FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Improving the analysis techniques for mixed-criticality multicore systems and controller area networks by leveraging periodic load patterns

Ishfaq Hussain
hussa@isep.ipp.pt

Doctoral Programme in Electrical and Computer Engineering

Supervisor: Dr. Muhammad Ali Awan

Co-Supervisor: Dr. Konstantinos Bletsas

Co-Supervisor: Prof. Pedro F. Souto

October 7, 2024

Improving the analysis techniques for mixed-criticality multicore systems and controller area networks by leveraging periodic load patterns

Ishfaq Hussain
hussa@isep.ipp.pt

Doctoral Programme in Electrical and Computer Engineering

Approved by:

President: Prof. José Nuno Moura Marques Fidalgo
External Referee: Prof. Rodolfo Pellizzoni
External Referee: Prof. Gerhard Fohler
FEUP Referee: Prof. Mário Jorge Rodrigues de Sousa
Supervisor: Dr. Muhammad Ali Awan
October 7, 2024

List of Publications

Conference Publications

1. Hussain, I., Awan, M. A., Souto, P. F., Bletsas, K., Akesson, B., & Tovar, E. (2019, November). Response time analysis of multiframe mixed-criticality systems. In Proceedings of the 27th International conference on real-time networks and systems (pp. 8-18).**(Best and outstanding student paper awards)**

2. Hussain, I., Souto, P. F., Bletsas, K., Awan, M. A., & Tovar, E. (2022, April). Schedulability analysis for CAN bus messages of periodically-varying size. In 2022 IEEE 18th International Conference on Factory Communication Systems (WFCS) (pp. 1-8). IEEE.

Journal Publications

1. Hussain, I., Awan, M. A., Souto, P. F., Bletsas, K., Akesson, B., & Tovar, E. (2021). Response time analysis of multiframe mixed-criticality systems with arbitrary deadlines. Real-Time Systems, 57(1-2), 141-189.

2. Hussain, I., Awan, M. A., Souto, P. F., Bletsas, K., & Tovar, E. (2022). Response time analysis of memory-bandwidth-regulated multiframe mixed-criticality systems. Journal of Systems Architecture, 123, 102346.

3. Hussain, I., Awan, M. A., Souto, P. F., Bletsas, K., & Tovar, E. (2022). Schedulability analysis for CAN bus with mixed-queue nodes and messages of periodically varying size (In process).

Abstract

Safety-critical embedded systems abound in domains such as automotive and avionics. Due to the severity of the consequences of failure of such systems (e.g., to human life, or the environment), engineers must ensure both functional and temporal correctness. Meanwhile, size, weight and power related constraints also exist, and efficient resource utilization is important as well. However, ensuring the safe execution of applications while also achieving efficient resource utilization is very challenging – and especially so, in the presence of shared resources, such as memory buses and memory controllers.

Nowadays, safety-critical systems are often mixed-criticality, which means that applications of different criticality levels are hosted on a common embedded platform. The well-known model of Vestal aims to avoid excessive pessimism in the quantification of the processing requirements of mixed- criticality systems, while still guaranteeing the timeliness of higher-criticality functions. This can bring important savings in system costs, and indirectly help meet size, weight and power constraints. This efficiency is promoted via the use of multiple worst-case execution time (WCET) estimates for the same task, with each such estimate characterized by a confidence associated with a different criticality level. However, even this approach can be very pessimistic when the WCET of successive instances of the same task can vary greatly according to a known pattern, as in MP3 and MPEG codecs or the processing of ADVB video streams.

This thesis attempts to deal with these challenges and make the following contributions:

- Initially, we devise and present a multiframe mixed-criticality model, which extends the existing mixed criticality models by allowing different jobs of a task, in the same system mode, to have different WCET estimates, according to a known repeating pattern. We also devise and present multiple response time analysis techniques, to validate the safe execution of tasks that conform to the proposed model, when scheduled on a uniprocessor platform. We performed extensive experiments with synthetic tasksets to compare the proposed analysis with the state of the art.

- In latter work, we extend the above-mentioned schedulability analyses, initially formulated for constrained-deadline tasks, to arbitrary deadlines.

- Although our schedulability analyses for multiframe mixed-criticality tasks were found very effective, they can be unsafe in the presence of shared resources when these multiframe mixed-criticality systems are deployed over MPSoC platforms. This arises from the fact that the WCET bounds for each task, which were obtained in isolation, are rendered invalid in the presence of unbounded interference from co-runner tasks over shared resources (i.e., memory bus and DRAM controller). To deal with this challenge, as a third contribution, we consider a memory access regulation mechanism (akin to Memguard), for memory access isolation among cores, and for such an arrangement we present two additional schedulability analysis techniques. Those techniques incorporate the memory access stalls introduced by the memory bandwidth regulation into the schedulability analysis of multiframe mixed criticality tasks over multicore platforms.

- In the second part of this thesis we worked on the Controller Area Network (CAN), which is a bus initially designed for in-vehicle communication. Currently, CAN is used in industrial control, railways, medical equipment, aviation and in automotive as a communication channel between different ECUs. Due to the increasing number of ECUs in automotive systems, and given the limited speed and buffer sizes of a CAN bus, the importance of efficient resource utilization is paramount. Conventional schedulability analysis assumes that the worst-case transmission time is the same for all instances of a given message, which is pessimistic when successive instances of the same message vary in their worst-case transmission time according to a known repeating pattern. We therefore extend the existing schedulability analysis techniques to specifically account for such messages (which we term *multisized*), in order to reduce the pessimism. In our first work on CAN, we assumed that, as the CAN standard itself stipulates, the highest-priority message in each node's queue will compete for transmission with messages originating from other nodes. However, in practice, many CAN implementations deviate from the standard. In such mixed

networks, some nodes use priority queues, while others use First-In-First-Out (FIFO) queues. Therefore, in subsequent work, we present two worst-case response time analysis techniques for multisized messages over CAN with mixed queues. Our experimental evaluation, considering multisized messages and both priority-queue-based and mixed queue CAN systems, demonstrates significant improvement when compared to the state-of-the-art schedulability analysis that assumes the same worst-case transmission time for all instances of a message.

Keywords: Mixed-CAcriticality systems, Multicore processors, Controller Area Network (CAN), Memguard, Multiframe tasks.

Resumo

Sistemas embebidos de segurança crítica são abundantes em domínios como o dos automóveis e aviónico. Devido à gravidade das consequências de falhas nesses sistemas (por exemplo, para a vida humana ou o meio ambiente), engenheiros devem garantir a correta execução funcional e temporal de tarefas. Ao mesmo tempo, também existem restrições relacionadas ao tamanho, peso e potência, sendo importante também a utilização eficiente dos recursos. No entanto, garantir a execução segura de aplicações e a utilização eficiente dos recursos é muito desafiador - especialmente na presença de recursos compartilhados, como barramento de memória e controladores de memória.

Atualmente, os sistemas de segurança crítica são frequentemente mistos, o que significa que aplicações com níveis de criticidade diferentes são hospedadas numa plataforma embebida comum. O conhecido modelo Vestal visa evitar pessimismo excessivo na quantificação dos requisitos de processamento de sistemas mistos de criticidade, garantindo ainda a temporização de funções de criticidade mais alta. Isso pode trazer importantes economias nos custos do sistema e, indiretamente, ajudar a atender às restrições de tamanho, peso e potência. Essa eficiência é promovida através do uso de várias estimativas do tempo de execução máximo (WCET) para a mesma tarefa, cada uma com uma confiança associada a um nível de criticidade diferente. No entanto, mesmo essa abordagem pode ser muito pessimista quando o WCET de instâncias sucessivas da mesma tarefa pode variar muito de acordo com um padrão conhecido, como em codecs MP3 e MPEG, ou no processamento de fluxos de vídeo ADVB.

Esta tese tenta lidar com esses desafios e fazer as seguintes contribuições:

- Inicialmente, apresentamos um modelo de criticidade mista de múltiplos contextos, que estende os modelos existentes de criticidade mista, permitindo que tarefas diferentes do mesmo sistema, no mesmo modo do sistema, tenham estimativas de WCET diferentes de acordo com um padrão repetitivo conhecido. Também apresentamos várias técnicas de análise de tempo de resposta para validar a execução segura de tarefas que se conformam ao modelo proposto quando agendadas numa plataforma de núcleo único. Realizamos extensas experiências com tarefas sintéticas para comparar a análise proposta com o estado da arte.

- Em trabalhos posteriores, estendemos as análises de escalonamento mencionadas acima, inicialmente formuladas para tarefas com prazo limitado, para prazos arbitrários.

- Embora as nossas análises de escalonamento para tarefas mistas de criticidade de múltiplos contextos tenham demonstrado ser muito eficazes, elas podem ser inseguras na presença de recursos compartilhados quando esses sistemas mistas de criticidade de múltiplos contextos são implantados em plataformas MPSoC. Isso ocorre porque os limites de WCET para cada tarefa, obtidos isoladamente, são invalidados na presença de interferência ilimitada de tarefas concorrentes sobre recursos compartilhados (ou seja, barramento de memória e controlador de DRAM). Para lidar com esse desafio, como terceira contribuição, consideramos um mecanismo de regulação do acesso à memória (semelhante ao Memguard), para isolamento do acesso à memória entre núcleos, e para essa configuração apresentamos duas técnicas adicionais de análise de escalonamento. Essas técnicas incorporam as interrupções de acesso à memória introduzidas pela regulação da largura de banda da memória na análise de escalonamento de tarefas mistas de criticidade de múltiplos contextos em plataformas de vários núcleos.

- Na segunda parte desta tese, trabalhamos na rede Controller Area Network (CAN), que é um barramento inicialmente projetado para comunicação veicular. Atualmente, o CAN é usado em controle industrial, ferrovias, equipamentos médicos, aviação e em mais de um bilhão de carros como canal de comunicação entre diferentes ECUs. Devido ao número crescente de ECUs em sistemas automotivos e dada a velocidade e o tamanho limitado dos buffers de um barramento CAN, a importância da utilização eficiente dos recursos é primordial. A análise de escalonamento convencional assume que o tempo de transmissão máximo é o mesmo para todas as instâncias de uma mensagem, o que é pessimista quando sucessivas instâncias da mesma mensagem variam no seu tempo de transmissão máximo de acordo com um

padrão repetitivo conhecido. Portanto, estendemos as técnicas existentes de análise de escalonamento para especificamente contabilizar essas mensagens (que denominamos multidimensionais), a fim de reduzir o pessimismo. No nosso primeiro trabalho sobre CAN, assumimos que, como o próprio padrão CAN estipula, a mensagem de maior prioridade na fila de cada nó competirá pela transmissão com mensagens originárias de outros nós. No entanto, na prática, muitas implementações de CAN divergem do padrão. Em redes mistas, alguns nós usam filas de prioridade, enquanto outros usam filas First-In-First-Out (FIFO). Portanto, em trabalhos subsequentes, apresentamos duas técnicas de análise de tempo de resposta máximo para mensagens de vários tamanhos em CAN com filas mistas. A nossa avaliação experimental, considerando mensagens multidimensionais e sistemas CAN baseados em filas de prioridade e de fila mista, demonstra uma melhoria significativa quando comparado com o estado da arte das análises de escalonamento que assumem um pior tempo de transmissão igual para todas as instâncias de uma mensagem.

Palavras-chave: Mixed-criticality systems, Multicore processors, Controller Area Network (CAN), Memguard, Multiframe tasks.

Acknowledgements

First and foremost, I would like to thank **Eduardo Tovar**, for believing in me and providing a CISTER like renowned institute for my PhD studies.

I extend my deepest gratitude to my esteemed supervisor, **Muhamad Ali Awan**, whose unwavering support, guidance, and expertise have been instrumental in shaping my research and propelling my academic aspirations. Your patience, encouragement, and intellectual stimulation have been invaluable throughout this endeavor.

I am also profoundly grateful to my thesis co-supervisors, **Konstantinos Bletsas**, and **Pedro F. Souto**, for their insightful feedback, valuable suggestions, and unwavering belief in my work. Your contributions have greatly enhanced the quality of my thesis and have broadened my understanding of the field.I would also like to thank **Benny Akesson** for his valuable feedback on some of my publications as this helped to improve those manuscripts significantly.

I would like to thank **Luis Almeida** for helping with FEUP related matters and providing valuable knowledge about real-time systems early on in my research career. I would also like to thank **Patrick Yomsi** for the valuable discussion and providing his feedback during the early phase of my PhD.

I am grateful to **David**, **Sandra**, **Ines** and **Fillipe** for helping with FCT calls and providing administrative support. Also, to **Cristiana** and **Marwin** who helped me with their day to day support and with administrative support. I would also like to thank **Gian** for translating abstract of my thesis into Portuguese.

To my fellow researchers and colleagues at CISTER, I express my sincere appreciation for your camaraderie and intellectual exchange. Our shared passion for knowledge has transformed our research environment into a vibrant learning community. My heartfelt thanks extend to **Aftab, Harrison, Kostiantyn, Shashank, Claudio, Lukas, Miguel, Mubarak, Javier, Yilian, Jatin, Gian, Enio, Jingjing, Radha, Yousef, Saied, Gowhar, Alam, Abdul, Mohammad, Reydal, Shardul, Lidia and Yimin**. I have had the privilege of learning, growing, and exploring the vast expanse of knowledge. Your collective dedication to education and research has provided me with an exceptional platform for intellectual growth.

Finally, I owe an immeasurable debt of gratitude to my family and friends, whose unwavering love, support, and encouragement have been the bedrock of my success. Your belief in me has sustained me through moments of doubt and fueled my determination to achieve my goals. This thesis is a testament to the collective efforts, guidance, and inspiration I have received from these remarkable individuals. I am deeply honored to have been a part of this community and am forever grateful for the contributions that have made this journey possible.

This work was partially supported by Portuguese National foundation for Science and Technology FCT under the individual research grant 2020.08045.BD.

Contents

*

List of Figures

List of Tables

List of Abbreviations

ECU	Electronic Control Unit
MPEG	Moving Picture Expert Group
WCET	Worst-Case Execution Time
L-WCET	Low-criticality WCET
H-WCET	High-criticality WCET
CSMA/CR	Carrier Sense Multiple Access/Collision Resolution
WCTT	Worst-Case Transmission Time
WCRT	Worst-Case Response Time
CPU	Central Processing Unit
COTS	Commercial off-the-shelf
MPSoC	Multiprocessor System-on-Chip
DRAM	Dynamic Random Access Memory
PT-AMC	Preemption Threshold Adaptive Mixed-Criticality
BNB	Branch and Bound
CrMPO	Criticality Monotonic Priority Ordering
HPA	Heuristic priority assignment
DVFS	Dynamic Voltage Frequency Scaling
EDF	Earliest Deadline First
EDF-VD	Earliest Deadline First - Virtual Deadline
FPPS	Fixed Priority Preemptive Scheduling
SCE	Single Core Equivalence
SMC	Static Mixed-Criticality
SMC-Arb	Static Mixed-Criticality with Arbitrary deadlines
SMMC	Static Multiframe Mixed-Criticality
SMMC-Arb	Static Multiframe Mixed Criticality with Arbitrary deadlines
AMC	Adaptive Mixed-Criticality
AMC-rtb	Adaptive Mixed-Criticality response time bound
AMC-rtb-Arb	Adaptive Mixed-Criticality response time bound with Arbitrary Deadlines
AMC-max	Adaptive Mixed-Criticality maximum
AMC-max-Arb	Adaptive Mixed-Criticality maximum with Arbitrary deadlines
AMMC	Adaptive Multiframe Mixed-Criticality
AMMC-rtb	Adaptive Multiframe Mixed-Criticality response time bound
AMMC-rtb-Arb	Adaptive Multiframe Mixed-Criticality response time bound with arbitrary deadlines
AMMC-max	Adaptive Multiframe Mixed-Criticality maximum
AMMC-max-Arb	Adaptive Multiframe Mixed-Criticality maximum with Arbitrary deadlines
CAN	Controller Area Network
FCFS	First Come First Serve
FIFO	First In First Out
FR-FCFS	First Ready First Come First Serve
PQ	Priority Queue
WQN	Work Conserving Queues with No reordering
WQR	Work Conserving Queues with Re-ordering

Chapter 1

Introduction and background

Many safety-critical systems host tasks of different criticality levels. For example, in an unmanned Aerial Vehicle (UAV) tasks of different criticalities, such as flight control system, navigation, mission systems, and even image recognition and weapon management can coexist. Also in automotive, high-criticality applications such as breaking, steering, and airbag control co-exist with low-criticality applications such as infotainment systems and climate control. The traditional approach has been to host each task in its own ECU. But this approach is about to hit its scalability limits. In [1], the authors report that one German premium vehicle has 70 to 100 ECUs, and as shown in Figure 1.1, the number of electric control units are expected to increase as the number of functions increase. The increasing number of ECUs will add to the weight and cost of the vehicle and also complicate the scheduling across those numerous networked nodes. In existing systems, applications of different criticality levels are usually hosted on different ECUs, leading to inefficient use of resources and higher cost.

Apart from these two examples, many real-time embedded systems in various application domains (i.e., health care, trains, and aerospace) host functions of different criticalities. Scalability and cost concerns in such systems favor mixed-criticality systems, where applications belonging to different criticality levels are hosted on a single hardware platform. Co-hosting applications of different criticalities on the same platform significantly improves system utilization. However, there exists another source of inefficiency, which comes from the task model. As observed in [3] there exist some applications in industry where the WCET of successive jobs of a task vary greatly according to a known periodic pattern. Assuming the largest WCET estimate for all the jobs for such applications brings pessimism and leads to inefficient use of resources.

It is appealing to host mixed-criticality applications with periodically repeating WCET estimates over commercial off-the-shelf multicore platforms to deal with size, weight, cost, and power-related constraints. However, the main challenge lies in the unbounded interference at shared resource levels, such as the last-level cache and shared memory controller levels. If this interference is not bounded, it might jeopardize the overall safety of the system, as a low-criticality task executing on another core might delay the execution of a high-criticality task, resulting in its deadline miss.

The worst case transmission time (WCTT) of messages in Controller Area Network (CAN) buses can also exhibit similar periodic variation patterns. Current models and analysis for CAN assume that the WCTT of a message does not change over time. By ignoring this variation and assuming only the largest WCTT for

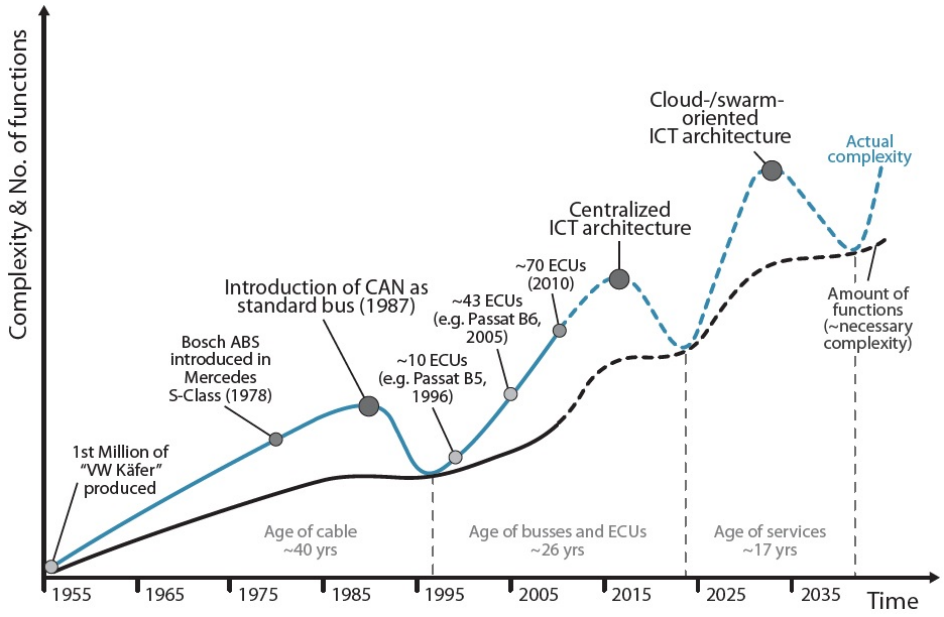


Figure 1.1: Evolution of the number of functions and of the ICT architecture complexity in the automotive domain. (Figure from [1])

successive instances of a message is pessimistic and can lead to a performance bottleneck. Also, due to the rapid increase in processing requirements of autonomous vehicles, the number of processing nodes attached to a CAN bus is also continuously increasing. This increase in the number of connected nodes demands for an increase in the bus message transmission rate, which unfortunately is not increasing at the same pace. Also, current models and analysis for CAN assume that the WCTT of a message does not change over time. As a consequence, for safety, the analysis must use the largest WCTT of a message for all its instances, resulting in an inefficient use of the CAN bus bandwidth.

This thesis attempts to address some of the scheduling challenges pertaining to the above-mentioned classes of systems, in two main directions. On one front, we focus on mixed-criticality systems where the worst-case execution times of subsequent instances of the same task vary with a known repeating pattern and leverage this information to remove the pessimism in the schedulability analysis, compared to state-of-the-art approaches that are agnostic to such variations. This new task model is called the *multiframe mixed-criticality* model, and we further formulate schedulability analysis for such tasks in multicore systems that additionally employ per-core memory access regulation, for higher timing predictability. On the other front, we focus on the scheduling of messages over a CAN bus, that display similar variation of their worst-case transmission times, analogously to multiframe tasks and their execution times. For such messages, which we term *multisized*, we similarly formulate novel schedulability analysis that leverages such variation patterns to eliminate much of the pessimism inherent in the application of state-of-the-art approaches (which are oblivious to them).

1.1 Thesis contributions

1.1.1 Multiframe tasks in uniprocessor mixed-criticality systems

A mixed-criticality system is a system which allows tasks of different criticality levels to co-exist on the same platform. These criticality levels are defined as ASIL in automotive and DAL in avionics. A crucial requirement in a mixed criticality system is to ensure that the execution of low-criticality tasks does not jeopardize the execution of high-criticality tasks. As shown in Table 1.1, an erroneous behavior by an A-assurance level task may result in loss of aircraft but on the other hand an erroneous behavior by a D-assurance level task may result in inconvenience. Thus tasks with A-assurance level must be given more stringent certification guarantees than E-assurance level tasks. For systems with tasks of different criticality levels co-executing over the same platform, Vestal [4] presented a multi-criticality task model. The model uses different worst-case execution time (WCET) estimates for a given task, reflecting the confidence levels associated with different criticality levels. I.e., for the same task, it considers both pessimistic WCET estimates (derived under the conservative techniques appropriate for high-criticality tasks) and less pessimistic ones (derived with the lower level of required confidence associated with-lower criticality tasks). Vestal conjectures that "the tighter the WCET bound or more assured the WCET estimate, the more effort required to obtain the bound" [4]. His schedulability analysis uses as inputs the WCET estimates with the degree of confidence appropriate for the criticality level of the the task under analysis.

A mode-based variant of Vestal's model [4] was later presented by Baruah et al. [5]. In this model, only tasks of certain criticality or higher are allowed to execute in a given mode of operation. Different WCETs are assumed for the same task in different modes of operation for efficient processing capacity utilization. System modes are indexed by the lowest task criticality present in them. In a given mode, for a task of higher criticality than that mode's nominal level, less pessimistic WCET estimates are assumed than the pessimistic ones befitting that task's criticality. As soon as any task overruns its assumed WCET, the lowest-criticality tasks present are abandoned/stopped; and the system switches to the next mode, assuming more pessimistic WCETs for the remaining tasks.

The simplest mixed-criticality adaptive model [6] assumes two criticality levels and two modes of operation (L and H). In the L-mode, all tasks are present, while in the H-mode only tasks of higher criticality are allowed to execute. This approach significantly reduces the resource use inefficiency from always assuming provably safe but very pessimistic WCET estimates for high-criticality tasks. However, another source of pessimism comes from the task model. Specifically, for some applications where the WCETs of successive jobs of the same task vary greatly by design, according to a known pattern. For such applications, assuming the maximum of the WCETs for all instances of the task (even the ones for which we *know* it is an overestimation) would grossly inflate the processor requirements. For example, in MPEG codecs [7], different kinds of frames (P, I or B) appear in a repeating pattern, with very different worst-case processing requirements. Moreover, Avionics Digital Video Bus (ADVB) [8] frames are transmitted uncompressed, to minimize encoding/decoding delays, but the fact that distinct types of ADVB frames exist (e.g., data, audio or video) implies different end-node processing requirements for each type. In other real-time industrial applications [9, 10] small amounts of data are collected periodically and then summarized and stored in batch after N periods, in an operation that involves costlier processing than that in the preceding periods. The *multiframe task model* was proposed by Mok and

Chen [3] precisely for systems with this kind of WCET variations. The schedulability analyses [3, 11, 12] for this model provide a way for efficiently dealing with patterned WCET variations in single-criticality scheduling. However, until now this model and its existing analyses did not consider the scheduling of mixed-criticality tasks.

The present work combines the mixed-criticality model of Vestal with the multiframe model, in order to achieve similar improvements in the schedulability testing of mixed-criticality systems whose tasks' WCETs vary according to known patterns. Our three main contributions are the following:

- The multiframe task-model is combined with the mixed-criticality Vestal model. It is termed the multiframe Vestal model for mixed-criticality systems, in its static and mode-based variants.
- We developed multiple multiframe mixed-criticality schedulability analyses for fixed-priority-scheduled mixed-criticality tasks deployed on a uniprocessor hardware platform.
- In experiments with synthetic workloads, the proposed analyses are compared in terms of scheduling success ratio, against the frame-agnostic analyses for the corresponding variants of the Vestal model.

The schedulability analysis techniques mentioned above for multiframe mixed-criticality fixed-priority-scheduled systems were initially formulated only for the constrained-deadline model [13]. In a follow-up work [14], we presented three additional analysis techniques (SMMC-arb, AMMC-rtb-arb and AMMC-max-arb) which apply to the more general arbitrary-deadline task sets. We also demonstrated that Audsley's priority assignment, which was optimal under several frame-agnostic schedulabulity analyses, is also optimal for our proposed schedulability analyses. Evaluation with synthetic task sets demonstrated that AMMC-max-arb outperforms all the other analyses (frame-agnostic or multiframe; for constrained- or arbitrary-deadline tasks), albeit at the cost of increased computational complexity.

1.1.2 Multiframe tasks in mixed-criticality systems on multiprocessors with shared resources

Commercial-off-the-shelf (COTS) multiprocessor systems-on-chip (MPSoC) are an increasingly popular choice for mixed-criticality systems due to the following reasons. These platforms are easily available on the market and offer significant advantages over single-processor architectures due to lower energy consumption, higher computing power, lower weight and smaller size. MPSoCs are designed to improve the average-case performance and share multiple resources for efficient resource utilization. These shared resources include caches, memory controller, memory bus and I/O devices.

The applications running on these platforms (on the same or different core) may interfere with each other's access to such shared resources, which may jeopardize the schedulability guarantees derived by analysis that examines those applications in isolation. Indeed, most of the theory developed for timing analysis of single-core processors relies on the assumption that, when a task is executing with the processor all to itself, its worst-case execution time is well-defined and bounds on it can be inferred from the state of the processor and the process (task). This does not hold any more on multicore processors, because the interactions with other cores make the worst-case execution time depend on their state as well. Additionally, the integration of applications with different criticalities on the same multicore platform in order to increase resource utilization, brings the risk of a

Table 1.1: Design Assurance Levels (DAL) specified in [2].

Condition	Level	Description
Catastrophic	A	Failure result in loss of airplane or multiple fatalities.
Hazardous	B	Failure has a large negative impact on safety or performance and may result in serious or fatal injuries to the occupants.
Major	C	Failure that reduce the aircraft capability or crew ability to operate in the adverse conditions.
Minor	D	Failure that has no effect on the safety of airplane. It may result in passenger inconvenience or routine flight plan change.
No Effect	E	Failure that has no effect on aircraft safety, operation or crew workload.

faulty task affecting the timeliness of a higher-criticality task, unless mechanisms to avert this exist. Of course, these challenges need to be addressed efficiently using the resources available on the platform.

In [15], we extended our multiframe mixed-criticality schedulability analyses for single-core platforms (discussed in Section 1.2.1) to COTS multicore platforms. To mitigate the interference on shared memory controller and memory bus, we considered a memory access regulation mechanism akin to MemGuard and assumed an even distribution of available memory bandwidth among cores. Within each regulation period, if a core tries to use more than the assigned budget it is simply stopped until the start of the next regulation period. We presented two different schedulability analysis techniques for both the static and the adaptive variant of the multiframe mixed-criticality model. In the simpler schedulablity analysis (presented in section 5.4) an aggregated interference from all the higher priority tasks is considered. Whereas the tighter analysis (presented in section 5.5) provides less pessimistic response time bounds by considering information about the release patterns of higher priority tasks in the response time computation process. A pruning method was also presented, for use with the tighter analysis to reduce the running time of the analysis, by eliminating from consideration, in the computation process, certain frame sequences that are subsumed by others. Our evaluation, using synthetic task sets, demonstrates up to 72% improvement in terms of schedulability ratio, when the proposed analysis technique is compared to the frame-oblivious analysis.

1.1.3 Messages with periodically varying size over Controller Area Network (CAN) bus

The CAN bus (“Controller Area Network") is a communication bus architecture, standardized as ISO 11898, that is ubiquitous in automotive systems. The original motivation behind CAN was to replace many kilometers of physical wire connecting the ever-more-numerous sensors and electronic control units with a lightweight and robust logical interconnect. It uses priority-based arbitration, which provides real-time guarantees and is amenable to schedulability analysis. There currently exist more than a billion (10^9) automotive systems with CAN buses in them [16]. The estimated number of CAN-enabled controllers sold in 2011 alone was around 850 million [17]. There are more than 20 CAN networks used in modern trucks and 70 to 100 CAN connected ECUs in advanced vehicles [1]. An ECU can be used for a variety of different applications, including engine management system, advanced driver assistance system, airbag management, ABS, cruise control, infotainment, and many other functions. The number of transmitted messages over the network can reach up to 6000

messages [18]. As becomes obvious from the above figures, the significance of CAN in the automotive domain is tremendous. It is also used in industrial control, medical equipment, trains, and many other domains [19].

There is a trend towards a higher number of ECUs in modern vehicles, driven, among other factors, by the increase in the number of sensors and the push towards self-driving cars. This, in turn, increases the communication bandwidth requirements. However, the maximum bandwidth offered by CAN is fixed (i.e., 1 Mbps for CAN and 10 Mbps for CAN-FD). The problem is further accentuated because of the constrained power resources in battery-operated vehicles. Therefore, novel schedulability analysis techniques are needed, which increase CAN utilization while still guaranteeing that the timing requirements are met.

Several works have already been presented in the literature proposing schedulability analysis techniques for CAN-based systems [20, 21, 22]. Those works assume that the same worst-case size for all instances of a given message. However, in some applications, the size of a given message may vary according to an *a priori* known repeating pattern. For example, consider a CAN node which collects payload readings from different sensors with periods that are multiple of one of them (Figure 1.2). Only one sensor is read in each time period T and this process is repeated periodically.

The CAN bus designer has two alternatives. First, to use one message per sensor reading. Second, to send the readings of the three messages in a single message. The former approach has the disadvantage of higher bandwidth overhead, because of the reduplication of message headers and the elevated message contention. Additionally, there might not always exist enough available message ids, so they would need to be used sparingly. On the other hand, by using the later approach same message (and a single id) can be used to piggy-back multiple sensor values, with the message length varying accordingly. However, the current analysis will then have to conservatively always assume the maximum possible message length, leading to pessimism. Our approach, by comparison, allows for using a single message id, in such cases, without any of the analytical pessimism that the state-of-the-art analysis incurs.

In [23], we therefore introduced the *multisized message model* for CAN, which allows for the worst-case payload size of different instances of a message to vary according to a known repeating pattern. By analogy, the multisized message model is to CAN what the multiframe[1] task model is to real-time task scheduling. Note that our contributions are purely at the analytical level, and do not involve any changes at all to CAN itself or its implementations. After all, CAN does not restrict messages of a given id to always have the same length. In [23], we presented two different schedulability analyses for multisized CAN messages. The first approach upper-bounds the aggregated bus utilization of each higher-priority message within the busy period of the message under analysis. This improves the schedulable CAN bus utilization, compared to conventional analysis. The second approach achieves further improvement by considering the actual possible sequences of message sizes, when computing the busy period length and the upper bound for the message response time.

The CAN standard prescribes priority-based arbitration in the transmission queues of all nodes and this is what [23] assumed. However, one challenge in the timing analysis of CAN is the existence of CAN controller implementations that use different arbitration methods. For example, in some networks, a mixed-queue policy is used, where some nodes use priority queues while others use FIFO arbitration. The existing analysis techniques for such mixed-queue CAN networks [24, 21] are pessimistic as they do not consider multisize messages. We therefore extended (Chapter 6) our analysis for multisized messaged to mixed-queue CAN implementations.

[1]The term “frame" therein carries specific meaning, not to be confused with the concept of transmission frames in communication protocols, such as CAN itself.

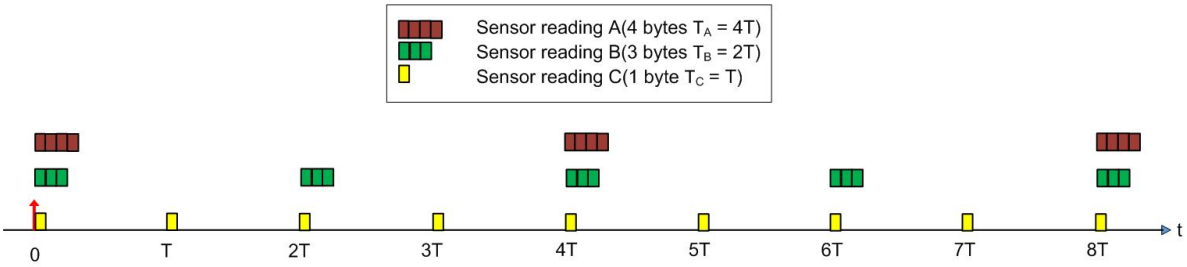


Figure 1.2: The transmission of different sensor readings with periods that are multiple of each other can be combined on the same message; the size of message instances will also vary periodically.

Extensive evaluation, using synthetic workloads, demonstrates the performance improvement from the proposed approaches compared to the state-of-the-art analysis.

1.2 Thesis statement

With this work, we aim to defend the following thesis:

It is possible to efficiently utilize system resources while simultaneously ensuring the timing predictability of the system – both in the case of real-time tasks of different criticalities, concurrently executing over a single-core or multicore platform, and also in the case of messages transmitted over a Controller Area Network, by introducing novel task (resp. message) models and techniques for worst-case response time analysis for applications that have periodically repeating WCET (resp. WCTT) variation for successive jobs (resp. messages).

1.3 Thesis structure

The remainder of this thesis is organized as follows:

- Chapter 2 provides a survey of relevant literature, with emphasis on the state-of-the art for multiframe tasks, mixed-criticality systems, mitigation of contention upon accesses to shared resources on multicores, and finally the CAN bus.
- Chapter 3 introduces the system models, terminology and notation used throughout this thesis, and details our experimental setup.
- Chapter 4 presents our schedulability analysis techniques for the multiframe mixed-criticality task model on a single-core platform, and their experimental evaluation by randomly generating tasksets.
- Chapter 5 presents the extension of the aforementioned techniques to multicore systems that incorporate memory bus access arbitration, and also provides evaluation.
- Chapter 6 introduces our schedulability analysis techniques for the multisized CAN message model. We consider both standards-compliant CAN implementations with priority queues and mixed-queue implementations (where some nodes have priority queues and other have work conserving queues). Again, experimental evaluation is provided.

- Chapter 7 concludes this dissertation and outlines some possible directions for future research.

Chapter 2

State of the art

In the previous chapter, we highlighted some of the issues and potential solutions pertaining to pessimism for mixed-criticality multicore systems and controller area networks. This chapter provides a review of related work on the real-time task models, mixed-criticality systems, and controller area networks.

We begin with different task models and analysis techniques in Section 2.1. Section 2.2 discusses various memory management and schedulability analysis techniques for multicore processors. We then provide a general overview of mixed-criticality scheduling techniques over uni/multicore platforms in Section 2.3. Section 2.4 presents schedulability analysis techniques for CAN with different nodes and message configurations.

2.1 Task models and analysis techniques

Over the years many different models have been used to describe the timing behavior of real-time applications and to specify their timing requirements. This section is further subdivided into two subsections; where the first subsection discuss different type of task models, scheduling and priority assignment techniques and second subsection discuss multiframe tasks and respective schedulability analysis techniques.

2.1.1 Real-time task models

For real-time industrial control applications, Liu and Layland [25] presented a simple periodic task model. Each task τ_i in the task set $\tau = \{\tau_1, \tau_2, \tau_3, \ldots \tau_n\}$ is represented by its worst-case execution time C_i (upper bound on the execution time of an application when executing alone on the processor) and interarrival time T_i (time period between two consecutive arrivals of that task). All the *jobs* (i.e., instances) of a task τ_i are assumed to have the same WCET estimate C_i and a relative deadline D_i equal to their period T_i. A task's utilization is defined as $\frac{C_i}{T_i}$ and the utilization of a task set is the sum of the utilizations of its tasks. The periodic task model of [25] was extended to the more general sporadic task model by Mok et al. [26]. Each task τ_i in this model is represented by its WCET C_i, minimum interarrival time T_i, and relative deadline D_i. The interarrival time between two consecutive arrivals of the task is at least T_i, but potentially arbitrarily larger. The deadline of the task was also defined more flexibly, as it could be either implicit ($D_i = T_i$), constrained ($D_i \leq T_i$), or arbitrary (which allows $D_i > T_i$).

For the uniprocessor scheduling of real-time tasks, Liu and Layland, already in [25], came up with what is now known as fixed-priority (preemptive) scheduling, whereby each task has a fixed priority, characterizing all its jobs, and at any time the job with the highest priority, among all ready jobs, is selected for execution on the processor. Liu and Layland considered a *rate monotonic* priority scheme (also their invention), whereby a task with a higher arrival rate will have higher priority. For this problem, they developed a schedulability analysis. That sufficient schedulability test is based on the overall system utilization and compares the latter with a threshold, parametric to the number of tasks (decreasing with the latter, asymptotically to 69%). If the system utilization does not exceed that threshold, then the system is deemed schedulable.

Liu and Layland [25] and Labetoulle [27] showed that, for implicit-deadline uniprocessor systems where the tasks have no fixed (or known) release offsets and do not self-suspend, rate monotonic is an optimal fixed priority assignment scheme. This means that, if a such a system is schedulable under any other task priority assignment, it is then also schedulable under rate monotonic. Moreover, for such systems, the worst-case scenario, maximizing the response time for all tasks, as shown by Liu and Layland [25], is the so-called *critical instant*, under which tasks initially arrive at the same time. For systems whose worst-case scenario corresponds to such a critical instant the closed-form worst-case tasks response time recurrence equations by Joseph and Pandya [28] apply. *Deadline monotonic* is a generalisation of rate monotonic, and it is optimal for constrained-deadline tasks, under the same conditions, as shown by Leung and Whitehead [29]. However, for arbitrary-deadline tasks or with tasks with (known) initial release offsets, deadline monotonic is not optimal. For such systems, Audsley came up with an optimal bottom-up priority assignment algorithm [30, 31]. Starting at the lowest priority level, it tries out all tasks until one is found schedulable with that priority; it then moves to the next-highest priority level and considers the remaining tasks; and so on.

Audsley et al. [32] further extended the periodic model with the jitter parameter, denoted by J_i and modeling the possibility of a task being released later than its nominal arrival time, by up to J_i time units. Jitter makes it possible for the releases of two consecutive jobs of a task to occur commensurately closer together in time than its minimum interarrival time. For tasks that arrive in bursts, Tindell et al. [33] came up with the bursty (periodic or sporadic) task model, which features two periods, inner and outer, for each task. Where the inner period represents the minimum separation time between consecutive arrivals within the bursts while the outer period represents the minimum inter-arrival time between bursts.

In the same seminal paper where Liu and Layland introduced their task model and fixed-priority scheduling [25], they also introduced another classic scheduling policy, namely Earliest Deadline First (EDF). Under EDF, ready jobs are ordered according to their absolute deadlines, and, at any time, the job with the earliest deadline is selected for execution. This means that prioritisation is at the job level, not at the task level. Additionally there is no need for a separate priority assignment step, as in fixed (task) priority scheduling, as this is taken care of on the fly, whenever jobs arrive, by maintaining the run queue sorted in order of absolute deadline. Preemptive EDF is an optimal hard real-time scheduling algorithm on uniprocessors; meaning that, if any task set is schedulable at all under any policy, then it is (also) schedulable by EDF. Additionally, in the case of implicit-deadline uniprocessor systems, its utilisation bound is 100%. Nevertheless, despite such advantages, EDF is not as widely adopted as fixed-priority scheduling, in industrial practice.

2.1.2 Multiframe task models

Mok and Chen [3] presented the example of MPEG codecs, which use three different types of frames: I, P and B. The period/minimum interarrival time between any two successive frames, of the same or different type, is the same. However, I-frames are larger (and more complex to decode) than P- and B-frames. The WCETs of the different frame types are very different, but the standard sporadic task model would model all the frames using the WCET of an I-frame, leading to severe overestimation of processing requirements. Therefore, a task which is actually schedulable might be declared unschedulable because of these pessimistic assumptions. Therefore, to remedy such pessimistic representation by existing task models, Mok and Chen [3] presented the *multiframe* task model, for applications where different invocations of the same task might have different WCET requirements, in a periodically repeating fashion. This variable worst-case execution time requirement is represented with a vector of worst-case execution times $\vec{C}$ in the task representation, rather than a single value C, as in the classical task model of Liu and Layland [25] and its immediate derivatives.

In the terminology of the multiframe task model, if the WCETs of a given task repeat every F jobs, where F represents smallest integer such that the WCET of k^{th}, $(k+F)^{th}$, $(k+2F)^{th}$,... jobs have same WCET estimate, then a sequence of F successive jobs by that task is called a *superframe* and each of those F jobs represents a *frame* by that task. Since the pattern is repeating, which frame is the first one is a matter of convention.

Mok and Chen [3] came up with schedulability analysis for multiframe tasks, which extended the standard WCRT recurrence for fixed-priority sporadic or periodic tasks. That analysis was based on a property called *accumulative monotonicity*. A multiframe task is accumulatively monotonic if the interference exerted by it (i.e., on a lower-priority task) is always maximized if its *peak frame* (i.e. the one with the greatest WCET) is released at the same instant as the task under analysis, irrespectively of the number of interfering jobs by the interfering task. A task that is not accummulatively monotonic, can be modelled as one, with some pessimism, by rearranging its frames in order of non-increasing WCET. Mok and Chen showed that, when used in conjunction with their analysis, rate monotonic priority assignment was optimal for implicit-deadline multiframe tasks. The authors also propose a utilization-based test, under rate monotonic, that considers the *peak utilization* of each task (i.e., WCET of its peak frame divided by the task period) and the maximum ratio, over all tasks, between the peak frame WCET and second-greatest frame WCET.

Han [34] demonstrated that many feasible multiframe task sets with peak utilization larger than one might be declared unschedulable by [3]. They proposed a test which transforms the original system to a system with harmonic task periods and later applies the test over the modified system and verifies the schedulability of the original task set. Although Han's analysis mischaracterized fewer schedulable task sets as infeasible, the proposed analysis was only applicable to periodic task sets, as compared to [3], which applies to both periodic and sporadic task sets. Kuo et al. [35] further improved that analysis by reducing the number of tasks, in a modified task set. The proposed methodology combines the tasks with harmonic period and later applies a simplified utilization-based schedulability test, for the purpose of task admission control. The proposed schedulability test is applicable to both multiframe and classic (non-multiframe) tasks.

Baruah et al. [11] presented a response-time-based schedulability test for non-accumulatively monotonic multiframe task sets. The analysis computes the response time for the peak frame of each task by incorporating the maximum cumulative interference from each of the higher-priority tasks. The improvement in accuracy is demonstrated via an example, by comparing the proposed approach with [3].

All the analyses presented so far for multiframe tasks are sufficient. Some are less pessimistic than the others, however, none of them is exact. Zuhily et al. [12] presented a more complex but exact schedulability analysis for non-accumulatively monotonic multiframe tasks with rate monotonic priority assignment. The authors also proved the intractability of the problem and presented a pruning method to remove the non-critical frames in the response time computation process. (A non-critical frame is a frame that cannot lead to maximum cumulative interference, when it is the first interfering frame, irrespective of the number of jobs by that task interfering with the task under analysis.)

The generalized multiframe task model by Baruah et al. [36] is a generalization of the multiframe task model and differs from it in the following: Different frames can have different deadlines and periods (and the former need not be equal to the latter). Therefore, a generalized multiframe task is represented with 3 tuples $\{\vec{C}_i, \vec{D}_i, \vec{T}_i\}$, where $\vec{C}_i = \{C_{i,0}, C_{i,1}, C_{i,2}, ...C_{i,N}\}$ represents a vector of worst-case execution times, $\vec{D}_i = \{D_{i,0}, D_{i,1}, D_{i,2}, ...D_{i,N}\}$ a vector of relative deadlines and $\vec{T}_i = \{T_{i,0}, T_{i,1}, T_{i,2}, ...T_{i,N}\}$ a vector of period/minimum inter arrival times, respectively. Both the multiframe and the generalized multiframe model follow a cyclic execution pattern. Moyo et al. [37] and Baruah et al. [36] presented a non-cyclic generalized multiframe model which models a workload that does not follow a periodic arrival of different frames.

2.2 Systems with memory bandwidth regulation

Commercial-off-the-shelf (COTS) multicore platforms offer high performance and help to address size, weight, and power consumption-related challenges. However, dealing with timing predictability in the presence of shared resources for such platforms is challenging. This is because single-core timing verification techniques, which require WCET estimates for the applications and later use these estimates for schedulability tests are no longer valid. The reason is that the WCET bounds which were obtained in full isolation for each task do not hold anymore because of the interplay of co-runners (tasks executing in parallel over other cores) at the shared resource level. For example, a request to a memory bus can face additional delays because of an ongoing request from another core. Unbounded interference by co-runners at the shared resource level might cause additional delays, which may lead the system to unpredictable behavior.

In COTS platforms, the DRAM memory system is composed of two subsystems; a memory device to store the data and a memory controller to handle the requests from cores or DMAs. An address bus and a data bus are used for the communication between a controller and the memory device. These memory devices are further organized into different ranks, whereas each rank consists of parallelly accessible banks, provided no conflict occurs at either bus. Each bank consists of storage lines organized as rows and columns and a row buffer to access these storage cells one at a time. For applications with higher row buffer locality an open row policy is suitable to serve the memory requests in lesser time. As the time to serve a request which is already cached in the row buffer is much smaller than a request which needs to be cached in the row-buffer.[1]. In a multicore scenario, two applications executing in parallel that need to access two different rows in the same bank, require the controller to load the new row every time, potentially inflating the access time. Modern multicore processors

[1]Regarding the memory management of the row buffer memory controller can employ either an open or close row policy or any proprietary in-between policy between the two extremes. In the open row policy, the memory buffer is kept open as long as possible. However, in the close row policy, the memory controller automatically pre-charges the row buffer after every request.

employ out-of-order execution of memory requests to increase the row buffer hit ratio. However, because of this reordering policy row buffer queuing delay of the core under analysis may increase significantly.

To avoid unbounded interference at the memory bank level while still allowing parallel access, some work suggests private banking [38], where cores are given exclusive access to one or more banks. In this way, interference at the bank level is eliminated, as no core can close a row opened by another core. The work by Reineke et al. [39] uses a close row policy with TDMA arbitration while the work by Wu et al. [40] uses an open row policy in conjunction with FCFS scheduling. The proposed approaches are highly inflexible because the bank partitioning is performed at design time and requires hardware support, which is not available in COTS platforms. To overcome this, Yun et al. [38] presented PALLOC, an OS-level memory allocation scheme that is aware of the operation of the platform DRAM banks. Unlike previous approaches, the proposed methodology allocates the memory row of each application to specific banks without any hardware support. To avoid different applications (cores) competing for access to DRAM memory, PALLOC assigns disjoint sets of DRAM banks to each application or core. Since the process assigned to one DRAM partition cannot access more than the assigned DRAM banks, a carefully designed bank assignment methodology is required because otherwise, PALLOC will result in an inefficient use of memory. Some other works before PALLOC also presented OS-based DRAM bank assignment [39, 40] but the objective was mainly either to increase the overall system throughput or reduce the energy consumption.

2.2.1 Bounding the bus contention

The co-runner tasks hosted by the competing cores can also contend for memory access at the bus level. To bound the interference at the bus level, various bandwidth regulation techniques and schedulability analyses incorporating the stall caused by the regulation methods are discussed in this section. In these methods, each core is assigned a replenishable budget Q (representing the maximum number of accesses a core can make) of total memory bandwidth. A core is halted until the next regulation period P if it exhausts its budget. Several hardware and software-based techniques are proposed in the literature for memory bandwidth regulation [41, 42, 43].

An OS-based memory regulation method was first presented by Bellosa [44]. The author presented multiple memory throttling implementation schemes to regulate the memory bandwidth of soft real-time applications. Yun et al. [45] presented a software-based memory throttling mechanism for mixed-criticality systems. The workload was partitioned between critical core (hosting hard real-time applications) and non-critical cores (hosting soft real-time applications). This work regulates the memory bandwidth of non-critical cores only. The memory demand of non-critical cores is continuously monitored using performance monitoring counters (PMCs) and if the budget exceeds a value Q, the core is halted by dequeuing all the tasks in the run queue until the next regulation period. At the beginning of every period, the scheduler replenishes the memory budget of each core and re-enqueues any pending previously dropped tasks. An analytical solution was also proposed to find the throttling parameters Q and P, for the interfering cores such that the schedulability of critical tasks over critical cores is satisfied with minimum performance degradation of the tasks running on non-critical cores. Two different memory budget assignment techniques were presented: static and dynamic. In the static approach, the memory budget Q of each interfering core is assigned statically at the beginning, while in the dynamic approach, budget assignment is performed based on the memory demand of each core dynamically. The evaluation of the

throttling mechanisms (both static and dynamic) demonstrated that the dynamic regulation scheme provides better performance for non-critical cores as compared to static assignment because these cores are throttled for less time.

In [46] Yun et al. presented MemGuard, which extended their previous work [45] and regulates the memory bandwidth of all the cores, not just the cores hosting soft real-time applications. In this work, two different regulation mechanisms (static and dynamic) were presented to provide isolation meanwhile achieving higher performance throughput. In the static approach, MemGuard partitions the minimum allowed DRAM access rate r_{min} among the different cores. This was achieved by first profiling the applications and later using PMCs to ensure that each application does not exceed the assigned bandwidth budget Q_i. Once the budget is depleted, the regulator requests the OS scheduler to dequeue all the pending requests from the core under analysis until the next regulation period. At the beginning of every period, the memory budget is replenished and, if there exist any dequeued applications for the respective core, those are enqueued again. Since, in the static approach, the total memory DRAM budget was only consumed up to the minimum DRAM access rate, the dynamic regulation mechanism tries to improve the overall system throughput by exploiting the spare memory bandwidth exceeding r_{min}. However, since the dynamic approach does not guarantee the assigned bandwidth, its use is not recommended for hard real-time systems.

Yao et al. [47] presented a schedulability analysis for MemGuard-based memory bandwidth-regulated multicore systems. Each core is assigned with a budget Q in the regulation period P. This work assumes WCET estimates comprising separate upper bounds for memory accesses and (net) computation time, though memory accesses and computation need not be in separate phases, as assumed in PREM [48] or AER [49] task models. A round-robin memory controller policy with a granularity of one time unit (i.e., any memory request can suffer interference at most once, from each competing core) was used to arbitrate the memory access requests coming from different cores. This arbitration policy provides isolation at the shared resource level and thus allows to compute the stall value without the knowledge of co-runner tasks. The schedulability analysis is performed based on synthetic tasks, which include the effect of interference caused by all the higher-priority tasks in the respective response time window. This work assumes a single memory controller and a constant memory access time for each memory request.

Mancuso et al. [50] presented a Single-Core Equivalence (SCE) framework, which provides system-level isolation. The SCE mechanisms are used to achieve isolation at the cache, DRAM bus, and shared DRAM bank level. Initially, the application is profiled to determine the memory rows that have a significant impact on the WCET and later these rows are locked in the caches to avoid any cross-core interference. Memguard [46] is used to bound the interference at the bus level. It assigns a specific budget to each core and restricts the core from further access to the bus until the next regulation period, if the budget is depleted. Finally, to exploit the parallel access to DRAM banks PALLOC is used, which assigns disjoint banks to different applications. The response time of each application is computed by integrating the additional delay caused by the memory regulation, assuming an equal distribution of bandwidth among different cores. Later, Mancuso et al. [51] also extended the SCE approach for MemGuard with uneven budget assignment for different cores. A method for computing the WCET estimates as a function of assigned memory budgets and a schedulability analysis using the estimated WCET values were also presented. In the empirical evaluation, 30% improvement in WCET estimates was demonstrated when comparing the uneven budget assignment with the even assignment.

The schedulability analysis methods presented above make certain assumptions about the inner workings

of DRAM systems. The validity of the system model directly depends on the underlying assumptions. These analysis techniques treat the memory system as a single unit using simple arbitration schemes such as round robin, FIFO or TDMA [52, 39, 40]. However, complex COTS systems have DRAM architectures that allow banks to be accessed in parallel. Moreover, an FR-FCFS arbitration scheme is commonly used, which prioritizes requests that are already available in the row buffer. To overcome these problems, Pellizzoni et al. [53] presented a schedulability analysis for memory bandwidth-regulated multicore systems with a FR-FCFS arbitration scheme. The proposed method uses a memory server that provides isolation among tasks of different criticality levels and enables the use of compositional analysis techniques that do not require information about co-runner tasks executing on other cores. Rather than using one parameter for memory regulation (as considered in MemGuard [41]), the proposed model uses two parameters, Q_p (memory budget of core under analysis) and QB_p (memory budget that can be assigned to other cores).

All the throttling techniques and schedulability analyses discussed so far consider only one memory controller over which to apply memory request arbitration. However, these analysis techniques are unsafe for architectures with multiple memory controllers, which are developed to satisfy the increasing memory bandwidth demands from memory-intensive applications. Awan et al. [54] presented a schedulability analysis for partitioned fixed-priority tasks scheduled over memory-bandwidth-regulated (via software or hardware) COTS platforms with multiple memory controllers. It was assumed that all the controllers are shared, so each core i was assigned with a replenishable memory budget Q_{ji}, which represents the maximum number of accesses of core i via controller j. The proposed analysis assumes that each memory access takes constant time and memory access or CPU computation can happen at any point in the task execution (i.e., no explicit memory and computation phases are required). As in the work of Yao et al. [55], various worst-case execution patterns for memory accesses and processing time were identified and worst-case stall terms and schedulability analysis were subsequently derived for those patterns.

2.3 Mixed-criticality scheduling (MCS)

Vestal's original mixed-criticality model [4] and its later variants (static [5] and adaptive mode-based [6]) characterize each task with multiple worst-case task execution time (WCET) estimates, with a corresponding degree of confidence associated with a different criticality level. This reflects the fact that, in the industry, the enormous costs of proving the safety of a WCET estimate (and the associated pessimism/overestimation) beyond doubt are justified only for high-criticality tasks [4]. For other tasks, less rigorously derived, WCET estimates are used – probably, but not provably, safe. Based on this model, several mixed-criticality scheduling arrangements have been devised for a variety of hardware platforms in the last decade [56, 57, 58, 59]. Most of these works are summarized in a survey paper by Burns and Davis [60].

In his seminal paper, Vestal considered fixed-priority-scheduled uniprocessors systems and relied on the well-known WCRT equations by Joseph and Pandya [28]. However, he advocated the use, in those WCRT equations, of WCET estimates (both for the task under analysis and also for the interfering tasks) with a degree of confidence (i.e., pessimism) matching the criticality of the task under analysis. For example, assuming just two criticality levels, each task has a "reasonable" but not provably safe, WCET estimate and a demonstrably

safe, but very pessimistic one. Then when testing the schedulability of a low-criticality task, the WCRT equations would use the "reasonable" estimates, for all tasks, as inputs; but when the testing the schedulability of a high-criticality task, the very pessimistic estimates ought to be used.

Baruah and Burns [5] first proposed the use of execution-time monitoring for mixed-criticality systems, which was already supported in most embedded platforms. Any job by a lower-criticality task requiring more execution time than its most conservative WCET estimate, is terminated. Baruah, Burns and Davis [6] showed how standard worst-case response time analysis can be applied to this model, (termed “Static Mixed-Criticality" or SMC), by assuming, for all tasks, WCET estimates associated with a criticality level never higher than that of the task under analysis. Baruah, Burns and Davis [6] also proposed an "Adaptive Mixed-Criticality" (AMC) scheduling technique that considers different modes of system operations. When operating in a specific mode, if any task attempts to exceed its assumed WCET estimate for the respective mode, all tasks with a criticality level lower than or equal to the the one corresponding to the current mode of operation are discarded and the system switches to the next-highest-indexed mode (with stricter WCETs assumed for the remaining tasks). Two schedulability tests for AMC were devised: AMC-rtb and the tighter, but more complex, AMC-max. AMC-max provides better schedulability performance as compared to AMC-rtb and SMC by considering all possible mode-switch instants in the response time computation process. However, it still does not provide exact worst-case bounds, and it is computationally complex in the case of multiple criticality levels. It was also demonstrated that Audsley’s priority assignment algorithm [31] is optimal, in conjunction with the schedubalility tests for both SMC and AMC.

Real time scheduling algorithms can be broadly categorized into preemptive or non-preemptive. One class does not dominate the other in terms of schedulability due to various reasons (i.e., preemption overhead, priority inversion etc.) Limited preemptive scheduling has emerged as a center point between the above two extremes. Limited preemptive approaches, in general purpose scheduling (i.e., not mixed-criticality), include Preemption Threshold Scheduling (PTS) [61], Fixed Preemption Points [62] and Deferred Preemption Scheduling [63].

In the context of mixed-criticality scheduling, Zhao et al. [64] extended AMC-rtb by incorporating preemption thresholds. The proposed approach presented four different rules for scheduling: Two modes are considered and the system is initially in the "low" mode (L-mode). While the system is in the low-criticality mode, a higher-priority ready job can only preempt the currently executing job if its priority is higher by a preemption threshold value. If a job tries to execute for more than its WCET estimate, a mode switch occurs. Once in the "high" mode (H-mode), a higher-priority job of a higher-criticality task can only preempt the currently executing job if its priority is higher by a preemption threshold. This work also presented a response-time-analysis-based schedulability test which was based on the concept of level-i busy period. The evaluation of the proposed Preemption Threshold Adaptive Mixed-Criticality (PT-AMC) demonstrates significant improvement over preemptive and non-preemptive approaches.

Later, the authors extended the above work in [65] by presenting two different variants, Preemption Threshold max (PT-max) and Preemption Threshold rtb (PT-rtb). The objective of these extended approaches was to reduce the stack size and enable the implementation of MCS over resource-constrained embedded platforms. For priority and preemption threshold assignment, they used Branch-and-Bound (BNB) and Priority Assignment algorithm with Deadline Monotonic and Maximum Preemption Threshold (PADMMPT), which invokes the proposed schedulability analyses PT-rtb or PT-max. Evaluation of the proposed approaches demonstrates that PT-max brings minor improvement in terms of schedulability ratio when compared with PT-rtb.

Another attempt to combine limited preemption and adaptive mixed-criticality scheduling was done by Burns and Davis [66]. They presented Adaptive Mixed-Criticality Non-Preemptive Regions (AMC-NPR), which allows tasks to have non-preemptive regions at the end of their WCET estimates for each mode. A method for determining the length of the non-preemptive regions and a priority assignment technique were also presented. Since standard automotive and avionics systems can have up to five criticality levels, the authors also discussed how AMC-NPR can be extended to multiple criticality levels, based on the work of Fleming et al. [67]. It was also demonstrated, via empirical evaluation, that AMC-NPR outperformed the previously presented approaches SMC, SMC-NO, CrMPO and standard AMC.

Zhao et al. [68] argued that existing response-time-bound-based analysis techniques for adaptive mixed-criticality scheduling are too complex for optimization purposes. Therefore, they presented response bound function (rbf) based analysis for adaptive mixed-criticality systems (AMC-rbf). The proposed analysis was proved to be safe with bounded pessimism over AMC-rtb. The effectiveness of the proposed AMC-rbf was demonstrated with the help of two design optimization problems. The first problem deals with software synthesis of multi-rate Simulink models with the objective of improving control quality by minimizing the functional delays. An experimental evaluation with synthetic task sets demonstrated that for relatively simpler problems, the AMC-rbf-based optimization algorithm provides solutions within 4% of the best solution. However, for complex problems, the running time is up to 2x faster than AMC-rtb or AMC-max based approaches. In the second set of experiments, AMC-rbf was compared to approaches that used AMC-max and AMC-rtb in combination with branch-and-bound and ILP-based task assignment algorithms, in partitioned multicores, and demonstrated good scalability for complex problems, with just 4% reduction in the fraction of schedulable task sets.

All the schedulability analysis techniques discussed for fixed-priority mixed-criticality systems are sufficient, therefore pessimistic. To avoid this pessimism, Asyaban et al. [69] presented an exact schedulability analysis for AMC. The exact schedulability test requires enumerating and evaluating all possible system behaviors, which due to state space explosion is potentially intractable. Therefore, a pruning method was also suggested which reduces the state space size. This work also proved that Audsley's priority assignment algorithm is not optimal for any exact schedulablity test for AMC systems. Inspired by Audsley's algorithm, a heuristic-based priority assignment method was also suggested, which assigns lower priorities to low-criticality tasks as far as possible.

A wide range of graceful degradation techniques is presented in the literature. Most of the analyses allow low-criticality jobs that were started before the mode change to execute up to their completion [6]. Some resilient mixed-criticality models allow the execution of low-criticality tasks with reduced priority [5, 70]. Others increase the periods and deadlines of low-criticality tasks [71, 72].

An alternative mixed-criticality model which was presented by Baruah et al. [73] allows tasks to change periods rather than WCET estimates. The proposed scheduling algorithm divides the test into three different phases. In the first pre-processing phase, a schedulability test is applied and, upon success, an additional parameter δ is determined for each task and later used for defining virtual deadlines. During the second phase, jobs are dispatched assuming the system behaves according to low-criticality specifications. However, if these assumptions are violated, successive jobs of some tasks arrive less than $T_i(LO)$ time units apart. During the third phase, if low-criticality specifications are violated, all the low-criticality tasks are abandoned and the rules used for job dispatching are also changed.

In [74], Huang et al. implemented some existing mixed-criticality scheduling approaches (incl. period transformation, fixed task-priority static mixed-criticality and zero-slack scheduling) in Linux and evaluated their overheads and scheduling performance. In follow-up work [75], they classified existing schemes in terms of their runtime enforcement mechanisms (which determine how long a job is allowed to execute and when it should be aborted) and their priority assignment methods (that determine the precedence of execution among the ready-to-execute jobs). Runtime enforcement mechanisms were classified into zero-slack scheduling, period transformation, static and adaptive mixed-criticality, whereas priority assignment methods were classified into fixed task priority, fixed job priority, and dynamic priority schemes. A slight improvement to AMC-max, named AMC-IA, was also presented. The proposed approach demonstrates nominal improvement over AMC-max in terms of schedulability ratio. However, it was claimed that AMC-IA is relatively easy to extend for systems with more than two criticality levels. Chen et al. [76] studied the fixed priority scheduling of mixed-criticality systems but with a more generalized priority assignment method, which considers different priorities for different execution phases of a MCS. Based on Audsley's priority assignment, a Heuristic Priority Assignment (HPA) method was suggested which assigns three different priorities: P_i^L(priority for jobs executing in the steady L-mode), P_i^T (priority for jobs executing in the transition phase) and P_i^H (priority for jobs released in the H-mode). The HPA algorithm starts by assigning P_i^L priorities using Audsley's priority assignment method. If this process fails, then HPA tries to find different priorities for different modes, again starting with P_i^L. Once a complete ordering of priorities is determined for L-mode, HPA continues with P_i^H and, upon finding a feasible solution, it finally determines P_i^T. A sufficient schedulability test was also presented for this priority assignment method based on the earlier work from [6], which studies the system for three different modes. The evaluation of the proposed methodology demonstrated that the flexible priority assignment method improves the system schedulability performance.

For the COTS platforms where the processing speed may vary, Baruah et al. [77] presented a mixed-criticality model and respective schedulability analysis under two processing speeds: normal and degraded. Under normal circumstances, it was assumed that all the tasks would be executed, while, in the degraded mode, only high-criticality tasks should be executed, with the extended WCET estimates. Even though the existing theory for MCS with multiple WCET estimates can easily be used by transforming the MCS hosted by varying-speed processors to the earlier, multiple WCET, model, it was demonstrated that the computational complexity can be greatly reduced using the scheduling strategies presented in [77]. It was also claimed that the proposed model is also applicable for time-sensitive data transmission over a faulty communication network. Later work [78] extended the scheduling models for systems which have uncertainty in both execution time and processor speed.

Most of the research works on mixed-criticality scheduling protect the execution of high-criticality tasks in the case of overrun by either aborting all the low-criticality tasks or by degrading their services. Huang et al. [79], on the other hand, used the dynamic processor speedup feature of modern processors to protect the execution of critical tasks. As in other works, multiple WCET estimates exist for the same task and there are different system modes. In the normal mode, whenever a high-criticality task exceeds its WCET estimate assumed for that mode, a mode switch occurs. In the new mode, the processor is sped up. This higher speed is calculated offline to suffice for all high-criticality tasks (including those caught up in the mode switch) to meet their deadlines even if they execute for as long as their stricter WCET estimates. The higher processor speed also helps provide better service to lower-criticality tasks, by not abandoning all of them immediately. On the

first opportunity (i.e., when the processor is next idle), the system returns to the normal mode, with all tasks present, and less pessimistic WCETs assumed, and with the processor reverting to the original speed. Since overrun situations happen rarely, it was demonstrated that the proposed methodology has minimal overhead and helps the system recover and switch to normal operations faster.

The objective of [79] was to use Dynamic Voltage and Frequency Scaling (DVFS) to preserve the real time guarantees of MCS. In another work, Huang et al. [80] used DVFS to ensure timeliness guarantees of high-criticality tasks and reduce the energy consumption. For this purpose, the authors formulated a convex program by integrating DVFS with EDF-VD (a scheduling algorithm for mixed-criticality systems discussed later in this chapter). To improve the overall system schedulability, a method was also presented for determining the range of the deadline-shortening factor x. An optimal algorithm and pruning method to reduce the problem space size were also suggested.

All the MC models discussed so far assume that a transition from low- to high-criticality mode happens at the time instant when a task has already executed for its low-criticality mode WCET estimate without signaling completion. An alternative model suggested by Agrawal et al. [81], assumes that, upon arrival, a job reveals which of its WCET estimates it will respect. The proposed semi-clairvoyant model provides a middle ground between clairvoyant and mixed-criticality scheduling. In contrast to clairvoyant scheduling, which requires precise values of a task parameter on arrival, semi-clairvoyant scheduling only requires to know if a job will execute with its low-criticality or high-criticality WCET estimate, which is a far less onerous requirement. A linear programming based optimal schedulability test was also presented and its correctness was proved for dual-criticality systems. It was also demonstrated that significant improvement can be achieved in terms of time and computation complexity when compared with the traditional mixed-criticality scheduling. Later, Burns and Davis [82] presented uniprocessor schedulability analysis for adaptive mixed-criticality scheduling with arbitrary-deadlines and a semi-clairvoyant task set.

Our short survey of the literature on mixed-criticality scheduling focuses primarily on approaches based on fixed priorities, as these were most relevant to the work described later in this thesis. However, mixed-criticality scheduling approaches based on an EDF policy also exist [83, 84, 85, 86]. Notably, Baruah et al. proposed EDF-VD [83] which stands for "Earliest Deadline First - with Virtual Deadlines". This approach is analogous to AMC, but differs in the baseline scheduling policy. Additionally, shorter deadlines are reported to the EDF scheduler for high-criticality tasks in the "low-criticality" mode. After a mode change, the actual deadlines are used for such tasks instead. EDF-VD scaled the deadlines of all high-criticality tasks by the same factor. Ekberg and Yi [84, 85] improved on EDF-VD by allowing for a different deadline scaling factor for each high-criticality task, calculated according to a heuristic. This brought better scheduling performance. Later, Easwaran came up with a scheduling test [87] that outperformed that of Ekberg and Yi.

2.4 Controller area network (CAN) scheduling

CAN is a serial bus communication protocol that allows event-triggered static priority arbitration among messages with non-preemptive message transmission. Since the CAN bus is shared among all the nodes, each node has to compete in the *contention round*, to get exclusive access to the bus for transmitting the message payload.

To ensure the safe transmission of messages over CAN, Tindell et al. [88] presented a schedulability analysis based on previously presented fixed priority uniprocessor scheduling theory. This schedulability analysis by Tindell et al. provided a method to compute the busy period and worst-case response time and hence first possible way to analyze CAN for timing correctness and ensuring the deadline guarantees by all the messages over CAN. [88, 89, 90] also highlighted the importance of appropriate priority assignment for obtaining higher schedulability performance. A "deadline minus jitter" monotonic priority order was also suggested as the optimal priority assignment. Tindell's seminal work not only led to many research works in CAN schedulability theory [91, 92, 93, 94] but also heavily influenced industrial on-chip CAN peripherals [95]. This work by Tindell et al. [88] implicitly assumes that, for constrained-deadline messages, the priority-level-m busy period only contains one arrival by the message under analysis. This is in contrast to the work by Harbour et al. [96], who demonstrated that, even for constrained-deadline tasks, multiple invocations of the task under analysis need to be inspected if the priority of the tasks varies during execution. Later Davis et al. [20] demonstrated that non preemptive scheduling (as happen for the CAN messages) is a special case of preemptive scheduling with varying execution priority. It was also shown with sample example that Tindell et al. [88] Analysis does not incorporate additional delays caused by priority inversion and might declare some of the messages schedulable, which are actually not. Davis et al. [20] resolved this problem by presenting a revised analysis, which draws on the work of Tindell et al. [88] for fixed priority non-preemptive scheduling and George et al. [97] for fixed-priority preemptive scheduling with arbitrary-deadline messages. Furthermore, Davis also demonstrated that "Deadline minus jitter" priority ordering which was declared optimal for CAN by Tindell et al. [88], is in fact not optimal, in light of the revised schedulability analysis.

To increase the overall CAN bus utilization, an offset scheduling-based mechanism was presented by Grenier et al. [98, 99] and Yomsi et al. [22], which effectively reduces the response time of messages by offsetting the release of different messages to avoid the message bursts that lead to undesirably large WCRT. The proposed approach has a limited use case as it is difficult to find a suitable offset in systems with higher utilization. For this very reason, Chen et al. [100] presented an improved version of the offset finding method. All schedulability analysis techniques discussed so far either assume that messages are queued periodically or sporadically. Mubeen et al. [101] extended the existing analysis to a mixed-message network. The proposed schedulability analysis assume that the messages, which are queued for transmission are activated periodically or sporadically. Mubeen et al. [102] employed offset-based techniques to reduce the deadline violations for mixed message networks.

Davis et al. [24, 21] presented schedulability analysis techniques for CAN, which account for mixed-queue networks (some nodes have priority queue scheduling policy while others have FIFO queues). For instance, Davis et al. [24] extended their previously presented work [20] for a mixed-queue network in [24] where some nodes have FIFO queues while others employ priority-based queues. The detrimental effect of FIFO queues on schedulability of messages was also demonstrated with evaluation. This work was also extended for a more generalized configuration with an arbitrary-deadline message model and reordering possibility [103].

Previously presented work by Davis et al. [20] can be used for such distributed systems with gateway interconnect. However, the provided response time bounds are quite pessimistic. To reduce this pessimism, Davis et al. [103, 21] presented a revised schedulability analysis for gateway-integrated CAN messages, which considers the classical concept of level-i busy period. The authors also advised using multiple FIFOs, each allocated with a small number of messages, for systems where priority queue implementation is not possible.

Xie et al. [104] proposed a more holistic approach that analyzes different possible arrival orders of messages using the concept of busy sequence and minimum distance constraint for the computation of WCRT.

Nahas et al. [105] presented two different approaches to reduce the messages length for resource constrained embedded systems. The proposed approaches use software based techniques to reduce the impact of bit stuffing on actual message length. The implementation cost of the proposed schemes in terms of processing and memory requirement was also evaluated and optimized. Polzlbauer et al. [106] presented an approach, which reduces the processing load at each ECU by filtering undesired messages. Filtration is performed using the message ID. The authors demonstrated that the problem is NP-complete and therefore used a filtration techniques based on simulated annealing. Azketa et al. [107] presented optimized schedulability analysis for segmented messages for a distributed real-time CAN system.

Chapter 3

System model and experimental evaluation framework

The schedulability analyses presented or discussed in this thesis make certain assumptions about the underlying system. Their validity depends on those assumptions. This chapter discusses the different system models considered in this manuscript along with the experimental setup, which is used to evaluate and compare the performance of the proposed methodologies. It includes detailed descriptions of (i) the multiframe mixed-criticality task model, (ii) the multicore hardware platform in consideration, including its shared resources and (iii) the multisized CAN message model. If required, later chapters will present modifications or extensions to the above-mentioned system models and the evaluation setup.

A brief outline of the chapter is as follows: Section 3.1 presents the different variants of the multiframe mixed-criticality system model. Later, Section 3.2 extends the system model of Section 3.1 to multicore platforms. Sections 3.3 discuss system models for controller area networks with priority-based queues hosting multisized messages. Section 3.3.3 extends the CAN system model for mixed-queue network. Section 3.4 briefly describes the evaluation setup used for the proposed schedulability analyses.

3.1 System model for multiframe mixed-criticality systems

In this section, we formalize the mixed-criticality multiframe task model, in both of its variants, static and adaptive mode-based.

Consider a set τ of n mixed-criticality sporadic tasks $(\tau_1, \ldots, \tau_n)$ on a uniprocessor system. Each task τ_i has a relative deadline D_i, a minimum inter-arrival time T_i, a criticality level κ_i, which is either high (H) or low (L). For the sake of conciseness, we denote a low-criticality and a high-criticality task as L-task and H-task, respectively. Unlike jobs of conventional (i.e., single frame) tasks, successive jobs of the same multiframe task can differ in their WCETs, in a pattern that repeats after F_i successive jobs, where F_i is smallest integer such that the WCET of k^{th}, $(k+F_i)^{th}$, $(k+2F_i)^{th}$,... jobs of τ_i have same WCET estimate. Accordingly, any F_i successive jobs of τ_i are called a *superframe* and we often refer to a job as a frame, especially when we want to distinguish between jobs in a superframe. In any schedule, the k^{th}, $(k+F_i)^{th}$, $(k+2F_i)^{th}$, ... jobs of τ_i all have the same

worst-case execution time behavior. However, even the same frame of the same task has multiple estimates for its WCET, in our model, with different degree of confidence. Each job has a WCET estimate per criticality level less than or equal to its task's criticality level. Thus, job j of L-task τ_i has a single WCET estimate, $C^L_{i,j}$, i.e., a low-criticality WCET, or L-WCET for short. On the other hand, job k of H-task τ_i, has two WCET estimates, one L-WCET, $C^L_{i,k}$, and one high-criticality WCET, or H-WCET for short, $C^H_{i,k}$. L-WCETs can be optimistic. Thus, $C^L_{i,j} \leq C^H_{i,j}$. We do not assume that $C^L_{i,j} > C^L_{i,k}$ implies $C^H_{i,j} < C^H_{i,k}$, for any j or k.

The semantics vary slightly, according to the particular variant of the model. The choice of scheduling algorithm is orthogonal to the above task model. In this thesis, we assume fixed-priority scheduling, i.e., each task has a fixed priority.

Table 3.1 summarizes most symbols used in the worst-case response time analysis for multiframe mixed-criticality systems over single-core and multicore platforms.

3.1.1 Static variant

In this variant, whenever a job of some L-task τ_i, corresponding to its j^{th} frame, executes for its entire L-WCET, $C^L_{i,j}$, without completing, then that particular job is terminated. This does not suppress the arrival of future jobs of τ_i, subject to interarrival time constraints, and respecting the actual frame sequence of the task. Under this model variant, the system is schedulable if (i) no L-task misses its deadline as long as all jobs of all tasks, irrespective of their criticality, execute for up to their $C^L_{i,j}$, and also (ii) no H-task misses its deadline, as long as every job of every task τ_i executes for up to its $C^{\kappa_i}_{i,j}$, where κ_i is the criticality of τ_i, i.e., as long as L-tasks execute up to $C^L_{i,j}$ time units and H-tasks execute up to $C^H_{i,j}$ time units.

3.1.2 Adaptive mode-based variant

As in the mode-based variant of Vestal's model introduced by Baruah and Burns [6], the system operates in different modes – two in this paper, which is also the simplest case. The system boots in the default L-mode, where all tasks (i.e., both of low and high criticality) are present. For the L-mode, WCET estimates are assumed, for all task frames, that are most likely but not necessarily safe. If at any point in time, any job attempts to exceed its assumed WCET estimate for the L-mode, a mode switch is triggered. This means that all low-criticality tasks (L-tasks) are discarded and only high-criticality tasks (H-tasks) are allowed to execute. The system is mixed-criticality-schedulable if (i) all tasks meet their deadlines in the L-mode, assuming that their jobs execute in accordance to the WCETs assumed for that mode; and (ii) all H-tasks (including any jobs thereof caught up in the mode switch) meet their deadlines in the H-mode, assuming that their jobs can execute for as long as their respective H-mode WCET estimates. The latter are provably safe and typically very pessimistic.

3.2 System model for multiframe mixed-criticality systems on MPSoCs with memory bandwidth regulation

The multiframe mixed-criticality model discussed in Section 3.1 is only applicable for single-processor systems. However, for systems where more than one core exists on a single die along with shared resources, additional

Table 3.1: Symbols used in the analysis of multiframe mixed-criticality systems on MPSoC's with memory bandwidth regulation

Symbols	Description
F_i	Number of frames of a task τ_i
$C^x_{i,j}$	WCET of j^{th} frame of τ_i in mode $x \in \{L,H\}$
$g^x(\tau_i, j)$	Cumulative worst-case execution requirement of j successive jobs of τ_i in mode $x \in \{L,H\}$, where $1 \leq j \leq F_i$
$G^x(\tau_i, t)$	Cumulative worst-case processor request of τ_i in mode x in any time interval of t time units
R^L_i	WCRT of τ_i in L-mode
R^H_i	WCRT of τ_i in H-mode
R^s_i	WCRT of τ_i, if caught in a mode switch at time s
$r^L_i(q), r^H_i(q)$	Completion time of the q^{th} job of τ_i in a level-i busy period in L-mode and H-mode, respectively.
D_i, T_i	Deadline and minimum interarrival time of a task τ_i
$P_1, P_2, P_3, ...P_K$	K identical cores.
$C^{e\|k}_{i,j}, C^{m\|k}_{i,j}$	Worst-case computation and memory access time for kth frame.

aspects of the platform need to be modelled. The following subsections extend the previously discussed multiframe mixed-criticality model to systems implemented on multicore processors, that employ memory bandwidth regulation mechanisms for bounding the contention at the level of shared memory.

3.2.1 Hardware platform

As our work relies on [55] and extends its results, it is bound by the assumptions therein. Specifically, K identical cores $\{P_1, P_2, ...P_K\}$ share a memory controller through which main memory is accessed. Any speculative units and prefetchers are disabled. The scheduling policy for both memory controller and interconnect is round robin. For the regulation of memory accesses, a mechanism like MemGuard[1] [41], that requires no information about tasks running on other cores, is used. This allows to compute the response time of a task independently of the workload on other cores. This relaxes the certification effort. The outer cache is partitioned or private to each core or can even be partitioned among tasks to avoid cache-related preemption delay issues. Performance measuring counters count the memory accesses. As in [41, 55], the memory access time is considered constant. Although not a requirement, the main memory can be partitioned (e.g. via PALLOC [38]) for improved predictability.

The memory access budget Q_k of a core k is the maximum number of accesses it can make in a regulation period P. If a core depletes its budget, it is stalled for the rest of the current regulation period. At the start of each regulation period, a core's budget is replenished. We assume that the regulation periods are equal and synchronized for all cores. The memory bandwidth share of a core is $b_k = \frac{Q_k}{P}$, and $\sum_{k=0}^{K} b_k \leq 1$, i.e., the total allocated bandwidth does not exceed the total available bandwidth. As in [55], the length P of the regulation period is expressed in multiples of the constant memory access latency.

[1] Specifically, all that is needed is (i) hardware timers (ii) performance monitoring counters to track last-level-cache misses and (iii) the generation of an exception upon a counter overflowing. All these features are commonly found in COTS platforms [41].

3.2.2 Task model

Like the task model presented previously, in Section 3.1, each task τ_i is characterized by the 5-tuple $\tau_i = (\vec{C}_i^L, \vec{C}_i^H, D_i, T_i, \kappa_i)$. Where $\vec{C}_i^L$ and $\vec{C}_i^H$ represent vectors of WCET estimates of τ_i in L and H-mode, respectively. However, each frame's κ-WCET, $C_{i,j}^k$, is given by $C_{i,j}^\kappa = C_{i,j}^{e|\kappa} + C_{i,j}^{m|\kappa}$, where $C_{i,j}^{m|\kappa}$ is the worst-case memory access time estimate and $C_{i,j}^{e|\kappa}$ is the worst-case CPU computation time estimate. Accordingly, the κ-WCET of job f is denoted by the pair $\left(C_{i,(f \bmod F_i)}^{e|\kappa}, C_{i,(f \bmod F_i)}^{m|\kappa}\right)$. Memory accesses and CPU computation do not overlap, to comply with [55]. Tasks are partitioned to the cores and scheduled preemptively with fixed priorities, assigned offline by any algorithm.

3.2.3 Mixed-criticality models

As in Section 3.1.2, we consider an adaptive mixed-criticality (AMC) approach and assume that a WCET estimate for a given criticality level is never less conservative than the WCET for a lower criticality level, thus providing a higher level of confidence that it will not be overrun. For example, in a system with only two criticality levels, L and H, the L-WCET is smaller than or equal to the H-WCET. Generalizing to our model, for every frame k of every H-task i, $C_{i,k}^{e|L} \leq C_{i,k}^{e|H}$ and $C_{i,k}^{m|L} \leq C_{i,k}^{m|H}$.

3.3 System model for multisized messages over a Controller Area Network

In this section, we describe the system model that is used to analyze the worst-case response time for CAN with multisized messages. This system model is based on Davis et al. [20] work.

3.3.1 Controller Area Network

The system consists of K nodes which are connected through a single shared CAN bus. For transmission, only the node with the highest priority messages is allowed to transmits its message payload over the CAN bus non-preemptively. However, if more than one node start transmitting message I'D during the arbitration round, each node can determine by itself if it is transmitting the highest priority by monitoring each bit on the bus. The node with the highest priority message keeps transmitting while all the other nodes change to receiver mode and wait for the next arbitration round for the transmission of their messages. This scheme is properly called Carrier Sense Multiple Access/Collision Resolution (CSMA/CR) [20]. CAN messages do not use explicit addresses to specify the receiving node; rather, this information is encoded in the identifier, which also represents the priority of each message. The length of an identifier can either be 11-bit (standard format) or 29-bit (extended format) long. Additionally, bit stuffing is used by the synchronization protocol to resynchronize all the nodes of the network[2].

[2]These protocols use transition edges for synchronization. Therefore, transmission of long sequences of codes without bit transition is avoided in order to avoid drift in the bit clock. A complemented bit in the stream is used after every 5 bits of the same transmission type are transmitted. More information about these protocols can be found in the Bosch report on CAN [108].

Table 3.3: Terminology for multiframe real-time tasks (left) and analogous concepts for multisized CAN messages (right).

multiframe task	multisized CAN message
job	message instance
superframe	message cycle
frame	message subtype
number of frames	message cycle length

Table 3.2: Symbols used in the analysis of multisized messages hosted over CAN

Symbols	**Description**
S_m	Number of subtypes of a message m_i
$C_{i,j}$	WCTT of j^{th} instance of m_i
J_i	Release jitter of message m_i
$g(m_i, j)$	Cumulative worst-case transmission time requirement of j successive jobs of m_i
$G^x(\tau_i, t)$	Cumulative worst-case transmission time request of m_i for any t time units
R_i^L	WCRT of m_i
v_i	Longest level-i busy period of m_i.
$w_m^{n+1}(q)$	Longest time from start of level L_m busy period until instance q of message m start transmission.
D_i, T_i	Deadline and minimum interarrival time of a message m_i
$N_1, N_2, N_3, ...N_K$	K nodes connected via a shared CAN bus.
τ_{bit}	Transmission time for one bit
$s_{i,j}$	Payload for jth subtype of message m_i in bytes

3.3.2 Message model

The assumed system model, other than capturing the possibility of a message's length varying according to a known cyclic pattern, does not deviate at all from the one assumed in [20].

Definition 1. ***(Multisized messages)*** *A multisized message is represented with a vector of WCTT estimates $\vec{C_m}$, which varies according to a known periodically repeating fashion.*

Definition 2. ***(Message cycle length)*** *The **message cycle length** S_m, characterizing a message m (and denoted as simply S, for ease of notation, whenever possible), is the least integer such that the worst-case transmission lengths of the k^{th} instance and the $(k+S)^{th}$ instance of message m are the same, for any k.*

Definition 3. ***(Message cycle)*** *A **message cycle** consists of any consecutive S instances of message m.*

Definition 4. ***(Message subtype)*** *Additionally, we call each of the S messages in a cycle, a **message subtype**.*

More formally, this is the case where for some integer $S(>1)$, the worst-case length of message subtype n and $n+S$ with the same identifier is the same. We assume a set of n sporadic messages $M = \{m_1, m_2, m_3, ..., m_n\}$, which are statically assigned to the CAN network with a K set of nodes/ECUs/processors. Each of these messages is represented by a 4-tuple: $m_i = \{\vec{C_i}, J_i, T_i, D_i\}$. Where $\vec{C_i}$ represents a vector $\{C_{i,1}, C_{i,2}, C_{i,3}, ..., C_{i,S}\}$ of worst-case transmission time (WCTT) estimates. This difference in WCTT estimates of successive instances of a message is because of different payload lengths. However, for multisized messages, we assume they vary between 0 and 8 bytes in a periodically repeating fashion after every S_m instances. For each of the subtypes,

WCTT can easily be computed by using the respective payload in one of the following equations:

$$C_{i,j} = (55 + 10 \cdot s_{i,j}) \cdot \tau_{bit} \qquad \text{if ids are 11-bit} \tag{3.1}$$

$$C_{i,j} = (80 + 10 \cdot s_{i,j}) \cdot \tau_{bit} \qquad \text{if ids are 29-bit} \tag{3.2}$$

The 2*nd* component, J_i in the message representation of m_i is referred to as the arrival jitter. The event that triggers the queuing of an instance of m_i is assumed to occur with a minimum inter-arrival time of T_i and is referred to as the message period. Each message has a hard deadline D_m representing the maximum allowed duration from the time an instance is queued for transmission until it is successfully received at the receiving node. These deadlines of m_i can either be constrained ($D_i \leq T_i$) or arbitrary ($D_i > T_i$). The worst-case response time for m_i is denoted by R_i and represents the maximum time from the release of a message instance until its reception by interested node. A message m_i is declared schedulable if $R_i < D_i$. The overall system is declared schedulable if all the messages in M are schedulable. A summary of notations used in this system model is given in Table 3.2.

3.3.3 CAN node models

Most work on CAN analysis assumes that nodes use a priority-based queuing policy. I.e., that when a node detects that the bus is idle it transmits the highest priority ready message. However, as we have already mentioned in Chapter 2, some works have considered other queuing policies. In this work we consider the following:

1. Priority Queue (PQ) All the nodes with priority queue scheduling policy ensures that at any time when arbitration starts, only the highest priority message queued at the node enters arbitration.

2. No-reordering permitted (WQN) i.e. instances of the same message are transmitted in the order in which they are queued. Note that this is weaker than FIFO, because an instance of a message that is queued after an instance of another message may be transmitted before the latter. Nevertheless, FIFO is an example of WQN.

3. Reordering permitted (WQR) which imposes no restriction on the order in which message instances are transmitted. In particular, a message instance that is queued after another instance of the same message may be transmitted before the latter.

3.4 Experimental evaluation framework

All the analysis techniques presented in this thesis are implemented for the purpose of performance evaluation using a Java-based framework [109]. The aforementioned Java software has two modules. One module generates parameterizable synthetic workloads. The other module performs functions such as, e.g., task-to-core assignment, priority assignment, bandwidth allocation, if applicable, and the actual schedulability testing, according to the particular analysis.

The experiments ran on a system with 8 Xeon E5-2420 2.20 GHz dual cores, 32 GB of RAM, and Linux Mint 18.3 Sylvia with openJDK 1.8.0_242.

Unbiased task utilizations in L-mode are generated using the UUnifast-discard [110, 111] algorithm and the total utilization is varied within $[0.1,\ max]$, where max represents the maximum upper limit. Task periods (10 msec to 1 sec) are log-uniform-distributed. The framework considered can be used to generate/analyze different deadline models (i.e., implicit (T_i=D_i), constrained ($D_i \leq T_i$) or arbitrary deadlines ($D_i > T_i$, possibly)). The task set size ($|\tau|$) is an input parameter. The number of H-tasks is $\lfloor \xi \times |\tau| \rceil$, where $\xi \in (0,1)$ is a user-defined fraction of H-tasks and $\lfloor \cdot \rceil$ denotes rounding to the nearest integer. The number of frames for each task is randomly selected over $[1,\ \alpha]$, where the upper-bound α is an input parameter. The L-WCET of the first frame is generated by multiplying the period of a task with its L-mode utilization, i.e., $C^L_{i,1} = T_i * U^L_i$. The L-WCETs of other frames are randomly generated within $[\beta \times C^L_{i,1},\ C^L_{i,1}]$, where $\beta \in (0,1]$ is an input parameter[3]. For any frame, its H-WCET is computed by linearly scaling its L-WCET (i.e., $C^H_{i,j} = \kappa \times C^L_{i,j}$), where κ is an input parameter.

More details about any modifications to the presented evaluation setup (i.e., task/message model, priority assignment, or analysis module) are discussed in the respective chapters.

[3]For convenience, without loss of generality (since shift-rotating the order of the frames results in an equivalent multiframe task), the first frame has the biggest L-WCET. Note that our analyses make no assumptions regarding the WCET of the first frame being the greatest.

Chapter 4

Multi-frame mixed criticality scheduling

In this chapter, we present our contributions pertaining to the schedulability analysis for the new multiframe mixed-criticality model, which allows tasks to have multiple, periodically repeating, WCETs in the same mode of operation.

4.1 Introduction

Having introduced the multiframe mixed-criticality model and the motivations behind it in previous chapters, in this chapter we present our analysis techniques for this new model. Those extend the analysis techniques for Static Mixed-Criticality scheduling (SMC) and Adaptive Mixed-Criticality scheduling (AMC) and the schedulability analysis for multiframe task systems, initially considering a constrained-deadline model, and then extended to the more general, but also more complex, arbitrary-deadline scenario. An optimal priority assignment for our schedulability analysis is also identified. Specifically, this chapter is organized as follows:

Section 4.2 discusses some important known results on the mixed-criticality scheduling of constrained-deadline tasks. The schedulability analysis techniques for the static and mode-based variants of the multiframe Vestal model for mixed-criticality systems are then presented in Section 4.3, assuming a constrained deadline model. Analogously, Section 4.4 discusses relevant results for the mixed-criticality scheduling of arbitrary-deadline tasks and Section 4.5 presents the generalization of our analysis techniques from Section 4.3 for arbitrary deadlines. Section 4.6 provides an example of the analyses applied to a task-set. In Section 4.7, we identify an optimal priority assignment scheme for multiframe mixed-criticality systems, using our analysis as the schedulability test. Section 4.8 provides an experimental evaluation, using synthetic task sets, of the schedulabiling performance of the proposed analyses, compared to those devised for the original (non-multiframe) Vestal model. Finally, Section 4.9 concludes this chapter.

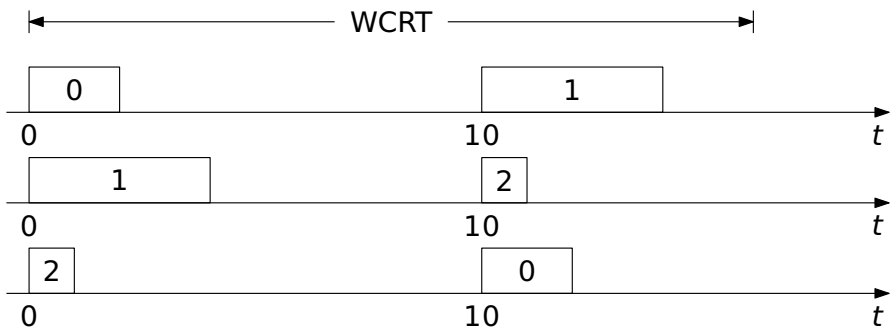


Figure 4.1: Interference of a multiframe task depends on the phasing of its jobs w.r.t the job under analysis. (The numbers inside the rectangles are the frame numbers of the interfering task.) In this example, we assume that the WCRT of the frame under analysis is between 14 and 20, and that the interfering task has a period of 10 and its WCET vector is $\vec{C} = (2,4,1)$.

4.2 Background on schedulability analyses of constrained-deadline tasks

Our analysis builds on the worst-case response time analysis for the multiframe task model by Baruah et al. [11] and on the different mixed-criticality schedulability analysis techniques by Baruah, Burns and Davis [6]. In this section, we summarize those existing results.

To determine the schedulability of a multiframe task, it suffices to check if the WCRT of the frame with the largest WCET is smaller than the task deadline (which, for now can be assumed to be constrained, i.e., $D_i \leq T_i$)[1]. Unlike what holds for the "single-frame" model, it is not enough to compute the number of job releases of each interfering task in the WCRT of the task under analysis. This is because each multiframe task τ_i is characterized by a vector of WCET $\left(C_{i,0}, C_{i,1}, \ldots, C_{i,(F_i-1)}\right)$, and the phasing of the released jobs affects the amount of interference. This is illustrated in Fig. 4.1, which shows the 3 possible phasings of jobs of a task τ_i with period $T_i = 10$ and WCET vector $(2,4,1)$, with respect to a frame under analysis whose WCRT is between 14 and 20 time units. During this interval there are at most 2 jobs of the interfering task, but the amount of interference depends on the first job in that sequence. As illustrated, this interference is worst, 6, if the first job in the sequence is job 0, F_i, ...On the other hand, if the first job in the sequence is job 2, $2+F_i$, ..., the interference is the least, 3.

To efficiently compute the interference exerted by a multiframe task, Baruah et al. [11] define the g function:

$$g(\tau_i,k) = \begin{cases} 0 & \text{if } k = 0 \\ max\left\{ \sum_{\ell=j}^{j+k-1} C_{i,(\ell \bmod F_i)} : 0 \leq j < F_i \right\} & \text{if } 1 \leq k \leq F_i \\ q \cdot g(\tau_i, F_i) + g(\tau_i, r) & \text{otherwise} \\ \quad \text{where } q = k \text{ div } F_i \text{ and } r = k \bmod F_i & \end{cases} \tag{4.1}$$

[1] The more general case of arbitrary deadlines is considered later in Section 4.5.

which bounds the cumulative WCET of any sequence of k jobs of task τ_i. This function is then used to define the G function:

$$G(\tau_i, t) = g\left(\tau_i, \left\lceil \frac{t}{T_i} \right\rceil\right) \tag{4.2}$$

which bounds the cumulative WCET of task τ_i over any time interval of duration t. Finally, the worst-case response time recurrence is:

$$R_i = g(\tau_i, 1) + \sum_{\tau_j \in hp(i)} G(\tau_j, R_i) \tag{4.3}$$

where $hp(i)$ is the set of tasks with priority higher than τ_i.

4.2.1 Static mixed-criticality (SMC)

In the SMC task model [5], each task has a criticality level κ_i it may have multiple WCET estimates one for each criticality level $C_i(\kappa_i)$. The pessimism of those estimates increases with the criticality level. When the job of a task exceeds the WCET estimate corresponding to its own level, it is terminated. With just two levels, this ensures that the interference of any job of a low-criticality task does not exceed C_i^L.

Thus, assuming only two criticality levels, the WCRT recurrence for a single-frame low-criticality task under SMC is

$$R_i = C_i^L + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j^L \tag{4.4}$$

and for a high-criticality task it is

$$R_i = C_i^H + \sum_{\tau_j \in hpL(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j^L + \sum_{\tau_k \in hpH(i)} \left\lceil \frac{R_i}{T_k} \right\rceil C_k^H \tag{4.5}$$

where $hpL(i)$ and $hpH(i)$ are the sets of higher-priority low- and high-criticality tasks, respectively, for τ_i.

4.2.2 Adaptive mixed-criticality (AMC)

Baruah, Burns and Davis [6] proposed a fixed-priority uniprocessor adaptive mixed-criticality scheduling algorithm (AMC). In the proposed model, the system operates in two different modes. During the default L-mode, tasks execute with their L-WCET estimates. If a task tries to execute for more than its L-WCET estimate, a mode switch occurs. Thereafter, only high-criticality tasks are allowed to execute, for up to their corresponding H-WCET estimates, while all the low-criticality tasks are discarded. In [6], two sufficient schedulability analyses (AMC-rtb and AMC-max) are devised. Both analyses verify the schedulability of all tasks in L-mode, and the schedulability of H-tasks in H-mode and when they are affected by a mode switch, under the AMC model. In this section, we briefly review the latter, because the analyses in L-mode and steady-state H-mode are essentially standard fixed-priority response time analysis.

4.2.2.1 AMC-rtb

In this analysis, the WCRT recurrence of a job of task τ_i that is affected by a mode switch is:

$$R_i^* = C_i^H + \sum_{\tau_j \in hpH(i)} \left\lceil \frac{R_i^*}{T_j} \right\rceil C_j^H + \sum_{\tau_k \in hpL(i)} \left\lceil \frac{R_i^L}{T_k} \right\rceil C_k^L \tag{4.6}$$

This analysis makes two conservative assumptions. First, that the number of L-jobs for each interfering L-task is maximum: R_i^L is the latest time the mode switch may occur. Second, that all jobs of each interfering H-task take their H-WCET, C_j^H. However, these two assumptions cannot simultaneously hold. The big advantage of this analysis is that the calculation of R_i^* is independent of the instant when the mode switch occurs.

4.2.2.2 AMC-max

This analysis reduces the pessimism by taking into account the instant s, relative to the release of the job under analysis, at which the mode switch occurs. This allows for a more accurate accounting of the number of jobs of an interfering L-task τ_j, and therefore of its interference:

$$IL_j(s) = \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j^L \tag{4.7}$$

Where $IL_j(s)$ represents interference from a higher priority L-task τ_j till mode switch instant s. As explained in [6], Equation (4.7) assumes $\lfloor \frac{s}{T_j} \rfloor + 1$ interfering jobs of τ_j (instead of $\lceil \frac{s}{T_j} \rceil$) in order to accommodate the possibility of any L-jobs released up to the mode switch instant (inclusive) being allowed to execute to completion.

By taking into account s, it is also possible to be less conservative in the estimate of the number of jobs of a H-task τ_k that complete after the mode switch.

$$M(k,s,t) = min \left\{ \left\lceil \frac{t - s - (T_k - D_k)}{T_k} \right\rceil + 1, \left\lceil \frac{t}{T_k} \right\rceil \right\} \tag{4.8}$$

The number of jobs of a H-task that complete before a mode switch is obtained by subtracting this value from an upper bound on the total number of jobs of task τ_k that may be released in time interval t, i.e. $\lceil t/T_k \rceil$. This allows the interference of H-task τ_k to be safely upper-bounded because $C_k^L \leq C_k^H$. Therefore, the interference of H-task τ_k in a time interval of duration t, when the mode switch occurs s time units after the beginning of that interval, is given by:

$$IH_k(s,t) = M(k,s,t) \cdot C_k^H + \left(\left\lceil \frac{t}{T_k} \right\rceil - M(k,s,t) \right) C_k^L \tag{4.9}$$

Thus, the WCRT recurrence used to compute the response time of a job of a H-task τ_i that is affected by a mode switch is:

$$R_i^{(n)}(s) = C_i^H + \sum_{\tau_j \in hpL(i)} IL_j(s) + \sum_{\tau_k \in hpH(i)} IH_k(s, R_i^{(n-1)}(s)) \tag{4.10}$$

The solution of this recurrence, $R_i(s)$ depends on the mode switch instant, s. Thus, the response time of τ_i when affected by a mode switch is given by:

$$R_i^* = max\{R_i(s) : 0 < s \leq R_i^L\}$$

However, Baruah, Burns and Davis [6] argue that it is enough to compute $R_i(s)$ only for values of s that correspond to the release of jobs of higher-priority L-tasks, when the first job of all these tasks is released at the same time as the job of the task under analysis and these tasks' jobs arrive as soon as possible. This is because IH_k, as given by (4.9), is non-increasing as s increases and IL_j, as given by (4.7), is a step function whose value increases when jobs of task τ_j are released.

4.3 Multiframe mixed-criticality scheduling with constrained-deadline tasks

Sections 4.3.1 and 4.3.2 extend the static and adaptive mixed-criticality schedulability analyses presented for single frame task model in [6] for multiframe tasks, respectively. Section 4.3.2 on adaptive multiframe mixed-criticality further categorized into a slightly pessimistic but less time-consuming response time-bound approach (AMMC-rtb) and a complex but less pessimistic response time analysis approach (AMM-max). This section also discusses the efficient implementation scheme for cumulative interference computation functions g^*.

4.3.1 Analysis for static multiframe mixed-criticality systems (SMMC)

In this section, we extend the SMC analysis to the multiframe model. Like in the multiframe model, there is a single mode of operation in SMC. However tasks may have different levels of criticality and different WCET-estimates, one per criticality-level $C_i(\kappa_i)$ with constraint $C_i(\kappa_1) \geq C_i(\kappa_2)$. For instance, for a system with two criticality levels as assumed in this thesis $C_i(H) \geq C_i(L)$, see Section 4.1. Therefore, depending on the level of criticality of the task under analysis, different WCET-estimates are used in the WCRT-recurrence, see (4.4) and (4.5). For example, the WCRT of a H-task, (4.5), uses the L-WCET estimate to compute the interference by L-tasks and the H-WCET estimate to compute the interference by H-tasks. Therefore, we need to define additional g/G functions that take into account the criticality level of the WCET estimate.

Let $g^L(\tau_i,k)$ be the cumulative L-WCET of any sequence of k jobs of task τ_i. This function is defined as [11]'s g function, see (4.1), except that for job j it uses its L-WCET, $C^L_{i,j}$ rather than its criticality-oblivious WCET, C_i:

$$g^L(\tau_i,k) = \begin{cases} 0 & \text{if } k = 0 \\ max\left\{ \sum_{\ell=j}^{j+k-1} C^L_{i,(\ell \bmod F_i)} : 0 \leq j < F_i \right\} & \text{if } 1 \leq k \leq F_i \\ q \cdot g^L(\tau_i,F_i) + g^L(\tau_i,r) & \text{otherwise} \\ \quad \text{where } q = k \text{ div } F_i \text{ and } r = k \bmod F_i & \end{cases} \tag{4.11}$$

Analogously, we define $g^H(\tau_i,k)$, the cumulative H-WCET of any sequence of k jobs of task τ_i.

$$g^H(\tau_i,k) = \begin{cases} 0 & \text{if } k=0 \\ max\left\{ \sum_{\ell=j}^{j+k-1} C^H_{i,(\ell \bmod F_i)} : 0 \le j < F_i \right\} & \text{if } 1 \le k \le F_i \\ q \cdot g^H(\tau_i,F_i) + g^H(\tau_i,r) & \text{otherwise} \\ \quad \text{where } q = k \text{ div } F_i \text{ and } r = k \bmod F_i & \end{cases} \tag{4.12}$$

Furthermore, we define the corresponding G functions, see (4.2):

$$G^L(\tau_i,t) = g^L\left(\tau_i, \left\lceil \frac{t}{T_i} \right\rceil\right) \tag{4.13}$$

$$G^H(\tau_i,t) = g^H\left(\tau_i, \left\lceil \frac{t}{T_i} \right\rceil\right) \tag{4.14}$$

With these definitions, it is straightforward to generalize the SMC's WCRT recurrences to the multiframe model. For L-tasks, the recurrence becomes similar to that for the single-criticality task model (4.3):

$$R_i = g^L(\tau_i,1) + \sum_{\tau_j \in hp(i)} G^L(\tau_j,R_i) \tag{4.15}$$

Indeed, like in the original multiframe analysis [11], it suffices to compute the response time for the frame with largest L-WCET, $g^L(\tau_i,1)$. Furthermore, each of the terms in the summation, $G^L(\tau_j,R_i)$, is an upper-bound of the interference by jobs of higher-priority task τ_j in the response time R_i of the taks under analysis, τ_i. This is because it represents the cumulative L-WCET, i.e., computed with same level of confidence as that of $g^L(\tau_i,1)$, of any sequence of τ_j's jobs in time interval, R_i.

However, for H-tasks, the WCRT-recurrence must use the g/G functions corresponding to the criticality level of each task:

$$R_i = g^H(\tau_i,1) + \sum_{\tau_j \in hpL(i)} G^L(\tau_j,R_i) + \sum_{\tau_k \in hpH(i)} G^H(\tau_k,R_i) \tag{4.16}$$

Again, it is enough to compute the response time of the job with the largest H-WCET, $g^H(\tau_i,1)$. The interference by higher-priority H-task τ_k is bound by $G^H(\tau_k,R_i)$, the cumulative H-WCET of a sequence of jobs of τ_k in the response time interval, R_i, of the task under analysis, τ_i. The interference by higher-priority L-task τ_j is bound by $G^L(\tau_j,R_i)$, because in SMC the system terminates a job of an L-task as soon as its execution time exceeds the respective L-WCET estimate.

4.3.2 Analysis for adaptive multiframe mixed criticality systems (AMMC)

In this section, we develop a response time analysis for the adaptive mode-based variant of the multiframe mixed-criticality task model (as presented in Section 3.1.2). The idea is straightforward; we use AMC to count the number of jobs of each interfering task, and use Multiframe g/G functions, or similar functions, to compute the cumulative WCET of these jobs and therefore the WCRT of the different tasks.

As usual, we need to check the schedulability in:

L-mode: This can be done for all tasks using Baruah's WCRT recurrence, see (4.3), and replacing g (Eq. (4.1)) and G (Eq. (4.2)) functions with g^L and G^L in Equation (4.11) and (4.13), respectively.

H-mode: Again, we can reuse Baruah's WCRT recurrence and replace g and G functions with g^H and G^H (see Equation (4.12) and (4.14)), respectively, for all H-tasks.

Mode switch: In this case, we must consider, for each H-task, the response time of each job in one of its superframes when it is caught by a mode switch, i.e. when it is released before the mode switch but completes only after.

We now focus on the response time of jobs caught by a mode switch. Whereas in the single frame model, all jobs have the same parameters, in the multiframe model, different jobs have different parameters. Therefore, for each task under analysis, τ_i, we need to:

1. analyse the WCRT of each job in a superframe of τ_i;
2. consider the phasing of the interfering tasks, i.e., for each higher-priority task, what is the first job in the sequence of interfering jobs of that task.

Figure 4.1 illustrates the need for the latter. With respect to (1), consider two different jobs j and k ($j \neq k$) of a superframe of τ_i, it may be the case that $C^L_{i,j} > C^L_{i,k}$ and $C^H_{i,j} < C^H_{i,k}$. Thus, the schedulability of job j cannot be inferred from the response time of job k, nor vice-versa. Of course, it may not be necessary to compute the response time of all jobs. E.g., if $C^L_{i,j} > C^L_{i,k}$ and $C^H_{i,j} \geq C^H_{i,k}$, clearly it is enough to analyze the response time of job j of τ_i. Task τ_i is schedulable, if the WCRT of all jobs in one of its superframes does not exceed D_i.

In each of the following subsections, we describe how to generalize each of the AMC analyses for constrained-deadline tasks to the multiframe model.

4.3.2.1 AMC-rtb to multiframe tasks (AMMC-rtb)

As summarized in Section 4.2.2.1, AMC-rtb considers that each higher-priority L-task may interfere up to R^L_i. In the multiframe model, for each higher-priority L-task τ_k, we need to consider the cumulative WCET of a sequence of τ_k's jobs. Thus, the interference of a higher-priority L-task, τ_k, on job j of H-task τ_i is given by:

$$G^L(\tau_k, R^L_{i,j})$$

where $R^L_{i,j}$ is the WCRT in L-mode of job j of task τ_i. For each higher-priority H-task, τ_k, AMC-rtb assumes that every job of τ_k executes for its H-mode WCET (H-WCET for short) over the entire WCRT of task τ_i's job j when it is caught by a mode switch, $R^*_{i,j}$. Thus the interference of τ_k, on that job is given by:

$$G^H(\tau_k, R^*_{i,j})$$

Accordingly, the WCRT of job j of a H-task τ_i released before the mode switch, but completing after it, is upper-bounded by:

$$R^*_{i,j} = C^H_{i,j} + \sum_{k \in hpL(i)} G^L(\tau_k, R^L_{i,j}) + \sum_{k \in hpH(i)} G^H(\tau_k, R^*_{i,j}) \tag{4.17}$$

The WCRT of H-task τ_i is therefore the maximum of the WCRT of all frames in a superframe of τ_i:

$$R_i^* = max\{R_{i,j}^* : 0 \leq j < F_i\} \tag{4.18}$$

We can trade-off the computation cost of R_i^* for its pessimism, by bounding the mode switch instant to R_i^L rather than to $R_{i,j}^L$, where

$$R_i^L = max\{R_{i,j}^L : 0 \leq j < F_i\} \tag{4.19}$$

In this case, it suffices to compute $R_{i,j}^*$ for the job j with the maximum $C_{i,j}^H$, therefore the WCRT of H-task τ_i released before the mode switch but, completing after it, can be upper bounded by:

$$R_i^* = g^H(\tau_i, 1) + \sum_{j \in hpL(i)} G^L(\tau_j, R_i^L) + \sum_{k \in hpH(i)} G^H(\tau_k, R_i^*) \tag{4.20}$$

4.3.2.2 AMC-max to multiframe tasks (AMMC-max)

AMC-max reduces the pessimism in AMC-rtb by explicitly considering the mode switch instant, s, as summarized in Section 4.2.2.2. In particular, rather than assuming that all jobs of a H-task complete after the mode switch, AMC-max provides a tighter upper-bound on the number of jobs of a H-task that complete after the mode switch. As a consequence, AMC-max may underestimate the number of jobs of that task that complete before the mode switch. (This is also done in AMC-rtb, where the number of jobs of a H-task that complete before the mode switch is assumed to be 0.) Thus, critical for the safety of AMC-max is that the cumulative WCET of a task in a time interval comprising the mode switch instant does not decrease, if the number of jobs of that task that complete after the mode switch is overestimated.

To safely apply AMC-max's accounting of H-jobs that complete after the mode switch to the multiframe model, the following must hold:

Lemma 1. *Consider a sequence of successive jobs by a multiframe task τ_i with a fixed length n such that the last $n^H < n$ of these jobs terminate after the mode switch. Let n'^H ($n \geq n'^H \geq n^H$) be an overestimation of n^H. Then assuming the termination of the last n'^H jobs after the mode switch does not decrease the cumulative WCET.*

Proof. The proof is by induction on the number of jobs that terminate after the mode switch.

Base step: $n'^H = n^H$ and therefore the two job sequences are the same and so are their cumulative WCET.

Induction step Consider a sequence of these n jobs such that the last h ($n > h \geq n^H$) terminate after the mode switch. Let j be the last job in that sequence that terminates before the mode switch (such a job exists because $h < n$). If we assume that job j terminates after the mode switch, the difference between the cumulative WCET of the new and the previous sequences is $C_{i,j}^H - C_{i,j}^L$. Because by assumption $C_{i,j}^H \geq C_{i,j}^L$ this difference is non-negative, and therefore, by the induction step assumption, the cumulative WCET is not smaller than that of the sequence with n^H jobs terminating after the mode switch. □

Thus, for multiframe tasks (just as for single-frame tasks), it is safe for the analysis to overestimate, in a given sequence of jobs by an interfering task, the number of jobs that execute for their H-WCET. In particular,

this means that we can use AMC-max's upper-bound on the number of jobs that complete after the mode switch, (4.8), and use it to derive the number of jobs that complete before the mode switch to be used in the WCRT recurrence.

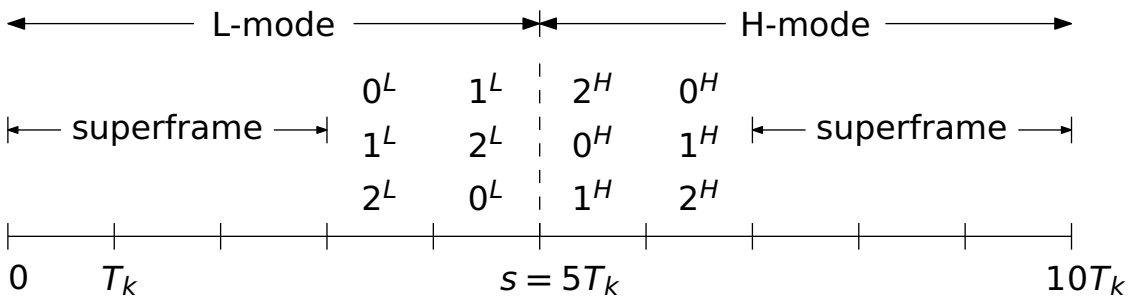


Figure 4.2: Interference of a H-task depends on the phasing of its jobs.

The challenge is that the phasing of the interfering task affects the amount of interference. The difference with respect to Baruah's multiframe analysis is that now there may be two subsequences, one before the mode switch and another after the mode switch. Figure 4.2 illustrates this. It shows an interfering task τ_k with 3 frames. The time interval under consideration is 10 times T_k and we assume that the mode switch occurs exactly at the middle of that interval. Thus, in this interval, there are 10 releases of jobs of τ_k, 5 before the mode switch and another 5 after the mode switch. Hence, there is a superframe plus 2 frames, both before and after the mode switch. The cumulative WCET of a super-frame is independent of the initial job in the superframe. Therefore, we need only to find the worst-case cumulative WCET of a sequence of 4 frames, such that the two first frames take their L-WCET, whereas the other 2 frames take their H-WCET. One of the following frame sequences leads to the worst-case cumulative execution time: $(0^L, 1^L, 2^H, 0^H)$, $(1^L, 2^L, 0^H, 1^H)$, $(2^L, 0^L, 1^H, 2^H)$, where we denote by i^L (i^H) a frame whose L-WCET (H-WCET) is equal to the L-WCET (H-WCET) of frame i.

Formally, we define function $g^*(\tau_i, \ell^L, \ell^H)$ that computes the cumulative WCET of a sequence of jobs of H-task τ_i, which is composed by a subsequence of ℓ^L jobs that take their L-WCET followed by a subsequence of ℓ^H that take their H-WCET:

$$g^*(\tau_i, \ell^L, \ell^H) = \begin{cases} g^H(\tau_i, \ell^H) & \text{if } \ell^L = 0 \\ g^L(\tau_i, \ell^L) & \text{if } \ell^L \neq 0 \wedge \ell^H = 0 \\ max \left\{ \sum_{k=j}^{j+\ell^L-1} C^L_{i,(k \bmod F_i)} \right. & \\ \quad \left. + \sum_{k=j+\ell^L}^{j+\ell^L+\ell^H-1} C^H_{i,(k \bmod F_i)} : 0 \leq j < F_i \right\} & \\ & \text{if } 1 \leq \ell^L < F_i \wedge 1 \leq \ell^H < F_i \\ q^L \cdot g^L(\tau_i, F_i) + g^*(\tau_i, r^L, r^H) + q^H \cdot g^H(\tau_i, F_i) & \\ & \text{otherwise} \end{cases} \tag{4.21}$$

where $q^L = (\ell^L \text{ div } F_i)$, $r^L = (\ell^L \bmod F_i)$, $q^H = (\ell^H \text{ div } F_i)$ and $r^H = (\ell^H \bmod F_i)$.

Furthermore, we define:

$$G^{L+}(\tau_i,t) = g^L\left(\tau_i, \left\lfloor \frac{t}{T_i} \right\rfloor + 1\right) \tag{4.22}$$

to account for the fact that AMC-max upper-bounds the number of jobs of higher-priority L-task τ_i by $\left\lfloor \frac{s}{T_i} \right\rfloor + 1$ rather than $\left\lceil \frac{s}{T_i} \right\rceil$, where s is the mode switch instant relative to the release of the job under analysis

Thus the WCRT of job j of H-task τ_i is upper bounded by:

$$R^*_{i,j}(s) = C^H_{i,j} + \sum_{k \in hpL(i)} G^{L+}(\tau_k,s) + \sum_{k \in hpH(i)} g^*(\tau_k, \ell^L(k,R^*_{i,j}(s),s), \ell^H(k,R^*_{i,j}(s),s)) \tag{4.23}$$

where:

$$\ell^L(k,t,s) = \left\lceil \frac{t}{T_k} \right\rceil - \ell^H(k,t,s) \tag{4.24}$$

$$\ell^H(k,t,s) = M(k,s,t) \tag{4.25}$$

Therefore:

$$R^*_{i,j} = max\{R^*_{i,j}(s) : 0 < s \leq R^L_{i,j}\}$$

Like in AMC-max, we need to compute the WCRT of job j of task τ_i only for the values of s that correspond to the release of jobs of higher-priority L-tasks, when the first job of each of these tasks is released at the same time as the job under analysis and the other jobs arrive as soon as possible.

Like in AMMC-rtb, to compute the WCRT of task τ_i, we need to compute the response time of all jobs in a superframe and take the maximum of these values:

$$R^*_i = max\{R^*_{i,j} : 0 \leq j < F_i\}$$

Again, we can trade-off computation cost for pessimism, by always considering all the s values smaller than R^L_i, see (4.19), i.e.:

$$R^*_i = max\{R^*_i(s) : 0 < s \leq R^L_i\} \tag{4.26}$$

In this case, rather than computing the WCRT for all the jobs in a superframe it suffices to compute the WCRT for the longest H-mode job in a superframe. Thus (4.23) becomes:

$$R^*_i(s) = g^H(\tau_i,1) + \sum_{j \in hpL(i)} G^{L+}(\tau_j,s) + \sum_{k \in hpH(i)} g^*(\tau_k, \ell^L(k,R^*_i(s),s), \ell^H(k,R^*_i(s),s)) \tag{4.27}$$

where ℓ^L and ℓ^H are given by (4.24) and (4.25), respectively.

4.3.2.3 Implementation of the g^* function

For an efficient implementation of the analyses described in the previous section, it is important to be able to efficiently implement the g^* function.

An efficient implementation of the g function is given in [11] that relies on an array M_k of length $F_k + 1$ per task τ_k, such that $M_k[i] = g(\tau_k, i)$, for $0 \leq i \leq F_k$. This array is computed before initiating the response time

analysis, and the complexity of its computation is $\mathcal{O}(F_k^2)$. Analogously, in order to implement the g^* function, we define a 2-dimensional $(F_k+1)\times(F_k+1)$ matrix, M_k per H-task τ_k, where $M_k[i][j] = g^*(\tau_k, i, j)$ for $0 \le i \le F_k$ and $0 \le j \le F_k$. Note that element $M_k[0][j] = g^H(\tau_k, j)$, and therefore $M_k[0][F_k]$ is the H-WCET of a superframe of τ_k. Likewise, element $M_k[i][0] = g^L(\tau_k, i)$, and therefore $M_k[F_k][0]$ is the L-WCET of a superframe of τ_k. Furthermore, all other elements of the last column and of the last line are not needed and therefore need not be computed. Indeed, the value of $g^*(\tau_k, \ell, h)$ can be efficiently computed from the elements of the M_k matrix, as follows:

$$(\ell \text{ div } F_k)\cdot M_k[F_k][0] + M_k[\ell \bmod F_k][h \bmod F_k] + (h \text{ div } F_k)\cdot M_k[0][F_k]$$

The computation of the matrix M_k is very similar to the computation of array M_k in [11]. The first step is to compute two $F_k \times F_k$ matrices L_k and H_k. These matrices are just like $F_k \times F_k$ matrix D_k in [11], except that L_k and H_k use the L-WCET and H-WCET of τ_k's frames, respectively. More specifically, element $L_k[i][j] = \sum_{\ell=i}^{i+j} C^L_{k,(\ell mod F_k)}$, i.e. element $L_k[i][j]$ is the the cumulative WCET of a sequence of $(i+1)$ L-frames that starts with frame j. Algorithm 1 shows the code segment presented in [11], adapted to initialize the elements of matrix L_k. Except for the notation, the only difference is the use of the L-WCET in Lines 2 and 5. The initialization of H_k is identical. The complexity of this code segment is $\mathcal{O}(F_k^2)$.

Algorithm 1 Computation of auxiliary matrix L_k.

// L_k is a $F_k \times F_k$ matrix, such that $L_k[i][j] = \sum_{\ell=i}^{i+j} C^L_{k,(\ell mod F_k)}$

```
1: for (j:=0; j < F_k; j:= j+1) // First line
2:     L_k[0][j] := C^L_{k,j};
3: for (i:=1; i < F_k; i:=i+1)
4:     for (j:=0; j < F_k; j:= j+1)
5:         L_k[i][j] := L_k[i-1][j] + C^L_{i,(i+j) mod F_k};
```

Algorithm 2 shows the code segment used for computing matrix M_k from matrices L_k and H_k. For the sake of readability, we use the $max(m,n)$ function, which returns the maximum of its two integer arguments. The algorithm uses the elements of L_k and H_k in a very similar way to the computation of the corresponding array in [11] from the elements of the D_k matrix. As a result, the complexity for computing each element of the M_k matrix, $\mathcal{O}(F_k)$, is the same as the complexity for computing each element of the array in [11]. However because the M_k matrix has $(F_k+1)^2$ elements rather than (F_k+1) elements, the complexity of the computation of matrix M_k is $\mathcal{O}(F_k^3)$ rather than $\mathcal{O}(F_k^2)$. Just like in Algorithm 1, note that all elements of the last row except for the first one are not computed, as they are not needed. The same for all elements of the last column except the first one.

4.4 Background on the schedulability analysis of arbitrary-deadline tasks

So far, we have formulated analysis for multiframe mixed-criticality systems under the assumption that task deadlines are constrained. We now extend those derivations for a more general *arbitrary-deadline* model, whereby it does not necessarily hold that $D_i \le T_i$ for a task τ_i. The previously formulated analysis no longer

Algorithm 2 Computation of $(F_k+1)\times(F_k+1)$ matrix M_k. $M_k[i][j]$ is the cumulative WCET of any sequence of i L-jobs followed by j H-jobs of task τ_k

```
1:  M_k[0][0] := 0 // First row is g^H: this is the algorithm in [11]
2:  for (h:=1; h ≤ F_k; h:=h+1) { // h is the length of the sequence
3:      M_k[0][h] := 0;
4:      for (i:=0; i < F_k; i:=i+1) // i is the first frame
5:          M_k[0][h] := max(M_k[0][h], H_k[h-1][i]);
6:  } // First column is g^L: this is the algorithm in [11]
7:  for (ℓ:=1; ℓ ≤ F_k; ℓ:=ℓ+1) { // ℓ is the length of the sequence
8:      M_k[ℓ][0] := 0;
9:      for (i:=0; i < F_k; i:=i+1) // i is the first frame
10:         M_k[ℓ][0] := max(M_k[ℓ][0], L_k[ℓ-1][i]);
11: } // Remaining elements: based on the algorithm in [11]
12: for (ℓ:=1; ℓ < F_k; ℓ:=ℓ+1) // ℓ is the length of the L-sequence
13:     for (h:=1; h < F_k; h:=h+1) { // h: H-sequence's length
14:         M_k[ℓ][h] := 0;
15:         for (i:=0; i < F_k; i:=i+1) // i: first frame in L-sequence
16:             M_k[ℓ][h] := max(M_k[ℓ][h],
17:                              L_k[ℓ-1][i] + H_k[h-1][(i+ℓ) mod F_k]);
18:     }
```

holds in that case, because a job by a task with $D_i > T_i$ may additionally suffer interference from earlier-released jobs *of the same task*. To address this, we draw from the work by Tindell et al. [33] on the fixed-priority preemptive scheduling of arbitrary-deadline single-frame tasks.

Under that approach [33], the schedulability of a task τ_i with arbitrary deadline ($D_i > T_i$) can be computed using the concept of level-i busy period [112], i.e., a time interval while the processor is busy executing jobs of tasks with priority equal or higher than that of τ_i. Indeed, Lehoczky proved that the longest response time for a job of a such a task τ_i occurs during a level-i busy period initiated by a critical instant, i.e., a time instant when all tasks with priority equal or higher than τ_i are released. However, unlike for constrained deadline tasks, the worst-case response time may not occur for the first job of τ_i after the critical instant. Therefore, to determine whether τ_i is schedulable, one needs to check that the response time of all jobs of τ_i in a level-i busy period (that starts at a critical instant) does not exceed D_i.

4.4.1 Analysis for fixed priority preemptive scheduling (FPPS) for single criticality tasks with arbitrary deadlines

To compute the response time of job q in a level-i busy period, i.e., of a job that starts $q \cdot T_i$ time units after the beginning of the busy period, Tindell [33] extended standard worst-case response time analysis for constrained deadline tasks [28] to compute the completion time of job $q+1$ with respect to the beginning of the level-i busy period:

$$r_i(q) = (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i(q)}{T_j} \right\rceil C_j \tag{4.28}$$

Indeed, the left hand side of (4.28) is the cumulative processor requirement by all jobs of tasks with higher priority than τ_i and of the first $q+1$ jobs after the critical instant. Because the last of these jobs is released $q \cdot T_i$

time units after the beginning of the level-i busy period, its response time is given by:

$$R_i(q) = r_i(q) - qT_i \tag{4.29}$$

As mentioned above, a task τ_i is schedulable if $R_i(q) \leq D_i$, for all jobs q in a level-i busy period. Note that we need not know *a priori* how many jobs there are in a level-i busy period. The computation of $r_i(q)$ can start for $q = 0$ and stop when either $R_i(q) > D_i$, in which case τ_i is not schedulable, or when $r_i(q) \leq (q+1) \cdot T_i$, i.e., at the last job of a level-i busy period, in which case τ_i is schedulable.

4.4.2 Mixed-criticality scheduling with arbitrary deadline tasks

Burns and Davis [113] extended SMC, AMC-rtb and AMC-max for task sets with constrained deadlines to task sets with arbitrary deadlines. Actually, SMC's extension is straightforward. To compute $r_i(q)$, (4.28), instead of using C_i and C_j, one should use $C_i^{L_i}$ and $C_j^{min(L_i,L_j)}$, where L_i and L_j are the criticality levels of tasks τ_i and τ_j, respectively. I.e. if the task under analysis is an L-task, then the worst-case completion time of its job q, $r_i(q)$, is computed using the L-WCET estimates of higher-priority tasks.

$$r_i^L(q) = (q+1)C_i^L + \sum_{j \in hp(i)} \left\lceil \frac{r_i^L(q)}{T_j} \right\rceil C_j^L \tag{4.30}$$

If the task under analysis is a H-task, then $r_i(q)$ is computed using the L-WCET for higher-priority L-tasks and H-WCET for higher-priority H-tasks as done in Eq. 4.31.

$$r_i^H(q) = (q+1)C_i^H + \sum_{j \in hpH(i)} \left\lceil \frac{r_i^H(q)}{T_j} \right\rceil C_j^H + \sum_{k \in hpL(i)} \left\lceil \frac{r_i^H(q)}{T_k} \right\rceil C_k^L \tag{4.31}$$

Finally, we can use Eq. 4.32 and 4.33 to compute overall response time of qth instance of task τ_i in the L and H-mode, respectively. A task τ_i is declared schedulable only if the response time of all of its busy period releases q in both L and H-mode is less than its deadline (i.e., $R_i^L < D_i$ and $R_i^H < D_i$). Where:

$$R_i^L(q) = r_i^L(q) - qT_i \tag{4.32}$$

and

$$R_i^H(q) = r_i^H(q) - qT_i \tag{4.33}$$

4.4.2.1 Adaptive mixed-criticality response time bound with Arbitrary deadlines (AMC-rtb-Arb)

In AMC, the analysis must ensure schedulability in both L- and H-mode as well as upon a mode switch.

The completion time of a job in L-mode of a task τ_i in a level-i busy period released qT_i time units after the beginning of that busy period can be adapted from (4.28) by using the appropriate WCET for the different jobs:

$$r_i^L(q) = (q+1)C_i^L + \sum_{j \in hp(i)} \left\lceil \frac{r_i^L(q)}{T_j} \right\rceil C_j^L \tag{4.34}$$

and:

$$R_i^L(q) = r_i^L(q) - qT_i$$

Task τ_i is schedulable in L-mode, if $R_i^L(q) < D_i$ for all jobs q in a level-i busy period in L-mode.

Likewise, we can obtain an expression for the steady H-mode. However, the steady H-mode is dominated by H-mode upon a mode switch.

$$r_i^H(q) = (q+1)C_i^H + \sum_{j \in hpH(i)} \left\lceil \frac{r_i^H(q)}{T_j} \right\rceil C_j^H \tag{4.35}$$

and $R_i^H(q)$ is obtained by using equation (4.33).

For AMC-rtb-Arb, the completion time of a job of a H-task τ_i affected by a mode switch is given by [113]:

$$r_i^H(q) = (q+1)C_i^H + \sum_{j \in hpH(i)} \left\lceil \frac{r_i^H(q)}{T_j} \right\rceil C_j^H + \sum_{k \in hpL(i)} \left\lceil \frac{r_i^L(min(q,p))}{T_k} \right\rceil C_k^L \tag{4.36}$$

where p is the last job number in a level-i busy period in L-mode.

The main issue in the analysis upon a mode switch is to bound the duration of the L-mode interval before the mode switch. Although, $r_i^L(q)$ appears to be a straightforward choice, the level-i busy period upon a mode switch may be longer than in L-mode. If τ_i is schedulable in L-mode, then the level-i busy period must be bounded, and there is a last job from τ_i in that period, job p. Clearly, the mode switch must occur before the completion time of that job, otherwise that job would not have been affected by the mode switch, nor any job that follows it. Therefore, (4.36) uses $r_i^L(min(p,q))$ rather than $r_i^L(q)$.

As before:

$$R_i^H(q) = r_i^H(q) - qT_i$$

and H-task τ_i is schedulable in H-mode (including mode switch), if $R_i^H(q) < D_i$ for all jobs q in a level-i busy period that comprises a mode switch.

4.4.2.2 Arbitrary-deadline analysis for AMC-max (AMC-max-Arb)

AMC-max-Arb differs from AMC-rtb-Arb only on the completion time expression for jobs of a H-task τ_i that are affected by a mode switch. Assuming the mode switch occurs at time s (measured with respect to the beginning of the level-i busy period), the completion time of job q of such a task is given by [113]:

$$r_i(q,s) = X(q,s,r_i(q,s))C_i^H + Y(q,s,r_i(q,s))C_i^L + IL(s) + IH(s,r_i(q,s)) \tag{4.37}$$

where $X(q,s,t)$ and $Y(q,s,t)$ are the number of the first $q+1$ jobs, i.e. until job q, of τ_i that complete until instant t after and before the mode switch, respectively. Thus $X(q,s,t) + Y(q,s,t) = q+1$. AMC-max-Arb conservatively (over-)estimates $X(q,s,t)$ as follows:

$$X(q,s,t) = min\left(\left\lceil \frac{t - s + (D_i - T_i)}{T_i} \right\rceil + 1, q+1 \right) \tag{4.38}$$

and derives $Y(q,s,t)$ from $X(q,s,t) + Y(q,s,t) = q+1$.

$IL(s)$ and $IH(s,t)$ in (4.37) are given by the same summations as in AMC-max (for constrained deadline task sets), respectively (4.7) and (4.9).

Similarly to AMC-max, $r_i(q,s)$ must be computed for every value of $s \in [0, r_i^L(q))$ equal to the release time of L-tasks with higher priority than τ_i. Let, $r_i^*(q)$ be the maximum of all these values. Then, the worst-case response time for job q in a level-i busy period that comprises a mode switch can be computed as:

$$R_i^*(q) = r_i^*(q) - qT_i \tag{4.39}$$

and task τ_i is schedulable in H-mode (including mode switch), if $R_i^*(q) < D_i$ for all jobs q in a level-i busy period that comprises the mode switch.

4.5 Generalization of multiframe mixed-criticality scheduling analyses for arbitrary-deadline tasks

We now formulate the generalized schedulability analyses techniques (SMMC-Arb, AMMC-rtb-Arb and AMMC-max-Arb) for arbitrary-deadline multiframe mixed-criticality tasks.

4.5.1 Static multiframe mixed criticality with arbitrary deadline (SMMC-Arb)

In this section, we extend the static multiframe mixed-criticality analysis for the arbitrary-deadline task model.

L-task analysis: The response time of an L-task in SMMC (and in SMC) considers the L-WCETs of all tasks, independently of their criticality. Therefore, the worst-case completion time of job q of L-task τ_i in a level-i busy period can be computed as follows:

$$r_i(q) = g^L(\tau_i, (q+1)) + \sum_{j \in hp(i)} G^L(\tau_j, r_i(q)) \tag{4.40}$$

Indeed, $g^L(\tau_i, (q+1))$ bounds the cumulative processing demand of any sequence of $q+1$ jobs, i.e. up to job q, of the task under analysis, using its own criticality level WCET-estimate. Furthermore, each of the terms in the summation, $G^L(\tau_j, r_i(q))$, is an upper bound of the interference of any sequence of jobs of higher-priority task τ_j in time interval $r_i(q)$, using their L-mode WCET estimate, i.e., computed with a confidence corresponding to the criticality level of the task under analysis.

Just as for SMC-Arb, the corresponding worst-case response time is:

$$R_i(q) = r_i(q) - qT_i \tag{4.41}$$

Also like in SMC-Arb, a multiframe L-task τ_i is schedulable under SMMC-Arb, if the response time of all its jobs in a level-i busy period does not exceed its deadline, D_i.

H-task analysis: In SMMC (and in SMC), the response time analysis for a H-task assumes that each task executes for the WCET estimate corresponding to its criticality level. Therefore, the worst-case completion

time of job q of H-task τ_i in a level-i busy period can be computed as follows: Since all the tasks are allowed to execute at all times, the completion time of the task under analysis τ_i can be computed as follows:

$$r_i(q) = g^H(\tau_i, q+1) + \sum_{j \in hpL(i)} G^L(\tau_j, r_i(q)) + \sum_{k \in hpH(i)} G^H(\tau_k, r_i(q)) \tag{4.42}$$

Indeed, the first term on the right-hand side of this equation, $g^H(\tau_i, q+1)$ bounds the cumulative WCET of any sequence of $q+1$ jobs, i.e. up to job q, of the task under analysis, using their H-WCET estimates. Furthermore, each term in the first summation on the right-hand side, $G^L(\tau_j, r_i(q))$, bounds the interference of the largest sequence of jobs of higher-priority L-task τ_j in a time interval of duration $r_i(q)$, using their L-WCET estimates. Finally, each term in the second summation on the right-hand side, $G^H(\tau_k, r_i(q))$, bounds the interference of the largest sequence of jobs of higher-priority H-task τ_k in a time interval of duration $r_i(q)$, using their H-WCET estimates. Note that the two summations are similar to those used in the estimate of the worst-case response time for SMMC (with constrained deadlines) (4.16) , R_i, except that time interval considered is $r_i(q)$, i.e. the completion time of job q of τ_i, rather than R_i.

The worst-case response time for such a job is determined as usual for a task with an arbitrary deadline, i.e. using (4.41). Furthermore, like in SMC-Arb or for a multiframe L-task under SMMC-Arb, a multiframe H-task τ_i is schedulable under SMMC-Arb, if the response time of all its jobs in a level-i busy period does not exceed its deadline, D_i.

4.5.2 Analysis for adaptive multiframe mixed criticality systems (AMMC)

In this section, we extend the AMMC analysis for tasks with constrained deadlines in Section 4.3.2 to tasks with arbitrary deadlines, by combining those analyses with the analyses for mixed-criticality systems with arbitrary deadlines(AMC-rtb-Arb and AMC-max-Arb) in [113]. We refer to these two new analyses as AMMC-rtb-Arb and AMMC-max-Arb.

4.5.2.1 Adaptive multiframe mixed criticality-rtb with arbitrary deadline (AMMC-rtb-Arb)

In AMMC the system operates in two modes. Therefore, we need to analyze both L- and H-modes.

L-mode analysis: In L-mode, independently of a task's level, AMC, and therefore AMMC, uses only L-mode WCET estimates. Therefore, the completion time of job q of a multiframe task τ_i, whether it is an L-task or a H-task, is given by:

$$r_i^L(q) = g^L(\tau_i, (q+1)) + \sum_{j \in hp(i)} G^L(\tau_j, r_i^L(q)) \tag{4.43}$$

The first term bounds the processing requirement of any sequence of $q+1$ jobs of the task under analysis, τ_i, using their L-WCET estimates. Also, each of the terms of the summation, $G^L(\tau_j, r_i^L(q))$, upper bounds the processing requirement in L-mode of any sequence of jobs of higher-priority task τ_j over a time interval equal to the completion time in L-mode of any sequence of $q+1$ jobs of τ_i, the task under analysis.

H-mode analysis: AMC-rtb-Arb (and AMC-rtb) computes the worst-case response time of a job of a H-task τ_i by bounding the number of jobs of each higher-priority task, both L- and H-tasks, and assuming that all jobs of H-tasks execute for their H-mode WCET. In AMMC-rtb-Arb we do the same, but we need to consider not only the number of jobs of each interfering task, see (4.36), but also their order. Therefore, the completion time of job q of a multiframe H-task τ_i in a level-i busy period comprising the mode switch instant is given by:

$$r_i^H(q) = g^H(\tau_i,(q+1)) + \sum_{j \in hpL(i)} G^L(\tau_j, r_i^L(min(p,q))) + \sum_{k \in hpH(i)} G^H(\tau_k, r_i^H(q)) \tag{4.44}$$

where p is the last job of τ_i in a level-i busy period in L-mode operation. Indeed, the first term on (4.44)'s right-hand side upper-bounds the processing requirement of any sequence of $q+1$ jobs of task τ_i, assuming their H-WCET. Each term of the second summation, upper-bounds the interference from the largest sequence of jobs of higher-priority H-task τ_k over $r_i^H(q)$, i.e., the worst-case completion time for job q of τ_i in a level-i busy period, assuming that each of these jobs takes their H-WCET. Finally, each term of the first summation, upper-bounds the interference from the largest sequence of jobs of higher-priority L-task τ_j over the longest completion time of either job q or job p of τ_i, whichever is smaller, assuming that each of these jobs takes their L-WCET. Indeed, as observed in [113], if $p < q$, the mode switch must occur before $r_i^L(p)$. Otherwise, τ_i would have not been caught by a mode switch.

Note that in this analysis we are trading-off computation time for pessimism. Indeed, $r_i^L(min(p,q))$ is the maximum response time in L-mode for any job sequence with the specified number of jobs, but this may occur for a sequence that starts with a job that is different from the job that leads to the maximum cumulative H-WCET of a sequence of $q+1$ jobs. This is similar to the trade-off described in Subsection 4.3.2.1. A tighter analysis would require to upper-bound the completion time of the F_i sequences of $q+1$ jobs, one per job in a superframe. For each sequence, the worst-case completion time would be computed using the cumulative H-WCET of the jobs in that sequence and also the respective L-mode completion time to upper-bound the interference by L-mode tasks. Finally, the response time would be computed by taking the maximum of these completion times and subtracting $q.T_i$.

4.5.2.2 Adaptive multiframe mixed criticality-max with arbitrary-deadlines (AMMC-max-Arb)

We now extend AMMC-max for arbitrary deadlines (AMMC-max-Arb). AMC-max and AMC-max-Arb differ from AMC-rtb and AMC-rtb-Arb, respectively, only in the analysis of H-mode operation. So do AMMC-max-Arb and AMMC-rtb-Arb. That is, AMMC-rtb-Arb analysis for L-mode operation (see Subsection 4.5.2.1) is also applicable to AMMC-max-Arb in L-mode operation. Therefore, in this section, we will only discuss the analysis of H-tasks in H-mode, including mode switch.

H-mode analysis: The completion time of job q of a H-task τ_i in a level-i busy period that comprises a mode switch, depends on:

1. The interference from jobs of higher-priority L-tasks, τ_j, released before the mode switch. This can be computed as for AMMC-max with constrained deadlines, in Subsection 4.3.2.2, using the $G^{L+}(\tau_j, s)$ function, (4.22).

2. The interference from jobs of higher-priority H-tasks, τ_k, released during the level-i busy period before completion of that job. Again, this can be computed as for AMMC-max with constrained deadlines, using the
$g^*(\tau_k, \ell^L(k,t,s), \ell^H(k,t,s))$ function, (4.21), and appropriate values for t and s.

3. The interference from the previous jobs of the task under analysis, released during the level-i busy period before job q.

To compute the last interference component, we observe that some of the jobs of τ_i before job q complete before the mode switch instant s, whereas the remaining ones complete after the mode switch. We observe that Lemma 1 is still valid for arbitrary-deadline tasks, because neither it nor its proof depend on $D_i \leq T_i$. Thus, to upper bound the interference, we upper bound the number of jobs that complete after the mode switch just like for AMC-max-Arb. Therefore we can use $X(q,s,t)$, given by (4.38). The number of jobs completed before the mode switch can also be estimated as for AMC-max-Arb: $Y(q,s,t) = q+1-X(q,s,t)$. Thus, the completion time of job q of a H-task τ_i in a level-i busy period comprising a mode switch can be computed using the g^* function as follows:

$$r_i^*(q,s) = g^*(\tau_i, Y(q,s,r_i^*(q,s)), X(q,s,r_i^*(q,s))) + \sum_{j \in hpL(i)} G^{L+}(\tau_j, s) + \sum_{k \in hpH(i)} g^*(\tau_k, l^L(k, r_i^*(q,s), s), l^H(k, r_i^*(q,s), s)) \tag{4.45}$$

As for AMC-max, AMC-max-Arb and AMMC-max, it suffices to compute $r_i^*(q,s)$ only for values of s that are equal to the release time of L-tasks with higher priority than τ_i. The justification is that $g^*(\tau_i, l, h)$ is non-increasing as s increases and $G^{L+}(\tau_j, s)$ is a step function whose value increases for values of s corresponding to releases of jobs of task τ_j. However, in a slight optimization over [113], and similarly as in AMC-rtb-Arb and AMMC-rtb-Arb, we only need to consider such values of s from the interval $[0, r_i^L(min(p,q)))$, instead of the potentially larger interval $[0, r_i^L(q))$, where p is the job number of the last job in a level-i busy period in L-mode. The justification is that $r_i^L(p)$ is the longest i-level busy period in L-mode, thus if τ_i is affected by a mode switch, this must occur before $r_i^L(p)$.

Let $r_i^*(q)$ be the maximum of all the $r_i^*(q,s)$ computed values. Then, the worst-case response time for job q in a level-i busy period that comprises a mode switch can be computed as for AMC-max-Arb, i.e.,

$$R_i^*(q) = r_i^*(q) - qT_i \tag{4.46}$$

Again, in this analysis (4.45), we are trading-off computation time for pessimism. Indeed, $r_i^L(min(p,q))$ is the maximum completion time in L-mode over all possible sequences with $min(p,q)+1$ jobs of τ_i. So, as described above, we must compute the completion time of $q+1$ jobs for every value of s smaller than $r_i^L(min(p,q))$ that is equal to the release time of an L-job with higher priority than τ_i. This value of s will affect every term on the RHS of (4.45), and in particular the first term, which is the maximum cumulative execution of $q+1$ jobs, such that some of them complete before the mode switch, s, and others complete after the mode switch. Assume that $r_i^*(q,s)$ takes the maximum value for some sequence of $q+1$ jobs of τ_i such that y of them complete before the mode switch and x of them complete after the mode switch. It may be the case

that the completion time of that same sequence of y jobs in L-mode is shorter than the value of s. Thus, the maximum completion time computed for that value of s has some pessimism. A tighter analysis would require to upper-bound the completion time of the F_i sequences of $q+1$ jobs of τ_i, one per frame in a superframe. For each sequence, the worst-case completion time would be computed using its cumulative WCET and also the respective L-mode completion time to upper-bound the values of s that would have to be used. Finally, the response time would be computed by taking the maximum of these completion times and subtracting $q.T_i$.

4.6 Example

In this section, we illustrate the application of the different analyses described in the previous section to the task set shown in Table 4.1. This task set comprises one L-task and two H-tasks. The tasks are listed in decreasing priority order, as found by Audsley's priority assignment, which is discussed in Section 4.7. We focus on the response time at mode switch. Furthermore, we consider only the response time analysis of task τ_3, which is the most interesting, as it is interfered with by the other two tasks. The response time analysis of the L- and H-modes, and also for the remaining tasks, is similar.

Table 4.1: Example multiframe task set τ with three tasks.

task	κ	C^L	C^H	T	D	Priority
τ_1	L	{1,2,6,4 }	{ }	10	10	H
τ_2	H	{3,5,2 }	{6,10,4 }	20	20	M
τ_3	H	{1,2 }	{ 2,4}	30	40	L

4.6.1 SMMC-Arb

For this analysis, the completion time of job q of task τ_i is given by recurrence (4.42), which we repeat here for convenience:

$$r_i(q) = g^H(\tau_i, q+1) + \sum_{j \in hpL(i)} G^L(\tau_j, r_i(q)) + \sum_{k \in hpH(i)} G^H(\tau_k, r_i(q))$$

Thus, the recurrence for completion time of the first job ($q = 0$) of τ_3 of the example task set becomes:

$$r_3(0) = 4 + G^L(\tau_1, r_3(0)) + G^H(\tau_2, r_3(0))$$

which converges to the value 33. Thus, the response time of the first job of τ_3, see (4.29), is:

$$R_3(0) = 33 - 0 * T_3 = 33$$

which is shorter than its deadline, $D_3 = 40$. However, it is larger than its interarrival time, $T_3 = 30$, thus we need to compute the response time for the 2nd job. The completion time of the second job, relative to the release of the first job, is obtained by solving the recurrence:

$$r_3(1) = 4 + 2 + G^L(\tau_1, r_3(1)) + G^H(\tau_2, r_3(1))$$

which converges to the value 35. Thus its response time is:

$$R_3(1) = 35 - 1 * T_3 = 5$$

Again, this is smaller than τ_3's deadline. Furthermore, it is smaller than τ_3's interarrival time, therefore there is no need to consider further jobs, and we can conclude that τ_3 is schedulable when it is assigned the lowest priority.

4.6.2 AMMC-rtb-Arb

For this analysis, we need first to analyze the response time in L-mode for the different jobs in a i-busy period, because the corresponding L-mode completion times are used in the H-mode completion times recurrences.

The recurrence for computing the completion time of job q of task τ_i in L-mode is given by recurrence (4.43), which we repeat here for convenience:

$$r_i^L(q) = g^L(\tau_i,(q+1)) + \sum_{k \in hp(i)} G^L(\tau_k, r_i^L(q))$$

Thus, the completion time of τ_3's first job in L-mode can be obtained by solving the recurrence:

$$r_3^L(0) = 2 + G^L(\tau_1, r_3^L(0)) + G^L(\tau_2, r_3^L(0))$$

which converges to 17. Therefore, $R_3^L(0) = 17$. Since this is shorter than both the deadline and the interarrival time of τ_3, we are done with L-mode analysis.

The recurrence (4.44) is used for computing the completion time of job q of task τ_i in H-mode, which we repeat here for convenience:

$$r_i^H(q) = g^H(\tau_i,(q+1)) + \sum_{j \in hpL(i)} G^L(\tau_j, r_i^L(min(p,q))) + \sum_{k \in hpH(i)} G^H(\tau_k, r_i^H(q))$$

Thus, the completion time of τ_3's first job upon a mode switch is given by:

$$r_3^H(0) = 4 + G^L(\tau_1, 17) + G^H(\tau_2, r_3^H(0)) \tag{4.47}$$

which converges to 30. Thus the response time of the first job is $R_3^H(0) = 30$, which is not greater than the deadline. Furthermore, it is also not greater than τ_3's interarrival time. So, this means that we have completed the response time analysis and that τ_3 is schedulable when it is assigned the lowest priority.

4.6.3 AMMC-max-Arb

As already mentioned, the AMMC-max-Arb analysis differs from AMMC-rtb-Arb only in the WCRT recurrence for the case of a mode switch. To reduce the pessimism of AMMC-rtb-Arb, the AMMC-max-Arb completion time recurrence for job q in a level-i busy period, (4.45), which we repeat below for convenience, takes

Table 4.2: Response time of all the tasks.

task	κ	D	WCRT		
			SMMC	AMMC-rtb-Arb	AMMC-max-Arb
τ_1	L	10	6	(6, -, -)	(6, -, -)
τ_2	H	20	20	(15, 20, 10)	(15, 20, 10)
τ_3	H	40	$R_3(0) = 33$ $R_3(1) = 5$	(17, 30, 14)	(17, 24, 14)

The WCRT triples in the AMMC-rtb-Arb and AMMC-max-Arb columns are for L-mode, mode switch and H-mode, respectively.

into account the mode switch instant, s.

$$r_i^*(q,s) = g^*(\tau_i, Y(q,s,r_i^*(q,s)), X(q,s,r_i^*(q,s))) + \sum_{j \in hpL(i)} G^{L+}(\tau_j, s) + \sum_{k \in hpH(i)} g^*(\tau_k, l^L(k, r_i^*(q,s), s), l^H(k, r_i^*(q,s), s))$$

As mentioned, it suffices to compute $r_i^*(q,s)$ only for values of s that are equal to the release time of L-tasks with higher priority than τ_i in $[0, r_i^L(min(p,q)))$, where p is the job number of the last job in a level-i busy period in L-mode. Once the completion time of job q has been determined, its worst-case response time can be easily computed: $R_i^*(q) = r_i^*(q) - q \cdot T_i$, where $r_i^*(q)$ is the maximum of all $r_i^*(q,s)$ values computed.

Thus to compute the completion time of the first job, $q = 0$, of τ_3, $r_3^*(0)$, we must consider the values of s in the interval $[0, r_3^L(0)) = [0, 17)$, corresponding to the release of task τ_1, the only L-task with a priority higher than τ_3. Because $T_1 = 10$, there are two possible values for s: 0 and 10.

Considering $s = 0$ first, we get the recurrence:

$$\begin{aligned} r_3^*(0,0) = & g*(\tau_3, Y(0,0,r_3^*(0,0)), X(0,0,r_3^*(0,0))) \\ & + G^{L+}(\tau_1, 0) + g^*(\tau_2, l^L(2, r_3^*(0,0), 0), l^H(2, r_3^*(0,0), 0)) \end{aligned}$$

which converges to 20. Since the corresponding response time is not greater than the D_3, we now compute the completion time of the first job for $s = 10$, using the recurrence:

$$\begin{aligned} r_3^*(0,10) = & g^*(\tau_3, Y(0,0,r_3^*(0,10)), X(0,0,r_3^*(0,10))) \\ & + G^{L+}(\tau_1, 10) + g^*(\tau_2, l^L(2, r_3^*(0,10), 10), l^H(2, r_3^*(0,10), 10)) \end{aligned}$$

which converges to 24. Thus: $r_3^*(0) = max\{20, 24\} = 24$, and its worst-case response time is: $R_3^*(0) = r_3^*(0) - 0 \cdot T_3 = 24$. Since this value is smaller than D_3, the first job of τ_3 meets the deadline. Furthermore, because $R_3^*(0) = 24 \leq T_3 = 30$, there is at most one active job of τ_3 in any 3-level busy period. Thus, we can conclude that τ_3 is schedulable under AMMC-max-Arb when assigned the lowest priority.

Table 4.2 summarizes the worst-case response times of all tasks of the example task set. Since for all tasks the WCRTs are not greater than the respective deadline, the task set is schedulable by all analyses.

4.7 Priority assignment

Deadline monotonic priority assignment is not optimal for AMC [6], and therefore it is not optimal for AMMC either; this applies to both AMMC-rtb and AMMC-max. (The same is true also for SMC and therefore SMMC.) However, Audsley's priority assignment algorithm [114] is optimal for AMC (and SMC) and, as we will proceed to show, it is also optimal for their multi-frame extensions, AMMC, both AMMC-rtb and AMMC-max, and SMMC. Furthermore, this is true also for their arbitrary-deadline variants.

Indeed, it has been shown [31] that Audsley's priority assignment is optimal for a given schedulability test S, if the following 3 conditions hold:

Condition 1 The schedulability of a task τ may, according to test S, depends on any independent properties of tasks with higher priority than τ, but not on any properties of those tasks that depend on their relative priority ordering.

Condition 2 The schedulability of a task τ may, according to test S, depends on any independent properties of tasks with lower priority than τ, but not on any properties of those tasks that depend on their relative priority ordering.

Condition 3 When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to test S, if it was previously schedulable at the lower priority.

where by *independent property* we mean a property that is independent of the priority assigned to a task. (Actually, the wording we are using for these conditions is by Davis and Burns [115].)

By analyzing the different worst-case response time analysis algorithms for multiframe tasks, it is straightforward to check that these conditions hold. We provide the arguments for the AMMC-max analysis, but they apply to the other multiframe mixed-criticality analyses, independently of whether the deadlines are constrained or arbitrary.

Our schedulability test is based on response-time analysis. In AMMC-max, for each task τ_i, we check whether its worst-case response time in L-mode is not greater than its (relative) deadline. Furthermore, if τ_i is an H-task, we check if its worst-case response time in H-mode, including the case of mode switch, is not greater than its deadline. These checks can be applied to each task independently of the relative priorities assigned to the tasks. In addition, they depend only on τ_i's deadline and on the response time. The deadline is a fixed attribute of a task and independent of the priority assigned to a task. Also, the worst-case response times of a task in both L- and H-mode do not depend on the relative ordering of higher-priority tasks or on the relative ordering of lower-priority tasks.

Indeed, in AMMC-max, the L-mode response time of task τ_i is computed with (4.15) (which is also valid for SMMC and AMMC-rtb), that we repeat here for convenience:

$$R_i^L = g^L(\tau_i, 1) + \sum_{\tau_j \in hp(i)} G^L(\tau_j, R_i^L)$$

We observe that $g^L(\tau_i, 1)$, which is given by (4.11), depends only on the L-WCETs of the jobs in τ_i's superframe, and therefore does not depend on the relative priority ordering of the tasks. Likewise, $G^L(\tau_j, R_i^L)$, which is given

by (4.13), depends on the L-WCETs of the jobs in τ_j's superframe, τ_j's period and R_i^L. Again, τ_j's attributes used in G^L function do not depend on the relative priority ordering of the tasks. R_i^L is also not dependent on the relative ordering among tasks with priority higher than τ_i or among tasks with priority lower than τ_i. Indeed, R_i^L is obtained by summing $g^L(\tau_i,1)$ and $G^L(\tau_j,R_i^L)$ for each task τ_j with a priority higher than τ_i, and therefore does not depend on the relative ordering of higher-priority tasks, or even on lower-priority tasks.

With respect to Condition 3, it is straightforward to check that, by swapping the priorities of any two tasks of adjacent priorities, the second term of the right-hand side of the schedulability test, (4.15), for the task being assigned the higher priority cannot be larger than what it was in the previous priority assignment. Therefore, if that task was previously schedulable at the lower priority, it will remain schedulable at the higher priority.

In AMMC-max, the response time of H-task τ_i in H-mode, including mode switch, is computed with (4.26), which we repeat here for convenience:

$$R_i^* = max\{R_i^*(s) : 0 < s \leq R_i^L\}$$

where s is the mode switch instant relative to the release instant of τ_i. As mentioned, the analysis needs to be performed only for values of s corresponding to the release of higher-priority L-tasks. These instants are independent of the relative priority order among higher-priority tasks or of the relative priority order among lower-priority tasks. $R_i^*(s)$ is computed with (4.27), which we repeat here for convenience:

$$R_i^*(s) = g^H(\tau_i,1) + \sum_{j \in hpL(i)} G^{L+}(\tau_j,s) + \sum_{k \in hpH(i)} g^*(\tau_k, \ell^L(k,R_i^*(s),s), \ell^H(k,R_i^*(s),s))$$

We observe that $g^H(\tau_i,1)$ is similar to $g^L(\tau_i,1)$, the only difference is that it uses the H-mode WCET of the jobs in a superframe instead of their L-mode WCET. Therefore it is also independent of the relative priority ordering among the other tasks. Following a line of argumentation similar to the one we presented above for $G^L(\tau_j,R_i^L)$, and taking into account that the values of s do not depend on the relative ordering of lower-priority tasks or even on higher-priority tasks, we conclude that $G^{L+}(\tau_j,s)$ is also independent of the relative priority ordering among higher-priority tasks or among lower-priority tasks. Finally, $g^*(\tau_k,\ell^L,\ell^H)$, which is given by (4.21), depends on $\ell^L(k,R_i^*(s),s)$ and $\ell^H(k,R_i^*(s),s)$ and also on attributes of H-task τ_k, i.e. its period and also the L- and H-WCET of the jobs in its superframe. These attributes of H-task τ_k are independent of the priorities of the other tasks. The values $\ell^L(k,R_i^*(s),s)$ and $\ell^H(k,R_i^*(s),s)$ are given by (4.24) and (4.25), respectively, and depend only on the period and the deadline of task τ_k, in addition to s and $R_i^*(s)$, which are also independent of the relative ordering among tasks with priority higher than τ_i, or among tasks with priority lower than τ_i. Indeed, $R_i^*(s)$ is obtained by summing $g^H(\tau_i,1)$, $G^{L+}(\tau_j,s)$ for each L-task τ_j with higher priority than τ_i, and $g^*(k,\ell^L(k,R_i^*(s),s),\ell^H(k,R_i^*(s),s))$ for each H-task τ_k with higher priority than τ_i, and therefore does not depend on the relative ordering of higher-priority tasks or on the relative ordering of lower-priority tasks.

With respect to Condition 3, we need to consider two cases, depending on the criticality of the task whose priority is lower after the priority swap. If the latter is an L-task, then the argument is exactly the same as presented above, for L-mode. If the task whose priority is lower after the priority swap is a H-task, then the third term of the right-hand side of the schedulabity test, (4.27), for the task being assigned the higher priority cannot be larger than what it was in the previous priority assignment. Therefore, if that task was previously schedulable at the lower priority, it will remain schedulable at the higher priority. This concludes our argument

that our schedulability test for constrained-deadline task-sets based on AMMC-max satisfies the 3 conditions stated above.

Finally, we note that the following theorem, which holds for mixed-criticality analyses, SMC and both AMC variants, with constrained deadlines [6] is also valid for the multiframe mixed-criticality analyses, SMMC and both AMMC variants, with constrained deadlines:

Theorem 1. *An optimal priority ordering exists that has all tasks with the same criticality assigned priorities in deadline monotonic priority order.*

The proof follows the standard argument of the optimality of deadline monotonic priority assignment for tasks with constrained deadlines. The idea is that if there is some schedulable assignment such that the priority assigned to two tasks is not according to their deadlines, then the assignment obtained by swapping the priorities of these two tasks is also schedulable.

Consider two tasks τ_i and τ_j with the same criticality level such that $D_i < D_j$. Assume that there is a priority assignment such that the priority of τ_j is higher than that of τ_i, which is deemed schedulable. If we swap the priority of τ_i and τ_j, obviously τ_i's response time under this new priority assignment will be shorter than under the original priority assignment, and therefore τ_i remains schedulable. Consider now τ_j, whose priority is lower in the new priority assignment. The cumulative processing demand by the completion of τ_j's job under the new priority assignment is equal to the cumulative processing demand by the completion of τ_i under the initial priority assignment. Indeed, because τ_i and τ_j are both constrained deadline tasks, there is only one job of each of these tasks in both schedules. Therefore, if τ_i, which has an earlier deadline than τ_j, was deemed schedulable under the initial priority assignment, so will τ_j be in the new priority assignment.

This theorem allows us to use a variant of Audsley optimal assignment [6] that in each step considers at most 2 tasks: the H- and L- tasks not yet assigned with largest deadline, thus reducing the worst-case number of tests required to $2n-1$, rather than $n(n+1)/2$ in the general case.

4.8 Evaluation

4.8.1 Experimental setup

A brief description of the experimental setup and platform used in this thesis for evaluation is already discussed in Section 3.4. For more details, we refer the reader to Section 3.4, as in this section we only discuss about taskset generation and results obtained using the schedulability analyses discussed in this chapter. The generation of the synthetic task sets is controlled through the following parameters:

- Task periods are generated in the range of 10 msec to 1 sec using a log-uniform distribution.
- The deadlines of the tasks are generated within a range of $[0.25, 4]T_i$ using a log-uniform distribution [113]. The analyses presented in Sections 4.2 and 4.3 hold for only constrained-deadlines, so their deadlines are selected with log-uniform distribution within a range of $[0.25, 1]T_i$.

- The UUnifast algorithm [116, 111] is used to generate the L-mode utilization ($U^L_{i,1}$) for the first frame of a task τ_i in an unbiased way. The L-WCET of the first frame[2] of each task is then $C^L_{i,1} = T_i \times U^L_{i,1}$. The number of frames for each task is selected randomly with a uniform distribution within a range of $[1, \alpha]$, where $\alpha > 1$ is a user-defined integer parameter. The L-WCETs of other frames of a task are randomly selected within an interval of $[\beta \times C^L_{i,1}, C^L_{i,1}]$ with uniform distribution, where $\beta \in (0, 1]$ is a task generation parameter limiting the L-WCET variation among a task's frames
- The user-defined fraction of H-tasks $\xi \in (0, 1)$ in the task set.
- The H-WCET of the j^{th} frame of H-task τ_i ($C^H_{i,j}$) is derived by linearly scaling up that frame's L-WCET ($C^L_{i,j}$) with a user-defined factor of κ, i.e., $C^H_{i,j} = \kappa \times C^L_{i,j}$.

We used Audsley's optimal priority assignment [31] for all schedulability tests. The target utilization is varied within a range of $(0, 1]$ with a step size of 0.1. Different random class objects are defined for utilization, minimum inter-arrival time, number of frames, deadline and L-WCET of each frame. These random class objects are seeded with different odd numbers and reused in successive replications [117]. For each set of input parameters, 1000 random task sets are generated. The parameters used in this evaluation are summarized in Table 4.3.

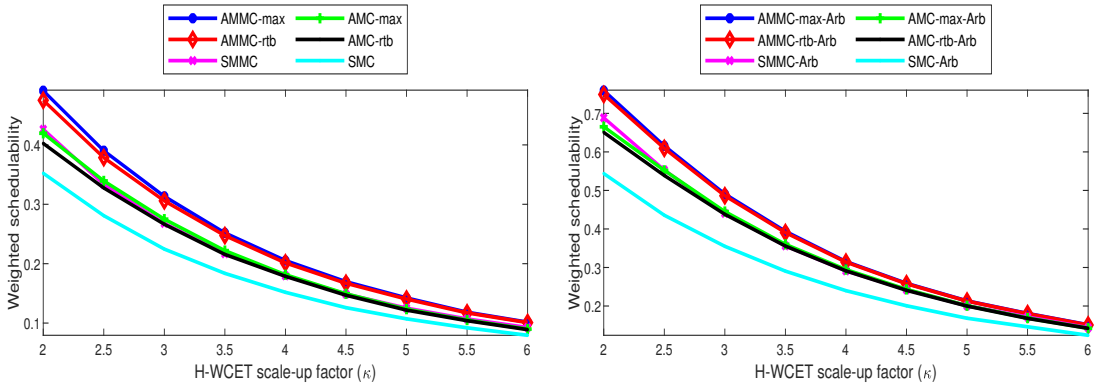
Table 4.3: Overview of Parameters (The triples in the values column are respectively the minimum value, the step size and the maximum value for the respective parameter.)

Parameters	Values	Default
H-WCET scale-up factor (ξ)	$\{2 : 0.5 : 6\}$	3
Task set size ($\lvert\tau\rvert$)	$\{8 : 4 : 32\}$	16
Fraction of H-tasks in τ (ζ)	$\{0.2 : 0.05 : 0.7\}$	0.4
Upper bound on # of frames ($\lvert\alpha\rvert$)	$\{3 : 1 : 10\}$	5
Lower bound on L-WCET (β) variation	$\{0.1 : 0.1 : 0.8\}$	0.2
Inter-arrival time (T_i)	$10ms$ to $1s$	N/A
Deadline (D_i)	$LogUnif\{[0.25 \quad 4]T_i\}$	N/A

To avoid generating a huge number of plots, in each of our experiments, we only vary one parameter at a time, with the other parameters conforming to their default values, displayed in Table 4.3. In our experimental setup we used weighted schedulability as the main evaluation criteria. The weighted schedulability representation condenses three dimensional-plots to two-dimensional plots [118, 66] by eliminating the axis of task set utilization. This performance metric gives more weight to task sets with higher utilization. Using notation from [66], let $S_y(\tau, p)$ represent the result (0 or 1) of the schedulability test y for a given task set τ with an input parameter p. Then $W_y(p)$, the weighted schedulability for that test y as a function p, is presented in (4.48), where $U^L(\tau)$ is the nominal system utilization of the task set τ in the L-mode.

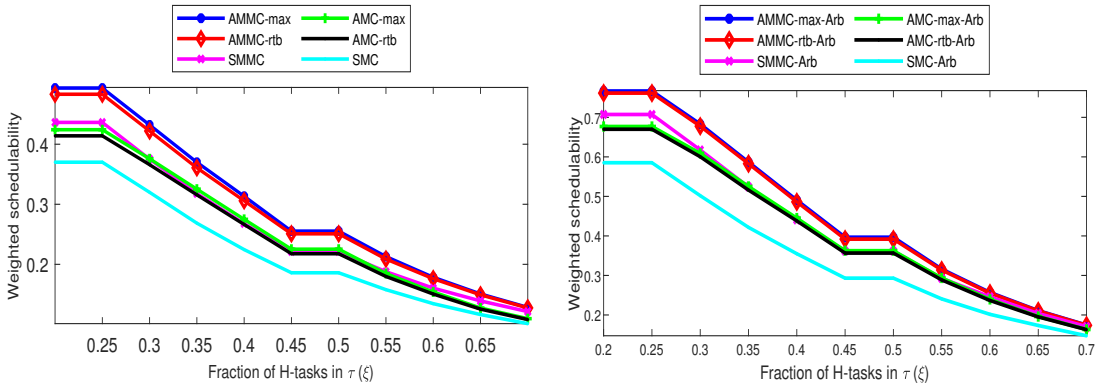
$$W_y(p) = \frac{\sum_{\forall\tau}(U^L(\tau) \cdot S_y(\tau, p))}{\sum_{\forall\tau} U^L(\tau)} \tag{4.48}$$

[2]For convenience, without loss of generality (since shift-rotating the order of the frames results in an equivalent multiframe task), the first frame has the biggest L-WCET. Please note that our analyses make no assumptions regarding the WCET of the first frame being the greatest.



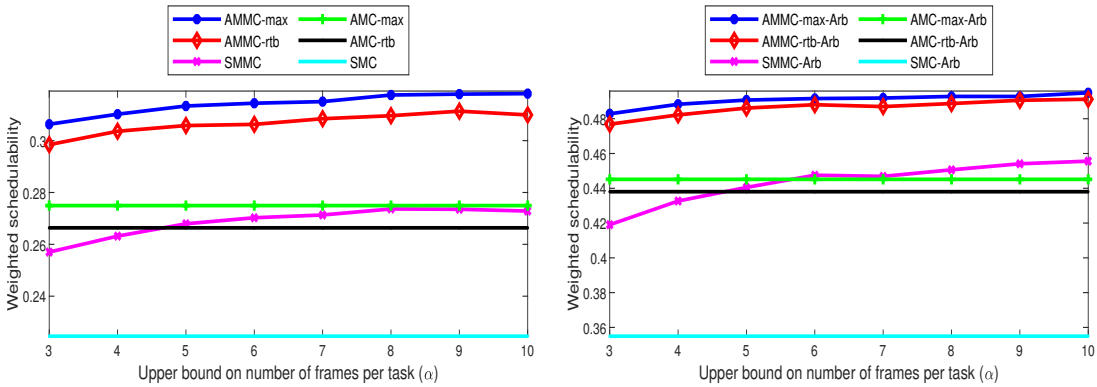
(a) Constrained-deadline analyses (b) Arbitrary-deadline analyses

Figure 4.3: Effect of H-WCET variation



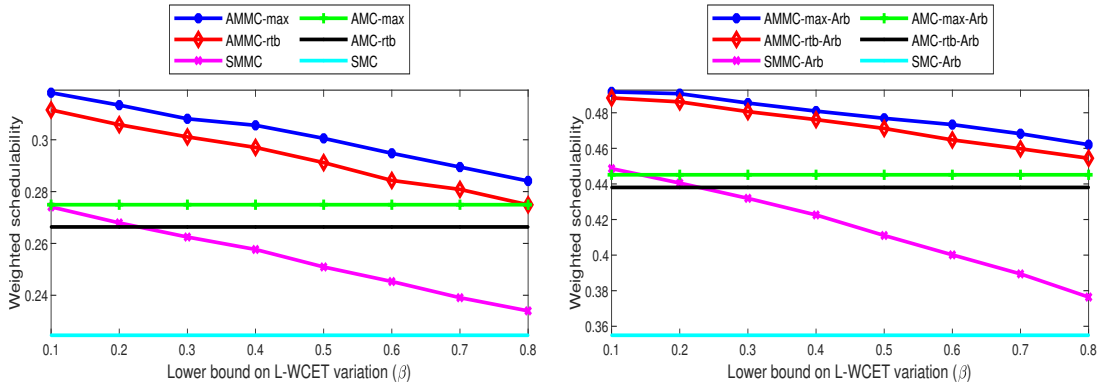
(a) Constrained-deadline analyses (b) Arbitrary-deadline analyses

Figure 4.4: Weighted Schedulability vs. H-task share



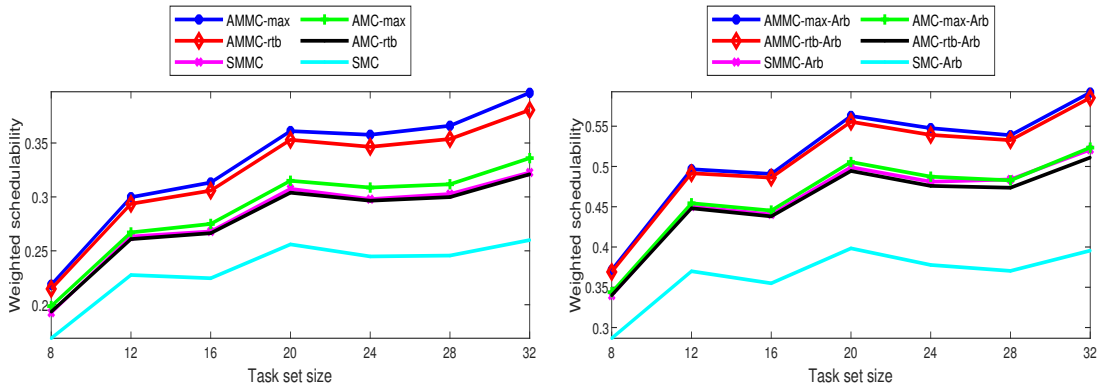
(a) Constrained-deadline analyses (b) Arbitrary-deadline analyses

Figure 4.5: Weighted schedulability vs. max number of frames in a task



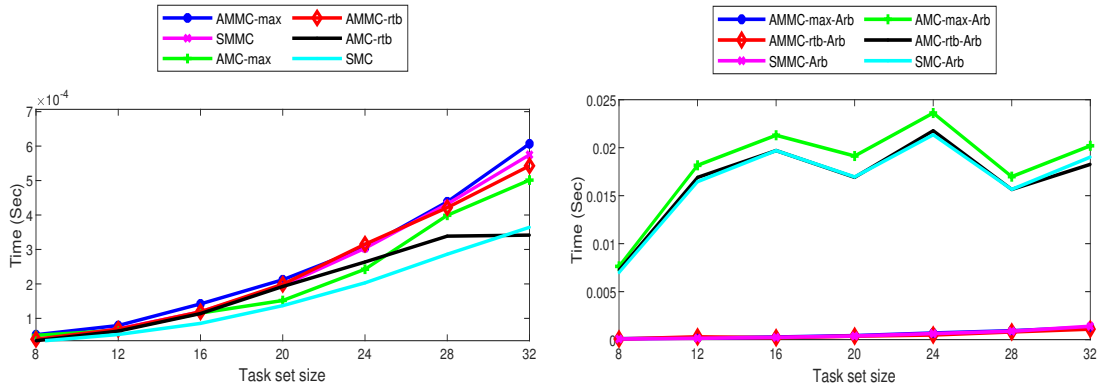
(a) Constrained-deadline analyses (b) Arbitrary-deadline analyses

Figure 4.6: Effect of L-WCET variation



(a) Constrained-deadline analyses (b) Arbitrary-deadline analyses

Figure 4.7: Effect of task set size variation over weighted schedulabaility



(a) Constrained-deadline analyses (b) Arbitrary-deadline analyses

Figure 4.8: Impact of task set size variation on the average running time

4.8.2 Results

We compared the weighted schedulability of the six proposed schedulability analyses (SMMC, AMMC-rtb, AMMC-max, SMMC-Arb, AMMC-rtb-Arb, AMMC-max-Arb) against the existing ones (SMC, AMC-rtb,

AMC-max, SMC-Arb, AMC-rtb-Arb and AMC-max-Arb). For the SMC, AMC-rtb, AMC-max, SMC-Arb, AMC-rtb-Arb and AMC-max-Arb tests, the multiframe task is transformed into an instance of the classic (i.e., single-frame) mixed-criticality task-model by pessimistically discarding frame information. In this transformation, for any task τ_i, its L-WCET and H-WCET are set to $C_i^L = \max_{j=0}^{F_i-1} C_{i,j}^L$ and $C_i^H = \max_{j=0}^{F_i-1} C_{i,j}^H$, respectively.

Please note that the implicit-deadline model ($D_i = T_i$) was used in the evaluation of the constrained-deadline schedulability analyses presented in our previous work [13]. The use of the constrained deadlines in this chapter for the algorithms presented in Sections 4.2 and 4.3 (constrained-deadline schedulability analyses) has led to a slight decrease in their weighted scheduling success ratio when compared to the results presented in [13].

Figures 4.3a and 4.3b present the effect of varying the H-WCET scale-up factor (κ) on all analyses. A higher value for κ increases the H-mode processor requirement of high-criticality tasks, hence the weighted schedulability of all analyses decreases accordingly. The multiframe-based analyses perform better than their corresponding frame-agnostic conventional analyses by leveraging frame information. As expected, AMC-max outperforms AMC-rtb and AMMC-max outperforms AMMC-rtb, albeit by very small margins in both cases. The difference between adaptive and static variants comes from the fact that static analyses do not drop L-tasks, this makes the WCRT of H-tasks computed with SMC more likely to exceed their deadlines. SMMC performs better at low values of κ compared to SMC, AMC-rtb and AMC-max, but the difference among them decreases with an increase in κ, because task sets become harder to schedule. The observations mentioned for constrained-deadline analyses are also true for the arbitrary-deadline analyses (see Figure 4.3b). The constrained-deadline analyses have more processing requirement when compared to arbitrary-deadline analyses as $D_i \leq T_i$. Hence, the arbitrary-deadline analyses when compared to their corresponding constrained-deadline analyses show better weighted schedulability ratio. Please note that the difference between the proposed arbitrary-deadline frame-based analyses and their corresponding single-frame based analyses is lower when compared to a similar comparison in constrained-deadline analyses. A higher κ reduces the overall schedulability ratio for each analysis, therefore, the differences in performance between the multiframe- and single-frame-based analyses also decrease.

A higher fraction of H-tasks (ξ) increases the processor requirement of the system in H-mode. Hence, the weighted schedulability of all analyses decreases (Figures 4.4a and 4.4b). Their absolute difference in terms of weighted schedulability also decreases, as the number of feasible task sets decreases with a higher fraction of H-tasks.

The potential for improvements in weighted schedulability for SMMC, AMMC-rtb, AMMC-max, SMMC-Arb, AMMC-rtb-Arb and AMMC-max-Arb over single-frame-based analyses increases when the number of frames per task is higher. A larger value of upper bound for the number of frames per tasks results in more frames per task on average as well. With more frames in a task, the probability of having low-WCET frames increases, which in turn magnifies the improvement from leveraging the frame information (or, equivalently, magnifies the pessimism when disregarding it). Hence, the weighted schedulability of the multiframe-based analyses improves when the upper bound on the number of frames per task is higher, as shown in Figures 4.6a and 4.5b. The classic SMC, AMC-rtb, AMC-max, SMC-rtb, AMC-rtb-Arb and AMC-max-Arb are insensitive to this parameter, as these analyses assume the maximum estimates for L-WCET and H-WCET over all frames. For low α, leveraging frame information does not compensate for the pessimism of static multiframe analyses (SMMC and SMMC-Arb), causing it to under perform compared to conventional single-frame adaptive analyses (AMC-rtb, AMC-max, AMC-rtb-Arb and AMC-max-Arb).

The WCET variation limiting parameter β defines a lower bound for the ratio between the smallest frame L-WCET and the greatest frame WCET of a given task. A higher value of β decreases this range, and consequently, increases the average execution requirement of task's frames, all other things remaining equal. Hence, the weighted schedulability of the multiframe-based analyses decreases with an increase in β (see Figures 4.6a and 4.6b). One important observation is that the absolute difference among the weighted schedulability of AMMC-max, AMMC-rtb and SMMC increases with an increase in β. The pessimism in AMMC-rtb against AMMC-max becomes a major differentiator when the potential of gaining from the multiframe properties becomes limited. The same behavior is shown by SMMC against SMC. These effects also holds true for arbitrary-deadline analyses (see Figure 4.6b). Similarly to the bound on the maximum number of frames per task, β has no effect on SMC, AMC-rtb, AMC-max, SMC-Arb, AMC-rtb-Arb and AMC-max-Arb, as the maximum of the WCET estimates over all frames are assumed in each mode.

The effect of variation in the number of tasks on the weighted schedulability is presented in Figures 4.7a and 4.7b. The important observation is that, similarly to single-criticality systems, the weighted schedulability improves with larger task set sizes, as more low-utilization tasks are easier to schedule compared to fewer high-utilization tasks. The fact that during task generation, the number of H-tasks is rounded up to the nearest integer as needed (i.e., when the target fraction of H-tasks would result in a non-integer number) explains the saw-tooth shape of the weighted schedulability plots of each analysis.

Table 4.4: Maximum absolute difference in schedulability success ratio of each multiframe analysis from its corresponding single-frame analysis.

	Varied parameters				
Analyses	κ	$\lvert\tau\rvert$	ξ	α	β
AMMC-max	**14.9%**	14.3%	14.6%	9.2%	8.6%
AMMC-rtb	**16.6%**	15.7%	15.1%	10.8%	10.5%
SMMC	19.3%	**20.0%**	15.6%	13.5%	14.2%
AMMC-max-Arb	29.1%	12.2%	**31.4%**	8.2%	8.2%
AMMC-rtb-Arb	28.1%	11.4%	**30.9%**	8.1%	8.1%
SMMC-Arb	28.4%	**29.6%**	26.3%	18.8%	18.0%

To quantify the benefits in terms of *non-weighted* schedulability success ratio, Table 4.4 shows the maximum absolute difference in non-weighted schedulability success ratio of each multiframe analysis over its corresponding single-frame analysis. In the best case, the AMMC-max, AMMC-rtb, SMMC, AMMC-max-Arb, AMMC-rtb-Arb and SMMC-Arb analyses achieve up to 14.9%, 16.6%, 20%, 31.4%, 30.9% and 29.6% higher schedulability successes ratio, respectively, compared to their corresponding single-frame analyses. The bold values in Table 4.4 shows the highest difference achieved by each analysis among all experiments.

Finally, we experimentally explore the running time of each analysis. The description about the platform used for the experiment is already described in the Section 3.4 of Chapter 3. Figure 4.8a presents the average running times of all constrained-deadline analyses for different task set sizes in seconds. AMMC-max takes longer than other analyses as it requires more computation while computing the processor requirement in each iteration. As expected, the running times of all tests increase with an increase in task set size as the feasibility testing has to be performed for each task. Figure 4.8b presents the average running time of the arbitrary-deadline task analyses. The frame-oblivious analyses are pessimistic and the length of the busy interval in the feasibility

Table 4.5: Average running time (in seconds) of all constrained-deadline analyses with default parameters.

AMMC-max	AMMC-rtb	SMMC	AMC-max	AMC-rtb	SMC
2.6233×10^{-4}	2.4362×10^{-4}	2.4725×10^{-4}	2.1804×10^{-4}	1.9268×10^{-4}	1.6601×10^{-4}

Table 4.6: Average running time (in seconds) of all arbitrary-deadline analysis with default parameters.

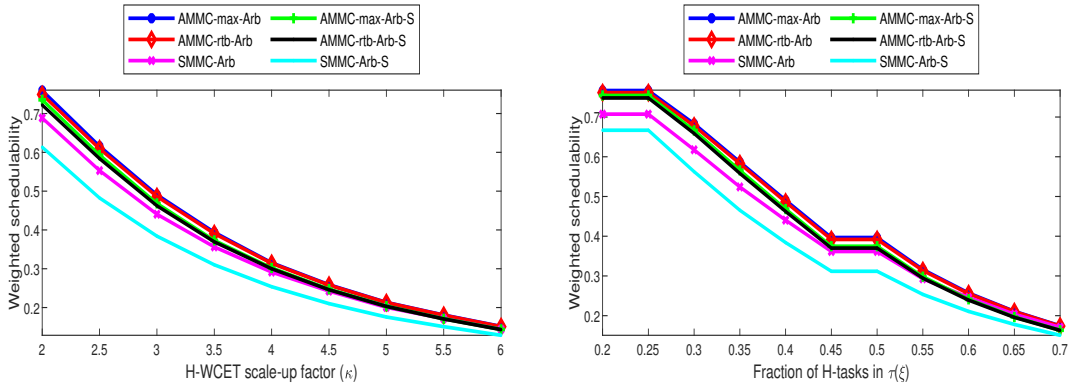
AMMC-max-Arb	AMMC-rtb-Arb	SMMC-Arb	AMC-max-Arb	AMC-rtb-Arb	SMC-Arb
5.49×10^{-4}	4.70×10^{-4}	5.30×10^{-4}	1.8144×10^{-2}	1.6632×10^{-2}	1.6589×10^{-2}

testing of such analyses becomes much longer when compared to multiframe analyses. Hence, in Figure 4.8b (contrary to Figure 4.8a), the frame-oblivious analyses take more time than multiframe analyses. For the default parameters, the average running times of all analyses are also presented in Tables 4.5 and 4.6. The observations presented above for Figures 4.8a and 4.8b are consistent with the data shown in Tables 4.5 and 4.6.

4.8.3 Case study

In safety-related applications, the computation of H-WCET and its certification for each frame is expensive and time consuming. This case study explores a scenario (also referred to as a special case) with multiple L-WCET estimates (multiple frames in L-mode) and a single H-WCET estimate (single frame in H-mode), and analyzes its scheduling performance degradation over the generic multiframe setting. In our results, we append "-S", from single (H-frame), as suffix to the acronym of each multiframe analysis to differentiate this special case from generic multiframe analyses. Only multiframe arbitrary-deadline task schedulability analyses are compared in this case study. In our experimental setup, the single H-WCET estimate of any H-task in the special case is computed to be $C_i^H = \kappa \times \max_{j=0}^{F_i} C_{i,j}^L$.

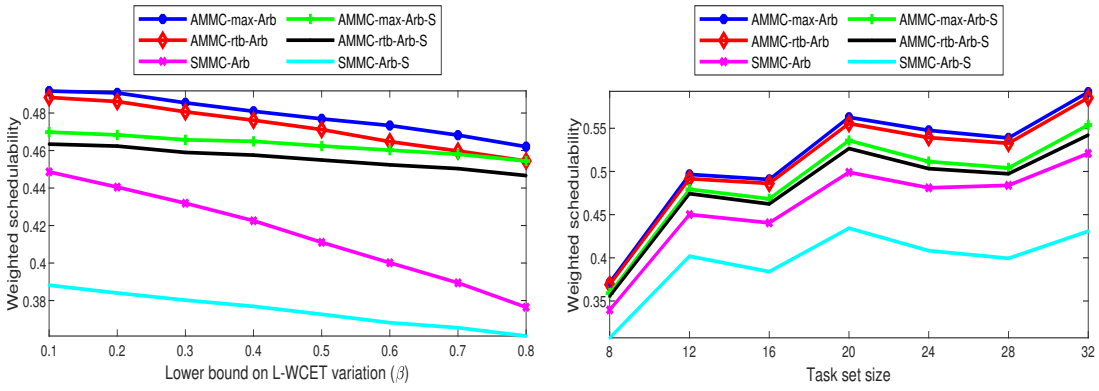
Figures 4.9a to 4.11 compare the weighted schedulability of the generic multiframe analyses with those of the special case by varying different parameters. As expected, the special case analyses lead to a lower weighted schedulability than the respective generic analyses. Indeed, because we set C_i^H to the product of the scale-up factor, κ, by the largest WCET of all L-frames of task τ_i, the execution demand of every task in the special case analysis is never lower than in the generic analysis, and is almost always larger.



(a) (b)

Figure 4.9: Weighted schedulability of different parameter variation a) H-WCET variation b) H-task share - case study

Independently of the factor of the experiment, the reduction in weighted schedulability is larger for SMMC-Arb-S than for the two AMMC variants. Furthermore, the reduction in weighed schedulability for AMMC-rtb-Arb-S is approximately equal to that for AMMC-max-Arb-S. These observations can be justified, a posteriori, by the differences in these analyses. Indeed, in SMMC-Arb-S, there is only one single mode of operation, which is affected by the increase in the WCET of the frames of H-tasks, whereas in the AMMC variants, the execution demand in L-mode is not affected, only the execution demand in H-mode and upon mode switch.



(a) (b)

Figure 4.10: Weighted schedulability of different parameter variation a) Effect of tskset set size b) Lower bound on L-WCET variation - case study

Another interesting observation, is that the effect of each factor on the weighted schedulability of a special case analysis is similar to that observed on the weighted schedulability of the respective generic analysis. Again this was somewhat expected since we can view each special case analysis as just the generic analysis of a task set with slightly different parameters, namely in terms of the WCET of H-tasks: in the case of the AMMC variants only the H-WCET of the H-tasks is different. Thus the observed effects of each factor on the weighted

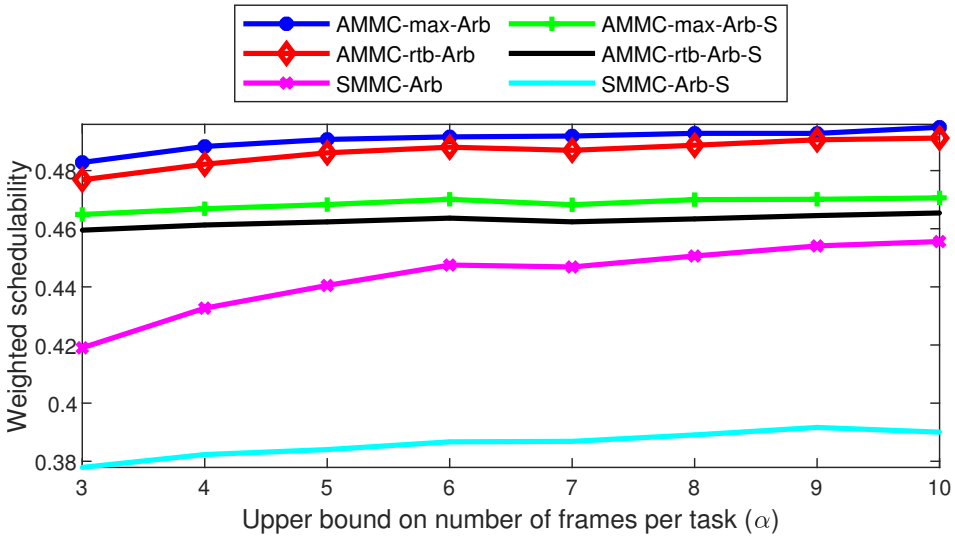


Figure 4.11: Weighted schedulability vs. max. number of frames in a task – case study.

Table 4.7: Maximum absolute reduction in non-weighted schedulability ratio of each special case analysis compared with the respective generic analysis.

	Varied parameters				
Analyses	κ	$\lvert\tau\rvert$	ξ	α	β
AMMC-max-Arb-S	5.3%	6.4%	**6.9**%	4.8%	4.3%
AMMC-rtb-Arb-S	5.2%	7.7%	**7.8**%	4.8%	4.3%
SMMC-Arb-S	15.2%	**21.1**%	14.6%	13.2%	12.3%

schedulability of each special case analysis can be justified in the same way as we have done in Section 4.8.2 for the respective generic analysis.

Finally, because weighted schedulability is biased towards task sets with higher utilization, Table 4.7 summarizes the results of this case study in terms of (non-weighted) schedulability ratio. The maximum absolute reduction was found to be 21.1%, 7.8% and 6.9% for SMMC-arb-S, AMMC-rtb-S and AMMC-max-S, respectively, when compared to the respective generic analyses.

4.9 Chapter summary

In this chapter, and in the context of uniprocessor fixed-priority-scheduled platforms, we extended the mode-based mixed-criticality model of Vestal, which assumed a single worst-case execution time per task per mode. This extension accommodates multiframe tasks that have different WCETs for their jobs, in a repeating pattern. For this extended model, we formulated schedulability analyses (SMMC, AMMC-rtb and AMMC-max) that leverage the frame information, resulting in greater accuracy over the state-of-the-art mixed-criticality schedulability tests (SMC, AMC-rtb and AMC-max) that, pessimistically, have to discard frame information and use a single WCET per task per mode, in order to be applied. These tests, formulated for constrained-deadline systems, are generalized for arbitrary-deadline tasks (as SMMC, AMMC-rtb and AMMC-max), drawing from the

existing results on arbitrary-deadline mixed-criticality (but single-frame) scheduling. Finally, we reason about task priority assignment for the scheduling problem considered, when the new analyses are used for schedulability testing, and prove the optimality of Audsley's algorithm for the general case, and of a faster variant thereof, in the case of constrained deadline systems. Experimental evaluation with synthetic task sets confirms the benefits (up to 31.4% in schedulability success ratio) of the new model and its analysis, compared to schedulability tests that disregard frame information. In the next chapter, we consider a partitioned multicore arrangement and incorporate the effects of memory stalls under memory access regulation into the schedulability analysis to make it more realistic and more applicable to real-world systems.

Chapter 5

Memory bandwidth regulated multiframe mixed criticality systems

In this chapter, we extend the shared-resource-oblivious nalysis techniques for multiframe mixed-criticality tasks presented previously, in Chapter 4, such that they become safe to use for commercial-off-the-shelf (COTS) multicore platforms with shared memory controllers. In particular, we add to the analysis techniques provisions for memory access regulation via periodically replenishable per-core memory access budgets, and account for associated stalls.

5.1 Introduction

We extended all of the schedulability analyses presented in Chapter 4, in accordance to the system model presented earlier in Section 3.2 by incorporating *memory access regulation* and *stall analysis*. Recall that, under that system model, all memory accesses go through a shared memory controller, and each core has a memory access budget, to consume within a given regulation period. Any core exceeding its budget is stalled until its replenishment, at the start of the next regulation period. Consequently, the challenge is to upper-bound the memory stalls, due to contention among cores and due to the memory regulation itself, and incorporate them into the analysis. The chapter is organized as follows:

Section 5.2 discusses relevant existing results on the stall analysis of memory-access regulated multicore systems (whereas multiframe mixed-criticality systems were already discussed in Chapter 4). In Section 5.3 we present a motivational example of a task set, deemed schedulable by the stall-oblivious analysis formulated in Chapter 4, that may nevertheless be unschedulable when running in a system with memory regulation. Section 5.4 then presents new stall-aware WCRT analysis for memory-bandwidth-regulated multiframe mixed-criticality systems. Sections 5.5 and 5.6 offer optimisations to the accuracy and the running time of that analysis, respectively. Section 5.7 experimentally evaluates the new analyses, in terms of weighted schedulability and running time, compared to the frame-agnostic analyses. Section 5.8 concludes the chapter.

5.2 Background

To quantify the interference from the sharing of memory and interconnect among cores, we build upon the schedulability analysis of Yao et al. [55], which provides stall analysis for memory-regulated multicore single-criticality systems. That stall analysis assumes that the task under analysis is the only task on its core. Therefore, an active task is either executing, accessing memory or stalled.

With memory regulation, a task may experience two kinds of stalls: 1) *contention stall*, arising from the sharing of the memory and the interconnect among the cores; 2) *regulation stall*, occuring when a core exhausts its budget. The worst-case stall experienced by a task depends essentially on two values: the fraction of the total memory bandwidth assigned to the core running the task, $b=\frac{Q}{P}$, where Q represents number of accesses a core can make in the regulation period P. The fraction of the task's total execution time that is spent accessing memory is represented as, $r=\frac{C^m}{C}$, where $C = C^m + C^e$. Indeed, if $b \leq \frac{1}{K}$, where K is the number of cores, the worst-case stall occurs by maximizing the number of regulation stalls. Otherwise, the worst case occurs by maximizing the number of contention stalls. If there is enough computation, then all memory accesses can suffer the maximum contention stalls. However, if there is not enough computation, some regulation periods may experience regulation stalls. Thus, Yao's analysis yields 3 cases:
Case 1: If $b \leq \frac{1}{K}$, then the stall is maximum when the number of regulation periods with a regulation stall is maximum, and it can be upper-bounded by:

$$stall(C^e, C^m) = \begin{cases} \frac{C^m}{Q}(P-Q) + (K-1)Q & \text{if } C^m \bmod Q \ = \ 0 \\ \left\lceil \frac{C^m}{Q} \right\rceil (P-Q) + (K-1)(C^m \bmod Q) & \text{otherwise} \end{cases} \tag{5.1}$$

Case 2: If $b > \frac{1}{K}$ and $r = \frac{C^m}{C} < \frac{1-b}{(K-1)b}$, then the stall is maximum when every memory access suffers interference by all other cores and it can be bounded by:

$$stall(C^e, C^m) = (P-Q) + (K-1) \cdot C^m \tag{5.2}$$

Case 3: If $b > \frac{1}{K}$ and $r = \frac{C^m}{C} \geq \frac{1-b}{(K-1)b}$, then stall is bounded by

$$stall(C^e, C^m) = \begin{cases} (1+A)(P-Q) + r_1 & if\ C \leq (1+A)Q \\ \left(1 + \frac{C}{Q}\right)(P-Q) + r_2 & \text{otherwise} \end{cases} \tag{5.3}$$

$$\text{where, } A = \left\lfloor \frac{C^e}{Q - RBS} \right\rfloor,\ RBS = \frac{P-Q}{K-1},$$

$$r_1 = \min\{P-Q, (K-1)(C^m - A \cdot RBS)\}, \tag{5.4}$$

$$r_2 = \min\{P-Q, (K-1)(C\, mod\, Q)\}$$

All three cases consider an initial (regulation) stall of $(P-Q)$ because, in the worst case, when a task is scheduled to run on a core, the latter's memory access budget is already depleted.

To apply the stall to the WCRT analysis of preemptive fixed-priority scheduling, Yao et al. [55] uses the

Table 5.1: Example task set and WCRT computed by stall-oblivious AMMC-max analysis.

Task	κ	$\vec{C}_i^L$	$\vec{C}_i^H$	$T = D$	WCRT	
					L-mode	Mode switch
τ_1	L	{(1,2),(2,2),(6,1),(4,3)}	{-}	20	7	NA
τ_2	H	{(3,2)(5,1)(2,1)}	{(6,4),(10,2),(4,2)}	30	13	19
τ_3	H	{(1,3),(2,1)}	{(2,6),(4,2)}	40	17	27

concept of a synthetic task with the following parameters:

$$\widehat{C_i^e(t)} = C_i^e + \sum_{j \in hp(\tau_i)} \left\lceil \frac{t}{T_j} \right\rceil C_j^e$$

$$\widehat{C_i^m(t)} = C_i^m + \sum_{j \in hp(\tau_i)} \left\lceil \frac{t}{T_j} \right\rceil C_j^m$$

I.e. it treats the job under analysis plus the jobs of interfering tasks arriving in the response time interval as a single job of a synthetic task. Thus, the classic WCRT equation becomes:

$$R_i = C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + stall(\widehat{C_i^e(R_i)}, \widehat{C_i^m(R_i)})$$

5.3 Example

We present an example of how the stall-oblivious analysis for AMMC-max presented in Section 4.3.2.2 of Chapter 4 may be unsafe in the presence of memory regulation.

Consider a quad-core system using memory regulation based on MemGuard, with a regulation period of 10 time units (worst-case memory access latency). Assume that cores P_1 to P_4 have memory budgets of 3, 3, 2 and 2, respectively.

P_1 runs three multiframe tasks whose characteristics are shown in Table 5.1. The two right-most columns in this table show the worst-case response times of all tasks computed by the stall-oblivious AMMC-max analysis (Section 4.3.2.2). Since no WCRT exceeds the respective task deadline, the stall-oblivious AMMC-max analysis deems the task set schedulable.

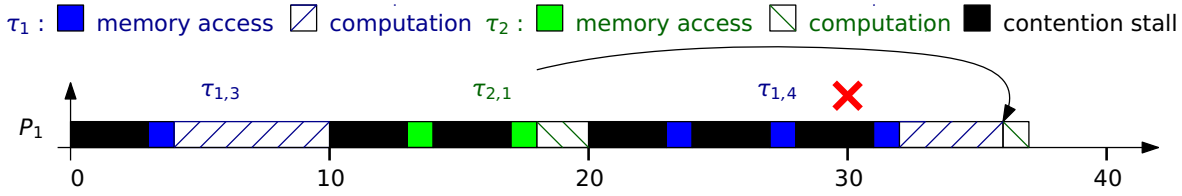


Figure 5.1: L-mode execution scenario with stalls for the task set in Table 5.1 showing that τ_2 misses its deadline, even though the AMMC-max stall-oblivious analysis deems it schedulable. Assuming a four core platform with a regulation period of 10. P_1's memory budget is 3.

However, this may not be the case as illustrated by the execution of P_1's tasks in Figure 5.1, where blue is used for τ_1, green for τ_2 and black for contention stalls. For each task, we use a solid pattern to represent memory accesses, and a diagonal line pattern for computation. In this example, we assume that the system is operating in L-mode and that τ_1's third frame and τ_2's first frame arrive at time 0, the beginning of a regulation period. (The $\tau_{i,j}$ labels above P_1's execution time line denote that P_1 is executing frame j of τ_i.) Assuming deadline monotonic scheduling, P_1's scheduler decides to run τ_1, which incurs a cache miss. Further assume that at the time P_1 issues a memory request, the other 3 cores also issue memory requests and that the memory controller, which executes a round-robin scheduler, determines that P_1's request is served only after the requests of the other 3 processors. Hence, τ_1 incurs a stall for 3 time units, after which it performs its memory access. Next, τ_1 performs computation for 6 time units, and completes at time 10. The scheduler now decides to run τ_2. Again, we assume that, before any computation, τ_2 suffers two last-level cache misses, both of which result in memory requests that incur 3-time-unit memory stalls. At time 18, when the second memory access completes, τ_2 starts performing computation. However, after 2 time units, at time 20, τ_1's fourth frame arrives, and the scheduler preempts τ_2, and assigns P_1 to τ_1. Again, τ_1 first performs 3 memory accesses, incurring a contention stall of 3 time units each. After the third memory access terminates, τ_1 executes for 4 time units completing at time 36. The scheduler then resumes τ_2; it executes for one time unit and completes at time 37. Since τ_2's relative deadline is 30 time units, in this execution τ_2 misses its deadline (represented by the red cross in Figure 5.1), thus showing that with memory regulation this task set is not schedulable.

5.4 Memory-aware WCRT analysis

We now extend the AMC-max based WCRT analysis for mixed-criticality multi-frame tasks [13], AMMC-max, presented in Section 4.3.2.2 of previous Chapter, to systems using memory regulation. Our approach is based on Yao's analysis [55], summarized in Section 5.2 and it is safe, simple and efficient, but somewhat pessimistic. Section 5.5 presents tighter analysis whose run-time is significantly larger.

The simplicity of the analysis arises essentially from our approach to upper bound the stall. This approach and the use of a single recurrence per task, see (4.27) and (4.26), lead to a rather efficient analysis.

As a first step, we extend g and G functions described in Section 4.3 of previous Chapter to compute the maximum cumulative computation time and the maximum cumulative memory access time required to compute the stall. More specifically, we define functions $g^{e|L}/g^{m/L}$ and $G^{e|L}/G^{m/L}$, based on g^L, (4.11), and G^L, (4.13), respectively. For example, we define:

$$g^{e|L}(\tau_i,k) = \begin{cases} 0 & \text{if } k = 0 \\ \max\limits_{0\le j<F_i} \sum\limits_{\ell=j}^{j+k-1} C^{e|L}_{i,(\ell \bmod F_i)} & \text{if } 1\le k\le F_i \\ q\cdot g^{e|L}(\tau_i,F_i) + g^{e|L}(\tau_i,r) & \text{otherwise} \\ \quad \text{where } q = k \text{ div } F_i \text{ and } r = k \bmod F_i & \end{cases} \tag{5.5}$$

Functions $g^{e|H}$ and $g^{m|H}$ are derived similarly from g^H and are presented as follows.

$$g^{e|H}(\tau_i,k) = \begin{cases} 0 & \text{if } k = 0 \\ \max\limits_{0 \le j < F_i} \sum\limits_{\ell=j}^{j+k-1} C^{e|H}_{i,(\ell \bmod F_i)} & \text{if } 1 \le k \le F_i \\ q \cdot g^{e|H}(\tau_i,F_i) + g^{e|H}(\tau_i,r) & \text{otherwise} \\ \quad \text{where } q = k \text{ div } F_i \text{ and } r = k \bmod F_i \end{cases} \tag{5.6}$$

$$g^{m|H}(\tau_i,k) = \begin{cases} 0 & \text{if } k = 0 \\ \max\limits_{0 \le j < F_i} \sum\limits_{\ell=j}^{j+k-1} C^{m|H}_{i,(\ell \bmod F_i)} & \text{if } 1 \le k \le F_i \\ q \cdot g^{m|H}(\tau_i,F_i) + g^{m|H}(\tau_i,r) & \text{otherwise} \\ \quad \text{where } q = k \text{ div } F_i \text{ and } r = k \bmod F_i \end{cases} \tag{5.7}$$

$$G^{e|L}(\tau_i,t) = g^{e|L}\left(\tau_i, \left\lceil \frac{t}{T_i} \right\rceil\right) \tag{5.8}$$

$$G^{m|L}(\tau_i,t) = g^{m|L}\left(\tau_i, \left\lceil \frac{t}{T_i} \right\rceil\right) \tag{5.9}$$

Likewise functions $G^{e|H}$ and $G^{m|H}$ are derived from G^H using H-mode execution and memory access times of task under consideration as follows:

$$G^{e|H}(\tau_i,t) = g^{e|H}\left(\tau_i, \left\lceil \frac{t}{T_i} \right\rceil\right) \tag{5.10}$$

$$G^{m|H}(\tau_i,t) = g^{m|H}\left(\tau_i, \left\lceil \frac{t}{T_i} \right\rceil\right) \tag{5.11}$$

Since we consider adaptive mixed-criticality with only two criticality levels, high (H) and low (L), the system operates in two modes. In L-mode all tasks execute, while just the H-tasks in H-mode. Hence, we must consider WCRT in L-mode, in (stationary) H-mode and in transient H-mode, i.e. when a task is caught by a mode transition. In all cases, we take into account the effect of memory regulation, by using the approach in [55] (summarized in Section 5.2), i.e. by adding a term that upper bounds the stall to the respective WCRT equation derived in [13]. For example, in L-mode, the WCRT becomes:

$$R_i^L = g^L(\tau_i,1) + \sum_{j \in hp(i)} G^L(\tau_j,R_i) + stall(\widehat{C_i^{e|L}(R_i^L)}, \widehat{C_i^{m|L}(R_i^L)}) \tag{5.12}$$

The *stall* term is computed using the algorithm in [55] and summarized in Section 5.2. The crux is what values to use for arguments $\widehat{C_i^{e|L}(R_i^L)}$ and $\widehat{C_i^{m|L}(R_i)}$. In the multiframe task model, the computation time and the memory access time of a sequence of jobs of a given length depends on the first job in that sequence. Furthermore, the sequence with the maximum computation time may be different from the sequence with the maximum memory access time. To ensure safety, $\widehat{C_i^{e|L}(R_i^L)}$ and $\widehat{C_i^{m|L}(R_i^L)}$ must be upper bounds of the computation and

memory access time of a synthetic task composed by the largest L-job of the task under analysis, τ_i, and all jobs of higher-priority tasks that arrive in the response time interval. Thus we bound each of these components independently:

$$\widehat{C_i^{e|L}(t)} = g^{e|L}(\tau_i,1) + \sum_{j\in hp(i)} G^{e|L}(\tau_j,t) \tag{5.13}$$

$$\widehat{C_i^{m|L}(t)} = g^{m|L}(\tau_i,1) + \sum_{j\in hp(i)} G^{m|L}(\tau_j,t) \tag{5.14}$$

This is safe, but may be pessimistic as, for any task τ_j, the sequence of jobs with the largest computation demand, $G^{e|L}(\tau_j,R_i^L)$, may differ from the sequence of jobs with the largest memory access time, $G^{m|L}(\tau_j,R_i^L)$.

In steady H-mode, the WCRT of an H-task τ_i resembles that for the L-mode, except that there is interference only by higher-priority H-tasks and H-mode WCET estimates are used:

$$R_i^H = g^H(\tau_i,1) + \sum_{\tau_j\in hpH(i)} G^H(\tau_j,R_i^H) + stall(\widehat{C_i^{e|H}(R_i^H)},\widehat{C_i^{m|H}(R_i^H)}) \tag{5.15}$$

where the arguments of $stall$, $\widehat{C_i^{e|H}(R_i^H)}$ and $\widehat{C_i^{m|H}(R_i^H)}$, are given by (5.16) and (5.17), which again upper-bound the two components independently:

$$\widehat{C_i^{e|H}(t)} = g^{e|H}(\tau_i,1) + \sum_{k\in hpH(i)} G^{e|H}(\tau_k,t) \tag{5.16}$$

$$\widehat{C_i^{m|H}(t)} = g^{m|H}(\tau_i,1) + \sum_{k\in hpH(i)} G^{m|H}(\tau_k,t) \tag{5.17}$$

In *transient H-mode* the WCRT equation becomes:

$$R_i^*(s) = g^H(\tau_i,1) + \sum_{j\in hpL(i)} G^{L+}(\tau_j,s) + \sum_{k\in hpH(i)} g^*(\tau_k,\ell^L(k,R_i^*(s),s),\ell^H(k,R_i^*(s),s)) + stall(\widehat{C_i^{e*}(R_i^*(s))},\widehat{C_i^{m*}(R_i^*(s))}) \tag{5.18}$$

where ℓ^L and ℓ^H are given by (4.24) and (4.25), respectively, and the arguments of $stall$, $\widehat{C_i^{e*}(R_i^*(s))}$ and $\widehat{C_i^{m*}(R_i^*(s))}$, are given by:

$$\widehat{C_i^{e*}(t)} = g^{e|H}(\tau_i,1) + \sum_{j\in hpL(i)} G^{e|L+}(\tau_j,s) + \sum_{k\in hpH(i)} g^{e|*}(\tau_k,\ell^L(k,t,s),\ell^H(k,t,s))$$

$$\widehat{C_i^{m*}(t)} = g^{m|H}(\tau_i,1) + \sum_{j\in hpL(i)} G^{m|L+}(\tau_j,s) + \sum_{k\in hpH(i)} g^{m|*}(\tau_k,\ell^L(k,t,s),\ell^H(k,t,s))$$

where the $g^{e|*}$ and $g^{m|*}$ functions are similar to the g^{*} function, (4.21), except that they use the respective component of the WCET.

$$g^{e|*}(\tau_i,\ell^L,\ell^H)=\begin{cases} g^{e|H}(\tau_i,\ell^H) & \text{if } \ell^L=0 \\ g^{e|L}(\tau_i,\ell^L) & \text{if } \ell^L\neq 0\wedge\ell^H=0 \\ max\left\{\sum\limits_{k=j}^{j+\ell^L-1} C^{e|L}_{i,(k \bmod F_i)} \right. & \\ \quad \left. +\sum\limits_{k=j+\ell^L}^{j+\ell^L+\ell^H-1} C^{e|H}_{i,(k \bmod F_i)} : 0\leq j<F_i\right\} & \\ & \text{if } 1\leq\ell^L<F_i\wedge 1\leq\ell^H<F_i \\ q^L\cdot g^{e|L}(\tau_i,F_i)+g^{e|*}(\tau_i,r^L,r^H)+q^H\cdot g^{e|H}(\tau_i,F_i) & \\ & \text{otherwise} \end{cases} \tag{5.19}$$

$$g^{m|*}(\tau_i,\ell^L,\ell^H)=\begin{cases} g^{m|H}(\tau_i,\ell^H) & \text{if } \ell^L=0 \\ g^{m|L}(\tau_i,\ell^L) & \text{if } \ell^L\neq 0\wedge\ell^H=0 \\ max\left\{\sum\limits_{k=j}^{j+\ell^L-1} C^{m|L}_{i,(k \bmod F_i)} \right. & \\ \quad \left. +\sum\limits_{k=j+\ell^L}^{j+\ell^L+\ell^H-1} C^{m|H}_{i,(k \bmod F_i)} : 0\leq j<F_i\right\} & \\ & \text{if } 1\leq\ell^L<F_i\wedge 1\leq\ell^H<F_i \\ q^L\cdot g^{m|L}(\tau_i,F_i)+g^{m|*}(\tau_i,r^L,r^H)+q^H\cdot g^{m|H}(\tau_i,F_i) & \\ & \text{otherwise} \end{cases} \tag{5.20}$$

Similarly, $G^{e|L+}$ and $G^{m|L+}$ are similar to G^{L+} function, (4.22), except that they use the respective component of WCET.

$$G^{eL+}(\tau_i,t)=g^{e|L}\left(\tau_i,\left\lfloor\frac{t}{T_i}\right\rfloor+1\right) \tag{5.21}$$

$$G^{mL+}(\tau_i,t)=g^{m|L}\left(\tau_i,\left\lfloor\frac{t}{T_i}\right\rfloor+1\right) \tag{5.22}$$

The WCRT is the maximum of all the responses computed for all mode switch instants s less than R_i^L:

$$R_i^*=max\{R_i^*(s):0<s<R_i^L\}$$

Actually, rather than considering all time instants less than R_i^L it is enough to consider only the release time instants of higher-priority L-tasks, assuming that they are released at the same time as the task under analysis as often as possible. Indeed, the original arguments in [119] about different values of s which needs to be considered for response time recurrence are also applicable to analyses presented in this Chapter. By analyzing, the RHS of equation (5.18), we observe that the interference by higher-priority H-tasks decreases with s, whereas the interference by higher-priority L-tasks is a step function that increases at the instant these tasks are

Table 5.2: Parameters for example taskset

Task	Criticality	L-WCET	H-WCET	Period	Deadline
τ_1	L	{3,4,7,7}	-	20	20
τ_2	L	{5,6,3}	-	30	30
τ_3	H	{4,3}	{8,6}	40	40

released. This is also true for their WCET components, i.e. the computation time and the memory access time, and therefore for the stall.

5.5 Exhaustive analysis

The analysis in Section 5.4 is pessimistic because it uses (i) Baruah's g/G functions in accounting the interference by higher-priority tasks and also (ii) independent upper bounds on the accumulated computation time and the accumulated memory access time of each job sequence of a higher-priority task. In this section, we address this pessimism with an exhaustive analysis, which builds on that of Zuhily and Burns [12]. In Section 5.6 we reduce its complexity.

5.5.1 Notation and underlying principles

Fixed-priority WCRT analysis relies on upper-bounding the number of job releases by each higher-priority task until the task under analysis completes. For multiframe tasks, this is not enough. For any given number of interfering jobs by a given higher-priority task, there are as many potential interfering sequences as frames in a superframe: one sequence per initial frame. Because each frame may have different WCET, each such sequence may cause different interference.

To enumerate all interfering higher-priority frame sequences of a given length, we adapt the notation of [12]. Let $L_i = \{0,\ldots,F_i\text{-}1\}$ denote the set of (the indexes of) initial frames of all job sequences of task τ_i. Assuming, without loss of generality, that tasks are numbered from higher to lower priority, each element of the Cartesian product $V_i = \prod_{j=1}^{i-1} L_j$ denotes a tuple of initial frames, one per higher priority task, of all the sets (combinations) of interfering job sequences, consisting of one such sequence per task in $hp(i)$.

For example, we have three tasks as shown in Table 5.2 with rate-monotonic priority. Then, $V_3 = \{(0,0),(0,1),(0,2),\ldots,(3,0),(3,1),(3,2)\}$ is the Cartesian product of $L_1 = \{0,1,2,3\}$ and $L_2 = \{0,1,2\}$. $(1,0) \in V_3$ denotes two sequences of interfering frames, one by τ_1, starting with frame 1, and another by τ_2, starting with frame 0. Note that $(1,0)$ just specifies the initial frames of the interfering sequences. The lengths of the sequences are specified differently. Moreover, to upper-bound the interference of every frame sequence, [12] defines the cumulative function:

$$g(\tau_i, j, k) = \sum_{\ell=j}^{j+k-1} C_{i,(\ell \bmod F_i)} \tag{5.23}$$

for k=1,2,.... I.e., $g(\tau_i, j, k)$ computes the cumulative WCET of a sequence of k frames of τ_i, starting with frame j. We generalize this cumulative function to:

$$g^{\kappa}(\tau_i, j, k) = \begin{cases} \sum_{\ell=j}^{j+k-1} C^{L}_{i,(\ell \bmod F_i)} & \text{if } \kappa = L \\ \sum_{\ell=j}^{j+k-1} C^{H}_{i,(\ell \bmod F_i)} & \text{if } \kappa = H \end{cases} \tag{5.24}$$

for mixed-criticality and define (5.25) to compute cumulative computation and

$$g^{c|\kappa}(\tau_i, j, k) = \begin{cases} \sum_{\ell=j}^{j+k-1} C^{e|L}_{i,(\ell \bmod F_i)} & \text{if } c = e \text{ and } \kappa = L \\ \sum_{\ell=j}^{j+k-1} C^{m|L}_{i,(\ell \bmod F_i)} & \text{if } c = m \text{ and } \kappa = L \\ \sum_{\ell=j}^{j+k-1} C^{e|H}_{i,(\ell \bmod F_i)} & \text{if } c = e \text{ and } \kappa = H \\ \sum_{\ell=j}^{j+k-1} C^{m|H}_{i,(\ell \bmod F_i)} & \text{if } c = m \text{ and } \kappa = H \end{cases} \tag{5.25}$$

5.5.2 AMMC-max exhaustive analysis

To perform an exhaustive analysis, we compute the response time of every frame of a multiframe task, considering all possible interfering sequences by higher priority tasks. Let $v \in V_i$ denote a set of interfering sequences, one per task with higher priority than τ_i, such that the first frame of each sequence is the corresponding element of v.

L-mode analysis The L-mode WCRT of frame j of a task τ_i subjected to v is:

$$R^{v|L}_{i,j} = C^{L}_{i,j} + \sum_{k \in hp(i)} g^{L}\left(\tau_k, v(k), \left\lceil \frac{R^{v|L}_{i,j}}{T_k} \right\rceil\right) + stall\left(\widehat{C^{e|L}_{i,j}(R^{v|L}_{i,j})}, \widehat{C^{m|L}_{i,j}(R^{v|L}_{i,j})}\right) \tag{5.26}$$

where $v(k)$ denotes element k of tuple v, and arguments $\widehat{C^{e|L}_{i,j}(R^{v|L}_{i,j})}$ and $\widehat{C^{m|L}_{i,j}(R^{v|L}_{i,j})}$ of the stall function are given by:

$$\widehat{C^{e|L}_{i,j}(t)} = C^{e|L}_{i,j} + \sum_{k \in hp(i)} g^{e|L}\left(\tau_k, v(k), \left\lceil \frac{t}{T_k} \right\rceil\right) \tag{5.27}$$

$$\widehat{C^{m|L}_{i,j}(t)} = C^{m|L}_{i,j} + \sum_{k \in hp(i)} g^{m|L}\left(\tau_k, v(k), \left\lceil \frac{t}{T_k} \right\rceil\right) \tag{5.28}$$

As usual, (5.26) is solved by recurrence. All iterations use the same value for v, i.e., the same first job for each interfering task. To compute the L-mode WCRT of frame j of task i, $R^{L}_{i,j}$, we consider all possible sequences of interfering tasks:

$$R^{L}_{i,j} = max\left\{R^{v|L}_{i,j} : v \in V_i\right\} \tag{5.29}$$

Finally, for the WCRT of task i in L-mode, we must consider all frames of τ_i:

$$R_i^L = max\left\{R_{i,j}^L : j = 0,...,F_i - 1\right\} \tag{5.30}$$

Steady H-mode analysis The WCRT of H-task τ_i in (steady) H-mode is computed similarly. The response time recurrence is obtained from (5.15), by applying the transformations that allow us to derive (5.31) from (5.12). The arguments of the stall function can be derived from the execution time terms in the RHS of the response time recurrence just like for L-mode.

$$R_{i,j}^{v|H} = C_{i,j}^H + \sum_{k \in hp(i)} g^H\left(\tau_k, v(k), \left\lceil \frac{R_{i,j}^{v|H}}{T_k} \right\rceil\right) + stall\left(\widehat{C_{i,j}^{e|H}(R_{i,j}^{v|H})}, \widehat{C_{i,j}^{m|H}(R_{i,j}^{v|H})}\right) \tag{5.31}$$

where $v(k)$ denotes element k of tuple v, and arguments $\widehat{C_{i,j}^{e|H}(R_{i,j}^{v|H})}$ and $\widehat{C_{i,j}^{m|H}(R_{i,j}^{v|H})}$ of the stall function are given by:

$$\widehat{C_{i,j}^{e|H}(t)} = C_{i,j}^{e|H} + \sum_{k \in hp(i)} g^{e|H}\left(\tau_k, v(k), \left\lceil \frac{t}{T_k} \right\rceil\right) \tag{5.32}$$

$$\widehat{C_{i,j}^{m|H}(t)} = C_{i,j}^{m|H} + \sum_{k \in hp(i)} g^{m|H}\left(\tau_k, v(k), \left\lceil \frac{t}{T_k} \right\rceil\right) \tag{5.33}$$

The WCRT of each frame of task i in H-mode is then obtained by taking the maximum of the WCRT for all sequences of interfering tasks, like in (5.29).

$$R_{i,j}^H = max\left\{R_{i,j}^{v|H} : v \in V_i\right\} \tag{5.34}$$

Finally, the WCRT of task i is obtained by taking the maximum of the WCRT of all of its frames, as in (5.30).

$$R_i^H = max\left\{R_{i,j}^H : j = 0,...,F_i - 1\right\} \tag{5.35}$$

Transient H-mode analysis We now focus on the WCRT analysis of H-tasks caught by mode-switch. Such tasks can suffer the interference of sequences containing not only H-jobs but also L-jobs, i.e jobs that complete before the mode switch, of higher priority H-tasks. To bound this interference, we combine the cumulative function, (5.23), with the g^* function, (4.21), and define the cumulative WCET of a sequence of $\ell^L + \ell^H$ frames of task τ_i, starting with frame j, such that ℓ^L frames complete in L-mode and the remaining in H-mode:

$$g^*(\tau_i, j, \ell^L, \ell^H) = g^L(\tau_i, j, \ell^L) + g^H(\tau_i, (j + \ell^L) \bmod F_i, \ell^H) \tag{5.36}$$

Likewise, we define $g^{e|*}$ and $g^{m|*}$ to compute the cumulative computation and memory access time of frame sequences.

$$g^{e|*}(\tau_i, j, \ell^L, \ell^H) = g^{e|L}(\tau_i, j, \ell^L) + g^{e|H}(\tau_i, (j+\ell^L) \bmod F_i, \ell^H) \tag{5.37}$$

$$g^{m|*}(\tau_i, j, \ell^L, \ell^H) = g^{m|L}(\tau_i, j, \ell^L) + g^{m|H}(\tau_i, (j+\ell^L) \bmod F_i, \ell^H) \tag{5.38}$$

Let $R_{i,j}^{v|*}$, with $v \in V_i$, denote the worst-case response time of frame j of task τ_i caught by a mode switch when subjected to the interference of a set of frame sequences, one per higher priority tasks, such that the first frame of each of these sequences is the corresponding element in v. Then, with AMMC-max, the WCRT recurrence for a task caught by a mode switch becomes:

$$\begin{aligned} R_{i,j}^{v|*}(s) = \; & C_{i,j}^{H} + \sum_{k \in hpL(i)} g^{L}\left(\tau_k, v(k), \left\lfloor \frac{s}{T_k} \right\rfloor + 1\right) + \sum_{k \in hpH(i)} g^{*}(\tau_k, v(k), \ell^L(k, R_{i,j}^{v|*}(s), s), \ell^H(k, R_{i,j}^{v|*}(s), s)) \\ & + stall\left(\widehat{C_{i,j}^{e*}(R_{i,j}^{v|*}}(s)), \widehat{C_{i,j}^{m*}(R_{i,j}^{v|*}}(s))\right) \end{aligned}$$

where $\ell^L(k, R_{i,j}^{v|*}(s), s)$ and $\ell^H(k, R_{i,j}^{v|*}(s), s)$ are appropriate values for the number of interfering jobs that complete before/after the mode change, respectively. We discuss the selection of these values in this section's last paragraph.

As for other modes, the arguments of *stall* are easily derived from the RHS of (5.39), with the help of component-wise cumulative functions (5.37) and (5.38) and are given as follows:

$$\widehat{C_{i,j}^{e|*}(t)} = C_{i,j}^{e|H} + \sum_{k \in hp(i)} g^{e|*}(\tau_i, j, \ell^L, \ell^H) \tag{5.39}$$

$$\widehat{C_{i,j}^{m|*}(t)} = C_{i,j}^{m|H} + \sum_{k \in hp(i)} g^{m|*}(\tau_i, j, \ell^L, \ell^H) \tag{5.40}$$

Note that (5.39) provides the response time of a frame j of task i when subject to the set of interfering sequences, $v \in V_i$, as a function of s. Thus:

$$R_{i,j}^{v|*} = max\{R_{i,j}^{v|*}(s) : 0 < s < R_{i,j}^{v|L}\}$$

As argued in Section 5.4, it suffices to compute $R_{i,j}^{v|*}(s)$ only for values of s equal to release times of higher priority L-tasks. Like for the other modes, from $R_{i,j}^{v|*}$ the response time of frame j of task i is computed by taking the maximum for all sequences of interfering tasks, and from that the response time of task i, by taking the maximum for all frames.

Quantification of $\ell^L(k, R_{i,j}^{v|*}(s), s)$ and $\ell^H(k, R_{i,j}^{v|*}(s), s)$

Equation (5.39) is to be solved via a recurrence relation, as in classic AMC-max. However, for the recurrence to converge, appropriate values for $\ell^L(k, R_{i,j}^{v|*}(s), s)$ and $\ell^H(k, R_{i,j}^{v|*}(s), s)$ must be selected. The quantification of interfering jobs completing before/after the mode change, does *not* carry over from classic AMC-max, because

it may prevent the recurrence from converging, as evidenced by the following counter-example, that uses classic AMC-max values for ℓ^L and ℓ^H (see (4.24) and (4.25)):

Consider the computation of the response time of a frame of H-task τ_i upon mode switch, using (5.39). Assume that the L- and H-WCET estimates of higher priority H-task τ_j are $\{20,5,1\}$ and $\{40,10,2\}$, respectively. Assume that frame 0 is the first one for τ_j. Assume that, in iteration k, there are two interfering H-jobs; then the cumulative interference is 50. Assume that in iteration $k+1$, the number of interfering jobs increases to 3, the first of which is an L-job and the othes are H-jobs, the cumulative interference will decrease to 32. As a result, the response time computed in iteration $k+1$ may be smaller than in iteration k. Such non-monotonicity in the cumulative interference with respect to time may prevent the recurrence from converging. To address this issue we define:

$$\ell^L(k,t,s) = max\left(\left\lfloor \frac{s}{T_k} \right\rfloor - 1, 0\right) \tag{5.41}$$

$$\ell^H(k,t,s) = \left\lceil \frac{t}{T_k} \right\rceil - \ell^L(k,t,s) \tag{5.42}$$

There are three key properties of the quantification of $\ell^L(k,t,s)$ and $\ell^H(k,t,s)$ via (5.41) and (5.42) respectively that ensure that (5.39) can be solved using a recurrence and that its solution is safe:

Lemma 2. (5.41) *provides a lower bound on the number of jobs of a periodic task τ_k with period T_k that execute completely in a time interval, I, of length s.*

Proof sketch: The minimum number of job releases of τ_k in I is $\lfloor \frac{s}{T_k} \rfloor$. If the release of a job is inside I and the release of the following job is also inside I, then the former job executes completely in I. Therefore, the number of jobs that execute completely in I is one less than the number of job releases in I.

Lemma 3. (5.42) *is an upper bound on the number of jobs of sporadic task τ_k with minimum inter-arrival time T_k released in a time interval I, of length t, that complete after time instant s, relative to the beginning of I. Also, the sum of* (5.41) *and* (5.42) *is equal to the maximum number of job releases of τ_k in I.*

Proof sketch: The maximum number of job releases of τ_k in I is $\lceil \frac{t}{T_k} \rceil$. From Lemma 2, at most $\ell^l(k,t,s)$ of those complete before s.

The third property is that (5.41) is independent of t. This is trivial to check, but it is crucial as it ensures that in the solution of (5.39) by a recurrence, the interference of jobs of higher priority H-task τ_k before the mode switch has the same value in all iterations. Furthermore, the first H-job of the interfering job sequence of τ_k is always the same frame, and therefore the interference by the H-job sequence is monotonically non-decreasing. (As shown, for the memory agnostic case in Lemma 1 of [14].) Therefore, in the solution of the recurrence (5.39), $R_{i,j}^{v|*}(s)$ is monotonically non-decreasing, thus ensuring termination, either by convergence or by deadline violation.

5.6 Pruning

The exhaustive analysis of Section 5.5 is tighter than the analysis of Section 5.4 because it exhaustively computes the WCRT of (i) every frame of all tasks and (ii) for all possible frame sequences of interfering tasks. Zuhily and Burns [12] use *pruning* to reduce running times. We now adapt that pruning approach to multi-frame mixed-criticality systems deployed over memory bandwidth regulated multicore platform presented in Section 3.2 of Chapter 3.

The work in [12] considered a single core platform whereas our work targets a multi-core platform with memory regulation. Therefore, we need to account for the two components of the WCET, because, as summarized in Section 5.2, in a multicore platform with memory regulation, a job may incur a stall which depends on those components. As an extreme example, two jobs with the same WCET but one with a zero-valued memory access time, and the other with a zero-valued compute time, will incur different stalls: whereas the first job will incur no stall the latter may incur a significant stall, i.e. comparable or even higher than the job's WCET. Thus, whereas [12] relies on the $<$ relation in $\mathbb{R}$ to prune jobs, our approach relies on the relation $<$ between elements of $\mathbb{R}^2$, the WCET pairs, defined as follows:

Definition 5. *Let* $(x_1, y_1), (x_2, y_2) \in \mathbb{R}^2$. *Then,* $(x_1, y_1) < (x_2, y_2)$ *iff* $(x_1 < x_2 \wedge y_1 \leq y_2 \vee x_1 \leq x_2 \wedge y_1 < y_2)$

Pruning frames: To compute the WCRT of a task, rather than consider every frame of that task, Zuhily and Burns [12] consider only the *peak frame* of each task, i.e., the one with the maximum WCET among its frames. In our model, since $\mathbb{R}^2$ is a poset under the $<$ relation defined above, there may not be a total order among the WCET pairs of a task's frames. So, for each task, we must compute the WCRT for each frame whose WCET pair is maximal among the set of WCET pairs of the frames. If multiple frames have the same maximal WCET pair, analyzing one of them suffices.

With AMC, we must compute WCRTs for H-tasks in both modes. Thus, we perform pruning independently for both modes, using the respective WCETs. I.e., for H-task τ_i, we define $\hat{F}_i^L$ and $\hat{F}_i^H$ – the sets of frames of τ_i whose WCET pairs are maximal in L- and H-mode, respectively. With these sets, to compute the L-mode WCRT of τ_i, we compute the response time for each of the frames in $\hat{F}_i^L$ and take the maximum. Similarly for the H-mode and mode switch, we consider only the frames in $\hat{F}_i^H$. For an L-task τ_i, we only define $\hat{F}_i^L$.

Pruning frame sequences: The analysis in Section 5.5 solves one recurrence for every element of the cartesian product of the L-sets of higher-priority tasks, i.e. the V-sets. In [12], to reduce the number of response time recurrences, Zuhily and Burns prune elements from the L sets, thus reducing the cardinality of their cartesian products and ultimately the number of recurrences.

The pruned set, $\hat{L}_i$, is obtained from set L_i by removing every frame j that fulfills the following condition: There exists a frame $k \neq j$ such that, for every sequence of ℓ frames (with $1 \leq \ell < F_i$), the cumulative WCET of the sequence starting with frame j is smaller than that of the same-length sequence that starts with frame k.

Our approach differs from that in [12] in two aspects: 1) For each task, we define set $\hat{L}^L$, furthermore for each H-task, we define also sets $\hat{L}^H$ and $\hat{L}^*$, since we must consider each mode; 2) We use the $<$ relation on $\mathbb{R}^2$, not the $<$ relation on $\mathbb{R}$, since we must ensure that the stall caused by the interfering task is also taken into account.

Table 5.3: Overview of Parameters for multiframe mixed-criticality systems over MPSoCs

Parameters	Values	Default
Number of cores (K)	$\{2,4,6,8\}$	2
Memory Intensity (γ)	$\{.1 : .1 : .9\}$	0.4
H-WCET scale-up factor (κ)	$\{2 : 0.5 : 6\}$	2
Task-set size ($\|\tau\|$)	$\{6,8,10,12\}$	10
Fraction of H-tasks in τ (ξ)	$\{0.05 : 0.05 : 0.6\}$	0.4
Upper bound on # of frames (α)	$\{3 : 1 : 8\}$	5
Lower bound on L-WCET (β)	$\{0.1 : 0.1 : 0.8\}$	0.2
Inter-arrival time (T_i)	$10msec$ to $1sec$	N/A

Thus, sets $\hat{L}_i^L$ and $\hat{L}_i^H$ are obtained from set L_i using the L-WCET and the H-WCET estimates, respectively, and by using the relation $<$ on $\mathbb{R}^2$, to compare the cumulative WCET pairs of sequences of ℓ jobs, with $1 \leq \ell < F_i$.

Set $\hat{L}_i^*$, used for mode switch analysis, is obtained from L_i, by removing every frame j for which there is a frame $k \neq j$ such that, for every sequence of ℓ^L L-mode frames followed by a sequence of ℓ^H H-mode frames, with $0 \leq \ell^L \leq F_i$ and $0 < \ell^H \leq F_i$, the cumulative WCET pair of the sequence starting with frame j is $<$ than that of the sequence starting with frame k.

With the definitions above, rather than solve one response time recurrence per element of $V_i = \prod_{j=0}^{i-1} L_j$, we need only solve one recurrence per element of the Cartesian product of the appropriate "pruned" sets. For example, to compute the WCRT of an H-task τ_i upon mode switch, we solve one recurrence per element of $\prod_{j=0}^{i-1} \hat{L}_j^*$.

5.7 Evaluation

We implemented both analyses for performance evaluation. Our simulator has two modules. One module generates parameterizable synthetic workloads. Another one assigns bandwidth and tasks to cores, and tests for schedulability. Synthetic task sets are generated as follows:

- The unbiased task utilizations in L-mode are generated using the UUnifast-discard [110, 111] algorithm and varied within $[0.1,\ 1] \times K$.
- Task periods (10 msec to 1 sec) are log-uniform-distributed. Deadlines of the tasks are assumed implicit (T_i=D_i), though analyses hold for constrained deadlines ($D_i \leq T_i$).
- Task-set size ($|\tau|$) is an input parameter. The number of H-tasks is $\lfloor \xi \times |\tau| \rceil$, where $\xi \in (0,1)$ is a user defined fraction of H-tasks and $\lfloor \cdot \rceil$ denotes rounding to the nearest integer.
- The number of frames of each task is randomly selected over $[1,\ \alpha]$, where upper-bound α is an input parameter.

- The L-WCET of the first frame is generated by multiplying the period of a task with its utilization, i.e., $C^L_{i,1}{=}T_i * U_i$. The L-WCETs of other frames are randomly generated within $[\beta \times C^L_{i,1},\ C^L_{i,1}]$, where $\beta{\in}(0,1]$ is an input parameter[1]
- For any frame, its H-WCET is computed by linear scaling its L-WCET (i.e., $C^H_{i,j} = \kappa \times C^L_{i,j}$), where κ is an input parameter.
- The total WCET is divided between memory access time and the CPU computation time using an input parameter γ. The memory access time of a frame in mode $x \in \{L,H\}$ is randomly selected within $[0,\ \gamma \times C^x_{i,j}]$; what remains is the CPU computation time, i.e., $C^{e|x}_{i,j}{=}C^x_{i,j}{-}C^{m|x}_{i,j}$.

For all the schedulability analysis We used Audsley's [31] optimal priority assignment scheme. For frame-agnostic analyses, we use for each task τ_i the parameters $g^{e|\kappa}(\tau_i,1)$, $g^{m|\kappa}(\tau_i,1)$ and $g^{e|\kappa}(\tau_i,1)+g^{m|\kappa}(\tau_i,1)$ for its worst-case CPU computation time, worst-case memory access time and total WCET. Table 5.3 summarizes the parameters. The triples represent minimum value, increment size and maximum value for a parameter. Each random parameter uses its own pseudo random number generator, which is seeded in the first replication [117].

For every parameter in this table, except the inter-arrival time, we run a two-factor experiment in which one of the factors is that parameter and the other factor is the task set L-mode utilization. In each of these experiments, the remaining parameters take their default values.

For each combination of input parameters, 100 task sets are generated. As in [55], a memory access takes 40 nsec and the regulation period is 100 μsec. Task-to-core assignment and bandwidth allocation are performed using memory-fit [120]. This heuristic assigns a task to the core that requires the least memory budget increase to accommodate it, as quantified via binary search.

Because we use a two-factor experimental design, we use weighted schedulability [121, 66] as evaluation metric, making it possible to show the results in two dimensional plots rather than in three dimensional plots.

Let $S_y(\tau,p)$ denote the result of schedulability test y for task set τ for an input parameter p. The weighted schedulability $W_y(p)$ is:

$$W_y(p) = \frac{\sum_{\forall\tau}(\bar{U}^L(\tau)\cdot S_y(\tau,p))}{\sum_{\forall\tau}\bar{U}^L(\tau)} \tag{5.43}$$

where, $\bar{U}^L(\tau) = \frac{U^L(\tau)}{K}$ is L-mode system utilization normalized by number of cores K. The experiments ran on a system with 8 Xeon E5-2420 2.20 GHz dual cores, 32 GB of RAM and Linux Mint 18.3 Sylvia with open jdk 1.8.0_242.

5.7.1 Results

In our experiments, the simpler multiframe analysis (AMMC-max, Section 5.4) outperforms the frame-agnostic AMC-max, AMC-rtb and SMC, and, surprisingly, almost matches the exhaustive analysis (AMMC-max-Z, Section 5.5).

[1] For convenience, without loss of generality (since shift-rotating the order of the frames results in an equivalent multiframe task), the first frame has the biggest L-WCET. Note that our analyses make no assumptions regarding the WCET of the first frame being the greatest.

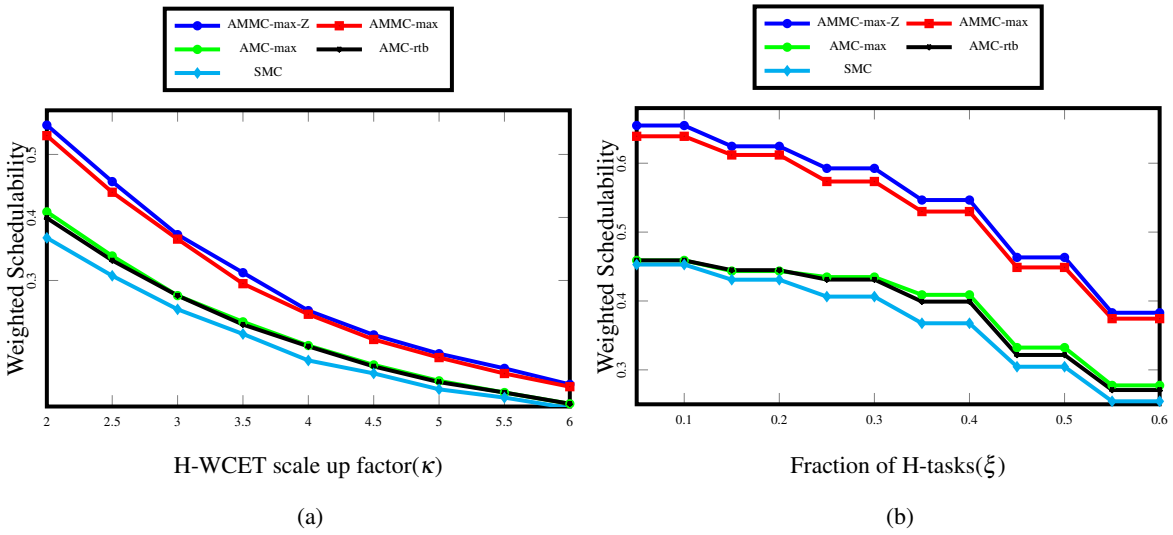


Figure 5.2: Weighted schedulability of different parameter variation a) Effect of H-tasks share in a taskset b) Effect of H-WCET variation

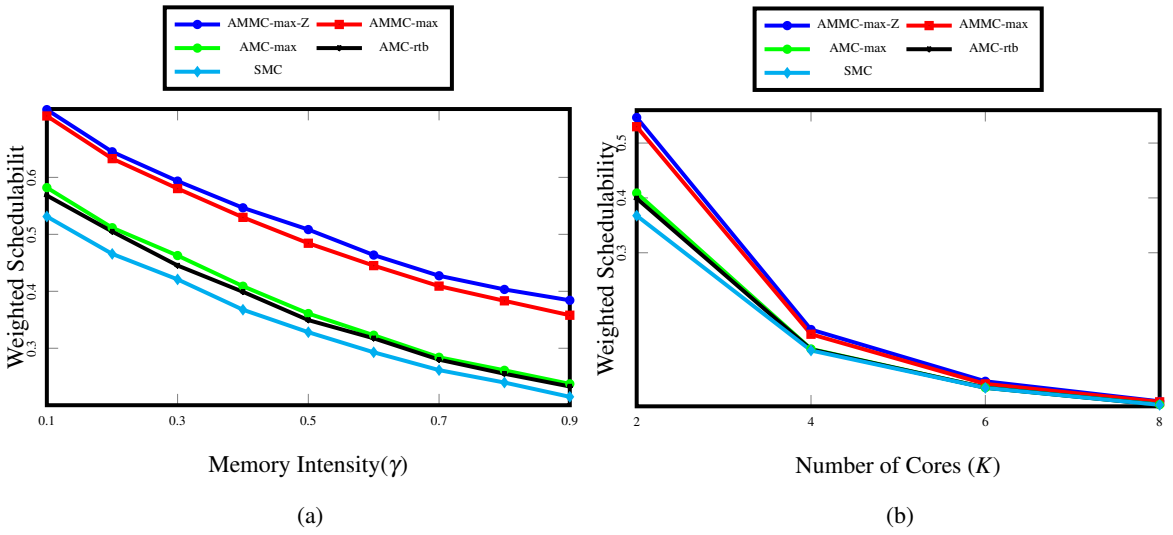
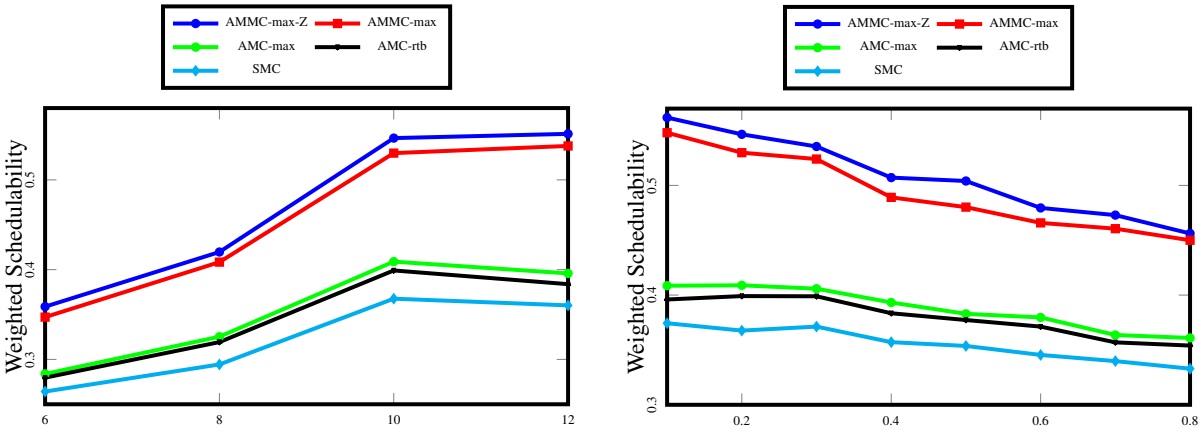


Figure 5.3: Weighted schedulability of different parameter variation a) Effect of number of cores b) Effect of memory intensity

The effect of the H-WCET scale-up factor (ξ) and the H-task fraction (ζ) are presented in Figures 5.2a and 5.2b. High values for either, increase the processing requirements of H-tasks, hence lowering the weighted schedulability (and performance difference among tests). The number of generated H-tasks is rounded up to the nearest integer when the target ζ yields a non-integer number. This explains the step-like behavior in Figure 5.2b.

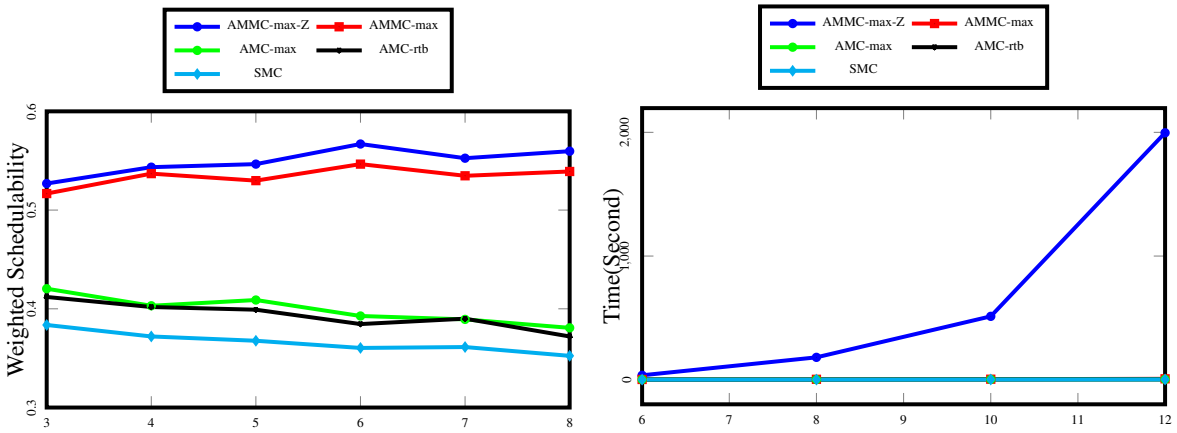
A higher memory intensity increases the overall stall, detrimentally for schedulability (Figure 5.3a). With

more cores, the overall interference among cores increases, impacting the schedulability for all tests (Figure 5.3b). It is expected, since the memory bandwidth is not scaled up with the increase in the number of cores. The higher interference also subsumes the benefits of better analysis. Many low-utilization tasks are easier to schedule than few high-utilization tasks, due to bin-packing fragmentation. Hence, the performance of all tests improves with task set size (Figure 5.4a). With more tasks, there is also more potential to leverage frame information, hence the greater performance difference between frame-agnostic and multiframe analyses with large task set sizes.



(a) (b)

Figure 5.4: Weighted schedulability of different parameter variation a) Effect of tskset set size b) Lower bound on L-WCET variation



(a) (b)

Figure 5.5: Weighted schedulability of different parameter variation a) Effect of number of frames b) Average running time

Table 5.4: Maximum absolute difference in unweighted schedulability success ratio compared to AMC-max over all experiments for each parameter varied.

Analysis	κ	$\lvert\tau\rvert$	ξ	α	β	γ	K
AMMC-max-Z	39%	45%	69%	54%	47%	**72%**	39%
AMMC-max	37%	41%	**66%**	50%	44%	62%	37%

Parameter β lower-bounds the ratio between the smallest and the biggest L-WCET, among a task's frames. As β tends to 1, the generated task sets approximate single-frame tasks, thereby reducing the potential benefits from leveraging the frame information. This effect is evident for all tests (Figure 5.4b). Varying the number of frames shows a roughly ascending trend for the multiframe analyses (Figure 5.5a). Indeed, having more frames per task means a higher percentage of frames not exhibiting the worst-case computation and/or memory access time – which multiframe analyses can leverage. The trend is opposite for the frame-agnostic approaches, because the single-frame representation of a multiframe task combines the worst-case computation and the worst-case memory access time over all of the latter's frames (possibly, different ones). The pessimism from that is more pronounced, the higher the number of frames is.

The test running time of the simpler multiframe analysis (AMMC-max) is of the same order of magnitude as that of the frame agnostic analyses, whereas the exhaustive analysis (AMMC-max-Z) is several orders of magnitude higher(Figure 5.5b).

Finally, Table 5.4 provides the maximum absolute difference in *unweighted* schedulability success ratio of each multiframe max analysis (AMMC-max-z and AMMC-max) from the single frame analysis (AMC-max), when varying different parameters. The maximum difference for each analysis is highlighted in Table 5.4. In the best case, AMMC-max-Z achieves up to 72% higher success ratio than AMC-max. The results for the simpler analysis is close to those of the exhaustive analysis, whereas its running time is much shorter.

5.8 Chapter summary

We formulated schedulability analysis for multiframe mixed-criticality tasks with memory regulation, that incorporates the related memory stalls. The state-of-the-art previously only offered analysis for systems with at most any two of the following three characteristics: (i) mixed criticalities; (ii) multiframe tasks; and (iii) memory access regulation. By covering all three of those our work allows for safely leveraging frame information of mixed-criticality tasks, for better schedulability, also on COTS multicore architectures; no longer just single cores. As our experiments with synthetic tasks demonstrated, the frame-aware and stall-aware schedulability analysis substantially outperforms its frame-agnostic (but still stall-aware) counterparts. This validates our approach as a tool for safely harnessing the power of COTS multicores for critical workloads.

Chapter 6

Controller Area Network with a combination of PQ and FQ nodes for multisized messages

In this chapter, we develop a schedulability analysis based on the multi-frame task analysis for CAN messages, that improves the schedulability of CAN networks where the size of message with a given identifier exhibit a periodic pattern similar to that of multi-frame tasks.

The rest of this chapter is composed of the following sections. In Section 6.1, background about CAN schedulability analysis. Section 6.2 presents a relatively quick but pessimistic analysis for multisized CAN messages in CAN networks whose nodes use a priority-queuing policy. Tighter analysis is developed in Section 6.3. Section 6.4 illustrates, via examples, the application of these two new analyses and the kind of improvement they can achieve over the state-of-the art. Section 6.5 presents existing schedulability test for mixed-queue network. It also presents schedulability test for simpler and tight extended schedulability analysis. Sections 6.7 is extended in Section 6.2 to mixed-queue networks, i.e. CAN systems whose nodes may have different queuing policies. Section 6.8 discusses the evaluation setup and the results. Finally, in Section 6.9 we conclude this chapter.

6.1 Background on PQ-based CAN schedulability analysis

The state-of-the-art in CAN schedulability analysis is found in [20], superseding all earlier seminal works [88, 89, 90] and fixing certain issues. All these works apply uniprocessor fixed-priority based response time analysis to the schedulability of the CAN network. Indeed, we can map the problem of scheduling messages on a bus to the problem of scheduling tasks on a single core processor. The main difference between these two problems arises from the fact that conventional fixed-priority-based response time analysis assumes preemptive tasks, whereas, in CAN, a message whose transmission has already began cannot be preempted by a higher-priority message that arrived meanwhile.

More specifically, the analysis by Davis et al. [20] assumes that nodes use a priority-queuing policy and bounds the time since the arrival of a message, m, until the end of its transmission. Each message is assumed to have a worst-case transmission time, C, a minimal inter-arrival time, T, and a deadline for its transmission, D, relative to its arrival. The worst-case response time (WCRT) of message m is given by:

$$R_m = J_m + w_m + C_m \tag{6.1}$$

where

J_m is the queuing jitter, the longest time between the arrival of a message and adding it to the transmission queue;

w_m is the queuing delay, the longest time the message remains in the transmission queue, until its successful transmission on the bus begins;

C_m is the longest message transmission time.

The queuing delay has two components:

B_m is the longest time m may have to wait for a lower-priority message to complete its transmission. This models the case when such a message is being transmitted when m is queued.

I_m is the worst-case interference caused by higher priority messages that arrive before m starts being transmitted and therefore win access to the bus.

The blocking time is given by:

$$B_m = \max_{i \in lp(m)} (C_i) \tag{6.2}$$

where $lp(m)$ is the set of messages of lower priority than m.

The computation of I_m relies on the concept of *busy period*, introduced by Lehoczky [112].

Davis et al. define a priority level-m busy period as a time interval $[t^s, t^e)$, i.e. a right-open interval, where:

t^s is the time when a message of priority m or higher is queued for transmission, and there are no messages of priority m or higher waiting to be transmitted that were queued strictly before t^s;

t^e is the earliest time when the bus becomes idle, i.e. ready for transmission, and there are no messages of priority m or higher waiting to be transmitted that were queued strictly before time t^e.

Note that, during a level-m busy period, messages with priority lower than m do not start transmission. Furthermore, all messages with priority higher than or equal to m that are queued during a busy period are transmitted during that busy period.

Thus the largest length, v_m, of a level-m busy period is given by the recurrence:

$$v_m^{n+1} = B_m + \sum_{\forall k \in hep(m)} \left\lceil \frac{v_m^n + J_k}{T_k} \right\rceil C_k \tag{6.3}$$

where $hep(m)$ is the set of messages with priority higher than or equal to m.

This recurrence is guaranteed to converge provided:

$$\sum_{\forall k \in hep(m)} \frac{C_k}{T_k} < 1 \tag{6.4}$$

i.e., the total bus utilization of messages with priority higher than or equal to m is less than 1.

The number of instances Q_m of message m that arrive during a level-m busy period is given by:

$$Q_m = \left\lceil \frac{v_m + J_m}{T_m} \right\rceil \tag{6.5}$$

Thus, the analysis computes worst-case response time of each of these instances.

Let q denote the order of an instance of a message m in a level-m busy period. Thus $q = 0$ for the first instance and $q = Q_m - 1$ for the last one.

The longest time from the start of the level-m busy period until instance q begins successful transmission, $w_m(q)$, is given by the solution to the following recurrence:

$$w_m^{n+1}(q) = B_m + qC_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n(q) + J_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{6.6}$$

where τ_{bit} is a CAN bit transmission time.

The response time of instance q is given by:

$$R_m(q) = J_m + w_m(q) - qT_m + C_m \tag{6.7}$$

Thus, the worst-case response time of message m is:

$$R_m = \max_{q=0..Q_m-1}(R_m(q)) \tag{6.8}$$

Note that if for some q, we have $R_m(q) > D_m$ the message is unschedulable, and the analysis should stop.

Finally, the worst-case transmission time C_m of a message m (which includes the transmission of its identifier and bits added for error-resilience and synchronization, including bit-stuffing) as a function of its size s_m in bytes is given by [20]:

$$C_m = (55 + 10 \cdot s_m) \cdot \tau_{bit} \qquad \text{if ids are 11-bit} \tag{6.9}$$

$$C_m = (80 + 10 \cdot s_m) \cdot \tau_{bit} \qquad \text{if ids are 29-bit} \tag{6.10}$$

6.2 Analysis for PQ-based CAN with multisized messages

The schedulability analysis for CAN with multisized messages builds on the worst-case response time analysis of the CAN protocol by Davis et al. [20] and on the response time analysis for the multiframe task model by

Baruah et al. [11]. These works are briefly summarized in Section 4.2 and Section 6.1, respectively. Details about the CAN system model for multisized messages is presented in Chapter 3 (Section 3.3).

6.2.1 Multisized messages

In order to upper-bound the interference caused by higher priority messages we adapt the $g(.)$ and $G(.)$ functions of multiframe tasks to multisized messages as follows:

$$g_m(k) = \begin{cases} 0 & \text{if } k = 0 \\ \max\limits_{0 \le j < S} \left(\sum\limits_{\ell=j}^{j+k-1} C_{m,(\ell \bmod S)} \right) & \text{if } 1 \le k \le S \\ q \cdot g_m(S) + g_m(r) & \text{otherwise} \\ \quad \text{where } q = k \text{ div } S, \text{ and} & \\ \qquad r = k \bmod S & \end{cases} \tag{6.11}$$

where $C_{m,0} \ldots C_{m,(S-1)}$ are the transmission times for each message in a sequence of S messages, and S is the *cycle length* for message m, i.e., the smallest integer such that $C_{m,k} = C_{m,(k+S)}$ for any integer k.

Similarly, we define the $G_m(t)$ function:

$$G_m(t) = g_m\left(\left\lceil \frac{t}{T_m} \right\rceil \right) \tag{6.12}$$

which upper-bounds the cumulative transmission request of message m within a time interval of duration t.

Consider a multisized message with a cycle length of 3 and a period of 200 τ_{bit} , and let its message subtypes have payloads of 2, 4 and 1 byte, respectively. Their corresponding worst-case transmission lengths (assuming 11-bit ids and applying Equation 6.9 would be 75, 95 and 65 τ_{bit}. Thus we have $g_m(1) = \max(75, 95, 65) = 95$, $g_m(2) = \max(75+95, 95+65, 65+75) = 170$ and $g_m(3) = \max(75+95+65, 95+65+75, 65+75+95) = 235$ τ_{bit}. And, over a time interval of length, e.g., 380 τ_{bit}, corresponding to the WCRT of some lower-priority message, the maximum cumulative transmission request (analogous to cumulative task execution request, in real-time task scheduling analysis) of message m is given by:

$$G_m(380) = g_m\left(\left\lceil \frac{380}{200} \right\rceil \right) = g_m(2) = 170 \ \tau_{bit}$$

In the rest of the chapter, we will generally use τ_{bit} as the time unit, for ease of notation, without loss of generality.

6.2.2 Response time analysis

With these definitions, we are ready to generalize the analysis in Davis et al. [20] to multisized messages.

The analysis of the response time of message m starts with the computation of the length of the level-m busy period. For this computation rather than using (6.3), we use the following recurrence:

$$v_m^{n+1} = B_m + \sum_{\forall k \in hep(m)} G_k(v_m^n + J_k) \tag{6.13}$$

where:

$$B_m = \max_{i \in lp(m)} g_i(1) \tag{6.14}$$

A possible starting value is $v_m^0 = B_m + g_m(1)$.

Indeed, when different subtypes of a message may have different lengths, the blocking term can be as large as the maximum transmission time of the largest message subtype of each of the messages with lower priority than m. Furthermore, it is not enough to count the number of releases in the interval under consideration, rather we need also to consider the phasing of these instances. Since the right-hand-side of (6.13) is monotonically non-decreasing with respect to v_m, the convergence condition is similar to (6.4):

$$\sum_{\forall k \in hep(m)} U_k < 1 \tag{6.15}$$

where U_k is defined as follows:

$$U_k = \frac{1}{S} \sum_{j=0}^{S-1} \frac{C_{k,j}}{T_k} \tag{6.16}$$

where S is the message cycle length of message k.

The next steps are to compute the response time for each of the:

$$Q_m = \left\lceil \frac{v_m + J_m}{T_m} \right\rceil \tag{6.17}$$

instances of message m during the maximum length level-m busy period.

The longest time from the start of the level-m busy period until instance q, which ranges from 0 to $Q_m - 1$, begins successful transmission is given by:

$$w_m^{n+1}(q) = B_m + g_m(q) + \sum_{\forall k \in hp(m)} G_k(w_m^n(q) + J_k + \tau_{bit}) \tag{6.18}$$

Possible starting values are $w_m^0(q) = B_m$, for $q = 0$ and $w_m^0(q) = w_m(q-1) + g_m(q) - g_m(q-1)$, otherwise.

The response time of instance q is given by:

$$R_m(q) = J_m + w_m(q) - q\,T_m + g_m(1) \tag{6.19}$$

Thus, the worst-case response time of message m is:

$$R_m = \max_{q=0..Q_m-1} (R_m(q)) \tag{6.20}$$

as in [20] (see (6.8)).

This analysis is safe but it may be pessimistic. Except for $q = 0$, we may be accounting the longest frame twice: once explicitly in the last term of (6.19) and possibly a second time implicitly in $g_m(q)$ in (6.18).

We can remove this pessimism by modifying (6.19) to:

$$R_m(q) = J_m + w_m(q) - qT_m + g_m(q+1) - g_m(q) \tag{6.21}$$

That is, to compute the interference by higher-priority messages we assume that the first q message instances take the longest to transmit, and therefore the $q+1$ instance will suffer maximum interference. On the other hand, by using $g_m(q+1) - g_m(q)$ in (6.21), we prevent the analysis from being pessimistic w.r.t. the cumulative transmission time of instances of message m.

6.3 Tighter analysis for PQ-based CAN with multisized messages

Note that the analysis in the previous section, even after the last change, although safe, may still be pessimistic. This is because we are considering the worst case for each instance q, but some of these worst cases may not occur simultaneously. For example, consider the multisized message example in Sec. 6.2.1. The worst case for $q = 0$ corresponds to the case where message subtype 1 is the first instance. However, for $q = 1$, the worst case happens when message subtype 0 is the first instance. It may be the case that when message subtype 0 is the first, the level-m busy-period is shorter than $T_m - J_m$ and therefore in that case, the sequence of message subtypes $0, 1$ will never occur in a level-m busy period.

We can avoid some of this pessimism, by computing the length of the level-m busy period for each of the S subtypes of message m as the first message instance in that busy period, and, for each of these busy periods, by computing the response time for each message instance in the busy period under consideration. With this approach, rather than solving just one recurrence per message m to compute the waiting time, we have to solve S recurrences. Thus, we can trade-off pessimism for analysis run time.

6.3.1 Analysis

In order to derive this tighter analysis we define $g_m(i,k)$, the cumulative transmission time of k consecutive instances of message m starting with subtype i, as:

$$g_m(i,k) = \sum_{j=i}^{i+k-1} C_m(j \bmod S) \tag{6.22}$$

where $C_m(j \bmod S)$ is the transmission time of subtype $j \bmod S$ of message m.

Likewise, we define $G_m(i,t)$, the cumulative worst-case transmission request of consecutive instances of message m starting with subtype i in a time interval with length t:

$$G_m(i,t) = g_m\left(i, \left\lceil \frac{t}{T_m} \right\rceil\right) \tag{6.23}$$

We can now compute the level-m busy period starting with subtype i of message m using the following recurrence:

$$v_m^{n+1}(i) = B_m + G_m(i, v_m^n(i) + J_m) + \sum_{\forall k \in hp(m)} G_k(v_m^n(i) + J_k) \tag{6.24}$$

A possible starting value is $v_m^0(i) = B_m + g_m(i,1)$. The number of instances of message m in $v_m(i)$ is given by:

$$Q_m(i) = \left\lceil \frac{v_m(i) + J_m}{T_m} \right\rceil \tag{6.25}$$

Again, for each instance q, ranging from 0 to $Q_m(i) - 1$, of message m in the level-m busy period we can compute its transmission start waiting time, using recurrence:

$$w_m^{n+1}(i,q) = B_m + g_m(i,q) + \sum_{\forall k \in hp(m)} G_k(w_m^n(i,q) + J_k + \tau_{bit}) \tag{6.26}$$

Possible starting values are $w_m^0(i,0) = B_m + \sum_{k \in hp(m)} g_k(1)$, and $w_m^0(i,q) = w_m(i,q-1) + g_m(i,q) - g_m(i,q-1)$, for $q \neq 0$.

The worst-case response time of instance q of a message sequence starting with subtype i of message m is given by:

$$R_m(i,q) = J_m + w_m(i,q) - q\,T_m + g_m(i,q+1) - g_m(i,q) \tag{6.27}$$

Thus, the worst-case response time, of an instance of m in a level-m busy period starting with subtype i of message m is given by:

$$R_m(i) = \max_{q=0..Q_m(i)-1}(R_m(i,q)) \tag{6.28}$$

Finally, the worst case response time of any instance of m in a level-m busy period is given by:

$$R_m = \max_{i=0..S-1}(R_m(i)) \tag{6.29}$$

6.4 Examples

We demonstrate the process of response time computation and the differences between our proposed approach and the-state-of-the-art with the help of two example message sets.

6.4.1 Example 1

This example illustrates our simpler analysis (Section 6.2) and compares it with the state-of-the art analysis.

Table 6.1 presents the attributes of the different messages in this example. The transmission times of the messages were selected among the values possible from Equation 6.9; ranging from 55 (for a 0-byte message) to 135 (for an 8-byte message), with a step of 10. All message jitters are zero.

The worst case response time of message 2 is computed as follows. The blocking term for message 2 is $B_2 =$ 105, which is the largest transmission time for all subtypes of message 3, the only message with lower priority.

Table 6.1: A set of multisized CAN messages, used in our first example. Column $\vec{C}_m$ contains the vector of the transmission times of the different message subtypes, for use by the multisized CAN analysis. Column C_m contains the worst-case transmission time of all message instances (a scalar), for use by the classic CAN analysis. Message parameters are measured in multiples of τ_{bit}.

message m	priority	$T_m = D_m$	$\vec{C_m}$	C_m
message 1	high	200	$\{75, 95, 65\}$	95
message 2	medium	350	$\{55, 75\}$	75
message 3	low	400	$\{105, 55\}$	105

The busy period window v_2, computed with recurrence (6.3) (for the standard analysis) and recurrence (6.13) (for the multisized message CAN analysis), respectively converges to 540, (6.30) and 350, (6.31).

$$\begin{aligned} v_2^5 &= B_2 + \left\lceil \frac{540}{T_2} \right\rceil * C_2 + \left\lceil \frac{540}{T_1} \right\rceil * C_1 \\ &= 105 + \left\lceil \frac{540}{350} \right\rceil * 75 + \left\lceil \frac{540}{200} \right\rceil * 95 \\ &= 105 + 2 * 75 + 3 * 95 = 540 \end{aligned} \tag{6.30}$$

$$\begin{aligned} v_2^3 = &\quad B_2 + G_2(350) + G_1(350) \\ = &\quad 105 + 75 + (75 + 95) = 350 \end{aligned} \tag{6.31}$$

In this particular case, the number of instances of message 2 that need to be analyzed (i.e., that are contained in its busy period) is 2 for the standard analysis, see (6.5), and 1 for the multisized CAN message analysis, see (6.17).

The computation of the response time of the first instance of message 2 requires its queueing delay $w_2(0)$ to be computed first, using recurrences (6.6) for the standard analysis and (6.18) for our simpler analysis, which converge to 295 (6.32) and 275 (6.33), respectively:

$$\begin{aligned} w_2^3(0) &= B_2 + 0 * C_2 + \left\lceil \frac{295 + 1}{T_1} \right\rceil C_1 \\ &= 105 + 2 * 95 = 295 \end{aligned} \tag{6.32}$$

$$\begin{aligned} w_2^3(0) &= B_2 + g_2(0) + G_1(295) \\ &= 105 + 0 + (75 + 95) = 275 \end{aligned} \tag{6.33}$$

Applying the response time equations, respectively (6.7) and (6.21), we obtain 370 ($> D_2$) and 350 ($= D_2$). Thus, message 2 is not schedulable under the standard analysis, whereas the single instance of message 2 is schedulable under the multisized CAN message analysis of Section 6.2.

In summary, the worst-case response time of message 2 using the analysis of Section 6.2 is 350 ($\leq D_2$) whereas using the classic analysis it is at least 370 ($> D_2$). This illustrates that, although the classic, state-

of-the-art analysis provides safe upper bounds for message transmission times and response time, it can be pessimistic. As a result a system that is actually schedulable might be declared unschedulable, as demonstrated with this example.

6.4.2 Example 2

The differences between the simpler analysis of Section 6.2 and the tighter, but more complex analysis of Section 6.3 can be more easily illustrated via a second example, with just two messages, one of which is multisized. Table 6.2 presents the attributes of those messages. All message jitters are zero. Recall also that $\tau_{bit} = 1$. Starting with our simpler analysis:

Table 6.2: The parameters of a message set used in Example 2. Column T_m and D_m specify the inter arrival time and deadline of each message, respectively. Vector of WCET estimates for multisized messages is presented in column $\vec{C_m}$, whereas corresponding scalar value used by all the state-of-the-art analysis is presented in column C.

message m	*priority*	T_m	D_m	$\overrightarrow{C_m}$	C_m
message A	*higher*	160	235	$\{95\}$	95
message B	*lower*	240	240	$\{65,135,55\}$	135

Message A has higher-priority, therefore it incurs no interference from message B. However, it has a blocking term B_A=$g_B(1)$=135 and it can suffer interference from previously released but yet undelivered instances of itself (given that D_A>T_A). The busy period v_A at this priority level is the solution to the recurrence $v_A^{n+1} = B_A + \lceil \frac{v_A^n}{T_A} \rceil C_A$, i.e., 420.

There exist $Q_A = \lceil v_a/T_A \rceil = \lceil 420/160 \rceil = 3$ instances of message A in this busy period of 420, and applying Equation 6.19, their response times are respectively $135+95 = 230$, $135+2*95-1*160 = 165$, and $135+3*95-2*160 = 100$. Therefore the message deadline $D_A = 235$ is met. Onwards with the analysis of the lower-priority message B:

This message has the lowest priority, so its blocking term B_B is zero. We will first analyse its schedulability using the simpler analysis of Section 6.2. From (6.13), the busy period at the priority level of message B is given by the solution to the recurrence $v_B^{n+1} = G_B(v_B^n) + G_A(v_B^n)$, initiated by $v_B^0 = g_B(1) = 135$:

$$v_B^1 = G_B(135) + G_A(135) = 135+95 = 230$$
$$v_B^2 = G_B(230) + G_A(230) = 135+2\cdot 95 = 325$$
$$v_B^3 = G_B(325) + G_A(325) = (135+65)+3\cdot 95 = 485$$
$$v_B^4 = G_B(485) + G_A(485) = (135+65+55)+4\cdot 95 = 635$$
$$v_B^5 = G_B(635) + G_A(635) = (135+65+55)+4\cdot 95 = \mathbf{635}$$

Therefore $v_B = 635$ and, from (6.17), there exist $Q_B = \lceil 635/240 \rceil = 3$ instances of message B in that busy period. Let us now find the time instants ($w_B(0)$, $w_B(1)$ and $w_B(2)$) when those message instances begin their

transmission by applying and solving (6.18), and from those, their response times, via (6.21):

$$\begin{aligned} w_B^0(0) &= 0 \\ w_B^1(0) &= g_B(0) + G_A(0+0+1) = 0+95 = 95 \\ w_B^2(0) &= g_B(0) + G_A(95+0+1) = 0+95 = 95 \\ \Rightarrow R_B(0) &= w_B(0) - 0*T_B + g_B(1) = 95+135 = 230 \end{aligned}$$

$$\begin{aligned} w_B^0(1) &= w_B^0(0) + g_B(1) - g_B(0) = 95+135-0 = 230 \\ w_B^1(1) &= g_B(1) + G_A(230+0+1) = 135+2\cdot 95 = 325 \\ w_B^2(1) &= g_B(1) + G_A(325+0+1) = 135+3\cdot 95 = 420 \\ w_B^3(1) &= g_B(1) + G_A(420+0+1) = 135+3\cdot 95 = 420 \\ \Rightarrow R_B(1) &= w_B(1) - 1*T_B + g_B(2) - g_B(1) \\ &= 420-240+200-135 = 245 \end{aligned}$$

Since $R_B(1) > D_B = 240$, message B is deemed unschedulable by the simpler analysis from Section 6.2. Let us now try the tighter analysis from Section 6.3. Note that because message A is not multisized, there is no difference between the two analyses. So, we consider only message B. From (6.24), the recurrence for the busy period at message B's priority level, for a message sequence starting with subtype i, is:

$$v_B^{n+1}(i) = G_B(i, t_B^n(i)) + G_A(t_B^n(i))$$

When the subtype of the sequence-initial message is $i = 0$, starting with $v_B^0(0) = g_B(0,1) = 65$:

$$\begin{aligned} v_B^1(0) &= G_B(0,65) + G_A(65) = 65+95 = 160 \\ v_B^2(0) &= G_B(0,160) + G_A(160) = 65+95 = \mathbf{160} \end{aligned}$$

Then $Q_B(0) = \lceil v_B(0)/T_0 \rceil = \lceil 160/240 \rceil = 1$, so there is a single instance of message B in this busy period – and since $v_B(0) \leq D_B$, it is schedulable.

When the subtype of the sequence-initial message is $i = 1$, starting with $v_B^0(1) = g_B(1,1) = 135$:

$$\begin{aligned} v_B^1(1) &= G_B(1,135) + G_A(135) = 135+95 = 230 \\ v_B^2(1) &= G_B(1,230) + G_A(230) = 135+2\cdot 95 = 325 \\ v_B^3(1) &= G_B(1,325) + G_A(325) = (135+55)+3\cdot 95 = 465 \\ v_B^4(1) &= G_B(1,465) + G_A(465) = (135+55)+3\cdot 95 = \mathbf{465} \end{aligned}$$

Then, $Q_B(1) = \lceil v_B(1)/T_1 \rceil = \lceil 465/240 \rceil = 2$ instances of message B are in this busy period. From (6.26), they start their transmissions at instants $w_B(1,0)$ and $w_B(1,1)$ computed below, allowing their response times to be derived using (6.27):

$$w_B^0(1,0) = G_A(0+0+1) = 95$$
$$w_B^1(1,0) = g_B(1,0) + G_A(95+0+1) = 0+95 = \mathbf{95}$$

$$\Rightarrow R_B(1,0) = w_B(1,0) - 0 \cdot T_B + g_B(1,1) - g_B(1,0)$$
$$= 95 + 135 - 0 = 230$$

$$w_B^0(1,1) = w_B^0(1,0) + g_B(1,1) - g_B(1,0)$$
$$= 95 + 135 - 0 = 230$$
$$w_B^1(1,1) = g_B(1,1) + G_A(230+0+1) = 135 + 2 \cdot 95 = 325$$
$$w_B^2(1,1) = g_B(1,1) + G_A(325+1) = 135 + 3 \cdot 95 = 420$$
$$w_B^3(1,1) = g_B(1,1) + G_A(420+0+1) = 135 + 3 \cdot 95 = \mathbf{420}$$

$$\Rightarrow R_B(1,1) = w_B(1,1) - 1 * T + g_B(1,2) - g_B(1,1) =$$
$$= 420 - 240 + 190 - 135 = 235$$

Therefore, both instances of message B in a busy period initiated by subtype $i = 1$ are schedulable.

Similarly reasoning, the busy period for a sequence initiated by a subtype-2 message is $v_B(2) = 150$. It contains a single instance of message B and its response time is equal to the above busy period $v_B(2)$, so it is also schedulable. Therefore, $R_B = \max(R_B(0,0), R_B(1,0), R_B(1,1), R_B(2,0)) = \max(160, 230, 235, 150) = 235 < D_B$. Hence, the system is schedulable, using the tighter analysis.

6.5 Background on mixed-queue networks

6.5.1 Analysis for priority queues

The analysis techniques for a network with only priority queues (Section 6.2) assume that sooner a message is queued, it goes into arbitration. However, this same assumption does not hold for FIFO queued messages. Instead, a FIFO message may have to wait for a maximum time of f_k before it can compete for access to the CAN bus. This additional jitter component in messages sent by WQN- or WQR-nodes affects the response time of messages that are sent by PQ-nodes. To take this into account, Davis and Navet [103] replace the J_m term in the equations in Section 6.1 with the sum $J_m + f_m$. More specifically, (6.3), the recurrence for computing the busy period of a level-m message, becomes:

$$v_m^{n+1} = B_m + \sum_{\forall k \in hep(m)} \left\lceil \frac{v_m^n + J_k + f_k}{T_k} \right\rceil C_k \tag{6.34}$$

and (6.6), the longest time from the start of the level-m busy period until instance q begins successful transmission, becomes:

$$w_m^{n+1}(q) = B_m + qC_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + f_k + \tau_{bit}}{T_k} \right\rceil C_k \tag{6.35}$$

Note that the number of instances of message m in the busy period, given by (6.5):

$$Q_m = \left\lceil \frac{v_m + J_m}{T_m} \right\rceil$$

needs not be modified.

The response time of instance q and of message m is computed as in [20], see (6.7) and (6.8).

6.5.2 Analysis for work-conserving queues

In this subsection we summarize the analysis of messages that are sent by nodes that do not use priority queues. Let m be such a message. Since its node does not use a priority queue, m may be queued after messages with lower priority. In particular, it may be queued after a message with the lowest priority, L_m, among all messages in m's group, $M(m)$, i.e. the messages sent by its node. To compute the worst-case response time of message m, [103] conservatively analyses all instances of message m that may arrive in a level-L_m busy period. Thus, the busy period considered in the analysis of message m sent by a WQN- or a WQR-node is given by the solution of the following recurrence:

$$v_m^{n+1} = B_{L_m} + \sum_{\forall k \in M(m)} \left\lceil \frac{v_m^n + J_k}{T_k} \right\rceil C_k + \sum_{\forall k \in hep(L_m) \wedge k \notin M(m)} \left\lceil \frac{v_m^n + J_k + f_k}{T_k} \right\rceil C_k \tag{6.36}$$

Note that the first summation accounts for all the message instances sent by m's node in a level-L_m. Immediately before the start of a level-L_m the transmitting queue at m's node is empty, therefore the buffering delay that any of those instances may incur is already taken into account by the transmitting time of the instances that cause those delays and therefore f_k is not added to J_k.

For the same reason, the number of instances of message m in the busy period is given by (6.5).

The longest time from the start of the level-L_m busy period until instance q (remember $q = 0$ for the first instance) begins successful transmission is given by:

$$w_m^{n+1}(q) = B_{L_m} + qC_m + I_m^*(q, w_m^n) + \sum_{\forall k \in hep(L_m) \wedge k \notin M(m)} \left\lceil \frac{w_m^n + J_k + f_k}{T_k} \right\rceil C_k \tag{6.37}$$

where $I_m^*(q,w)$ denotes the interference by instances of messages that are sent by m's node on instance q of message m during an interval of duration w starting at the beginning of a level-L_m busy-period. Depending on the queuing policy used by a node, [103] derives different bounds for $I_m^*(q,w)$, $I_m^{WQN}(q,w)$ and $I_m^{WQR}(q,w)$ for WQN-nodes and WQR-nodes, respectively as summarized below.

The expressions to compute the response times of each instance of message m, $R_m(q)$, and of all instances of m, R_m, respectively, (6.7) and (6.8), need no changes.

Interference in WQN-nodes In nodes with a WQN-policy (Section 3.3), instances of message m are sent in the order in which they are queued, and the time to transmit previous instances of the same message is already taken into account in the second term of the RHS of (6.37). Thus, what remains to be bounded is the interference by all instances of other messages. Because, a WQN-policy allows instances of other messages to be transmitted before any instance of message m, even if the latter was enqueued earlier, we get:

$$I_m^{WQN}(q,w) = \sum_{j \in M(m) \wedge j \neq m} \left\lceil \frac{w + J_j + \tau_{bit}}{T_j} \right\rceil C_j \tag{6.38}$$

Again, note that the buffering time is already accounted for by the transmitting time of the message instances that contribute to that time.

Interference in WQR-nodes In nodes with a WQR-policy (Section 3.3), instances of message m may not be sent in the order in which they are queued. In the worst case, all instances of message m that arrive in a level-L_m busy period before the start of transmission of instance q of message m are sent before the latter. Therefore, the interference by instances of messages from the same node becomes:

$$I_m^{WQR}(q,w) = \sum_{j \in M(m) \wedge j \neq m} \left\lceil \frac{w + J_j + \tau_{bit}}{T_j} \right\rceil C_j + max\left(0, \left\lceil \frac{w + J_m + \tau_{bit}}{T_m} \right\rceil - (q+1)\right) C_m \tag{6.39}$$

Note that both for WQN- and WQR-nodes, we need to consider the level-L_m busy period. Furthermore, the length of this busy period is the same for all messages in $M(m)$, thus for the analysis of messages of these nodes it is enough to compute only one busy-period per node, rather than a busy period per message.

Finally, [103] provides the following upper-bound for the buffering time of a message m belonging to a WQN- or WQR-node:

$$f_m = R_m - J_m - C_m \tag{6.40}$$

The buffering time of a message belonging to a priority-based queue is zero.

6.5.3 Schedulability test

The schedulability test in [103], shown in Algorithm 3, is essentially that from [24]. Note that in this test the inner loop is repeated until the buffering times stabilize, or the response time of some message exceeds its relative deadline. The reason for this is that the upper-bound for the buffering time given by (6.40) leads to a circularity between the response times when the priorities of messages of different queues are interleaved. I.e. let messages m and k belong to different queues and $k \in hp(L_m)$ and $m \in hp(L_k)$. In this case, R_m depends on f_k and therefore R_k, and R_k depends on f_m and therefore R_m. This solution is guaranteed to work because, as the authors of [24] point out, the response times are monotonically non-decreasing with the buffering times, and the buffering times are monotonically non-decreasing with the response times.

6.5.4 Schedulability test for adjacent priorities

The upper-bound on the buffering time given by (6.40) is very conservative. Essentially, it bounds the buffering time by assuming that messages sent by other nodes do not contribute to the response time.

Algorithm 3 Schedulability Test

```
1:  initialize: all f_m = 0, repeat = true
2:  while repeat do
3:      repeat = false
4:      for each priority m, highest first do
5:          Calculate R_m (using appropriate equations)
6:          if R_m > D_m then
7:              return unschedulable.
8:          end if
9:          if m belongs to a work-conserving queue then
10:             if f_m ≠ R_m − J_m − C_m then
11:                 f_m = R_m − J_m − C_m
12:                 repeat = true
13:             end if
14:         end if
15:     end for
16: end while
17: return schedulable
```

To avoid this pessimism, [103] proposes the use of an adjacent priority assignment for messages of the same group, when the respective node queuing policy is not priority based. I.e., to assign adjacent priorities to messages belonging to the same node when its queuing policy is not priority based, thus leading to priority bands. If this is the case, the messages that may lead to buffering time of higher priority message k also have higher priority than the message under analysis. Thus, the transmission time of these messages will be accounted for as interference, and we can make $f_k = 0$.

The assignment of adjacent priorities also simplifies the schedulability test shown in Algorithm 3; it becomes essentialy the inner loop, but without lines 9-14.

The use of an adjacent priority assignment does not reduce the schedulability according to Algorithm 3. Indeed, Davis and Navet in [103] show that, if there is a priority assignment that is schedulable according to Algorithm 3, then there is also an adjacent priority assignment that is schedulable under the same algorithm. This is done by appropriately changing the arguments provided in the proof of a similar result, i.e. Theorem 1, in [24]. This proof provides an algorithm for construction of an adjacent priority assignment from a schedulable priority assignment as per Algorithm 3, and argues that the new assignment must still be schedulable according Algorithm 3. The key argument is that given a relative order $\mathscr{O}$ of the messages corresponding to the schedulable priority assignment, if we move a message m to a position immediately after the message of the same group with a priority immediately below it, this new order is still schedulable. The reason is that this change in the message ordering does not increase the right-hand-side (RHS) of any of the equations used to compute the response time of the messages whose position in the order is affected by this move, i.e. both the message that is moved, m, and the messages whose relative priority with respect to m changes.

6.6 Analysis for FIFO queues

Although queues with FIFO policy can be modeled as using a WQN policy, it may be possible to reduce the pessimism by modeling more accurately a FIFO policy.

In [24], Davis et al. analyze the CAN-bus with FIFO queues under the assumption of constrained deadlines. This analysis does not use the concept of level-m busy period. Instead, they bound the push-through interference as a blocking term. In particular, they claim that this interference cannot be larger than C_m, therefore they use as blocking term $max(B_m, C_m)$ rather than B_m.

Another feature of their analysis, that stems from the assumption of constrained deadlines is that when analyzing the response time of a message m, in the worst-case there is only one instance of each message in the queue. Indeed, if that were not the case, then a second instance would have arrived before the previous was transmitted, and therefore the deadline of the latter would have been missed. A consequence of this is that this analysis is what they call as FIFO symmetric, i.e. leads to the same response-time upper bound to all the messages sent through the same queue.

Our analysis follows the approach of Davis et al. in [24], but uses the concept of priority level busy period to generalize to arbitrary deadlines.

The analysis for priority queues in Section 6.5.1 still applies. All we need to do is to change the analysis for the WQN policy.

The main difference concerns the interference of messages in the same group. Interference by messages from other nodes is modeled in the same way. I.e. we assume that the transmission of message m may be interfered by the transmission by other nodes of messages whose priority is higher than L_m. Thus, the recurrences for computing both the level-m busy period, (6.36), and the longest time to start transmission of message instance q, (6.37), need no changes. We need only to derive $I^*_m(q,w)$ for queues with FIFO policy.

Whereas with the WQN-policy instance q of message m may suffer the interference by instances of all other messages in $M(m)$ that arrive before the start of its transmission, with a FIFO-policy, instance q of message m may suffer the interference by other messages in $M(m)$ that are added to the queue earlier. Therefore:

$$I_m^{FIFO}(q,w) = \sum_{j \in M(m) \wedge j \neq m} \left\lceil \frac{q \cdot T_m + J_m + J_j + \tau_{bit}}{T_j} \right\rceil C_j \tag{6.41}$$

because instance q of message m can be added to the queue at the latest $q \cdot T_m + J_m$ after the start of the level-L_m busy period.

Note that $I_m^{FIFO}(q,w)$ is independent of w, nevertheless we keep the latter as the second argument, so that (6.37) still applies.

6.7 Analysis for mixed-queue network with multisized messages

We extend the analysis in Section 6.2 to CAN systems whose nodes may use different queuing policies. We present the extended analysis for each of the queuing policies in its respective subsection. A similar extension of the tighter analysis in Section 6.3 can be found in the Appendix.

In this analysis, we apply the same devices as Davis and Navet in [103]. More specifically, let m be the message under analysis:

1. For non-priority-based policies, the analysis considers all instances of message m in a level-L_m busy period, where L_m is the lowest priority of all messages in $M(m)$, i.e. sent via the same queue as m.

2. We add a term f_k to the arrival jitter, J_k, of message k with higher priority than m and belonging to a different queue from m, to model the additional delay message k may incur, because of its queue policy, from its insertion in its queue to its participation in the CAN-bus priority-based arbitration scheme.

3. For messages belonging to nodes different from PQ-nodes, we upper-bound separately the interference by messages in m's group, and the interference by messages belonging to other nodes.

Note that 1) by itself does not add any pessimism to the analysis. The issue is that (an instance of) message m belonging to a node that does not use a priority-based policy may suffer the interference by a lower priority message transmitted via a different queue, because the latter has higher priority than another message that is ahead of m in its queue. This priority inversion is independent of whether messages, both the message under analysis and the other messages, are multisized or not: all instances of a message have the same priority. Thus, in both cases, by considering all instances of message m in a level-L_m busy period, we ensure that the analysis is safe.

Likewise, independently of message k being multisized or not, it may suffer additional delay because its queue uses a policy that is not priority-based. As Davis and Navet [103] observed, messages transmitted via priority queues can enter the priority-based arbitration scheme as soon as they are added to their respective queue, therefore, for those messages, $f_k = 0$.

Finally, for a message m that is sent via a queue that is not priority-based, the interference caused by messages in m's group, may be different from the interference by messages belonging to other nodes, thus we may need to use different expressions. Again, this is independent of whether messages are multisized or not.

6.7.1 Analysis for priority queues

For messages sent via priority queues, we need to apply only the second device discussed above, i.e. we need only to consider the jitters induced by the policies of the queues used for the transmission of higher priority messages. As mentioned above this is modeled by adding f_k to J_k, therefore (6.13), the recurrence for computing an upper bound of the length of level-m busy period, becomes:

$$v_m^{n+1} = B_m + \sum_{\forall k \in hep(m)} G_k(v_m^n + J_k + f_k) \tag{6.42}$$

where:

$$B_m = \max_{i \in lp(m)} g_i(1)$$

is the same as (6.14), in Section 6.2, and the functions G and g are defined by (6.12) and (6.11), respectively. A possible starting value is $v_m^0 = B_m + g_m(1)$.

Likewise, (6.18), the longest time from the start of the level-m busy period until instance q begins successful transmission, becomes:

$$w_m^{n+1}(q) = B_m + g_m(q) + \sum_{\forall k \in hp(m)} G_k(w_m^n(q) + J_k + f_k + \tau_{bit}) \tag{6.43}$$

where q ranges from 0 to $Q-1$, and Q is still given by (6.17). Possible starting values are $w_m^0(q) = B_m$, for $q = 0$, and $w_m^0(q) = w_m(q-1) + g_m(q) - g_m(q-1)$, otherwise.

The response time of instance q is still given by (6.21), which we repeat here to ease the reading:

$$R_m(q) = J_m + w_m(q) - q\,T_m + g_m(q+1) - g_m(q)$$

and the worst-case response time of message m is:

$$R_m = \max_{q=0..Q_m-1}(R_m(q))$$

as in [20] (see (6.8)).

6.7.2 Analysis for work-conserving queues

To extend the analysis in Section 6.2 to work-conserving queues we apply the three devices used by Davis and Navet. Like in [103], we isolate the interference by messages sent via the same queue in a term $I_m^*(q,w)$ that we will derive for each of the 3 policies under consideration. This approach leads to a generic analysis that we now present.

As already justified, for messages transmitted via work-conserving queues that are not priority based, we need to consider all instances of the message under analysis m that may be added to the queue during a level-L_m busy period, where L_m is the lowest priority of all messages transmitted via the same queue as m. Therefore, the busy period for the analysis of message m can be obtained by combining (6.13) and (6.36) as follows:

$$v_m^{n+1} = B_{L_m} + \sum_{k \in M(m)} G_k(v_m^n + J_k) + \sum_{\forall k \in hp(L_m) \wedge k \notin M(m)} G_k(v_m^n + J_k + f_k) \tag{6.44}$$

Note that in the transmission time of the messages in $M(m)$ we do not take into account f_k for the reasons already presented in Section 6.5. Since for all messages in $M(m)$ we need to consider the level-L_m busy period, we need to solve Recurrence (6.44) for only one message in $M(m)$, e.g. L_m, not for all messages.

Separating the interference by messages in $M(m)$ from that by other messages, we combine (6.18) with (6.37) so that the time to start the transmission of instance q of message m becomes:

$$w_m^{n+1}(q) = B_{L_m} + g_m(q) + I_m^*(q, w_m^n(q)) + \sum_{\forall k \in hp(m) \wedge k \notin M(m)} G_k(w_m^n(q) + J_k + f_k + \tau_{bit}) \tag{6.45}$$

where q ranges from 0 to $Q_m - 1$, and Q_m is still given by (6.17). Possible starting values are $w_m^0(q) = B_{L_m}$, for $q = 0$ and $w_m^0(q) = w_m(q-1) + g_m(q) - g_m(q-1)$, otherwise.

The equations to compute $R_m(q)$ and R_m are the same as in Section 6.2, already reproduced in the previous subsection.

We now derive the expressions of $I^*_m(q,w)$ for the 3 non-priority-based policies.

Interference by same-node messages in WQN-nodes With this policy, any instance of a message different from m that arrives before instance q of message m starts its transmission, may be transmitted before. Therefore, we adapt (6.38) as follows:

$$I_m^{WQN}(q,w) = \sum_{j \in M(m) \wedge j \neq m} G_j(w + J_j + \tau_{bit}) \tag{6.46}$$

That is, for each messsage j in m's group that is different from m, we consider the maximum transmission time of all instances of that message in time w.

Interference by same-node messages in WQR-nodes Again, we adapt the expression from Davis and Navet in [103], reproduced in (6.39), as follows:

$$I_m^{WQR}(q,w) = \sum_{j \in M(m) \wedge j \neq m} G_j(w + J_j + \tau_{bit}) + max(0, G_m(w + J_m + \tau_{bit}) - g_m(q+1))) \tag{6.47}$$

The second term on the RHS upper-bounds the interference by other instances of message m. Essentially, all instances that may arrive in interval w may interfere with instance q. However, the transmission time of the q instances that arrive before instance q is already accounted for in the second term of (6.45). Furthermore, we exclude the transmission time of instance q itself (remember that q ranges from 0 to $Q_m - 1$). Note that using $g(q+1)$ is safe, because 1) (6.47) is used in the RHS of (6.44), and the latter's second term already accounts for $g(q)$; 2) (6.21), which provides the response time of instance q of message m already takes into account the transmission time of that instance.

Interference by same-node messages in FIFO-nodes In this case, we adapt (6.41) as follows:

$$I_m^{FIFO}(q,w) = \sum_{j \in M(m) \wedge j \neq m} G_j(q \cdot T_m + J_m + J_j + \tau_{bit}) \tag{6.48}$$

6.7.3 Buffering time upper-bound

To upper-bound the buffering time, we use the same assumption as [24] and [103], i.e. that the computed response time is caused only by messages belonging to m's group. However, the transmission time of the different subtypes of message m may be different. Therefore, we adapt the buffering time upper-bound for work-conserving queues with a non-priority policy (6.40) from [103] as follows:

$$f_m = R_m - J_m - (g_m(S) - g_m(S-1)) \tag{6.49}$$

i.e. we assume that the computed response time occurs for the smallest subtype of message m, thus maximizing the buffering time. We recall that for priority-based queues, $f_m = 0$.

6.7.4 Schedulability test

We adapt the schedulability test of [103], Algorithm 3, by using the equations of the analysis derived in this section. Although these equations still lead to a possible circularity in the response time dependencies for non-adjacent priority assignments, both R_m has a monotonically non-decreasing dependence on f_m and f_m has a monotonically non-decreasing dependence on R_m. Thus Algorithm 3 eventually decides whether or not the system is schedulable.

6.7.5 Adjacent priority assignment

The use of an adjacent priority assignment for the analysis in this section has essentially the same advantages as those described in both [24] and [103].

First, we can use $f_k = 0$ not only for messages belonging to PQ-nodes but also for messages belonging to other nodes. Indeed, the arguments presented in [24] and [103] are still applicable to multisized messages: with adjacent priority assignment, except for the first message, which is already considered in the blocking term, no message that has a lower priority and belongs to a different node may delay the transmission of a message with higher priority. As a consequence, the schedulability test becomes simpler.

Second, the arguments used in Theorem 1 of both [24] and [103] still apply to the analysis presented in this section. Thus, for the schedulability test in Algorithm 3, the use of an adjacent priority assignment does not necessarily mean a sub-optimal priority assignment.

Third, OPA-FP/WQ priority assignment presented in [103] that is based in the OPA-FP/FIFO in [24], a variation of Audsley's Optimal Priority Assignment algorithm [31], is still optimal for the schedulability test in this section as it satisfies the 3 conditions in [24], which are a restatement of the 3 conditions in [115].

We may need to provide a more detailed justification of why the 3 conditions in [24] are still satisfied, if we use the OPA-FP/WQ priority assignment in our experimental evaluation.

6.8 Experimental evaluation

We implemented our multisized message schedulability analysis (Section 6.7) using a Java tool [109] and compared the maximum achievable utilization with the existing analysis (Section 6.5) with FIFO queue scheduling policy. For evaluating the analysis for mixed-queue networks, we consider five different configurations, as done in [21]. In configuration #1, all the nodes use priority queue scheduling policy. Whereas for configurations #2, #3, #4 and #5 fraction of FIFO nodes is increased from $1/4$, $1/2$, $3/4$ to 1, respectively. More details about these configurations and factors used in our experiments are summarized in Table 6.4. We used TDMPO and TDMPO-FP/FIFO priority assignment [21] for network with only PQ nodes (Conf. #1) and with mixed-queue or all-FIFO nodes network (conf. #2 to #5), respectively. In TDMPO algorithm priorities are assigned based on the transmission deadline of each messages. A TDMPO-FP/FIFO assign the priority bands to different priority queued messages and FIFO-groups (group of messages assigned to each FIFO node) by their transmission deadline. For FIFO-group messages priority band is assigned based on the shortest transmission deadline of any message in that group. As shown in Table 6.3, our experimental setup considers different nodes and message set

size configurations. It is assumed that all the nodes are connected via a single shared, connected bus. To evaluate the performance of different configurations for our proposed analysis, 10,000 message sets are generated as follows:

1. The period of each message was generated in the range 10-1000 ms using a log-uniform distribution.
2. The length of message cycle for each m_i in the message set is selected between 1 and 5 using uniform random distribution.
3. The payload size of the largest subtype of every message is assumed to be 8 bytes. For each of the other subtypes, it is selected in the range $[1,\ 7]$ using uniform random distribution.
4. We assume messages with 11 bit identifiers, and use Eq. (6.9) to compute the WCTT of each message subtype.
5. The jitter of each message was selected randomly between 2.5 ms and 5 ms.
6. Messages are assigned randomly to nodes, and all nodes are assigned the same number of messages as each other. All the messages are equally divided among different nodes, and it is done randomly.

Since the existing analysis assumes that all instances of a message with the same id have the same size, for each message we use the maximum size of all message subtypes. I.e., in the existing analysis, we assume that all the instances of all messages have a payload with 8 bytes. As in [21] and [103] we use the maximum achievable utilization as an evaluation metric. It was computed for each message set using a binary search algorithm in combination with the schedulability analysis. For each message set, and a given analysis technique, the value for the bit-time is modified to the maximum value for which the message set is schedulable. This means that for an initially schedulable message set the bit-time is potentially increased, and for an initially unschedulable message set, it is decreased. The maximum feasible bit-time is calculated with a granularity of 1 *ns*. The maximum achievable utilization of a message set is computed using the maximum bit-time determined by binary search. More specifically, we use the maximum bit-time to compute the $C_{i,j}$ of each message subtype, which is then used in Eq. 6.9 to compute the maximum achievable utilization for the message set. Please note that for both analyses, i.e. the analysis of multisized messages and the analysis for single sized messages, we use the WCTT of each subtype of each multisized message. I.e., to evaluate the existing analysis, we use the payload of the multisized messages, not the 8-byte payload assumed by the analysis.

Finally, we used mean of these maximum achievable utilization values computed for each analysis over a set of 10000 message sets to find the difference between proposed and existing and refer to it as mean of maximum bus utilization difference.

$$U(M) = \sum_{i=1}^{n} \frac{1}{S_i} \sum_{j=0}^{S_i-1} \frac{C_{i,j}}{T_i} \tag{6.50}$$

To present the results, we plot the frequency of the different computed utilizations with a bin size of 1%. For example, the 50% bin, counts the number of message sets with maximum achievable utilization in the range from 50.00% to 50.99%.

Table 6.4: Different configuration considered for the analysis.

Configuration	Fraction of FIFO nodes	Priority assignment
#1	All PQ	$TDMPO$
#2	$1/4$ FQ	$TDMPO-FP/FIFO$
#3	$1/2$ FQ	$TDMPO-FP/FIFO$
#4	$3/4$ FQ	$TDMPO-FP/FIFO$
#5	All FQ	$TDMPO-FP/FIFO$

Table 6.3: Overview of the parameters.

Parameters	*Values*
Number of nodes	$\{8, 16, 24\}$
Number of messages	$\{40, 80, 120\}$
Jitter	$[2.5\,ms,\ 5\,ms]$
Inter arrival time	$[10\,ms,\ 1\,s]$

6.8.1 Results

In this section, we compare proposed multisized response time analyses techniques with the state of the art approaches. Two different set of experiments are performed (i.e., Message set size variation and node set size variation). In the message set size variation experiment a CAN network with 8 nodes is considered. However, the messages set size is varied from 40, 80 to 120. We also compared the proposed analysis with the existing analysis technique for different number of nodes in the network. For these set of experiment number of number of messages assigned to each node is kept to 10. However, the number of nodes in the network are varied from 8, 16 to 24.

6.8.2 Message set size variation

Figure 6.1 presents the maximum achievable utilization for the proposed schedulability analysis (solid line) and the respective state-of-the-art analysis (dotted line), which consider standard single-size messages. With the increase in the number of FIFO queue nodes, the overall maximum achievable utilization decreases, and so does the difference between the proposed and existing analysis.

With all 8 nodes using priority policy, the mean of maximum achievable utilization difference was 10.41%. With a quarter of nodes using FIFO queues, this difference reduces to 3.05%. For half and three quarter of nodes using FIFO queue this difference is reduced to 1.56% and 1.01% respectively. Finally, for all the nodes using FIFO queues, the difference is reduced to 0.73%.

We also repeated the experiments for sets of 40 and 120 messages that are uniformly distributed among the 8 node. These results are depicted in Figures 6.2 and 6.3, respectively, and summarized in Table 6.5. The same pattern for the improvement in the mean maximum achievable bus utilization observed for message sets with 80 messages can be observed for message sets of sizes 40 and 120. For networks with only PQ-nodes (conf. #1), as we increase the message set size from 40 to 80 and to 120, overall maximum achievable utilization increases, therefore difference between the proposed and existing schedulability analysis also increases. However, for

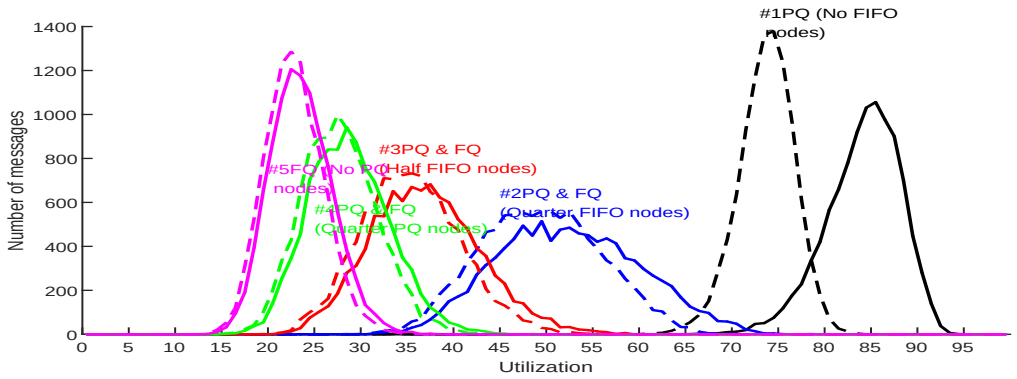


Figure 6.1: Frequency distribution of maximum bus utilization (8 nodes, 10,000 message sets with 80 messages each)

configurations with FIFO nodes (conf. #2 to #5), the difference between the average maximum achievable utilizations decreases with the size of the message set. As we increase number of FIFO nodes in the network number of total messages hosted by FIFO nodes also increase which leads to increase in priority inversion and decrease in maximum achievable utilization. This decrease in maximum achievable utilization difference also leads to reduction in difference between proposed and existing analysis.

6.8.3 Nodes set size variation

We also compared the proposed schedulability analysis with the existing approaches for a CAN system with 16 and 24 nodes with a message set size of 160 and 240, respectively. These results are plotted for different configurations in Figures 6.5 and 6.6, respectively, and summarized in Table 6.6. As can be seen in Table 6.6, the mean of maximum achievable utilization difference was slightly increased as compared to 8 nodes and 80 messages configuration. Though the average number of messages assigned to each node was same (10 messages per node) however, as reported in [21], more messages with smaller utilization reduce the impact of non-preemptive transmission, and therefore schedulability improves.

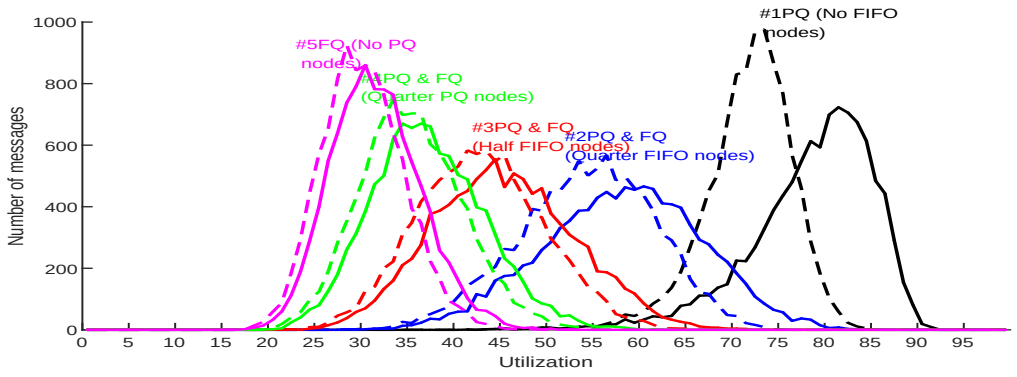


Figure 6.2: Frequency distribution of maximum bus utilization(8 nodes, 10,000 message sets with 40 messages each)

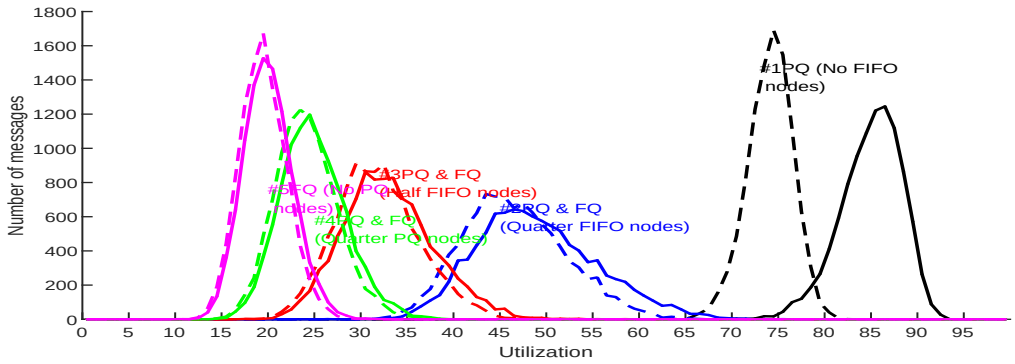


Figure 6.3: Frequency distribution of maximum bus utilization (8 nodes, 10,000 message sets with 120 messages each)

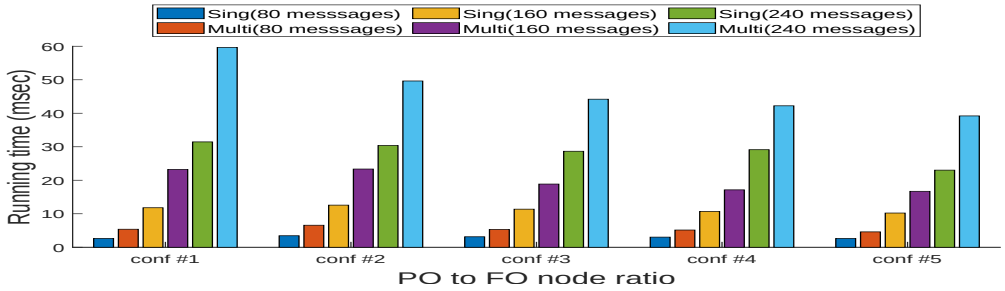


Figure 6.4: Average runtime of the different analyses per message set.

Finally, the running time of the proposed analysis is also compared with the existing approaches for CAN with different node set sizes. The experiments were conducted on a platform with 32 GB of RAM and with eight Intel(R) Xeon(R) E5-2420 v2 CPUs, each core with a maximum frequency of 2.20Ghz. We used Linux Mint V 18.3 Sylvia OS with Java OpenJDK 11.0.21. Figure 6.4 presents the average runtime per message set of the different analyses for different number of nodes and different configurations. As expected, the simulation time of both analyses increases with the increase in the number of nodes. Also, our proposed analysis for mixed queue CAN and multisized messages takes longer than the existing analysis.

Table 6.5: Difference of mean of max bus utilization for different message set size

Config.	Node type	Priority ordering	Mean of improv. on the max bus utilization (%).		
			$n = 40$	$n = 80$	$n = 120$
#1	All PQ	TDMPO	6.89	10.41	10.88
#2	1/4FQ, 3/4 PQ	TDMPO-FP/FIFO	3.84	3.05	2.43
#3	1/2 FQ, 1/2 FQ	TDMPO-FP/FIFO	2.53	1.56	1.18
#4	3/4 FQ, 1/4 FQ	TDMPO-FP/FIFO	1.81	1.01	0.73
#5	All FQ	TDMPO-FP/FIFO	1.38	0.73	0.51

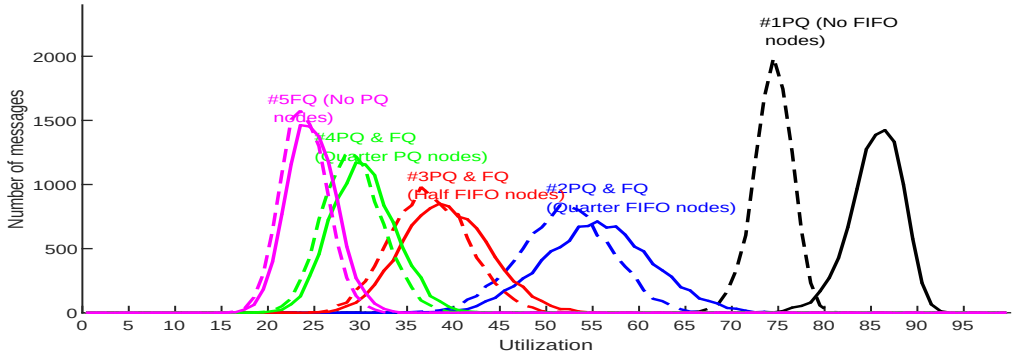


Figure 6.5: Frequency distribution of maximum bus utilization (16 nodes, 10,000 message sets with 160 messages each)

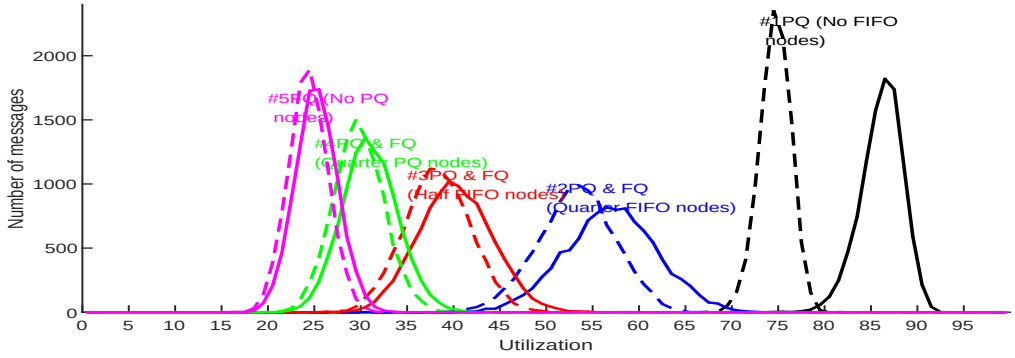


Figure 6.6: Frequency distribution of maximum bus utilization (24 nodes, 10,000 message sets with 240 messages each)

Table 6.6: Mean of max utilization difference for varying number of nodes and messages

Config.	Node type	Priority ordering	Mean of max bus util diff. (%)		
			8 nodes	16 nodes	24 nodes
#1	All PQ	TDMPO	10.41	11.15	11.46
#2	1/4FQ, 3/4 PQ	TDMPO-FP/FIFO	3.05	3.53	3.75
#3	1/2 FQ, 1/2 FQ	TDMPO-FP/FIFO	1.56	1.83	1.93
#4	3/4 FQ, 1/4 FQ	TDMPO-FP/FIFO	1.01	1.17	1.23
#5	All FQ	TDMPO-FP/FIFO	0.73	0.84	0.88

6.9 Chapter summary

In this chapter, we showed how to provide less pessimistic, but always safe, estimates on the worst-case response times of the CAN messages, by (i) expressing in a more fine-grained manner the transmission request requirements of CAN messages, without any modification to the CAN protocol and (ii) devising schedulability analyses that are able to leverage the additional information. This can make the difference between a system erroneously being deemed unschedulable or being proven schedulable. Furthermore, it can help safely accommodate additional functionality (and associated message traffic) in an existing system, by safely improving the attainable utilization. In this regards, initially we developed schedulability analysis technique for CAN with PQ nodes hosting multisized messages. Afterward, we extended the proposed analysis for a more general mixed-queue network, where some nodes use priority queue scheduling while others use a different policy, namely FIFO and work conserving queues that allow message reordering. The experimental evaluation of the proposed analyses with synthetic message sets demonstrate an improvement of up to 10% in terms of maximum achievable utilization difference.

Chapter 7

Conclusions and future directions

This chapter concludes this thesis by looking at our contributions and how they validate the Thesis Statement. It also presents some possible future directions.

7.1 Contributions, in Relation to the Thesis Statement

Already in the Introduction (Chapter 1), we noted how existing timing analysis techniques for mixed-criticality systems, with or without shared system resources and memory arbitration, can be too pessimistic, if periodic variation patterns in worst-case execution times are not taken into consideration. Similarly for message payloads on Controller Area Networks.

In our survey of the the relevant state-of-the-art literature (Chapter 2), among other things, we identified existing models and techniques that we would extend – namely, the different variants of Vestal's mixed criticality model, the multiframe real-time task model, the corpus of work on multicore systems with per-core regulation of access to shared memory, and the timing analysis techniques for CAN networks. Those elements inspired our new system models, introduced in Chapter 3: the *multiframe mixed-criticality model* and the multisized CAN message model.

In Chapter 4, we provided analysis techniques for the static and adaptive variants of the new multiframe mixed-criticality model, where each task is represented with a vector of periodically repeating WCET estimates per criticality level. Those techniques applied to uniprocessor systems and, initially, just to constrained-deadline tasks, but were subsequently generalized to arbitrary deadlines. Our experiments with synthetic task sets demonstrated notable reduction in pessimism, compared to techniques agnostic to the period variations in task WCETs. This supports the Thesis Statement.

In Chapter 5, we built up on the system model and techniques previously derived for uniprocessors, and extended them to multicore platforms, where the different cores access memory through a shared memory controller, access to which is regulated. Under this memory bandwidth regulated multiframe mixed-criticality system model, each task is represented, by a vector of tuples per criticality level – the elements of a tuple denoting, respectively, worst-case estimates for processor-based execution and time accessing memory. Again, our techniques were shown, in the experiments to be considerably less pessimistic (equivalently: more efficient

in utilizing system resources), compared to techniques oblivious to the patterns of variation in the worst-case load (in terms of processor-based execution time and memory access requirements) of each task. These findings also directly support the Thesis Statement.

In Chapter 6, we formulated our schedulability analysis techniques for the multisized CAN messages. Under this model, a message is characterized by a vector of transmission times, analogously to the vector of WCETs used to describe a multiframe real-time task. We initally presented analysis for CAN implementations with a (pure) priority queue scheduling policy. Subsequently, we presented analysis for mixed-queue implementations, where some nodes use priority queues whereas other use other work-conserving queue policies, e.g., FIFO. The evaluation and comparison of the proposed schedulability analyses with the existing approaches, via synthetic task sets, demonstrated significant improvement in terms of average maximum achievable utilization. Although our new proposed analysis technique always perform better than existing approaches in that regard, the degree of relative improvement becomes smaller as the number of FIFO nodes in the network increases. In any case though, the experimental findings, once again support the Thesis Statement, with respect to Controller Area Networks.

7.2 Limitations and Possible future directions

7.2.1 Combining Vestal's mixed-criticality model and the Predictable execution task model

In the design of critical real-time embedded systems, predictability in timing behavior and in system resource usage is necessary. The schedulability analysis techniques presented in Chapter 4 can derive the appropriate timing safety guarantees for each task without using more conservative estimates than needed. However, the application execution model does not consider the memory requests during any specific phase of task execution. This makes the estimation of contention over shared memory resources potentially more complex and their usage potentially less efficient than under the PREM model, which explicitly separates computation and accessing of memory requests into difference phases.

Specifically, the 2-phase PREM model, via compiler support, first fetches from memory (into the cache) all the locations that a task will access, and only subsequently proceeds with computation. This removes a lot of the uncertainty in WCET estimation stemming from the cache state and memory access delays, leading to better predictability and tighter WCET estimates. In a short paper [122], we explore different possibilities for how Vestal's model and the PREM model could be combined (PREM-MCS). For example, in the following illustration we demonstrated how an overrunning task, executing over core $P0$, will impact the execution of other tasks on the same and other cores, which are in their execution phase (7.1) or memory phase (Figure 7.2). More details about this example and some other models can be found in [122]. In that work, we also focus on the semantics of multiple (static or probabilistic) per-task estimates of processor computation time and number of memory accesses, how these can be derived, the associated compiler and O/S support required, and its impact over analysis techniques. However, the analysis for analytically deriving response time bounds and an evaluation study for the different PREM-MCS model variants presented still need to be developed.

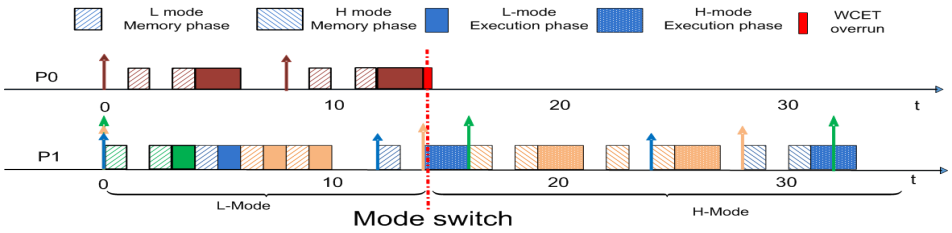


Figure 7.1: Sample schedule of example task set when mode switch happen during computation phase of task.

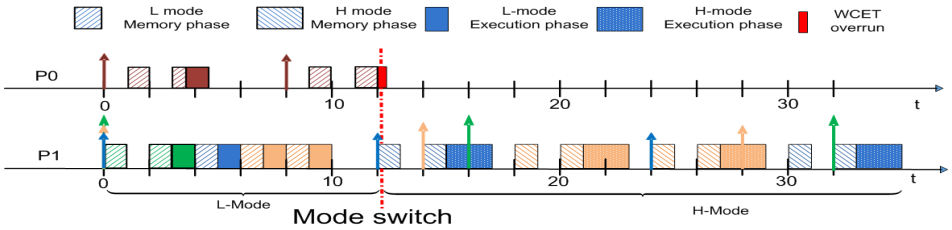


Figure 7.2: Sample schedule of example task set when mode switch happen during memory phase of task.

7.2.2 Mixed-criticality Scheduling for the Three-Phase Task Model

The three-phase task model, also known as *Acquisition-Execution-Restitution* (AER) is related to the PREM model. Their main difference is that a task is structured into three phases. Again, to better deal with the contention at the shared resource level, AER separates the memory and computation parts of the application. This task model is positioned for critical applications. As with PREM, we envision the combination of AER with Vestal's mixed-criticality model, that would inherit the strengths of both, as an interesting future direction. This would requires the development of new system models, improved tasks priority assignment methods, and schedulability analysis techniques accounting for the different phases of an application.

7.2.3 Mixed-criticality systems over MPSoCs with multiple memory controllers

Increasing bandwidth requirements have brought about architectures with multiple memory controllers, that access different memory regions and are potentially shared among all cores. Memory-intensive applications such as autonomous driving in the automotive sector and video/radar applications in the avionics domain can greatly benefit from this architectural feature, to satisfy their memory needs.

The schedulability analysis techniques for multiframe mixed-criticality multicore systems consider architectures with just one memory controller, which is shared among all the cores. Therefore the stall and response time bounds, provided in Chapter 5 of this thesis, do not hold for architectures with multiple memory controller. This requires the development of new worst-case memory regulation stall computation techniques and updated schedulability analysis which incorporate this stall, to ensure the safe deployment of mixed-criticality systems over such platforms. Different assignment configurations for the memory controllers can also be explored, to optimize for performance and improved schedulability, all while ensuring isolation among tasks of different criticality levels. Additionally, different memory bandwidth regulation and assignment techniques provide another

interesting direction for these systems. The different applications would first be profiled for their memory footprint and later that profile would guide the static or dynamic memory bandwidth allocation, memory mapping and partitioning.

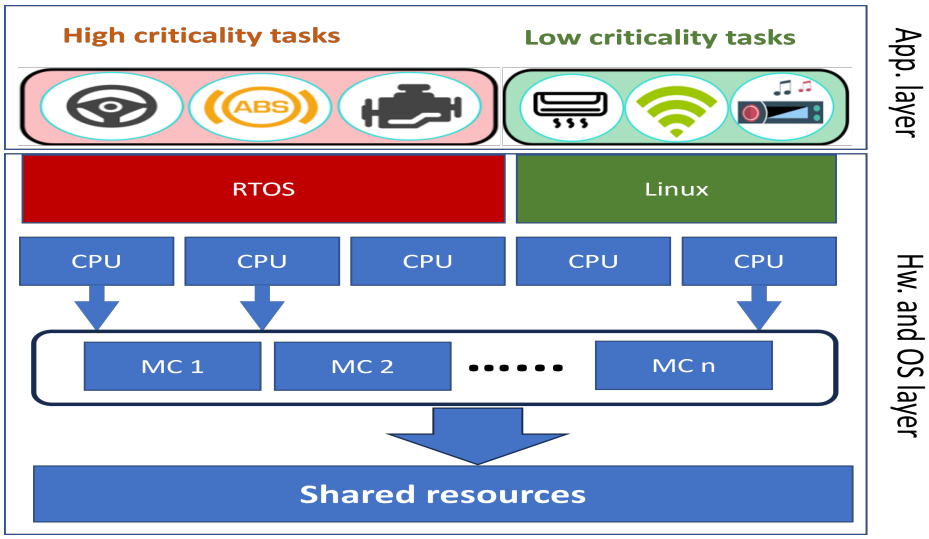


Figure 7.3: MPSoCs with multiple memory controller hosting mixed-criticality systems.

Bibliography

[1] C. Buckl, A. Camek, G. Kainz, C. Simon, L. Mercep, H. Stähle, and A. Knoll, "The software car: Building ict architectures for future electric vehicles," in 2012 IEEE International Electric Vehicle Conference, pp. 1–8, March 2012.

[2] R. F. S. 167, Software considerations in airborne systems and equipment certification. RTCA, Incorporated, 1992.

[3] A. K. Mok and D. Chen, "A multiframe model for real-time tasks," IEEE Transactions on Software Engineering, vol. 23, pp. 635–645, Oct 1997.

[4] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in Proceedings of the 28th IEEE Real-Time Systems Symposium, pp. 239–243, IEEE, 2007.

[5] S. Baruah and A. Burns, "Implementing mixed criticality systems in Ada," in Proceedings of the 16th Ada-Europe Conference, pp. 174–188, 2011.

[6] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in Proceedings of the 32nd IEEE Real-Time Systems Symposium, pp. 34–43, 2011.

[7] D. Le Gall, "Mpeg: A video compression standard for multimedia applications," Communications of the ACM, vol. 34, pp. 46–58, Apr. 1991.

[8] AERONAUTICAL RADIO, INC., ARINC specification 818-2 Avionics Digital Video Bus (ADVB) High Data Rate, 818-2 ed., 2013.

[9] C. Bailey, A. Burns, A. Wellings, and C. Forsyth, "Keynote paper: A performance analysis of a hard real-time system," Control Engineering Practice, vol. 3, no. 4, pp. 447–464, 1995.

[10] J. P. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in Proceedings of the 10th IEEE Real-Time Systems Symposium, pp. 166–171, 1989.

[11] S. K. Baruah and A. Mok, "Static-priority scheduling of multiframe tasks," in Proceedings of the 11th Euromicro Conference on Real-Time Systems, pp. 38–45, June 1999.

[12] A. Zuhily and A. Burns, "Exact scheduling analysis of non-accumulatively monotonic multiframe tasks," Journal of Real–Time Systems, vol. 43, no. 2, pp. 119–146, 2009.

[13] I. Hussain, M. A. Awan, P. F. Souto, K. Bletsas, B. Akesson, and E. Tovar, "Response time analysis of multiframe mixed-criticality systems," in Proceedings of the 27th Conference Real-Time and Networked Systems, pp. 8–18, 2019.

[14] I. Hussain, M. A. Awan, P. F. Souto, K. Bletsas, B. Akesson, and E. Tovar, "Response time analysis of multiframe mixed-criticality systems with arbitrary deadlines," Journal of Real–Time Systems, 2020.

[15] I. Hussain, M. A. Awan, P. F. Souto, K. Bletsas, and E. Tovar, "Response time analysis of memory-bandwidth-regulated multiframe mixed-criticality systems," Journal of Systems Architecture, vol. 123, p. 102346, 2022.

[16] R. I. Davis, I. Bate, G. Bernat, I. Broster, A. Burns, A. Colin, S. Hutchesson, and N. Tracey, "Transferring real-time systems research into industrial practice: Four impact case studies," in Proceedings of the 30th Euromicro Conference on Real-Time Systems, pp. 7:1–7:24, 2018.

[17] CIA, "Automotive networks. CAN in automation (cia)." doi: http://www.can-cia.org/index.php?id=416.

[18] P. Thorngren, "keynote talk: Experiences from east-adl use," in EAST-ADL Open Workshop, Gothenberg, 2013.

[19] I. Standard, "Road vehicles–controller area network (CAN)–part 1: Data link layer and physical signalling," ISO, vol. 11898, p. 1, 2003.

[20] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised," Journal of Real–Time Systems, vol. 35, pp. 239–272, 2007.

[21] R. I. Davis, S. Kollmann, V. Pollex, and F. Slomka, "Schedulability analysis for Controller Area Network (CAN) with FIFO queues priority queues and gateways," Journal of Real–Time Systems, vol. 49, no. 1, pp. 73–116, 2013.

[22] P. M. Yomsi, D. Bertrand, N. Navet, and R. I. Davis, "Controller area network (CAN): Response time analysis with offsets," in Proceedings of the 9th IEEE international workshop on factory communication systems, pp. 43–52, IEEE, 2012.

[23] I. Hussain, P. F. Souto, K. Bletsas, M. A. Awan, and E. Tovar, "Schedulability analysis for can bus messages of periodically-varying size," in Proceedings of the 18thIEEE International Conference on Factory Communication Systems (WFCS), pp. 1–8, 2022.

[24] R. I. Davis, S. Kollmann, V. Pollex, and F. Slomka, "Controller area network (CAN) schedulability analysis with fifo queues," in Proceedings of the 23rd Euromicro Conference on Real-Time Systems, pp. 45–56, 2011.

[25] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," Journal of the ACM, vol. 20, pp. 46–61, 1973.

[26] A. K. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.

[27] O. Grandstrand, "Some theorems on real time scheduling.," in Computer Architecture and Networks (J. Labetoulle, ed.), pp. 285–298, New York, Amsterdam: North-Holland,, 1974.

[28] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," The Computer Journal, vol. 29, no. 5, pp. 390–395, 1986.

[29] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," Performance Evaluation, vol. 2, no. 4, pp. 237 – 250, 1982.

[30] N. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times: Technical report ycs164," Department of Computer Science, University of York, 1991.

[31] N. C. Audsley, "On priority assignment in fixed priority scheduling," Information Processing Letters, vol. 79, no. 1, pp. 39–44, 2001.

[32] N. Audsley, K. Tindell, and A. Burns, "The end of the line for static cyclic scheduling?," in Fifth Euromicro Workshop on Real-Time Systems, pp. 36–41, 1993.

[33] K. W. Tindell, A. Burns, and A. J. Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks," Journal of Real–Time Systems, vol. 6, no. 2, pp. 133–151, 1994.

[34] C.-C. Han, "A better polynomial-time schedulability test for real-time multiframe tasks," in Proceedings of the 19th IEEE Real-Time Systems Symposium, pp. 104–113, 1998.

[35] T.-W. Kuo, L.-P. Chang, Y.-H. Liu, and K.-J. Lin, "Efficient online schedulability tests for real-time systems," IEEE Transactions on Software Engineering, vol. 29, p. 734–751, aug 2003.

[36] S. Baruah, D. Chen, S. Gorinsky, and A. Mok, "Generalized multiframe tasks," Journal of Real–Time Systems, vol. 17, pp. 5–22, Jul 1999.

[37] N. T. Moyo, E. Nicollet, F. Lafaye, and C. Moy, "On schedulability analysis of non-cyclic generalized multiframe tasks," in Proceedings of the 22nd Euromicro Conference on Real-Time Systems, pp. 271–278, IEEE, 2010.

[38] H. Yun, R. Mancuso, Z.-P. Wu, and R. Pellizzoni, "PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms," in Proceedings of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 155–166, April 2014.

[39] J. Reineke, I. Liu, H. D. Patel, S. Kim, and E. A. Lee, "Pret dram controller: Bank privatization for predictability and temporal isolation," in Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pp. 99–108, ACM, 2011.

[40] Z. P. Wu, Y. Krish, and R. Pellizzoni, "Worst case analysis of dram latency in multi-requestor systems," in Proceedings of the 34rd IEEE Real-Time Systems Symposium, pp. 372–383, 2013.

[41] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in Proceedings of the 19th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 55–64, April 2013.

[42] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory access control in multiprocessor for real-time systems with mixed criticality," in Proceedings of the 24th Euromicro Conference on Real-Time Systems, pp. 299–308, 2012.

[43] M. A. Awan, K. Bletsas, P. F. Souto, B. Akesson, and E. Tovar, "Mixed-criticality scheduling with dynamic memory bandwidth regulation," in Proceedings of the 24th IEEE Conference on Embedded and Real-Time Computing and Applications, pp. 111–117, 2018.

[44] F. Bellosa, "Memory access-the third dimension of scheduling," Tech. Rep. TR-I4-97-01, Department of Computer Science IV (Operating Systems), 1997.

[45] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory access control in multiprocessor for real-time systems with mixed criticality," in Proceedings of the 24th Euromicro Conference on Real-Time Systems, pp. 299–308, 2012.

[46] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in Proceedings of the 19th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 55–64, April 2013.

[47] G. Yao, H. Yun, Z. P. Wu, R. Pellizzoni, M. Caccamo, and L. Sha, "Schedulability analysis for memory bandwidth regulated multicore real-time systems," IEEE Transactions on Computers, vol. 65, no. 2, pp. 601–614, 2015.

[48] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for COTS-based embedded systems," in Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 269–279, April 2011.

[49] G. Durrieu, M. Faugère, S. Girbal, D. G. Pérez, C. Pagetti, and W. Puffitsch, "Predictable flight management system implementation on a multicore processor," in Proceedings of the Embedded Real Time Software, 2014.

[50] R. Mancuso, R. Pellizzoni, M. Caccamo, L. Sha, and H. Yun, "Wcet (m) estimation in multi-core systems using single core equivalence," in Proceedings of the 27th Euromicro Conference on Real-Time Systems, pp. 174–183, IEEE, 2015.

[51] R. Mancuso, R. Pellizzoni, N. Tokcan, and M. Caccamo, "WCET Derivation under Single Core Equivalence with Explicit Memory Budget Assignment," in Proceedings of the 29th Euromicro Conference on Real-Time Systems, vol. 76 of Leibniz International Proceedings in Informatics (LIPIcs), (Dagstuhl, Germany), pp. 3:1–3:23, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

[52] M. Paolieri, E. Quinones, F. J. Cazorla, and M. Valero, "An analyzable memory controller for hard real-time cmps," IEEE Embedded Systems Letters, vol. 1, no. 4, pp. 86–90, 2009.

[53] R. Pellizzoni and H. Yun, "Memory servers for multicore systems," in Proceedings of the 22nd IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 97–108, 2016.

[54] M. A. Awan, P. F. Souto, K. Bletsas, B. Akesson, and E. Tovar, "Worst-case stall analysis for multi-core architectures with two memory controllers," in Proceedings of the 30th Euromicro Conference on Real-Time Systems, vol. 106 of Leibniz International Proceedings in Informatics (LIPIcs), pp. 2:1–2:22, 2018.

[55] G. Yao, H. Yun, Z. P. Wu, R. Pellizzoni, M. Caccamo, and L. Sha, "Schedulability analysis for memory bandwidth regulated multicore real-time systems," IEEE Transactions on Computers, vol. 65, pp. 601–614, Feb 2016.

[56] M. A. Awan, K. Bletsas, P. F. Souto, B. Akesson, and E. Tovar, "Mixed-criticality scheduling with dynamic memory bandwidth regulation," in Proceedings of the 24th IEEE Conference on Embedded and Real-Time Computing and Applications, pp. 111–117, 2018.

[57] M. A. Awan, K. Bletsas, P. F. Souto, B. Akesson, and E. Tovar, "Mixed-Criticality Scheduling with Dynamic Redistribution of Shared Cache," in Proceedings of the 29th Euromicro Conference on Real-Time Systems, vol. 76 of Leibniz International Proceedings in Informatics (LIPIcs), pp. 18:1–18:21, 2017.

[58] G. Gracioli, R. Tabish, R. Mancuso, R. Mirosanlou, R. Pellizzoni, and M. Caccamo, "Designing mixed criticality applications on modern heterogeneous MPSoC platforms," in Proceedings of the 31th Euromicro Conference on Real-Time Systems, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[59] M. Hassan and R. Pellizzoni, "Bounding dram interference in COTS heterogeneous MPSoCs for mixed criticality systems," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 11, pp. 2323–2336, 2018.

[60] A. Burns and R. Davis, "Mixed criticality systems – a review (12th ed.)," tech. rep., Department of Computer Science, University of York, Mar. 2019.

[61] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with preemption threshold," in Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No. PR00306), pp. 328–335, IEEE, 1999.

[62] G. Yao, G. Buttazzo, and M. Bertogna, "Bounding the maximum length of non-preemptive regions under fixed priority scheduling," in Proceedings of the 15th IEEE Conference on Embedded and Real-Time Computing and Applications, RTCSA '09, (Washington, DC, USA), pp. 351–360, IEEE Computer Society, 2009.

[63] R. J. Bril, J. J. Lukkien, and W. F. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption," Journal of Real–Time Systems, vol. 42, pp. 63–119, 2009.

[64] Q. Zhao, Z. Gu, and H. Zeng, "Pt-amc: Integrating preemption thresholds into mixed-criticality scheduling," in Proceedings of the 50th ACM/IEEE Conference on Design Automation Conference, pp. 141–146, March 2013.

[65] Q. Zhao, Z. Gu, and H. Zeng, "Design optimization for autosar models with preemption thresholds and mixed-criticality scheduling," Journal of Systems Architecture, vol. 72, pp. 61–68, 2017.

[66] A. Burns and R. Davis, "Adaptive mixed criticality scheduling with deferred preemption," in Proceedings of the 35rd IEEE Real-Time Systems Symposium, pp. 21–30, IEEE, Dec 2014.

[67] T. Fleming, Extending mixed criticality scheduling. PhD thesis, University of York, 2013. doi:https://etheses.whiterose.ac.uk/5688/.

[68] Y. Zhao and H. Zeng, "An efficient schedulability analysis for optimizing systems with adaptive mixed-criticality scheduling," Journal of Real–Time Systems, vol. 53, pp. 467–525, 2017.

[69] S. Asyaban and M. Kargahi, "Feasibility interval for fixed-priority scheduling of mixed-criticality periodic tasks with offsets," IEEE Embedded Systems Letters, vol. 11, pp. 17–20, March 2019.

[70] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele, "Service adaptions for mixed-criticality systems," in 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 125–130, IEEE, 2014.

[71] C. Gill, J. Orr, and S. Harris, "Supporting graceful degradation through elasticity in mixedcriticality federated scheduling," in Proceedings of the 6th Workshop on Mixed-Criticality Systems, pp. 19–24, 2018.

[72] M. Jan, L. Zaourar, and M. Pitel, "Maximizing the execution rate of low-criticality tasks in mixed criticality systems," in Proceedings of the 1st Workshop on Mixed-Criticality Systems, pp. 43–48, 2013.

[73] S. Baruah, "Certification-cognizant scheduling of tasks with pessimistic frequency specification," in Proceedings of the 7th IEEEInternational Symposium on Industrial Embedded Systems, pp. 31–38, 2012.

[74] H.-M. Huang, C. Gill, and C. Lu, "Implementation and evaluation of mixed-criticality scheduling approaches for periodic tasks," in Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 23–32, 2012.

[75] H.-M. Huang, C. Gill, and C. Lu, "Implementation and evaluation of mixed-criticality scheduling approaches for sporadic tasks," ACM Transactions on Embedded Computing Systems, vol. 13, pp. 126:1–126:25, Apr. 2014.

[76] Y. Chen, K. G. Shin, and H. Xiong, "Generalizing fixed-priority scheduling for better schedulability in mixed-criticality systems," Information Processing Letters, vol. 116, no. 8, pp. 508–512, 2016.

[77] S. Baruah and Z. Guo, "Mixed-criticality scheduling upon varying-speed processors," in Proceedings of the 34rd IEEE Real-Time Systems Symposium, pp. 68–77, IEEE, 2013.

[78] Z. Guo and S. Baruah, "The concurrent consideration of uncertainty in wcets and processor speeds in mixed-criticality systems," in Proceedings of the 23rd International Conference on Real Time and Networks Systems, pp. 247–256, 2015.

[79] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Run and be safe: Mixed-criticality scheduling with temporary processor speedup," in Proceedings of the 52nd ACM/IEEE Conference on Design Automation Conference, pp. 1329–1334, IEEE, 2015.

[80] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Energy efficient dvfs scheduling for mixed-criticality systems," in Proceedings of the 19th Asia and South Pacific Design Automation Conference, pp. 1–10, 2014.

[81] K. Agrawal, S. Baruah, and A. Burns, "Semi-clairvoyance in mixed-criticality scheduling," in Proceedings of the 40rd IEEE Real-Time Systems Symposium, pp. 458–468, IEEE, 2019.

[82] A. Burns and R. I. Davis, "Schedulability analysis for adaptive mixed criticality systems with arbitrary deadlines and semi-clairvoyance," in Proceedings of the 41st IEEE Real-Time Systems Symposium, pp. 12–24, IEEE, 2020.

[83] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in Proceedings of the 24th Euromicro Conference on Real-Time Systems, pp. 145–154, July 2012.

[84] P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic tasks," in Proceedings of the 24th Euromicro Conference on Real-Time Systems, pp. 135–144, July 2012.

[85] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," Journal of Real–Time Systems, vol. 50, no. 1, pp. 48–86, 2014.

[86] M. A. Awan, K. Bletsas, P. F. Souto, B. Akesson, and E. Tovar, "Mixed-criticality scheduling with dynamic redistribution of shared cache," arXiv preprint arXiv:1704.08876, 2017.

[87] A. Easwaran, "Demand-based scheduling of mixed-criticality sporadic tasks on one processor," in Proceedings of the 34rd IEEE Real-Time Systems Symposium, pp. 78–87, IEEE, 2013.

[88] K. W. Tindell and A. Burns, "Guaranteeing message latencies on Controller Area Network (CAN)," in Proceedings of the 1st International CAN conference, pp. 1–11, 1994.

[89] Tindell, Hansson, and Wellings, "Analysing real-time communications: controller area network (CAN)," in Proceedings of the 15th IEEE Real-Time Systems Symposium, pp. 259–263, 1994.

[90] K. W. Tindell, A. Burns, and A. J. Wellings, "Calculating Controller area network (CAN) message response times," Control Engineering Practice, vol. 3, no. 8, pp. 1163—-1169, 1995.

[91] I. Broster, A. Burns, and G. Rodriguez-Navas, "Probabilistic analysis of CAN with faults," in Proceedings of the 23rd IEEE Real-Time Systems Symposium, pp. 269–278, IEEE, 2002.

[92] I. Broster, A. Burns, and G. Rodriguez-Navas, "Timing analysis of real-time communication under electromagnetic interference," Journal of Real–Time Systems, vol. 30, pp. 55–81, 2005.

[93] T. Nolte, H. Hansson, and C. Norstrom, "Minimizing CAN response-time jitter by message manipulation," in Proceedings. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 197–206, IEEE, 2002.

[94] T. Nolte, Share-driven scheduling of embedded networks. PhD thesis, Institutionen för Datavetenskap och Elektronik, 2006. doi: https://www.diva-portal.org/smash/get/diva2:120500/FULLTEXT01.pdf.

[95] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg, "Volcano-a revolution in on-board communications," Volvo technology report, vol. 1, pp. 9–19, 1998.

[96] M. G. Harbour, M. H. Klein, and J. P. Lehoczky, "Fixed priority scheduling periodic tasks with varying execution priority.," in Proceedings of the 12th IEEE Real-Time Systems Symposium, pp. 116–128, Citeseer, 1991.

[97] L. George, N. Rivierre, and M. Spuri, Preemptive and non-preemptive real-time uniprocessor scheduling. PhD thesis, Inria, 1996. doi: https://inria.hal.science/inria-00073732/document.

[98] M. Grenier, L. Havet, and N. Navet, "Pushing the limits of CAN-scheduling frames with offsets provides a major performance boost," in Proceedings of the 20th Euromicro Conference on Real-Time Systems, 2008. doi: https://hal.science/insu-02270103/.

[99] M. Grenier, L. Havet, N. Navet, et al., "Scheduling messages with offsets on controller area network-a major performance boost," 2009. doi: https://doi.org/10.1201/9780849380273.

[100] Y. Chen, R. Kurachi, H. Takada, and G. Zeng, "Schedulability comparison for CAN message with offset: Priority queue versus fifo queue.," in Proceedings of the 19th Conference Real-Time and Networked Systems, pp. 181–192, Citeseer, 2011.

[101] S. Mubeen, J. Mäm-Turja, and M. Sjödin, "Extending schedulability analysis of controller area network (can) for mixed (periodic/sporadic) messages," in ETFA2011, pp. 1–10, 2011.

[102] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Worst-case response-time analysis for mixed messages with offsets in controller area network," in Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012), pp. 1–10, IEEE, 2012.

[103] R. I. Davis and N. Navet, "Controller area network (CAN) schedulability analysis for messages with arbitrary deadlines in fifo and work-conserving queues," in Proceedings of the 9th IEEE international workshop on factory communication systems, pp. 33–42, 2012.

[104] Y. Xie, G. Zeng, Y. Chen, R. Kurachi, H. Takada, and R. Li, "Worst case response time analysis for messages in controller area network with gateway," IEICE TRANSACTIONS on Information and Systems, vol. 96, no. 7, pp. 1467–1477, 2013.

[105] M. Nahas, M. J. Pont, and M. Short, "Reducing message-length variations in resource-constrained embedded systems implemented using the controller area network (CAN) protocol," Journal of Systems Architecture, vol. 55, no. 5-6, pp. 344–354, 2009.

[106] F. Pölzlbauer, R. I. Davis, and I. Bate, "Analysis and optimization of message acceptance filter configurations for controller area network (CAN)," in Proceedings of the 25th International Conference on Real-Time Networks and Systems, pp. 247–256, 2017.

[107] E. Azketa, J. J. Gutiérrez, J. C. Palencia, M. G. Harbour, L. Almeida, and M. Marcos, "Schedulability analysis of multi-packet messages in segmented CAN," in Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012), pp. 1–8, 2012.

[108] R. Bosch et al., "Can specification version 2.0," Rober Bousch GmbH, Postfach, vol. 300240, p. 72, 1991.

[109] B. Nikolic, M. A. Awan, and S. M. Petters, "Sparts: Simulator for power aware and real-time systems," in Proceedings of the 8th IEEE International Conference on Embedded Software and Systems, (Changsha, China), pp. 999–1004, IEEE, Nov. 2011.

[110] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," Journal of Real–Time Systems, vol. 30, no. 1-2, pp. 129–154, 2005.

[111] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in Proceedings of the 30th IEEE Real-Time Systems Symposium, pp. 398–409, 2009.

[112] J. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in Proceedings of the 11th IEEE Real-Time Systems Symposium, pp. 201–209, 1990.

[113] A. Burns and R. I. Davis, "Response time analysis for mixed criticality systems with arbitrary deadlines," in Proceedings of the 5th Workshop on Mixed-Criticality Systems, York, 2017.

[114] N. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," technical report, Dept. Computer Science, University of York, UK, 1991.

[115] R. I. Davis and A. Burns, "Improved Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems," Journal of Real–Time Systems, vol. 47, pp. 1–40, 2011.

[116] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," Journal of Real–Time Systems, vol. 30, no. 1-2, pp. 129–154, 2009.

[117] R. Jain, The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling. Wiley professional computing, Wiley, 1991.

[118] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," Proceedings of the 6th Workshop on Operating System Platforms for Embedded Real-Time Applications, vol. 10, pp. 33–44, 2010.

[119] S. K. Baruah, H. Li, and L. Stougie, "Mixed-criticality scheduling: Improved resource-augmentation results.," in CATA, pp. 217–223, 2010.

[120] M. A. Awan, P. F. Souto, K. Bletsas, B. Akesson, and E. Tovar, "Memory bandwidth regulation for multiframe task sets," in Proceedings of the 25th IEEE Conference on Embedded and Real-Time Computing and Applications, pp. 1–11, 2019.

[121] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," Proceedings of the 6th Workshop on Operating System Platforms for Embedded Real-Time Applications, vol. 10, pp. 33–44, 2010.

[122] I. Hussain, M. A. Awan, K. Bletsas, P. F. Souto, and E. Tovar, "Considerations on combining vestal's mixed-criticality task model and the predictable execution model (prem) for real-time systems," 2021.