Comparison of Generally Applicable Mechanisms for Preventing Embedded Event-Driven Real-Time Systems from Interrupt Overloads

Josef Strnadel
Faculty of Information Technology, IT4Innovations Centre of Excellence
Brno University of Technology
Brno, Czech Republic
strnadel@fit.vutbr.cz

***Abstract*—This paper is related to event-driven embedded systems whose function can be divided into the real-time (RT) and non-RT parts. Such a system must be designed so that all predetermined timing constraints are met at runtime, even in adverse conditions such as an excessive rate of events caused by interrupts. In other words, the RT part is expected not to fail because of its criticality. In the paper, the interrupt overload problem (IOV) is introduced, then typical generally applicable solutions to the predictability part of the problem are presented along with our adaptive hardware/software mechanism for preventing the RT part from the same type of the problem; finally, the solutions are compared to summarize their attributes from the RT point of view.**

***Keywords*-event-driven; real-time; system; interrupt; overload; timing disturbance; predictability; problem; management; adaptation; prevention**

I. Introduction

This article is devoted to embedded, event-driven *real-time* (RT) systems, main characteristics of which can be summarized as follows:

- they have very limited computational resources (e.g., comparing to personal computers), but are able to comply criteria such as low power consumption, low weight, small size etc.,
- occurence of their input stimuli is signaled by events,
- they must react to events both correctly and on-time, according to their specification [1].

In the paper, it is supposed that RT properties are guaranteed by an *RT operating system* (RTOS) constructed to facilitate the design of RT applications [2], [3]. Hence, an RT application is supposed not to fail because of its criticality. If a failure of the system is not acceptable then (at least) *fault hypothesis* (utilized to specify assumptions about types and frequency of faults the system must be resilient to) and *load hypothesis* (for specifying assumptions about the peak load induced by the environment) must be specified [4].

This paper abstracts from the latter hypothesis (not reducing its weight nohow) and focuses just on problems related to the first-mentioned hypothesis and its typical solutions in the area of embedded, event-driven RT systems.

A. Formulation of the IOV Problem

Basically, two approaches to an event detection are possible in event-driven systems:

1) *polling-loop* based detection for which it is typical that a special, event-related flag is continuously tested by a CPU in order to detect whether an event has occured or not,
2) *interrupt* (INT) based detection main advantage of which is that no CPU time is consumed w.r.t. an event until an event-related INT is triggered.

In relation to 2) it should be noted that each (enabled/unmasked) INT means that an extra CPU time and a memory space are required to store the CPU context (typically consisting of registers such as program counter, status word etc.) of unfinished work being executed by the CPU (and being interrupted) due to the occurence of the INT; after the context is stored, the corresponding INT *service routine* (ISR) starts to be executed by the CPU. After the ISR is completed (and if there are no pending INTs at the moment), the context is restored back in order to resume the CPU execution being interrupted due to the INT triggering.

Since the CPU executes the code of an ISR prior to the main program loop, occurence of each enabled INT delays the loop for a certain time (let it be denoted by $t_{INTover}$) needed to process the CPU context and service the INT. If it holds $t_{INT} \leq t_{INTover}$, where t_{INT} is the time between successive occurrences of INTs then no CPU time remains to execute the main program loop because of the excessive *INT interarrival rate* ($f_{INT} = t_{INT}^{-1}$). It should be emphasized there that – excluding an ISR code – any code such as user/RT tasks/functions and/or RTOS kernel code is executed within the main program loop. Consequently, due to the principle of the INT-based event detection mechanism, a system may stop working correctly or collapse suddenly as f_{INT} increases. This is typically denoted as the *interrupt overload* (*IOV*) problem, seriousness of which grows with the criticality of the main program loop. Since such failures of the system are undesirable, a critical system must be designed so it may never give up to operate correctly.

II. Solutions to the IOV Problem

Mechanisms for softening impacts implying from the IOV problem can be classified according to the sub-problem they solve, i.e., the *timing disturbance* problem implying mainly from the disturbance due to soft RT tasks and the priority inversion phenomena [8], [9], [10] and the *predictability* problem originating from system's inability to predict when a new INT is going to be triggered [11], [12], [13].

Since impacts of the timing disturbance problem can be minimized using special instruments such as common (joint) ISR/task priority space [14] or resource access protocols [2], effects of the predictability problem cannot be minimized in such an easy way due to its aperiodical nature/origin. Thus, the paper is devoted just to the solutions of the latter (predictability) problem.

A. State of the art

Solutions to the predictability problem such as [8], [11], [12], [13], [15] are typically designed to bound t_{INT} (or, f_{INT}). In [11] so-called *interrupt limiters* (*ILs*) are designed to prevent RT systems from the problem – they are divided into the two types: *software* (SW) limiters (*SILs*) and *hardware* (HW) limiters (*HILs*). Typical parameters w.r.t. SILs are summarized in the Tab. I. SILs can be classified [11] to the *polling*, *strict* and *bursty* sub-types (for their overheads, see Fig. 1), principles of which are outlined in the next.

Polling SIL (Fig. 1a) checks periodically (each $t_{arrival}$ units, at the t_{poll} cost) whether an event flag – being adjusted somehow – is set or not (for the purpose, either a special on-chip timer able to produce an INT (t_{tmr}, t_{exp}) is typically utilized, but it can be replaced by a well-tuned loop); if the flag is set then a special task corresponding to the event is started (t_w) and INTs become disabled for further $t_{arrival}$ units of time.

Drawback of the (polling) principle implies directly from its active waiting construction (this leads to an extra CPU load – instead of switching the CPU to a sleep-mode or so, the CPU is consumed although no event occurs); it can be removed e.g. by the *strict SIL* principle (Fig. 1b) utilizing

Table I
Parameters of SW interrupt limiters, SILs [11]

Parameter	*Description*
t_{poll}	time needed to test an event flag
t_{ictx}	overhead of saving/restoring the ISR context
t_{flip}	time needed to disable/enable INTs
t_w	ISR execution time
t_{adj}	overhead of adjusting and start a timer
t_{exp}	overhead of timer expiration service
t_{cnt}	time to increment the INT counter value and test if it is below the threshold value + overhead of clearing a counter
t_{tmr}	time to expire a timer
t_{cri}	time to disable INTs after t_{INT} drops below the $t_{arrival}$ value; during the time, unlimited number of INTs can occur to overload the system

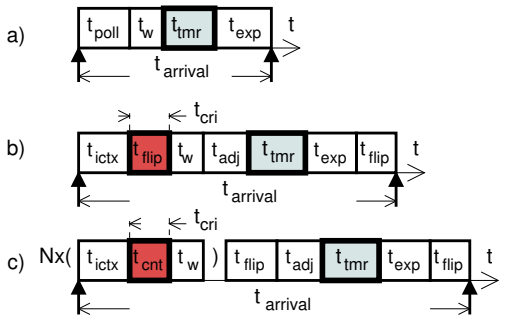

Figure 1. Overheads w.r.t. the (a) polling, (b) strict and (c) bursty SIL principles. The light-blue (t_{tmr}) boxes represent an action running in parallel with the CPU, red boxes bound the t_{cri} intervals and white boxes represent executions performed by the CPU

the ISR prologue – being executed at the end of the context switch w.r.t. an INT (t_{ictx}) – to disable any further INT (except those from timers) at the t_{flip} cost. After the INT is serviced (t_w), it configures a one-shot timer (t_{adj}) to expire after t_{tmr} units (the expiration cost is t_{exp}). After the expiration, INTs are re-enabled (t_{flip}) to let a further INT to be processed within the consecutive $t_{arrival}$ period. Main disadvantages of the approach can be seen in the following facts: i) INTs are practically doubled as each INT request triggers the (further) INT utilized to signalize the expiration and ii) INTs are disabled each time an INT is triggered, leading to the degradation of reactivity of an RT system.

The disadvantages can be minimized using the *bursty SIL* mechanism (Fig. 1c) being configured by the *maximum arrival rate* of INTs ($f_{arrival} = 1/t_{arrival}$) and the *maximum burst size* (N) parameters; the idea of the mechanism is to disable INTs after a burst of N INT requests (where $N \geq 2$) rather than after each request (the strict SIL behavior) – a special counter is needed to count (t_{cnt}) the number of triggered INTs (to be serviced within the $t_{arrival}$ interval) until it reaches N; then, INTs are disabled (t_{flip}) to be re-enabled again in the new $t_{arrival}$ period; for that purpose, a timer must be configured again (t_{adj}) to expire (t_{tmr}) at the cost t_{exp} and then, are INTs re-enabled (t_{flip}). This approach represents the actual state of industrial practice such as AUTOSAR [15]. Although the CPU-load overhead to the INT-throughput ratio can be well-tuned comparing to the strict SIL, an extra CPU is (still) needed especially to compare t_{cnt} to N after an INT occurs as well as to service the start/end of the $t_{arrival}$ period and overload of t_{cnt}.

The main advantage of the SIL approaches can be seen in the fact that they can be easily implemented on the platform they are going to protect. However, since they are based on SW instruments it may happen an extra CPU time they need is consumed at expenses of RT tasks in such a way that timing constraints of the tasks cannot be met. This drawback can be removed using the latter (HIL) solution to the IOV problem, but at the price of adding an extra hardware for that

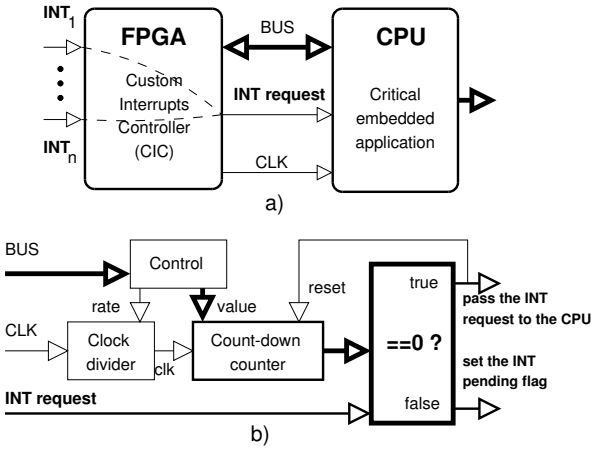

Figure 2. Custom Interrupts Controller (CIC) proposed in [8]

purpose. For example, in [8] it is supposed INT requests are processed (by a special CIC hardware being implemented for that purpose in an FPGA) before they are forwarded to a device the critical part of an embedded application runs on (see Fig. 2a). Herein, the HIL guarantees that at most one INT is directed to the device within the $t_{arrival}$ interval (i.e., it is a HW analogy to the strict SIL solution). For that purpose, a simple circuitry (Fig. 2b) is utilized in the HIL; the circuitry is designed to limit f_{INT} to a predefined, fixed $f_{arrival}$ rate – as the rate is fixed, such a solution to the IOV problem is denoted as "static" (later in this article). Because of its simplicity, the solution is widely utilized in many embedded applications.

Another, more complex approach to HIL can be found e.g. in [13], where an I/O interface is assigned its own *Real-Time Bridge* (RTB) able to catch all traffic to deliver it predictably according to both the system state and actual scheduling policy. However, the approach is applicable just to high-performance, PC-based systems (it was tested on the 1Ghz Intel Q6700 quad-CPU platform) able to communicate via the PCI(e) bus.

B. Proposed Solution

At the moment, it can be concluded that i) the IOV problem has not been solved yet and ii) existing solutions to the problem are either limited to solving one of the timing disturbance and predictability problems, they are too complex for (limited) embedded realizations, they require significant modifications and/or extensions of available industrial components or they inherently worsen schedulability of RT tasks as they increase the CPU utilization factor.

We have decided i) to take an advantage of existing [13] and joint task scheduling, e.g. [8], ideas, but ii) to modify and extend related concepts in order to move their applicability from the fixed, e.g. PCI(e)/PC-related, instruments to general-purpose, platform-independent ones, commonly present in most of recent embedded platforms. Our modification/extension of the existing concepts as well as contribution to solving the IOV problem can be seen in the design of a highly-efficient, but simply to implement load-monitoring based mechanism (with its own, generally applicable protocol and interface) able to adapt the INT management to the actual load of the critical part of an RT system.

Since crucial principles and attributes of our solution to the IOV problem have been already published e.g. in [5] being focused to the theoretical analysis of the solution, [6] giving a summary of its realization overheads and [7] presenting its architectural details, this paper concentrates maily to comparison of our solution to the others rather than to repeat the published work. However, main principles of our solution are summarized below to increase readability of the paper.

To make our solution adaptive, it has been based on a unit for signaling the load and a unit for monitoring the load; actually, the first unit is realized by an MCU utilized to execute a critical (SW) part of the system and to produce the `MON_PRI` to `MON_SLACK` signals[1] and the second (HW) unit is realized by an FPGA designed to monitor the signals to protect the MCU from consequences implying from the IOV problem (see Fig. 3). To minimize negative impacts to RT properties the units operate in parallel.

Key features of our concept are summarized below:

- the FPGA is able to adapt the INT service rate to the actual SW/MCU load (as a consequence, forwarding of INTs to the MCU is avoided if their related services are of low-priority and SW is overloaded; moreover, idle intervals in the SW execution can be utilized to service significantly higher number of INTs during MCU underload comparing to existing IOV approaches),
- the MCU running the (critical) SW is not disturbed by INTs with low-priority services
- the minimal forwarding delay is guaranteed for the highest-priority INT(s). From the RT point of view it is advantageous that load-related computations are done at the FPGA side as a low-cost function of monitoring signals being produced by the MCU – the saved CPU time can be utilized for useful, e.g. RT, SW executions.

[1]for more details to the signals, please refer to [5], [6]

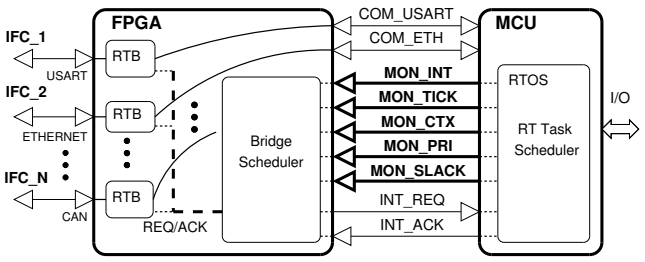

Figure 3. Schema/interface of the proposed solution to the problem [5]

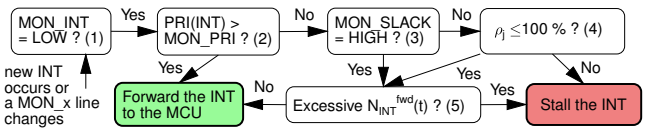

Figure 4. Operational principle of the proposed adaptive HIL solution

The operational principle of our concept is outlined in Fig. 4; comments to the states (1) to (5) follow:

1) An INT can be forwarded[2] to the MCU if no ISR is being executed at the moment, which is the necessary condition (1) for the forwarding if the ISR nesting is disallowed at the SW level – then, it makes no sense to forward an INT to the MCU during the execution. If the nesting is allowed then the result of (1) is not reflected.
2) An INT is forwarded to the MCU when the priority of its service task is higher than the running-task priority (2); no futher condition must be evaluated/fulfilled before the forwarding.
3) An INT can be forwarded to the MCU if the running-task priority is below the hard-priority level or the idle task is running (3); the forwarding is conditioned by (4) and/or (5).
4) An INT (INT_j) can be forwarded to the MCU only if there is enough CPU time to process it along with all uncompleted hard-priority executions. This is possible if the corresponding CPU-load factor (ρ_j) is below 100% for the uniprocessor system; the forwarding is conditioned by (5).
5) If the maximum number of INTs serviceable along with hard-priority executions could not be exceeded by forwarding a new INT request to the MCU then the forwarding is started.

III. Comparison of Solutions to the IOV Problem

Properties of our adaptive solution to the IOV problem were compared to attributes of embedded solutions mentioned in II-A. The comparison was made in the form of statistics being vizualized in the Fig. 5 to Fig. 7. Data for the comparison were collected for the following (13) CPU-utilization factors of the main-loop (U_{main} [%]) formed of hard tasks only[3] in the uniprocessor (1-CPU) environment: 0.1, 0.25, 0.5, 1, 2.5, 5, 10, 25, 50, 75, 90, 95, 98. All related experiments were performed under the following parameters (see Tab. I): t_{poll} = t_{flip} = .8 μs, t_{cnt} = t_{adj} = 1.6 μs, t_{ictx} = t_{exp} = 2.4 μs, t_w = 10 μs; the ARM® Cortex™ operated at the 125MHz bus-clock rate with caches disabled. For each of the U_{main} values, an experiment has been repeated 1000 times, where each repetition took 10 hyperperiods (major cycles) of hard tasks for the corresponding U_{main}.

In the Fig. 5, it can be seen that the IOV prevention techniques (horizontal axis) denoted as a) – f) need more CPU time (vertical axis) than our adaptive solution denoted there as g) – especially, this holds for higher $f_{arrival}$ and f_{INT} rates. In the extreme case, the amount of CPU time required by a–f is greater than the CPU time being available for the purpose (i.e., than 100 % herein) because they have no information about the actual CPU load and consequently, about the CPU time that remains (i.e. that is free) at the moment. In the case of our architecture (g) the CPU load at the MCU/SW side is monitored by our FPGA-based HIL to prevent the CPU from the overload, so the amount of CPU time required both to execute hard-level tasks and service INTs forwarded to the MCU never exceeds the maximum amount of available CPU time. let it be noted there that the requirements changes and are limited according to the actual CPU load, i.e., they decrease (increase) with the increasing (decreasing) value of U_{main}.

In the Fig. 6, the approaches a) – g) are compared according to the ratio of the number of INTs they were able service and the CPU load required for the servicing.

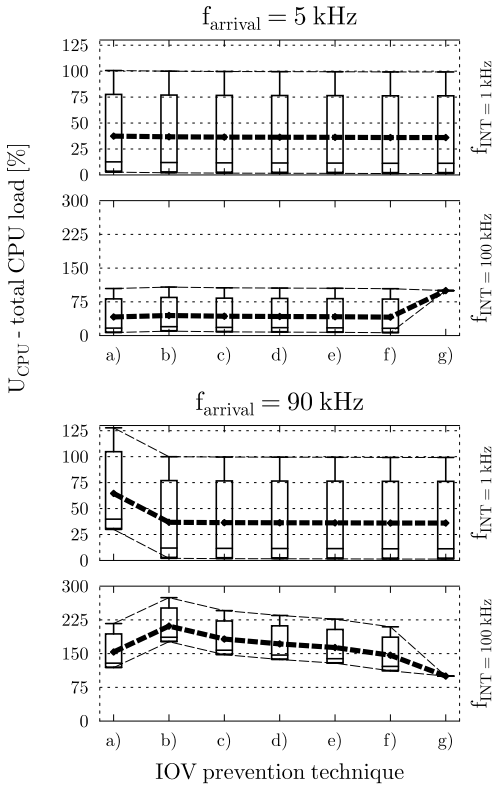

Figure 5. Impact of $f_{arrival}$ and f_{INT} rates to the total CPU load required by those IOV prevention techniques: a) polling SIL, b) strict SIL, c) bursty SIL w. burst=2, d) bursty SIL w. burst=4, e) bursty SIL w. burst=16, f) static HIL, g) proposed adaptive HIL

[2]after further conditions (2) – (5) are evaluated for making the decision about the forwarding

[3]inclusion of soft tasks would have no impact to the results

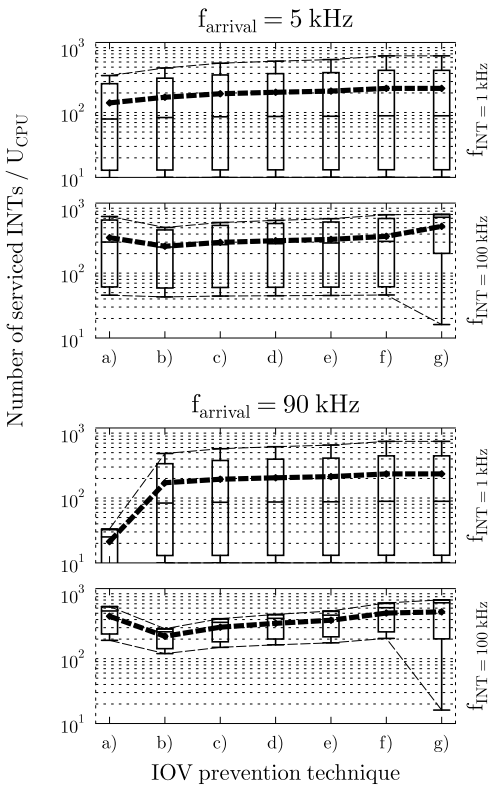

Figure 6. Impact of $f_{arrival}$ and f_{INT} rates to the number of serviced INTs per total CPU load required to service the INTs by the IOV prevention techniques.

In other words, it is illustrated there how many INTs were serviced by the approaches per one unit (i.e., 1 %) of the U_{CPU} load.

For the lowest U_{main} values (such as 0.1 %), it can be seen that our approach is able to adapt its $f_{arrival}$ rate to service more INTs per U_{CPU} than the other approaches – this is because it is able to detect and then consume any idle/slack time in the SW execution to service INTs that exceed the $f_{arrival}$ rate being fixed for the approaches.

Next, it can be observed there that for high U_{main} values (such as 98 %) our approach is able to adapt its $f_{arrival}$ rate again, but to service less INTs per U_{CPU} than the other approaches because the CPU time is almost exhausted by hard-level executions of tasks. The approaches a) – f) are not able to adapt their $f_{arrival}$ rates to the situation, so they forward the same INT traffic to the CPU as they have forwarded for low U_{main} values. As the number of INTs they forward is greater comparing to our approach, their ratio is greater too, but at the price of overloading the system because of the excessive INT rate (f_{INT}).

The last figure (Fig. 7) is utilized to vizualize that (and how) our IOV solution is able to adapt itself to changes of f_{INT} and U_{main}. The figure is divided into 3 parts, each dedicated to a particular f_{INT} range (horizontal axis)

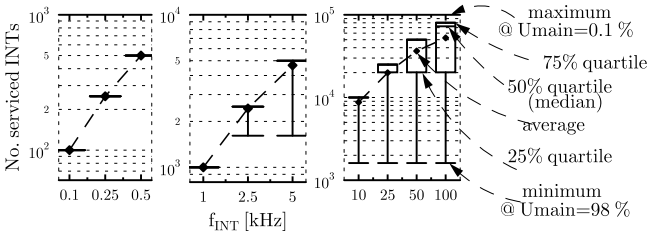

Figure 7. Impact of the INT rate (f_{INT}) and the main-loop CPU load (U_{main}) to the number of INTs that can be serviced by the proposed IOV solution in the system underload situation (in which timing constraints of hard tasks are guaranteed).

because the number of serviced INTs (vertical axis) differ a lot for the ranges. Alike in the case of Fig. 6 it can be seen there that for high U_{main} values, our approach is able to lower its $f_{arrival}$ and INT service rates to the (minimum) value (in the figure, it can be observed that for $U_{main} = 98\%$ and $f_{INT} = 2.5kHz, \ldots, 100kHz$ that no more than about 1600 INTs can be serviced to guarantee timeliness of hard tasks) at which timing constraints of hard tasks can be met and vice versa, to increase the values if there is enough CPU time to service the INTs as well as to execute hard tasks.

Using the absolute numbers, it can be concluded that – using various configurations/types of the realization instruments mentioned in the paper – for our adaptive approach it holds i) the maximum number of INTs it is able to manage moves from 34 to 250, ii) its maximum throughput is 44×10^6 INTs per second, iii) the maximum number of supported priority levels (INTs) ranges from 4 to 256 (from 34 to 250) and iv) its maximum operating frequency is 663 MHz.

IV. Conclusion

Our research has resulted into the design of a general-purpose mechanism able to prevent embedded, event-driven RT systems from the predictability part of the IOV problem by adapting the f_{INT} (and consequently, INT service) rate to an actual load of the CPU utilized to execute the RT part of an embedded application.

Although common components such as FPGA [16] and RT kernel [17] running on the MCU [18] were utilized to perform a prototype of our solution, the proposed mechanism is general enough to abstract from those components, each of which can be substituted by a different one.

Further extensions of the proposed principle can be seen e.g. in adding a cooperation with advanced scheduling policies such as sporadic servers, overload-scenario mechanisms and/or device drivers designed to solve the aperiodic event occurence problem at other levels of an operating system. Also, it can be advantageous to profit from the multi-core organization of recent MCUs and utilize it e.g. for the enhancement and/or alternative design of the adaptive principle proposed in this article.

ACKNOWLEDGMENT

This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project No. ED1.1.00/02.0070 (CZ.1.05/1.1.00/02.0070) and the inner university project No. FIT-S-14-2297 (Architecture of parallel and embedded computer systems).

REFERENCES

[1] A. M. K. Cheng, *Real-Time Systems, Scheduling, Analysis, and Verification*, John Wiley & Sons, Hoboken NJ, United States, 2002.

[2] F. Cottet, J. Delacroix, C. Kaiser, Z. Mammeri, *Scheduling in Real-Time Systems*, John Wiley & Sons, Hoboken NJ, United States, 2002.

[3] P. A. Laplante, *Real-Time Systems Design and Analysis*, Wiley-IEEE Press, Hoboken NJ, United States, 2004.

[4] H. Kopetz, *On the Fault Hypothesis for a Safety-Critical Real-Time System*, Automotive Software – Connected Services in Mobile Networks, Lecture Notes in Computer Science 4147 (1) (2006) 31–42, DOI 10.1007/11823063_3.

[5] J. Strnadel, Monitoring-Driven HW/SW Interrupt Overload Prevention for Embedded Real-Time Systems, in: Proceedings of the 15th International IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), IEEE CS, US, 2012, pp. 121 – 126, DOI 10.1109/DDECS.2012.6219037.

[6] J. Strnadel, *Load-Adaptive Monitor-Driven Hardware for Preventing Embedded Real-Time Systems from Overloads Caused by Excessive Interrupt Rates*, in: Architecture of Computing Systems - ARCS 2013, Lecture Notes in Computer Science, Springer, 2013, pp. 98 – 109, DOI 10.1007/978-3-642-36424-2_9.

[7] J. Strnadel, *On Design of Priority-Driven Load-Adaptive Monitoring-Based Hardware for Managing Interrupts in Embedded Event-Triggered Real-Time Systems*, in: Proceedings of the 16th International IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), IEEE CS, 2013, pp. 24 – 29, DOI 10.1109/DDECS.2013.6549783.

[8] L. E. L. del Foyo, P. Mejia-Alvarez, *Custom Interrupt Management for Real-time and Embedded System Kernels*, in: Proceedings of the Embedded Real-Time Systems Implementation Workshop at the 25th IEEE International Real-Time Systems Symposium, IEEE Computer Society, Washington DC, United States, 2004, p. 8, DOI 10.1.1.100.7025.

[9] L. E. L. del Foyo, P. Mejia-Alvarez, D. Niz, *Predictable Interrupt Management for Real Time Kernels over Conventional PC Hardware*, in: Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, IEEE Computer Society, Washington DC, United States, 2006, pp. 14–23, DOI 10.1109/RTAS.2006.34.

[10] L. E. L. del Foyo, P. Mejia-Alvarez, D. Niz, *Integrated Task and Interrupt Management for Real-Time Systems*, ACM Transactions on Embedded Computing Systems 11 (2), 2012, 32:1–32:31, DOI 10.1145/2220336.2220344.

[11] J. Regehr, U. Duongsaa, *Preventing Interrupt Overload*, in: Proceedings of the ACM SIGPLAN/SIGBED Conference On Languages, Compilers, And Tools For Embedded Systems, ACM, New York, United States, 2005, pp. 50–58, DOI 10.1145/1070891.1065918.

[12] J. Regehr, *Safe and Structured use of Interrupts in Real-Time and Embedded Software*, in: Handbook of Real-Time and Embedded Systems, I. Lee, J. Y.-T. Leung, and S. H. Son Eds., 1st Edition, Chapman & Hall/CRC, Boca Raton, FL 33487, United States, 2007, pp. 16–1 – 16–12, DOI doi:10.1.1.79.4651.

[13] R. Pellizzoni, *Predictable And Monitored Execution For Cots-Based Real-Time Embedded Systems*, Ph.D. thesis, University of Illinois at Urbana-Champaign, Bonn, Germany (Dec. 2010).

[14] *Lynx Software Technologies Patented Technology Speeds Handling of Hardware Events*, 2014. Available from http://www.lynx.com/whitepaper/lynx-software-technolog ies-patented-technology-speeds-handling-of-hardware-events/

[15] AUTOSAR, *Specification of Operating System*, Technical report, Automotive Open System Architecture GbR (AUTOSAR), 2014. Available from *http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/system-services/standard/AUTOSAR_SWS_OS.pdf*

[16] *Xilinx Spartan 6 FPGA Family*, 2014. Available from http://www.xilinx.com/products/silicon-devices/fpga/spartan-6/index.htm

[17] *µC/OS-II: The Real-Time Kernel*, 2014. Available from http://micrium.com/rtos/ucosii/overview/

[18] *ARM Cortex-A9*, 2014. Available from http://www.arm.com/products/processors/cortex-a/cortex-a9.php