Analysis of Possibilities to Overcome the Transient Faults in Real-time Systems with Time Redundancy

Sandra Djosic[1], Milun Jevtic[2] and Milunka Damnjanovic[3]

***Abstract* – In this paper we analyze timing constrains of one fault tolerant real-time system. Our goal is to estimate a probability of overcoming a transient fault detected during tasks executions. The faults are overcoming using technique of executing task again and time redundancy. Response time analysis (RTA) is the basis of our research and it is used to find minimum time between two consecutive faults which real-time system can tolerate. We have modified RTA to get more reliable real-time system.**

***Keywords* – Real-time system, Fault tolerance, Time redundancy.**

I. INTRODUCTION

A system is said to be real-time if the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed, [1]. Real-time systems play an important role in many areas of the daily life: robotics, cosmic research, automotive industry, process control, factory automation….

Those systems have been designed in order to be safe and extremely reliable. Reliability in a real-time system means that it can run continuously for extended periods of time: typically for years without any failures, [2]. They are usually realized as real time systems with the ability of tolerating some faults, [3]. A fault-tolerant system has to ensure that faults in the system do not lead to a failure.

The focus of our research is transient faults. Transient faults are temporary malfunctions of the computing unit or any other associated components, and cause an incorrect result to be compute. Transient faults can be caused by a variety of sources, such as atmospheric nuclear particles (alpha-particles, protons and neutrons) or electrical noise (power supply noise or electromagnetic interference), [3].

The key to fault tolerance is redundancy. It can be said that redundancy is the addition of information, resources, or time beyond what is needed for normal system operation [4]. There are of three kinds: hardware redundancy, software redundancy and time redundancy. Hardware redundancy is the addition of extra hardware to the system, such as spare processors which are used if one of the running processors fails. Software redundancy is the use of extra software modules to verify the result, or to use multiple versions of a program. Time redundancy is the use of additional time to perform the functions of a system. This time might be used to re-execute a faulty task or to execute a different version of the task. We are particularly interested in time redundancy techniques, since they are cost-effective as well as more suitable to applications where there are severe constraints on space and weight.

So, if a fault occurs during real-time task execution then it is necessary to overcome that fault and satisfies all timing constraints. We assume that the faults are overcoming using time redundancy and technique of executing task again. Our first analysis of transient fault tolerance in hard real-time systems with time redundancy was presented in [5]. In this paper we analyze possibilities to overcome the transient faults using response time analysis (RTA). More about RTA can be found in [6].

We use response time analysis to find minimum time (period) between two consecutive faults which real-time system can tolerate. For that period RTA guaranties that analyzed real-time system will be fault tolerant. If new fault occurs during that calculated period of time then RTA cannot guaranties overcoming of that fault. We saw that as a problem and our task was to modify RTA in order to obtain a guarantee for the case that extra fault occurs.

During modification we started with assumption to provide enough time redundancy for re-execution of the highest priority (the most critical) for the case of fault tolerance. Consider added extra time we modify the base RTA equation and present it in the paper. Our goal is to analyze how added time redundancy can be used for tolerance some new (extra) faults in the RTS. We use MATLAB for all calculations related to the RTA and the modify RTA.

The rest of the paper is organized as follow: Section II deals with the existing RTA applied on non-faulty RTS (part A) and faulty RTS (part B). Part C of Section II presents the research problem and our modification of RTA. Section III offers our conclusion.

II. ANALYSIS OF REAL-TIME SYSTEMS TIMING CONSTRAINS

A. Non-faulty RTS

In this paper we consider only a uniprocessor system where algorithm for scheduling real-time tasks could be Rate Monotonic, Deadline Monotonic [7] or any other priority assignment algorithm. We assume that each task is assigned a unique priority and that a task can be immediately preempted by a higher priority task. At run time, the highest priority task from the set of runnable tasks is allocated processor time.

[1]Sandra Djosic is with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Nis, Serbia, E-mail: sandra.djosic@elfak.ni.ac.rs.

[2]Milun Jevtic is with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Nis, Serbia, E-mail: milun.jevtic@elfak.ni.ac.rs.

[3]Milunka Damnjanovic is with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Nis, Serbia, E-mail: milunka.damnjanovic@elfak.ni.ac.rs.

We assume a set of n tasks, $\Gamma=\{\tau_1,..., \tau_n\}$ in which tasks are ordered according to the assigned priorities, where 1 denotes the highest priority and n denotes the lowest priority. Each task τ_i is assumed to have a minimum inter-arrival time T_i, worst case execution time (WCET) C_i and deadline D_i. We assume that $D_i \leq T_i$, for $i = 1, 2, \ldots, n$. We use hp(i) to denote the set of tasks with higher priorities than i, hp(i)=$\{\tau_j \in \Gamma \mid p_j > p_i\}$.

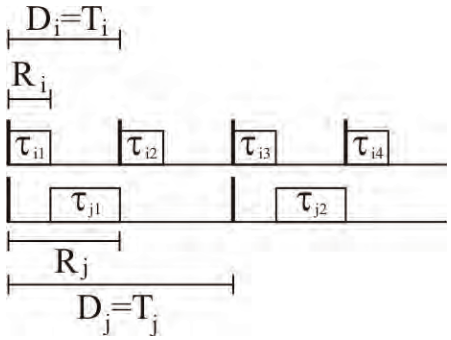

Fig. 1. Response time of tasks when there are no faults in the system

If there are no faults in the system then the response time of task τ_i can be evaluated using Eq. (1). Here the response time R_i, of a task τ_i, is expressed as the sum of its WCET C_i and interference due to preemption by higher priority tasks. If we can find $R_i \in [0, D_i]$, which satisfies the Eq. (1):

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \quad (1)$$

then task τ_i is feasible. The smallest value of R_i which satisfies the Eq. (1) is the worst case response time of task τ_i. Since R_i appears on both sides solutions can be obtained using the following recurrence relation:

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j \quad (2)$$

Iteration starts with $R_i^0 = C_i$. When $R_i^{n+1} = R_i^n$ we have found a minimum solution, that is R_i. If $R_i^{n+1} > D_i$ then task τ_i is infeasible and iteration is terminated.

TABLE I
TASK SET - CASE I

Task	Task characteristics				R_i
	C_i	T_i	D_i	p_i	
τ_1	30	100	100	1	30
τ_2	35	175	175	2	65
τ_3	25	200	200	3	90
τ_4	30	300	300	4	150

Fig. 1 presents scheduling of two periodic real-time tasks τ_i and τ_j when there is no fault in the system. System of these two tasks are schedulable i.e. both tasks execute before their deadlines, D_i and D_j. Response time of tasks τ_i and τ_j are the output results of RTA and they are also shown on Fig. 1.

We illustrate this procedure for a task set consisting of four periodic tasks. Timing characteristics for these four tasks are shown in Table I. Using Eq. (2) we found the value for the response times of the complete task set (last column in Table I). For all four tasks we got that $R_i < D_i$, for i = 1, 2, 3 and 4, which means that all tasks finished before their deadlines. It can be concluded that the real-time task set – case I is schedulable.

B. Faulty RTS

The fault-free assumption for one RTS is in fact not realistic because "non-faulty systems hardly exist, there are only systems which may have not yet failed", [6]. So, if a fault occurs during real-time task execution then it is necessary to overcome that fault and satisfy all timing constraints of real-time tasks. We consider transient fault and assume that the effects of a fault can be eliminated by simple re-execution of the affected task at its original priority level. Now, the response time analysis can be describe using Eq. (3):

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \left\lceil \frac{R_i}{T_F} \right\rceil \max_{j \in hp(i) \cup i} (C_j) \quad (3)$$

Eq. (3) has one more addend (then Eq. (1)) due to possible faults in the system. If we assume that inter-arrival time between faults is T_F then there can be at most $\left\lceil \frac{R_i}{T_F} \right\rceil$ faults during the response time R_i of task τ_i. Since these faults could occur during the execution of task τ_i or any higher priority task which has preempted τ_i, each fault may add $\max_{j \in hp(i) \cup i} (C_j)$ to the response time of task τ_i. So, the third addend in Eq. (3) presents an extra time needed tasks recovery due to faults.

Since R_i appears on both sides we need again recurrence relations:

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j + \left\lceil \frac{R_i^n}{T_F} \right\rceil \max_{j \in hp(i) \cup i} (C_j) \quad (4)$$

Eq. (4) calculates the response times of tasks in the presence of faults if the interval between two faults is T_F. Recurrence relations also starts with $R_i^0 = C_i$. When $R_i^{n+1} = R_i^n$ we have found a minimum solution, that is R_i. If $R_i^{n+1} > D_i$ then task τ_i is infeasible and iteration is terminated. Minimum value for T_F which satisfies Eq. (4) presents minimum time between two consecutive faults which real-time system can tolerate.

Fig. 2 illustrates RTA applied on faulty RTS. It can be seen scheduling of the same real-time tasks τ_i and τ_j when two faults occur in the system. Time between two consecutive faults T_F is long enough and real-time system can tolerate these faults. First fault occurs just a little bit before the end of tasks τ_{j1} execution. Real-time system overcomes this fault by executing task τ_{j1} again. Output results of RTA, response time

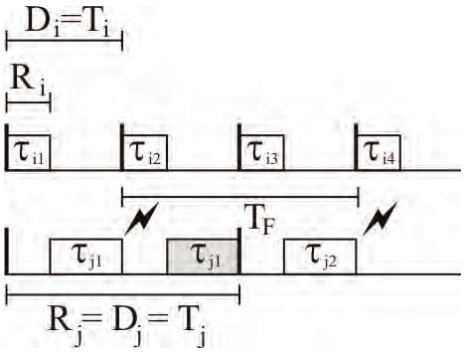

Fig. 2. T_F is long enough and RTS is fault tolerant

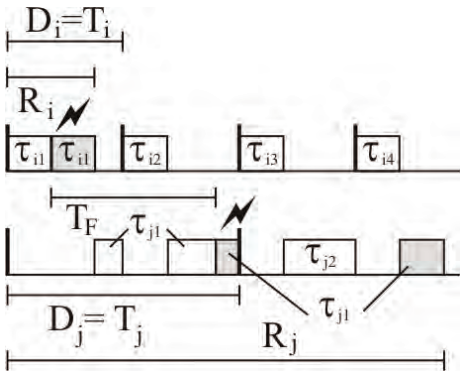

Fig. 3. T_F is not long enough that RTS stays fault tolerant

of tasks τ_i and τ_j, for the assumed value T_F are shown on Fig. 1.

Fig. 3 presents scheduling of the same real-time tasks τ_i and τ_j when two faults occur in the system. Now, time between two consecutive faults T_F is not long enough and real-time system cannot tolerate these faults. First fault occurs just a little bit before the end of tasks τ_{i1} execution. Real-time system can overcomes this fault by executing task τ_{i1} again. Second fault occurs just a little bit before the end of tasks τ_{j1} execution. Now time redundancy is not enough to tolerate this fault. Systems starts procedure for overcoming fault by executing task τ_{j1} again but timing characteristics of tasks τ_{j1} cannot be satisfied and τ_{j1} missing its deadline i.e. $R_j > D_j$. This is not acceptable in one hard real-time system, so in this case real-time system is not fault tolerant.

TABLE II
TASK SET - CASE I

Task	Task characteristics				T_F=300	T_F=200	T_F=275
	C_i	T_i	D_i	p_i	R_i	R_i	R_i
τ_1	30	100	100	1	60	60	60
τ_2	35	175	175	2	100	100	100
τ_3	25	200	200	3	155	155	155
τ_4	30	300	300	4	275	**340**	275

In Table II, we present the response times of the task set used in Table I for two different fault inter-arrival times. With a minimum fault inter-arrival time of T_F = 300 the task set is schedulable, but it is not schedulable with T_F = 200.

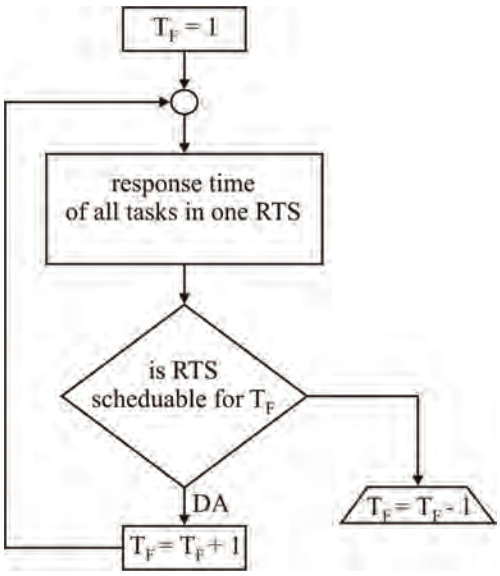

Fig. 4. Algorithm for finding minimum T_F

Based on Eq. (4) we realized algorithm (Fig. 4) for finding minimum time between two consecutive faults which real-time system can tolerate. We start from the minimum possible value for T_F and for that value we calculate response time for each task. After that, it is necessary to check is the real-time system fault tolerant. Depends on answer we continue process with increment value of T_F ("true" answer) or finish it ("not true") finding minimum T_F.

Using presented algorithm we calculated minimum T_F for the same task set – case I. The value for T_F is 275 and the response time of the tasks is shown in the last column of Table II.

C. Modification of RTA

Using RTA and our presented algorithm we can find minimum time (period) between two consecutive faults which one real-time system can tolerate. If minimum time between two faults is equal or greater then T_F then RTA can guaranty that this real-time system will be fault tolerant. But if minimum time between two faults is less then T_F then RTA cannot guaranties tolerance of that fault. For that case we have modified RTA to get fault tolerant RTS. The basic idea for modification was to ensure enough redundancy for the highest priority task and to use this extra spare time for potentially less priority tasks re-execution.

Let's illustrate idea with one simple real-time task set shown in Table III. If we apply presented algorithm on task set from Table III we can conclude that RTS is fault tolerant if T_F is 60 time units.

Let's ensure 100% redundancy for the task τ_1, doubling his execution time. New WCET for task τ_1 is:

$$C_1 = C_{1task} + C_{1extra} = 20+20 = 40$$

where C_{1task} is WCET of tasks τ_1 and C_{1extra} is an added time needed for tasks τ_1 re-execution. We applied presented algorithm with new input parameters and have got the result shown in Table IV.

TABLE III
TASK SET - CASE II

Task	Task characteristics				T_F=60
	C_i	T_i	D_i	p_i	R_i
τ_1	20	100	100	1	40
τ_2	25	175	175	2	95
τ_3	20	200	200	3	160
τ_4	25	300	300	4	300

Now, response time for task τ_1 is R_1 = 80. This value is not correct because we already doubling WCET for task τ_1 for fault tolerant case. More correct value for R_1 is 40 time units, even in the worst case. Time period of 40 units is long enough for executing task τ_1 and to re-execute it in the presence of fault.

TABLE IV
TASK SET - CASE IIMOD

Task	Task characteristics				T_F=275	T_{Fmod}=143
	C_i	T_i	D_i	p_i	R_i	R_{imod}
τ_1	40	100	100	1	80	40
τ_2	25	175	175	2	145	90
τ_3	20	200	200	3	165	175
τ_4	25	300	300	4	275	285

To get more correct result we needed to modify Eq. (3) and we got new equation:

$$R_{i\,mod} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_{i\,mod}}{T_j} \right\rceil C_j + \left\lceil \frac{R_{i\,mod}}{T_{F\,mod}} \right\rceil \max_{j \in hp\,mod(i)} (C_j) \qquad (5)$$

The main difference between Eq. (3) and Eq. (5) is within the third addend. We needed to eliminate the possibility that fault can occurs within task τ_1. Because of that we have new task set for the third addend hpmod(i)=$\{\tau_j \in \Gamma_{mod} | p_j \geq p_i\}$ where is $\Gamma_{mod}=\{\tau_2,..., \tau_n\}$.

Appropriate recurrence relation for Eq. (5) is:

$$R_{i\,mod}^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_{i\,mod}^{n}}{T_j} \right\rceil C_j + \left\lceil \frac{R_{i\,mod}^{n}}{T_{F\,mod}} \right\rceil \max_{j \in hp\,mod(i)} (C_j) \qquad (6)$$

Initial and ending conditions are the same as for Eq. (4). Using Eq. (6) and algorithm for finding minimum T_F we realized application in MATLAB which can help us to analyze timing constrains of one fault tolerant real-time system.

We applied the realized application on the task set Case IImod and the results are shown in the last column of Table IV. It can be concluded that response time for τ_1 is 40 time units what is in accordance with our starting assumption. Also, value for T_{Fmod} = 143 is less then T_F = 275 what is good for one RTS. If minimum time between two consecutive faults, which real-time system can tolerate, is less then the RTS is more faults tolerant. Because of that the modified analysis gave us better result then original.

III. CONCLUSION

The modify RTA gave us more precise analysis results which show us that RTS can tolerate more faults then unmodified RTA was given. For the tasks set – case IImod (Table IV) value for T_F is reduced from 275 to 143 which is improvement of 48%. So, the main contribute of our paper is increasing number of faults which RTA considers during analysis. Now, for all that faults modify RTA can guarantees that RTS will be fault tolerant. Using modify RTA we get one more fault tolerant RTS. The modify RTA can be used for estimating the possibility of overcoming transient faults in one process control real-time system i.e. it can be concluded how much is one RTS fault tolerant.

ACKNOWLEDGEMENT

This paper is supported by Project Grant III44004 (2011-2014) financed by Ministry of Education and Science, Republic of Serbia

REFERENCES

[1] N. Nissanke, *Realtime Systems*, Prentice Hall, 1997.
[2] K. Juvva, "Real-Time Systems", Carnegie Mellon University 18-849b Dependable Embedded Systems ili http://www2.cs.cmu.edu/~koopman/des_s99/real_time/
[3] Nobuyasu Kanekawa, Eishi H. Ibe, Takashi Suga, Yutaka Uematsu, *Dependability in Electronic Systems: Mitigation of Hardware Failures, Soft Errors, and Electro-Magnetic Disturbances*, Springer, 2010.
[4] S. Đošić, M. Jevtić, "Planiranje zadataka u sistemu za rad u realnom vremenu sa redundansom u vremenu za prevazilaženje otkaza", Zbornik radova V simpozijuma industrijske elektronike, INDEL 2004, Banja Luka, pp. 146-149, novembar 2004.
[5] S. Đošić, M. Jevtić, "Analysis of transient fault tolerance in hard real-time systems with time redundancy", Facta Universitatis, Series: Automatic control and robotics, vol. 8, no 1, pp. 149-163, 2009.
[6] M. George de A. Lima and Alan Burns, "An Optimal Fixed-Priority Assignment Algorithm for Supporting Fault-Tolerant Hard Real-Time Systems", IEEE Trans. Computers, vol.52, no.10, pp. 1332-1346, Oct. 2003.
[7] F. Cottet, J. Delacroix, Z. Mammeri, "Scheduling in Real-Time Systems", John Wiley & Sons, 2002.