Scalable Design of a Programmable NMR Voter with Inputs’ State Descriptor and Self-checking capability

Aleksandar Simevski\textsuperscript{1}, Elena Hadzieva\textsuperscript{2}, Rolf Kraemer\textsuperscript{1,3} and Milos Krstic\textsuperscript{3}
1) Brandenburg University of Technology, Konrad-Wachsmann-Allee 1, D-03046 Cottbus, Germany
2) Faculty of Electrical Engineering and Information Technologies, Karpos II, P.O Box 574, 1000 Skopje, Macedonia
3) IHP, Im Technologiepark 25, D-15236 Frankfurt (Oder), Germany
E-mail: \{simevski, kraemer, krsitic\}@ihp-microelectronics.com, hadzieva@feit.ukim.edu.mk

Abstract—One of the most frequently used techniques for increasing systems’ fault-tolerance is N-Modular Redundancy (NMR). The voter in an NMR system besides outputting the voting result can also indicate the situation on its input side, for instance, how many modules agree to the result of voting and which of the modules do not agree. Having this information, elaborated systems may initiate various actions e.g., interrupt or reset a processing unit or increment an error counter. Furthermore, since voters are a single point of failure in an NMR system, additional measures to increase its dependability are needed. Here, we present a design method for an NMR voter which along with the voting result, outputs the state of its inputs. It also makes self-checks of the consistency of its operation and signals errors. At last, the voter allows for each of its inputs to be defined whether the input takes part in voting or not i.e., the voter is programmable. The design method is based on a binary matrix (built according to the voter inputs) which has specific mathematical properties that enable scaling the design, as well as building the mentioned capabilities.

I. INTRODUCTION

The ever-increasing demand for dependability always imposes designers to search for new solutions. One of the strategies that is widely used is N-Modular Redundancy (NMR), especially Triple-Modular Redundancy (TMR) where N is three. NMR is straightforward to implement: to increase the dependability of a module, one should replicate it N times, feed the same input to all N modules and vote upon the outputs of the modules. The outputs of these modules are inputs to a voter. The voter’s output is equal to one of the inputs which belongs to the largest group of equal inputs. Majority voting requires at least \(N/2+1\) inputs to be equal, otherwise the module operation is considered to be compromised and an erroneous situation is signalled. For instance, in TMR there should be at least 2 inputs that are equal. In majority voting an odd \(N\) is preferable.

Elaborated systems employ various mechanisms to increase their dependability. Consider the following scenario. A dependable system acquires information from a sensor, and sends it to four processors to process it. It expects the results from all four processors to be identical. Four situations are possible: all four results are equal; here no action is required and the final result is equal to all results from the processors; another possibility is when one result is different from the other three equal results; in this situation the system assumes that the right result is equal to the three equal results; nevertheless, the system has to note the single disagreement, usually interrupting a microprocessor, or incrementing a counter which counts such disagreements; third possibility is when two equal results differ from the other two; here, a serious situation is signalled, non-maskable interrupts are invoked, parts of the system are reset and this kind of disagreement is noted while the system considers the two equal results to be the right one; further complication in this case is when two-by-two results agree, so two candidates for the right result are present; here, the system must additionally choose one of them and signal an ambiguous situation; the most “fatal” possibility is when all results are different; in this situation usually a global reset is invoked and the system operation is restarted.

The system from the example scenario may also find beneficial to know which of the processors do not agree with the others. It can actually recognize a permanent or transient fault at the processors’ subsystems observing their responses over a period of time. Basing on this information, it can make some decisions (for instance, not to deliver tasks to the processors with permanent faults anymore and shut it off the power supply to save power; on the other hand, if a transient fault is detected, the system can restart the affected processor(s) to bring them to a consistent state.) In a low-power scenario where dependability is required, the system may exhibit the following adaptive behaviour. Normally, only two processors are assigned to process the sensor information. As long as the processors agree i.e., their results are equal, the system considers that no faults are present. When they disagree, the system repeats the assignment procedure but additionally assigning the task to another, third processor. Only in critical situations all four processors are assigned to do the processing.

It is clear in these scenarios that the system has to choose the final result by some kind of voting. Additionally, it has to know which of the processors disagree and how many of them disagree, in order to take the appropriate action. Furthermore, in the low-power scenario the system has to
have either separate 2MR, 3MR and 4MR voters, or one programmable voter which can be all of them. It should be capable to dynamically form 2MR to NMR systems with any combination of active processors which is in effect an NMR on demand (NMROD) system. Dependability modelling of such systems is given by Al-Hashimi et al. [1] and Lombardi et al. [2].

One of the problems in NMR systems is the dependability of the voter itself – it is a single point of failure in the system. Correct operation of the system as a whole is impossible with a faulty voter. That’s why it is of paramount priority to increase voter’s dependability. (References regarding this topic are given in Section II.)

In this paper, we present a method for design of programmable NMR voters suitable for the illustrated scenarios. The voters describe what happens at their inputs. That is, they output the number of inputs that differ from the output of voting as well as which of the modules errs. Actually, the voter besides the output of voting also outputs an inputs’ state descriptor (ISD in further text). The ISD consists of the following:

- an $\lceil \log_2 N \rceil$-bit signal (or N-bit signal in decoded form) which tells the number of inputs that differ from the voter’s output;
- an N-bit signal in which each bit tells whether the output equals to the appropriate input;
- one-bit signal which tells whether an indecisive situation is present (e.g., two-by-two inputs are equal).

To increase the confidence that the voter functions correctly, consistency self-checks are employed. All these functions and actually the whole design is based on a binary matrix built in a manner shown in Section IV. This matrix has specific mathematical properties elaborated in the same Section. These properties actually give the opportunity to have scalable design of a programmable NMR voter with ISD and self-check capabilities. The designer easily generates an NMR voter just specifying the parameter N, and the width W of the input signals of the voter.

The rest of the paper is organized as follows. Section II presents related work that other authors have done in this area as well as references to other aspects of NMR. Section III gives the design specification of the NMR voter and introduces the basic definitions for the sake of the following Sections. The proposed method is presented in Section IV, and its theoretical analysis and practical results are described in Section V and Section VI, respectively. The conclusion is given in Section VII.

II. RELATED WORK

A similar motivation and approach to ours we find in the work of Jiang et al. [3]. In their work they present an NMR structure with concurrent error location capability which determines if an error occurred during voting, and if so, the location of the error. The error can occur in the replicated modules, the voter or in the very module that is checking for errors and their locations. They compare their work to the scheme proposed by Gatains [4] which is a totally self-checking TMR system, based on two-rail checkers. Nevertheless, while Jiang et al. [3] and Gatains [4] focus on locating the error by using special circuits that observe the redundant modules’ outputs and the voter output, our primary motivation is to make an easy-scalable design of a programmable NMR voter with ISD, and self-checks which increase its dependability. Other method for increasing an NMR voter’s reliability based on error correction by ADR (Alternate-Data Retry) is introduced by Takaesu and Yoshida in [5].

Regarding scalability and adaptability, Lo et al. [6] present a design of a reconfigurable NMR system. The procedure for design is the same for any value of N which makes it easy-scalable design. Furthermore, Ruiz et al. [7] show a generic strategy intended for automatic generation of replicated modules and appropriate majority voter to protect the modules with NMR techniques during HDL design time.

Dependability and other performance-related analyses of NMR systems is found in the work of Srihari [8], Koren and Su [9], and Beaudry [10].

III. BASIC DEFINITIONS AND DESIGN SPECIFICATION

The characteristics of the NMR system are mostly determined by the type of the voter. According to the application requirements, the system designer chooses the most appropriate voting algorithm. Here, we make a short classification of voters based on their properties. Basing on this short classification, we also present the type of voter that we intend to design and set the stage for the following Sections. Classification of voters with all analyses is given by Parhami [11].

Let the set of the inputs of the NMR voter be $A = \{x_0, x_1, \ldots, x_{N-1}\}$. N is the level of redundancy (N from NMR). Let the absolute difference between two input values $x_i$ and $x_j$ be $\delta$: $|x_j - x_i| = \delta$. The easiest to implement and actually the most used algorithm is the exact voting algorithm where $\delta$ must be 0 in order to consider $x_i$ and $x_j$ equal. Inexact voting algorithm on the other hand defines $\sigma$ for which if $\delta < \sigma$ then $x_i$ and $x_j$ are considered equal. The third type is approved voting where each input consists of a set or range of approved values. The voter chooses the most appropriate set of values presented on the inputs. In this paper we deal with exact voters and we will denote the single voting output with $y$.

A general M-out-of-N voter considers the voting successful if there are at least M equal inputs (of N inputs in total). If $M \leq N/2$ then ambiguous situations exist, in which 2 or more candidates can be legitimate outputs of the voting process. For instance, let in a 2-out-of-4 system $x_0 = x_1 = 23$, and $x_2 = x_3 = 45$. Both values 23 and 45 are legitimate candidates to be the output of voting.

Here, we elaborate a design strategy for an exact, programmable, 1-out-of-N voter for NMR systems. The output of voting $y$ is always equal to $x_i$, where $x_i$ belongs to the largest group of equal inputs. Besides determining the output of voting $y$, the voter also gives a complete description of the situation at the inputs $x_i$. That is, the number of inputs which differ
from the output – \( nrDiff \) (or equivalently the number of equal inputs – \( nrEq \)), the signals \( e_i \), \( i = 0, 1, \ldots, N – 1 \) which tell if \( x_i \) is equal to \( y \), and the ambiguous situation signal – \( amb \), form the Inputs’ State Descriptor – ISD. Furthermore, the voter checks the consistency of its operation and signals it with the \( err \) signal. (In further text, we will use the shorter notation \( i = r_l, r_h \) to express an integer index range from \( r_l \) to \( r_h \)).

The voter is programmed in the following way. Each input \( x_i \) has an associated programming bit \( pb_i \), which tells if the input is to be considered for voting, i.e., it is an active input. For example, if in a 4MR system \( pb_0 = pb_2 = 0 \) and \( pb_1 = pb_3 = 1 \), a 2MR system is formed, composed of modules 1 and 3, while 0 and 2 do not take part in it, i.e., they are inactive. This enables dynamically forming 1MR to NMR systems with any combination of redundant modules. There are two exotic situations: when no inputs are active (0MR) the situation is considered illegal and it is not guaranteed what is on the output \( y \); 1MR always transfers the active input to the output \( y \). In other words, for proper operation at least one input should be defined as active.

**Fig. 1. Programmable NMR voter with ISD and self-checks**

Fig. 1 shows the interface of a programmable NMR voter. The \( x_i \) inputs and the output \( y \) can be \( W \)-bits wide. Later, in Sections V and VI we present analysis and results for both programmable and non-programmable NMR voters designed with our method. The non-programmable voter can not be programmed and the \( pb_i \) inputs are not present.

Last characteristic, but not least important is that the design is easy-scalable.

**IV. SCALABLE DESIGN METHOD**

We base our method on a binary matrix which contains information which of the inputs are equal between themselves. Actually all information needed is included in this matrix, enabling us to easily determine the output, the ISD and make self-checks. In Subsection IV-A we present the matrix and its properties, and in Subsection IV-B we show how we determine the voter’s outputs using the matrix.

**A. Matrix construction and properties**

An \( N \times N \) binary matrix \( A = [A_{ij}]_{i,j=0}^{N-1} = [A_{ij}]_{i,j=0}^{N-1} \) is built as follows. If \( x_i = x_j \) then \( A_{ij} = A_{ji} = 1 \) otherwise \( A_{ij} = A_{ji} = 0, i, j = 0, \ldots, N-1 \). For example, let \( N = 4 \), \( x_0 = 20, x_1 = 30, x_2 = 20, x_3 = 10 \), and all inputs are active (\( pb_i = 1 \)). The matrix would be:

\[
A = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

The following properties of a matrix built in this way are obvious immediately.

**Property 1:** \( A = A^T \). That is \( A \) is always symmetric which implies that it is a hermitian matrix \( A = A^T \) and its eigenvalues are always real numbers \( \lambda_i \in \mathbb{R}, i = 0, N – 1 \).  

**Property 2:** \( A_{ij} = 1 \) for \( i = j \). All elements of the main diagonal are 1 since all elements are equal to themselves. This implies that \( \text{trace}(A) = N \).

**Property 3:** If all inputs are different, then \( A = I \) (identity matrix). On the other hand, if all inputs are equal, the matrix elements are all ones \( A_{ij} = 1 \).

Since for the sake of programming the voter has additional \( N \) input bits which define which of its inputs are active, we will consider each inactive input to be different from all other inputs no matter if they are active or not, placing 0 at the appropriate place.

What is not obvious immediately are the following assertions.

**Assertion 1:** A is positive semi-definite matrix.

**Assertion 2:** \( \lambda_i \in \mathbb{N}_0 \). That is, all eigenvalues of A are natural numbers or zero.

We leave the proofs of assertions 1 and 2, since they are beyond the scope of the paper and are actually rather long.

**Assertion 3:** The biggest number of equal inputs is equal to the maximal eigenvalue of A. (Put in a more convenient form: the biggest number of equal inputs is equal to the maximal number of ones in a row (column)).

**Proof:** Let the most frequent voter input appears \( r \) times, that is,

\[
\exists k_1, k_2, \ldots, k_r \in \{0, 1, 2, \ldots, N – 1\},
\]

\[
x_{k_1} = x_{k_2} = \ldots = x_{k_r}
\]

We don’t loose generality if we take \( k_1 < k_2 < \ldots < k_r \).

Since \( A \) is positive semidefinite matrix, all of its eigenvalues \( \lambda_i \) are nonnegative, so its spectral radius \( \rho(A) = \max_{i=0,1,\ldots,N-1} |\lambda_i| = \max_{i=0,1,\ldots,N-1} \lambda_i \). From linear algebra we know that the inequality

\[
\rho(A) \leq ||A||
\]

holds for every norm of \( A \) (Meyer [12], p.497). If we choose \( || \cdot ||_1 \) – norm, then we obtain

\[
\rho(A) \leq ||A||_1 = \max_j \sum_i |A_{ij}| = r
\]

since the largest absolute row sum in \( A \) is \( r \).
On the other hand, according to the Collatz-Wielandt formula for nonnegative matrix (Meyer [12], p. 670),
\[ \rho(A) = \max_{y \geq 0, y \neq 0} \frac{f(y)}{y}, \quad f(y) = \min_{0 \leq i \leq N-1, y_i \neq 0} \frac{[Ay]_i}{y_i}, \]
where \( y_i \) is the \( i \)-th component of the \( N \)-dimensional vector \( y \) and \([Ay]_i \) is the \( i \)-th component of the \( N \)-dimensional vector \( Ay \).

Let us choose a vector \( \bar{y} \) to be defined as
\[ \bar{y}_i = \begin{cases} 1, & i = k_1, k_2, \ldots, k_r \\ 0, & \text{otherwise} \end{cases}, \quad i = 0, N - 1 \]
which is equivalent to
\[ \bar{y}_i = \begin{cases} 1, & i = k_1, k_2, \ldots, k_r \\ 0, & \text{otherwise} \end{cases}, \quad i = 0, N - 1. \]

Then
\[ [Ay]_i = \begin{cases} r, & i = k_1, k_2, \ldots, k_r \\ 0, & \text{otherwise} \end{cases}, \quad i = 0, N - 1, \]
and
\[ \rho(A) = \max_{y \geq 0, y \neq 0} f(y) \geq f(\bar{y}) = \min_r \frac{r, r, \ldots, r}{r} = r. \quad (4) \]

Now, inequalities (3) and (4) imply that
\[ \rho(A) = \max_{i = 0, 1, \ldots, N-1} \lambda_i = r \quad (5) \]
which concludes the proof.

B. Construction of ISD and self-checks

Enough information for the inputs we get from the elements of the matrix \( A \) which are above (or below) the main diagonal. This follows from Properties 1 and 2. The elements of the matrix above the main diagonal from the example in Subsection IV-A are:

\[
\begin{pmatrix}
0 & 1 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{pmatrix}
\]

If we fill the missing places with zeros, we get a reduced \((N - 1) \times (N - 1)\) matrix:

\[
AR = \begin{pmatrix}
0 & 1 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{pmatrix}
\]

where we keep the original row and column enumeration as in matrix \( A \) for the elements above the main diagonal, that is \( i = 0, N - 2 \) and \( j = 1, N - 1 \).

We can now easily determine the ISD signals \( nRDiff, e_i \) and \( amb \). According to Assertion 3 we determine that \( nRDiff = \min_{i = 0, 1, \ldots, N-2} \{|A_{i,j}|, A_{i,j} = 0, j = 1, N - 1\} \). At the beginning we set \( e_i = pb_i, \quad i = 0, N - 1 \). (Let \( e_i = 1 \) mean that \( x_i \neq y_i \), and \( e_i = 0 \) mean that \( x_i = y_i \).) We search all rows \( i = 0, N - 2 \) of the \( AR \) matrix, to find a row \( i = 1 \), with the smallest number of zeros. (Here, we do not count the zeros which come from inactive inputs.) We assign \( y = x_1, e_1 = 0 \). Looping through columns \( j = \frac{1}{2}, N - 1 \) of row \( I \) we determine \( e_j \), for \( j \neq I \). That is, only if \( A_{j,j} = 1 \) we set \( e_j = 0 \), else it is left to the initialized value. If there is more than one row which has the smallest number of zeros then \( amb = 1 \), else \( amb = 0 \).

For instance, in our example the row \( i = 0 \) has the smallest number of zeros. Then, \( y = x_0 = 20 \), \( nRDiff = 2 \) (there are two zeros in row \( i = 0 \)), \( e_0 = 0 \). (At the beginning we have set \( e_0 = 1 \), but here we correct this because row \( 0 \) has the smallest number of zeros, \( aRDiff = 0 \). Now we loop through row \( 0 \) to determine \( e_j \), for \( j \neq 0 : A_{0,j} = 0 \) \( \Rightarrow e_j = 1 \), \( A_{0,j} = 1 \) \( \Rightarrow e_j = 0 \), \( A_{0,j} = 0 \) \( \Rightarrow e_j = 1 \), thus reflecting which input is equal to the output of voting. Since only row \( 0 \) has the smallest number of zeros, the situation is not ambiguous and \( amb = 0 \).

We were able to easily determine the ISD basing on Assertion 3, but also basing on the fact that the matrices \( A \) and \( AR \) are being built with the basic syllogism:

\[
\text{if } x_1 = x_j \land x_i = x_k \text{ then } x_j = x_k. \quad (6)
\]

For example, for \( N = 4 \), \( x_0 = x_1 \neq x_3 \)

\[
AR = \begin{pmatrix}
1 & 1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{pmatrix}.
\]

If \( AR_{12} \) is set to 0 instead of 1

\[
AR = \begin{pmatrix}
1 & 1 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{pmatrix},
\]
we see that such kind of a matrix could not be built according to our philosophy because it is contradictory. It states that \( x_0 = x_1 \), and \( x_0 = x_2 \), but in the same time states that \( x_1 \neq x_2 \). We label this matrix as erroneously built.

Assertion 4: The corresponding \( A \) matrix of an erroneously built \( AR \) matrix has at least one eigenvalue which is not a natural number or zero. That is, \( \exists \lambda_i \in \mathbb{R}, i = 0, N - 1 \). (We are currently working on the proof of this assertion.)

Exactly these “illegal” matrices allow the voter to make self-checks. Namely, for each \( i = 0, N - 2 \) of the \( AR \) matrix, and each \( j \) and \( k \) where \( j > i \) and \( k > j \) we check if the basic syllogism (6) is satisfied. If so, \( err = 0 \) else \( err = 1 \). Nevertheless, \( err = 0 \) does not mean that the voter is 100% operating correctly. It does mean that an error is not detected while building the matrix, but parts of the voter which later use the matrix may err and these errors are not caught.

Until now we have shown the way to a scalable design of a programmable NMR voter with ISD and self-checks, as well as its use-cases. In the following Sections V and VI we give the performance analyses and results in terms of propagation delay expressed through the levels of logic, and the complexity analysis expressed through the number of gates.

V. PERFORMANCE ANALYSES

A software voter based on this method is straightforward to implement. Nevertheless, our main focus in this paper is hardware. We give in this section an evaluation of the
performance of hardware voters designed by this method, i.e., propagation delay and area. Furthermore, in Section VI we present a practical implementation results for these figures, changing the basic parameters of N and the width W of the voter’s inputs. We investigate both a programmable and non-programmable voter built by this method.

**Propagation delay.** We estimate the propagation delay expressed through the levels of logic needed. Main building blocks are comparison circuits and multiplexers. Comparison circuits take approximately \( \log_2 w \) levels of logic, if \( w \) is the width of the signals they compare. Multiplexers also take \( \log_2 n \) levels of logic to choose among \( n \) inputs.

In our case, the voter makes \( N(N - 1)/2 \) comparisons between each of the input signals to build the \( AR \) matrix. Nevertheless, we can do all comparisons in parallel, so building the matrix takes approximately \( \log_2 W \) levels of logic. Next, for each row of the \( AR \) matrix the voter has to count the number of zeros. Instead of counting zeros, we can shift a variable once to the left, each time a zero is encountered in a row, thus getting the result in a decoded form. Pure shifting for a constant value takes zero levels of logic since only rewiring is needed (or one logic level if buffers are used). For each row the voter has to determine for which value (0 to N-1) the variable will be shifted and select among \( 2^{N-1} \) possible input values – that is \( \log_2 2^{N-1} \) levels of logic. This, repeated for each row means \( (N - 1) \log_2 2^{N-1} = (N - 1)^2 \) levels. Furthermore, the voter has to check first if the number of zeros of the current column is equal to the number of zeros of the previous column (for an ambiguous situation), and then to check if it is bigger. The order of these checks is not important, but they must be performed serially. The voter compares the N-bit wide, decoded number of zeros. Checking equality takes \( \log_2 (N) \), while checking which is bigger takes \( 2 \log_2 N \) levels of logic. That means \( 3 \log_2 N \) for these operations. At the end, one level of logic is needed to determine \( e_j \) for \( j \neq I \) in parallel.

These are the main operations which contribute to the levels of logic. The rest of the assignments take either constant or insignificant levels of logic or are parallel to the described operations. For instance, the self-checks account for N comparisons of two bits made in parallel with the operation of counting zeros. Summing up, the levels of logic are \( \approx \log_2 W + (N - 1)^2 + 3 \log_2 N + 1 = (N - 1)^2 + \log_2 2WN^3 \). We can see that the \( W \) parameter has a small impact on the propagation delay since it plays a role only during matrix construction, and its contribution is only \( \sim \log_2 W \).

This was an evaluation for a non-programmable voter. A programmable voter has to take into account the \( pb_i \) bits which define which input is active. We can simplify this if we assume that each input signal has 1 bit more. Thus, building the matrix takes \( \log_2 (W + 1) \). Furthermore, it has to know the actual number of active inputs which can be from 1 to N. That is choosing between N values, or plus \( \log_2 N \) levels of logic. At the end, when counting zeros, the voter has to shift only if the zero comes from active inputs. This is plus \( \log_2 N \) logic levels for each row i.e., \((N - 1) \log_2 N\). Now, the final approximation for the logic levels is \( \approx \log_2 (W + 1) + (N - 1)^2 + 3 \log_2 N + 1 + (N - 1) \log_2 N + \log_2 N = (N - 1)^2 + \log_2 2(W + 1)N^{N+3} \).

**Area.** We try to estimate the area through the number of gates needed. As said before, we need to make \( N(N - 1)/2 \) comparisons. The number of gates of a comparison circuit is roughly proportional to W, so the number of gates for building the matrix is \( \propto NW(N - 1)/2 \). Next, for each of the N-1 rows there are N-1 zero checks of one bit (which is negligible), and an N-bits wide comparison of the number of zeros. That is, we have approximately a gate count proportional to \( N(N - 1) \). Summing up, we have a number of gates proportional to \( N(N - 1)(W/2 + 1) \). In this case too, we see that area grows more rapidly when N is increasing (almost proportional to the square of N). On the other side, the area increases proportionally to W (if we take N as a constant).

For a programmable voter, we take the assumptions from above. The number of gates needed to build the matrix is now \( \propto N(W + 1)(N - 1)/2 \). We also have to add the number of gates of the circuitry for determining the number of active inputs which is proportional to N. Thus, we make an approximation of the total number of gates of \( \propto N(1\log_2 (W + 1)/2 + 1 + 1) \). We see that the proportionality is approximately the same for both non-programmable and programmable voters.

In the next Section VI we present practical implementation results of voters designed according to this method. It is interesting to see how these analyses fit the practical figures.

**VI. IMPLEMENTATION RESULTS**

In this section we present the synthesis results that we got for voters built according to this method. The synthesis of the voters is done in 0.13 micron technology. The synthesizer is instructed to do no optimizations in respect to the propagation delay and area in order to get the worst case. Figures of the levels of logic – LoL, length of the critical path – CPL in ns, number of gates, area in \( \mu m^2 \) and number of nets are given for both programmable and non-programmable voters, varying N and W. For the purpose of comparison, the same figures for a 3MR voter built in a traditional way, along with the figures for a non-programmable 3MR voter built according to our method are also presented.

Tables I and II present the figures for non-programmable voters varying N and W, respectively, while III and IV do the same for programmable voters.

| Table I | SYNTHESIS RESULTS FOR NON-PROGRAMMABLE VOTER. W IS FIXED TO \( 16 \) WHILE N VARIES. |
|---------|----------|----------|-------------|-------------|-----------|
| N | LoL | CPL (ns) | Nr. gates | Area (\( \mu m^2 \)) | Nr. nets |
| 2 | 9 | 2.43 | 52 | 583 | 84 |
| 3 | 11 | 4.52 | 141 | 1438 | 189 |
| 4 | 23 | 5.39 | 274 | 2889 | 338 |
| 5 | 27 | 7.20 | 424 | 4569 | 504 |
| 6 | 37 | 9.69 | 608 | 6663 | 704 |
| 7 | 46 | 10.67 | 846 | 9249 | 958 |
| 8 | 59 | 13.92 | 1116 | 12234 | 1244 |
### TABLE II
SYNTHESIS RESULTS FOR NON-PROGRAMMABLE VOTER. N IS FIXED TO 4 WHILE W VARIES.

<table>
<thead>
<tr>
<th>W</th>
<th>LoL</th>
<th>CPL (ns)</th>
<th>Nr. gates</th>
<th>Area (µm²)</th>
<th>Nr. nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
<td>4.36</td>
<td>100</td>
<td>934</td>
<td>104</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>4.61</td>
<td>138</td>
<td>1207</td>
<td>146</td>
</tr>
<tr>
<td>4</td>
<td>18</td>
<td>4.67</td>
<td>179</td>
<td>1569</td>
<td>195</td>
</tr>
<tr>
<td>8</td>
<td>19</td>
<td>5.19</td>
<td>178</td>
<td>1816</td>
<td>210</td>
</tr>
<tr>
<td>16</td>
<td>23</td>
<td>5.39</td>
<td>274</td>
<td>2889</td>
<td>338</td>
</tr>
<tr>
<td>32</td>
<td>24</td>
<td>5.57</td>
<td>458</td>
<td>4993</td>
<td>586</td>
</tr>
</tbody>
</table>

### TABLE III
SYNTHESIS RESULTS FOR PROGRAMMABLE VOTER. W IS FIXED TO 16 WHILE N VARIES.

<table>
<thead>
<tr>
<th>N</th>
<th>LoL</th>
<th>CPL (ns)</th>
<th>Nr. gates</th>
<th>Area (µm²)</th>
<th>Nr. nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>17</td>
<td>3.78</td>
<td>80</td>
<td>825</td>
<td>114</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>6.61</td>
<td>205</td>
<td>2076</td>
<td>226</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
<td>8.87</td>
<td>406</td>
<td>492</td>
<td>474</td>
</tr>
<tr>
<td>5</td>
<td>44</td>
<td>11.57</td>
<td>646</td>
<td>6674</td>
<td>732</td>
</tr>
<tr>
<td>6</td>
<td>60</td>
<td>15.17</td>
<td>971</td>
<td>9977</td>
<td>1074</td>
</tr>
<tr>
<td>7</td>
<td>74</td>
<td>18.36</td>
<td>1306</td>
<td>13759</td>
<td>1427</td>
</tr>
<tr>
<td>8</td>
<td>90</td>
<td>22.96</td>
<td>1836</td>
<td>18660</td>
<td>1973</td>
</tr>
</tbody>
</table>

### TABLE IV
SYNTHESIS RESULTS FOR PROGRAMMABLE VOTER. N IS FIXED TO 4 WHILE W VARIES.

<table>
<thead>
<tr>
<th>W</th>
<th>LoL</th>
<th>CPL (ns)</th>
<th>Nr. gates</th>
<th>Area (µm²)</th>
<th>Nr. nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33</td>
<td>9.61</td>
<td>231</td>
<td>2134</td>
<td>239</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>9.97</td>
<td>271</td>
<td>2413</td>
<td>285</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>9.93</td>
<td>308</td>
<td>2758</td>
<td>328</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>9.78</td>
<td>312</td>
<td>3024</td>
<td>348</td>
</tr>
<tr>
<td>16</td>
<td>35</td>
<td>8.87</td>
<td>406</td>
<td>4092</td>
<td>474</td>
</tr>
<tr>
<td>32</td>
<td>35</td>
<td>9.00</td>
<td>590</td>
<td>6195</td>
<td>722</td>
</tr>
</tbody>
</table>

If we check the results for the logic levels and gate count from the tables we see that they approximately match the analyses. It is straightforward to observe that the logic levels approximate the results calculated with the formulas shown in Section V. On the other side, for the gate count (or area) we have to include offset and scale constants to see that the analyses from Section V apply, since we have only analyzed the proportionality of the gate count with N and W, not the actual figures. Pursuing the actual figures is more difficult and not worth the effort in this situation.

We can get an approximate result in the following way. First, (for W=16=const.) we multiply N(N-1) with a scaling constant 20 for non-programmable and 33 for programmable, to get that the gate count is very close to 20N(N-1) and 33N(N-1), respectively. On the other side, for N=4=const. the gate count is approximately 10W+110 for non-programmable and 10W+250 for programmable voters. For N=4 and synthesis with optimized area, the formulas 20W+30 for non-programmable and 20W+150 for programmable voters give very close results to the ones got from synthesis. All these constants can be numerically determined from the table data for each N or W and each specific technology and synthesis process.

Figures 2, 3, 4 and 5 graphically represent the results for the levels of logic and gate count from the previous tables, contrasting the programmable vs. non-programmable voters.

Table V shows the figures for a traditional 3MR voter (varying W), while Table VI shows the same for a non-programmable 3MR voter designed according to our method. The traditional 3MR voter has three inputs (each of them W-
Fig. 5. Gate count for N=4

Fig. 6. Levels of logic of a simple, traditional 3MR voter vs. non-programmable 3MR voter

bit-wide), W-bits wide output, and one bit output signal which tells if at least two inputs are equal or not. This is the simplest 3MR voter.

TABLE V
SYNTHESIS RESULTS FOR A SIMPLE, TRADITIONAL 3MR VOTER.

<table>
<thead>
<tr>
<th>W</th>
<th>LoL</th>
<th>CPL (ns)</th>
<th>Nr. gates</th>
<th>Area (µm²)</th>
<th>Nr. nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>1.77</td>
<td>11</td>
<td>106</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>1.93</td>
<td>11</td>
<td>252</td>
<td>37</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>2.17</td>
<td>54</td>
<td>451</td>
<td>66</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>2.34</td>
<td>52</td>
<td>583</td>
<td>76</td>
</tr>
<tr>
<td>16</td>
<td>11</td>
<td>2.75</td>
<td>108</td>
<td>1190</td>
<td>156</td>
</tr>
<tr>
<td>32</td>
<td>12</td>
<td>3.08</td>
<td>209</td>
<td>2543</td>
<td>305</td>
</tr>
</tbody>
</table>

TABLE VI
SYNTHESIS RESULTS FOR A NON-PROGRAMMABLE 3MR VOTER.

<table>
<thead>
<tr>
<th>W</th>
<th>LoL</th>
<th>CPL (ns)</th>
<th>Nr. gates</th>
<th>Area (µm²)</th>
<th>Nr. nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
<td>3.26</td>
<td>45</td>
<td>420</td>
<td>48</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>2.99</td>
<td>66</td>
<td>559</td>
<td>72</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>3.19</td>
<td>88</td>
<td>753</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>3.90</td>
<td>80</td>
<td>887</td>
<td>110</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>4.32</td>
<td>141</td>
<td>1488</td>
<td>189</td>
</tr>
<tr>
<td>32</td>
<td>18</td>
<td>4.56</td>
<td>242</td>
<td>2653</td>
<td>338</td>
</tr>
</tbody>
</table>

Tables V and VI show that the non-programmable 3MR voter designed by our method (with ISD and self checks) has approximately 1.5 times more levels of logic than the traditional, simple 3MR voter. For the gate count (or area) when W is increasing, the difference between the gate counts (areas) of the voters is decreasing. Thus, for W=1 our voter has approximately 4 times more gates than the traditional, while for W=32 our voter is only around 1.15 times bigger than the traditional. Figures 6 and 7 depict the results from Tables V and VI graphically.

VII. CONCLUSION

Systems which require high dependability in their operation have various mechanisms to achieve it. Outlining our motivation in Section I we gave an example dependable system which gathers information from sensors and sends it to four processors to process it, expecting equal results returned by the processors. The system reacts differently when one, two or all results differ. It also observes which processors err and notes their errors. In low-power scenario, the system assigns tasks only to two processors, thus forming 2MR system, and only when errors occur it includes the other processors, thus dynamically forming 3MR or 4MR system.

In order to allow such scenarios, a scalable design method for programmable NMR voters which indicate the state of their inputs is proposed. The indication of the inputs’ state consists of information which inputs differ from the voting output, the actual number of inputs which differ from the output and whether an ambiguous situation is present at the input’s side. Additionally, self-checking mechanism is employed since the dependability of the voter itself is of primary importance in NMR systems. The design method is based on a binary matrix built according to the values of the inputs, examining whether they are equal between each other or not. This matrix has specific properties which enable scalability and

2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2012)
easy construction of the voters.

Performance analysis of voters designed by this method described in Section V shows that the propagation delay is \( \approx (N - 1)^2 + \log_2 2W^N \) for a non-programmable NMR voter, and \( (N - 1)^2 + \log_2 2(W + 1)^N \) for programmable. The area requirements are proportional to \( N(N - 1)(W/2 + 1) \) for non-programmable and to \( N\{(N - 1)((W + 1)/2 + 1) + 1\} \) for programmable voter. Practical implementation results described in Section VI confirm these analyses through a number of data gathered for voters with various \( N \) and \( W \) parameters. They also show that a non-programmable 3MR voter designed by this method has approximately 50\% more levels of logic than the simple, traditional 3MR voter, while their area ratio decreases from 4 for \( W=1 \) to 1.15 for \( W=32 \).

REFERENCES


