On the Probability of Detecting Data Errors Generated by Permanent Faults Using Time Redundancy

Joakim Aidemark, Peter Folkesson, and Johan Karlsson
Department of Computer Engineering
Chalmers University of Technology
S-412 96 Göteborg, Sweden
{aidemark, peterf, johan}@ce.chalmers.se

Abstract

Time redundant execution of tasks and comparison of results is a well-known technique for detecting transient faults in computer systems. However, time redundancy is also capable of detecting permanent faults that occur during or between the executions of two task replicas, provided the faults affect the results of the two tasks in different ways. In this paper, we derive an expression for estimating the probability of detecting data errors generated by permanent faults with time redundant execution. The expression is validated experimentally by injecting permanent stuck-at faults into a multiplier unit of a microprocessor. We use the derived expression to show how tasks can be scheduled to improve the detection probability of errors generated by permanent faults. We also show that the detection capability of permanent faults is low for the Temporal Error Masking (TEM) technique (i.e. triplicated execution and voting to mask transient faults) and may not be increased by scheduling. Thus, we propose complementing TEM with special test tasks.

1. Introduction

The importance of dependability in embedded systems will increase dramatically as future computers take a more active role in everyday control applications such as drive-by-wire or brake-by-wire systems in vehicles. In addition, the ongoing reduction of device geometries and supply voltages increases the risk for not only transient faults but also for permanent faults. Even if permanent faults will occur with lower probability than transient faults it is never satisfactory or even acceptable to leave fault/error possibilities unanalyzed for safety-critical systems.

Many modern microprocessors provide extensive on-chip error detection mechanisms (EDMs) such as error detection and correction on memory, caches and registers, illegal op-code detection, address range checking. Many of these mechanisms will detect errors generated by permanent faults in the same way as errors generated by transient faults. However, the effects of certain faults occurring, e.g. in the arithmetic units of a microprocessor such as adders and multipliers may pass undetected, which justifies the use of software implemented error detection techniques such as executable assertions [1] or time redundant execution [2, 3].

Time redundancy has become increasingly attractive for achieving fault-tolerance due to the declining prices of high-performance microprocessors. In this paper, time redundant execution and comparison of results are referred to as Temporal Error Detection (TED). TED can be applied on different levels such as the instruction level [4], procedure level [5] or at the task level [6]. Recent studies utilize modern technologies in processors to reduce the time overhead [7]. In [8] we introduced a real-time kernel, which enables transient faults to be tolerated by using TEM. The real-time kernel executes all critical tasks twice and compares the results to detect errors. A third execution is started if an error is detected by the comparison or by other EDMs in the system. This allows the kernel to mask a transient fault by conducting a majority vote on the three results.

In this paper, we investigate how TED and TEM techniques are affected by permanent faults in the arithmetic units of a processor during system operation. We derive an expression for calculating the probability that data errors generated by permanent faults are detected by TED and TEM. The expression is based on the probability that a fault occur in a certain time interval during the time redundant execution of a task and the probability that the fault is activated (i.e. a faulty component is used, and an error is generated when executing an arithmetic operation).

The derived expression shows that the error detection capability of TED can be increased through appropriate scheduling. For TEM, however, the detection probability

Proceedings of the 9th IEEE International On-Line Testing Symposium (IOLTS'03)
0-7695-1968-7/03 $17.00 © 2003 IEEE

may not be increased through scheduling. Thus, we propose the use of a special test task that execute in parallel with TEM, which improve the detection capability of permanent faults. We have validated the expression by conducting fault injection experiments on a multiplier unit of a microprocessor.

In the next section, the expression for estimating the probability of detecting errors generated by permanent faults using time redundancy is derived. In Section 3, results from fault injection experiments are used to investigate the accuracy of the expression. Section 4 discusses the implications of the expression to improve the detection probability for TED and TEM. Finally, the conclusions of this study are given in Section 5.

2. Detecting Data Errors with TED and TEM

In this section, we derive an expression for estimating the probability of detecting data errors generated by permanent faults. First, the assumptions used are given. Then, the expression is derived for a single periodic task using TED. The expression is extended to include multiple periodic tasks and finally, an expression for estimating the probability of detecting errors with the TEM approach is also shown.

2.1. Assumptions

In our analysis, we make the following assumptions:

A1: Only single permanent faults occurring in the arithmetic units, i.e. faults affecting arithmetic operations such as MUL, DIV, ADD and SUB, during actual system operation are considered.

A2: Permanent faults occur randomly with uniform distribution in time.

A3: The same sequence of operations is executed regardless of the input data, which is the case for, e.g. various digital signal processing applications and control algorithms.

A4: All arithmetic operations are considered as independent events, i.e. if a fault is activated, the resulting data error will not be masked by any further executions of operations.

A5: Permanent faults occurring during the execution of an operation are considered to have the same activation probability as if the faults had occurred just before the execution of the operation. Consequently, this implies the highest activation probability since the actual probability presumably is lower at the end of the operation. (The time to execute the operation may also be considered negligible compared to the time to execute the task.)

A6: We assume that the probability that an operation x activates a fault is constant for all executions of the operation. This is reasonable if the same subset of possible inputs is used for each execution of x (see Section 2.2).

2.2. Detecting Data Errors with TED

TED is able to detect data errors generated by permanent faults if two time redundant executions produce different results. This is the case when a fault occurs after the first execution of a task has started and the fault does not affect the first execution in the same way as the second execution.

An example of this is shown in Figure 1 where a periodic task, T_A is executed twice. The time redundant replicas are denoted $T_{A,1}$ and $T_{A,2}$ respectively. When the task has been executed twice, the results are compared and the microprocessor is idle until the next invocation of the task. The time for executing the comparison of results is considered negligible compared to the time to execute the task and is therefore not shown in the figure.

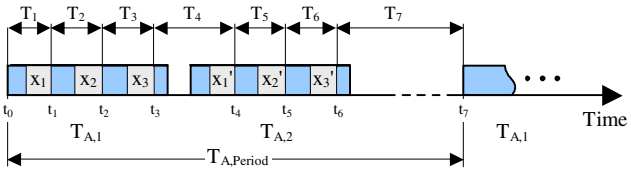

Figure 1. A single periodic task

x_1, x_2 and x_3 represent one specific arithmetic operation (e.g. a MUL operation with arbitrary operands) that is executed three times in the task. x_1', x_2' and x_3' are the repeated operations in the second execution of the task. (Note that the times to execute the operations are exaggerated in the figure.) The tasks period time in Figure 1 is divided into seven time intervals, i.e. the time interval T_1 start at time t_0 and ends after the first operation x_1 at time t_1 etc.

The probability that a data error is detected by time redundant execution depends on the point in time a fault occurs and the probability that a fault is activated.

Let P_i denote the probability that a fault occurs in a certain time interval T_i. Using assumption **A2**, P_i can be computed as:

$$P_i = \frac{k \cdot T_i}{k \cdot T_{Period}} = \frac{T_i}{T_{Period}} \qquad (2.1)$$

where T_i is the length of time interval i, T_{period} is the period time of the task and k is the total number of periods.

Let Px_j denote the probability that a fault is activated when a certain operation x_j is executed. Px_j can be calculated as:

$$Px_j = \frac{\sum_{k=1}^{M} Input_{act,k}}{M} \qquad (2.2)$$

where M is the number of possible inputs for the specific operation and $Input_{act,k}$ is 0 for the inputs that did not activate the fault and 1 for the inputs that activated the fault. Note that different subsets of the M possible inputs may be used for different executions of x. The use of each subset should then be considered as a different operation.

Let Px,tot be the total probability that a fault is activated by any of several executions of a specific operation x. The probability Px,tot that the fault is activated for two subsequent executions of operation x, x_1 and x_2, can be derived as:

Px,tot = P(At least one of the operations x_1 or x_2 activates the fault)
= 1-P(x_1 does not activate the fault AND x_2 does not activate the fault)

According to assumption **A4**, the probability of activating a fault by operation x_1 is independent of the probability of activating a fault by operation x_2. Px,tot can then be expressed as:

= 1-P(x_1 does not activate the fault) · P(x_2 does not activate the fault)
= 1-(1-P(x_1 activates the fault)) ·(1-P(x_2 activates the fault))

which can be generalized to:

$$Px,tot = 1-(1-Px_1)(1-Px_2)... (1-Px_i) \qquad (2.3)$$

for an arbitrary number i of repeated executions of operation x.

Using equation (2.1) to (2.3), the probability of detecting an error in the example given in Figure 1 can be derived as follows. If a fault occurs during time interval T_1 or T_7, it can never be detected since it will always affect $T_{A,1}$ and $T_{A,2}$ in the same way. (Note that we assume (**A5**) that a fault occurring at the end of an operation has the same activation probability as if the fault had occurred in the beginning of the operation.) If the fault occurs during time interval T_2, an error will be detected if the first operation of $T_{A,2}$ (x_1' in Figure 1) activates the fault (i.e. since the fault will affect x_2 and x_3 in $T_{A,1}$ and $T_{A,2}$ in the same way). Thus the probability that an error is detected $P(D)$ can be computed as $P(D) = P_2 \cdot Px_1'$, where P_2 is the probability that a fault occur in time interval T_2, and Px_1' is the probability that a fault is activated when executing operation x_1'.

Deriving $P(D)$ for the remaining time interval T_3 to T_6 in the example in Figure 1 can be done in the same way. If the fault occurs during time interval T_3, the error will be detected if either the first and/or the second operation of $T_{A,2}$ (x_1' or x_2' in Figure 1) activates the fault which gives $P(D)=P_3 \cdot (1-(1-Px_1')(1-Px_2'))$. For T_4 there are three possibilities that the fault can be activated (by x_1', x_2' or x_3' in Figure 1), thus, $P(D)=P_4 \cdot (1-(1-Px_1')(1-Px_2')(1-Px_3'))$. At T_5 there are again only two possibilities that the fault is activated differently between $T_{A,1}$ and $T_{A,2}$ (by x_2' or x_3' in Figure 1), $P(D)=P_5 \cdot (1-(1-Px_2')(1-Px_3'))$, and during T_6 there is only one (x_3' in Figure 1) , $P(D)=P_6 \cdot Px_3'$.

Using assumption **A6**, i.e. $Px=Px_1=Px_2=Px_3$, the total probability, $P(D_x)$ of detecting an error generated by a permanent fault for one task can then be computed as:

$$P(D)=P_2 \cdot Px+P_3 \cdot (1-(1-Px)^2)+P_4 \cdot (1-(1-Px)^3)+P_5 \cdot (1-(1-Px)^2)+P_6 \cdot Px \qquad (2.4)$$

The expression 2.4 can be generalized for an arbitrary number of time intervals to:

$$P(D_x)=\left(\sum_{i=1}^{(N-3)/2}[P_{i+1}+P_{N-i}] \cdot [1-(1-Px)^i]\right)+P_{(N+1)/2} \cdot [1-(1-Px)^{(N-1)/2}] \qquad (2.5)$$

where N is the number of intervals. Note that the number of intervals is always odd since the number of operations is always even for a task that is executed twice.

If L different operations can activate a certain fault, the probability of detecting the error $P(D)$ is the probability of the union of the events that can detect the error for each operation $l \in L$ (details can be found in [9]):

$$P(D) = P\left(\bigcup_{l=1}^{L} D_l\right) \qquad (2.6)$$

2.3. Multiple Periodic Tasks

There may be several tasks, which are double executed in a schedule. The probability of detecting errors generated by permanent faults can thereby be estimated based on all double executed tasks in the interval where all periodic tasks are invoked at least once. This interval is called the Least Common Multiple time (LCM) [10]. This means that the detection probability for all tasks in the LCM interval can be computed from the equations in the last section as:

$$P(D_{LCM,x}) = \frac{1}{LCM}\sum_{k=1}^{Q} T_{Period,k} \cdot P(D_{x,k}) \qquad (2.7)$$

where Q is the number of double executed tasks in LCM.

Some systems allow pre-emption of tasks, i.e. a higher priority task can interrupt lower priority tasks and after the higher priority task has finished, the lower priority task can resume its execution. This means that the time between the operations in the lower priority task can increase (a time interval T_i can increase), and thus, the probability of detecting errors will be higher according to equation (2.1) and (2.5). Thus, the lowest error detection probability is reached when no tasks are pre-empted.

2.4. Detecting Data Errors with TEM

The TEM approach is targeted for *tolerating* data errors generated by transient faults. Errors are masked by a majority vote on the results from three executions of the task. The disadvantage of TEM is that a permanent fault may cause the second and third execution to produce identical erroneous results, which thus are selected as the

output by the majority vote.

Figure 2 shows an example of a periodic task that is executed two times to detect errors and a third execution is only started if an error is detected. x_1, x_2 and x_3 represent a specific arithmetic operation that is executed three times in each task replica.

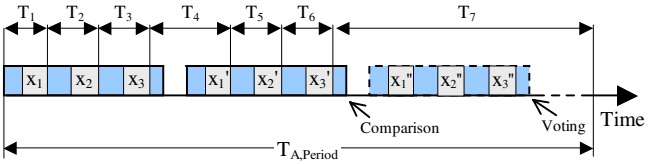

Figure 2. Time interval in a TEM execution

Using equation (2.1) to (2.3), the probability of detecting an error in the example given in Figure 2 can be derived in the same way as for TED (see [9] for details).

A fault occurring during time interval T_1 to T_4 will never be detected since it will affect two executions in the same way. A fault occurring during time interval T_7 is not detected since the third execution is only started if a fault is detected by the comparison. Thus, the probability, $P(D_{TEM,x})$ that the majority voter detects an error generated by a permanent fault can be computed as:

$$P(D_{TEM,x}) = P_5 \cdot Px_1'' + P_6 \cdot (1-(1-Px_1'')(1-Px_2''))$$

where P_5 and P_6 is the probability that a fault occurs in the time interval T_5 and T_6, and Px_i is the probability that the fault is activated when executing operation x_i. The expression can be generalized to:

$$P(D_{TEM,x}) = \sum_{i=1}^{(N-3)/2} P_{i+(N+1)/2} \cdot \left[1-(1-Px)^i\right] \qquad (2.8)$$

where N is the number of time intervals.

3. Evaluation

In this section, we experimentally investigate the validity of two properties that the derivation of equation (2.5) is based on:

- **P1:** The detection probability is different for different time intervals.
- **P2:** The detection probability increases with the number of operations executed.

3.1. Experimental setup

Target system: As mentioned in the previous sections, we focus on the arithmetic units of a microprocessor. Specifically, this evaluation is performed on the multiplier unit of a CPU, i.e. faults are only injected into the multiplier. Thus, only multiplication operations can activate these faults. The target processor is a RTL level VHDL model of the Thor microprocessor [11] where the multiplier unit is replaced with a structural gate-level VHDL description.

The multiplier performs 32-bit by 32-bit integer and floating-point multiplications. In order to reduce chip size and increase clock speed, the multiplier is designed 34x14 bits and performs multiplication in several cycles.

The multiplier consists of three parts, a *Booth algorithm*, a *Wallace tree*, and an *Adder*. The Booth algorithm and the Wallace tree are both used to speed up the multiplication. The Booth algorithm reduces the number of partial products to be summed, and the Wallace tree accelerates the addition of partial products.

The Thor microprocessor includes several internal EDMs. However, the only internal EDM that can be triggered by faults injected into the multiplier performing integer multiplications is the *Overflow check,* i.e. an overflow of a signed integer or floating point operation.

Fault injection environment: The VHDL model of the Thor processor was executed using the ModelSim EE Simulator on five Unix workstations. Faults are injected by using simulator commands, which force signal values to zero or one. In addition, special command files were defined that start simulations of the processor and save the signals of interests into log files. The saved signals were then examined off-line in order to derive the results.

Workload: Both the workload program and the input data to the program must be considered when evaluating the impact of permanent faults in microprocessors. We chose a matrix multiplication program, and varied the input to the program to obtain representative results for the multiplier. To constrain the evaluation time, the matrix program is limited to multiplication of two 1 x 2 integer matrices $[a_1a_2]\cdot[b_1b_2]^T=[a_1\cdot b_1+a_2\cdot b_2]$. The matrix program was executed twice and the results from the two executions were compared to detect errors. The TEM technique was not experimentally evaluated. Note however, the number of faults that would have been detected by the TEM technique can be estimated using equation (2.8). The input data to the matrix operation are signed integers, which can have values of $\pm 2^{31}$-1. Thus, to avoid triggering of an overflow exception during fault free operation, the input values were randomly chosen as:

$$Indata = random\left(\left\lfloor\sqrt{\frac{\max\ Int}{2}}\right\rfloor\right)$$

Figure 3 shows the time redundant execution of the matrix multiplication task that is used in the experiments.

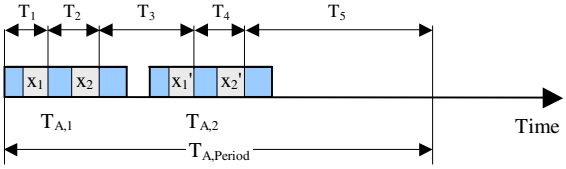

Figure 3. Execution of the matrix operation

Two MUL operations (called x_1 and x_2) are executed for each execution of the task (called $T_{A,1}$ and $T_{A,2}$). The intervals between the operations are denoted T_1 to T_5.

Fault model and fault locations: The fault models used in the experiments are the common logic-level stuck-at-(0/1). This fault model represents physical defects in the circuit caused by, e.g. electrical stress, hot electronic trapping, thin-oxide breakdown, electromigration, radiation etc. The physical defects may cause transistor stuck-on/off or shorts/open connection lines, which can manifest themselves as stuck-at-0 or stuck-at-1 in logical gates. Faults were injected in 24635 signals in the multiplier combinatorial logic, i.e. 10632, 4997 and 9006 locations respectively in the Booth algorithm, Wallace tree and Adder.

3.2. Verification of Property P1

A set of experiments was conducted to verify that the probability of detecting errors using TED depends on the point in time a fault occurs. Since faults are only injected into the multiplier, only multiply operations can activate them. Therefore, the detection probability of a fault should be the same at all times during a time interval. In addition, only faults occurring after the execution of the first operation (x_1 in Figure 3), but before the execution of the last operation (x_2' in Figure 3) can be detected.

Experiment definition: A fault F is defined as: $F = \{L, I\}$ x T, where the location (L) and the input data (I) are chosen as a set combined with the injection time (T) for the fault injection. 250 set of locations and four input data values, and 50 time points were selected randomly from each domain using uniform sampling distribution for each of the three parts of the multiplier (Booth, Wallace and Adder). In addition, both stuck-at-0 and stuck-at-1were injected for each fault F, which results in a total of $250 \cdot 50 \cdot 3 \cdot 2 = 75000$ injected faults.

Results: The results from the experiments are shown in Table 1. As expected, most errors were detected by TED in interval T_3 and no errors were detected by TED in interval T_1 or T_5. Faults injected after the last MUL (interval T_5) will not be activated until the next iteration, where they will have the same effect as for faults injected in interval T_1 (here, only one iteration is executed).

Table 1. Results of fault injection experiments

	T_1		T_2		T_3		T_4		T_5		All	
	#	%	#	%	#	%	#	%	#	%	#	%
Correct result	5076	37,6	5067	37,5	9024	37,6	9335	47,9	4500	100,0	33002	44,0
Detected by overflow EDM	6687	49,5	6696	49,6	11888	49,5	8334	42,7	0	0,0	33605	44,8
Detected by TED	0	0,0	1170	8,7	3088	12,9	1831	9,4	0	0,0	6089	8,1
Wrong result	1737	12,9	567	4,2	0	0,0	0	0,0	0	0,0	2374	3,1

A detailed investigation shows that faults injected into a signal in the multiplier result in the same detection probability irrespective of when in a time interval between multiply operations the fault was injected. Faults occurring during the execution of the operation also have lower detection probability than faults occurring between the operations.

The number of faults that would have been detected if the TEM technique were used for the fault injection experiments can be estimated using expression (2.8), i.e. $P(D_{TEM}) = P_4 \cdot Px$. Thus, the detection probability of TEM corresponds to the percentage of faults detected when injecting faults in the time interval T_4, i.e. 1831/75000=2.3% of all injected faults while TED detects 8.1% of all injected faults.

3.3. Verification of Property P2

Another set of experiments was conducted to verify that the probability of detecting errors also depends on the probability Px that a fault is activated when a certain operation x is executed (see equation (2.2)), and the number of operations i, executed that can activate a fault (see equation (2.3)).

Experiment definition: In these experiments, a fault F is defined as: $F = L$ x I x T where the location (L) and the input data (I) are selected randomly from the respective domain using uniform sampling distribution. Using the results from Section 3.2 enable us to decrease the number of time-points (T) to inject faults to one time-point for each time interval since the effect of an injected fault is equal for all points in a time interval between multiply operations. 246 locations were chosen, of which 106 signals were in Booth, 50 signals in the Wallace tree and 90 signals in the Adder part. 50 sets of four input data values and 1 time point in each of the intervals T_2, T_3 and T_4 were chosen. Both stuck-at-0 and stuck-at-1 faults were injected for each fault F, which results in a total of $(106+90+50) \cdot 3 \cdot 2 = 73800$ injected faults.

Results: The activation probability for one multiplication Px is estimated by using the results obtained when injecting faults during the time interval T_4, i.e. these faults can only affect the last multiplication, x_2'. Thus, the activation probability, Px, corresponds to the number of errors detected either by the TED or the overflow EDM. (The experimental results show that Px is 70% for the Wallace part, 58% for the Booth part and 31% for the Adder part).

To investigate the accuracy of equation (2.3), the activation probability Px obtained when injecting faults into time interval T_4 is used to estimate the activation probability for two multiplications as $Px,tot = 1-(1-Px)^2$. The activation probability for two multiplications can also

be estimated (called Px,tot_{FI}) by using the results obtained when injecting faults during the time interval T_3, i.e. these faults can affect both the x_1' and the x_2' multiplication. Table 2 shows the difference between Px,tot and Px,tot_{FI} for the various parts of the multiplier, where the average difference ranges from 0,001 to 0,014.

Table 2. Confidence of the proposed expression

	Average difference ($Px,tot - Px,tot_{FI}$)	90% confidence interval
Booth algorithm	-0,001	± 0,07
Wallace tree	0,014	± 0,1
Adder	-0,006	± 0,06

There are two possible reasons for the differences between the obtained and estimated Px values. First, only 50 multiplications were performed for each fault to investigate the activation frequency. Thus, some variance can be expected as shown in Table 2. Second, if assumption **A4** is not valid, i.e. if some errors are masked, Px obtained from interval T_3 is expected to be higher than Px obtained from interval T_4. From the analysis we can see that we have a certain variance, but it is low. This makes us conclude that even if it is possible for errors to compensate each other, this probability is low. Thus, equation (2.3) appears to be correct.

The results are shown in detail in Table 3. As shown in the table, 4.4% of the faults injected during interval T_2 caused wrong results. The reason for this is that the injected faults are activated by the second MUL operation in both executions, which caused two equal but faulty results to be produced.

Table 3. Results of fault injection experiments

	T_2		T_3		T_4	
	#	%	#	%	#	%
Correct result	10015	40,7	10035	40,8	12126	49,3
Detected by overflow EDM	11226	45,7	11225	45,6	9793	39,8
Detected by TED	2269	9,2	3340	13,6	2681	10,9
Wrong result	1090	4,4	0	0,0	0	0,0

4. Implications of the expression

We have validated the expression derived in Section 2 for a multiplier unit. However, this is just an indication that the derived expression is valid in general. The expression needs to be validated in more detail and for other parts of a processor. However, assuming that the expression is valid, some suggestions for increasing the probability of detecting errors can be made.

4.1. Increasing the Error Detection Probability

Reflecting on the derived expression, it can be seen that the time between operations relates to the probability of detecting errors. Thus, the probability of detecting errors can be increased by enlarging the interval between the first and second execution.

Consider a simple example with one periodic tasks which have the period time $T = 10$ ms and the execution time $C = 2$ ms, and the deadline d equal to the period time. The task performs one multiplication 0.5 ms after the start and another multiplication 0.75 ms after the first multiplication. Figure 4 suggests two different time redundant execution schedules for TED.

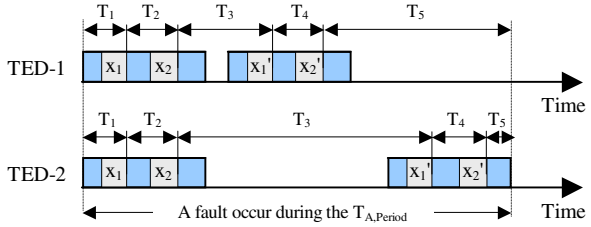

Figure 4. Time redundant execution of tasks

Using equation (2.5), the probability of detecting errors with the TED scheduling examples can be estimated as:

$$P(D_x) = (P_2 + P_4) \cdot Px + P_3 \cdot (1-(1-Px)^2) \qquad (4.1)$$

In TED-1 the tasks are executed one after the other, while in TED-2 the tasks are separated, thereby increasing the time interval T_3. Enlarging the interval T_3 without changing the period time of the task will increase P_3 (see equation (2.1)). Thus, the probability of detecting errors will be higher for TED-2 according to equation (4.1).

In TEM, the probability that the majority voter detects an error generated by a permanent fault can be estimated using equation (2.8) as:

$$P(D_{TEM,x}) = P_4 \cdot Px \qquad (4.2)$$

Since P_4 is computed from time interval T_4, which is located during execution of a single task replica, it is not possible to increase the probability of detecting errors by changing the schedule of the tasks in TEM.

Figure 5 shows the probability of detecting an error for TED-1, TED-2 and TEM using equation (4.1) and (4.2), where different values of Px are used.

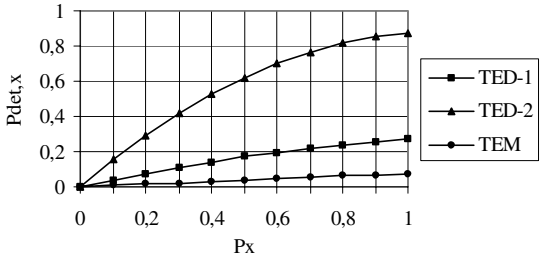

Figure 5. Detection probability for TED and TEM

The figure shows that the probability of detecting errors increases as the time interval between the first and second execution of the task is extended for TED while the

probability of detecting errors generated by permanent faults in the TEM approach is always lower.

A solution, which might improve the error detection capability of TEM, is to use a specially designed test tasks that check the arithmetic units for permanent faults. One approach could be to execute a number of operations in a test task that utilize all the arithmetic units, i.e. various logic and arithmetic operations. An approximation of the detection probability of such a test would be $(1-(1-Px)^i)$, where Px is the probability that the fault should be activated with the operation x and i is the number of such operations in the test. Scheduling a test task in combination with TEM can be performed by setting a low priority of the test tasks, which then execute only when extra time is available. The task can also be scheduled with a certain periodicity (based on the intensity of permanent faults) or be activated when suspecting a permanent fault, e.g. when three different results are obtained in TEM.

5. Conclusion

In this paper, the possibility of detecting errors generated by permanent faults with time redundancy was investigated. The probability of detecting an error depends on the intensity of permanent faults, the duration of the faults, the points in time the faults occur, the probability that the faults are activated and the task schedule.

Based on the investigation, an expression was derived for estimating the probability of detecting errors generated by permanent faults in the arithmetic parts of a microprocessor using time redundancy. To verify that the expression is valid, faults were injected into a structural gate-level VHDL model of a multiplier in the Thor microprocessor. The results correspond to the derived expression. However more experiments with different workloads are needed to verify that the expression is valid for all arithmetic parts of a processor.

The derived expression can be used to determine the detection probability of time redundancy based techniques. Furthermore, it can be used to optimize the task schedule of double executed tasks to maximize the probability of detecting errors generated by permanent faults.

The paper also considers the Temporal Error Masking approach (TEM), which is able to mask transient faults by triple time-redundant execution and voting. However, by using the derived expression, we identified that the probability of detecting errors generated by permanent faults in TEM is low. Therefore, a special test task that checks the arithmetic units for permanent faults in combination with TEM was proposed.

Further work will focus on investigating the inclusion of test tasks in combination with TEM. We believe that this complementary error detection technique in combination with TEM will significantly reduce the probability of value failures caused by both transient and permanent faults.

6. Acknowledgements

We would especially like to thank Örjan Askerdal at Chalmers University for his valuable suggestions and comprehensive comments on the paper. We thank Stefan Asserhäll and Torbjörn Hult at Saab Ericsson Space AB for providing the VHDL model of the Thor processor and for their technical assistance. This work was supported by ARTES and the Swedish Foundation for Strategic Research (SSF).

7. References

[1] D.M. Andrews. "Using Executable Assertions for Testing and Fault Tolerance", *9th Annual Int'l.. Symp. on Fault-Tolerant Computing*, 1979. New-York, USA.

[2] Johnson B.W., *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wessley, 1989.

[3] T. Lovric, "Dynamic Double Virtual Duplex System: A Cost-Efficient Approach to Fault-Tolerance", *Dependable Computing for Critical Applications 5*, IEEE Computer Society, 1998, pp. 57-74.

[4] N. Oh, P.P. Shirvani and E.J. McCluskey, "Error Detection by Duplicated Instructions In Super-scalar Processors," *IEEE Transaction on Reliability*, Sep. 2001.

[5] Oh, N., and E.J. McCluskey, "Procedure Call Duplication: Minimization of Energy Consumption with Constrained Error Detection Latency" *Proc. IEEE Int'l Symp. on Defect and Fault Tolerance in VLSI Systems*, 2001, pp. 182 –187.

[6] A. Damm, "The Effectiveness of Software Error-Detection Mechanisms in Real-Time Operating Systems", *Digest of Papers, 16th Int'l Symp. on Fault-Tolerant Computing Systems*, Washington, DC, USA, 1986, pp. 171-176.

[7] E. Rotenberg, "AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors", *Int'l Conf. on Dependable Systems and Networks*, Madison, WI, USA, 1999, pp. 84-91.

[8] J. Aidemark, J. Vinter, P. Folkesson, and J. Karlsson, "Experimental Evaluation of Time-redundant Execution for a Brake-by-wire Application", *Int'l. Conf. on Dependable Systems and Networks*, Washington DC, USA, 2002.

[9] J. Aidemark, Ö. Askerdal, "Use of Time Redundancy for Detection of Data Errors caused by Non-Transient Faults", *Report No. 03-09*, Department of Computer Engineering, Chalmers University of Technology, Sweden, 2003.

[10] J. Leung, and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks, *Performance Evaluation*, December 1988.

[11] *Saab Ericsson Space AB, Microprocessor Thor, Product Information*, September 1993.