Concept of Adaptive Embedded HW/SW Architecture for Dynamic Prevention from Interrupt Overloads

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

I. Introduction

Because of high reactivity, availability and other factors, occurrence of the events w.r.t. an embedded system is usually signalized by interrupts. If the interrupts are not processed properly the system may become overloaded due to unpredictable streams of interrupts. As a consequence, the system can fail to operate correctly or, it can collapse suddenly as the rate of interrupts (f_{int}) increases. To avoid this, the system should be designed to *degrade gracefully* rather than continue to execute part of its workload. In safety&time-critical systems, the requirement is yet more strict – the system may never give up (to recover) even if the load hypothesis is violated by the reality [2].

II. Related Works

In the past, several works have dealt with the problem of bounding minimum interarrival time between interrupts to $t_{arrival}$ (or, maximum interrupt rate to $f_{arrival}$). Representatives of the approaches presented in the works are mentioned below [3]. The first group of approaches is based on the HW interrupt controller – typically called a *Hardware Interrupt Limiter* (HIL) – utilized to filter (decrease rate of) interrupt requests incoming to an MCU. Interrupt requests are processed by the HIL before they are directed to the MCU an embedded SW runs on.

On top of few HIL approaches, many SW solutions (called *interrupt schedulers*, IS) to the problem exist, e.g.: a) *polling IS* is driven by a periodic timer that expires each $t_{arrival}$ time units. After the expiration event flags are read to react to, b) *strict IS* is based on the following idea: interrupt prologue is modified to disable interrupts and configure a one-shot timer to overflow after $t_{arrival}$ is over; after it overflows interrupts are re-enabled, c) *bursty IS* is designed to reduce high overheads typical for strict IS. Bursty IS is driven by *burst size* (N) and *burst arrival rate* ($f_{arrival}$) parameters:interrupts are disabled, interrupt counter incremented and timer started to overflow in $t_{arrival}$ ticks after the burst of N interrupt requests, rather than after each interrupt request. Interrupts are re-enabled and the counter is reset after the timer overflows.

Disadvantages of the existing solutions can be summarized as follows: Polling IS is simple to implement, but offers low reactivity (big interrupt response delay) at high overhead. Strict&bursty ISes offer higher reactivity, but still at overhead close to polling IS. Moreover, they are questionable from the time-criticality point of view because they are allowed to disable interrupts for the predefined time – along with other interrupts, timer interrupts utilized to increment the system time are disabled too. Disadvantage of the HIL approach can be seen in the fact that an extra HW is needed to process interrupts. However, HIL is able both to enhance reactivity and to reduce overhead comparing to IS approaches.

III. Concept of the Proposed Architecture

From the adaptability point of view, existing approaches can be seen as static because their interrupt arrival limits are fixed and independent on the actual MCU load – the limiters have no information about the load, so they are not able to adapt to the factor.

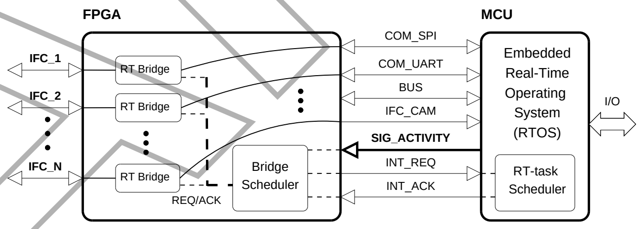

Figure 1. Camea AX32 [1]

Proposal of our HW/SW architecture can be seen in Fig. 1 – it is composed of FPGA (Xilinx Spartan-6) and MCU (ARM Cortex-A9) parts utilized to perform HW and SW functionalities of the system. SW is supposed to be driven by an embedded real-time (RT) operating system (RTOS) able to guarantee timeliness of all reactions.

SW may not overload due to excessive rates of interrupt stimuli, so the FPGA is to be designed to pre-process all interrupts before they are directed to the MCU. To each of communication interfaces (`IFC_i`) able to generate an interrupt request a separate `RT Bridge` responsible for processing stimuli related to the source is assigned. For this purpose, we have combined the `RT Bridge` concept [4] with HIL and MCU-load signaling techniques to guarantee that during high-load of the MCU all interrupts are buffered by the FPGA until the MCU is underloaded. Then, they are directed to the MCU. From the design point of view, the following key elements must be involved in the HW/SW architecture:

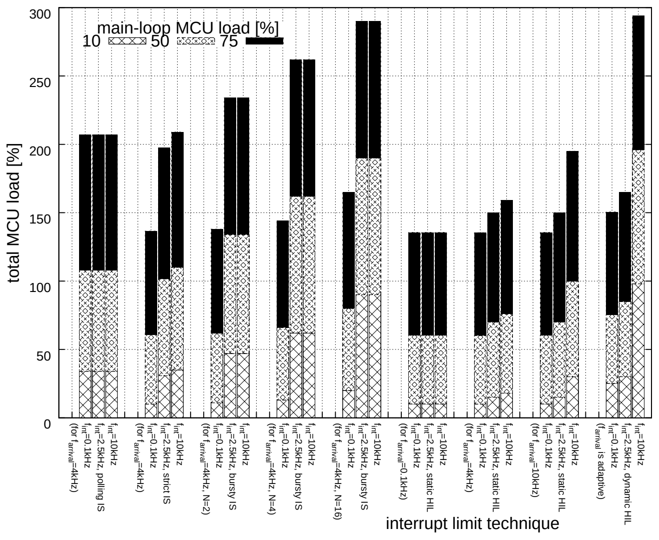

Figure 2. Total MCU load for various techniques and main-loop loads

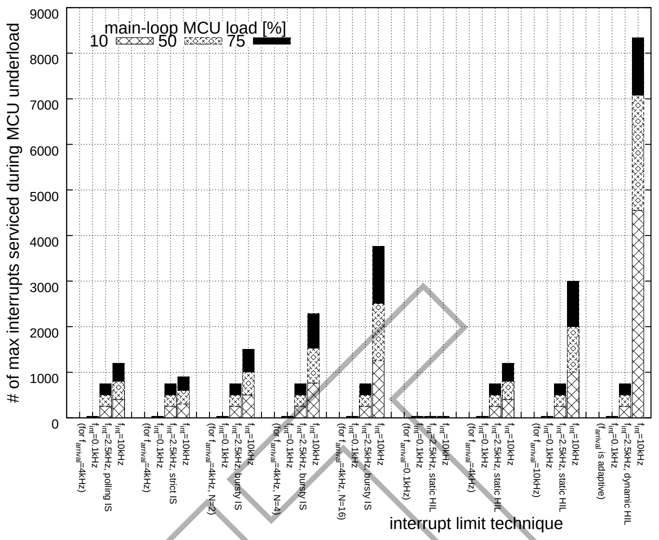

Figure 3. Max. # of interrupts serviced by the MCU during MCU underload for various techniques and main-loop MCU loads

- RT Bridge for each `IFC_i`. Buffers inside the bridges must be of ”sufficiently large” capacity to store all the stalled communication,
- Mechanism utilized to signal MCU’s activity (`SIG_ACTIVITY`) to the FPGA. The signalization must be done at lower than interrupt priority level,
- `Bridge Scheduler` responsible for controlling RT Bridges (i.e., buffering/redirecting stimuli) on basis of the MCU load evaluated from `SIG_ACTIVITY`.

IV. Pre-Implementation Analysis

A model of the concept presented in the section III was analyzed and compared to the approaches mentioned in the section II. The results are summarized in Fig. 2 and Fig. 3.

The vertical axis of the figures represent interrupt overload prevention techniques and their particular variants determined by $f_{arrival}$ and by *burst size* values (where applicable); for each of the techniques, data are plotted for 3 various f_{int} values: $0.1kHz$, $2.5kHz$ and $10kHz$. The last 3 columns on the right hand side belong to the concept presented in the paper – labeled ”*dynamic HIL*”. The horizontal axis of Fig. 2 (Fig. 3) represent the total MCU load in % (# of max. interrupts serviced by the MCU during MCU underload within 0.1s window) – aggregated values are plotted for 3 various MCU loads of the main loop: 10 %, 50 % and 75 %.

In the figures, it can be seen that (especially for the highest f_{int} value) our approach is able to service significantly higher number of interrupts during MCU underload than other approaches at comparable values of the MCU load – this is because our concept is able to utilize main-loop idle intervals to service excessive interrupts.

V. Future Plans

The results outlined in the section IV seem very promissing, so the next logical steps can be seen a) in the implementation of the concept to the AX32 platform running an embedded RTOS, b) in performing detail experiments on AX32 with the goal to confirm or disprove the results and c) to evaluate the solution in several embedded real-life applications. On top of the mentioned, the impact of RT-task schedulers such as `Dover` or `DASA` implemented on the MCU side to the overall robustness could be studied.

Acknowledgment

This work has been partially supported by the RECOMP MSMT project (National Support for Project Reduced Certification Costs Using Trusted Multi-core Platforms), GACR project No. 102/09/1668 (SoC circuits reliability and availability improvement), Research Plan No. MSM 0021630528 (Security-Oriented Research in Information Technology) and grant BUT FIT-S-11-1.

References

[1] CAMEA, spol. s r.o., *AX32 platform*. Accessible from *http://www.camea.cz/en*.

[2] 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, Vol. 4147/2006, Springer Berlin, 2006, pp. 31 – 42.

[3] J. Regehr and U. Duongsaa, *Preventing interrupt overload*. In: Proceedings of the ACM SIGPLAN/SIGBED Conference On Languages, Compilers, And Tools For Embedded Systems. ACM, 2005, pp. 50 – 58.

[4] R. Pellizzoni, *Predictable And Monitored Execution For Cots-Based Real-Time Embedded Systems*. Ph.D. Thesis, University of Illinois at Urbana-Champaign, 2010, p. 154.