Some Aspects about the Efficiency of Bit Voters Implementation

Szabolcs Szilágyi [*], Daniela E.Popescu [**] and Mirela Pater[***]
[*] Department of Computer Science,
University of Oradea, Faculty of Electrotehnics and Informatics,
410087 Oradea, Romania, E-Mail: depopescu@uoradea.ro

[**] Department of Computer Science,
University of Oradea, Faculty of Electrotehnics and Informatics,
410087 Oradea, Romania, E-Mail: sszilagyi@uoradea.ro

[***] Department of Computer Science,
University of Oradea, Faculty of Electrotehnics and Informatics,
410087 Oradea, Romania, E-Mail: mirelap@uoradea.ro

Abstract *- Voting is a fundamental operation in the realization of ultrareliable systems that are based on multi-channel 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 are described. Examples of resultant designs are given and each design is evaluated with respect to cost and performance.*

Keywords: *hardware voter, m-out-of-n voter, majority circuit, ALTERA*

I. 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. 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, handling of imprecise or approximate data, combination with standby or active redundancy, voting on digital "signatures" obtained from computation states to reduce the amount of information to be voted on, and dynamic modification of vote weights based on a *priori* reliability data 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. This has become a very active research area.

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.

This paper considers the design of bit-voters and we compare our design based on the performance and the cost implied by each method applied for design.

A. Gate-Level Design

A voter can be constructed as a two-level AND-OR digital logic circuit with

$g = n!/[m!\ (n - m)!]$ m-input AND gates and a single g-input OR gate for small values of the parameters m and n. Also, a two-level OR-AND realization, requiring

$g' = n!/[(m - 1)!\ (n - m + 1)!]$ $(n\text{-}m+1)$-input OR gates and a single g'-input AND gate, is possible. In the first realization, all distinct 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".

The two-level AND-OR realization is "simpler" than the two-level OR-AND version (in terms of both gate count and gate-input count) if $m > (n + 1)/2$. The complexities are equal for odd n if $m = (n + 1)/2$.

As an example, for a 2-out-of-5 voter, the two-level AND-OR design uses 10 two-input AND gates and a single 10-input OR gate while the OR-AND design is less complex with 5 four-input OR gates and one 5-input AND gate.

For large values of n, two-level designs are impractical. Assuming the use of f-input gates and ignoring the possibility of gate sharing, the total number of gates in the two-phase AND-OR and OR-AND realizations will change from $g + 1$ and $g' + 1$ to:

$G = g$ liub((*m*-l)/(*f*-l))+ liub((g-l)/(f-l))

$G' = g'$ liub((*n*-*m*)/(*f*-l))+liub((g'-1)/(f-l))

With gate sharing, an exact general gate-count analysis becomes difficult. However bounds for the number of gates can be obtained that are close to actual values and show the excessive complexity of this approach for large values of n. It is thus imperative to explore more structured design techniques.

B . Decomposition-Based Design

Hierarchical decomposition strategy (divide-&-conquer) can be used to facilitate the design. There are two ways to proceed with the decomposition approach:

1) Picking a partitioning scheme and then designing a suitable merging network.

2) Selecting a merging network and then designing the required partitioning algorithm.

With the first approach, we 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 voters to realize these contributions, and finally, design a logic network for combining the results. Because the subsets can be selected in many different ways, this approach does not lend itself to general analyses. We will thus limit our discussion to a simple example:

Consider the design of a 3-out-of-5 voter using the subsets $S_1 = \{x_1, x_2, x_3\}$ and $S_2 = \{x_4, x_5\}$. The combinations that match or exceed the threshold of 3 are:

3-of-3 in S_1 + (2-of-3 in S_1 and l-of-2 in S_2) + (l-of-3 in S_1 and 2-of-2 in S_2)

This yields the logical expression:
$x_1x_2x_3x_4 + (x_1x_2 + x_2x_3 + x_3x_1)(x_4 + x_5) +$
$+ (x_1 + x_2 + x_3)x_4x_5$
which directly translates into a 4-level logic circuit with 10 gates and 25 input lines.

We next explore the second decomposition strategy with *multiplexers* used as merging networks. Our interest in this approach arises from the availability of multiplexers as off-the-shelf universal components. The strategy is to select a subset of the inputs as control inputs to a multiplexer, determine the residual input functions, and then repeat the process for each function, if needed, until easily realizable functions are obtained.

For example, with a 2-input multiplexer in the first decomposition stage, the residual functions correspond to an m-out-of-(n-l) voter and an (m-l)-out-of-(n-l) voter. To design a 3-out-of-5 voter using 2-input multiplexers, we take x_1 as the first control variable. The residual functions corresponding to $x_1 = 0$ and $x_1 = 1$ are $x_2x_3x_4 + x_2x_3x_5 + x_2x_4x_5$ and $x_2x_3 + x_2x_4 + x_2x_5 + x_3x_4 + x_3x_5 + x_4x_5$, yielding the result:

$[x'_1[x'_2(x_3x_4x_5) + x_2h] + x_1[x'_2h + x_2(x_3 + x_4 + + x_5)]]$

where $h = x_3x_4 + x_3x_5 + x_4x_5$ has the 2-input multiplexer realization

$h = [x'_3(x_4x_5) + x_3(x_4 + x_5)]$.

The resulting circuit implementation using **ALTERA MAX+PLUS II** is shown in Figure 1a. Clearly, a 4-input multiplexer can replace the last two levels. With 8-input multiplexers, the expression becomes:

$[x'_1x'_2x_3h_1 + x'_1x_2x'_3h_1 + x'_1x_2x_3h_2 +$
$+ x_1x'_2x'_3h_1 + x_1x'_2x_3h_2 + x_1x_2x'_3h_2 + x_1x_2x_3]$

where $h_1 = (x_4x_5)$ and $h_2 = (x_4 + x_5)$. The resulting circuit is depicted in Figure 1b.

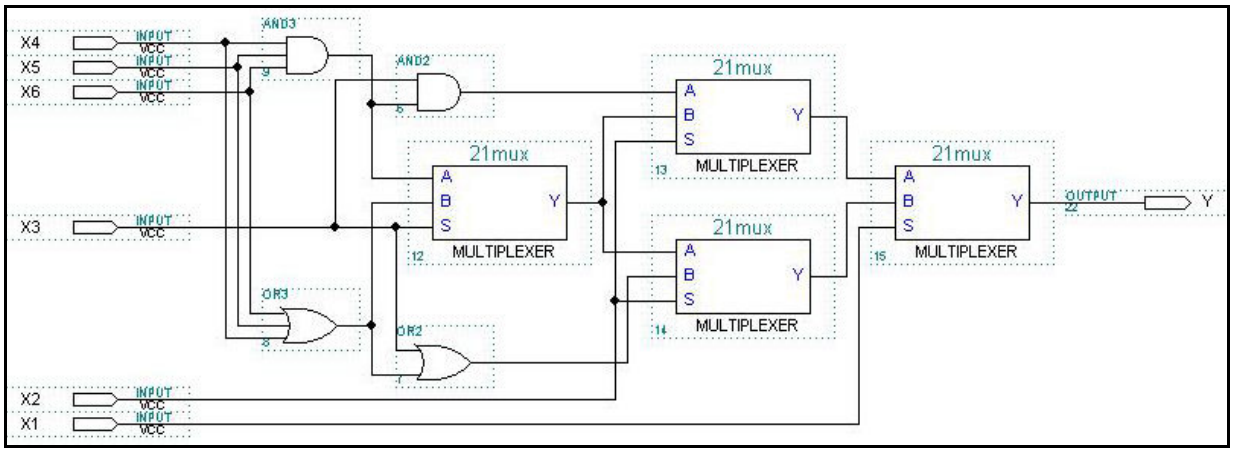

Figure 1a

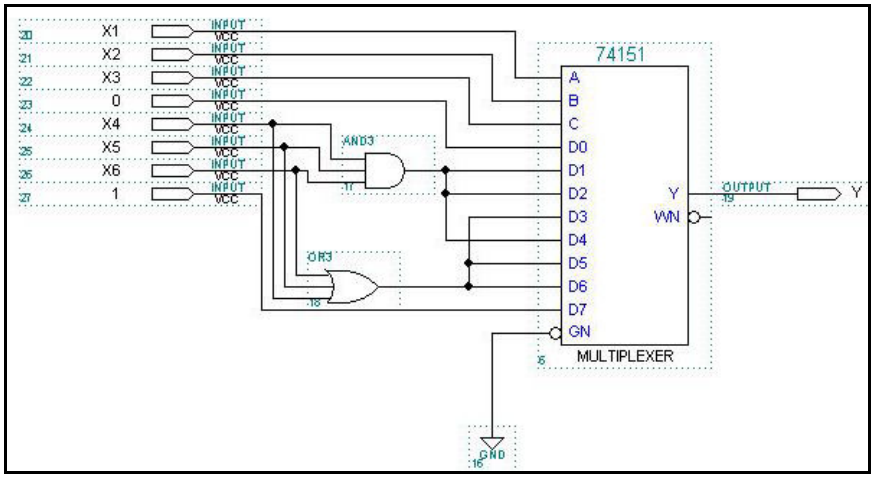

Figure 1b

C. Arithmetic-Based Design

In the "arithmetic" approach, the sign of $-t + \sum(x_i v_i)$ is computed. The products $x_i v_i$ can computed by AND gates and then added by standard carry-save technique to yield the final result. If the v_i s are fixed, the hardware realization can be optimized in each case by compressing the constant 0s in the binary numbers to be added.

Consider as an example a voter with 6 bit-inputs having fixed associated votes of 2, 2, 2, 2, 1, 1 and the threshold of 5. The arithmetic expression to be evaluated is:

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

The multiple-operand binary addition $(1011)_2 + (x_1x_4)_2 + (x_2x_5)_2 + (x_3x_6)_2$ can be performed by 4 full adders and 2 half adders organized in a 4-level circuit.(Figure 2) The leftmost 1 can be ignored since it only causes a complementation that cancels the complementation needed for obtaining the resultant output from the sign bit.

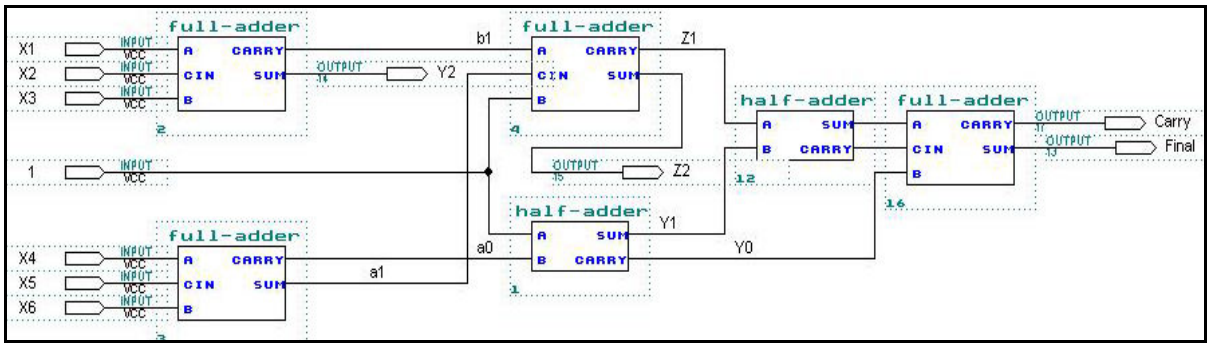

Figure 2

Instead of using full adders and half adders, one can use larger building blocks known as parallel counters, and parallel compressors, which convert a number of input bits to a smaller number of output bits while maintaining the arithmetic value being represented.

D. Design with Selection Networks

The design of an *m*-out-of-*n* voter is equivalent to selecting the m^{th} largest value from among *n* input bits. Selection networks can be built from 2-sorter (comparator) cells. Knuth defines three types of selection networks with *n* inputs:

1) Select the *m* largest values and move them to *m* outputs 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.

Denoting the number of 2-sorter or comparator cells by $U(m, n)$, $V(m, n)$, and $W(m, n)$ for type-1, type-2, and type-3 selectors above, we have:

$$U(m,n) < V(m,n) < W(m,n)$$

When dealing with bits, a two-sorter simply consists of a pair of 2-input gates: An OR to produce the larger and an AND to produce the smaller of the two values.

Type-3 selectors do more than what is required here. Type-2 selectors do exactly what we want. However, for most practical values of *m* and *n*, a type-1 selector augmented by an AND or OR circuit (that indicates whether all of the *m* largest values are 1s or whether all of the $n - m + 1$ smallest values are not all 0s) is both faster and more economical.

Consider the design of a 4-out-of-8 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 3.

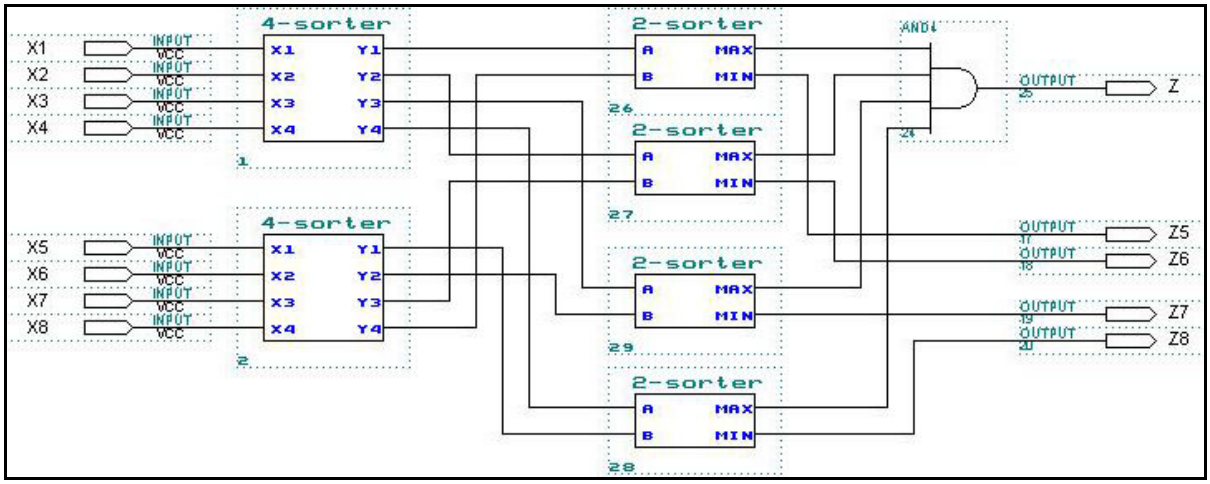

Figure 3

This selector requires 14 comparators (sorters) or 28 two-input gates with 4 gate levels of delay. A 4-input AND gate connected to the upper 4 outputs completes the circuit. Note that a 5-out-of-8 majority voter results if we connect an OR gate to the lower 4 output lines.

II. COMPARISON OF VARIOUS DESIGNS

We will compare the designs only for simple majority voters (i.e., when m = glib(n/2)+1. Figure4 shows the cost of majority voters designed based on 2-level logic expressions ("gate-level"), two-input multiplexer decomposition, the arithmetic-based approach, and selection networks, assuming maximum gate fan-in of 4. Figure 4 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 theory of selection networks is well-developed and efficient designs are available.

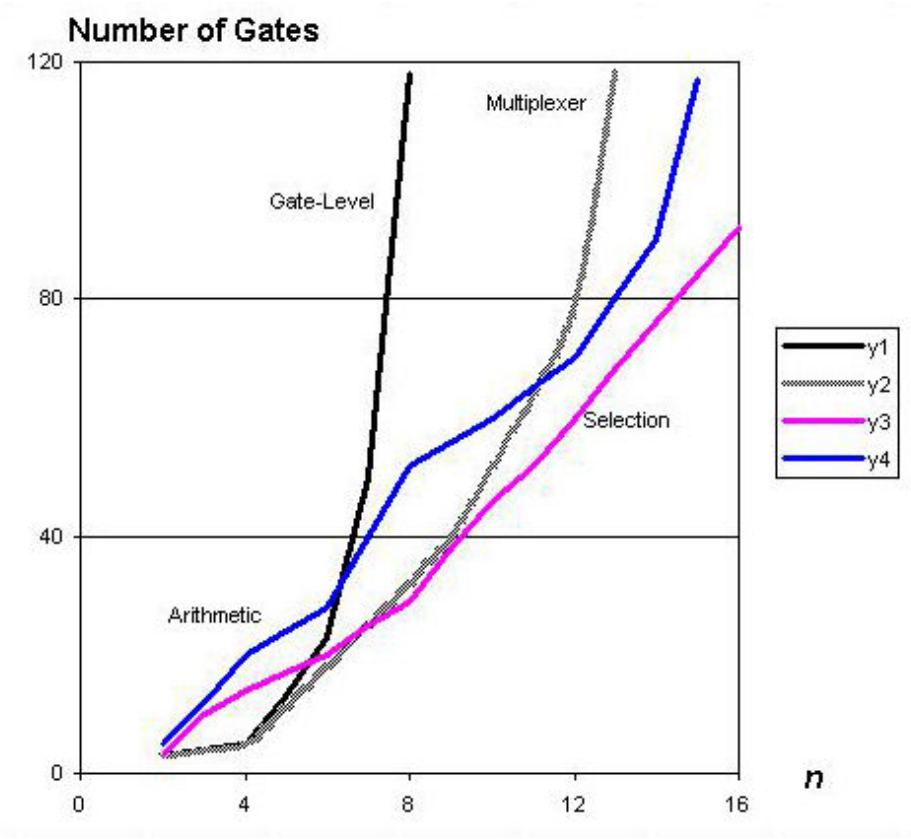

Figure 4

A comparison of delays is much more difficult. If the designs are used with pipelining, the differences in latencies (number of gate levels) are not significant 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.

III. CONCLUSIONS

In this paper, we have explored and compared some useful design techniques for m-out-of-n bit-voters. Despite the fact that the designs are quite practical, and in some cases asymptotically optimal, no claim is made as to their absolute efficiency or optimality. There may be other methods that yield better designs for a given set of requirements.

REFERENCES

[1] Vári K., Ştefan, “Sisteme Tolerante la Defecte”, Editura Universităţii din Oradea, 2001

[2] Parhami, B., "Voting Networks", IEEE Transactions on Reliability, Vol. 40, pp. 380-386, Aug. 1991.

[3] Knuth, D.E., The Art of Computer Programming — Vol. 3: Sorting and Searching, Addison-Wesley, 1973, Section 5.3.4, pp. 220-246.

[4] Parhami, B., "Design of *m*-out-of-*n* Bit-Voters", IEEE Transactions on Reliability, pp. 1260-1264, Aug. 1991.

[5] Swartzlander, E.E., "Parallel Counters," IEEE Transactions on Computers, Vol. C-22, No. 11, Nov. 1973, pp. 1021-1024.

[6] Waser, S. and M.J. Flynn, Introduction to Arithmetic for Digital System Designers, Holt, Rinehart, & Winston, 1982.

[7] Parhami, Behrooz., “Introduction to Parallel Processing – Algorithms and Arhitectures”, New York, pp. 129-133