Analysis of Real-Time Systems Timing Constrains

Sandra Đošić and Milun Jevtić

Abstract - In this paper we analyze timing constrains of one fault tolerant hard real-time system with time redundancy. Our goal is to analyze possibility for overcoming transient faults, which are detected during tasks executions, using technique of executing task again or executing some alternative task. We created and presented in the paper program for estimation of possibility to overcome transient failure in one real-time system. On the basis of timing characteristics of real-time tasks and the value of redundant time we can find the value for minimum time between two consecutive faults which real-time system can tolerate.

Keywords - Real-time systems, Response time analysis, Fault tolerance.

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].The classical conception is that in a hard real-time system, the completion of an operation after its deadline is considered useless - ultimately, this may cause a critical failure of the complete system. A soft real-time system on the other hand will tolerate such lateness, and may respond with decreased service quality (e.g., omitting frames while displaying a video).

One of the goals during real-time systems designing process is to create predictable real-time systems. Analysis of real-time systems timing constrains is fundamental for design such systems. Designing predictable real-time systems is easier with the assumption that there is no fault during system execution. However, this fault-free assumption is, in fact, not realistic because "non-faulty systems hardly exist, there are only systems which may have not yet failed", [2]. So, if a fault occurs during real-time tasks execution then it is necessary to overcome that fault and satisfied all real-time tasks timing constraints.

Focus of our research is fault tolerant hard real-time systems and in this paper we will analyze timing constrains for such systems, [3]. We also wrote program for that analyses. Input data for program are timing characteristics of real-time tasks and the result is minimum time between two consecutive faults which real-time system can tolerate. Due to result of analysis we can conclude how much is one real-time system fault tolerant.

Sandra Đošić and Milun Jevtić are with the Department of Electronics, Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia, E-mail: (milun.jevtic, sandra.djosic)@elfak.ni.ac.rs.

II. Software realization of algorithm for analysis RTS timing characteristics

A. Response time analysis

One of the goals of our research is to the design predictable hard real-time systems. Response time analysis is one approach that has successfully been used to achieve this goal. The basis of response time analysis is Eq. (1) and more about that analysis can be found in [3].

$$R_i(T_E) = C_i + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i(T_E)}{T_j} \right\rceil C_j + \left\lceil \frac{R_i(T_E)}{T_E} \right\rceil \max_{\tau_k \in hpe(i)} \overline{C_k} \qquad (1)$$

We use response time analysis for set $\Gamma = \{\tau_1, \ldots, \tau_n\}$ of n real-time tasks, called primary tasks, that must be scheduled by the system in the absence of faults. Any primary task τ_i, in a set Γ, has a period T_i, a deadline D_i ($D_i \leq T_i$), and a worst-case execution time, C_i. Each primary task τ_i can have some alternative tasks $\overline{\tau_i}$ associated with it, [4]. Each alternative task represents some extra processing that is necessary to recover a task from a given faulty state caused by a fault. Any alternative task has a worst-case execution time, called worst-case recovery time, $\overline{C_i}$.

We also consider n different priority levels (1, 2, ... , n), where 1 is the lowest priority level. We denote the priority of primary task τ_i and alternative tasks $\overline{\tau_i}$ as p_i and $\overline{p_i}$, respectively. We also assume in the analysis that there is a minimum time between two consecutive fault occurrences, T_E.

The input parameters of this analysis are: the task attributes (T_i, D_i, C_i and $\overline{C_i}$), the primary task priorities (p_i) and the assumed value of T_E. The priorities of alternative tasks are assumed to be the same as their primary tasks $p_i = \overline{p_i}$.

If there is no faults in the system then the worst-case response time of task τ_i is the time necessary to execute τ_i and all tasks τ_j such that $p_j > p_i$. When faults are considered in the system, we have to include in the calculation of the worst-case response time of τ_i the time necessary to recover the faulty task. We use time redundancy for systems recovering, [5].

Since R_i appears on both sides of the Eq. (1), the solution can be obtained iteratively by forming a recurrence relation with $R_i^0 = C_i$. This iterative procedure finishes either when $R_i^{m+1} = R_i^m$ (the worst-case response time of τ_i

is found) or when $R_i^{m+1} > D_i$ (τ_i is considered unschedulable).

Fig. 1 illustrates possible scenarios of real-time tasks scheduling with different assumed value of T_E.

The first scenario, Fig. 1(a) presents scheduling of two periodic real-time tasks τ_1 and τ_2 when there is no fault in the system. System of these two tasks are schedulable i.e. both tasks execute before their deadlines, D_1 and D_2.

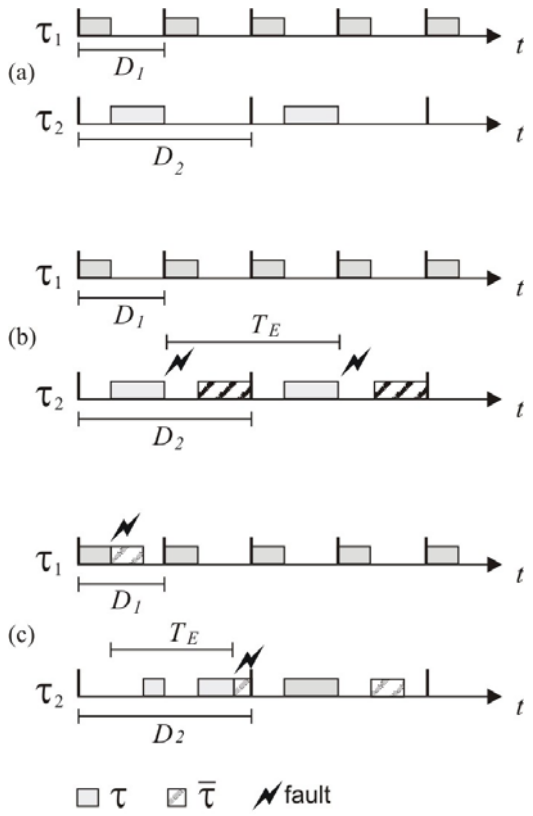


Fig. 1. Illustration of possible real-time tasks schedule when: (a) there is no fault; (b) value for T_E is long enough and real-time system is fault tolerant; (c) value for T_E is not long enough that real-time system stays fault tolerant

Fig. 1(b) presents scheduling of the same real-time tasks τ_1 and τ_2 when two faults occur in the system. Time between two consecutive faults T_E is long enough and real-time system can tolerate these faults. First fault occurs just a little bit before the end of tasks τ_2 execution. Real-time system overcomes this fault by executing task τ_2 again or executing alternative tasks with less or equal execution time as task τ_2. Second fault occurs again just a little bit before the end of tasks τ_2 execution. Time redundancy is enough to tolerate this fault too. Like before, when the first fault occurs, system overcomes fault by executing task τ_2 again or executing some alternative tasks.

Fig. 1(c) presents scheduling of the same real-time tasks τ_1 and τ_2 when two faults occur in the system. Now, time between two consecutive faults T_E 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_1 execution. Real-time system can overcomes this fault by executing task τ_1 again or executing alternative tasks with less or equal execution time as task τ_1. In this case, second fault occurs just a little bit before the end of tasks τ_2 execution. Now time redundancy is not enough to tolerate this fault. Systems starts procedure for overcoming fault by executing task τ_2 again but timing characteristics if tasks τ_2 cannot be satisfied and τ_2 missing its deadline. This is not acceptable in one hard real-time system, so in this case real-time system is not fault tolerant.

B. The Algorithm

Based on the Eq. (1) we realized algorithm for analysis real-time systems timing constraints shown in Fig. 2. Input data are number of real-time tasks n, task period T_i, worst-case execution time C_i, worst-case recovery time $\overline{C_i}$, task deadline d_i and task priority p_i. For these parameters algorithm have to check if the real-time system is fault tolerant. We considered that fault can occur during tasks execution and that is necessary to execute some recovery tasks for faults overcome. The goal of algorithm is to find minimum time between two consecutive faults which real-time system can tolerate.

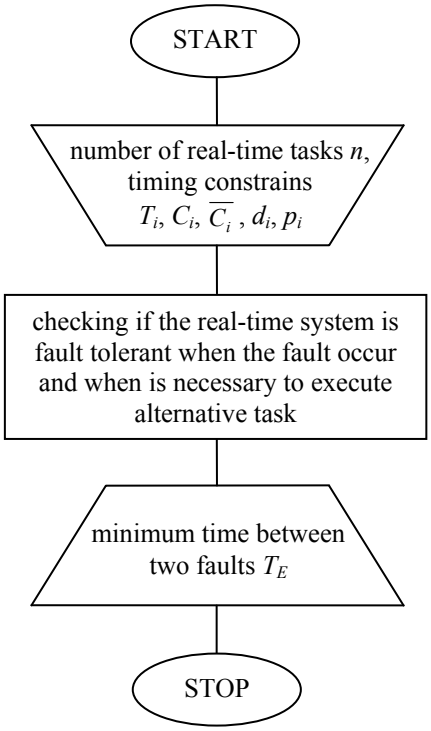


Fig. 2. Algorithm for analysis RTS timing constrains

Fig. 3 shows more detailed algorithm for analysis real-time systems constrains. Input data for shown algorithm are number of real-time tasks n, task period T_i, worst-case

execution time C_i, worst-case recovery time $\overline{C_i}$, task deadline d_i and task priority p_i, step (1) on Fig. 3. In the beginning we assume in the analysis that minimum time between two consecutive fault occurrence is $T_E = 1$ step (2) on Fig. 3.

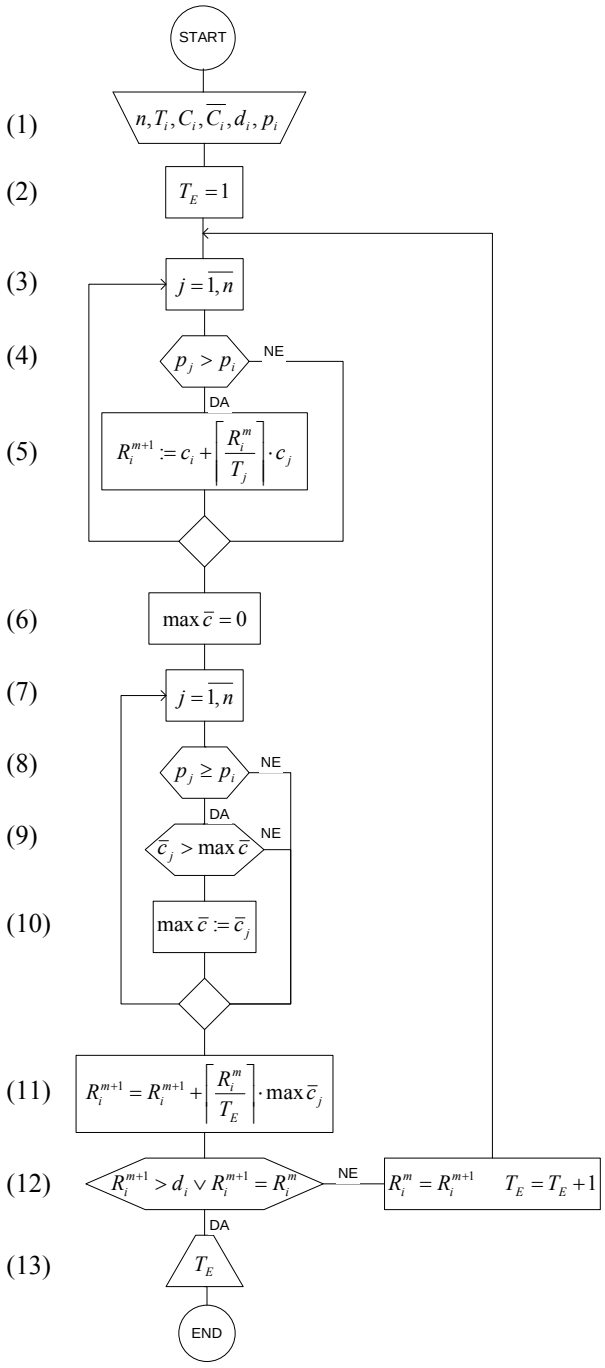


Fig. 3. More detailed algorithm for analysis RTS timing constrains

In the first algorithm loop, step (3), step (4) and step (5) on Fig. 3, we calculate the first and the second addend of Eq. (1). In this loop only task with higher priority then priority of task τ_i are important for us.

The second loop in algorithm, step (6) to (10) on Fig. 3, describe process of finding maximum worst-case recovery time from the tasks with equal or higher priority then priority of task τ_i.

Step (11) on Fig. 3 calculates the worst-case response time R_i for task τ_i. According to Eq. (1) this process is iterative and it finishes either when $R_i^{m+1} > d_i$ (τ_i is considered unschedulable) or when $R_i^{m+1} = R_i^m$ (the worst-case response time of τ_i is found), step (12) on Fig. 3. If the condition step (12) is true then we have output result T_E step (13) on Fig. 3. If the condition step (12) is false then we must increase T_E and continue iterative process until it is necessary.

Using algorithm shown on Fig. 3 we wrote code and generated .exe file "AlgFix.exe" which could be started from command line with command:
ALGFIX [<input_file>] [<output_file>].
As you can see from the above command, optionally the name of the input and output file could be written. If you don't write name for the input and output file then their standard name "AlgFix Input.txt" and "AlgFix Output.txt" are considered.

Input file is .txt format with parameters separate with space. In the first line, the number of real-time tasks n should be written. After that in the next n line we have to specify timing characteristics of n real-time tasks: period T_i, worst-case execution time C_i, worst-case recovery time $\overline{C_i}$, deadline d_i and task priority p_i.

Output file is also .txt format with parameters separate with space. The first line is required result T_E - minimum time between two consecutive faults which real-time system can tolerate. In the next n line are parameters R_i^m and R_i^{m+1} for each of n real-tasks.

C. Results of Software Realization

In order to prove the correctness of the realized algorithm and the whole program, we do a number of tests and two of them are shown on Fig. 4.

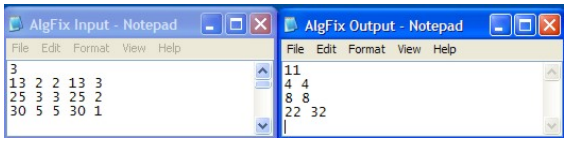
Fig. 4(a) presents input and output file for case I of three real-time tasks scheduling according with rate monotonic algorithm, [6], [7].

Timing characteristics for these three tasks are shown in Table I and inputs file "AlgFix Input" on Fig. 4(a). For these parameters, we started our program for analyses. Program considers that faults can occur in real-time system during task execution and that system recovers executing task again. Therefore, for this case worst-case recovery time is equal as worst-case execution time, $C_i = \overline{C_i}$.

The output file "AlgFix Output.txt" shows results of timing analyses. From that file, we can see that real-time systems can tolerate minimum time between two

consecutive fault occurrences of 11 time units. For $T_E = 11$ parameters are $R_1(11) = 4$, $R_2(11) = 8$ and $R_3(11) = 22$ and this is also shown in output file. For all three tasks we got that $R_i(11) < d_i$, for $i = 1, 2$ and 3, what means that all tasks finished before their deadlines. It can be concluded that system is schedulable.

(a)



(b)

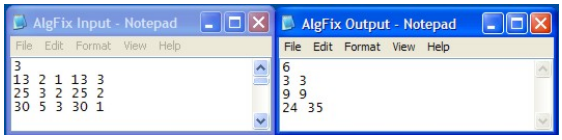


Fig. 4. Input and output file of realized software for
(a) case I - system recovers executing task again
(b) case II - system recovers executing alternative task

For $T_E = 10$ parameters are $R_1(10) = 4$, $R_2(10) = 8$ and $R_3(10) = 32$ and they are also shown in output file "AlgFix Output.txt". For task τ_3 we got that $R_3(10) > d_3$ what means that this task overflows its deadline, so the whole system is not schedulable.

TABLE I
REAL-TIME TASKS TIMING CHARACTERISTICS - CASE I

Task	Task characteristics				
	T_i,	C_i,	$\overline{C_i}$,	d_i,	p_i
τ_1	13	2	2	13	3
τ_2	25	3	3	25	2
τ_3	30	5	5	30	1

Table II presents manually obtained results for the same input parameters. If we compare the output file "AlgFix Output.txt" and Table II, it can be conclude that we got the same results, manually and software obtained.

TABLE II
RESULTS OF TIMING ANALYSES FOR CASE I

Task	$R_i(11)$	$R_i(10)$
τ_1	4	4
τ_2	8	8
τ_3	22	**32**

The second case presents real-time system that recovers from the fault executing some alternative tasks. Usually those tasks have less worst-case execution time then primary tasks, i.e. $\overline{C_i} < C_i$. This case is shown on Fig. 4(b).

Fig. 4(b) presents input and output file for case II of three real-time tasks scheduling also according with rate monotonic algorithm. Timing characteristics for these three tasks are shown in Table III and inputs file "AlgFix Input" on Fig. 4(b). For these parameters, we started our program for analyses. Program considers that faults can occur in real-time system during task execution and that system recovers executing alternative tasks whose worst-case recovery time is less then as worst-case execution time of primary task, $\overline{C_i} < C_i$.

TABLE III
REAL-TIME TASKS TIMING CHARACTERISTICS – CASE II

Task	Task characteristics				
	T_i,	C_i,	$\overline{C_i}$,	d_i,	p_i
τ_1	13	2	1	13	3
τ_2	25	3	2	25	2
τ_3	30	5	3	30	1

The output file "AlgFix Output.txt" shows results of timing analyses. From that file, we can see that real-time systems can tolerate minimum time between two consecutive fault occurrences of 6 time units. For $T_E = 6$ parameters are $R_1(6) = 3$, $R_2(6) = 9$ and $R_3(6) = 24$ and this is also shown in output file. For all three tasks we got that $R_i(6) < d_i$, for $i = 1, 2$ and 3, what means that all tasks finished before their deadlines. It can be concluded that system is schedulable.

For $T_E = 5$ parameters are $R_1(5) = 3$, $R_2(5) = 9$ and $R_3(5) = 35$ and they are also shown in output file "AlgFix Output.txt". For task τ_3 we got that $R_3(5) > d_3$ what means that this task overflows its deadline, so the whole system is not schedulable.

TABLE IV
RESULTS OF TIMING ANALYSES FOR CASE II

Task	$R_i(6)$	$R_i(5)$
τ_1	3	3
τ_2	9	9
τ_3	24	**35**

Table IV presents manually obtained results for the same input parameters. If we compare the output file "AlgFix Output.txt" and Table IV, it can be conclude that we got the same results, manually and software obtained.

III. CONCLUSION

In this paper, we presented program for analyzing timing constraints of real-time tasks in one real-time system. We considered that these tasks are schedule according with rate monotonic algorithm and that faults can occur during tasks execution. We also considered that real-time system recovers from faults executing task again (case I) or executing some alternative tasks (case II). In both

cases, we use time redundancy for systems recovery after faults. For these two cases, we do a number of tests and prove the correctness of the realized algorithm and the whole program.

We specially presented two cases of tree real-time tasks whose input parameters are almost the same, the only difference is value for worst-case recovery time. Case I presents real-time system who recovers from faults executing task again, so $C_i = \overline{C_i}$. Case II presents real-time system who recovers from faults executing some alternative tasks whose worst-case recovery time are less then tasks worst-case recovery time, i.e. $\overline{C_i} < C_i$. If we compare output results for case I and case II we can conclude that if worst-case recovery time is less than minimum time between two consecutive fault occurrences which systems can tolerate is also less. This reduction of parameter T_E indicates increasing real-time fault tolerance, what is good.

Realized program offers the possibility to analyze timing constraints of multiple real-time tasks very fast, much faster than manually obtained. From the output program result, we also got information about minimum time between two consecutive fault occurrences that systems can tolerate. This is important information from which we can conclude how much is one real-time system fault tolerant.

REFERENCES

[1] Nissanke, N.,*"Realtime Systems"*, Prentice Hall, 1997.

[2] Laprie, J.C., *"Dependability: Basic Concepts and Terminology"*, Springer-Verlag, 1992.

[3] Lima, G., Burns, A., *"An Optimal Fixed-Priority Assignment Algorithm for Supporting Fault-Tolerant Hard Real-Time Systems"*, IEEE Transaction on Computers, Vol. 52, No. 10, October, 2003, pp. 1332-1346.

[4] Johnson, B., *"Design Analysis of Fault-Tolerant Digital Systems"*, Addison-Wesley Publishing Company, 1988.

[5] Đošić, S., Jevtić, M., *"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, novembar 2004, pp. 146-149.

[6] Cottet, F., Delacroix, J., Mammeri, Z., *"Scheduling in Real-Time Systems"*, John Wiley & Sons, 2002.

[7] Juvva, K., *"Real-Time Systems",* Carnegie Mellon University, 18-849b Dependable Embedded Systems, or http://www.ece.cmu.edu/~koopman/des_s99/real_time/index.html