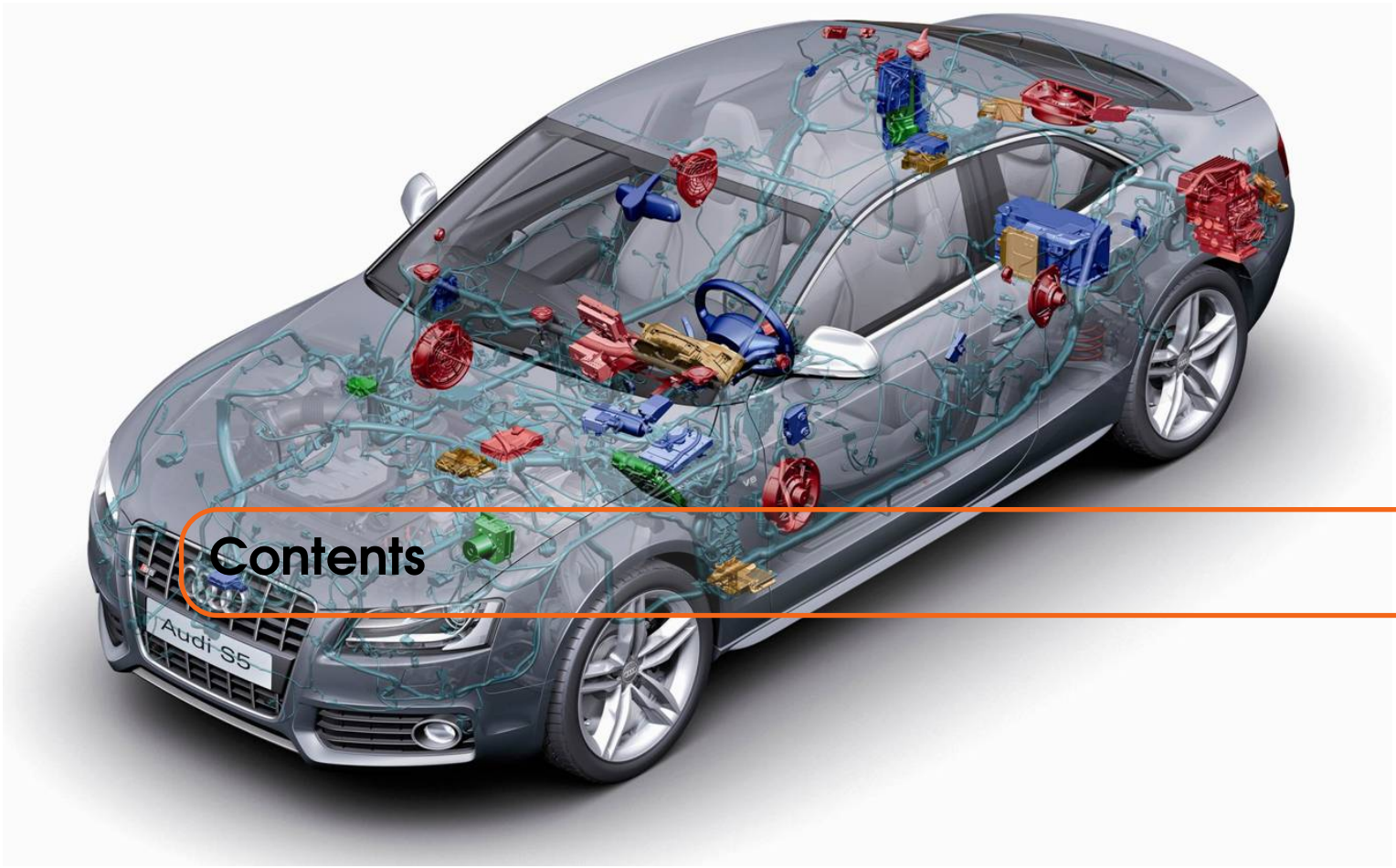
Mixed Criticality support for Automotive Real Time Systems.

PUBLISHED BY PUBLISHER

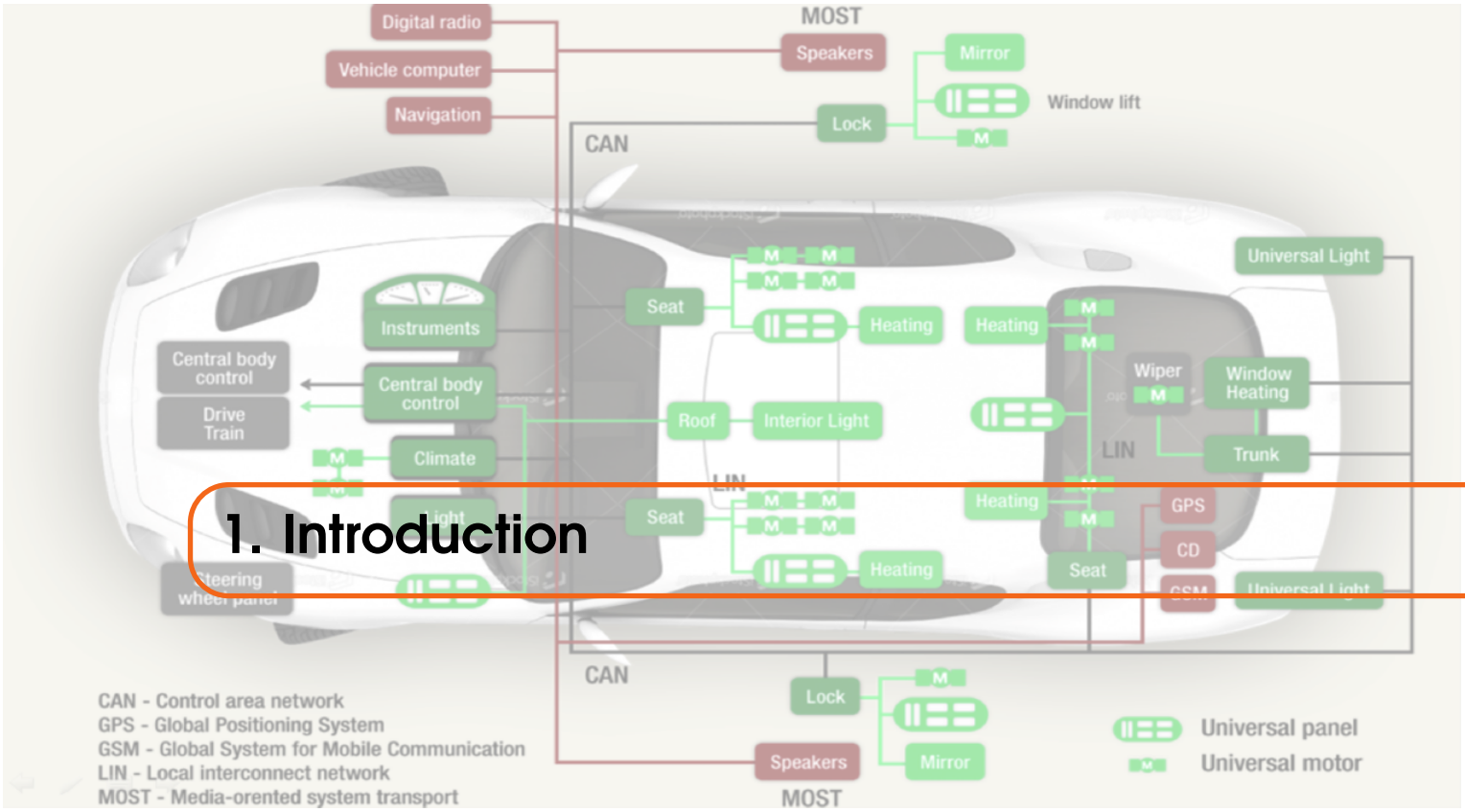
BOOK-WEBSITE.COM

First printing, March 2013



Contents

Part One



1. Introduction

1.1 Motivation

The automotive industry is today the sixth largest economy in the world, produc- ing around 70 million cars every year and making an important contribution to government revenues all around the world. As for other industries, significant improvements in functionalities, performance, comfort, safety, etc. are provided by electronic and software technologies. Indeed, since 1990, the sector of embedded elec- tronics, and more precisely embedded software, has been increasing at an annual rate shared between electronic and software components. These general trends have led to currently embedding up to 500 MB on more than 70 microprocessors connected on communication networks. The following are some of the various examples. Figure 1.1 shows an electronic architecture embedded in a Laguna (source: Renault French carmaker) illustrating several computers interconnected and controlling the engine, the wipers, the lights, the doors, and the suspension or providing a support for inter- action with the driver or the passengers. In 2004, the embedded electronic system of a Volkswagen Phaeton was composed of more than 10,000 electrical devices, 61 micro- processors, three controller area networks (CAN) that support the exchanges of 2500 pieces of data, several subnetworks, and one multimedia bus. In the Volvo S70, two networks support the communication between the microprocessors controlling the mirrors, those controlling the doors and those controlling the transmission system and, for example, the position of the mirrors is automatically controlled according to the sense the vehicle is going and the volume of the radio is adjusted to the vehi- cle speed, information provided, among others, by the antilock braking system (ABS) controller. In a recent Cadillac, when an accident causes an airbag to inflate, its micro- controller emits a signal to the embedded global positioning system (GPS) receiver that then communicates with the cell phone, making it possible to give the vehicle's position to the rescue service. The software code size of the Peugeot CX model (source: PSA Peugeot Citroen French carmarker) was 1.1 KB in 1980, and 2 MB for the 607 model in 2000. These are just a few examples, but there are many more that could illustrate this very large growth of embedded electronic systems in modern vehicles. The automotive industry has evolved rapidly and will evolve even more rapidly under the influence of several factors such as pressure from state legislation, pressure from customers, and technological progress (hardware and software aspects).

Indeed, a great surge for the development of electronic control systems came through the regulation concerning air pollution. But we must also consider the pressure from Vehicle Functional Domains and Their Requirements consumers for more performance (at lower fuel consumption), comfort, and safety. Add to all this the fact that satisfying these needs and obligations is only possible because of technological progress. Electronic technology has made great strides and nowadays the quality of electronic components—performance, robustness, and reliability—enables using them even for critical systems. At the same time, the decreasing cost of electronic technology allows them to be used to support any function in a car. Furthermore, in the last decade, several automotive-embedded networks such as local interconnect networks (LIN), CAN, TTP/C, FlexRay, MOST, and IDB-1394 were developed. This has led to the con- cept of multiplexing, whose principal advantage is a significant reduction in the wiring cost as well as the flexibility it gives to designers; data (e.g., vehicle speed) sampled by one microcontroller becomes available to distant functions that need them with no additional sensors or links. Another technological reason for the increase of automotive embedded systems is the fact that these new hardware and software technologies facilitate the introduction of functions whose development would be costly or not even feasible if using only mechanical or hydraulic technology. Consequently, they allow to satisfy the end user requirements in terms of safety, comfort, and even costs. Well-known examples are electronic engine control, ABS, electronic stability program (ESP), active suspension, etc. In short, thanks to these technologies, customers can buy a safe, efficient, and personalized vehicle, while carmakers are able to master the differentiation between product variations and innovation (analysts have stated that more than 80% of inno- vation, and therefore of added value, will be obtained thanks to electronic systems). Furthermore, it also has to be noted that some functions can only be achieved through digital systems. The following are some examples: (1) the mastering of air pollution can only be achieved by controlling the engine with complex control laws; (2) new engine concepts could not be implemented without an electronic control; (3) mod- ern stability control systems (e.g., ESP), which are based on close interaction between the engine, steering, and braking controllers, can be efficiently implemented using an embedded network. Last, multimedia and telematic applications in cars are increasing rapidly due to consumer pressure; a vehicle currently includes electronic equipment like hand-free phones, audio/radio devices, and navigation systems. For the passengers, a lot of entertainment devices, such as video equipment and communication with the out- side world are also available. These kinds of applications have little to do with the vehicle's operation itself; nevertheless they increase significantly as part of the software included in a car. In short, it seems that electronic systems enable limitless progress. But are elec- tronics free from any outside pressure? No. Unfortunately, the greatest pressure on electronics is cost! Keeping in mind that the primary function of a car is to provide a safe and efficient means of transport, we can observe that this continuously evolving "electronic revolu- tion" has two primary positive consequences. The first is for the customer/consumer, who requires an increase in performance, comfort, assistance for mobility efficiency (navigation), and safety on the one hand, while on the other hand, is seeking reduced fuel consumption and cost. The second positive consequence is for the stakehold- ers, carmakers, and suppliers, because software-based technology reduces marketing time, development cost, production, and maintenance cost. Additionally, these inno- vations have a strong impact on our society because reduced fuel consumption and exhaust emissions improve the protection of our natural resources and the environ- ment, while the introduction of vision systems, driver assistance, onboard diagnosis, etc., targets a "zero death" rate, as has been stated in Australia, New Zealand, Sweden, and the United Kingdom. However, all these advantages are faced with an engineering challenge; there have been an increasing number of breakdowns due to failure in electric/electronic sys- tems. For example, Ref.6 indicates that, for 2003, 49.2% of car breakdowns were due to such problems in Germany. The quality of a product obviously depends on the quality of its development, and the increasing complexity of in-vehicle embed-

ded systems raises the problem of mastering their development. The design process is based on a strong cooperation between different players, in particular Tier 1 sup- pliers and carmakers, which involves a specific concurrent engineering approach. For example, in Europe or Japan, carmakers provide the specification for the subsystems to suppliers, who, in turn, compete to find a solution for these carmakers. The chosen suppliers are then in charge of the design and realization of these subsystems, includ- ing the software and hardware components, and possibly the mechanical or hydraulic parts as well. The results are furnished to the carmakers, or original equipment manufacturer (OEM), who install them into the car and test them. The last step consists of calibration activities where the control and regulation parameters are tuned to meet the required performance of the controlled systems. This activity is closely related to the testing activities. In the United States, this process is slightly different since the suppliers cannot really be considered as independent from the carmakers. Not all electronic systems have to meet the same level of dependability as the pre- vious examples. While with a multimedia system customers require a certain quality and performance, with a chassis control system, safety assessment is the predominant concern. So, the design method for each subsystem depends on different techniques. Nevertheless, they all have common distributed characteristics and they must all be at the level of quality fixed by the market, as well as meeting the safety requirements and the cost requirements. As there has been a significant increase in computer- based and distributed controllers for the core critical functions of a vehicle (power train, steering or braking systems, "X-by-wire" systems, etc.) for several years now, a standardization process is emerging for the safety assessment and certification of automotive-embedded systems, as has already been done for avionics and the nuclear industry, among others. Therefore, their development and their production need to be based on a suitable methodology, including their modeling, a priori evaluation and validation, and testing. Moreover, due to competition between carmakers or between suppliers to launch new products under cost, performance, reliability, and safety constraints, the design process has to cope with a complex optimization problem. In-vehicle embedded systems are usually classified according to domains that correspond to different functionalities, constraints, and models. They can be divided among "vehicle-centric" functional domains, such as power train control, chassis control, and active or passive safety systems and "passenger centric" functional Vehicle Functional Domains and Their Requirements domains where multimedia/telematics, body/comfort, and human–machine interface (HMI) can be identified.

1.2 Citation

This statement requires citation [**book_key**]; this one is more specific [**article_key**].

1.3 Lists

Lists are useful to present information in a concise and/or ordered way[1].

1.3.1 Numbered List

1. The first item
2. The second item
3. The third item

1.3.2 Bullet Points

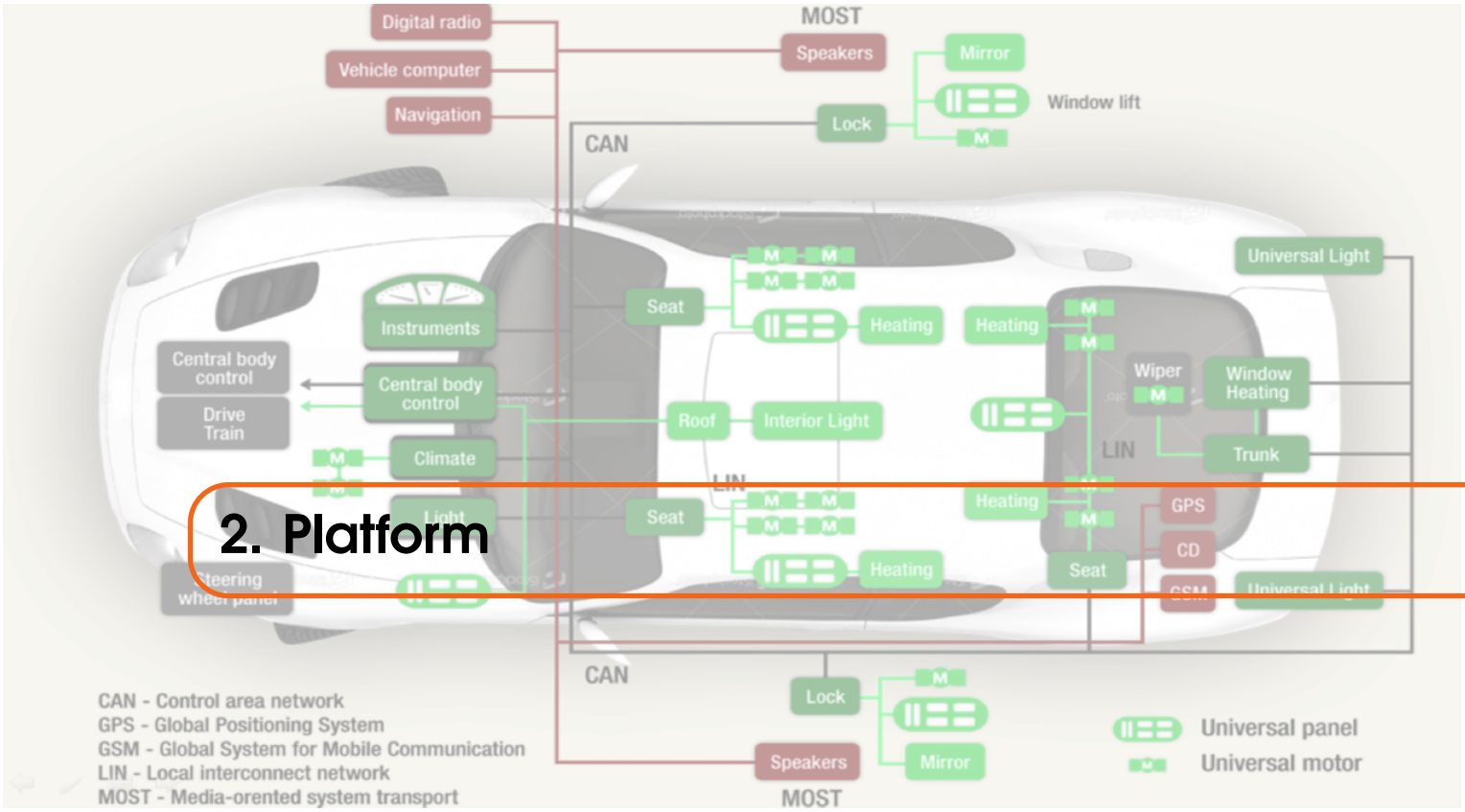
- The first item
- The second item

[1] Footnote example...

- The third item

1.3.3 Descriptions and Definitions

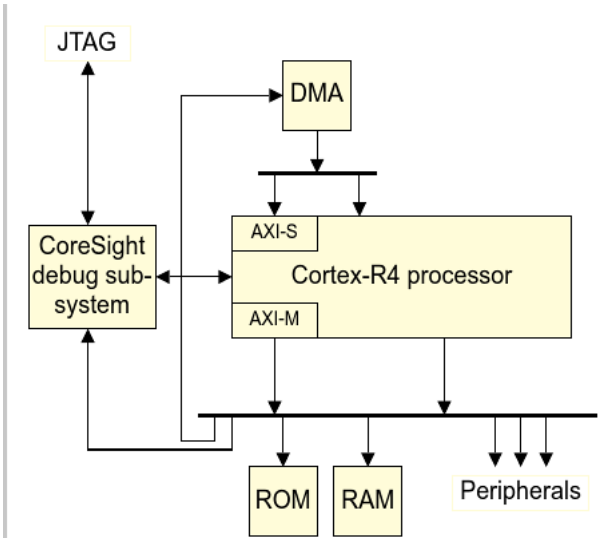
Name Description
Word Definition
Comment Elaboration



2. Platform

2.1 Cortex-R4

Cortex R4 processor is a midrange processor used in real-time systems. It implements ARM-V7R Architecture and supports THUMB-2 technology for optimum code density and processing throughput. The pipeline has single Arithmetic Logic Unit but uses limited dual-issuing to of instructions for efficient utilization of other resources such as registers. The processor has Tightly coupled Memory(TCM) port for low latency and deterministic access to local RAM in addition to caches for high performance general memory. The Level 1 memory and processor ports of Cortex R4 uses Error Correction Code(ECC) for greater reliability and address safety. An overall component level diagram of Cortex-R4 is shown below:



The main features of the Cortex-R4 are listed below:

- A Dual issue integer unit with integral coresight logic.
- High speed Advanced Microprocessor Bus Architecture(AMBA) and Advance eXtensible Interface(AXI) for master and slave interfaces.
- Dynamic branch prediction with history buffer and 4-entry return stack.
- Low interrupt latency.
- Non Maskable Interrupt.
- Optional Floating point unit.
- Harvard L1 Memory system with:
  - Optional Tightly Coupled Memory(TCM) with support for error correction or parity checking.
  - Optional caches with error correction codes(ECC).
  - Optional ARM-V7R Memory Protection Unit(MPU).
  - Optional parity and error correction support for all RAM blocks.
- An L2 Memory Interface with single 64-bit master AXI interface and 64 bit slave AXI interface to TCM RAM blocks and cache RAM blocks.
- A Performance measuring Unit(PMU).
- Vectored Interrupt Controller(VIC).

2.1.1 Components

TCM Interfaces

TCM Interface for Cortex-R4 are composed of two interfaces, ATCM and BTCM. An ATCM usually holds the exception handler code that must be accessed at high speed without any cache miss. BTCM meanwhile holds block of data for intensive operations such as Audio and Video processing.

Power Management

Processor includes number of processor features for power management:

- Accurate branch and return prediction.
- Cache uses sequential access information to reduce the number of accesses.
- Extensive use of clocks and gates to disable the inputs to unused functional block.

Cortex-R4 supports 4 different power levels:

- **Run Mode**: The normal mode where all the functionalities of the processor are available.
- **Dormant Mode**:Dormant mode disables with processing logic but not the TCM and cache RAMs. Before entering the mode the processor state is saved to the memory and restored when exiting the mode.
- **Standby Mode**: In standby mode most of the clocks of the device will be disabled, while the device will be kept powered on.
- **Shutdown Mode**: In this mode entire device including cache and TCM logic are disable and should be saved to the external memory.

System Reset Levels

The processor has following reset inputs: nRESET, PRESETDBGn, nSYSPORESET, nCPUHALT. Using the combination of given resets following reset modes can be realized:

- **Power On Reset**:Also known as hard reset or cold reset. Both nRESET and nSYSPORESET are asserted and this is done during initial system reset. This will reset entire system.
- **Processor Reset**: Also known as soft reset or warm-reset. nRESET is asserted while nSYSPORESET is set to 1.

- **Halt**: Both nSYSPORESET and nRESET are set to 1 , while nCPUHALT is asserted by setting to 0.
- **Normal Mode**: All three nRESET, nSYSPORESET and nCPUHALT are set to 1. One use case is doing DMA into TCM using the processor.

2.1.2 Initialization

Most of the architectural registers in the processor, such as r0-r14, s0-s31 and d0-15 are not reset. Thus these needs to be reset explicitly during the initialization phase. So of the main activities to be carried out includes: programming registers like stack pointers, processor features and interface initialization and are mentioned below:

MPU

If the processor is built with Memory Protection Unit(MPU), before using it program and enable on of its region and enable MPU in system control registers. If MPU regions are enabled, program them to enable access to TCM regions before using them.

FPU

If the processor is build with Floating Point Unit, enable it before accessing VFP instructions. This is done by enabling access to FPU in Coprocessor Access Registers. Also enable FPU by setting the EN bit in FPEXC(Floating point Exception) register.

Cache

If the processor are build with instruction or data cache they must be invalidated before use otherwise the behavior is unpredictable. If ECC is to be supported for the cache, it must be enabled before invalidating the cache by programming the Auxiliary Control Register. This enabled correct error code or parity bits are calculated.

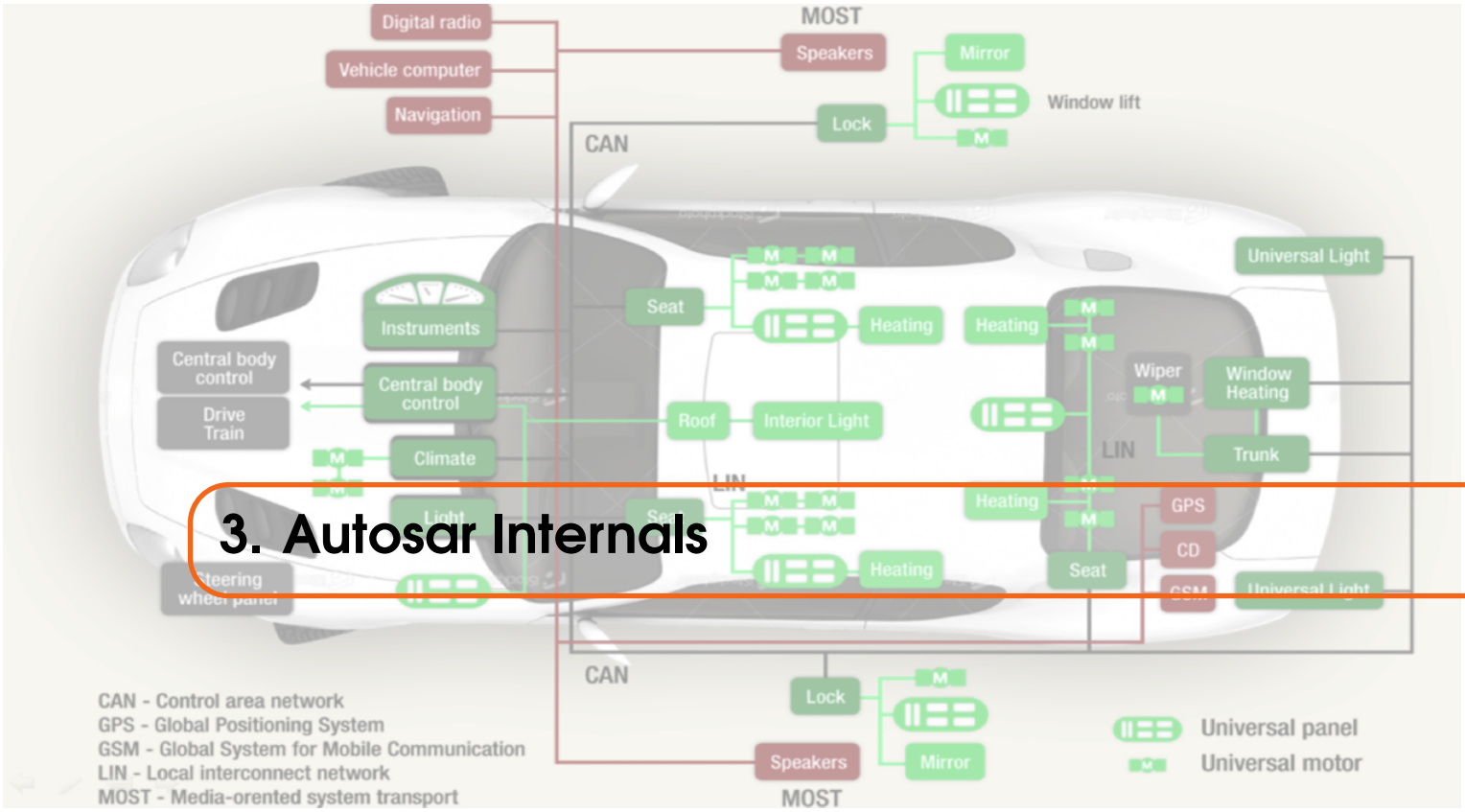
TCM

TCM is not explicitly initialized by Processor and as such should be initialized as per the requirement. Additionally main instruction may require data to be preloaded into TCM. There are multiple ways TCM memory can be preloaded and are listed below:

- The boot code can copy the required data from ROM using the memory routine. For this TCM need to be enabled and separate base address to TCM during copy and normal program execution are provided.
- Boot code includes routine to copy data from the Debug Communications Channel(DCC) to TCM.
- Processor is put into debug halt state and Instruction Transfer Register(DBITR) is used to transfer data that is executed by the processor and overrides the boot code.
- SoC includes DMA device which can directly write the data from ROM code to TCM through AXI slave.
- Debug Access Port(DAP) can be used to generate AMBA transactions to write data directly to TCM through AXI slave.

Refer to Auxiliary Control Registers to check for initializing TCM with ECC[1]. Processor can be pin configured to enable TCM at reset.

[1] Error Correction Code



3. Autosar Internals

3.1 Design and Approach

AUTOSAR provides a standardization of the software stacks that are employed in Automotive systems. It allows collaboration between various partners by standardization of the data exchange format and integration of basic software and application software from various partners on a single ECU via a middleware and across the vehicle network. AUTOSAR supports broad variety of domains which includes but not limited to:

- body/comfort.
- driver assistant systems.
- powertrain.
- chassis control.
- occupation and pedestrian safety.

3.2 OS Internals

AUTOSAR extends OSEK/VDX Operating systems standards which finds wide adoption in automotive systems. OSEK OS is an event triggered operating system. This provides high flexibility in the design and maintenance of the system. Event triggering gives freedom of selecting from a wide range of events to drive scheduling at runtime and this includes angular rotation, local timer , global time source, error events etc.

3.2.1 BSW Scheduler

BSW Scheduler provides the basic abstraction for handling of task behavior specific to the timing and control. AUTOSAR Specification neither provides nor recommends any specific implementation of the BSW Scheduler but specifies the feature set that are to be provided. BSW Scheduler is not a competing entity to AUTOSAR OS Scheduler, instead it provides following features:

- embed BSW modules in AUTOSAR Context.
- trigger main processing functions of the BSW Module.
- Apply data consistency mechanism for the BSW modules(e.g. Locking mechanisms.)

For example, BSW Module provides EnableAllInterrupts and DisableAllInterrupts APIs as a meaning of serializing the execution of the critical sections. The BSW scheduler files encompasses a SchM.h file providing the abstraction to start the task corresponding to the BSW Scheduler and set of support file which are of the form BSW_<Module Prefix>.h. The module specific files are a form of contract to support the data consistency mechanism promised by the BSW Scheduler. BSW

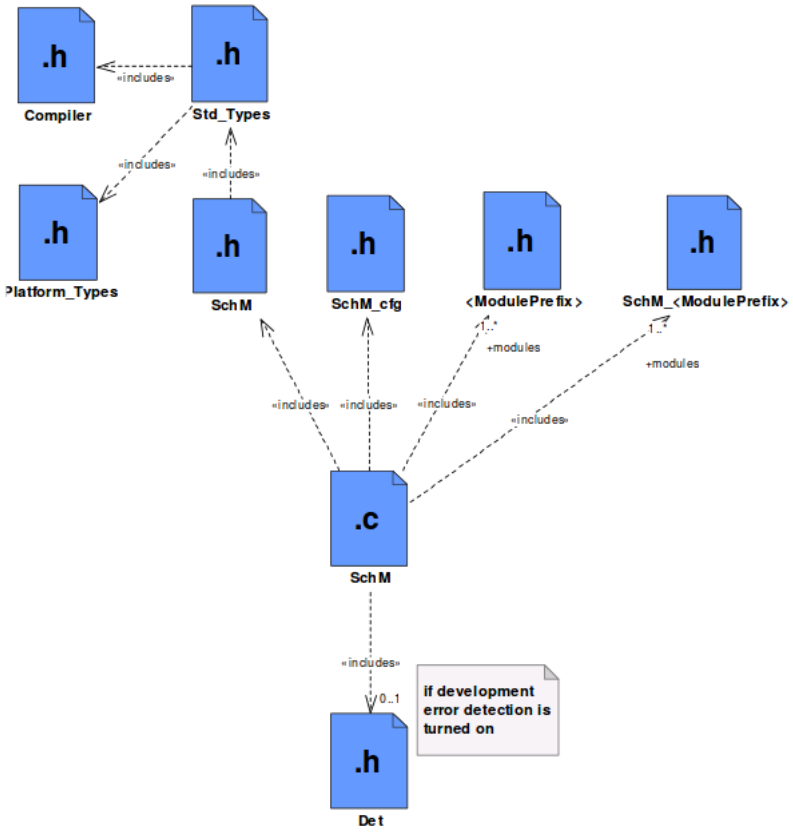


Figure 2: Header file structure of the BSW Scheduler

Scheduler can be implemented as a task that will handle read write operations. BSW Scheduler cannot invoke the main functions that can enter the wait state. Thus BSW Scheduler handler the entities, which being the individual actions that can be handled in the context of a task, while the task level control resides with AUTOSAR OS Scheduler.

3.2.2 OS Interrupts

Interrupts under AUTOSAR are classified into cat1 and cat2. The difference between the two are as follows:

- cat1 interrupts are not allowed to make OS calls. The only APIs they can call are to enable and disable interrupts. While cat2 interrupts can call most OS calls, while others are illegal.
- cat1 interrupts have lower latency than cat2.
- There exists no support for cat1 interrupts from OS, to abstract the interrupt and its handler from the hardware in portable way and is depends on the platform and the tool-chain.
- In cat1 interrupt handler there is no need to lock the critical region as it will be shared with task or interrupt of lower priority.

Initialization steps for the interrupt handler involves defining vector table and correct entry for handler function. cat2 interrupt uses ISR macro to define the interrupt handler, advantage of this

being, it encapsulates the handler function. Mutual exclusion in cat2 interrupts are provided by BSW Scheduler in the for of disabling and enabling interrupt sub group. In case of cat1 interrupts if the mutual exclusion is required, it can be done in terms of disabling all the interrupts. As this seriously affects the overall performance of the system, cat1 region should be kept to minimum.

3.2.3 Time Service Module

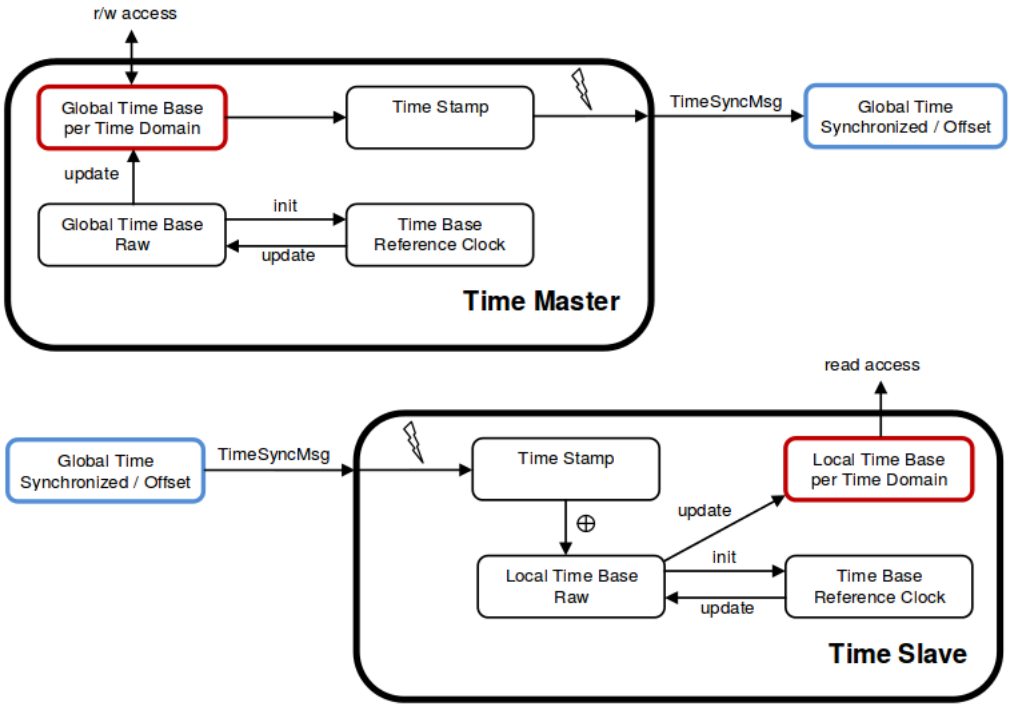
Time Service Module is part of the service layer and provides following features:

- Time measurement.
- Time based state machine.
- Timeout supervision
- Busy Loop feature.

Time Service Module does not sit on top of the timer stack and does not use and provide all the features of GPT Driver.

3.2.4 Time Base Manager

Synchronized time base manager provides APIs to its customers or time bases, which are synchronized with other time bases of the distributed systems, thus acting as a time broker. Time Base Manager acts as the central module to which provides absolute time of the system. Typical use case include Sensor fusion(Temporal correlation various sensor data.), Event data recording(Temporal correlation of the logs while reporting on system crash or on detection of an anomaly), and diagnostic event storage. The customers to the Time Base Manager can be triggered or active. AUTOSAR OS is currently supported as a triggered customer. A Global time network consists of atleast one Time master and atleast one Time Slave.
Figure below shows a representation of such a global time network.

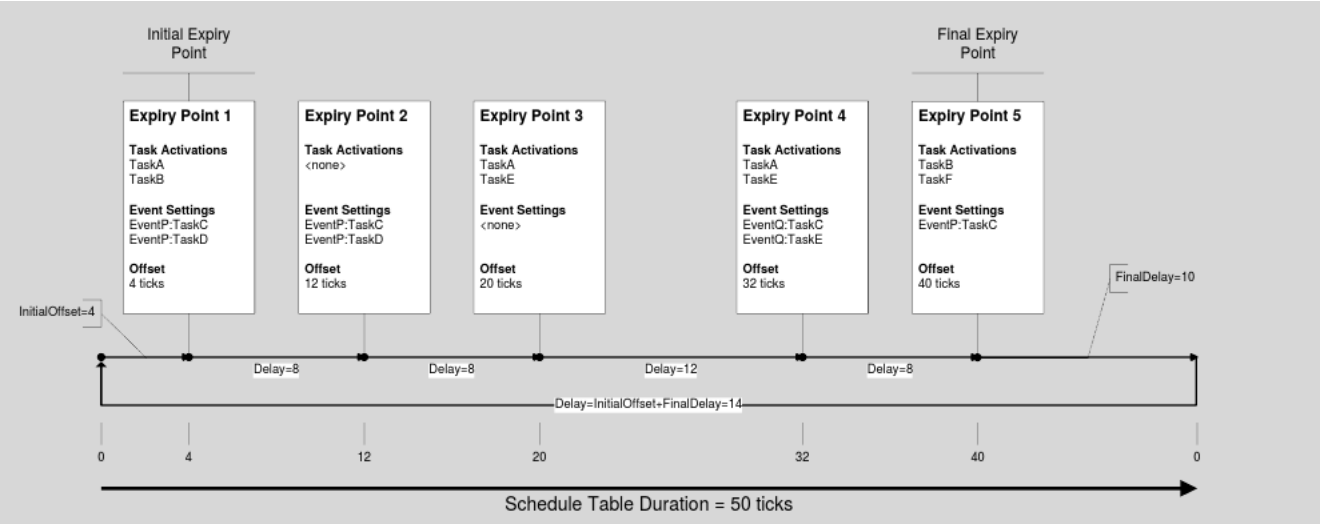


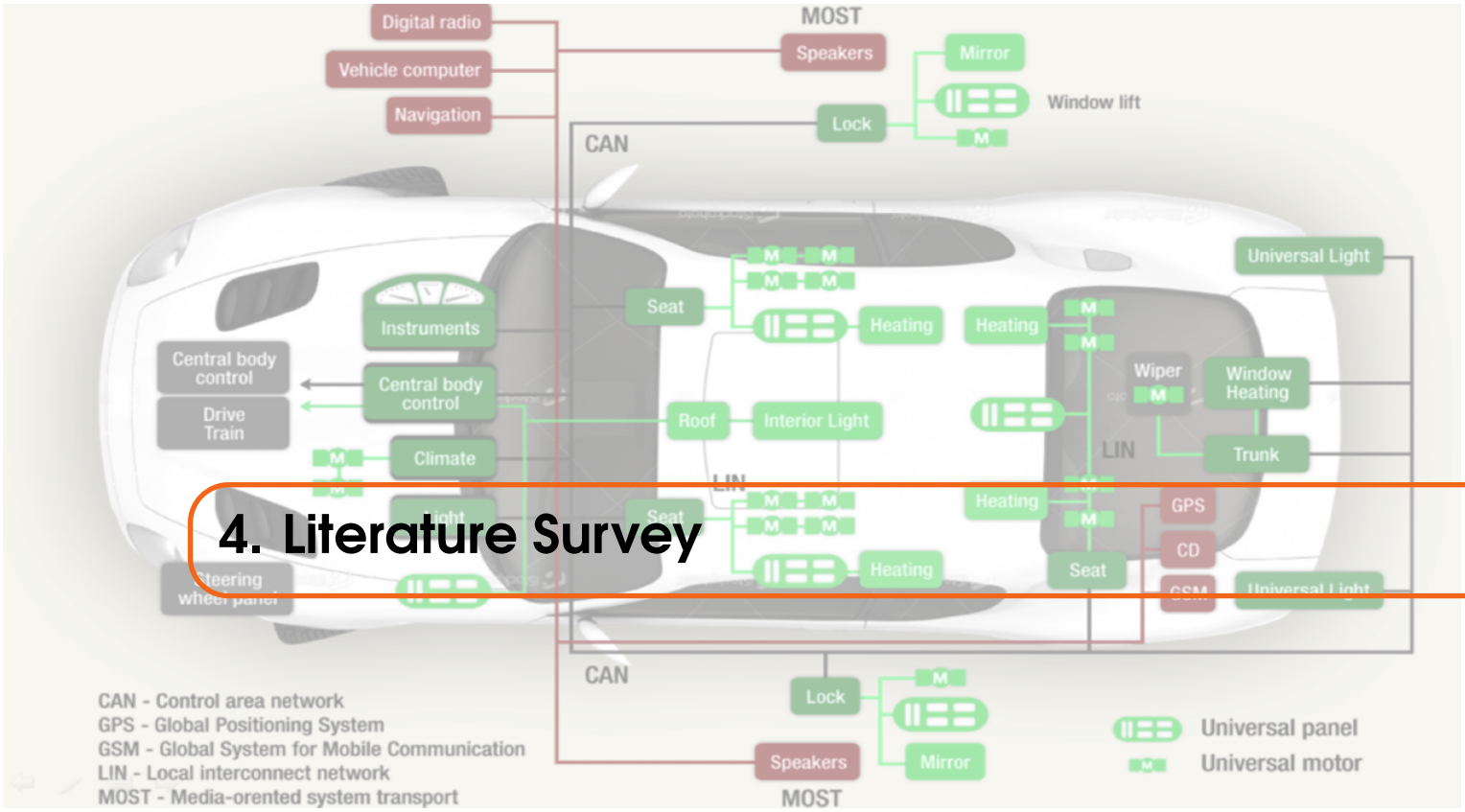
Two different sets of time bases are provided. Time Domains 0-15 are synchronized time bases while Time Domains 16-31 are offset time bases.

3.2.5 Core OS Specifications

In AUTOSAR systems internal communication is provided by RTE or COM at least one of which will be present in the ECUs. AUTOSAR OS thus when used in an AUTOSAR system does not need to support communication. One of the key component AUTOSAR OS extends on OSEK/VDX is Schedule tables. In OSEK it is possible to provide statically defined activation mechanism using an OSEK counter and alarms. To provide runtime modification of the activation, it is necessary to maintain synchronization between the alarms. Schedule table provides a solution to this issue of synchronization by providing an encapsulation to the statically defined expiry point. In a schedule table an expiry point corresponds to one or more actions that should occur in the system and corresponds to a fixed offset from the beginning of a schedule table. Each schedule table is defined by a set of ticks and the os module will iterate over the ticks and invoke appropriate actions at expiry points.

Figure below shows an example of schedule table.





4. Literature Survey

4.1 Adaptive Variable Tasks

4.1.1 Elastic task model for Adaptive Rate Control

A novel periodic task model is proposed in which tasks periods are provided with elastic coefficients. Under this approach tasks can change there execution rate to provide different Quality of Service(QoS). To simplify the schedulability analysis simple assumptions are taken, as is the case with RM and EDF. For which tasks with cyclic execution and fixed demand are considered. But this assumption are too restrictive for application for which timing constraints can be more flexible and dynamic. Works have been done to done theoretical support to such tasks using probabilistic guarantee, capacity reservation based approach or by providing an upper bound on the job chains with variable executions. Varying tasks rate provides the possibility of handling a graceful task degradation rather than outright rejection of some tasks. A number of approaches have been proposed for dynamic and static load balancing to handle overload conditions. In *"QoS Negotiation in Real-Time Systems and Its Applications to Automated Flight Control"* QoS is proposed as a set of negotiation options based on reward and rejection penalties.
This Paper tries to propose a generic way for tasks to change there execution rates and provide varying QoS under different condition. The elastic task model works by compressing the taskset under overload condition. The compression of tasks is handled depending upon the elastic coefficient e. Another advantage of elastic task model is when considering a new task into otherwise schedulable task set. Under a rigid task model this will not be feasible, but by using elastic task compression the new task could be accommodated.
The basic algorithm used to tune the elastic tasks is given below:

Algorithm Task_compress(Γ, U_d) {

$U_0 = \sum_{i=1}^{n} C_i/T_{i_0}$;
$U_{min} = \sum_{i=1}^{n} C_i/T_{i_{max}}$;
if ($U_d < U_{min}$) return INFEASIBLE;

do {

$U_f = E_v = 0$;
for ($each\ \tau_i$) {
if (($e_i == 0$) or ($T_i == T_{i_{max}}$))
$U_f = U_f + U_i$;
else $E_v = E_v + e_i$;
}

$ok = 1$;
for ($each\ \tau_i \in \Gamma_v$) {
if (($e_i > 0$) and ($T_i < T_{i_{max}}$)) {
$U_i = U_{i_0} - (U_0 - U_d + U_f)e_i/E_v$;
$T_i = C_i/U_i$;
if ($T_i > T_{i_{max}}$) {
$T_i = T_{i_{max}}$;
$ok = 0$;
}
}
}

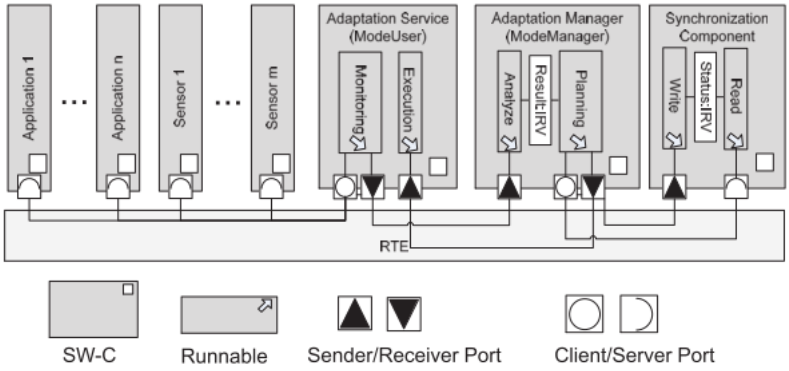
} **while** ($ok == 0$);
return FEASIBLE;
}

4.1.2 Towards runtime adaptation in AUTOSAR

This paper put forwards a method to extend the AUTOSAR mode management to implement runtime adaptation. To realize runtime adaptation in AUTOSAR relocation of software components at runtime is required, which in turn is based on computation of adaptations during runtime. To meet the Adaptation requirements the new model tries to meet to following challenges.

- Real time requirements.
- Self describing components.
- Scheduling.
- ECU Independent addressing.

The runtime adaptation is implemented by two components: Adaptation Service and Adaptation Manager. Relocating software requires control over starting and stopping of AUTOSAR Software Components, this is done by extending the system by a runtime control based on MAPE(Monitor, ANalyze, Plan and Execute) cycle.



- **Monitoring Mechanism** : Monitoring unit extends the mode manager. A request to change the mode to meet the new requirements is initiated to Mode Manager.
- **Analyze**: Mode manager does the analyze and planning phase. A single adaptation manager can exist in entire automotive network or to properly extend Mode manager, one Adaptation Manager shal exist per ECU. This also has the added benefit of redundancy avoiding single point of failure.
- **Synchronization Mechanism**: Mode Manager shall notify the Synchronization manager of the changed execution and will synchronize it across the ECUs in network.
- **Real-Time requirements**: The Adaptation mechanism implemented shall always be determined with respect to the component timing requirement.
- **Self-Describing components**: In order to check for the timing requirements the components shall publish the attributes needed for runtime verification such as end to end deadline.
- Scheduling: EDF Scheduling to provide better efficiency.

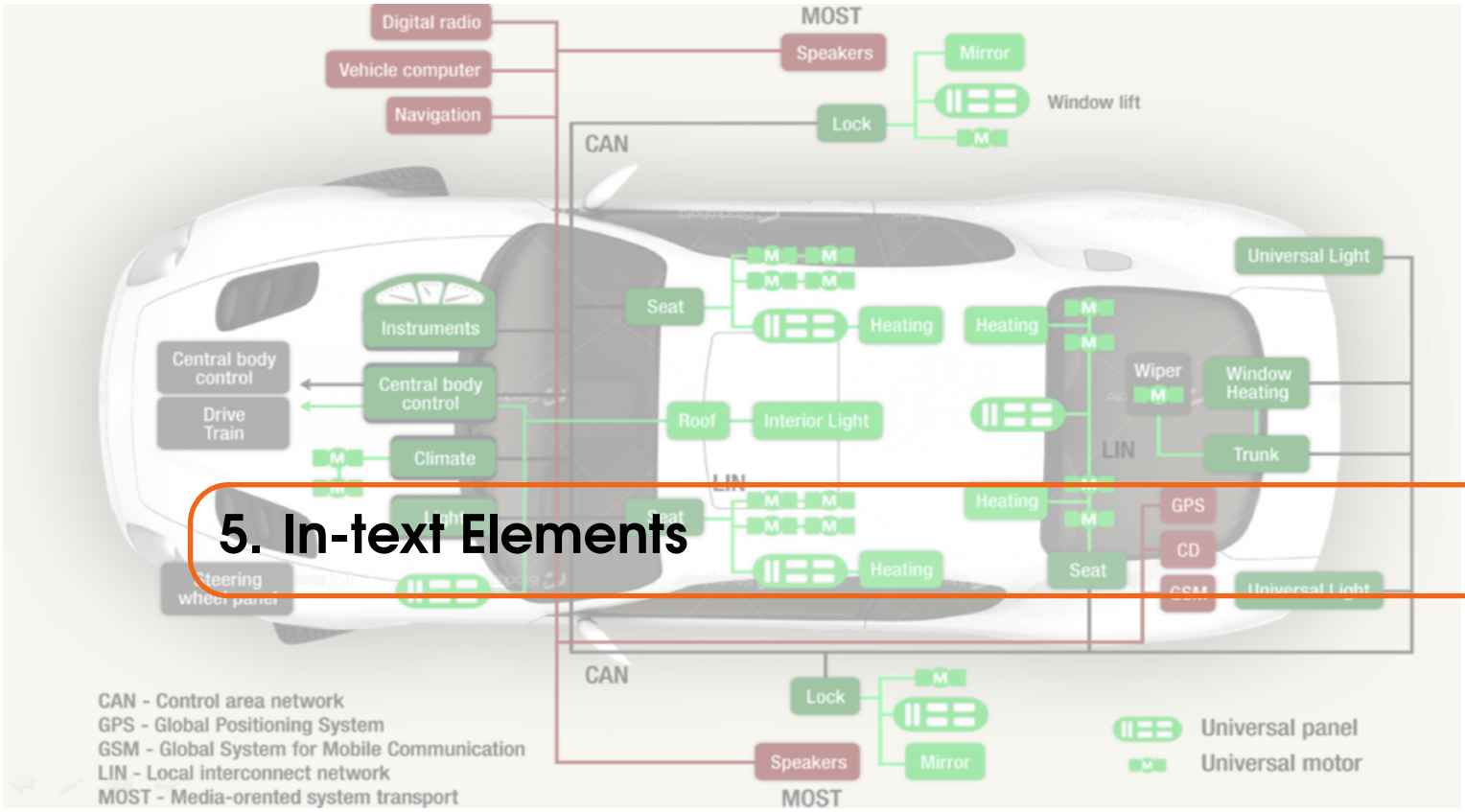
4.1.3 Fast and Tight Analysis of AUTOSAR Schedule Tables.

Proposes how to quantify the response time of the schedule table driven task set considering the offset and resource based priority threshold interference. Two different analysis of rta and maximum time to release of a given task under a given time frame t, as well as maximum time to completion of the given task is put forward. First method being tight fit while the second method is an approximate estimation of the above mentioned parameters.

4.1.4 Evaluation and Implementation of Mixed-Criticality Scheduling Approaches for Periodic Tasks

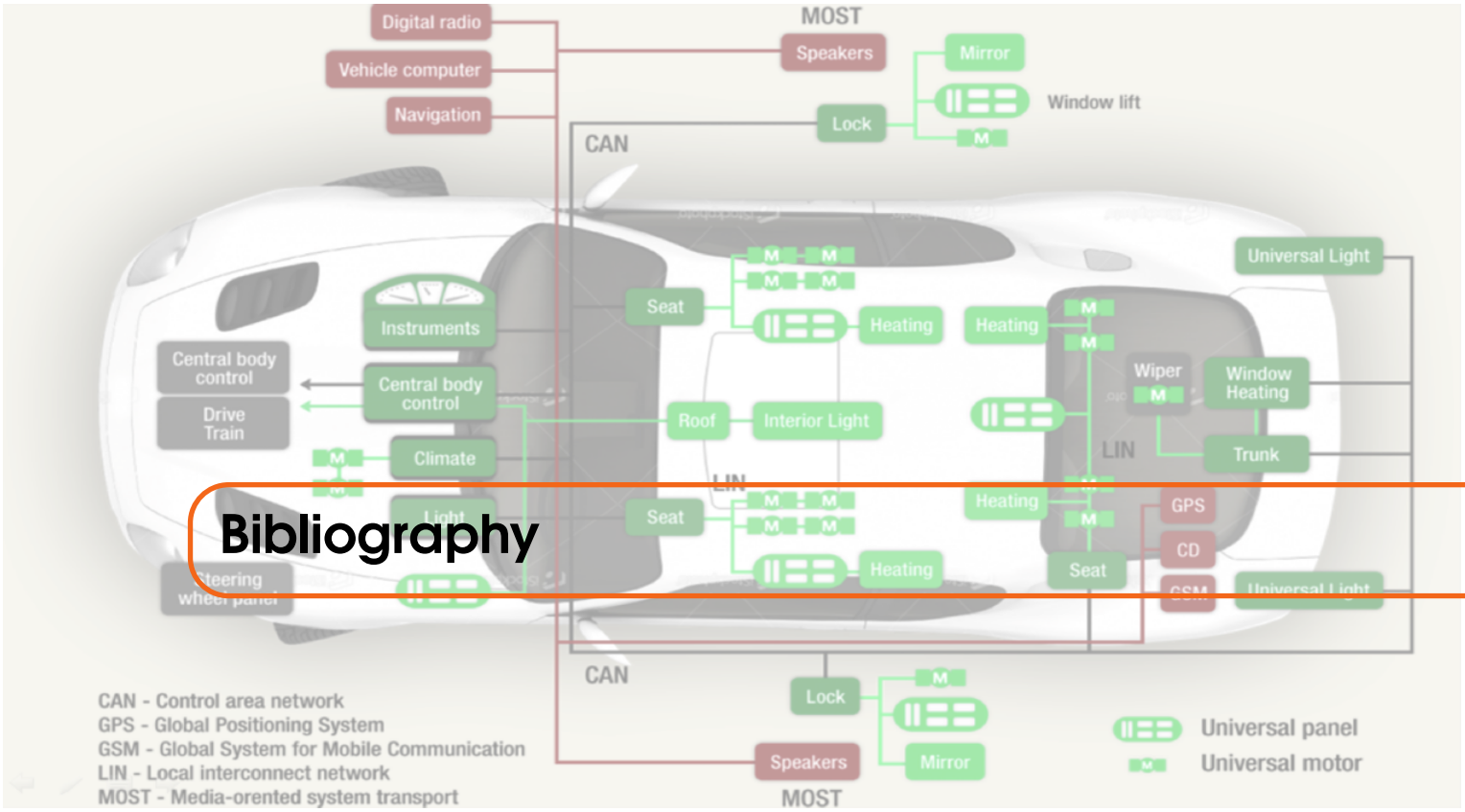
The paper proposes a quantitative comparison of MC Approaches including: Priority assignment, Period transformation and Zero-slack scheduling. Zero slack scheduling is improved by addressing two previously known issues: How to accommodate execution of a task after its deadline and how to handle previously unknown interferences.

4.1.5 Towards the design of certifiable mixed-criticality systems.



5. In-text Elements

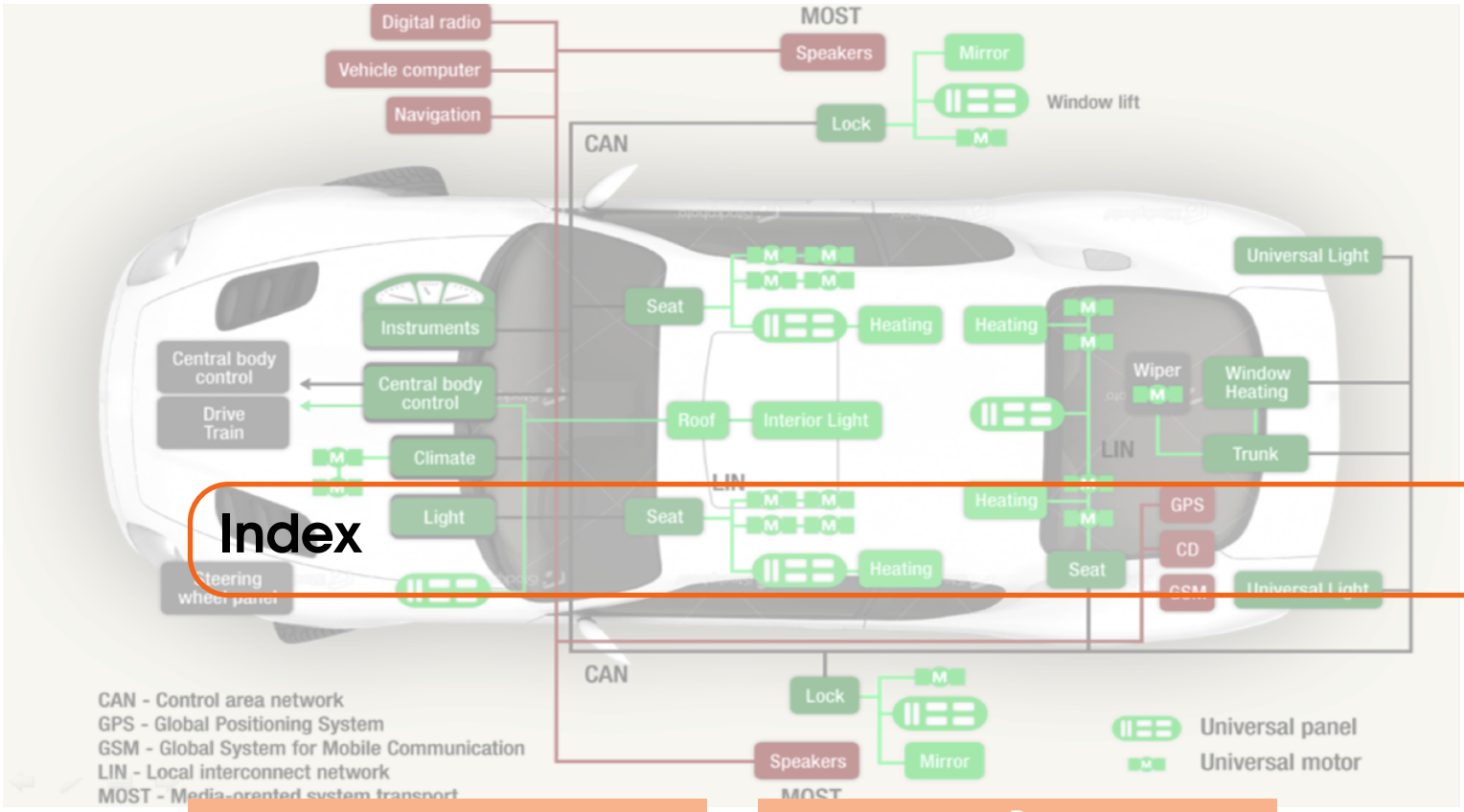
5.1 New section



Bibliography

Books

Articles



Index