Teijo Lehtonen | Juha Plosila | Jouni Isoaho

On Fault Tolerance Techniques towards Nanoscale Circuits and Systems

Turku Centre *for* Computer Science

TUCS Technical Report
No 708, August 2005

On Fault Tolerance Techniques towards Nanoscale Circuits and Systems

Teijo Lehtonen
University of Turku, Department of Information Technology
Lemminkäisenkatu 14 B, 20520 Turku, Finland
tetale@utu.fi
Juha Plosila
University of Turku, Department of Information Technology
Lemminkäisenkatu 14 B, 20520 Turku, Finland
juplos@utu.fi
Jouni Isoaho
University of Turku, Department of Information Technology
Lemminkäisenkatu 14 C, 20520 Turku, Finland
jisoaho@utu.fi

TUCS Technical Report
No 708, August 2005

Abstract

The move towards nanoscale circuits poses new challenges to circuit design. As the dimensions shrink, it is becoming increasingly difficult to control the variance of physical parameters in the manufacturing process, for instance the concentration of dopants, the thickness of the gate and the insulation oxides, the width and thickness of metal wires, etc. This results in decreased yield which increases the costs per functioning chip. Electromigration causes intermittent and permanent failures after some period of operation, which means that these faults cannot be observed in the manufacture test. The problem of electromigration increases when going further to nanometer regime because of the decreasing width and increasing deviation of wires. Lowering the supply voltages make the circuits more vulnerable to noise and background radiation resulting in a higher soft error rate.

The only reasonable way to cope with these reliability problems is to build the circuits fault tolerant. Therefore, the yield can be maintained at an acceptable level by admitting some amount of faults in a chip. Electromigration problems can be overcome by the use of built-in redundancy and dynamically reconfigurable circuit structure. The soft errors can be handled by using static redundancy methods like hardware, information and time redundancy.

This report discusses fault tolerance techniques for nanoscale structures. It begins with a study of phenomena that the move towards nano introduces. A gategorization for fault types is presented and the different impacts of scaling into nano regime are connected to these types. Later in the report a number of fault tolerance techniques are examined and their suitability for nanoscale circuits and systems is evaluated. Each technique is connected to one or several fault types according to their properties for fault tolerance perspective.

Finally it is concluded that no single technique is enough for tolerating all the types of faults in nanosacle circuits and systems. Therefore a combination of two or more techniques is needed. The optimal mixture is design specific according to its usage purpose and proneness to different defect sources.

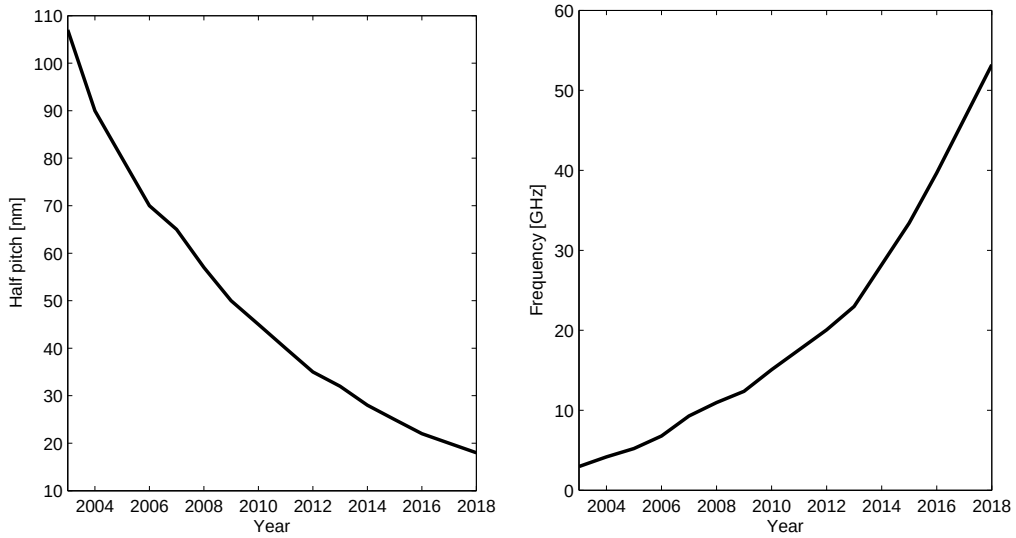
Keywords: fault tolerance, nanoscale circuits, static redundancy, dynamic redundancy, error correcting codes, fault detection

TUCS Laboratory
Communication Systems
Distributed Systems Design

Contents

1 Introduction

The technology development that has been the trend for already decades is expected to continue at the same speed or possibly at slightly slower course for at least the next 10 years. The nano age has already began (dimensions less than 100 nm) and the 50 nm half pitch[1] is expected to be achieved by the end of this decade. At the same time the operation frequency is expected to increase to 15 GHz and a single die can consist of over 4 billion transistors (see Fig. 1). [26]



(a) The expected half pitch lengths.

(b) The expected frequencies.

Figure 1: Trends in circuit development [26].

When going even further, new technologies are expected to replace CMOS that has been the state-of-the art technology for already many decades. Promising technigues include e.g. *single-electron transistors (SET)*, *carbon nanotubes*, *quantum cellular automata (QCA)*, and *molecular transistors*. [25]

The development trend introduces a wide variety of problems to the reliability of circuits. The smaller structures are harder to manufacture and the deviations are larger. Higher frequencies pose strict limits to timing and therfore also adds the propability of timing errors. The increased integration of devices on a single die raises the propability of errorneous components in a die. E.g. if the probability of an error in a transistor is 10^{-9} then for a chip containing 100 000 of these transistors the overall probability of no errors is 99,99% but for a chip of 100 million transistors the probability is only 90,48%. Increasing further the amount of transistors in a single chip results in dramatically low values for the probability of faultless circuits, e.g. for one billion transistor circuit the percentage is only

[1]Half pitch is the half of the average of the wire width added with the distance between two adjacent wires. The technology half pitch is either DRAM metal half pitch or ASIC/MPU metal-1 half pitch, which one is smaller (lately it has been the DRAM metal half pitch).

36,79%! Chips containing one billion transistors in a single chip are expected to be in production at 2012 [26].

The reliability problems are even worse for techniques beyond CMOS. The new technologies that are expected to be available in the near future, are predicted to have device failure rates of 10% to 30%. [48]

The state-of-the-art method for coping with manufacture errors is to abandon every circuit not operating completely correctly. The validation is done with the help of manufacture tests. As the failure rates increase, more and more circuits are put aside and thus the manufacture yield decreases. This raises the cost per functioning chip. The decreasing yield has been tried to be handled many ways. One way has been to use the errorneous circuits in systems where the presence of errors does not matter or the circuits can be used because most of the circuit operates normally and the error affects only a small fraction of the circuit. Examples of circuits which can be used although there are errors, are video image packing MPEG encoder and memory for phone answer machine, while e.g. toy manufacturer can use chips that fail in floating-point unit and are otherwise operational. [9] Another method to gain better yield, is the *design for manufacture (DFM)*, which means creating layouts that are easier to manufacture and thus contain less errors. Such methods are e.g. to extend from minimum dimensions, where it is possible. [13, 18]

Previosuly fault tolerance was issue in only safety-critical designs but because of the increasing propability of failures, the fault tolerance will be part of nearly every design process in future. The introduction of fault tolerance gives the possibility to accept circuits containing some failures and thus achieve better manufacture yield. The fault tolerance gives also answer to the failures introduced during the usage of the chip, which obviously cannot be handled by methods used in conjunction with the manufacture process.

The fault tolerance methods used in older circuits sturctures have mainly been designed for coping with situations of single errors. In future, multiple errors are expected to occur in circuits because of the increasing failure rate, and therefore the fault tolerance methods capable of handling multiple faults are needed.

In this report we give an overview of the available fault tolerance technigues and evaluate their efficiency to the demands of future nanoscale circuits. The report is organized as follows.

In Section 2 the fault sources are examined and the faults are categorized to three categories: *permanent*, *intermittent* and *transient errors*. At the end of the section also the used fault models are shortly presented. Fault tolerance methods can be divided to static and dynamic redundancy. The former ones are discussed in Section 3 and the latter in Section 4. Conclusions are presented in Section 5.

2 Faults in Nanoscale Circuits

As moving to the very-deep sub-micron (VDSM) or nano regime in CMOS chips, there will be a whole new bunch of new problems and at the same time a list of old problems are getting more severe. The problems rise from the shrinking geometries, smaller transistors, lower power voltages, higher frequencies, denser transistor integrations, etc.

In this section we first classify the fault types to three gategories and later discuss the most common error sources and connect them to different error types. At the end of the section the fault models used in circuit design are introduced. In the following sections means to cope with these presented errors are studied.

2.1 Error Types

The errors can be divided into three main groups: *permanent, intermittent* and *transient errors* according to their stability and occurence. In the following these main groups are shortly defined. [12]

2.1.1 Permanent Errors

Permanent errors are irreversible physical changes in a chip. The most common sources for this kind of errors are the manufacture processes, but permanent errors occur also during the usage of the circuit, especially when the circuit is old and therefore starts to wear out. Common to all permanent errors is that once they have occured, they will not vanish and though the test to detect them can be easily repeated with the same results.

The manufacture testing is proposed to detect the permanent errors caused by the manufacture processes and dismiss the circuits containing errors. If a permanent error occurs during the usage of the chip, the errorneous circuit needs to be replaced.

2.1.2 Intermittent Errors

Intermittent errors are occasional error bursts that usually repeat themselves every now and then but are not continuous as permanent errors. Errors are caused by unstable or marginal hardware, which are activated by environment change such as temperature or voltage change. Intermittent errors often precede the occurrence of a permanent error, for instance if there is an increased resistance in a wire before it totally breaks down creating an open. This type of errors are commonlly observed also when a circuit operates normally correctly but for some input instance it operates incorrectly because of some path in a circuit may be slower than supposed to but not totally unoperable.

Intermittent errors are very hard to detect because they may occur only under certain environment constraints or for some specific input combination. The way to repair these errors is to change the faulty circuit.

2.1.3 Transient Errors

Transient errors are temporal single misfunctions caused by some temporary environmental conditions which can be external phenomenom such as radiation or noise originating from the other parts of the chip. Transient errors do not make any permanent marks on the chip and therefore they are also called *soft errors (SE)*. A common impact of an transient error is a change of value in a single bit. Another term *single-event upset (SEU)* is used for soft error, which describes the fact that misfunctions (upsets) are commonly caused by single events such as an absorbed radiation.

The occurence of transient errors is commonly random and therefore hard to detect. Because of the random nature of these errors, a common measure for transient errors is the probability of occurence called the *soft error rate (SER)*. This rate describes both the tolerance of the circuit to variable effects causing soft errors, and the amount of these effects in the environment where the circuit is operating, e.g. SER is much higher in space because of the larger amount of background radiation than for the same chip operating in terrestrial conditions. The SER can be decreased by special consern to e.g. low-noise properties during the circuit design.

2.2 Error Sources

The error sources can be classified according to the phenomenom causing the error. Such origins are for instance: *the manufacture process*, *physical changes during operation*, *internal noise* caused by other parts of the circuit and *external noise* originating from the chip environment.

2.2.1 Manufacture Process

The most common defects in a chip are *spot defects* and *bridging faults* caused by silicon impurities and lithography and process variations. These defects cause permanent errors in a circuit. The probability of these defects is likely to increase as a greater amount of transistors will be integrated in a single chip and the size of chips is increasing, and at the same time the devices and wires get smaller. This results in decreasing yield which means higher costs per functioning chip.

The move towards nano scale circuits rises also a list of new problems originating from the manufacturing process. As the dimensions shrink the proportional extent of deviations become larger and their effects more severe. Lithography deviation is the main reason for gate length deviations. Doping profile fluctuations on the other hand cause deviation of the threshold voltage. These together with the increase of resistive vias and contacts result in large operation speed deviation. At the same time the operation frequency of the circuits is expected to increase rapidly. The worst case scenario of series of ”slow” devices may lead to timing violations and therefore to malfunction of the circuit. This is considered an in-

termittent error because the circuit might work correctly for the most of the time, which would not be the case for permanent error.

Metal slivers are small pieces of metal between two metal wires (see Fig. 2). In normal conditions this metal piece does not touch the wires but when the temperature increases it creates a short between wires because the metal widens as the temperature increases. This is a typical intermittent error but high voltage may also cause this short to ”burn in” resulting in a permanent error.

On the opposite to slivers are the *cracks* in metal wires (see Fig. 2) which result in opens at low temperatures but may be totally functioning at normal temperature or occur as an increased resistance. This is another typical intermittent error source. [19, 21, 56]

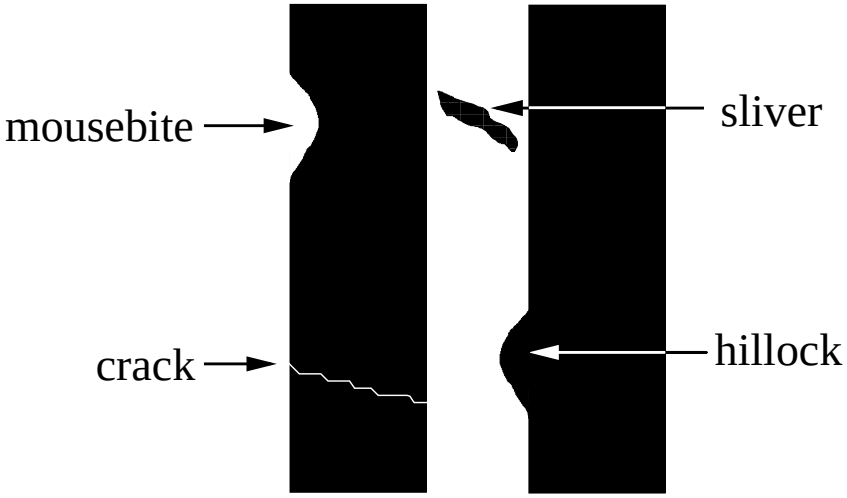


Figure 2: Manufacture faults in wires.

2.2.2 Physical Changes During Operation

Electromigration means current-induced atomic transport which is generated by collisions of electrons with metal atoms. This phenomenom is especially critical if there are so called *mousebites* or *hillocks* in the metal wires (see Fig. 2). In a place of mousebite the wire is narrower which means that the current-density is higher and so the electromigration is stronger. Additionally the wire is already narrower in that place so the narrowing impact of electromigration causes rapidly increasing resistivity and finally an open. In place of a hillock material is accumulated and electromigration moves more material to a such place because now the current density is lower than in other parts of the wire. This can eventually result in a short to another wire. The electromigration impact is observed often first as an intermittent error and as a permanent error if an open or short is created.

Electromigration is becoming more and more severe problem as the dimensions of wires and insulators between wires decrease at the same time as the deviation increases. Also the increasing operating temperature is reported to strengthen

the consequences of electromigration. One way to reduce the electromigartion problems has been the use of copper in wires because it provides higher electromigration threshold than aluminium used before. [6]

The ever thinner gate oxides are prone to current tunneling resulting in breakdown which means a permanent error. On the other hand so called *soft breakdown (SBD)* in ultrathin gate oxides slows down transistors causing intermittent errors same way as other device deviations. The soft breakdown differs form the traditional *hard breakdown* in a way that it causes current fluctuations while the hard breakdown results in shortcuts and thus measurable current. The SBD effects can be thought as so tiny breakdowns that they only partially violates the operation of the transistor, which is also the reason for the name ”soft”. SBD is also a typical source for increased power consumption.

2.2.3 Internal Noise

As the circuit dimension decrease also the supply voltage is scaled down. There are many reasons for this but the main one is the durability of components and wires. High voltages expose the ever thinner gate oxides for breakdown and high current densities in narrow wires accelerate electromigration. Also the energy consumption is decreased as the supply voltage is decreased. The noise tolerance of the nano scale circuits will be smaller because of this decreasing trend of the supply voltages (see Fig. 3). The impact is further strengthened by the large deviation of the threshold voltage which means that for some circuits or some parts of the circuit the noise tolerance is negligible.

At the same time the noise sources are increasing as the proportional fluctuations in the manufacturing processes get larger. Especially the resistive vias and contacts as well as deviations on wires introduce noise to signals.

The crosstalk noise between signal lines is about to increase, because the height and width of the signals will not be scaled by the same factor. The width of the smallest wires will be of the same scale as the length of the gate but the height of the wire is larger in order to keep the resistance of the wires tolerable. This increases the capasitive surface among adjacent wires. At the same time, the wire spacing and also the distance of adjacent layers gets smaller, which additionally increases the capacitive coupling.

The higher frequency in nanoscale circuits gives rise also to inductance based noise called the *skin effect*. As the current changes rapidly it flows near the wire surface which means increased resistance. Because it depends on frequency, how near to the wire surface the current flows, the wire resistance will vary with frequency.

One type of noise are the timing inaccuracies. As the deviations of components and wires increase it will be impossible to get a signal to two or more distinct nodes in the circuit at the exactly same moment. The increasing timing jitter is a problem not only in synchronous systems but also in asynchronous designs that use delay elements, because the delay length cannot be set that easily.

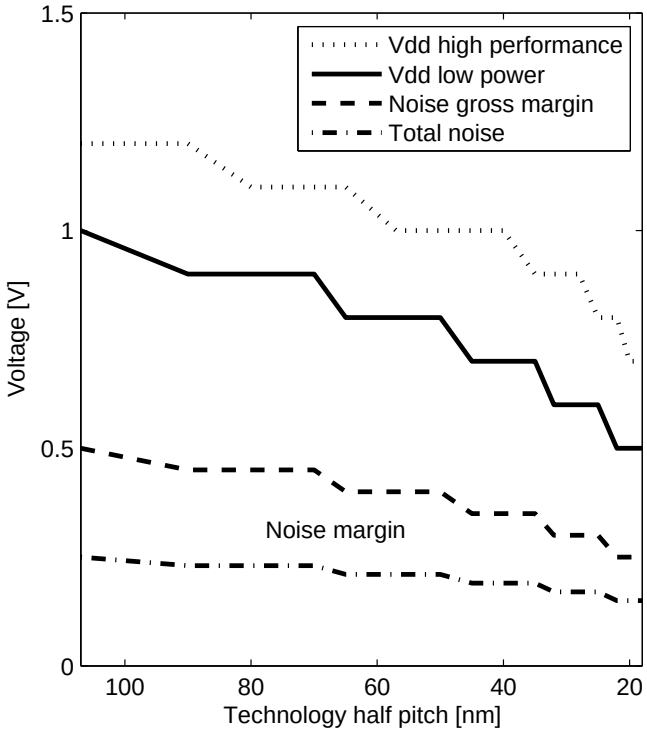


Figure 3: The noise margin decreases as the technology is scaled down [26]. The gross noise margin is calculated from the low power V_{dd} and the total noise with independent noise of 50 mV and proportional noise of 0.2.

The impacts of noisy circuits can usually be modelled as transient or sometimes as intermittent errors. A lot of work has been done to minimize the impact of noise to circuit functionality and make chips more tolerant to noise. When moving towards nanoscale circuits many of these methods become unusable because of the changing noise sources and their relative importance in overall noise. Therefore new methods are and need to be developed.

2.2.4 External Noise

Radiation has not been regarded as a severe noise source unless the circuit is to be used in space, aeroplanes, nuclear plant or similar places where background radiation is higher than usual terrestrial amounts. As moving towards nanoscale circuits the radiation should be taken into consideration also in other circuits, because the shrinking dimensions cause increasing propability that an α -particle, proton or neutron hitting the chip also causes a bit value to change. The occurence of upset is more likely because lowering supply voltages together with smaller transistors means that the charge the particle introduces is enough to flip a bit resulting in an error. The charge needed to flip a bit and cause a particle-induced transient is called *critical charge* and it is dependent on the charge a transistor can hold. E.g. for 90nm CMOS technology the charge a transistor holds is 1-10fC, which can be

compared to the charge of over 100fC caused by an α -particle hitting the circuit. [30]

Other external noise sources are *electromagnetic interference (EMI)* and *electrostatic discharge*. The errors are normally transient but in principle also permanent errors may result especially from electrostatic discharge. [12]

2.3 Fault Models

The actual defects on a circuit cannot be directly considered in the design and evaluation of the circuit and therefore special fault models are needed. Fault models are simplifications of the phenomena caused by defects on the circuit. The oldest and most commonly used model is the *stuck-at fault model*. The defects are modelled as a circuit node sorted to either power supply (stuck-at-1) or to ground (stuck-at-0). The modelling can be done at transistor level but most commonly gate level modelling is used, which means that according to the model a circuits input or output can be sorted to logic 1 or 0. Stuck-at fault model is very simple and so it is easy to use, and therefore it will be still used in the future, although other fault models will be more and more common than they are today. [1]

Bridging fault model models connections between nodes in the circuit. This is actually an extention to stuck-at fault model, where the connections were only to power supply or to ground, but bridging can occur also between two nodes or signal lines. The bridging fault model is expected to gain more importance when moving towards nanoscale circuits, because as the dimensions shrink the connection between two nodes is coming more probable. The bridging models used today though have to be enchanced to take into account the increasing parameter fluctuations in the nanoscale circuits [16]. A common way to detect bridging faults has been *I_{DDQ} testing*, which is based on the observation of the leakage current caused by bridging faults. The expected subthreshold leakages in the nanoscale circuits are causing growing problems to I_{DDQ} testing and therefore also new logic testing models are needed for bridging faults [1].

The importance of the *delay fault model* is expected to increase as the operating frequency rises. According to the model a fault occurs when the circuit is not able to produce the correct output in the specified time interval although it might operate correctly at a lower frequency. The delay fault models will consentrate in the future on path delays rather than gate delays because the relative delay of connections will become much higher than the delay of gates. [1]

The emerging technologies demand their own fault models. *Stuck-on/stuck-off fault model* has been suggested to detect background charge fluctuations in *single-electron transistors (SET)* as well as permanent defects due to manufacturing problems and transient device failures due to external influences. The error occurence is expected to be random [51]. Totally new fault models have also been suggested e.g. to *quantum-dot cellular automata (QCA)* [14].

3 Static Fault Tolerance

The circuit is said to be utilizing static fault tolerance when it is built in such a way that a fault somewhere in the circuit will not violate the correct operation of the circuit. The word static stands for the fact that fault tolerance is built into the system structure and it efficiently masks the fault effects. The method to create such fault masking properties is to use some kind of redundancy. Therefore the name static redundancy.

Static redundancy can be categorized to hardware, information and time redundandy according to the resource that is used to create the redundancy. Also a combination of these can be used. Such hybrid redundancy is discussed in Section 3.4.

The methods presented in the literature are most commonly designed for or demonstrated with single errors. As going further to nanoscale devices the defect density is expected to increase and so the scenario of multiple errors will be faced. Therefore methods capable of tolerating several failures are the main focus in this section. Previously also some parts of the design have been left out from the fault analysis on the basis that it is a minor part of the whole design and could therefore presumed to be faultfree. This is not the case for nanoscale devices and so a new look also to these componets has to be taken.

3.1 Hardware Redundancy

Hardware redundancy generally means multiplying the processing module and providing voting circuit to decide the correct output value based on redundant module outputs. Higher reliability is gained because when a redundant component fails, the voter can decide the correct output based on the results of other redundant modules. The basic principle can be used at many different abstracation levels, the modules can be as simple as single gates but also as complex as whole processors or even larger constructions. The voter can be a simple bitwise hardware implementation or software algorithm running on a processor. General to all hardware redundancy realizations is the need for extra space or chip area. Therefore the methodology is also called as physical, area or space redundancy.

The most common hardware redundancy realization is *triple modular redundancy (TMR)*, which consists of three redundant modules and a voting circuit (see Fig. 4). The voter normally performs majority voting, which means that the output is the same as the ouput of two-of-three of the modules. TMR is capable of masking a single error in processing modules. The weak point of the circuit is the voting circuitry and an error there could cause the whole circuit to fail. This has been tackled by multiplying also the voter three times and connecting the module outputs to all three voters [27]. Other possibility is to modify the voting circuitry in such a way that possible errors can be detected. Approaches include e.g. a voter that is on-line self-testing for internal faults [10, 43] and an I_{DDQ} checkable voter [8]. Another crucial design point is the synchronization of the voter input, and the

simplest method for realizing it is to insert registers to voter inputs [27]. Mismatch and crosstalk in lines from module outputs to voter inputs can cause severe misfunctions [17], which is especially important issue to handle because of the increasing crosstalk capacitance in nanoscale wires.

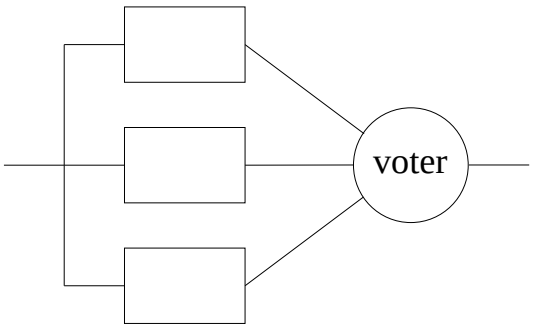


Figure 4: Triple modular redundancy.

A more generalized hardware redundancy realization is *n-modular redundancy (NMR)*, which means that there are n redundant modules and a voter. This structure is capable of detecting $\lfloor (n-1)/2 \rfloor$ errors in different processing modules. The most common structures besides TMR are 5- and 7-modular redundancies capable of detecting 2 and 3 errors respectively.

3.1.1 Voters

The voting algorithms can be divided according to their functionality to generic and hybird voting algorithms and to purpose-built voters. Generic voters use only the information of input signals to produce the output while hybrid voters have also some extra information such as the reliability of different modules or history of previous votings. The purpose-built voters are e.g. special microprocessor systems designed for space shuttles and they are not discussed here. [40]

Generic voters create output according to the present output values of redundant modules. The most common algorithm is the *exact majority voting*, which means that when the majority of the module outputs has the same value, this value is forwarded to output. This is easily achieved in bitwise voting because the only possible values are logic 0 and 1. If the module outputs are not just one bit wide but for instance integers, then it is possible that there is no majority agreement. In this case the voter can have a benign output ”no result”, which is an exception signal. The values of different modules can be slightly different because of noise or e.g. sensor elements cannot be physically at the exactly same place. Therefore *inexact majority voting* has been introduced, where the output is decided if the majority of the module outputs lie inside a certain *threshold*. A benign ”no result” is outputted only if majority of the outputs are more than a threshold apart from each other. The threshold effect can be easily achieved by dropping a couple

LSBs from the voting procedure [27]. The *plurality voting* means voting where there does not necessarily have to be majority having the same value (exact) or values in threshold limits (inexact) but only the number of module outputs having the same value or values in threshold apart is larger than the number of modules having an other value. E.g. if there are five modules it needs that two of them have the same value if the other three have all different values. In case of inexact voting the selection of the output value can be a random selection of one of the majority or plurality output values or it can be *mid-value selection (MVS)*, where the output is counted as the mid-value of the majority or plurality outputs. [31, 52]

Another voting scheme is *median voting*, which means the selection of the median of all the module outputs as the voter output. An efficient software realization is to sort the output values and then select $[(n+1)/2]$th value as the output, where n is the (odd) number of redundant modules. [35]

In *weighted average voting* every module output gets a weight and the output is counted as the average of the module outputs scaled by these weights. The output is scaled back by the sum of weights in order to produce output that is at the same scale as the inputs, and in the case of bitwise voting, the output is returned to one of the logic states according to a threshold value, which can also be adjusted somewhere else than the mid-point of logic states (see Fig. 5). The weights are counted based on the output value distance of the other output values. If an output value is far away from all the other output values, it is given a smaller weight than an output value that has many other output values near its value. Advanced methods to count these values have also been presented. These include e.g. a voter that has a soft threshold created according to distance to other values and it is controlled by a special roll-off parameter which gives the possibility to adjust the voter behaviour from majority voter to average voter [38] and a fuzzy voter, which uses fuzzy set theory to adjust the weights [36]. Circuit realizations of voters are e.g. a weighted bit-wise voters with threshold used with self-purging systems [11, 45] and an analog weighted average voter [50] together with threshold circuit using capasitive threshold logic (CTL) [49]. The adjustment of the threshold is a cruicial task for the operation of the circuit. Threshold can be static, based on circuit realization, it can be set after manufacture, or the threshold can be dynamic

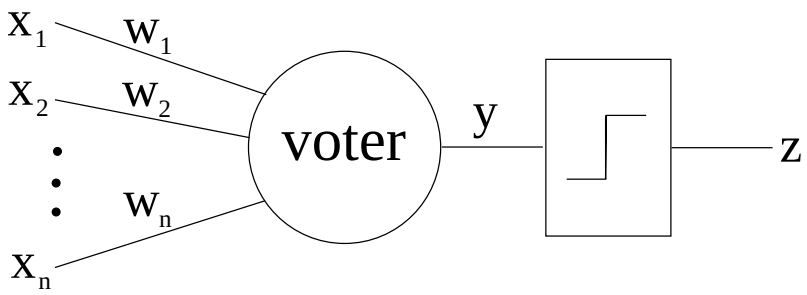


Figure 5: Weighted average voting with threshold: $y = \sum_{i=1}^{n} w_i x_i / \sum_{i=1}^{n} w_i$, $z = 1$ if $y \geq T$ and $z = 0$ if $y < T$, where T is the set threshold.

adjusting to the operation environment. E.g. the use of artificial neural network (ANN) learning algorithm in adjusting the thresholds has been suggested. [48]

Different voting schemes are more appropriate for some applications than the others. Majority voting is rather safe voting scheme because it is more likely to output a benign than a faulty result. The weighted average voting on the other hand results in more correct results but also the amount of incorrect results increases, though the safety is not that good [39]. The above mentioned is true when module outputs are wider than just one bit as for bitwise voting the safety of majority voter is lousy because in case of multiple errors the output is likely to produce incorrect error instead of bening result. The weighted average voting for bit-wise voters is shown to result in higher number of correct results in the presence of multiple defects. [51]

Hybrid voters combine the information of present module outputs and some other information regarding the module circuits or output sequence. The voting procedure can be totally based on the history, e.g. deciding the weights in weighted average voting based on history records [37] or choosing for the output of the voter the output of the module that has been best in the history. The best module is the one that has had output closer than a threshold to majority of the module outputs for the most times [34]. The use of history values can be also a backup system if no agreement can be found among the module outputs [34]. In addition to using the history data to detect the most reliable modules, it can also be used to predict the next output value. Many system outputs are somehow dependable of the previous outputs, which gives the legitimation for this procedure. One of the simplest way to predict the value is in the case of module output disagreement to check if any one of them is closer than a threshold to the previous voter output, and if such an output is found, to select for the voter output the module output that is closest to the previous voter output [39].

3.2 Time Redundancy

The basic principle in time redundancy is to use the same resource many times and compare the results gained from different rounds of computation. The method therefore saves area when not that much extra hardware is needed and at the same time uses more time, which might be acceptable for a certain type of applications. The method of repeating the same calculation many time is effective to detect transient errors but permanent and in many cases also intermittent errors occur at the same place during all calculations and cannot therefore be detected and corrected. This problem can be overcome by somehow encoding the operand before processing and decoding afterwards. Commonly first the operation is performed with the uncoded operand and secondly with coded ones.

The first presented coding method is *alternating logic*, which uses complementation as the coding method. In order to be able to use this coding, the self-duality of the circuit is required or possibly extra input is needed. The *recomputing*

with shifted operands (RESO) means shifting the operands before calculation to left and back to right after calculation. This method demands extra width to operation or cyclic shift can be used, which on the other hand means complex logic for carry signals on adder circuits. Another coding approach is *recomputing with swapped operands (RESWO)*, where upper and lower parts of the operands are swapped before calculation and back after it. The method needs no extra bits and the logic for handling carry bits in adder circuits is more straightforward than in RESO. [27]

The error correcting properties are gained by repeating the operation at least three times and performing voting for the three results. Different codings are used for different calculation rounds, e.g. no coding, shift 1 and shift 2. Bit-wise majority voting can be problematic because the arithmetic operations commonly affect many bits. The different voting approaches were already discussed in the previous section. [27]

3.3 Information Redundancy

Information redundancy in general means adding extra bits to stored or transmitted data. Special *error correcting codes (ECC)* are used to detect the exact location of an error, which makes it possible to correct it. The codes can be classified to *separable* and *nonseparable codes* according to the way how the extra information is added to the data. If the data is left as is and only extra bits are added for the correction, the code is separable. A nonseparable code needs a separate decoding circuit to return the data to its normal form before further processing.

A simple separable ECC is the *parity code*, which means adding one bit to every word and the value of this added bit is adjusted to make the number of bits with value 1 odd (odd-parity code) or even (even-parity code). When a parity is counted separately for both rows and columns for instance in a memory block, the exact location of a single error can be detected. [27, 33]

The most common ECC is the *Hamming code*, which uses the concept of overlapping parity where there are several parity bits and every data bit is part in adjusting several of them. The Hamming code fulfilling the rule $2^c \geq d + c + 1$, where c is the number of check bits and d is the number of data bits, corrects single errors (SEC) or it can be used to detect double errors (DED). The modified Hamming code for both correcting single errors and detecting double errors can be achieved by adding one extra check bit, which is used as the parity bit of the whole code word. [27, 33] Another example of an overlapping parity code is the *Hsiao code* [33]. Both the Hamming and Hsiao codes are separable as the parity bits are added to the code words without altering the data itself.

The *Dual rail (DR) code* is a coding method, where every bit is doubled and the result is a separable code. In addition a parity bit is introduced, thus the code needs $2k + 1$ bits, where k is the number of information bits. [47] Also an extended version of the dual rail code has been presented. In this version also the parity bit is doubled so $2(k + 1)$ bits are needed. [46] The DR code is

able to correct single bit errors as the Hamming code but the needed amount of check bits is much higher. The benefit of the DR code is its ability for crosstalk minimization, which is simply a result of the property that signal wire and its dublicate always have similar transactions. The DR and even more the extended DR performs better than the Hamming code when transmission delay or energy consumption is regarded in the presence of crosstalk noise and when the wires are placed optimally. In nanoscale circuits crosstalk capasitance is expected to be dominant compared to other wire capasitances and therefore the benefits of DR coding methods are extremly cruicial. [46, 47]

As the probability for multiple errors increases when going further into the nano regime, also error correcting codes capable of correcting several errors are needed. A popular ECC for multiple error correction is a cyclic code *Bose-Chaudhuri-Hocquenghem (BCH) code*. The class *cyclic code* means a code where cyclic shift of a codeword generates another codeword. Cyclic codes in their standard form are nonseparable but they can be easily modified to be also separable. The BCH code is similar to Hamming code when used as a single error correcting code, but it can be built to have the ability to correct as many simultaneous errors as needed. [55]

The *Reed-Solomon code* is another cyclic code that can be used to correct multiple errors. The code is nonbinary, which means that instead of bits, groups of m bits (e.g $m = 8$, a byte) are used as symbols for the code. If a word contains k groups of data and its totally length is n groups, at most $\lfloor (n-k)/2 \rfloor$ errors can be corrected. [33]

The methods mentioned above are mainly used to correct errors caused by noise in signal transmissions and upsets occured in memories. The *redundant residue number system (RRNS)* is used for error correction in arithmetic operations such as addition, substraction and multiplication. In residue number system each number is presented as the residues for a set of relatively prime moduli. E.g. if the moduli set is $\{3,5\}$ then a 9_{10} is presented as 04_{RNS} because 9=0 mod 3 and 9=4 mod 5. The set $\{3,5\}$ can be used to present numbers $0_{10} - 14_{10}$, which is determined by multiplying the moduli of the set ($3 \cdot 5 = 15$). The operations are performed bitwise as mod m_i, where m_i is the moduli used to create the numbers at this bit location. When extra moduli are added to the set, a redundant residue number system is gained, which can be used to detect and correct errors. If r redundant moduli are added, the system can correct up to $\lfloor r/2 \rfloor$ errors. The use of RRNS has been proposed to be used e.g. in digital filters and in software-defined radio. [15, 22, 27]

Another attempt to make arithmetic operations tolerable to multiple errors, is the use of *serialized data* and arithmetics based on *stochastic computing*. The main idea is to represent every value as the amount of 1's in a word and the error tolerance is gained by adding extra bits to the operands. The serialization increases remarkably the length of operands but the hardware needed to fulfill the operations is very light and therefore the approach can be used to build massive parallel structures. The results show small inaccuracies also for a rather high number of

errors and therefore the method is suitable e.g. for digital filters, where small deviations form the accurate values are accepted. [42]

The use of error correcting codes means that there is need for encoding, decoding and also sometimes for a special correction circuits. The fault tolerance of these circuits has not been a matter of concern previously and generally they have been presumed to be faultfree. As the defect density increases also these circuits need to be made fault tolerant and thus there is a demand for development of fault-tolerant, low-power, high-speed and area-efficient encoder and decoder circuits. [44]

3.4 Hybrid Approaches

Methods that combine the aspects of many different redundancy types are called hybrid approaches. The method to combine hardware redundancy or more specifically triple modular redundancy and time redundancy is called *time shared triple modular redundancy (TSTMR)*. In this approach there are three identical processing elements and a voting circuit just like in TMR. The time domain approach is inserted in a way that every operand are divided into three parts and also the width of processing elements is one third of the original ones (see Fig. 6).

The procedure starts by the operation for the lower parts of the operands, the result is voted among the three module outputs and saved to a register. Next the same is done to the middle parts and finally to upper parts of the operands. When all parts are calculated the results are combined to create the final result. Special logic is inserted to handle the carry propagation from one phase to another. The benefit of the method is the lower area overhead than in TMR and also the time

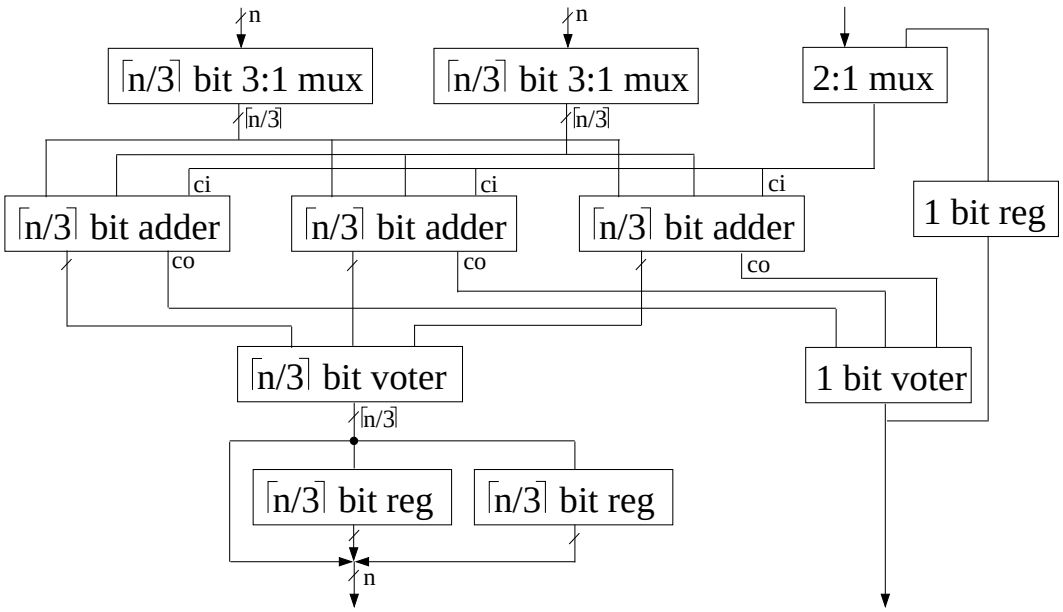


Figure 6: Time shared triple modular redundancy adder.

required for computation is normally less than three full width operations because of the smaller carry chains. The same method is also called *recomputing with triplication with voting (RETWV)* [23], *hardware partition in time redundancy (HPTR)* [2] or *recomputing with partitioning and voting (RWPV)* [3] and its usage is presented for adders and multipliers [23] as well as for dividers [20].

An extension to same methodology is *quadruple time redundancy (QTR)*, where operands are divided into four parts and the computation has four phases. The idea in this extension comes from the fact that the bit width of the operand is commonly dividable by four but seldom by three. [54]

4 Dynamic Fault Tolerance

The effect of dynamic redundancy is based on active actions as opposite to the passive operation of static redundancy. The dynamic redundancy operation can be divided into four phases the first of which is *fault detection*. After the detection of a fault situation the next thing to do is to locate the fault. Hence, the second phase is *fault location*. The third phase is *fault containment* which means isolating the error source so that no new errors can occur. The final phase is *fault recovery* meaning usually reconfiguration of the circuit so that the erroneous part has been disabled. [27]

The use of dynamic redundancy means introduction of special control circuitry and elements. The design of these control parts is not always that straightforward. The benefit gained with dynamic redundancy is better reliability especially in the occurence of permanent and multiple errors, and quite often the reliability is also gained with smaller area overhead than in the corresponding static redundancy approach.

4.1 Fault Detection

The fault detection is a cruical part of dynamic redundancy approaches. If the fault is not detected the circuit can produce erroneous outputs and regarded faultfree because the fault detection circuitry has not indicated the occurence of a fault. The fault detection can be organized as *periodic tests*, *self-checking circuits* or *watchdog timers* [33].

The purpose of the perioric tests is to stop the circuit operation every now and then and perform self-test. The method cannot guarantee that every fault is detected because the test is run only every now and then and also the time needed for testing is a drawback of this method.

Watchdog timers are used especially in multi-processor environment. A control circuit sets a timer when a processor starts to execute a certain job. At some predefined point of the procedure the processor resets the counter. If for some reason the processor halts during the operation, the control circuit detects it by observing the timer value exceeding some limit and can for instance reset the halted processor or start a reconfiguration process. [27]

There are numerous ways to create self-checking circuits. The most straightforward method is *duplication with comparison (DWC)*, which simply means creating two identical modules and comparing their outputs (see Fig. 7). Sometimes there can be small deviations between two components although they are operating correctly. In these cases the comparison process can ignore least significant bits (LSBs). The method can also be expanded to tolerate common mode failures which are failures that affect same way both modules. The expansion is to use complementary logic in the duplicated module and the comparison result is correct when the outputs are complements of each other. [27]

Dublication can be done also in time domain. The information is sent twice

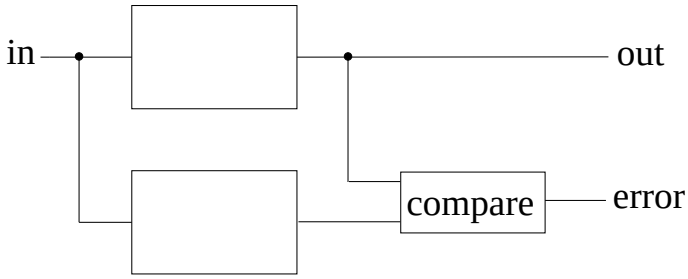


Figure 7: Dublication with comparison.

and the two results are compared. At the second time the data can be complemented or swapped and inverse transform proceeded at the output in order to gain better detection abilities. [27]

A combination of the use of both time domain and space is called *recomputing with dublication with comparision (REDWC)*. In this method the operands are divided into two blocks and the duplicated modules are only half the width of original blocks. First the lower parts of the operands are fed to the modules and the outputs are compared. This is followed by a similar process to upper parts of the operands. This method results in less area overhead than DWC but the time needed for single operation doubles. [27]

A number of codes can be used to detect errors in data. Possibly the most common *error detecting code* is the parity code, which is extensively used e.g. in memory circuits. A single parity bit is added to every word and the value is set in a way to make the number of logic ones in the word even or odd depending on whetheher even or odd parity code is used. The code can detect single errors but in case of multiple errors the parity bit may have correct value and therefore the error is not detected.

The Hamming code uses interleaved parity and it can be used to detect double errors (DED) or with extended Hamming code even triple errors (TED). The Hamming code is one of the most used methods in error detection and correction and it was already introduced as a error correcting code in Section 3.

The parity and Hamming coding methods have been used in adaptive error detection system, which is built to gain more energy efficient design without affecting the error detection capabilities. The system monitors the noise level of the trasmission channel and dynamically changes to a code that has better error detection capabilities in case of an increased noise level and respectively changes to a lighter code when the noise level is lower. In the design, parity, Hamming DED and extended Hamming TED codes are used. [41]

Errors are commonly expected to occur bitwise randomly. This is not a realistic model for all cases. In some situations errors only to one direction can occur, which means that 0 can become 1 but 1 cannot become 0 or vice versa. Errors occuring only to one direction are called *unidirectional* and errors that can occur to both directions are *bidirectional*. Errors also do not always occur alone but they

can occur as bursts. Bursts are common for instance in communication signalling caused by some disturbancies at the signaling media. Intermittent faults in the signaling medium typically cause error bursts.

M-out-of-n codes are able to detect not only all single errors but also multiple unidirectional errors. The basic principle in code is that every code word has m 1s and $n - m$ 0s while the total length of codewords is n. The code is not separable except only in special cases. Such a special case is *k-out-of-2k code* where there is two bits per one data bit. The code is called *k-pair two-rail code*, when the check bits are bitwise complement of the information bits [27, 33]. An example of the use of m-out-of-n codes is e.g. an on-line error detecting adder, where 1-out-of-3 code has been used [53].

The *Berger code* demands the fewest number of checkbits of the available separable codes for detecting arbitrary multiple unidirectional errors. The code is based on counting the number of 1s in the word and appending a complement of it to the word. The amount of checkbits is $\lceil \log_2(I+1) \rceil$, where I is the number of information bits [27, 33]. If it is not necessary to detect all the possible unidirectional erros, but only max t of them, then the amount of checkbits can be further decreased. Such codes are e.g. *modified Berger code*, *Borden code* and *Bose-Lin codes* [33]. A set of codes for detecting unidirectional errors and error bursts of length t, called *the unidirectional burst error detecting codes* includes e.g. codes by Berger, Bose, Blaum [33].

One way to detect an error occured during signal transmission is to count a *checksum* of the sent words and send it together with the words. At the receiving end the checksum is recalculated and compared with the received one. Examples of checksums are *single- and double precision checksums* as well as *Honeywell* and *residue checksums*. The length of the single-precision and residue checksum is the same as the width of the words and the others have the length of double the width of the words. The all listed checksums are calculated by summation of all words and they differ in the way the words are organized and carry bits handled in the summation process [27].

Cyclic codes are a set of codes that are able to detect single errors and adjacent multiple errors, which makes them extremely suitable for serial transfers to handle burst errors. The number of adjacent errors that can be detected is $n-k-1$, where k is the number of data bits and the coded word contains n bits. A generator polynomial of degree $n-k$ is used. Nonseparable but efficient circuit realizations can be made using *linear feedback shift registers (LFSR)*, and the codes can also be made separable by small changes in the generation process. [27]

Commonly used codes are the *cyclic redundancy check codes (CRC)*. For instance, CRC-8 (8 for the degree of the generator polynomial) is used in a self-calibrating design to detect the errors on the transmission channel. The self-calibration in this design means that the voltage-swing for transmission channel is scaled dynamically in order to obtain minimum energy consumption. [57]

Arithmetic codes are used to detect errors in arithmetic operations. One example of such a code is the AN *code* which can be used for addition and substraction.

Every operand is multiplied by A before operation and the result should be evenly dividable by A, otherwise an error has occured. A commonly used AN code is $3N$ code. Other arithmetic codes include *residue* and *inverse-residue codes*. In these codes, the operands are divided by integer m (e.g. 3 for mod-3 residue), and the remainder is appended to the data, thus the code is separable. Operation for the residue is proceeded module m and it is checked against the remainder of the operation result when divided by m. The procedure is eligible for addition, multiplication and ALUs [33]. Also residue number systems can be used for error detection. They were described in Section 3 along with their error correcting capabilities. [27]

4.1.1 Checkers

Checkers are circuits that are used to determine if an error has occured or not. A checker commonly compares two values and signals an error if they differ or equal (depending on the used method). The accurate design of checkers is very important because an error in the checker circuit may invalidate the fault tolerance of the whole system.

In the design of checker circuits the concepts of *fault-secure* and *self-testing* design are commonly used. The circuit is fault-secure if every valid input produce either correct output or a non-code output, which can be easily observed. The self-testing on the other hand means a circuit where for every fault (in some set of faults) there is an input combination, which produces an non-code output so that the fault can be observed. The circuit that is both fault-secure and self-testing is called *totally self-checking (TSC)*. [33]

The fault-security in a circuit can be achieved by designing the circuit in a way that there is separate logic for separate outputs. This ensures that a single error affects only one output signal. The self-testability is achieved by non-redundant design.

A common checker is a two-rail checker, which has four inputs and two outputs. The inputs consist of two input pairs, and a pair consist of two complementary signals. The circuit checks that the signals of an input pair are indeed complementary. The output signals are complementary when there are no errors and in the case of an error in the input or in the checker circuit the output signals are the same. A checker for a larger amount of input pairs can be created by forming a tree connection from the two-pair checkers. A totally self-checking two-rail checkers have been presented, and a combination of these checkers forming a larger checker is also TSC. [33]

4.2 Fault Location

After the detection of a fault situation the fault source has to be located. One way to accomplish this is to start a specific self-diagnostic procedure after fault detection [27]. The self-diagnostics may consume too much time to be used in

some time-critical applications but for the most applications the extra time can be afforded because it is needed only in the case of a fault occurence which is reasonably seldom.

Another way to locate the fault is to use two different detection methods. For example, if both duplication with comparison and parity check are used, the faulty module can be easily located. [4]

4.3 Fault Recovery

There are two common ways to achieve the fault recovery. The transmission or calculation repetition, the so called *automatic repeat request (ARQ)* uses extra time while the use of *spare modules* and *reconfiguration* causes area overhead.

The ARQ is suitable for recovering from transient and in some cases also intermittent errors. Against permanent errors it cannot be used, because the system will not work regardless of how many times the operation is repeated. The goodness of ARQ has been evaluated against forward error correction (FEC), which means error correcting codes. ARQ is found to be more energy efficient (0.25μm technology), because of the energy consumption of codec components. Thus, the gap between ECC approaches is expexted to decrease as technology is scaled down because of the larger power consumption of the communication channel and decreasing power need for logic blocks. [7]

The use of spare modules and reconfiguration is especially suitable for the recovery from permanent and intermittent errors. On the other hand, it is not very efficient to abandon a whole module and replace it with a spare one if the error is only temporary. Therefore a combination of ARQ and reconfiguration might give the best result. The operation is first repeated and if the error stays, then reconfiguration process is begun. [33]

The spare modules can be divided into *hot and cold spares*, which indicates whether the spare modules are immediately ready to use (hot) or do they need to be initialized before usage (cold). The cold spares are also called *standby spares* and a hot spare system *reconfigurable duplication*. The standby spare system leads to higher reliability but the reconfigurable dublication has higher safety. [27, 33]

The use of spares has been also combined with N-modular redundancy. In these systems N modules are part of the voting procedure and in addition to them, there are spare modules (see Fig. 8). After voting the output is compared to every module's output and in the case of a disagreement, the module is replaced with a spare. The combination leads to higher reliablity than the normal NMR system. [27, 33]

A *self-purging system* is a normal NMR module system, but in the case of a module failure the system is self-reconfigured and the result is (N-1)MR system. The isolation of modules is done in the similar way as in NMR with spares or it can be based on evaluation of the distancies between different module outputs. This is called the *shift-out modular redundancy*. In connection with self-purging systems the use of weighted average voting with dynamically adjustable threshold

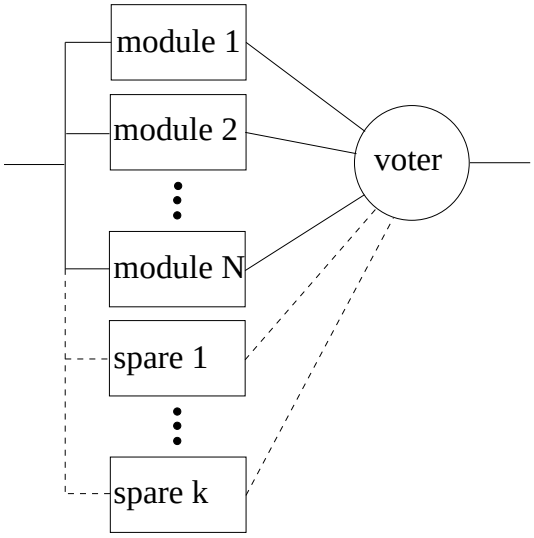


Figure 8: N-modular redundancy with k spares.

is suggested. [11, 27, 45]

Still another combination of dynamic redundancy and static NMR is the *triple-duplex architecture*. In this combination every module is dublicated and in the case of a disagreement one of these two modules it is isolated from the voting unit. One system therefore consist of six modules and a voter unit (see Fig. 9). [27]

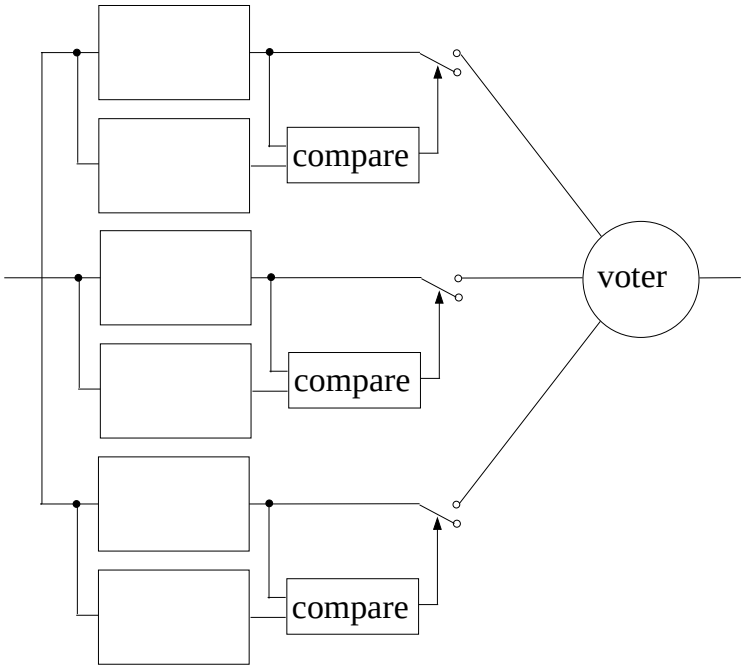


Figure 9: Triple-duplex architecture.

The addition of spare modules to system causes easily a large area overhead, because for every different kind of module a special spare has to be inserted. If the modules are similar (but not identical) the *heterogenous redundancy* can be used. Redundant blocks that can be programmed FPGA-like to execute many different functions are inserted to circuit. One such module can be used to replace different kinds of modules and therefore the overall amount of spare modules can be decreased. The disadvantage is that these programmable modules are generally slower than fixed-logic blocks and also the reconfiguration takes more time. [32]

4.4 Reconfigurable Arrays

The fabrication of custom circuits using future nano technologies is expected to be very hard or even impossible but fabrication of regular array structures such as two-dimensional crossbars have already been succesful. Therefore it is very likely that many future nano circuits will be programmable arrays. [24]

A reconfigurable array can be made fault-tolerant by adding redundant rows and/or columns to the circuit and provide it with mechansim for reconfiguration. The reconfiguration can be performed off-line e.g. right after fabrication or at compile-time or it can be done on-line during operation. Strategies for off-line reconfiguration are e.g. *rippling replacement*, where an extra column is inserted to design and if an errorneous node is detected it is replaced with its following neighbour node which is again replaced by its following neighbour and so on, and *stealing strategy*, which consists of one extra column and one extra row and is an extension to rippling strategy overcoming the limit of only one tolerable error per row because the replacement can now be found from either next column or row. Still another off-line reconfiguration strategy is the *repair most replacement strategy*, in which many spare rows/columns are inserted and whole row/column is replaced in case of a detected error. [27]

Real-time reconfiguration is based on programmable switches. *Successive row/column elimination (SRE/SCE)* strategies simply pass one row or column with the help of these switches and at the same time provide test input and output to the eliminated row/column for further diagnostics. A combination of both row and column elimination is *alternate row and column elimination (ARCE)*, in which rows and columns are eliminated in an alternating fashion. So if the previous fault was handled by row elimination the next one will be handled by column elimination. ARCE gives better tolerance to multiple errors than SRE or SCE. [27]

Also the use of time redundancy has been presented for array structures. The approach is simply to use some elements many times in case some cells are errorneous in a FFT array. For the needed routing spare links are introduced in the circuit structure. [5]

5 Conclusions

The probability for fault occurences is growing as moving towards nano regime. In Section 2 possible fault sources were discussed and the faults were classified two three categories. The most common error sources and their classification is presented in Table 1. From the table it is easy to see that permanent and intermittent errors originate mainly from manufacture process and from physical changes during operation while internal and external noise are main sources of transient errors.

Table 1: The most common error sources for different error types.

	Error type		
Error source	Permanent	Intermittent	Transient
Manufacture process	*spot defects* *bridging faults* *metal slivers*	*gate length deviations* *doping profile fluctuations* *resistive contacts* *resistive vias* *metal slivers* *metal cracks*	
Physical changes during operation	*electromigration* *current tunneling in gate oxides resulting in breakdown*	*electromigration* *soft breakdown*	
Internal noise		*crosstalk* *skin effect* *timing noise*	*crosstalk* *skin effect* *timing noise*
External noise	*electrostatic discharge*		*radiation* *EMI* *electrostatic discharge*

Different fault tolerance methods fit better to one kind of errors than for others. Static redundancy methods were discussed in Section 3 and dynamic redundancy methods in Section 4. The most important ones of the presented methods are listed in Table 2 under the error type they fit best.

Table 2: The most suitable methods to cope with different error types.

	Error type		
Method	Permanent	Intermittent	Transient
Hardware redundancy	*TMR/NMR weighted average, median voting*	*TMR/NMR plurality, median voting*	*TMR/NMR majority voting*
Time redundancy		*RESO* *RESWO*	*repeated operation* *alternating logic* *RESO* *RESWO*
Information redundancy		*Reed-Solomon code*	*parity code* *Hamming code* *Dual-rail code* *BCH code* *Reed-Solomon code* *RRNS*
Hybrid approaches	*TSTMR* *QTR*	*TSTMR* *QTR*	*TSTMR* *QTR*
Dynamic redundancy	*standby spares* *reconfigurable dublication* *NMR with spares* *self-purging system*	*standby spares* *reconfigurable dublication* *NMR with spares* *self-purging system*	*ARQ* *triple-dublex architecture*

From the table it can be seen that some kind of hardware redundancy is always needed to tolerate permanent errors. The dynamic spare approaches are

better than static ones but also static hardware redundancy is a possible solution. The weighted average voting is preferred because it has the ability of minimizing the effect of a module that is constantly operating erroneosuly and thus provides tolerance also for further errors.

The transients on the other hand can be best tolerated by the means of static information redundancy or by dynamic methods using ARQ. Also other static approaches can be used but they commonly result in larger area or time overhead.

The intermittent errors are probably the most difficult ones to deal with. The error may occur very seldom and be limited to a certain node in the system and thus the methods that are eligible for transients can be used for intermittents too. The intermittent errors can also be very frequent and hard to be limited, in which case it is best to handle them as permanent errors.

Although many methods are listed under two or even three error types, they cannot be regarded as the best or universal choices. Many of them either loose their abilities for fault tolerance in case of multiple errors, which will be common in nanoscale systems, or even if they could be expanded to tolerate multiple errors, the area and/or time overhead would be unacceptably high.

As a conclusion it can be stated that no single method is good for all kinds of errors. Hence the best fault tolerance can be gained by utilizing a variety of different fault tolerance methods. The system could e.g. use ECC to tolerate transient errors, and if some error permanently stays or repeats itself frequently (intermittent), the module can be replaced by a spare module. Controlling when to proceed with different operations will be the cruicial part of such systems.

References

[1] Robert C. Aitken: *Nanometer technology effects on fault models for IC testing*, Computer, November 1999, Volume: 32, Issue: 11, pages: 46 - 51, ISSN: 0018-9162.

[2] Sami A. Al-Arian, Mehmet B. Gumusel: *HPTR: Hardware partition in time redundancy technique for fault tolerance*, Proceedings of the IEEE Southeastcon '92, 12-15 April 1992, pages: 630-633 vol.2.

[3] Hussain Al-Asaad, Edward Czeck: *Concurrent error correction in iterative circuits by recomputing with partitioning and voting*, Digest of Papers of the Eleventh Annual 1993 IEEE VLSI Test Symposium, 6-8 April 1993, pages: 174-177.

[4] Sobeeh Almukhaizim, Yiorgos Makris: *Fault tolerant design of combinational and sequential logic based on a parity check code*, Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 3-5 November 2003, pages: 563 - 570, ISSN: 1063-6722.

[5] A. Antola, R. Negrini, M.G. Sami, N. Scarabottolo: *Fault-tolerance in FFT arrays: time-redundancy approaches*, Conference Record. of IEEE International Conference on Communications, 16-19 April 1990, pages: 779-785 vol.3.

[6] Roman Barsky, Israel A. Wagner: *Reliability and yield: a joint defect-oriented approach*, Proceedings of the 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2004. 10-13 October 2004, pages: 2 - 10, ISSN: 1550-5774.

[7] Davide Bertozzi, Luca Benini, Giovanni De Micheli: *Low power error resilient encoding for on-chip data buses*, Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, 4-8 March 2002, pages: 102 - 109.

[8] Alessandro Bogliolo, Michele Favalli, Maurizio Damiani: *Enabling testability of fault-tolerant circuits by means of IDDQ-checkable voters*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, August 2000, Volume: 8, Issue: 4, pages: 415 - 419, ISSN: 1063-8210.

[9] Melvin A. Breuer, Sandeep K. Gupta, T. M. Mak: *Defect and error tolerance in the presence of massive numbers of defects*, IEEE Design & Test of Computers, May-June 2004, Volume: 21, Issue: 3, pages: 216 - 227, ISSN: 0740-7475.

[10] José Manuel Cazeaux, Daniele Rossi, Cecilia Metra: *New high speed CMOS self-checking voter*, Proceedings of the 10th IEEE International On-Line Testing Symposium, 2004, 12-14 July 2004, pages: 58 - 63.

[11] C. W. Chiou, T. C. Yang: *Self-purging redundancy with adjustable threshold for tolerating multiple module failures*, Electronics Letters, 25 May 1995, Volume: 31, Issue: 11, pages: 930 - 931, ISSN: 0013-5194.

[12] Cristian Constantinescu: *Trends and challenges in VLSI circuit reliability*, IEEE Micro, Volume: 23, Issue: 4, July-Aug. 2003, pages: 14-19, ISSN: 0272-1732.

[13] Michel Cote, Philippe Hurat, Mike Rieger, Alexander Miloslavsky, Denis Goinard: *How can design for manufacturing improve mask cost and yield?*, IEEE/CPMT/SEMI 29th International Electronics Manufacturing Technology Symposium, July 14-16, 2004, pages: 273 - 276, ISSN: 1089-8190.

[14] Timothy J. Dysart, Peter M. Kogge: *Strategy and prototype tool for doing fault modeling in a nano-technology*, Third IEEE Conference on Nanotechnology, 12-14 August 2003, Volume: 1, pages: 356 - 359.

[15] Mark H. Etzel, W. Kenneth Jenkins: *Redundant residue number systems for error detection and correction in digital filters*, IEEE Transactions on Acoustics, Speech, and Signal Processing, October 1980, Volume: 28, Issue: 5, pages: 538 - 545, ISSN: 0096-3518.

[16] Michele Favalli, Marcello Dalpasso: *Bridging fault modeling and simulation for deep submicron CMOS ICs*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, August 2002, Volume: 21, Issue: 8, pages: 941 - 953, ISSN: 0278-0070.

[17] Michele Favalli, Cecilia Metra: *TMR voting in the presence of crosstalk faults at the voter inputs*, IEEE Transactions on Reliability, Sept. 2004, Volume: 53, Issue: 3, pages: 342 - 348, ISSN: 0018-9529.

[18] John Ferguson: *Shifting methods: adopting a design for manufacture flow*, Proceedings of the 5th International Symposium on Quality Electronic Design, 2004, pages: 171 - 175.

[19] David J. Frank, Robert H. Dennard, Edward Nowak, Paul M. Solomon, Yuan Taur, Hon-Sum Philip Wong: *Device scaling limits of Si MOSFETs and their application dependencies*, In Emerging Nanoelectronics, volume 1, edited by Adrian M. Ionescu and Kaustav Banerjee, Kluwer Academic Publishers, 2005, ISBN 1-4020-7533-2.

[20] W. Lynn Gallagher, Earl E. Swartzlander, Jr.: *Fault-tolerant Newton-Raphson and Goldschmidt dividers using time shared TMR*, IEEE Transactions on Computers, June 2000, Volume: 49, Issue: 6, pages: 588-595, ISSN: 0018-9340.

[21] Charles Hawkins, Ali Keshavarzi, Jaume Segura: *View from the bottom: nanometer technology AC parametric failures - why, where, and how to detect*, Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 3-5 Nov. 2003, pages: 267-276, ISSN: 1063-6722.

[22] Makoto Honda, Hiroshi Harada, Masayuki Fujise: *Design of fault-tolerant digital filters based on redundant residue number arithmetic for over-the-air reconfiguration in software radio communication systems*, IEEE 55th Vehicular Technology Conference, 6-9 May 2002, Volume: 1, pages: 280 - 284.

[23] Yuang-Ming Hsu, Earl E. Swartzlander, Jr.: *VLSI concurrent error correcting adders and multipliers*, The IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, 27-29 Oct. 1993, pages: 287-294.

[24] Jing Huang, Mehdi B. Tahoori, Fabrizio Lombardi: *On the defect tolerance of nano-scale two-dimensional crossbars*, Proceedings of the 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 10-13 October 2004, pages: 96 - 104, ISSN: 1550-5774.

[25] James A. Hutchby, George I. Bourianoff, Victor V. Zhirnov, Joe E. Brewer: *Extending the road beyond CMOS*, In Emerging Nanoelectronics, volume 1, edited by Adrian M. Ionescu and Kaustav Banerjee, Kluwer Academic Publishers, 2005, ISBN 1-4020-7533-2.

[26] *International Technology Roadmap for Semiconductors 2004*, (http://public.itrs.net).

[27] Barry W. Johnson: *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley Publishing Company, 1989, ISBN 0-201-07570-9

[28] Barry W. Johnson, James H. Aylor, Haytman H. Hana: *Efficient use of time and hardware redundancy for concurrent error detection in a 32-bit VLSI adder*, IEEE Journal of Solid-State Circuits, Feb. 1988, Volume: 23, Issue: 1, pages: 208-215, ISSN: 0018-9200.

[29] Barry W. Johnson: *Reliability and fault tolerance issues in intelligent computing systems*, Proceedings of the 5th IEEE International Symposium on Intelligent Control, 5-7 Sept. 1990, pages: 267-272 vol.1.

[30] Tanay Karnik, Peter Hazucha, Jagdish Patel: *Characterization of soft errors caused by single event upsets in CMOS processes*, IEEE Transactions on Dependable and Secure Computing, April-June 2004, Volume: 1, Issue: 2, pages: 128 - 143, ISSN: 1545-5971.

[31] M. D. Krstic, M. K. Stojcev, G. Lj. Djordjevic, I. D. Andrejic: *A mid-value select voter*, Microelectronics and Reliability, Volume 45, Issues 3-4, March-April 2005, pages 733-738.

[32] Vinu Vijay Kumar, John Lach: *Heterogeneous redundancy for fault and defect tolerance with complexity independent area overhead*, Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 3-5 November 2003, pages: 571 - 578, ISSN: 1063-6722.

[33] Parag K. Lala: *Self-Checking and Fault-Tolerant Digital Design*, Morgan Kaufmann Publishers, 2001, ISBN 0-12-434370-8.

[34] G. Latif-Shabgahi, J. M. Bass, S. Bennett: *Component-oriented voter model for dependable control applications*, Microprocessors and Microsystems, Volume 25, Issue 3, 30 May 2001, pages 167-176.

[35] G. Latif-Shabgahi, J. M. Bass, S. Bennett: *Efficient implementation of inexact majority and median voters*, Electronics Letters, 20 July 2000, Volume: 36 , Issue: 15, pages: 1326 - 1328, ISSN: 0013-5194.

[36] G. Latif-Shabgahi, A. J. Hirst: *A fuzzy voting scheme for hardware and software fault tolerant systems*, Fuzzy Sets and Systems, Volume 150, Issue 3, 16 March 2005, pages 579-598.

[37] G. Latif-Shabgahi, J. M. Bass, S. Bennett: *History-based weighted average voter: a novel software voting algorithm for fault-tolerant computer systems*, Proceedings of the Ninth Euromicro Workshop on Parallel and Distributed Processing, 7-9 February 2001, pages:402 - 409.

[38] G. R. Latif-Shabgahi: *A novel algorithm for weighted average voting used in fault tolerant computing systems*, Microprocessors and Microsystems, Volume 28, Issue 7, 1 September 2004, pages 357-361.

[39] G. Latif-Shabgahi, S. Bennett, J. M. Bass: *Smoothing voter: a novel voting algorithm for handling multiple errors in fault-tolerant control systems*, Microprocessors and Microsystems, Volume 27, Issue 7, 1 August 2003, pages 303-313.

[40] G. Latif-Shabgahi, Julian M. Bass, Stuart Bennett: *A taxonomy for software voting algorithms used in safety-critical systems*, IEEE Transactions on Reliability, September 2004, Volume: 53, Issue: 3, pages: 319 - 328, ISSN: 0018-9529.

[41] Lin Li, N. Vijaykrishnan, Mahmut Kandemir, Mary Jane Irwin: *Adaptive error protection for energy efficiency*, International Conference on Computer Aided Design, 9-13 November 2003, pages: 2 - 7.

[42] C. A. L. Lisbôa, L. Carro: *Arithmetic operators robust to multiple simultaneous upsets*, Proceedings of the 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 10-13 October 2004, pages: 289 - 297, ISSN: 1550-5774.

[43] Cecilia Metra, Michele Favalli, Bruno Riccò: *Compact and low power on-line self-testing voting scheme*, Proceedings of the 1997 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 1997, 20-22 October 1997, pages: 137 - 145.

[44] Stanislaw J. Piestrak, Abbas Dandache, Fabrice Monteiro: *Design of fault-secure encoders for a class of systematic error correcting codes*, Proceedings of the 2001 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 24-26 October 2001, pages: 314 - 319.

[45] J. M. Quintana, M. J. Avedillo, E. Rodríguez-Villegas, A. Rueda: *Efficient νMOS realization of threshold voters for self-purging redundancy*, Proceedings of the 13th Symposium on Integrated Circuits and Systems Design, 18-24 September 2000, pages: 321 - 326.

[46] D. Rossi, S. Cavallotti, C. Metra: *Error correcting codes for crosstalk effect minimization*, Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 3-5 November 2003, pages: 257 - 264, ISSN: 1063-6722.

[47] Daniele Rossi, Cecilia Metra, André K. Nieuwland, Atul Katoch: *Exploiting ECC redundancy to minimize crosstalk impact*, IEEE Design & Test of Computers, January 2005, Volume: 22, Issue: 1, pages: 59 - 70, ISSN: 0740-7475.

[48] Alexandre Schmid, Yusuf Leblebici: *A highly fault tolerant PLA architecture for failure-prone nanometer CMOS and novel quantum device technologies*, Proceedings of the 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 10-13 October 2004, pages: 39 - 47, ISSN: 1550-5774.

[49] A. Schmid, Y. Leblebici: *Realisation of multiple-valued functions using the capacitive threshold logic gate*, IEEE Proceedings - Computers and Digital Techniques, 18 November 2004, Volume: 151, Issue: 6, pages: 435 - 447, ISSN: 1350-2387.

[50] Alexandre Schmid, Yusuf Leblebici: *Regular array of nanometer-scale devices performing logic operations with fault-tolerance capability*, 4th IEEE Conference on Nanotechnology, 16-19 August 2004, pages: 399 - 401.

[51] Alexandre Schmid, Yusuf Leblebici: *Robust circuit and system design methodologies for nanometer-scale devices and single-electron transistors*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, November 2004, Volume: 12, Issue: 11, pages: 1156 - 1166, ISSN: 1063-8210.

[52] M. K. Stojcev, G. Lj. Djordjevic, M. D. Krstic: *A hardware mid-value select voter architecture*, Microelectronics Journal, Volume 32, Issue 2, February 2001, pages 149-162.

[53] Whitney J. Townsend, Jacob A. Abraham, Parag K. Lala: *On-line error detecting constant delay adder*, 9th IEEE On-Line Testing Symposium, 7-9 July 2003, pages: 17 - 22.

[54] Whitney J. Townsend, Jacob A. Abraham, Earl E. Swartzlander, Jr.: *Quadruple time redundancy adders*, Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 3-5 Nov. 2003, pages: 250-256, ISSN: 1063-6722.

[55] Richard B. Wells: *Applied Coding and Information Theory for Engineers*, Prentice Hall, Inc., 1999, ISBN 0-13-961327-7.

[56] Hon-Sum Philip Wong, David J. Frank, Paul M. Solomon, Clement H. J. Wann, Jeffrey J. Welser: *Nanoscale CMOS*, In Emerging Nanoelectronics, volume 1, edited by Adrian M. Ionescu and Kaustav Banerjee, Kluwer Academic Publishers, 2005, ISBN 1-4020-7533-2.

[57] Frédéric Worm, Paolo Ienne, Patrick Thiran, Giovanni De Micheli: *A robust self-calibrating transmission scheme for on-chip networks*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, January 2005, Volume: 13, Issue: 1, pages: 126 - 139, ISSN: 1063-8210.

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 952-12-1596-8
ISSN 1239-1891