# Fault-tolerant Scheduling of Stateful Tasks in Uniprocessor Real-time Systems

Petr Alexeev[1], Pontus Bostrm[1], Marina Waldn[1], Mikko Huova[2], Matti Linjama[2] and Kaisa Sere[1]

[1]*Department of Information Technologies, Bo Akademi University, Turku Centre for Computer Science, Turku, Finland*

[2]*Institute of Hydraulics and Automation, Tampere University of Technology, Tampere, Finland*

Keywords: Real-time Systems, Deadlines, Recovery Block, Deadline Mechanism, Backwards-DM, Static Scheduling.

Abstract: The recovery block (RcB) approach is intended for masking software faults. This approach can be implemented for real-time systems by establishing primary and alternative versions for each task and providing a fault-tolerant preemptive schedule which masks possibly missed deadlines. Existing scheduling algorithms require rearrangement of tasks parameters during run-time. Applying these algorithms for stateful tasks which keep their state between releases is difficult. We defined two off-line scheduling algorithms: Backwards-Direct-Deadline-Monotonic (B-D-DM) and Earliest-Deadlines-as-Late as possible-Deadline-Monotonic (EDL-DM). These algorithms are based on existing algorithms used for periodic tasks: Backwards-DM, EDL and DM. The main advantage of our algorithms is the ability to calculate all parameters of the schedule off-line and implement RcB for stateful tasks. We propose a feasibility check for the resulting schedule. The proposed algorithm B-D-DM was implemented in a case study of a control system designed in Simulink.

## 1 INTRODUCTION

Hard real-time systems are used in many areas to control physical processes. Frequently these processes are potentially dangerous for humans or the environment. There is a need for ensuring safety by designs of such systems. Unfortunately high design complexity of modern real-time systems does not allow preventing all software design errors. In some cases software errors can cause serious accidents or even catastrophes.

Software design diversity implemented by a recovery block approach (RcB) is a widely used fault-tolerance method. In real-time systems this method can be implemented by the Deadline Mechanism, proposed in (Chetto and Chetto, 1989), which requires defining *primary* and *alternative* versions of the software and providing a scheduling algorithm. Examples of such algorithms are described in (Chetto and Chetto, 1989; Chetto, 1994; Ding and Guo, 2009; Han et al., 2003). These algorithms have assumptions, like:

- They should be able to quickly abort the execution of the primary task.
- Execution results of tasks do not depend on their previous states. This allows to omit execution of either primary or alternative task if it is not needed.
- Parameters of the scheduler, e.g., tasks priorities, should be quickly adjustable during real-time execution.

Unfortunately these assumptions do not hold for all real-time systems. Let us consider each assumption in detail.

Threads and processes allocate their own memory, e.g., stack, that cannot be deallocated immediately. In some cases threads use shared variables in a common address space. This implies the need for keeping a strict policy of allocation and deallocation of these variables to prevent memory leaks. This policy demands to deallocate some variables before a thread can be aborted. For this reason the process of task abortion takes some time. In some cases a task cannot be aborted.

Software modules can have internal memory (state) which affects on their execution results. This state is stored between their invocations and needs to be updated during each invocation of the module. The execution results of these modules depend on its previous states.

Some real-time execution platforms do not allow rearranging task priorities during run-time. For these cases only static (off-line) scheduling algorithms can be used.

In this paper we propose two fault-tolerant scheduling algorithms based on the Deadline Mechanism. Our algorithms are free from the described assumptions and exploit the Earliest Deadline as Late (EDL) as possible algorithm, proposed in (Chetto and Chetto, 1989). We applied our algorithms to a case study of the hard real-time control system used for hydraulics.

The rest of the paper structured as follows. The next Section presents some terminology of real-time software used in this paper. Detailed description of problems to be solved is presented in Section 3. This section also shows disadvantages of existing methods and the example of the case study used in the research. Proposed scheduling algorithms and feasibility check are presented in Section 4. In Section 5 a comparison of the proposed fault-tolerant scheduling algorithms with existing methods is presented. Section 6 summarizes the main results.

## 2 TERMINOLOGY

A real-time system consists of a set of preemptive *tasks* $\Gamma \triangleq \{\tau_1, \tau_2, \cdots, \tau_N\}$. Each task has its own *priority* $p_i$. An execution of the task $\tau_i$ with the index $j$ is called *job* or *task instance* $\tau_{ij}$.

The worst-case execution time (WCET) of the task defines maximum possible execution time for the task $C_i$. End-to-end execution of the job can take more time than $C_i$ because of preemption by high-priority tasks. In real-time systems each task is accompanied with a *deadline* $D_i$ - the maximum time duration allowed to the task for execution. The end-to end execution of a job is described by the following parameters:

- Release time (offset) $r_{ij}$. This is the time at which a job becomes ready for execution.
- Finishing (termiantion) time $f_{ij}$. This is the time at which a job finishes its execution.

If $\tau_i$ is requested with some fixed time interval this task called *periodic* with period $T_i$. For periodic tasks we can define some parameters as *absolute* and *relative*. Relative parameters are marked by capital letters and calculated for a one period of execution and use start time of the period as the starting point. Absolute parameters are marked with small case letters and calculated for all periods starting from some starting point.

Different scheduling algorithms can be used to arrange priorities and release times for tasks in a task set. If priorities are not changed during the execution time this scheduling algorithm called *off-line* or *static*. In this case priorities can be calculated off-line even before the compilation of the software code. *Dynamic* scheduling algorithms adjust tasks parameters according some rule during run-time.

For the set of periodical tasks *hyperperiod* or *planning cycle* is established $T_H = LCM\,(T_1, T_2, \cdots, T_N)$[1]. Periodical tasks are defined by tuples $(T_i, C_i, D_i)$.

Data transfers between tasks affects scheduling scheme. We assume that the Non-Preemptive Protool (NPP) is used for all possible data transfers between tasks. NPP provides overhead for large number of data transfers, but, as it is noted in Section 7.10 of (Buttazzo, 2011), it prevents both deadlocks and priority inversion.

Each hard real-time scheduling scheme has to be checked for feasibility. We say that schedule is *feasible* if it allow all tasks to complete before their deadlines.

## 3 PROBLEM DESCRIPTION

The execution result of task can be obtained at the moment of finishing time $F_i$. In the worst case $F_i = D_i$, in the best case $F_i \le C_i$ but it is safe to consider only $F_i = C_i$. For this reason the maximum time jitter of the execution result $J_i = D_i - C_i$. If $C_i \ll D_i$[2] this value will be big.

In the case of rate monotonic-based (RM-based) scheduling algorithms, like described in (Ding and Guo, 2009; Han et al., 2003), deadlines are assumed to be equal to periods $D_i = T_i$. For this reason $J_i = T_i - C_i$. Consider the following example: $C_i = 1$, $T_i = 100$. This implies that the execution result can be obtained at random time in the interval $[1, 100]$ and $J_i = 99$. This can be unacceptable if corresponding task performs interaction with peripheral devices, sensors or actuators.

Existing fault-tolerant scheduling algorithms usually assume tasks as invocations of stateless functions because execution results of such tasks depend on their current input data only. This allows, for example, omitting execution of alternative version of the task if corresponding primary version terminated successfully as proposed in (Chetto, 1994; Chetto and Chetto, 1989; Han et al., 2003; Ding and Guo, 2009).

Unfortunately, this approach is inapplicable for many data processing algorithms where some variables (memory blocks, delay blocks) are used to store

[1] The LCM abbreviation denotes least common multiple.

[2] The notation $A \ll B$ means that $A$ is much less than $B$.

information from previous invocations. An interesting example of such algorithm is a digital filter where delay blocks are essential. The stateful behaviour of tasks must be taken into account by the new fault-tolerant scheduling algorithm.

The invocation of stateful primary and alternative versions of functions cannot be omitted. Both primary and alternative versions of task should have the ability to be completed before the deadline if primary version completes within its WCET.

## 3.1 The Case Study

The need for the new scheduling algorithm was inspired by the case study of the controller of digital hydraulics described in publications of Linjama and Huova (Linjama et al., 2007). A detailed description of the case study and the implementation of the proposed fault-tolerant scheduling algorithms with experimental results is presented in (Huova et al., 2012).

This controller is a non-linear digital control system, which consists of a plant (hydraulics), sensors, actuators and a computing device with the software. The whole software was designed as a multirate Simulink model. ANSI C source code was automatically generated from this model. All software generation and scheduling parameters were placed inside of the model.

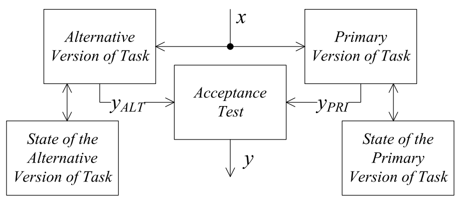
The model of the control system was decomposed into the following periodic real-time stateful tasks:

- Task 1. Sensors data acquisition. Period 1 ms.
- Task 2. Main Control Algorithm (MCA) and actuators interaction. Period 10 ms.

The MCA is a complicated and potentially unreliable part of the system. The RcB approach was proposed to ensure safety by masking possible software faults in the MCA. A simplified version of the MCA was designed as an implementation of the alternative version of MCA. The computation time for the primary version was about 1 ms, for the alternative version it was about 0.1 ms.

The specific of the hydraulics plant requires updating states of these actuators (on/off valves) periodically with minimal jitter. This requires minimizing the jitter for the output of task 2.

Existing fault-tolerant scheduling algorithms does not take into account the stateful behaviour of tasks (both versions of MCA should keep its own state between invocations) and do not allow to meet the requirement for minimizing of the jitter of its output. This forced us to propose the new fault-tolerant scheduling algorithm based on existing RcB approaches.



Alternative Version of Task
$x$
Primary Version of Task
Acceptance Test
$y_{ALT}$
$y_{PRI}$
State of the Alternative Version of Task
$y$
State of the Primary Version of Task

Figure 1: Recovery blocks approach for a stateful task.

## 4 THE PROPOSED APPROACH

The RcB approach requires decomposing each task $\tau_i$ into the *primary version* (PRI) $\tau_{PRIi}$, *alternative version* (ALT) $\tau_{ALTi}$ and the *acceptance test* (AT) $AT_i$ as presented in Figure 1. The scheduling algorithm has to provide the ability to complete both $\tau_{PRIi}$, $\tau_{ALTi}$ and $AT_i$ before the deadline $D_i$ if the real computation time for the $\tau_{PRIi}$ does not exceed $C_{PRIi}$. The states of both $\tau_{PRIi}$ and $\tau_{ALTi}$ are independent, so possible fail of $\tau_{PRIi}$ will not affect on the $\tau_{ALTi}$ and its state.

As shown in Figure 1, the $AT_i$ determines which of two outputs (from $\tau_{PRIi}$ or from $\tau_{ALTi}$) should be used for the resulting value of the whole RcB. This implies the following precedence constraint: $(\tau_{PRIi}, \tau_{ALTi}) \prec AT_i$. The order of execution of $\tau_{PRIi}$ and $\tau_{ALTi}$ is indifferent. For this reason it is safe to assume $\tau_{PRIi} \prec \tau_{ALTi} \prec AT_i$.

In order to follow the Deadline Mechanism (Chetto and Chetto, 1989) the priority of the $\tau_{PRIi}$ should be lower than priority of the $\tau_{ALTi}$. There are no restrictions to the priority of $\tau_{ALTi}$ relatively to the priority of the $AT_i$. We implemented sequential execution of $\tau_{ALTi}$ and $AT_i$ as one recovery (REC) task $\tau_{RECi}$. In case if $\tau_{PRIi}$ takes more computation time than expected it will be preempted by the $\tau_{RECi}$. A similar approach was used in (Ding and Guo, 2009; Han et al., 2003; Chetto and Chetto, 1989). The precedence constraint is the following:

$$\tau_{PRIi} \prec \tau_{RECi} \mid i = 1, 2, \cdots, N \tag{1}$$

We are going to consider two algorithms for recovery tasks scheduling. In Section 3 it was shown that deadlines allow defining upper limit of jitter range for task output result. Hence, deadline-based scheduling algorithms are appropriate.

## 4.1 Static EDL Algorithm

One of the algorithms, used in RcB implementation, known as Deadline Mechanism, is the EDL algorithm, defined in (Chetto and Chetto, 1989). This algorithm is based on Earliest Deadlines First (EDF) strategy and proposes to release tasks as late as possible while keeping their deadlines. It was shown in

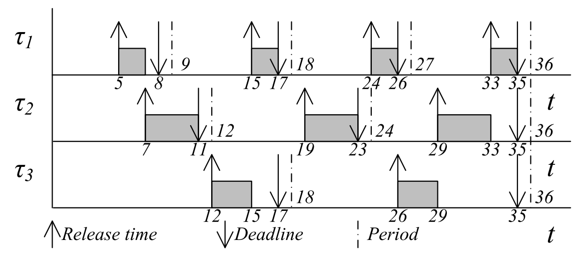


Figure 2: Example of schedule produced by the static EDL.

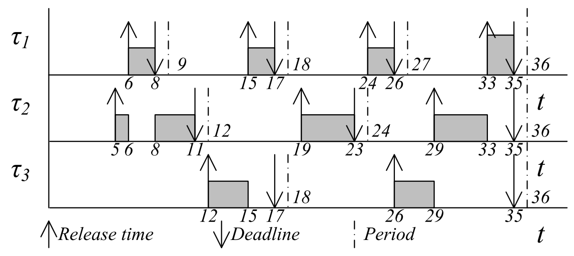


Figure 3: Example of schedule provided by the B-DM.

(EL Ghor et al., 2011) that it is possible to define static schedule for a periodical task set by EDL. This approach requires to calculate release times for all instances of all tasks within hyperperiod. In this case EDL is a static algorithm.

Within hyperperiod $T_H$ EDL schedule can be transformed to EDF by replacing time $t$ to $T_H - t$. In general case EDF defines preemptive schedule that requires tasks priorities rearrangement during the runtime. Unfortunately priorities modification during the run-time was impossible for our case study. For this reason we used non-preemptive version of the EDF that allows arbitrary selection of the priorities. This enables using immutable priorities and calculating off-line all release and finish times for all jobs. Consider the process of defining static EDL schedule.

This process starts by defining absolute deadline $d_{ij}$ for each task instance $\tau_{ij}$ within hyperperiod. According to deadlines absolute finishing times $f_{ij}$ for all task instances are also calculated. It is easy to see that $r_{ij} = f_{ij} - C_i$ because we selected non-preemptive EDL. The schedule is feasible if all tasks are released no earlier than the start of the period:

$$r_{ij} \geq (j-1)\,T_i \mid j = 1, 2, \cdots, \frac{T_H}{T_i} \tag{2}$$

Consider example of task set $\Gamma = \{\tau_i = (T_i, C_i, D_i)\}$ composed from tasks: $\tau_1 = (9,2,8)$, $\tau_2 = (12,4,11)$, $\tau_3 = (18,3,17)$. The hyperperiod will be 36. The resulting schedule is presented in Figure 2.

For the task with both shortest deadline and period ($\tau_1$) the relative release time $R_{1j}$ vary in different period. If $\tau_1$ will invoke recovery routines its corresponding primary task will have no more than 5 CPU cycles for the execution in the worst case (minimal relative release time $R_{1min} = 5$).

Another scheduling algorithm for recovery tasks can be defined to provide more execution time for the task with the shortest deadline. In (Ding and Guo, 2009; Han et al., 2003) Backwards-Rate-Monotonic (B-RM) algorithm is proposed to schedule periodical recovery tasks. This algorithm combines Rate-Monotonic (RM) priority arrangement[3] with as late as possible approach. Unfortunately, B-RM does not take into account deadlines; this does not allow minimizing finishing time jitter. For this reason we used static Backwards-Deadline-Monotonic (B-DM) algorithm.

## 4.2 Backwards-Deadline-Monotonic Algorithm

B-DM algorithm requires to use DM priorities arrangement[4] and to schedule tasks as late as possible keeping their deadlines at the same time. It requires to calculate release and finishing times for all task instances within hyperperiod. For all tasks without taking into account possible preemption these parameters can be calculated as $r_{ij} = (j-1)\,T_i + D_i - C_i$, $d_{ij} = r_{ij} + C_i$. For the task with the highest priority these parameters are correct, but for tasks with lower priorities release times needs to be corrected because of the preemption from tasks with higher priority. For this reason we used $\mathcal{B}$ - a set of CPU cycles occupied by jobs. The schedule is calculated by iteratively updating both $\mathcal{B}$ and release time for each job starting from the task with the highest priority. The feasibility rule for the schedule is the same as for EDL defined by (2).

Consider schedule produced by the B-DM algorithm for the example from the Subsection 4.1. The resulting schedule is presented in Figure 3 which is similar to the schedule presented in Figure 2. For the task with both shortest deadline and period ($\tau_1$) the relative release time $R_{1j}$ does not vary in different period.

Minimum release times are the following: $(R_{1min}, R_{2min}, R_{3min}) = (6,5,8)$. For the same task set EDL provides $(R_{1min}, R_{2min}, R_{3min}) = (5,5,8)$. For this example B-DM comparing to EDL allows to get bigger values of minimal relative re-

[3]Shorter period implies higher priority, priorities not changed during the run-time.

[4]Shorter relative deadline implies higher priority, priorities not changed during the run-time.

lease times. This property does not hold for all task sets. For example, for the task set $\Gamma = \{\tau_1 = (9,2,8), \tau_2 = (12,4,10), \tau_3 = (18,3,15)\}$ EDL provides $(R_{1min}, R_{2min}, R_{3min}) = (4,5,8)$, while B-DM provides $(R_{1min}, R_{2min}, R_{3min}) = (6,4,8)$.

B-DM is preemptive algorithm, so it requires to establish task priorities by DM rule. Figures 2 and 3 illustrates that for both B-DM and EDL there is no idle time between each task release and its deadline regardless if the preemption is used or not. If there would be any idle time in this interval, the task can be released later which contradicts with EDL definition.

## 4.3 Resulting Schedule of Primary and Recovery Tasks

Both static EDL and B-DM can be used to define release times for recovery tasks, while primary tasks should have lower priorities to ensure their preemption. We define *safety level* (SL) to be the lowest possible priority level for recovery task.

Precedence constraints (1) can be ensured by defining deadlines for primary tasks to be no higher than release times of corresponding recovery versions. As release time for each task instance of $\tau_{RECi}$ can vary between their periods, the final deadline for correspondent primary version $\tau_{PRIi}$ can be calculated as minimal relative release time $R_{RECiMIN}$ of $\tau_{RECi}$. The DM algorithm can be used to define priorities of primary tasks according to their deadlines. All these priorities have to be lowered below SL.

The final scheduling algorithm is formed by the combination of B-DM or static EDL algorithm for recovery tasks and DM algorithm for primary tasks. The scheduling scheme in this case is defined by priorities for all tasks and release times for recovery tasks. All these parameters can be calculated off-line, hence the complete algorithm is static. We defined fault-tolerant algorithm based on B-DM as Backwards-Direct-Deadline-Monotonic (B-D-DM). The similar algorithm based on static EDL we defined as EDL-DM.

Each task set in this case is defined by tuple $(T_i, C_{RECi}, C_{PRIi}, D_i)$. The whole task set is presented by $\Gamma_{PRI} = \{\tau_{PRIi} = (T_i, C_{PRIi}, R_{RECiMIN})\}$ and $\Gamma_{ALT} = \{\tau_{ALTi} = (T_i, C_{ALTi}, D_i)\}$. Set $\Gamma_{ALT}$ is scheduled by B-DM or EDL, while set $\Gamma_{PRI}$ is scheduled by DM.

The example of the schedule, provided by B-D-DM algorithm for the task set $\Gamma = \{(30,4,8,25), (60,8,16,55)\}$ is presented in Figure 4.

Feasibility $\Gamma_{ALT}$ can be ensured by condition (2). This feasibility does not depend on $\Gamma_{PRI}$ because priorities of all tasks $\tau_{PRIi}$ are below SL. Feasibility of

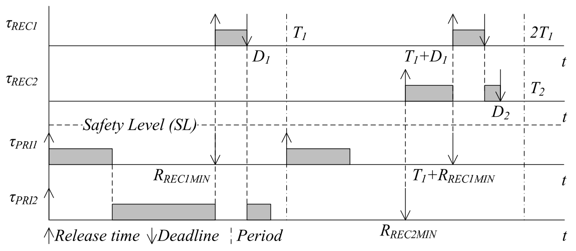


Figure 4: Example of the schedule provided by the B-D-DM.

$\Gamma_{PRI}$ can be ensured by feasibility check for DM proposed in (Audsley et al., 1991). The algorithm of this check is based on the response time analysis. Figure 4.17 in (Buttazzo, 2011) contains implementation of this algorithm.

## 5 DISCUSSION

The contribution of this paper is the combination of existing algorithms EDL, DM and B-DM for ensuring safe execution of periodical stateful tasks by the RcB implementation with the following characteristics:

1. Priorities for primary and alternative tasks are immutable and assigned by the Deadline Mechanism.
2. Primary and alternative tasks have isolated states.
3. Precedence constraints are ensured by assigning deadlines for primary tasks equal to release times of alternative tasks.
4. The resulting schedule ensures execution of all alternative tasks, all primary tasks are executed if all of them meet their deadlines.

A deadline miss by primary task is masked by using execution results of corresponding alternative task. The moment of switching from the output of the primary task to the output of the alternative task depends on the acceptance test.

In order to increase the reliability of the approach alternative tasks need to be reliable. Formal verification tools can be used to prove absence of certain problems such as functional errors (the program computes the wrong result), run-time exceptions, non-termination of loops and recursion or absence of deadlocks. Additionally safe WCETs for all recovery tasks can be estimated. This allows defining designs that can be considered as safe if there are no hardware faults.

Algorithm B-D-DM was adapted to the case study described in Subsection 3.1 and successfully examined with the workbench. Experimental results are presented in (Huova et al., 2012).

The following disadvantages have been discovered concerning the proposed scheduling algorithms.

1. A deadline miss of a primary task can cause other primary tasks to miss their deadlines.
2. Both approaches requires redundancy in the program memory and data memory.
3. A hardware fault can cause the system to be unsafe.

Despite of highlighted disadvantages the EDL-DM and B-D-DM algorithms can ensure safety of complex software designs and can be relatively easily implemented even with design tools like Simulink.

## 6 CONCLUSIONS AND FURTHER RESEARCH

In this paper we proposed combining existing scheduling algorithms B-DM, EDL and DM to extend the Deadline Mechanism implementation of the RcB approach to periodical stateful tasks, widely used in practice, e.g. digital filters. These type of systems can be difficult for various existing fault-tolerant scheduling algorithms. As the result we defined two fault-tolerant off-line scheduling algorithms: B-D-DM and EDL-DM.

Both proposed algorithms assign all scheduling parameters of alternative tasks according EDL or B-DM algorithms while primary tasks are scheduled according to DM rule. These algorithms take into account task deadlines that allow specifying the time jitter for each task. The parameters of the resulting schedule allow both primary and alternative tasks to be completed for each period. We considered details of both EDL and B-DM algorithms to determine all parameters of the final schedule and feasibility check for alternative versions of tasks. The feasibility of the primary tasks can be checked by a known algorithm.

The results of this feasibility checks are safe but too pessimistic. The improvement of this algorithm requires taking into account additional parameters of tasks and defining additional properties of B-D-DM and EDL-DM algorithms. This improvement goes beyond of the scope of this paper.

## ACKNOWLEDGEMENTS

The work has been partially funded by the EFFIMA program coordinated by FIMECC Ltd. (Finnish Metals and Engineering Competence Cluster) via project Digihybrid, as well as partially funded by the Academy of Finland via project EDiHy.

## REFERENCES

Audsley, N. C., Burns, A., Richardson, M. F., and Wellings, A. J. (1991). Hard real-time scheduling: The deadline-monotonic approach. *Proc IEEE Workshop on Real-Time Operating Systems and Software*, pages 1–6.

Buttazzo, G. C. (2011). *Hard Real-Time Computing Systems*. Springer US.

Chetto, H. (1994). Guaranteed deadlines with dynamic recovery blocks in distributed systems. In *Proceedings: Sixth Euromicro Workshop on Real-Time Systems, 1994*, pages 199–204.

Chetto, H. and Chetto, M. (1989). Some results of the earliest deadline scheduling algorithm. *IEEE Trans. Softw. Eng.*, 15(10):1261–1269.

Ding, W. and Guo, R. (2009). An effective RM-based scheduling algorithm for fault-tolerant real-time systems. In *International Conference on Computational Science and Engineering CSE '09*, volume 2, pages 759–764.

EL Ghor, H., Chetto, M., and Chehade, R. H. (2011). A real-time scheduling framework for embedded systems with environmental energy harvesting. *Comput. Electr. Eng.*, 37(4):498–510.

Han, C.-C., Shin, K. G., and Wu, J. (2003). A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults. *IEEE Trans. Comput.*, 52(3):362–372.

Huova, M., Ketonen, M., Alexeev, P., Bostrm, P., Linjama, M., Waldn, M., and Sere, K. (2012). Simulations with fault-tolerant controller software of a digital valve. In *Proceedings of the Fifth Workshop on Digital Fluid Power - DFP12, Tampere Finland*, pages 223–242.

Linjama, M., Huova, M., Bostrm, P., Laamanen, A., Siivonen, L., Morel, L., Walden, M., and Vilenius, M. (2007). Design and implementation of energy saving digital hydraulic control system. In *Vilenius, J. & Koskinen, K.T. (eds.) The Tenth Scandinavian International Conference on Fluid Power, May 21-23, 2007, Tampere, Finland, SICFP'07*, pages 341–359.