Deadline-Aware Interrupt Coalescing in Controller Area Network (CAN)

Christian Herber, Andre Richter, Thomas Wild, Andreas Herkersdorf

Technische Universität München - Institute for Integrated Systems
Munich, Germany
{christian.herber, andre.richter, thomas.wild, herkersdorf}@tum.de

***Abstract*—The introduction of virtualized multi-core processors in automotive embedded systems opens up opportunities like safe consolidation of previously distributed electronic control units (ECUs) on a shared platform. On the other hand, challenges arise in areas like I/O processing due to overheads experienced in virtualized environments. Designs of I/O controllers have to be adjusted to allow efficient, scalable, and real-time capable communication under these circumstances.**

Interrupts are an essential part in real-time communication. However, they introduce significant computational overheads, because they force multiple context switches within the CPU. Interrupt coalescing reduces the burden of interrupt processing by merging multiple interrupts within the I/O hardware. However, existing coalescing approaches are not feasible for real-time networks like CAN due to the latencies they introduce.

In this paper, we introduce a deadline-aware approach of interrupt coalescing for CAN controllers. It minimizes the amount of interrupts forwarded while guaranteeing the systems real-time capability. We provide three approximations of the method, which can be implemented in hardware. We evaluate the reduction of interrupts that can be achieved with each approach and determine the hardware cost with a prototypical FPGA implementation.

***Index Terms*—Controller area network, CAN, interrupt coalescing, automotive electronics, embedded systems.**

I. Introduction

Automotive embedded systems are composed of distributed electronic control units (ECUs) that are interconnected by a variety of fieldbuses like Controller Area Network (CAN), FlexRay and MOST. The approach of introducing new ECUs for every new function has led to a high complexity with up to 100 ECUs and more than 2.7 km in wire length.

OEMs are planning to reduce the number ECUs by integrating previously distributed functions on virtualized multi-core platforms [1]–[3]. These act as domain controllers in one of the functional domains like powertrain, chassis, infotainment, driver assist etc. Simple, distributed nodes remain within each domain to provide access to sensors and actuators. Centralized control units face significantly increased communication requirements compared to traditional ECUs.

Virtualization is an important technology to leverage isolated computing resources on shared multi-core platforms [4]–[7], but also introduces significant overheads. Traditionally, I/O processing has been the major contributor of computational overheads in virtualized systems [8]. For example, interrupts in a virtualized x86 system introduce an additional computational overhead in of around 10,000 cycles per interrupt [9].

While interrupts for CAN I/O events are considered essential for low receive latencies, they come along with several downsides. First, interrupt requests (IRQs) force the processor to suspend running tasks and switch to a privileged mode to run an interrupt service routine (ISR). The time needed for context switching is lost for the execution of tasks. This time is greater for virtualized systems. Other problems are cache pollution due to frequent execution of ISRs and unnecessary wake-ups from low power states.

Interrupt coalescing has originally been introduced for UART [10] as a mechanism to reduce the rate of IRQs. Later on, it was adopted for Ethernet based systems [11]. IRQ reduction is achieved by issuing interrupts only after a predefined number of packets are buffered or after a timeout. This approach reduces the computational overhead for IRQ processing, but increases average and maximum packet latencies. While such added latencies can be tolerated in most Ethernet systems, they can cause deadline violations in systems with real-time requirements.

In this paper, we introduce deadline-aware interrupt coalescing for CAN. Existing coalescing mechanisms are applied and extended to incorporate knowledge about the last possible time at which an interrupt has to be forwarded (deadline). This allows us to reduce the number of interrupts without drawbacks in real-time performance. While we implemented the concept for CAN in this paper, deadline aware interrupt coalescing could be applied to many other interconnects and sources of interrupts.

We provide a detailed presentation of three possible implementations for CAN, which differ in complexity and effectiveness (Section III). We evaluate their ability to reduce the amount of necessary interrupts. Additionally, we show that the coalescing mechanism leads to an advantageous shaping of IRQ inter-arrival times (Section IV). Finally, we present a prototypical implementation and the associated HW costs for all proposed variations of deadline-aware interrupt coalescing (Section V).

II. Related Work

Interrupts are an important mechanism to provide low latencies in I/O reliant applications, but also require significant

computational resources to be processed. The time spent serving interrupt requests (IRQs) is made up of context-switches, data transfers and interrupt controller interaction and is essentially lost for actual applications. For high interrupt rates, livelocks can occur, where interrupts arrive faster than the CPU is capable of processing [11].

A number of similar approaches aimed at reducing the interrupt rate have been introduced. Mogul and Ramakrishnan [11] proposed disabling interrupts temporarily if the interrupt processing cannot keep up with arrival rates. Another approach modifies the network interface controller (NIC) to enable coalescing of interrupts within the hardware [12]. Today, such interrupt throttling mechanisms are included in many NICs and only issue IRQs after a certain number of received frames or a timeout.

By reducing the number of interrupts, interrupt coalescing increases packet latencies. Finding a good trade-off between I/O processing overhead and packet latency is an important design goal. While Ethernet communication is usually not constraint by real-time requirements, added latencies can degrade the TCP throughput [13].

Interrupt processing overheads are greater in virtualized systems, where privileged operations are executed by the hypervisor and applications are run in virtual machines (VMs). Interrupts are first processed by the hypervisor or a dedicated driver domain and forwarded to the receiving virtual machines (VMs) afterwards, which forces additional context switches.

Coalescing of virtual interrupts issued towards VMs reduces the number of switches between VMs and the hypervisor or driver domain. It has been implemented in e.g. XEN [9] and VMWare ESX [14]. Using a Xeon 5560 platform and Gigabit Ethernet, Dong et al. [9] demonstrated that virtual interrupt coalescing reduces the CPU load by 71% for TCP and 24% for UDP for 9 VMs.

The best I/O performance is achieved using NICs with dedicated virtualization support [15]. VMs can bypass the hypervisor and directly access these devices for data and control path operations. However, hypervisor involvement is still necessary for interrupt processing, therefore forcing VM exits for each interrupt. A software extension to enable exitless interrupts has been shown in [16], but comes along with the downside that all interrupts have to be directed towards VMs. Another solution is proposed by Guan et al. [17], who propose an event based polling scheme to reduce the computational overhead.

We propose a concept for deadline-aware interrupt coalescing, which reduces the computational overhead for processing IRQs in CAN based embedded systems. In contrast to existing approaches, the throttling of IRQs is constrained by real-time requirements. Therefore, the hardware must have reliable knowledge of how long it can delay IRQ forwarding (deadline-awareness).

III. Deadline-Aware Interrupt Coalescing

CAN is the most prevalent bus in automotive systems. It is a broadcast medium, i.e. every node receives every transmitted message. Based on a unique message ID, the content can be identified. The ID also serves as strict priority in the bus access arbitration scheme.

Interrupts are important to achieve low latency communication in CAN [18]. Each CAN node is programmed to accept a subset of all received frames depending on its function. The I/O controller notifies the arrival of frames to the host system by issuing an interrupt request (IRQ). In a virtualized system, this forces a VM exit, and the hypervisor will forward the interrupt to the VM. The received data is then copied to the main memory by an interrupt service routine (ISR) executed within the VM. After completion of the ISR, the hypervisor notifies the interrupt controller and the interrupted task is resumed. Fig. 1 shows an exemplary illustration of the interrupt handling behavior of the hypervisor, which can involve more than two VM exits depending on its implementation and the platform. Time spend with additional hypervisor routines and context-switching equals the overhead contributed from each IRQ.

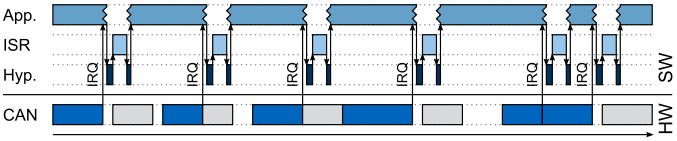


Fig. 1. CAN interrupt handling in a virtualized environment

CAN is used in real-time systems, i.e. message transmissions have to meet predefined deadlines. The worst-case response time (WCRT) of each message m can be calculated analytically [19] and should be smaller than its deadline D_m to guarantee a system's schedulability. Messages are either transmitted in cyclic manner or sporadically with a minimum inter-arrival time T_m. Fig. 2 shows an exemplary configuration of these values for a single message.

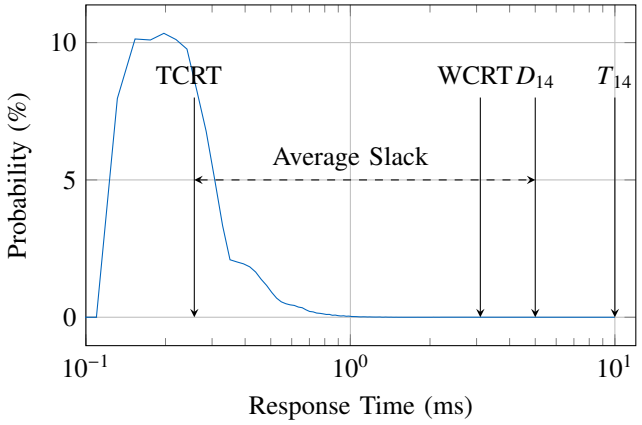


Fig. 2. Response time distribution of CAN message $m = 14$ with cycle time $T_{14} = 10$ ms at 90 % bus load on 500 kbit/s bus

Average latencies in a physical system differ greatly from their computed worst-case latencies [20]. The distribution of response times shown in Fig. 2 shows that typical case

response time (TCRT) and the corresponding WCRT and deadline respectively differ in more than one order of magnitude. The gap between a message's TCRT and its deadline constitutes an average slack, that we will leverage in our interrupt coalescing mechanism. If messages arrive with high slack, IRQs do not have to be delivered immediately after message reception, but can be delayed and coalesced with other IRQs.

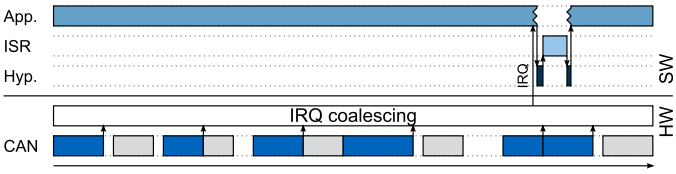


Fig. 3. Deadline-aware interrupt coalescing for CAN

We propose a concept, in which an IRQ is forwarded just in time before any deadline violation can occur. If multiple messages are currently buffered in the I/O controller, their reception can be signaled using one instead of multiple IRQs (see Fig. 3). To realize this coalescing, timing information has to be available in the I/O controller.

We assume that $t^i_{D,m}$ describes the instant of time, at which an interrupt has to be forwarded to guarantee the timely delivery of message m^i. The deadline D_m of a message m^i describes the time available after the release of a message to conclude the transmission by issuing an IRQ. The timeout for an IRQ delivery can be computed as

$$t^i_{D,m} = t^i_{release,m} + D_m, \tag{1}$$

where $t^i_{release,m}$ is the release time of message instance m^i. If a message gets released while the bus is idle, it gets transmitted immediately and every receiver can derive the release time accurately. Otherwise, the release of a message is masked by ongoing transmissions. Therefore, approximations have to be made, which allow deadline-aware interrupt coalescing while providing real-time guarantees.

We propose three approaches that approximate the ideal concept of deadline-aware interrupt coalescing. They have in common that interrupts are triggered based on timeout, which indicates the (approximate) next deadline of a buffered message. The timer is updated after a message reception if the expected slack is smaller than the current timeout value. The slack of a received message m^i can be computed as the difference between the nominal deadline D_m and the time elapsed between the initial release $t^i_{release,m}$ and the successful reception $t^i_{Rx,m}$

$$t^i_{slack,m} = D_m - (t^i_{Rx,m} - t^i_{release,m}). \tag{2}$$

The approaches we introduce differ in their approximation of the slack. We present them in the order of increasing complexity.

a) Fixed delay: Interrupts are forwarded after a fixed delay. The delay is equal to the minimum slack of received messages that can occur in a worst-case scenario. This approach is similar to existing approaches used in Ethernet NICs.

b) Message based delay: The worst-case slack for each message is known. Upon message reception, the timer may be updated based on the messages slack. This poses an improvement, because the worst-case slack varies for all messages.

c) Dynamic deadline estimation: After reception, the deadline of the message is estimated and the timer is updated if necessary. A detailed description of the estimation algorithm is presented in the remainder of this Section.

The release time of a message can be estimated in a pessimistic way so that real-time constraints are not at risk. We propose a simple estimation, which makes use of the last time the bus was idle. After reception of a message, the receiver can safely assume that the message was not released during the last idle time. We use the end of this period as a pessimistic estimate of the release time. An example of the deadline estimation is illustrated in Fig. 4. $\langle\widehat{\cdot}\rangle$ denotes that a value is estimated. The estimation error is given as

$$e^i_{est,m} = t^i_{slack,m} - \widehat{t}^{\,i}_{slack,m} \geq 0. \tag{3}$$

It shows that the estimated slack is strictly smaller than or equal the actual slack and therefore fulfills the necessary condition of pessimism.

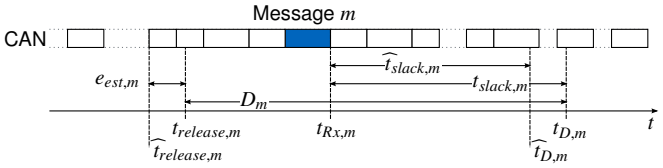


Fig. 4. Deadline estimation of a message based on the last idle time.

The effectiveness of the proposed approaches depends on the accuracy of their approximation. It will be evaluated in the following Section.

IV. Simulation & Results

We verified the concept by simulating the interrupt behavior of a CAN node with and without interrupt coalescing. The simulation uses realistic automotive communication patterns. A detailed description of the simulation scenario and results are presented in the subsequent Sections.

A. Simulation Scenario

We simulated deadline-aware interrupt coalescing using a realistic setting for future automotive IT architectures. Our point of interest is the central ECU within a domain, the so called domain controller. In this scenario, 50% of the messages are received by this node on average.

CAN message sets are randomly generated at predefined bus loads, with the available bandwidth being 500 kbit/s. Message cycle times T_m are chosen from four harmonic sets with 10 ms, 20 ms, 50 ms, and 100 ms respectively. Message deadlines

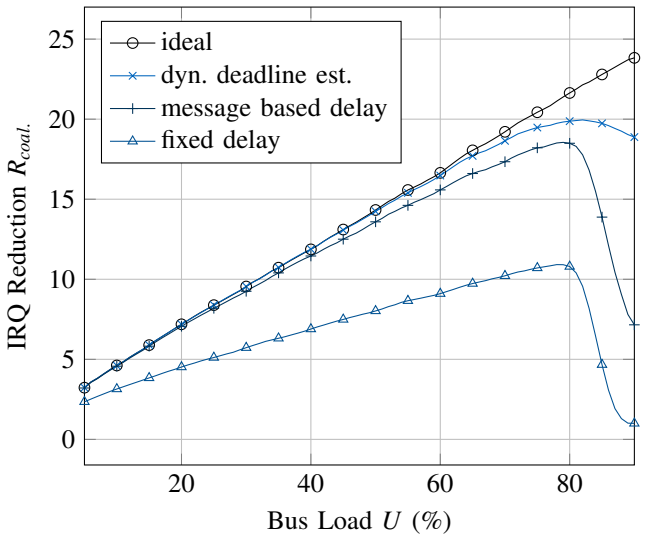


Fig. 5. Reduction of interrupts using different approximations of deadline-aware interrupt coalescing

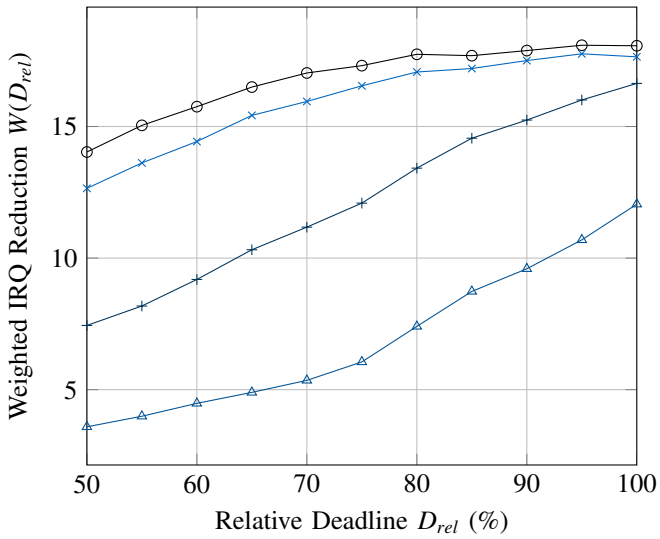


Fig. 6. Weighted reduction of interrupts depending on latency requirements. A small relative deadline implicates low latency requirements.

D_m are randomly chosen from the range between 50% and 100% of a messages respective cycle time. The payload varies between 1 and 8 bytes. To capture the effects of oscillator inaccuracies, the release of messages is subject to a drift of ±1.5% of their cycle time. The random values used to generate the scenario are uniformly distributed.

Using an event based simulation of CAN, we measured the number of interrupts forwarded in state of the art systems and with variations of deadline-aware interrupt coalescing. To be able to draw general conclusions independent of single message set configurations, we repeat the simulation 10,000 times for every bus load. This gives us a realistic estimate on what can be achieved in the average case.

B. IRQ Reduction

The reduction of IRQs $R_{coal.}$, which can be achieved using the proposed coalescing schemes, will be used as central measure to evaluate our different approaches. We define it as the average amount of IRQs that can be reduced to a single IRQ. It compares our approach with state of the art systems, where interrupts are used for each message. Assuming IRQ_{coal} and IRQ_{SotA} to be measures for the number of IRQs forwarded in systems with and without interrupt coalescing respectively, it can be computed as

$$R_{coal.} = \frac{IRQ_{SotA}}{IRQ_{coal.}}. \tag{4}$$

The simulation results presented in Fig. 5 show the IRQ reduction for ideal interrupt coalescing and the three approximations presented in Section III. In ideal deadline-aware interrupt coalescing, all timing information is available and perfect coalescing decisions can be made. It is not reachable in an actual implementations. CAN controllers without interrupt coalescing would be represented by the horizontal line $R_{coal.} = 1$.

In the ideal case, the IRQ reduction scales linearly with the bus load. It can be credited to the fact that messages arrive in shorter intervals and are therefore easier to coalesce. Even for high bus loads, where the worst-case slack of messages is small, the mechanism shows no deviation from this linear trend. This shows that on average, messages experience low delays on the CAN bus and arrive with high slack even in high load scenarios.

All approximations show visible degradation compared to ideal coalescing for high bus loads. Interrupt coalescing with fixed delay can only work if the slack of all messages is greater than the minimum transmission time of a CAN frame. Otherwise, interrupts will be forwarded before another frame can arrive. Because of the small worst-case slack in highly loaded CAN systems, the performance drops significantly at loads around 80%. The mechanism stops working at loads around 90%.

The use of dedicated delays for interrupts generated by different messages (message based delay) nearly doubles the IRQ reduction for most loads. However, the performance drops at a similar point, when the static worst-case slack approaches the minimum transmission time of a CAN message. Because some messages will still have significant slack even for high loads, the IRQ reduction is still working to some degree.

Dynamic deadline estimation proves to give the best approximation of ideal deadline-aware interrupt coalescing at all bus loads. Similar to the static implementations of the method, the IRQ reduction deviates from the ideal case especially for high bus loads. Here, the deadline estimation accuracy drops, because the bus is busy for longer periods and therefore masking the release of messages. However, the degradation is less steep and a high IRQ reduction rate can be sustained even for bus loads around 90%.

We further evaluate, how the distribution of deadlines influences the reduction of IRQs. In this analysis, we keep the

the relative deadline of all messages in a specific message set constant. It is defined as the ratio between the nominal deadline D_m and the cycle time T_m of a message m

$$D_{rel} = D_m / T_m. \tag{5}$$

We introduce the measure of weighted IRQ reduction, which can be computed as

$$W(D_{rel}) = \frac{\sum_{\forall U} U \cdot R_{coal.}(U, D_{rel})}{\sum_{\forall U} U}. \tag{6}$$

It is a weighted average of the IRQ reduction introduced in (4) as a function of the relative deadline D_{rel}. The weighting emphasizes the importance of high bus loads U. It allows a dedicated inspection of a single parameter without neglecting the influence of the bus load. Such weighted measures are widely used in the context of schedulability analyses and were introduced in [21].

Fig. 6 shows that dynamic deadline estimation outperforms the static implementations independent of the relative deadline. Especially in scenarios, where low latencies (small D_{rel}) are required, dynamically estimating message deadlines nearly doubles the effectiveness of the coalescing mechanism. When deadlines are close to the respective cycle times, the static approach using message based delays reaches a performance similar to dynamic deadline estimation.

C. Deadline Estimation

The discrepancy between ideal deadline-aware interrupt coalescing and the approach using dynamic deadline estimation can be accredited to the necessary pessimism in the estimation mechanism. We recorded the average estimation error throughout the simulation, which is presented alongside the percentage of accurate predictions (deadline hits) in Fig. 7.

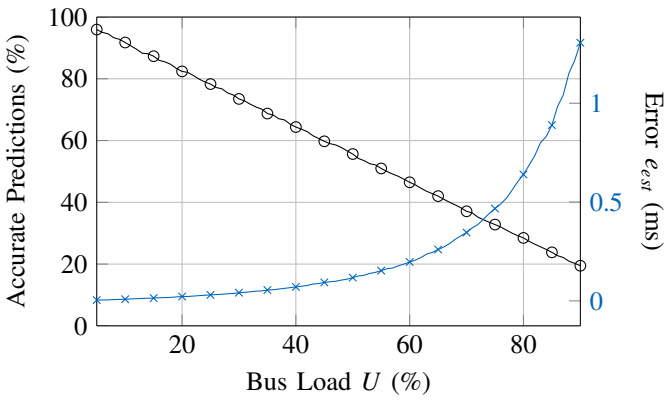


Fig. 7. Percentage of accurate predictions and average error due to overestimation of deadlines.

Accurately predicting a message deadline is only possible if and only if it is released onto an idle bus. The probability of the bus being idle is inversely proportional to the bus load. While at 90% bus load only 20% correct predictions can be made, the measurement of the IRQ reduction showed that meaningful results can be obtained nevertheless.

The estimation error increases significantly with the bus load and exceeds 1 ms for loads greater 85%. Deadlines in automotive CAN networks range between 5 ms and 100 ms and exceed the estimation error by up to two orders of magnitude. Therefore, the estimation mechanism proves to be feasible even for high bus loads.

D. IRQ Inter-Arrival Time

The inter-arrival time of IRQs is an important figure to evaluate the impact of interrupts on a systems performance. During this time, a CPU can process tasks without being interrupted. If no tasks are pending, the time can be used to enter low power states. Additionally, the cache will not be polluted by the IRQ processing in the hypervisor during this interval.

We evaluated the inter-arrival time of CAN receive IRQs with and without interrupt coalescing at different bus loads. The coalescing mechanism uses dynamic deadline estimation. Probability density functions (PDF) and cumulative distribution functions (CDF) of the IRQ inter-arrival time $t_{irq2irq}$ are shown in Fig. 8.

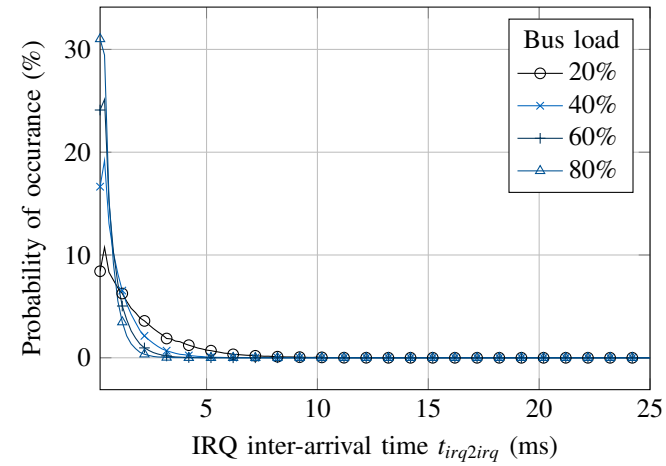
The PDF of IRQ inter-arrival times without coalescing presented in Fig. 8a shows that low inter-arrival times are most probable. After a short peak between 400 and 600 µs, it is monotonically decreasing. The inter-arrival time decreases with increasing bus load.

Fig. 8b shows the IRQ shaping property of the coalescing mechanism. The distribution of inter-arrival times peaks around 10 ms (smallest cycle time in this scenario) at all bus loads. An additional peak around 20 ms exists for low bus loads at nodes, that only receive messages with 20 ms cycle time or higher. Additionally, IRQs are received with a spacing between 5 ms and 10 ms, which corresponds to the distribution of deadlines for high priority messages.

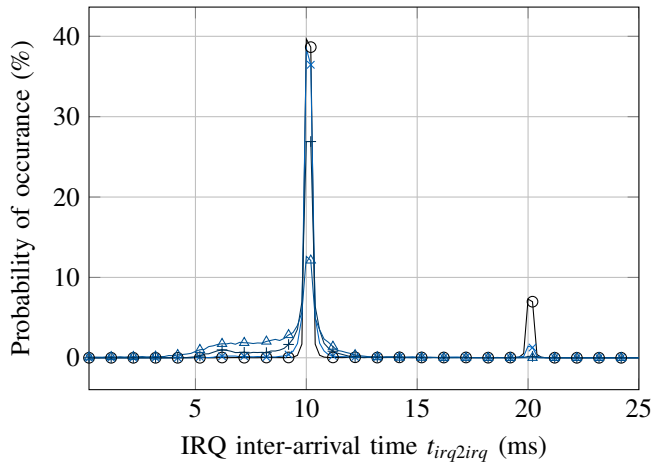
The high density of IRQ inter-arrival times around 10 ms occurs due to a resonance in the coalescing. Typically, CAN nodes are not trying to transmit at the same time, but the transmissions spread out equally in time. Messages can be transferred directly or after a short wait and therefore arrive with low slack. The first message with low deadline transmitted after an IRQ has been forwarded will determine the time, when the next IRQ will be released. After the IRQ has been forwarded, the same message is likely to be transmitted soon after and therefore a deterministic IRQ pattern emerges.

While such deterministic patterns in the IRQ inter-arrival times are likely to occur, it cannot be guaranteed to happen over long time intervals. If message transmissions are increasingly aligned, the queuing delays of all messages increase as well. Therefore, they are received with small slack and IRQs are forwarded in short intervals. Because such scenarios are unlikely, they do not have significant influence on the distribution of Fig. 8.

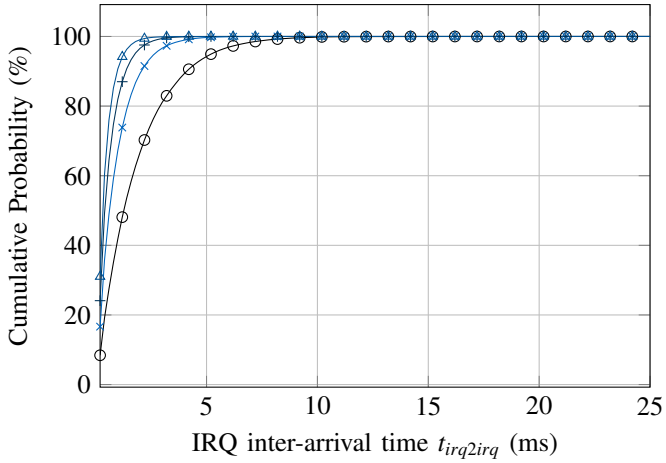
The cumulative distribution of IRQ inter-arrival times illustrated in Fig. 8c shows that an IRQ will be followed by another IRQ with more than 50% probability in less than 1 ms. For high bus loads, this point is around 300 µs.



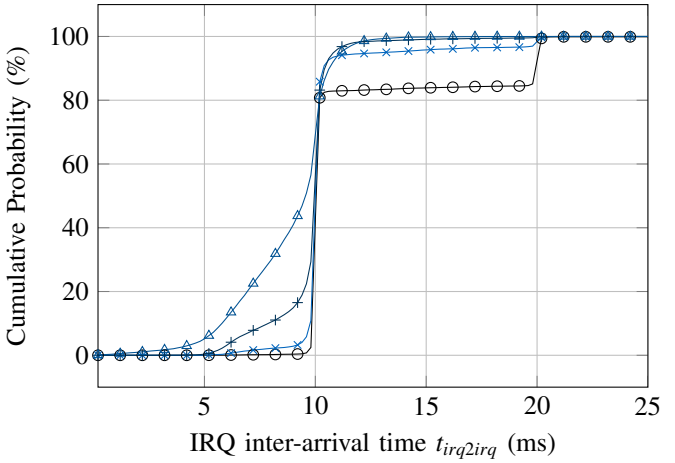
(a) No interrupt coalescing (PDF)



(b) Deadline aware interrupt coalescing with dyn. deadline estimation (PDF)



(c) No interrupt coalescing (CDF)



(d) Deadline aware interrupt coalescing with dyn. deadline estimation (CDF)

Fig. 8. IRQ inter-arrival time evaluated for different bus loads: The plots present probability density functions (PDF) and cumulative distribution functions (CDF) with and without interrupt coalescing.

Using deadline-aware interrupt coalescing (see Fig. 8d), the 50% mark can be shifted to 9.5 ms even for high bus loads. The probability for IRQ inter-arrival times of at least 5 ms for 80% bus load is approximately 95%. This predictable behavior and the long spacing between consecutive IRQs can be used for uninterrupted task processing (no VM exits, cache pollution etc.) or to utilized deep sleep states with low probability of wake ups caused by interrupts triggered from CAN events.

V. Implementation

We implemented the three proposed coalescing mechanisms as extension to a virtualized CAN controller [22] and evaluated them with respect to their resource requirements. For our implementation, we assume a virtualized system, in which multiple VMs access the CAN bus through a shared, virtualized CAN controller (see Fig. 9). The virtualized controller consists of a protocol layer, which essentially implements the CAN protocol, and a virtualization layer, which offers abstract data path operations (prioritized buffering, receive filters etc.) in the form of multiple virtual controllers. Each virtual CAN controller is directly connected to a VM. As part of the virtualization layer, the interrupt coalescing module should be able to handle interrupts from multiple virtual CAN controllers.

We implemented all three versions of deadline-aware interrupt coalescing in a conditional Verilog module, i.e. parts of the module are not synthesized based on a design-time parameter. Using this parameter, we will compare the hardware cost of all implementations. The architecture of the module is shown in right half of Fig. 9, where conditional modules, ports and signals are drawn colored and dashed/dotted, respectively.

After successful message reception from the CAN bus, an IRQ is signaled to the *timeout_update* module by a rising flag in *irq_in*. Which virtual CAN controllers are associated with the IRQ is determined by *irq_vcan_in*. The component maintains a timeout register for each virtual CAN controller. The timeout value is decremented after each bit time (2 µs at 500 kbit/s) on the CAN bus indicated by *can_sample_point* until it reaches zero. It is set if the slack of a received message

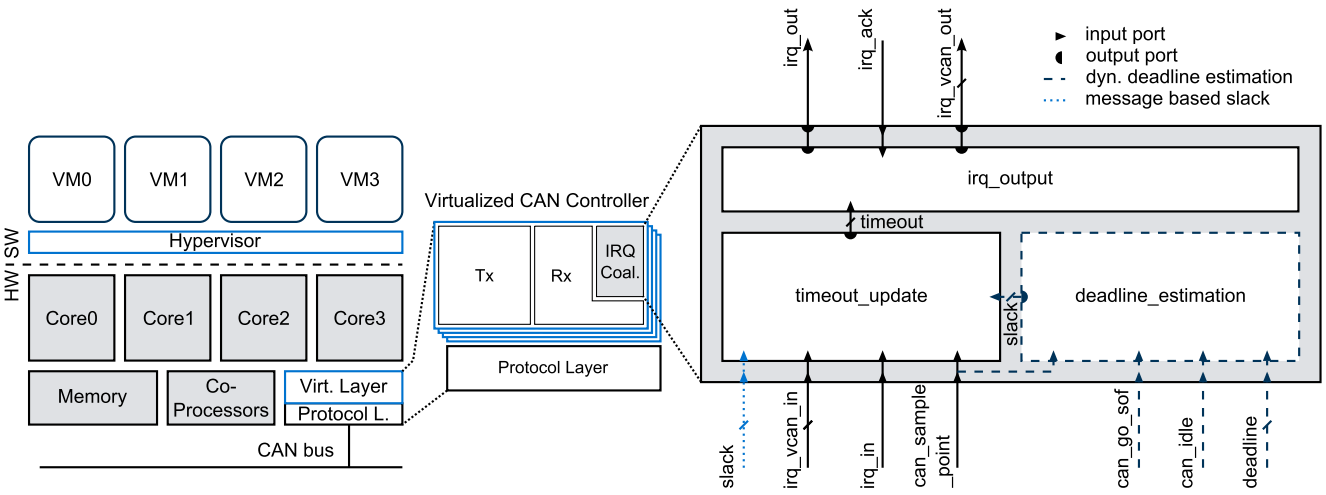


Fig. 9. Architecture of a virtualized system (left) with virtualized CAN controller (mid). As part of the virtualized CAN controller, the interrupt coalescing module (left) is illustrated as conditional implementation, covering all three design alternatives.

is smaller than the current value or no IRQ is currently pending. The implementations differ in how they obtain the slack value. For fixed delay, the slack is a constant value. In message based delay, a design time parameter is input through the *slack* signal. It can be different for every message. In dynamic deadline estimation, an additional module computes the slack during run-time.

When a timeout reaches zero, an IRQ has to be forwarded. The *irq_output* module monitors the timer values and handles the handshaking in IRQ signaling (*irq_out* and *irq_ack*). The virtual CAN controller and therefore also the VM to which the IRQ is directed is indicated by *irq_vcan_out*.

In the case of dynamic deadline estimation, the corresponding module calculates the slack by estimating the time at which the IRQ has to be forwarded to avoid deadline violations. It maintains a time stamp representing the last time the CAN bus was idle. It is taken at the start of a CAN frame (rising edge in *can_go_sof*) if *can_idle* was high before. The slack is calculated using (2) and the time stamp as an estimate for the release time.

Static information about worst-case slacks and design time deadlines of received messages required by the module are stored alongside the reception filters in the Rx path of the virtualization layer. Because we integrated them with an existing RAM module, no additional hardware was required.

TABLE I
HARDWARE RESOURCE REQUIREMENTS

module	Registers	LUTs as Logic
Fixed delay	74	160
Message based delay	74	244
Dyn. deadline estimation	108	267
Protocol Layer	844	1239
Virtualization Layer	534	1735

We synthesized the module for the VC709 (Virtex-7) FPGA board using Vivado 2013.3. The utilization of FPGA resources for the coalescing modules supporting up to four virtual CAN controllers is shown in Table I. As expected, the implementation costs are increasing with the complexity and granularity of the respective method. When transitioning from fixed to message based delay, additional logic is required, because received frames can have smaller slack than the current timeout value.

Dynamic deadline estimation requires an additional module to calculate the remaining slack of each received message. Compared to the implementation using a static, message based delay, around 45% additional registers 10% additional LUTs are necessary. The register overhead mainly stems from the additional timers that are used in the deadline estimation module.

For comparison, we additionally synthesized the virtualized CAN controller. The protocol layer is based on an OpenCores CAN controller[1]. The interrupt coalescing extensions require between 11.8% and 17.3% additional registers and between 17.6% and 29.3% additional LUTs compared to the protocol layer. When also considering the virtualization extensions, the relative overhead decreases to 5.4-7.8% in registers and 3.4-5.7% in LUTs.

VI. CONCLUSION

In this paper, we presented a concept for deadline-aware interrupt coalescing applied to controller area network (CAN). The proposed method tackles the problem of high interrupt processing overheads in real-time computer systems. Multiple events are signaled by a single interrupt without violating real-time constraints.

The general concept of deadline aware interrupt coalescing can be effectively applied to any kind of interrupt sources, if the following criteria are met: 1. The average slack associated

[1]http://opencores.org/project,can

with the forwarding of interrupts is large compared to the inter-arrival of interrupts triggers. 2. (Partial) information about deadlines of interrupt forwarding is available.

These conditions are fulfilled for CAN. Especially for future domain controlled architectures, where a single ECU consolidates multiple previously distributed functions, the high inter-arrival rate of relevant messages allows an efficient coalescing of interrupts.

We proposed three versions of deadline-aware interrupt coalescing for CAN, which differ in how they derive deadline information. Two approaches rely on static worst-case timing information. The third approach uses pessimistic online estimations of message deadlines. Based on these deadline estimations, reliable forwarding decisions can be taken.

We evaluated all three approaches with respect to their ability to reduce the number of forwarded interrupts. The best results can be obtained using dynamic deadline estimations. At 80% bus load, only 1 in 20 interrupts has to be forwarded compared to state-of-the-art controllers. The advantage of dynamic deadline estimation over static approaches is greater in the presence of low latency requirements.

The interrupt coalescing enforces a highly deterministic shaping in interrupt inter-arrival times. We were able to show that interrupts arrive in intervals greater than 9.5 ms with 50% probability at 80% bus load (300 µs without coalescing).

Finally, we implemented the proposed design alternatives as submodule of a virtualized CAN controller for Virtex-7 FPGA. As expected, dynamic deadline estimation requires the most hardware resources. However, compared to the overall design of a virtualized CAN controller, it only adds an overhead of 7.8% in registers and 5.7% in LUTs.

Acknowledgments

This work was funded within the project ARAMiS by the German Federal Ministry for Education and Research with the funding IDs 01|S11035. The responsibility for the content remains with the authors.

References

[1] G. Gut, C. Allmann, M. Schurius, and K. Schmidt, "Reduction of electronic control units in electric vehicles using multicore technology," in *Proceedings of the 2012 International Conference on Multicore Software Engineering, Performance, and Tools*. Springer-Verlag, 2012, pp. 90–93.

[2] D. Reinhardt and M. Kucera, "Domain controlled architecture: A new approach for large scale software integrated automotive systems," in *Pervasive and Embedded Computing and Communication Systems*, 2013, pp. 221–226.

[3] M. Di Natale and A. L. Sangiovanni-Vincentelli, "Moving from federated to integrated architectures in automotive: The role of standards, methods and tools," *Proceedings of the IEEE*, vol. 98, no. 4, pp. 603–620, 2010.

[4] D. Reinhardt, D. Kaule, and M. Kucera, "Achieving a scalable e/e-architecture using autosar and virtualization," *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, vol. 6, no. 2, pp. 489–497, 2013.

[5] K. Sandstrom, A. Vulgarakis, M. Lindgren, and T. Nolte, "Virtualization technologies in embedded real-time systems," in *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*. IEEE, 2013, pp. 1–8.

[6] M. Strobl, M. Kucera, A. Foeldi, T. Waas, N. Balbierer, and C. Hilbert, "Towards automotive virtualization," in *Applied Electronics (AE), 2013 International Conference on*. IEEE, 2013, pp. 1–6.

[7] R. Schneider, A. Kohn, K. Schmidt, S. Schoenberg, U. Dannebaum, J. Harnisch, and Q. Zhou, "Efficient virtualization for functional integration on modern microcontrollers in safety-relevant domains," in *SAE World Congress 2014*. SAE International, 2014.

[8] A. Menon, J. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel, "Diagnosing performance overheads in the xen virtual machine environment," in *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*. ACM, 2005, pp. 13–23.

[9] Y. Dong, D. Xu, Y. Zhang, and G. Liao, "Optimizing network i/o virtualization with efficient interrupt coalescing and virtual receive side scaling," in *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*. IEEE, 2011, pp. 26–34.

[10] L. C. Eggebrecht, *Interfacing to the IBM personal computer*, 2nd ed. Sams, July 1990.

[11] J. C. Mogul and K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," *ACM Transactions on Computer Systems*, vol. 15, no. 3, pp. 217–252, 1997.

[12] P. Druschel, L. L. Peterson, and B. S. Davie, "Experiences with a high-speed network adaptor: A software perspective," in *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, ser. SIGCOMM '94. New York, NY, USA: ACM, 1994, pp. 2–13.

[13] M. Zec, M. Mikuc, and M. Zagar, "Estimating the impact of interrupt coalescing delays on steady state tcp throughput," in *SoftCOM 2002: international conference on software, telecommunications and computer networks*, 2002, pp. 219–224.

[14] I. Ahmad, A. Gulati, and A. Mashtizadeh, "vic: Interrupt coalescing for virtual machine storage device io," in *2011 USENIX Annual Technical Conference (USENIX ATC11)*, 2011.

[15] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High performance network virtualization with sr-iov," *Journal of Parallel and Distributed Computing*, 2012.

[16] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, A. Schuster, and D. Tsafrir, "Eli: bare-metal performance for i/o virtualization," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 411–422, 2012.

[17] H. Guan, Y. Dong, K. Tian, and J. Li, "Sr-iov based network interrupt-free virtualization with event based polling," *Selected Areas in Communications, IEEE Journal on*, vol. 31, no. 12, pp. 2596–2609, 2013.

[18] M. Di Natale and H. Zeng, "Practical issues with the timing analysis of the controller area network," in *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*. IEEE, 2013, pp. 1–8.

[19] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.

[20] H. Zeng, M. Di Natale, P. Giusto, and A. Sangiovanni-Vincentelli, "Statistical analysis of controller area network message response times," in *Industrial Embedded Systems, 2009. SIES'09. IEEE International Symposium on*. IEEE, 2009, pp. 1–10.

[21] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," *Proceedings of OSPERT*, pp. 33–44, 2010.

[22] C. Herber, A. Richter, H. Rauchfuss, and A. Herkersdorf, "Self-virtualized can controller for multi-core processors in real-time applications," in *International Conference on Architecture of Computing Systems (ARCS)*, 2013, pp. 244–255.