parallel counters [8, 26] and parallel compressors [10], which convert a group of input bits to fewer output bits while maintaining the arithmetic value being represented. In fact, when input votes are all equal, a parallel counter followed by a comparator or a single 'accumulative parallel counter' can directly compute the output. Cost and delay analyses for such reductions can be found in [8, 10, 26] and related work in the field of computer arithmetic.

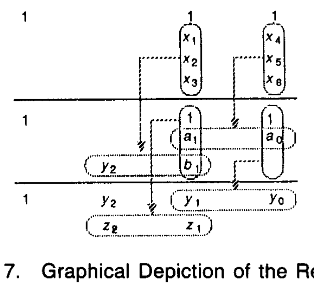

Figure 7. Graphical Depiction of the Reduction Steps in an Arithmetic-Based Realization of the Voter of Example 10.

2.5. *Designs Based on Decomposition*

Hierarchical decomposition (divide and conquer) can be used to facilitate the design of threshold bit-voters, although in general it does not guarantee an optimal network. There are two ways to proceed with the decomposition approach:

1. Pick a combining network and then design the partitioning scheme.
2. Pick a partitioning scheme and then design the needed combining network.

The multiplexer-based design technique of section 2.3 is an example of the first approch. This section 2.5 presents some ideas related to the second approach.

The basic idea is to —

- Divide the inputs into disjoint subsets
- Enumerate the various combinations in which different subsets can contribute votes in such a way that the voting threshold is matched or exceeded
- Provide smaller bit-voters to realize these contributions
- Design a logic network for combining the results.

If necessary, decomposition can be applied to the smaller bit-voters as well. This proceeds recursively until voters with known designs are obtained. As an extreme case, the decomposition continues until *n*-out-of-*n* or 1-out-of-*n* bit-voters, corresponding to AND and OR gates respectively, are reached.

Example 11: Consider the design of a 3-out-of-5 bit-voter (with equal input votes). Divide the inputs into subsets $S_1 \equiv \{x_1,x_2,x_3\}$ and $S_2 \equiv \{x_4,x_5\}$. The combinations that match or exceed the threshold of 3 are:

$$\text{3-of-3 in } S_1 + (\text{2-of-3 in } S_1 \text{ and 1-of-2 in } S_2)$$
$$+ (\text{1-of-3 in } S_1 \text{ and 2-of-2 in } S_2).$$

This formulation yields the logic expression —

$$x_1x_2x_3 + (x_1x_2+x_2x_3+x_3x_1)(x_4+x_5)$$
$$+ (x_1+x_2+x_3)x_4x_5$$

which directly translates into a *4*-level circuit using 10 gates with 25 input lines. The number of gates and gate inputs can be reduced to 9 and 23, respectively, and the number of gate levels reduced to 3 if we implement the 2-out-of-3 voter in OR-AND form and then combine the two cascaded AND gates that result. □

When all input votes are equal, decompositions with almost equal-size subsets tend to work best. When the votes are unequal, the inputs can be divided into subsets each containing equal-vote inputs. Further decomposition then proceeds as before. This tends to simplify the design, although again there is no guarantee that it yields the best network.

More — on the outside.

Example 12: With 6 inputs having the associated votes of 2, 2, 2, 1, 1, 1, and the voting threshold of 5, the 2 input subsets are $S_1 \equiv \{x_1,x_2,x_2\}$ and $S_2 \equiv \{x_4,x_5,x_6\}$. The combinations that match or exceed the threshold of 3 are:

$$3\text{-of-}3 \text{ in } S_1 + (2\text{-of-}3 \text{ in } S_1 \text{ and } 1\text{-of-}3 \text{ in } S_2)$$

$$+\ (1\text{-of-}3 \text{ in } S_1 \text{ and } 3\text{-of-}3 \text{ in } S_2).$$

This formulation yields the logic expression:

$$x_1x_2x_3 + (x_1x_2+x_2x_3+x_3x_1)\ (x_4+x_5+x_6)$$

$$+\ (x_1+x_2+x_3)x_4x_5x_6$$

which directly translates into a *4*-level circuit using 10 gates with 27 input lines (maximum fan-in of 4). Expanding the above expression results in the logic expression of example 8. Again, using an OR-AND realization of the 2-out-of-3 voter and merging cascaded ANDs, can reduce the complexity to 9 gates with 26 inputs and the delay to 3 gate levels. □

2.6. *Design with Selection Networks*

The design of an *m*-out-of-*n* bit-voter is equivalent to selecting the m^{th} largest bit value from among *n* input bits. The design of sorting and selection networks built from 2-sorter (comparator) cells has received much attention [7, 15]. For our problem, we can either use an *n*-sorter (descending order) and look at its m^{th} output or take advantage of the generally simpler selection networks. Knuth [15] defines three types of selection networks that accomplish the following, when provided with *n* input values:

1. Select the *m* largest values and move them to *m* output lines in no particular order.
2. Select the m^{th} largest value and move it to a specified output line.
3. Select the *m* largest values and move them to *m* output lines in sorted order.

Each network requires at least as many 2-sorters as the preceding one. Denote the number of 2-sorter or comparator cells by $U(m,n)$, $V(m,n)$, $W(m,n)$ for type-1, type-2, type-3 selectors above. Then —

$$U(m,n) \leq V(m,n) \leq W(m,n).$$

Because we are dealing with bits, a *2*-sorter simply consists of a pair of *2*-input gates: An OR gate to produce the larger and an AND gate to produce the smaller of the 2 input values. Table 1 shows the number of 2-sorters required for the three types of selectors by listing the triples U,V,W for small values of *m* and *n* [15].

Clearly type-3 selectors do more than is required here. Type-2 selectors do exactly what we want. However, for most practical values of *m* and *n*, particularly where *m* is neither too small nor too close to *n*, a type-1 selector augmented by an AND or OR circuit (that indicates whether all of the *m* largest values are **1**'s or whether the $n-m+1$ smallest values are not all **0**'s) is faster and more economical (see table 1).

TABLE 1
Minimum-Cost $(m,\ n)$-Selectors of Three Types
[For each *m* and *n*, the triple U,V,W denoting the costs of the three selector types is listed]

n	$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 5$	$m = 6$
2	1, 1, 1	0, 1, 1				
3	2, 2, 2	2, 3, 3	0, 2, 3			
4	3, 3, 3	4, 5, 5	3, 5, 5	0, 3, 5		
5	4, 4, 4	6, 7, 7	6, 7, 8	4, 7, 9	0, 4, 9	
6	5, 5, 5	8, 9, 9	8, 10, 10	8, 10, 12	5, 9, 12	0, 5, 12

Example 13: Consider the design of a 4-out-of-8 bit-voter. The required type-1 selection network that selects the 4 largest bit values and moves them to the upper half of the output lines is given in figure 8, where each heavy vertical line represents a comparator that moves the larger of the 2 input values to the upper line and the smaller value to the lower line. The verification that figure 8 indeed represents a type-1 (4,8)-selection network is simple if we note that the network consists of 2 *4*-sorters followed by 4 comparators each of which compares the j^{th} largest output of the upper *4*-sorter to the j^{th} smallest output of the lower unit. The selector of figure 8 requires 14 comparators or 28 2-input gates with 4 gate levels of delay. A *4*-input AND gate connected to the upper 4 outputs completes the circuit. A 5-out-of-8 majority voter results if we connect an OR gate to the lower 4 output lines in the circuit of figure 8. □

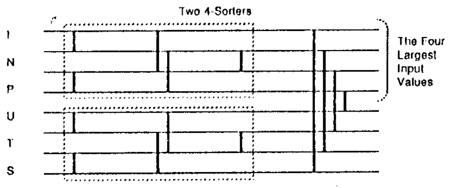

Figure 8. Network of Comparators to Select the 4 Largest of 8 Values. One Additional 4-Input Logic Gate Can Convert This Network into a 4-out-of 8 or 5-out-of-8 Bit-Voter

2.7. *Comparison of Various Designs*

Because of the many parametes involved, a general comparison of the design strategies discussed in section 2 is quite difficult. Thus, the designs are compared only for simple majority bit-voters ($m = \text{gilb}(n/2)+1$, equal input votes). Figure 9 shows the cost of simple majority voters designed based on 2-level logic expressions ('gate-level'), the arithmetic-based approach, selection networks, and *2*-input multiplexer decomposition, assuming maximum allowable gate fan-in of 4. Figure 9

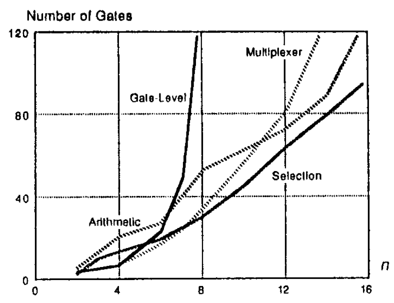

Figure 9. Cost of Simple Majority Bit-Voters with Different Designs as a Function of Input Size n

indicates that the gate-level or multiplexer-based approach is best for small values of n whereas selection networks offer the most economical solution for larger values of n. The crossover points change if the assumptions (such as gate fan-in and fan-out limitations) are altered.

Although the arithmetic-based approach appears to be inferior to the one based on selection, it can become quite competitive if the resulting arithmetic circuit is fully optimized to eliminate all redundant elements (only a small amount of optimization has been done to obtain the data for figure 9). Figure 9 relates only to simple majority voters. The strength of the arithmetic-based approach shows up in the case of unequal input votes where the selection-based approach does not directly apply (eg, example 10). Similarly, the pure multiplexer-based approach is not particularly efficient for simple majority voting but can yield good designs for specific values of the parameters in weighted t-out-of-Σv_i voting. A combination of the approaches might provide the best solution in a particular case.

Since the objective is to use our designs with pipelining, the differences in latencies (number of gate levels) are not important as far as throughput is concerned. However, the number of gate levels does affect the cost due to the requirement for latches between pipeline stages. A general analysis is impractical because the number of logic signals going from one pipeline stage to the next cannot be expressed as a simple function of the relevant parameters. Also, these analyses have conveniently ignored the cost associated with interconnections. For these reasons, it is advisable to proceed with and check out several designs for any set of application parameters before a final selection is made. Clearly, more work remains to be done in quantifying the tradeoffs, identifying promising combinations of these approaches, and possibly devising new design methods.

3. WORD-VOTING NETWORKS

An m-out-of-n word-voter is a circuit with n word-inputs x_i, $i = 1,2,..,n$, and 1 word-output y, where either at least m of the inputs have the value y or else a separate 'lack of quorum' signal is raised. An alternative (pursued in the following pages) is to produce an output vote w along with a value y having maximal vote, so that comparing w to m yields the desired quorum information and indicates the validity of y as the output. When two or more values have equal votes, any one of them is an acceptable output. The 'lack of quorum' signal could be envisaged for bit-voters as well where it would be needed only if $m \geq n/2+1$. Again, a simple majority word-voter corresponds to the special case of $m = \text{gilb}(n/2)+1 = \text{liub}((n+1)/2)$.

We proceed directly to the general case where a vote v_i is associated with the input x_i and obtain simple unweighted word-voters as a special case of the general design. Thus the word-voter receives $\langle x_i, v_i \rangle$ pairs as inputs and produces the single $\langle y, w \rangle$ pair as output. Another interesting special case is when the input votes v_i are in $\{0,1\}$, with $v_i = 0$ corresponding to a 'disabled' computation unit (perhaps due to a previously detected disagreement). Although we conside 1 output pair, the technique can be extended to q output pairs in order to provide tolerance to internal voter faults through multi-channel computation. In the remainder of section 3, the voter is assumed to be perfect.

3.1. *Using Bit-Voters*

A straightforward approach for t-out-of-Σv_i voting on a set of n k-bit words is to use independent t-out-of-Σv_i bit-voting for each of the k bit positions. This can be done sequentially using 1 bit-voter or in parallel by k voters. There are two difficulties with this approach.

1. For $m \leq n/2$, the independent bit-voters can produce incorrect outputs.

Example 14: Consider 2-out-of-4 voting with 2-bit inputs **01**, **10**, **10**, **11**. Independent 2-out-of-4 bit-voting for the 2 bit positions produces **11** as the output whereas the correct output is **10**. The problem does not result from the **0/1** asymmetry of bit-voters as defined here. No matter how bit-voters are defined, the one dealing with the right-hand bit position in this example has no basis for selecting **0** or **1** as its output. □

2. The independent bit-voters can indicate a quorum when no quorum exists.

Example 15: Consider 3-out-of-4 voting with 2-bit inputs **00**, **10**, **10**, **11**. The 2 independent 3-out-of-4 bit-voters produce the output **10** even though **10** does not have the required quorum of 3. □

It should be clear from the above discussion and examples that implementing a k-bit word-voter by k independent bit-voters is unacceptable for $m \leq n/2$ but works for $m > n/2$ provided that a quorum actually does exist. Thus, in this case, an erroneous result is obtained for $m > n/2$ iff a quorum does not exist. This event has the probability:

$$E = \text{binf}(m-1;\ (1-p)^k, n)$$

More — on the outside.

Notation

p	probability that any given data bit is in error (assuming s-independence of errors in different bit positions)
binf(.;..)	binomial Cdf (see "Information for Readers & Authors" at the rear of this issue).

Note that $(1-p)^k$ is the correctness probability for a k-bit word and that an error occurs if fewer than m of the n words are correct.

Given the bit-error probability p and a maximum tolerable voting-error probability $E_{\max}$, the above equation can be used to find the maximum allowable word length $k_{\max}$ for each m and n, if independent bit-voting is to work.

Example 16: Consider 3-out-of-4 voting $(m=3,\ n=4)$. The equation for E reduces to:

$$E = [1-(1-p)^k]^2\ [1+2(1-p)^k+3(1-p)^{2k}]$$

Since the second term on the r.h.s. is less than 6, the above error probability is less than $E_{\max}$ if —

$$[1-(1-p)^k]^2 \le E_{\max}/6$$

$$k < \frac{\log\ (1-\sqrt{E_{\max}/6})}{\log(1-p)}.$$

As a numerical example, for $E_{\max}=10^{-7}$ and $p=10^{-5}$, the sufficient condition is $k < 13$. Thus, problems may arise even for fairly short word lengths. □

A remedy for the problem illustrated by examples 15 and 16 (but not for that of example 14) in the case of bit-sequential voting is to equip the bit-voter with n disagreement flip-flops that are set to **1** at the beginning of each word-voting cycle and are reset to **0** whenever the corresponding input differs from the bit-voter output. Then the voter result is valid iff after considering all the k bit positions, at least m flip-flops remain set. This condition can be verified by the bit-voter itself as step $(k+1)$ of the bit-sequential voting process.

3.2. *Three-Phase Word-Voters*

Assume that k-bit input data values and their associated b-bit votes are provided in parallel on $k+b$ lines per input. The requirements for digit-serial computation, which is more practical due the need for fewer interconnections and less complex circuits, is discussed in section 3.4. Cellular voting networks can be constructed on the basis of the following algorithm in which each phase corresponds to a subnetwork of the voting network.

Algorithm 1: 3-Phase Voting

1. Sort the input pairs $\langle x_i, v_i \rangle$ according to the data fields x_i.
2. Combine all pairs with equal data values by summing their votes.
3. From among the remaining pairs, select one with the largest vote. □

Example 17: A 5-voter receiving the input pairs $\langle 5,1\rangle$, $\langle 4,2\rangle$, $\langle 5,3\rangle$, $\langle 7,2\rangle$, $\langle 4,1\rangle$ should produce the output pair $\langle 5,4\rangle$ since the sum of votes for the data value $x=5$ (viz, $v=1+3=4$) is larger than those of the input values 4 and 7. Figure 10 shows the 3 subnetworks and how they cooperate to obtain the voting result for the input data of this example. □

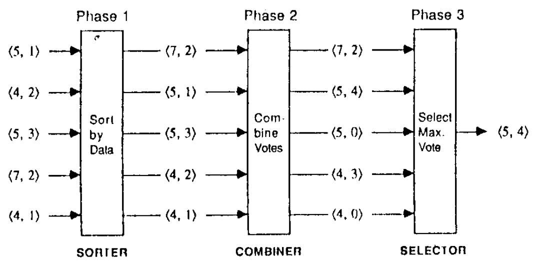

Figure 10. The Stucture of a Three-Phase Cellular Voting Network

The design of sorting networks has been studied extensively. Knuth [15] provides an excellent exposition of two decades of work in this area starting from its inception in 1954. Two other books [2,7] contain references to more recent work on sorting networks. Sorting networks can be constructed from simple 2-input, 2-output comparator cells according to several schemes. Each comparator either passes the 2 inputs directly to the 2 outputs or switches the input values to make their order consistent with the desired output order. The only modification required for our application is to add logic to the cells for passing or switching the votes in tandem with the data values (figure 11).

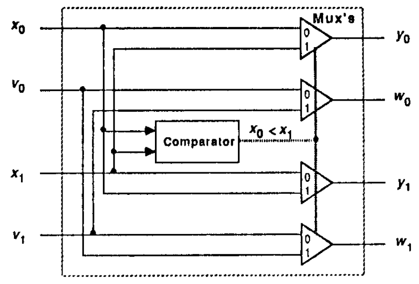

Figure 11. Internal Structure of an Augmented 2-Sorter Cell

Even though the design of optimal sorting networks is still an open problem (viz, truly optimal rather than asymptotically optimal), for small values of n likely to be of practical interest in voting schemes, optimal or near-optimal n-sorters are known.

Figure 12 shows several cost-optimal sorting networks. The interested reader can verify that these are in fact sorting networks by using the **0-1** principle: If an n-input network correctly sorts all 2^n possible combinations of **0**'s and **1**'s, then it is a sorting network. The 5-input network in figure 12 can be understood by viewing it as composed from a 2-sorter and a 3-sorter, followed by a (2,3)-merger as outlined by the dotted boxes. The structures of the 7-sorter and 9-sorter of figure 12 are more complicated. Table 2 presents data on the fastest and lowest-cost sorting networks known for small values of n [15].

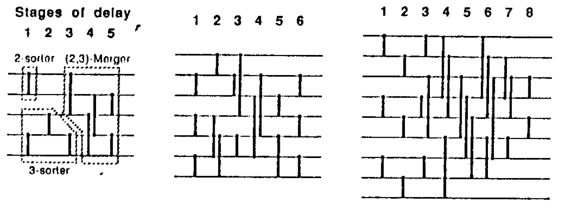

Figure 12. Cost-Optimal Sorting Networks for n = 5, 7, and 9

The combining subnetwork required for phase-2 of the algorithm essentially performs a partitioned semigroup computation. The operation of addition must performed in partitions that are delimited by pairs of unequal values. This can be done by a circuit in which n overlapping binary trees are embedded such that the tree rooted on line i accumulates data from line i and all subsequent lines that carry equal values (all the way to line n in the worst case). Just as sorting networks can be synthesized from 2-sorters, the n-combiner is constructed from 2-combiners. Figure 13 shows the combiner cell needed and figure 14 depicts the designs for 5-, 7-, and 9-combiners. It is easy to show that the cell count $C_c(n)$ and the delay $D_c(n)$ for an n-combiner are (lg denotes $\log_2$):

$$C_c(n) = (n-1)+(n-2)+(n-4)+\ldots+(n-2^{\text{gilb}(\lg n)})$$

$$= n\ (\text{gilb}(\lg n)+1) - (2^{\text{gilb}(\lg n)+1}-1)$$

$$= n\ \text{gilb}(\lg n) - (2^{\text{gilb}(\lg n)+1}-n-1)$$

$$< n \lg n$$

$$D_c(n) = 2\ \text{liub}(\lg n) - 1$$

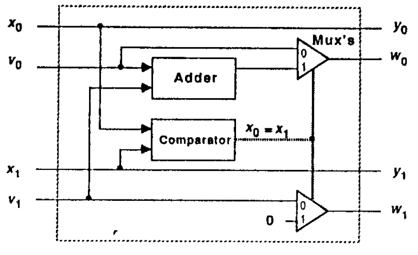

Figure 13. Internal Structure of a 2-Combiner Cell

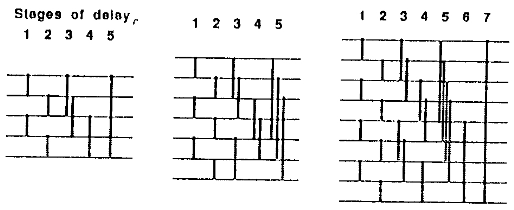

Figure 14. Combining Networks for n =5, 7, 9

TABLE 2
Minimal-Cost and Minimal-Delay n-Sorters
[In general, minimum cost and minimum delay cannot be attained simultaneously]

n	$C_s^{min}(n)$	$D_s^{min}(n)$
2	1	1
3	3	3
4	5	3
5	9	5
6	12	5
7	16	6
8	19	6
9	25	7
10	29	7
11	35	8
12	39	8
13	46	9
14	51	9
15	56	9
16	60	9

The max-selector subnetwork for phase-3 is essentially an liub$(n/2)$-leaf binary tree of 2-selector cells whose design is depicted in figure 15. The cell count and delay of an n-input max-selector network are:

$$C_m(n) = n-1$$

$$D_m(n) = \text{liub}(\lg n)$$

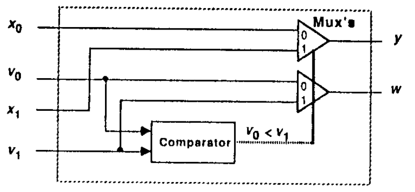

Figure 15. Internal Structure of a 2-Selector Cell

Figure 16 shows the designs of 5-, 7-, and 9-input max-selectors. If the max-selector subnetwork is replaced by a sorting

More — on the outside.

subnetwork that sorts by vote values, then the total votes of all distinct input values is available at the output. This might be a useful feature if one wants to synthesize large voting networks from smaller ones. Thus, for example, 2 n-voters and 1 $2m$-voter ($m<n$) can be used to synthesize a $2n$-voter by connecting the first m outputs of each n-voter to the inputs of the $2m$-voter. The resulting $2n$-voter is not perfect, ie, it can produce an incorrect output with a very small probability (that can be easily quantified). With this modification, the example shown in figure 10 produces, respectively from top to bottom, the output pairs ⟨5,4⟩, ⟨4,3⟩, ⟨7,2⟩, ⟨5,0⟩, ⟨4,0⟩.

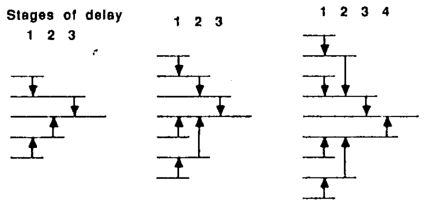

Figure 16. Max-Selector Networks for n = 5, 7, and 9

A more useful practical modification is to produce the maximal-vote result on a set of q output lines. This allows the voter to feed directly the next stage of a multiple-channel computation. Also if voter inputs are stored in the computation units for a period of time equal to voting delay or if original values on input lines are carried along with the transformed values in the 3-phase voter, a set of comparisons performed in a single extra stage can identify the diagreeing unit(s). The above can be accomplished by adding a phase-4 consisting of $q-1$ 'distributor' cells. The number of circuit levels added is 1 with umlimited fan-out and liub$(\log_f q)$ if fan-out is limited to at most f. It is also possible to combine phases 3 & 4 by using the networks in figure 14 in which each cell places the input with the larger vote on both outputs. The resulting circuit has the same delay as the 4-phase circuit but is more complex.

The complexity and delay of the 3-phase voting network is easily obtained by adding the parameters of 3 phases with suitable weights to account for the unequal costs and delays of the 3 cell types. Figure 17 shows the structure a complete 5-voter, with the intermediate values also shown for the input data of example 17. These 3-phase voting networks are fairly complex. However, if standard pipelining techniques are used, the throughput of such a cellular voter is dictated by the largest cell delay and is independent of its size. Although from a practical point of view it does not make sense to talk about extremely large voting networks, it is interesting to note that asymptotically the complexity (both in cell count and in delay) of 3-phase voting networks is dominated by the respective parameters of the sorting phase since sorting cannot be done with fewer than $O(n \lg n)$ cells or with less than $O(\lg n)$ cell delays [1]. Practical sorting networks actually have delays proportional to $(\lg n)^2$ [7,15].

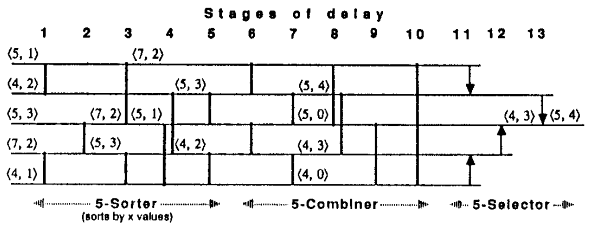

Figure 17. Complete Structure of a 3-Stage 5-Input Voter

3.3. *2-Phase Word-Voters*

Having obtained a design for cellular voting networks, it is reasonable to ask whether the design can be simplified. This section shows that the first 2 phases of the 3-phase voter can be combined by merging the functions of 2-sorter and 2-combiner cells. This reduces the total number of cells but increases the cell complexity, leading to an obvious tradeoff due to the fact that the number of cells in phases 1 & 2 are not equal. To show this, we need the following lemma.

Lemma 1: In any sorting network, two input values x_j and x_k ending up on adjacent output lines y_i and y_{i+1} must have been compared to each other in some stage. □

Since after sorting, equal input values end up on adjacent lines, by lemma 1 they must have been compared within the sorting subnetwork. This allows us to integrate the combining function with the sorting function, thus effectively merging phases 1 & 2 of the 3-phase voter. Figure 18 shows a 2-sorter/combiner cell which acts as a 2-sorter when the input values are unequal and as a 2-combiner when they are equal. Since 2-sorter/combiner cells are building blocks for synthesizing voting networks, we call them '2-voters'.

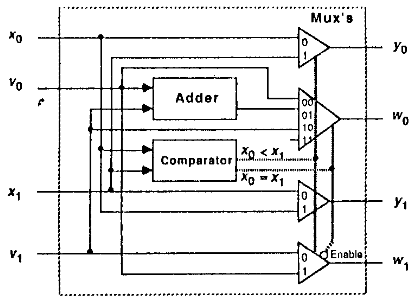

Figure 18. Internal Structure of a 2-Voter (2-Sorter/Combiner) Cell

Theorem 2: If in any sorting network, the 2-sorter cells are replaced by 2-voters (cells that act as 2-sorters when the input values are unequal and as 2-combiners when they are equal), the outputs are identical to those obtained from a 2-phase sorter/combiner network. □

The complexity and delay of the *2*-stage voting network is easily obtained by adding the respective parameters of the 2 phases with suitable weights to account for the unequal costs and delays of the 2 cell-types. Figure 19 shows a complete *2*-phase *5*-voter constructed from *2*-voter cells along with example values from the data of Example 17. Comparing this to the *5*-voter of figure 17, we see that 8 combiner cells and their 5 levels of delay have been eliminated, but each of the 9 cells in phase 1 is rendered slightly more complex and slower. Some quantitative comparisons are presented in section 3.5.

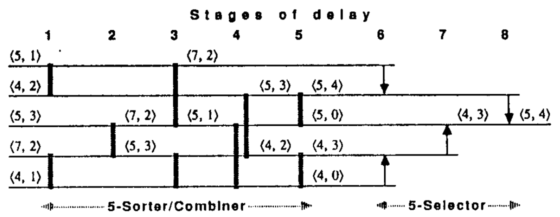

Figure 19. Complete Structure of a Two-Stage, 5-Input Voter

A reasonable question at this point is whether the merging of phases can be carried one step further to obtain a *1*-phase voting network consisting of 1 cell-type. The following result shows that no network consisting of 1 type of *2*-input, *2*-output cells can correctly compute the voted output as defined in this paper.

Theorem 3: No voting network can be constructed from 1 cell-type with 2 data/vote inputs and 2 data/vote outputs. □

One might think that if data and vote formats were identical, a 2-voter could be made to behave as a 2-max-selector by simply flipping the data/vote pairs at input and output. However, even with this property, 1 cell type would still be inadequate due to the spurious "combining" that might occur in the case of tie votes.

3.4. *Digit-Serial Operation*

Sections 3.2 & 3.3 assumed that a cell receives all of the digits of the data values x and the votes v in parallel. This might be practical for small values of n and short word lengths but not otherwise. Thus we are motivated to consider the design of digit-serial voting networks in order to reduce the number of interconnections required. Although digit-serial operation increases the number of clock cycles per operand feed, some of the loss is regained because the simpler digit-oriented cells can be driven at a much higher clock rate.

The augmented *2*-sorter cell of figure 11 can be easily modified for digit-serial operation. The serial x inputs must be supplied starting from the most significant digit and the output of the digit comparator used to set a flip-flop that controls the multiplexers. As for the vote inputs, they cannot be handled in an on-line, serial fashion concurrently with the data digits because initially the cell cannot decide how to route the 2 vote values until the outcome of the data comparison is known. One way around this problem is to supply the vote values in parallel. This approach is feasible only if the votes are small integers. A more practical option is to provide the votes digit-serially following the last data digits. In this case, input lines for data and vote values can be shared. In fact as far as the sorting phase is concerned, data and vote digits can be treated identically and thus they need not be distinguishable.

The *2*-combiner cell of figure 13 can be similarly modified for digit-serial operation. Again, the vote outputs can be produced only after the comparison is completed. With parallel vote inputs, the design is straightforward. The vote value can also follow the corresponding data value in a digit-serial format provided that the least-significant digit appears first. However, this format is incompatible with the requirement of the selection phase where the relative magnitudes of vote values must be compared. The use of a redundant number representation system [3,19] that can support serial addition starting from the most significant end of the vote values is a possible solution.

The only remaining problem is that votes must precede data values in phase 3 whereas they follow the data in phase 2. If each *2*-selector cell is provided with 2 shift registers of length k, then correct operation can be effected as follows. During the first k clock cycles, the new data are shifted in while the old data are shifted out. Then for the next b clock cycles, the votes are accepted, compared, and output in the proper order. This is followed by the output of the associated data from, and acceptance of the next data into, the shift registers. Thus, the operational delay of the voting network is increased by k clock cycles while its throughput is unaffected. One could of course require that vote values precede data values. This would allow simpler selector cells but would require the inclusion of shift registers in the sorter/combiner cells which are more numerous than selector cells, thus leading to higher cost.

3.5. *Comparisons of Costs and Speeds*

We now proceed to quantify the relative advantages of *3*-phase versus *2*-phase voter designs. Although I do not claim that either design is optimal, the following informal argument suggests that *2*-phase cellular voters are very close to optimal in terms of the number of cells. It can be shown [20] that the vote-tallying function performed by the sort-combine phase is asymptotically as complex as sorting and that weighted voting cannot be accomplished without explicitly tallying all the votes, even though only the maximal vote is of interest. My *2*-phase voter design only contains $n-1$ simple *2*-selector cells beyond those required for the sort/combine part and the argument in the proof of theorem 3 suggests that these might be necessary in any design. Thus, simplification, if possible, occurs in the internal design rather than the multiplicity or interconnection structure of cells.

In what follows, we ignore the complexity and delay associated with the selection phase which is common to *3*-phase and *2*-phase designs. Thus the figures provide an indication of the differences rather than the ratios of complexities and delays.

More — on the outside.

This is sufficient for judging when the *3*-phase or *2*-phase design should be preferred over the other. We take the complexity and delay of a *2*-sorter as a unit-measure and denote by γ' and γ'' (respectively δ' and δ'') the relative complexities (respectively delays) of the *2*-combiner and the *2*-voter cells. $C_s(n)$ and $D_s(n)$ are the number of *2*-sorter cells and the number of cell levels needed in an n-sorter. The complexities C' and C'' and delays D' and D'' of *3*-phase and *2*-phase designs are:

$$C'(n) = C_s(n) + \gamma' C_c(n)$$

$$= C_s(n) + \gamma' (n \operatorname{gilb}(\lg n) - 2^{\operatorname{gilb}(\lg n)+1} + n + 1)$$

$$C''(n) = \gamma'' C_s(n)$$

$$C'(n) - C''(n) = \gamma'(n \operatorname{gilb}(\lg n) - 2^{\operatorname{gilb}(\lg n)+1} + n + 1)$$

$$- (\gamma'' - 1) C_s(n)$$

$$D'(n) = D_s(n) + \delta' D_c(n)$$

$$= D_s(n) + \delta'(2 \operatorname{liub}(\lg n) - 1)$$

$$D''(n) = \delta'' D_s(n)$$

$$D'(n) - D''(n) = \delta'(2 \operatorname{liub}(\lg n) - 1) - (\delta'' - 1)D_s(n)$$

Actual values for γ', γ'', δ', δ'' are technology- and implementation-dependent. However, by inspecting figures 11, 13, 18 and keeping in mind that these designs can be varied depending on speed and cost constraints as well as parallel or serial operation, the following are safe bounds on the values of these parameters:

$$1 \le \gamma' < \gamma'' < 3$$

$$1 \le \delta' \le \delta'' < 2.$$

The lower values of γ' and γ'' correspond to the case where input values are presented sequentially and the addition and comparison hardware are shared. The lower values of δ' and δ'' correspond to the high-speed parallel realizations shown in figures 13 and 18.

Table 3 contains the values of $C'(n) - C''(n)$ and $D'(n) - D''(n)$ as functions of γ', γ'', δ', δ''. Figure 20 depicts γ'' as a function of γ' for values of n such that $C'(n) - C''(n) = 0$. In the region above each of the lines in figure 20, the *3*-phase design is better for the corresponding value of n while the *2*-phase design is advantageous below the lines. Thus given efficient realizations for the cells of figures 11, 13, 18, one can easily determine γ' and γ'' using appropriate indicators of complexity for the corresponding technology and then use figure 20 to determine which of the two designs is less expensive. As n increases, the lines in figure 20 generally move downward (eventually tending to $\gamma'' = 1$ for very large n), with some local fluctuations resulting from the discrete nature of the problem.

TABLE 3
Differences of Complexity and Delay Parameters for 3-Phase (C', D') Versus 2-Phase (C'', D'') n-Voters.

n	$C'(n) - C''(n)$	$D'(n) - D''(n)$
2	$1+\gamma'-\gamma''$	$1+\delta'-\delta''$
3	$3(1+\gamma'-\gamma'')$	$3(1+\delta'-\delta'')$
4	$5(1+\gamma'-\gamma'')$	$3(1+\delta'-\delta'')$
5	$9(1+\gamma'-\gamma'')-\gamma'$	$5(1+\delta'-\delta'')$
6	$12(1+\gamma'-\gamma'')-\gamma'$	$5(1+\delta'-\delta'')$
7	$16(1+\gamma'-\gamma'')-2\gamma'$	$6(1+\delta'-\delta'')-\delta'$
8	$19(1+\gamma'-\gamma'')-2\gamma'$	$6(1+\delta'-\delta'')-\delta'$
9	$25(1+\gamma'-\gamma'')-4\gamma'$	$7(1+\delta'-\delta'')$
10	$29(1+\gamma'-\gamma'')-4\gamma'$	$7(1+\delta'-\delta'')$
11	$35(1+\gamma'-\gamma'')-6\gamma'$	$8(1+\delta'-\delta'')-\delta'$
12	$39(1+\gamma'-\gamma'')-6\gamma'$	$8(1+\delta'-\delta'')-\delta'$
13	$46(1+\gamma'-\gamma'')-9\gamma'$	$9(1+\delta'-\delta'')-2\delta'$
14	$51(1+\gamma'-\gamma'')-10\gamma'$	$9(1+\delta'-\delta'')-2\delta'$
15	$56(1+\gamma'-\gamma'')-11\gamma'$	$9(1+\delta'-\delta'')-2\delta'$
16	$60(1+\gamma'-\gamma'')-11\gamma'$	$9(1+\delta'-\delta'')-2\delta'$

From table 3 the value of $D'(n) - D''(n)$ is positive for $n \le 16$ provided that:

$$\delta'' < 16/9 + 7(\delta' - 1)/9.$$

Even if δ' is very close to 1, it is highly unlikely that $\delta'' \ge 16/9 \approx 1.78$ for a particular implementation. Thus it is safe to conclude that the *2*-phase design always has the edge in terms of delay. However if $\delta' < \delta''$, the *3*-phase design supports a higher throughput, given the fact that a *2*-selector cell is simpler than a *2*-sorter cell, a *2*-combiner cell, or a *2*-voter cell.

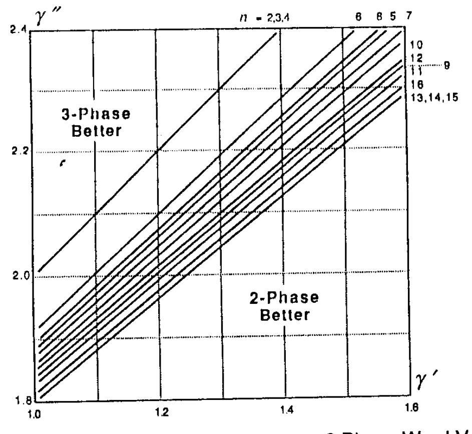

Figure 20. Complexity of 3-Phase Versus 2-Phase Word-Voters

4. VARIATIONS & EXTENSIONS

This section examines issues related to the design of optimal voting networks in certain special practical cases, motivates the extension of the ideas presented here to other types of voting schemes, and offers several concluding observations.

Having derived a general technique for designing weighted word-voting networks, one reasonably looks for simpler designs in special cases where the full power of the general voting scheme is not needed. One such case is simple majority voting where all inputs have equal votes and the output should yield the value carried by a majority of inputs. The only simplification that seems possible in this case is that vote tallying can stop once a majority is obtained. Assuming parallel inputs, this translates into a savings of at most 1 bit in the b-bit adder embedded in the 2-combiner or 2-voter cells. For serial inputs, the savings are even less important. With m-out-of-n voting, given that $m \ll n$ and any of the values having at least m votes is acceptable as output, the savings resulting from tallying votes only up to m can be important. At the other extreme, when m is close to n, one may choose to tally 'negative' rather than 'positive' votes associated with each value. Consensus or n-out-of-n voting is of course trivial.

Approximate voting presents a whole set of new problems. One may of course modify the design such that approximate equality (to within ϵ) is used as the criterion for the combining operation. However, the output is then sensitive to the order in which comparisons are performed [16]. There seems to be no efficient hardware-implementable voting algorithm for this case. However if the median-voting strategy is used (the voted result is the median of input values), the complexity becomes manageable. Median voting with equal input votes essentially requires a selection network which is less complex than a sorting network [15]. Median voting with unequal input votes is a new concept that needs further investigation. Consensus or n-out-of-n voting with approximate data only requires verifying that the largest and smallest input values are within ϵ of each other. This can be accomplished quite easily in a bit-serial design.

I argued in section 3.1 that word-voting cannot always be effected through independent bit-voting. This is because individual bits in a word are usually not computed separately and thus are likely to contain correlated errors. However, the same cannot be said about more complex data structures. A structure that is incorrect overall, can contain many correct components and this can be used to advantage in constructing a voting procedure that increases the likelihood of obtaining a correct result. Thus, it might make sense to use independent word-voting to compute the voting result for a vector of words or for a record. This approach takes care of data errors within the structures but is incapable of handling structural errors such as incorrect pointers or missing elements. Clearly, special structure-voting schemes have to be devised with specific instances for vector-voting, record-voting, set-voting, tree-voting, and the like.

Example 18: Suppose that we want to perform 3-out-of-3 or consensus voting on the sets $S_1 \equiv \{1, 2, 3, 4\}$, $S_2 \equiv \{2, 3, 4, 5\}$, $S_3 \equiv \{3, 4, 5\}$. Clearly if the sets are considered as single entities, no consensus exists since the three sets are all different. However, consensus does exist that 3 and 4 are members of the set and that 6, say, is not. Thus, the set $\{3, 4\}$ might be a reasonable voting result, depending on the application. In practice, it is unlikely that such complex voting decisions occur at a very high rate and thus they can be delegated to software routines. □

Finally, I have assumed that input votes are given, and have dealt only with hardware design techniques, given the input vote values. In practice, one must also deal with procedures for assigning votes to the inputs in an 'optimal' manner. Optimality in this context can be defined in various ways depending on reliability and safety concerns. The basis for vote assignment can be histories associated with the various computation channels (eg, past disagreements with the voted result), static correctness probabilities (given or derived), or dynamic 'dependability' designations carried along with the data [18]. I have likewise not discussed criteria for selecting a particular voting scheme (eg, 3-out-of-5 *vs* 4-out-of-5) and have considered the voting scheme as given. This too must be dealt with through suitable probabilistic analyses and is being investigated.

It is interesting to observe some relationships between voting networks and other classes of networks that have been studied. One may observe that if all the input votes v_i are equal, the vote-tallying function, performed by the sort-combine phase, sorts the input data. On the other hand if all the data values x_i are equal, then the voting network becomes a multiple-operand addition network. If in the special case of equal input votes we additionally have $x_i \in \{\mathbf{0},\mathbf{1}\}$, then the voting network behaves like a parallel counter [8, 26] in the sense that its output indicates the multiplicity of the **1**'s or **0**'s at the input. With $v_i \in \{0,1\}$, we have what may be called a 'partitioned parallel counter' in the sense that counting occurs within partitions having identical data tags. Thus voting networks can be viewed as generalized sorters, multiple-operand adders, and parallel counters.

APPENDIX-Proofs

Proof of Theorem 1: The complexity of a logic circuit can be judged in various ways. Two common measures are the logic-gate count and gate-input count, both of which yield the same result here. As for logic-gate count, there are $g+1$ gates in the AND-OR design versus $g'+1$ gates in the OR-AND design. Since $g/g' = (n-m+1)/m$, we have $g<g'$ iff $m>(n+1)/2$ and $g=g'$ iff $m=(n+1)/2$. If the number of gate inputs is considered, there are $g(m+1)$ for the AND-OR design *vs* $g'(n-m+2)$ for the OR-AND alternative. Again since $g/g' = (n-m+1)/m$, the AND-OR design is simpler than the OR-AND version iff $(n-m+1)\ (m+1) < m(n-m+2)$ which is equivalent to the desired result. *Q.E.D.*

Proof of Lemma 1: Consider the output values y_i and y_{i+1} in a network that sorts in ascending order and receives n different

More — on the outside.

values as inputs. With these assumptions, we can write $y_1 < \ldots < y_i < y_{i+1} < \ldots < y_n$. Let $y_i = x_j$ and $y_{i+1} = x_k$. If the values of y_i and y_{i+1} (alternatively, x_j and x_k) are never compared, then exchanging the values of the j^{th} and k^{th} input lines leads to the exchange of y_i and y_{i+1} at the output side. This is true because the result of comparing $x_k = y_{i+1}$ to any $x_l \neq x_j$ is the same as that of comparing $x_j = y_i$ to x_l. Thus x_k ends up on y_i and x_j on y_{i+1}. Clearly, the network does not sort properly in this case and therefore it cannot be a sorting network.

Q.E.D.

Proof of Theorem 2: (By contradiction)—Without loss of generality, consider a network that sorts in ascending order. Clearly, the proposed modified network sorts properly and does some combining, so the only potential problem is that combining might be incomplete in that for some input pattern, there exists a pair of output lines y_i and y_{i+l} carrying $\langle x,v\rangle$ and $\langle x,v'\rangle$, respectively, with $v' \neq \mathbf{0}$ and all of the lines $y_{i+1},\ldots,y_{i+l-1}$ (if any) carrying $\langle x,0\rangle$; ie, there exist equal data values whose corresponding votes have not been combined. Let S and S' be the set of input lines that have contributed to the formation of $\langle x,v\rangle$ and $\langle x,v'\rangle$, respectively. Since combining takes place any time a pair of equal values are compared, no member of S has been compared to any member of S'. If we change the inputs in S to $x+\epsilon$, such that $x+\epsilon$ is less than the next larger input value after x, at the output we have $y_i = \langle x+\epsilon,v\rangle$ and $y_{i+l} = \langle x,v'\rangle$.Clearly, the sorting outcome is flawed and the network could not have been a valid sorting network.

Q.E.D.

Proof of Theorem 3: Suppose that such a network exists. Without loss of generality, assume that the voted output is produced on the uppermost output line y_1. Consider the last (rightmost) cell that touches line 1. Assume that this cell is non-redundant in that its removal invalidates the voting network. Thus there exists an input pattern for which the cell under consideration combines or interchanges its 2 input pairs. Assume that combining takes place at this cell for some input pattern, with the pairs $\langle x,v_1\rangle$ and $\langle x,v_2\rangle$ combined to yield $\langle x,v_1+v_2\rangle$ on line 1. It is easy to adjust the input weights, if necessary, such that there exists some other output line carrying a different value x' with the vote v_1+v_2-1, viz, 1 less than the maximum vote. Now multiply each input vote by 3. This does not change the functioning of the cells or the final output, except that the output votes are tripled as well. The above-mentioned 2 lines now have votes $3v_1+3v_2$ and $3v_1+3v_2-3$, respectively. Now reduce 1 of the input votes contributing to $3v_1+3v_2$ by 2 and increase 1 of the input votes contributing to $3v_1+3v_2-3$ by 2. This does not change the functioning of the network except that the upper output line now carries the vote $3v_1+3v_2-2$ and the other line has a vote of $3v_1+3v_2-1$. Thus the upper output line does not always carry the value with the highest vote as assumed. Similar reasoning takes care of the case where the last cell touching line 1 interchanges its inputs.

Q.E.D.

REFERENCES

[1] M. Ajtai, J. Komlos, E. Szemeredi, "An $O(n \log n)$ sorting network", *Proc. ACM Symp. Theory of Computing*, 1983, pp 1-9.

[2] S. G. Akl, *Parallel Sorting Algorithms*, 1985, chap 2, pp 17-39; Academic Press.

[3] A. Avizienis, "Signed-digit number representation for fast parallel arithmetic", *IRE Trans. Electronic Computers*, vol EC-10, 1961, pp 389-400.

[4] A. Avizienis, et al, "The STAR (Self-Testing-And-Repairing) computer: An investigation of the theory and practice of fault-tolerant computer design", *IEEE Trans. Computers*, vol C-20, 1971 Nov, pp 1312-1321.

[5] L. Chen, A. Avizienis, "*N*-Version programming: A fault tolerance approach to reliability of software operation", *Proc. Int'l Symp. Fault-Tolerant Computing* (Toulouse, France), 1978 Jun, pp 3-9.

[6] Y. Chen, T. Chen, "Implementing fault tolerance via modular redundancy with comparison", *IEEE Trans. Reliability*, vol 39, 1990 Jun, pp 217-225.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to Algorithms*, chap 28, 1990, pp 634-653; McGraw-Hill.

[8] S. Dormido, M. A. Canto, "Synthesis of generalized parallel counters", *IEEE Trans. Computers*, vol C-30, 1981 Sep, pp 699-703.

[9] P. D. Ezhilchelvan, I. Mitrani, S. K. Shrivastava, "A performance evaluation study of pipeline TMR systems", *IEEE Trans. Parallel and Distributed Systems*, vol 1, 1990 Oct, pp 442-456.

[10] D. D. Gajski, "Parallel compressors", *IEEE Trans. Computers*, vol C-29, 1980 May, pp 393-398.

[11] H. Garcia-Molina, D. Barbara, "How to assign votes in a distributed system", *J. ACM*, vol 32, 1985 Oct, pp 841-860.

[12] D. K. Gifford, "Weighted voting for replicated data", *Proc. Seventh ACM Symp. Operating System Principles* (Pacific Grove, CA), 1979 Dec, pp 150-159.

[13] A. L. Hopkins, T. B. Smith, J. H. Lala, "FTMP-A highly reliable fault-tolerant multiprocessor for aircraft", *Proc. IEEE*, vol 66, 1978 Oct, pp 1221-1239.

[14] B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, 1989; Addison-Wesley. (Discussion on voting, pp 51-62)

[15] D. E. Knuth, *The Art of Computer Programming — vol. 3: Sorting and Searching*, sec 5.3.4, 1973, pp 220-246; Addison-Wesley.

[16] P. R. Lorczak, A. K. Caglayan, D. E. Eckhardt, "A theoretical investigation of generalized voters for redundant systems", *Proc. Int'l Symp. Fault-Tolerant Computing*, (Chicago), 1989 Jun, pp 444-451.

[17] F. P. Mathur, A. Avizienis, "Reliability analysis and architecture of a hybrid-redundant digital system: Generalized triple modular redundancy with self-repair", *AFIPS Conf. Proc.*, vol 36 (Spring Joint Computer Conf.), pp 375-383; AFIPS Press.

[18] B. Parhami, "A data-driven dependability assurance scheme with applications to data and design diversity", *Proc. Int'l Working Conf. Dependable Computing for Critical Applications* (Santa Barbara, CA), 1989 Aug, pp 105-112. Also in *Dependable Computing for Critical Applications*, edited by A. Avizienis and J.-C. Laprie, 1991, pp 257-282; Springer-Verlag.

[19] B. Parhami, "Generalized signed-digit number systems: A unifying framework for redundant number representations", *IEEE Trans. Computers*, vol 39, 1990 Jan, pp 89-98.

[20] B. Parhami, "The parallel complexity of weighted voting", *Proc of the 4th Int'l Conf Parallel & Distributed Computing and Systems*, Washington, DC, 1991 Oct.

[21] W. H. Pierce, "Adaptive decision elements to improve the reliability of redundant systems", *IRE Int'l Conv. Record*, 1962 Mar, pp 124-131.

[22] D. P. Siewiorek, "Reliability modeling of compensating module failures in majority voted redundancy", *IEEE Trans. Computers*, vol C-24, 1975 May, pp 525-533.

[23] D. P. Siewiorek, et al, "C.vmp: A voted multiprocessor", *Proc. IEEE*, vol 66, 1978 Oct, pp 1190-1198.

[24] D. P. Siewiorek, R. S. Swarz, *The Theory and Practice of Reliable System Design*, 1982 (Discussion on voting, pp 117-122); Digital Press.

[25] J. R. Sklaroff, "Redundancy management techniques for space shuttle computers", *IBM J. Research and Development*, vol 20, 1976 Jan, pp 20–28.

[26] E. E. Swartzlander, "Parallel counters", *IEEE Trans. Computers*, vol C-22, 1973 Nov, pp 1021–1024.

[27] Z. Tong, R. Y. Kain, "Vote assignments in weighted voting mechanisms", *IEEE Trans. Computers*, vol 40, 1991 May, pp 664–667.

[28] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components", in *Automata Studies* (Ann. Mathematics Studies, No 34), ed by C. E. Shannon and J. McCarthy, 1956, pp 43–98; Princeton Univ. Press.

[29] S. Waser, M. J. Flynn, *Introduction to Arithmetic for Digital System Designers*, chap 6, 1982, pp 215–233; Holt, Rinehart, & Winston.

[30] J. H. Wensley, C. S. Harclerode, "Programmable control of a chemical reactor using a fault-tolerant computer", *IEEE Trans. Industrial Electronics*, vol IE-29, 1982 Nov, pp 258–264.

AUTHOR

Dr. Behrooz Parhami, Professor & Vice Chairman; Department of Electrical and Computer Engineering; University of California; Santa Barbara, California 93106 USA.

Behrooz Parhami (S'70-M'73-SM'78) received his PhD in computer science from University of California at Los Angeles in 1973. His research deals with parallel architectures and algorithms, high-speed computer arithmetic, and dependable computing. In his previous position with Sharif University of Technology in Tehran, Iran (1974-88), he was involved in educational planning, curriculum development, standardization efforts, technology transfer, authoring of two textbooks, and various editorial responsibilities, including a 5-year term as Editor of *Computer Report*, a Farsi-language computing periodical. He is a member of the Association for Computing Machinery and the British Computer Society and a Distinguished Member of the Informatics Society of Iran for which he served as a founding member and President during 1979-84. He also served as Chair'n of IEEE Iran Section (1977-86) and received the IEEE Centennial Medal in 1984.

Manuscript TR90-142 received 1990 August 1; revised 1991 March 4.

IEEE Log Number 00125

◄**TR**►

Annual Reliability and Maintainability Symposium

The P. K. McElroy Award for Best Paper

was bestowed on John L. Stevenson & Joel A. Nachlas for their paper "Microelectronics Reliability Predictions Derived from Component-Defect Densities" that was given at the 1990 Symposium in Atlanta. For more information, see the *gold* section of your copy of the 1991 *Proceedings*.

Each year the Symposium presents The P. K. McElroy Award for the *best paper* at the previous Symposium. The Award consists of a plaque and a $1000 honorarium. There are two criteria for *best paper*:

- The content of the written paper is lucid, excellent, and important to the theory and/or practice of R&M engineering.
- The verbal presentation of the paper at the Symposium is likewise lucid and excellent.

P. K. McElroy was an intensely practical person. Papers that receive the Award must be able to make a difference to R&M engineers and/or managers. It is not enough that the content be competent and important; that competence and importance must be readily obvious in both the written and verbal presentations.

Before the Symposium, the content of each written paper is examined by the Program Committee for technical excellence and clarity of exposition. The best of the papers are chosen and referred to a select group of past General Chair'n of the Symposium. Each person in that group listens to each presentation, and that group choses the *best paper* to receive the P. K. McElroy Award.

More — on the outside.