A Dynamic Fault Classification Scheme

Bernhard Fechner
Bernhard.Fechner@fernuni-hagen.de
FernUniversität in Hagen
Universitätsstraße 1
58084 Hagen

Tel.: +49 2331-987-4414

Fax.: +49 2331-987-308

ABSTRACT:

In this paper, we introduce a novel and simple fault rate classification scheme in hardware. It is based on the well-known threshold scheme, counting ticks between faults. The innovation is to introduce variable threshold values for the classification of fault rates and a fixed threshold for permanent faults. In combination with field data obtained from 9728 processors of a SGI Altix 4700 computing system, a proposal for the frequency-over-time behavior of faults results, experimentally justifying the assumption of dynamic and fixed threshold values. A pattern matching classifies the fault rate behavior over time. From the behavior a prediction is made. Software simulations show that fault rates can be forecast with 98 % accuracy. The scheme is able to adapt to and diagnose sudden changes of the fault rate, e.g. a spacecraft passing a radiation emitting celestial body. By using this scheme, fault-coverage and performance can be dynamically adjusted during runtime. For validation, the scheme is implemented by using different design styles, namely Field Programmable Gate Arrays (FPGAs) and standard-cells. Different design styles were chosen to cover different economic demands. From the implementation, characteristics like the length of the critical path, capacity and area consumption result.

1 INTRODUCTION

The performance requirements of modern microprocessors have increased proportionally to their growing number of applications. This led to an increase of clock frequencies up to 4.7 GHz (2007) and to an integration density of less than 45 nm (2007). The Semiconductor Industry Association roadmap forecasts a minimum feature size of 14 nm [6] until 2020. Below 90 nm a serious issue occurs at sea level, before only known from aerospace applications [1]: the increasing probability that neutrons cause Single-Event Upsets in memory elements [1][3]. The measurable radiation on sea level consists of up to 92 % neutrons from outer space [2]. The peak value is 14400 neutrons/cm^2/h [4]. Figure 1 shows the number of neutron impacts per hour per square centimeter for Kiel, Germany (data from [5]).

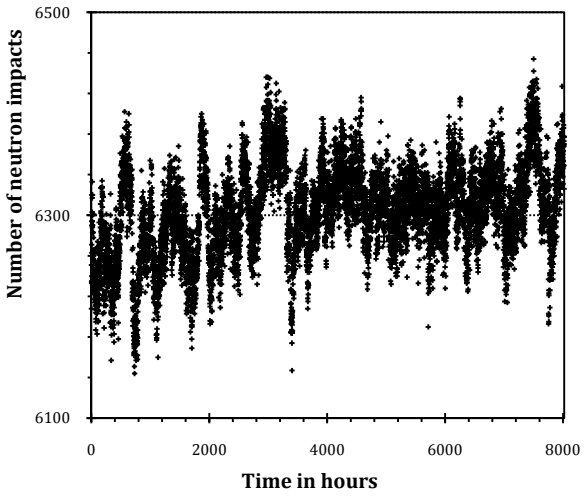

Figure 1: Number of neutron impacts per hour for Kiel, Germany (2007)

This work deals with the handling of faults after they have been detected (*fault diagnosis*). We do not use the fault rate for the classification of fault types such as transient, intermittent or permanent faults. We classify the current fault rate and forecast its development. On this basis the performance and fault coverage can be dynamically adjusted during runtime. We call this scheme *History Voting*.

The rest of this work is organized as follows: in Section 2, we present and discuss observations on fault rates in real-life systems. In Section 3, we discuss related work. Section 4 introduces History Voting. We seamlessly extend the scheme to support multiple operating units used in multicore or multithreaded systems. The scheme was modeled in software and its behavior simulated under the influence of faults. Section 5 presents experimental results and resource demands from standard-cell and FPGA (Field Programmable Gate Array)-implementations. Section 6 concludes the paper.

2 OBSERVATIONS

Figure 2 shows the number of transient single bit errors (x-axis) for 193 systems and the number of systems which shows faulty behavior (y-axis) over 16 months [7]. For many systems the number of faults is small. Few systems encounter an increased fault rate. From this, intermittent or permanent faults can be concluded.

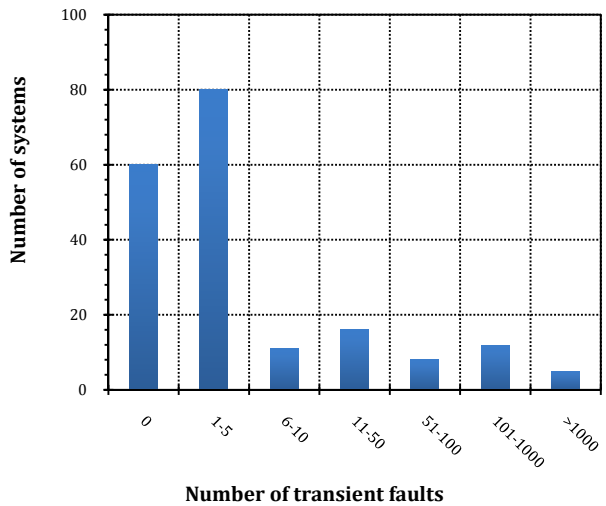

Figure 2: Number of transient errors for 193 systems

Figure 3 shows the daily number of transient single bit memory errors for a single system [7]. The faults within the first seven months were identified as transient. The first burst of faults appears at the beginning of month eleven, leading to the assessment of intermittent faults.

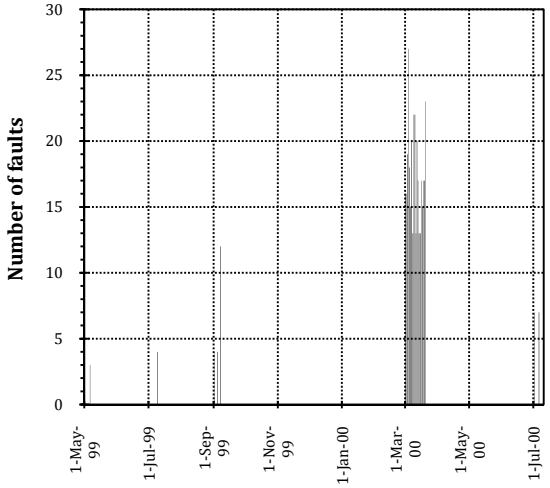

Figure 3: Number of daily transient single bit errors of a single system

Actual data is depicted in Figure 4. Here, the number of detected faults (y-axis) in the main memory and caches of all 19 partitions for the SGI Altix 4700 installation at the Leibniz computing center (Technical University of Munich) [8] is shown. The data was gained from the system abstraction layer (*salinfo*). In the observation interval (24.07.-31.08.2007, x-axis) two permanent faults in Partition 3 and 13 can be recognized, since we have a massive increase of errors (y-axis). If permanent faults would be concluded from a history, they would be recognized too late, leading to a massive occurrence of side-effects. For the detection of permanent faults, the fault rate in the observation interval alone is relevant. In partitions 2, 5, 9, 10 and 14, an increase of the fault rate before a massive fault appearance can be depicted. As much as a sudden growth, a decrease of the fault rate can be observed.

From [7] further properties of intermittent faults can be derived which help to classify these faults: a repeated manifestation at the same location and that they occur in bursts. Naturally, intermittent and permanent faults and can be recovered by replacing the faulty component.

3 RELATED WORK

A state-comparator detects faults in a duplex-system. For three or more results or states a majority voter is used for the selection of a correct state, being able to mask a fault. Besides majority voting many other selection criteria exist [17]. The classification of faults from their frequency was done in early IBM mainframes. One example is the automated fault diagnosis in the IBM 3081 [14]. In [10] an analysis of faults over time is used to forecast and separate permanent from intermittent faults. The ES/9000 Series, model 900 [15] implements a retry and threshold mechanism to tolerate transient faults and classify fault types. In [12] the fault rate is used to construct groups of faults. Detailed logs from IBM 3081 and CYPER-systems help to detect similarities and permanent faults. In [16] the history of detected faults within a NMR-system is used to identify the faultiest module. The offline dispersion frame mechanism by Lin and Siewiorek [11] diagnoses faults in a Unix-type filesystem. The heuristic is based on the observation of faults over time. Different rules are derived, e.g. the two-in-one rule which generates a warning if two faults occur within an hour. Similar is the approach from Mongardi [13]. Here, two errors in two consecutive cycles within a unit lead to the interpretation of a permanent fault. In [9] the *α-count* mechanism is developed. It permits to model the above-named and additional variants. Faults are weighted. A greater weight is assigned to recent faults. Additionally, a unit is weighted with the value α. Faults are classified, but intermittent and permanent faults are not differentiated. Latif-Shabgahi and Bennett [16] develop and analyze a flexible majority voter for TMR-systems. From the history of faults the most reliable module is determined. In [18] the processor that will probably fail

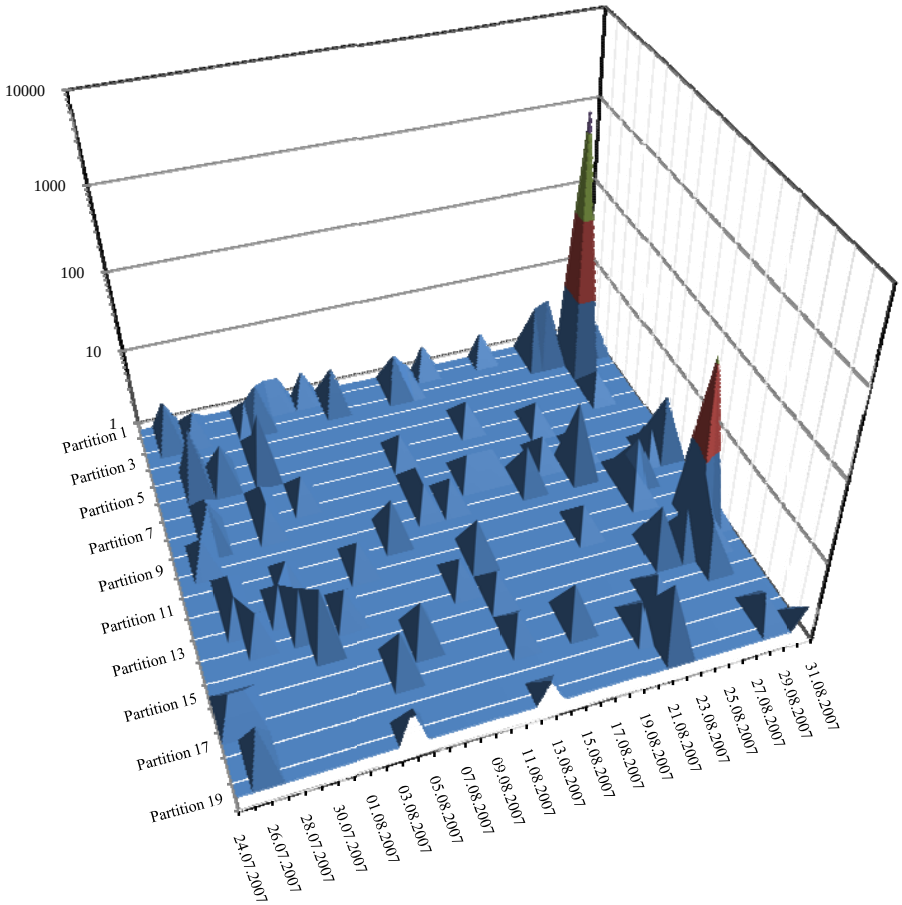

Figure 4: Field data for the SGI Altix 4700

in the future is determined from the list of processors which took part in a redundant execution scheme. Therefore, processors are assigned weights. Like [9] we separate between techniques which incur interference from outside and mechanisms based on algorithms. The latter are used in this work.

4 DIAGNOSIS AND PREDICTION

From the mentioned work and the observations, we can conclude that a classification of faults is possible by measuring the time between them. The limits for a classification are fluent, since the application, technology and circuit will determine the frequency of faults. History Voting for redundant (structural and temporal) systems classifies fault rates during runtime. It predicts if the fault rate will increase. Based on this forecast, the performance of a system can be dynamically adjusted if the prediction has a certain quality.

We hereby exclude the possibility that the system performance is decreased unreasonable due to a false prediction. A trust γ is assigned to units. These can be e.g. the components of a NMR-system, the cores in a multicore system or threading units in a multithreaded system. If the trust is zero, a faulty unit can be identified. An additional innovation is to dynamically adjust the threshold values to the fault scenario. From this we can exclude a false detection of a permanent fault on an unexpected high fault rate, e. g. the bypass flight of a space probe on a radiation emitting celestial body.

History Voting consists of two parts:

1. Adjustment of threshold values to the operating environment
2. Prediction of the fault rate and calculation of trust and prediction quality

4.1 *Adjustment of threshold values*

All related works from above use fixed threshold limits for the classification of fault types. This assumption only matches an environment where the fault rate is known. To flexibly classify the fault rate we introduce three threshold variables: φ_ι, φ_τ and φ_π. These represent the upper borders for the rate of permanent (φ_π), intermittent (φ_ι) and transient faults (φ_τ). After a reset the variables are set to known maximum values of the expected fault scenario. For security and from the observations, φ_π is set to a fixed value and is not dynamically adjusted. A counter Δ_i is needed to measure the time (in cycles) between faults. In the fault-free case, Δ_i is increased every cycle, else the trend of the fault rate is classified by the function i(a,b), defined by:

$$i : \mathbb{N}\times\mathbb{N} \to \{0,1\}$$

$$i(a,b) := \begin{cases} 0 \; if \;\; \varphi_\iota < \Delta(a,b) \le \varphi_\tau \text{ and} \\ 1 \; if \;\; \varphi_\pi < \Delta(a,b) \le \varphi_\iota \end{cases}$$

$$\text{with} \quad \begin{array}{l} \Delta : \mathbb{N}\times\mathbb{N} \to \mathbb{N} \\ \Delta(a,b) := |a-b|. \end{array}$$

Table 1 shows the coding of i(a,b) and the consequence on the trust (γ). γ>> represents the bitwise shifting of value γ to the right, γ++ an increase of value γ. If a fault cannot be tolerated, the trust is decremented until the minimum (null) is reached.

Table 1: Coding of fault rates

Coding i(a,b)	Fault rate	Consequence
0	Normal	γ++
1	Increase	γ>>

If Δ_i is substantially greater than φ_τ, this could mean that the diagnose unit does not respond. In this case, tests should be carried out to prevent further faulty behavior.

4.2 *Forecast of fault rates, calculation of trust and prediction quality*

A further observation is that a fault-tolerant system which does not know its past cannot express if a transient or intermittent (e.g. through frequent usage of a faulty component) fault occurred. Thus, it cannot predict these faults and cannot derive the system behavior over time. Therefore, we include a small memory (the *history*). The history holds the last three fault rates interpreted by *i(a,b)*.

For a prediction, neuronal nets or methods like branch prediction can be used. However, these methods are costly regarding time and area. We use a simple pattern matching depicted in Table 2. Here, the predicted fault rate and a symbolic representation are shown. For a time-efficient implementation, the history is limited in size. Three recent fault rates are considered. If the prediction matches the current fault rate development, the prediction quality η is increased, else decremented. Here, also the field data from Figure 4 was taken into account. The symbols are shown in Table 3.

Table 2: History and forecasted fault rates

H[1]	H[2]	H[3]	Fault rate	Prediction
0	0	0		0
0	0	1		0
0	1	0		0
0	1	1		1
1	0	0		0
1	0	1		1
1	1	0		0
1	1	1		1

Table 3: Symbols (History Voting)

Symbol	Description
γ_i	Trust of unit i
φ_τ	Upper threshold: normal fault rate
φ_ι	Mid-threshold: increased fault rate
φ_π	Lower (fixed) threshold: permanent fault
η	Quality of prediction
υ	If $\eta>\upsilon$, trust can be adjusted (quality-threshold)
Δ_i	Value of cycle-counter when detecting fault i
H[i]	Entry i in the history
Entries	Maximal number of entries in the history
Prediction	Prediction of the fault rate from the history (s. Figure 5)
Predict	Pattern matching to forecast the fault (s. Figure 5)

Figure 5 shows the algorithm for the calculation of trust, the prediction and its quality.

First, we test if an irreparable internal (INTFAULT) or external fault (EXTFAULT) was signaled. If so, the fail-safe mode must be initiated. If no such fault occurred, the cycle counter Δ_i is increased. If a fault is detected (INT-/ and EXTFAULT are excluded here), the forecasted fault rate i(a,b) is entered into the history H. Over *Predict*, a prediction (*Prediction*) is made. The last fault rate is compared with the current one. If the prediction is correct, η is increased. Else it will be decremented until the minimum is reached. Only if $\eta>\upsilon$ the prediction can modify the trust γ and thus the system behavior. The more dense faults occur in time, the less trust a unit gets. The greater the trust, the higher the probability of a correct execution. A slow in- or decrease of the fault rate signals a change within the operating environment and threshold values are modified. Δ_{i-1}, the last distance of faults in time will be compared with the actual Δ_i. If the elevation or decrease is over 50 % $\left(\left(\Delta_i > \left(\Delta_{i-1} >> 1\right)\right), \left(\Delta_i \le \left(\Delta_{i-1} >> 1\right)\right)\right)$, we have a sudden change of the fault rate and threshold values will not be adjusted.

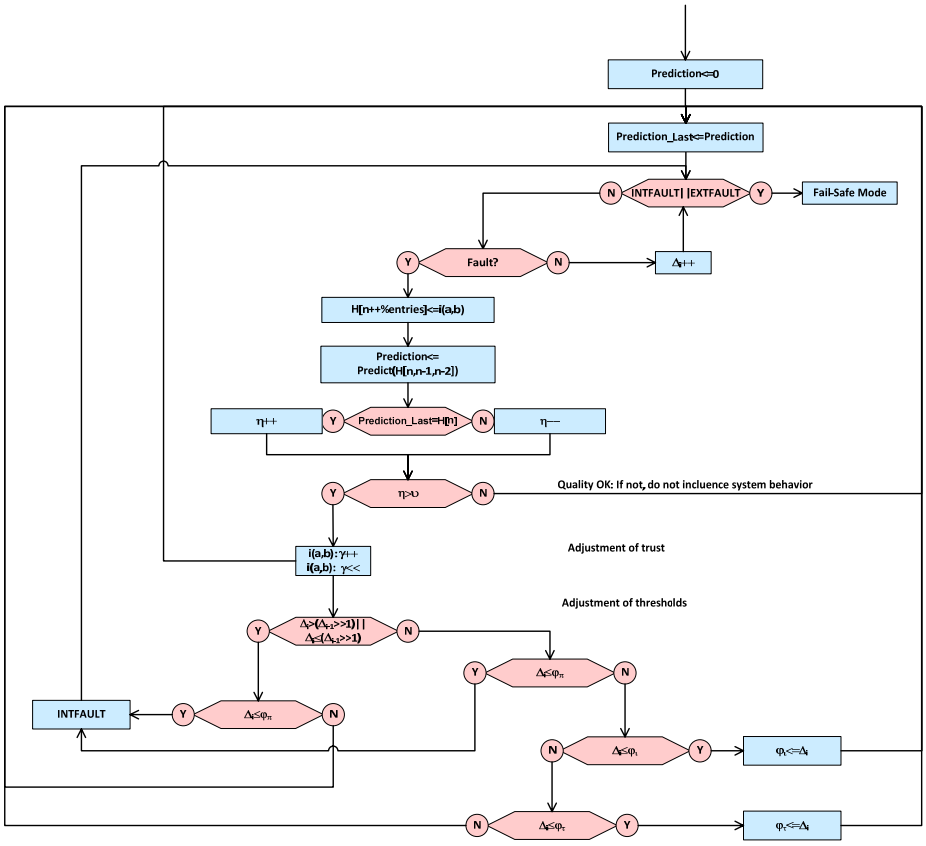

Figure 5: Calculation of trust and prediction

Two possibilities to signal permanent internal faults exist:

- The trust γ_i in unit i is less than the value p_t (slow degradation, not shown in Figure 5)
- Δ_i is less than threshold φ_π (sudden increase)

Hereby, we assume that no unit permanently locks resources, since then the trust of other units will decrease disproportionately.

5 EXPERIMENTAL RESULTS

To judge the mechanism, it was modeled in software. Figure 6 shows the successful adjustment of threshold values (fault rate $\lambda=10^{-5}$). The distance of faults in time, the threshold values and the accuracy are shown (from top to bottom: φ_τ, φ_ι, φ_π and accuracy in %). We purposely chose nearly equal (difference 100 cycles) starting values for φ_ι and φ_τ, since we wanted to show the flexibility of the mechanism. We see how the fault rate is framed by threshold values. Threshold φ_π is set to a value where a permanent fault can be ascertained (100 cycles).

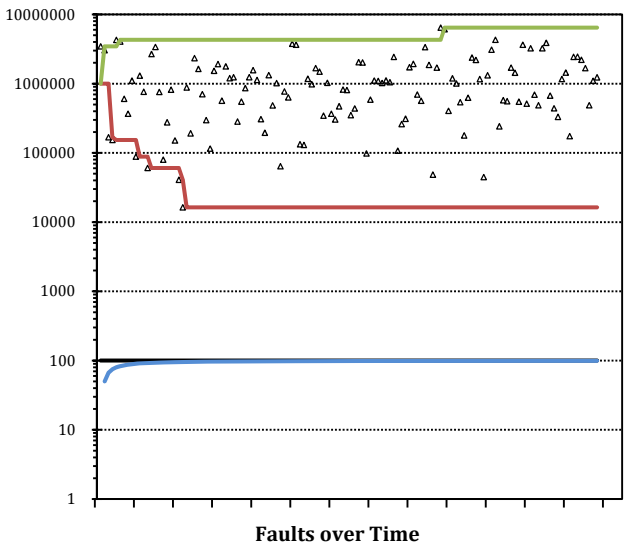

Figure 6: Successful adjustment of threshold values

In the beginning, the accuracy is low due to the initial values of φ_τ and φ_ι but evening out at about 98 %. If these values are correctly initialized, the accuracy would have been 100 %. Table 4 shows the resource demands for History Voting (FPGA, Xilinx Virtex-e XCV1000).

Table 4: Resource demands (FPGA)

Place and route		
Critical path (ns)	9.962	
Energy consumption (at 200 MHz)		
	mA	mW
1,8 V voltage supply	6.88	12.39
Area		
Slices	188	
Slice-FFs	200	
4-Input LUTs	233	
IOBs	4	
Gate Count	3661	

Table 5 shows the resource demands for History Voting for a standard-cell design by using a 130 nm, 6 metal layer CMOS technology.

Table 5: Resource demands (standard-cell)

Place and route	
Critical path (ps)	3308
Area (λ^2)	1175 x 1200
Transistors	5784
Capacity (pF)	8.8

6 CONCLUSION

In this work we presented a novel fault classification scheme in hardware. Apart from other schemes, the developed *History Voting* classifies the fault rate behavior over time. From this, a prediction is made. Only if the prediction quality exceeds a known value, the trust in units can be adjusted. From the implementations a proof of concept is made. From the results, we see that the scheme is relatively slow. Since faults – apart from permanent ones – occur seldom in time, this will not appeal against the scheme. It will easily fit even on small FPGAs. For the scheme, many application areas exist. Depending on the size of the final implementation, it could be implemented as an additional unit on a space probe, adjusting the system behavior during the flight. Another application area are large-scale systems, equipped with application specific FPGAs e.g. to boost the performance of cryptographic applications. Here, the scheme could be implemented to adjust the performance of a node, e.g. identify faulty cores using their trust or automatically generate warnings if a certain fault rate is reached.

REFERENCES

[1] E. Normand. *Single-Event Upset at Ground Level*. IEEE Trans. on Nuclear Science, vol. 43, no. 6, part 1, p.2742-2750, 1996.

[2] T. Karnik et al. *Characterization of Soft Errors caused by Single-Event Upsets in CMOS Processes*. IEEE Trans. on Dependable and Secure Computing, vol. 1, no. 2, p.128-143, 2004.

[3] R. Baumann, E. Smith. *Neutron-induced boron fission as a major source of soft errors in deep submicron SRAM devices*. In Proc. of the 38th Int'l. Reliability Physics Symp., p.152-157, 2000.

[4] F. L. Kastensmidt, L. Carro, R. Reis. *Fault-tolerance Techniques for SRAM-based FPGAs*. Springer-Verlag, ISBN 0-387-31068-1, 2006.

[5] National Geophysical Data Center (NGDC). *Cosmic Ray Neutron Monitor* (Kiel). ftp://ftp.ngdc.noaa.gov/STP/SOLAR_DATA/COSMIC_RAYS/kiel.07. Revision 12/2007/ cited 21.01.2008.

[6] The International Technology Roadmap for Semiconductors (ITRS). *Front End Processes*. http://www.itrs.net/Links/2005ITRS/FEP2005.pdf, 2005 edition/ cited 18.01.2008.

[7] C. Constantinescu. *Trends and Challenges in VLSI Circuit Reliability*. IEEE Micro, vol. 23, no. 4, p.14-19, 2003.

[8] M. Brehm, R. Bader, R. Ebner, Leibniz-Rechenzentrum (LRZ) der Bayerischen Akademie der Wissenschaften, *Hardware Description of HLRB II*, http://www.lrz-muenchen.de/services/compute/hlrb/hardware. Revision 29.03.2007/ cited 30.11.2007.

[9] A. Bondavalli, et al. *Threshold-Based Mechanisms to Discriminate Transient from Intermittent faults*. IEEE Trans. on Computers, vol. 49, no. 3, p.230-245, 2000.

[10] M. M. Tsao, D. P. Siewiorek. *Trend Analysis on System Error Files*. In Proc. of the 13th Int'l Symp. on fault-Tolerant Computing (FTCS-13), p.116-119, 1983.

[11] T.-T. Y. Lin, D. P. Siewiorek. *Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis*. IEEE Trans. on Reliability, vol. 39, p.419-432, 1990.

[12] R. K. Iyer et al. *Automatic Recognition of Intermittent Failures: An Experimental Study of Field Data*. IEEE Trans. on Computers, vol. 39, no. 4, p.525-537, 1990.

[13] G. Mongardi. *Dependable Computing for Railway Control Systems*. In Proc. of the 3rd Dependable Computing for Critical Applications (DCCA-3), p.255-277, 1993.

[14] N. N. Tendolkar, R. L. Swann. *Automated Diagnostic Methodology for the IBM 3081 Processor Complex*. IBM Journal of Research and Development, vol. 26, p.78-88, 1982.

[15] L. Spainhower et al. *Design for fault-Tolerance in System ES/9000 Model 900*. In Proc. of the 22nd Int'l Symp. Fault-Tolerant Computing (FTCS-22), p.38-47, 1992.

[16] G. Latif-Shabgahi, P. Bennett. *Adaptive Majority Voter: A Novel Voting Algorithm for Real-Time fault-Tolerant Control Systems*. In Proc. of the 25th Euromicro Conference, vol. 2, p.2113ff, 1999.

[17] J. M. Bass, G. Latif-Shabgahi, P. Bennett. *Experimental Comparison of Voting Algorithms in Cases of Disagreement*. In Proc. of the 23rd Euromicro Conference, p.516-523, 1997.

[18] P. Agrawal. *Fault Tolerance in Multiprocessor Systems without Dedicated Redundancy*. IEEE Trans. on Computers, vol. 37, no. 3, p.358-362, 1988.