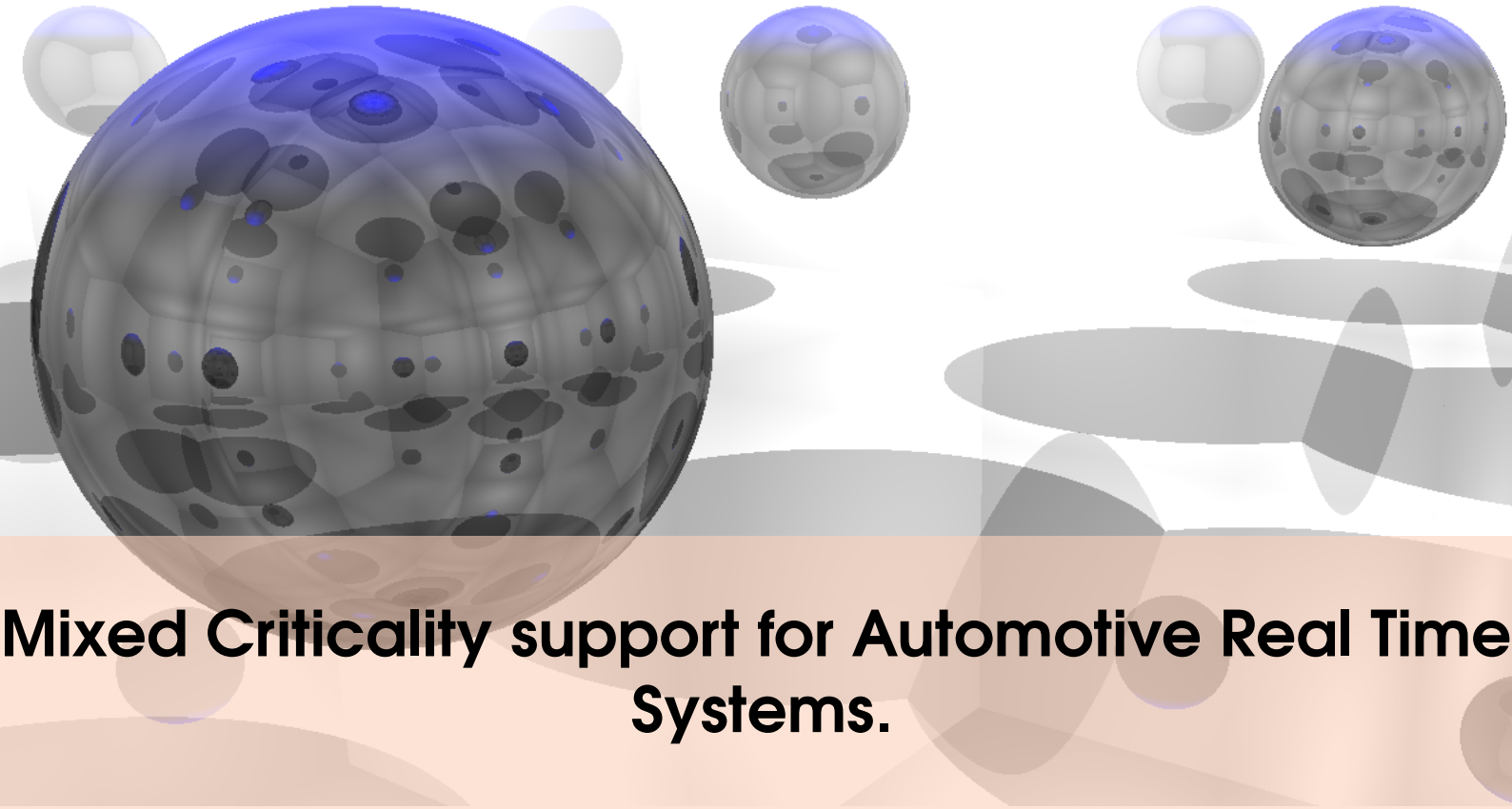


Mixed Criticality support for Automotive Real Time Systems.

PUBLISHED BY PUBLISHER

BOOK-WEBSITE.COM

First printing, March 2013

Contents

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 Hardware Platform

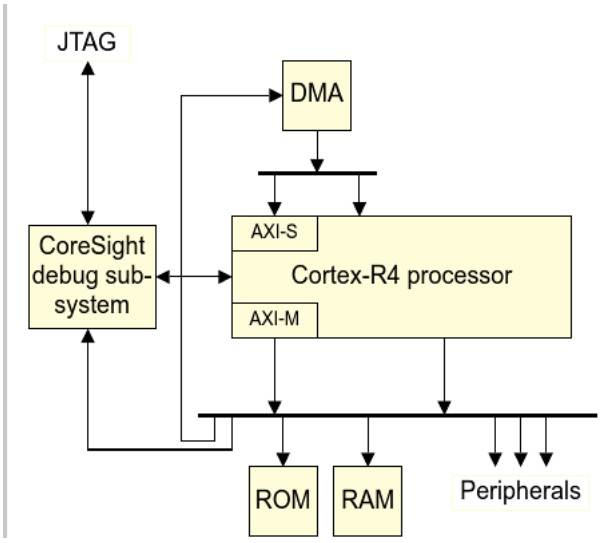
The platform used for the experiments is TI Hercules TMS5703137.
The platform is designed to conform to functional safety standards including ISO26262. The platform employs a dual core lockstep processing using cortex R4. Some of the features which are important for providing AUTOSAR support are mentioned below.

2.1.1 Architecture: 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).

2.1.2 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.

2.1.3 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.

2.2 CORTEX M0+

A low area low power 2 stage pipelined architecture for low power modules.
compact with total of 56 instructions.
Uses AHB lite bus for communication with peripherals and uses 32 bit addressing for a total of upto 4gb of addressable memory.
Support two main control modes: Kernel mode and User mode. User mode can be further divided in to two sub categories of privileged and non-privileged.
PSR or Programme status register has 3 main types: Application Program SR, Interrupt Program SR, Exception Program SR.
CMSIS Support package provided, which is a a standardized way to access peripheral registers and exception vectors.
Memory types can be: Normal, Device, Sharable, Strongly Ordered, Execute Never.

Order of execution of the program instructions can be guaranteed by:

- Data Memory Barrier(DMB): Assures that the memory operations are completed before next memory operation instruction.
- Data Synchronization Barrier(DSB): Ensures that memory operations are completed before start of the next instruction execution.
- Instruction Synchronization Barrier(ISB): Ensures that modification to all the instruction before this operation are reflected for all the instruction that executes after this.

Some use case of these memory barriers are:

- Vector Table: On update of vector table DMB makes sure further instructions are correctly reflected.
- Self Modifying codes: Should make use of DMB followed by ISB.
- Memory Map Switching: DSB followed by ISB.
- MPU Programming use DSB followed by ISB.
- VTOR Programming: While updating vector offset register, use DSB.

2.2.1 AUTOSAR Specific Features

This section covers the features of TMS5703137 platform which are significant in supporting AUTOSAR specifications.

Timing Protection Support

- **RTI**:
  Real Time Interrupt(RTI) provides a time tick source to drive the schedule tables in AUTOSAR.

[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

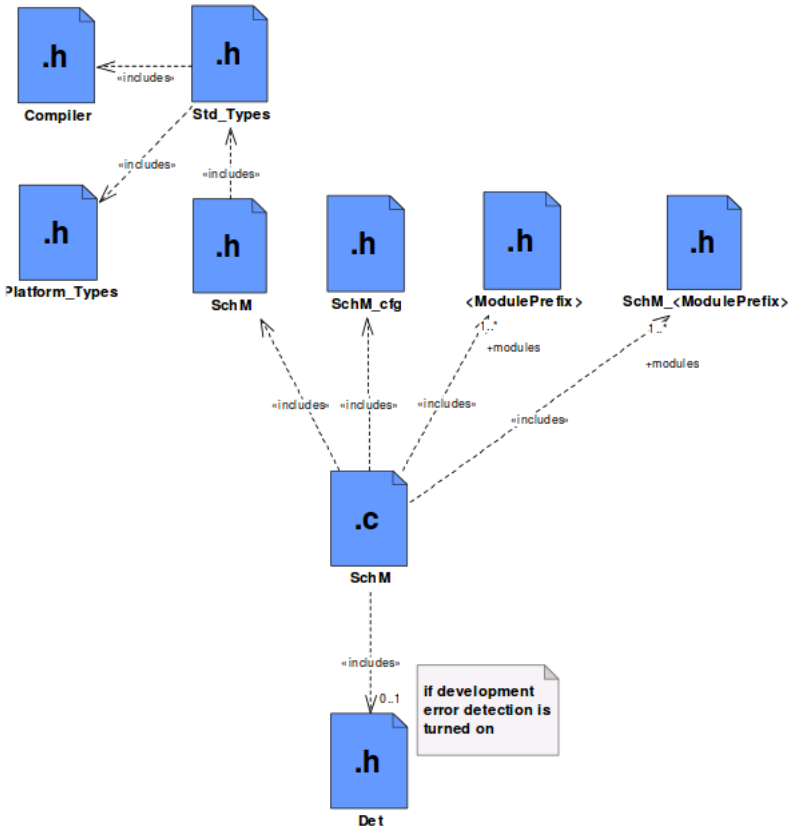


Figure 2: Header file structure of the BSW Scheduler

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

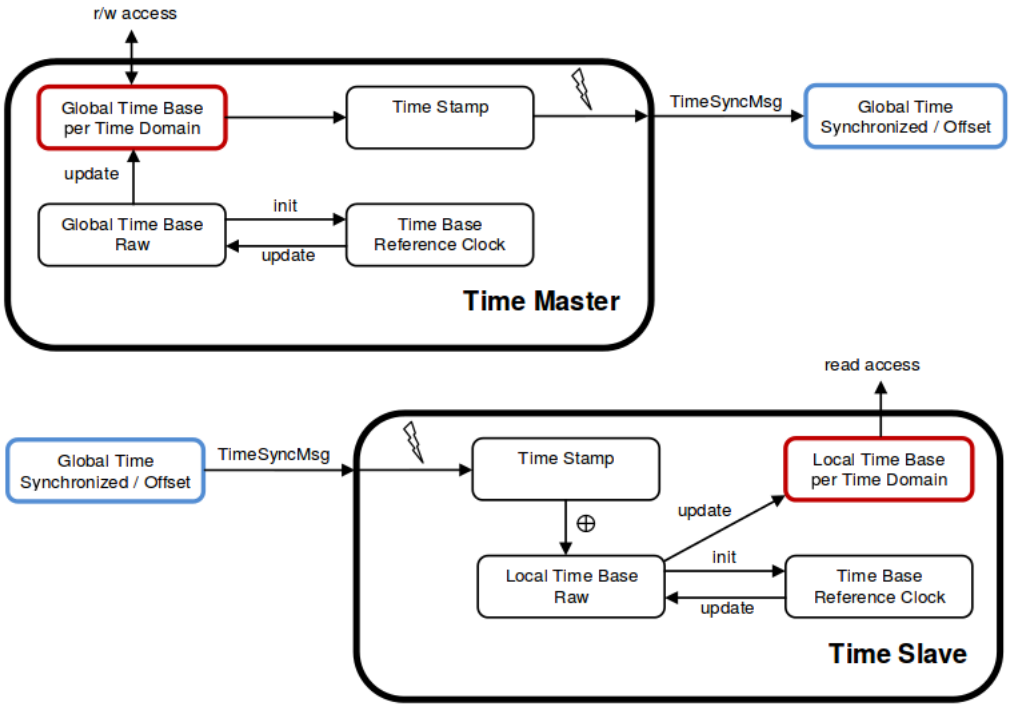
Time Service Module is part of the service layer and provides following features:

- 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

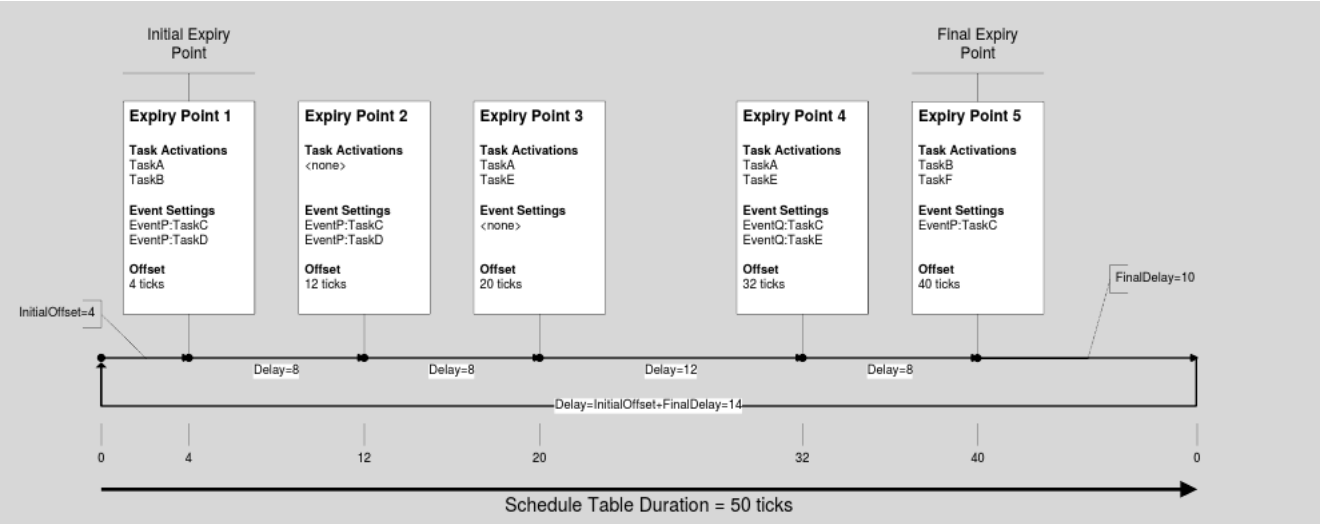
Synchronized time base manager provides APIs to its customers or time bases, which are synchronized with other time bases of the distributed systems, thus acting as a time broker. Time Base Manager acts as the central module to which provides absolute time of the system. Typical use case include Sensor 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.



Schedule Table Processing

- Schedule table lifecycle usually consists of running, next and stopped. For extended tasks with capability of alarms, it includes extra life cycle state of waiting.
- **Repeated and Single-Shot schedule table**: Schedule table may or may not be repeated on completion of execution. single-shot schedule table is useful for processing a sequence of action in response to some trigger. While repeating schedule table is useful for building tasks that do periodic operations or system which needs synchronization with a driver source.
- Periodic schedule table allow value of final delay in range *OsCounterMinCycle* and *OsCounterMaxAllowedValue*. This value is between 0 and *OsCounterMaxAllowedValue* for single-shot schedule table.
- *StartScheduleTableRel()* and *StartScheduleTableAbs()* are to be implemented to provide ability to control schedule table.
- *NextScheduleTable()* is provided to chain another schedule table to be executed at the end of currently executing schedule table.
- Synchronization is necessary in some instance, as the schedule table width may not match with counter modulus. As a result a true repetition of the tasks cannot be achieved.
- There are two types of synchronization available with Autosar:
  Implicit(Counter driving the schedule table is the one with which synchronization is required. e.g. Flexray or TTL) and
  Explicit(Another counter other than the one driving the schedule table is the one with which synchronization is required. e.g Synchronization with periodically broadcast global time.).

- **Synchronization bounds**: The range of adjustment operating system can make are defined by: *OsScheduleTableMaxShorten* and *OsScheduleTableMaxLengthen*.
- APPLICATION_ACCESSIBLE is used to provide access control to the objects belonging to trusted applications, OS provides
- OS should provide memory protection when accesses are made from category 2 ISRs. In case of category 1 ISR, OS is unaware of their existence. :
- **Memory Protection**: OS should provide memory protection scheme based on the data code and stack sections of the executable.
- Memory protection is limited in hardware with no memory protection mechanisms.
- When a memory violation is detected, Operating system module shall raise protection hook with status code E_OS_PROTECTION_MEMORY.
- **Timing Protection**: AUTOSAR as such does not provide deadline monitoring, as the effect may be due to a cascaded failure.
- Provides *execution timing protection* to guarantee statically configured upper bound, called execution budget on the execution time of Tasks and ISRs.
  In case of a timing error due to execution budget being exceeded, Operating System module shall raise *ProtectionHook()* with E_OS_PROTECTION_TIME.
- ***NOTE:** Execution time enforcement requires hardware support, e.g. A timing enforcement interrupt. Its priority should be high enough interrupt the supervised tasks or ISRs.*
- The blocking time from lower priority tasks is prevented by *locking time protection* to guarantee a statically configured upper bound.
  In case of a timing error due to locking operating system module shall call *ProtectionHook()* with E_OS_PROTECTION_LOCKED.
- *inter-arrival timing protection* guarantee a statically configured lower bound called **Time frame** on the time between a task being permitted to transition to READY state or a category 2 ISR arriving. If an attempt is made to release a task before OsTaskTimeFrame, then the operating system module shall not perform the release and shall call protectionHook() with E_OS_PROTECTION_ARRIVAL.

3.2.6 Mode Change Support

An AUTOSAR complaint system can implement mode management support as per AUTOSAR Mode Management Guide.
A mode as per AUTOSAR specification specifies state of ECU wide global variable maintained at scheduler level.
Mode management in AUTOSAR specifically aim to pipeline or synchronise execution of software entities. E.g. Specific modules can be enabled or disabled depending on a mode characteristic or multiple modules can be triggered in an order specified by a set of mode conditions.

- Mode management consists of two parties: A mode manager and set of users who can subscribe to the same.
- Mode users are notified through defined set of mode switches, which can read for currently active mode.
- Two distinctive type of mode communication exists between mode manager and mode user:
  - **Mode Switch**: In mode switch, mode manager notifies all the registered users through registered switches, of the transition from one mode to another.
  - **Mode Request**: Users trigger a request to mode manager to enter specific mode, which can be serviced by the mode manager or ignored.

4. Literature Survey

4.1 Adaptive workload management through elastic scheduling

Real time computing systems, timing constraints guaranteed are based on off line calculated parameters based on fixed parameters and worst case execution time. But a precise estimation of tasks' computation time is tricky due to the unpredictability of low level processor mechanisms including DMA, branch prediction, cache management. An underestimation of the resources results in critical failure, while overestimation will result in wastage of resources. A new methodology is put forward to automatically adapt the task rates of the periodic task set with forcing programmer to provide a priori estimates of the tasks' computation time.
There are task scenarios, for example sensors for robotic perception that should increase the acquisition rate and hence the frequency of execution to meet a given demand.
Similarly, there are situations where a task may experience an overload condition and graceful degradation of the service need to be provided.
Key Idea: *Does not rely on WCET based task budgeting, but uses a dynamic approach to tune the workload utilization.*

- Kuo and Mok(1991): Load scaling technique to gracefully degrade the workload by period adjustment.
- Nakajima and Tezuka (1994): Using real time task to support adaptive application. Period of a task is increased whenever a deadline miss is detected for the same.
- Seto et al.(1997): Method for computing task period to minimize a performance index defined over task set. This approach being effective at design time to optimize a discrete control system.
- Lee et al.(1996): Proposes number of methods to dynamically adjust task rates in overload conditions.
- Becarri et al.(1999): Proposes several method to handle overload through period adjustment, but not to improve utilization in underloaded condition.
- Stankovic et al and Lu et.al proposed feedback mechanism to observe and adjust workload to improve utilization.

Task equated to spring with rigidity coefficient and length constraints.

Note: **Elastic or rigidity coefficient of the task can be set inversely proportional to the importance of the task. Hence a task of lower priority is more acceptable to be shrunk to accomodate a higher priority task.**

Scope

- Main contribution is task compression algorithm. Task budgets are not known a priori and are monitored at runtime.
- Feedback based architecture for elastic rate adaptation. The design of this approach is similar to bandwidth sharing server implementation of Lippari et. al.
- Feedback mechanism is used to measure average execution time c_i and maximum execution time C_i. Execution time estimate is calculated as: $Q_i = c_i + k(C_i - c_i)$. Execution budget is calculated as **sum of (Q_i/ T_i)**.
- This value is used again by the elastic algorithm for the recalculation.

4.2 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 ($T_i > T_{i_{max}}$) {
$T_i = T_{i_{max}}$;
$ok = 0$;
}
}
}

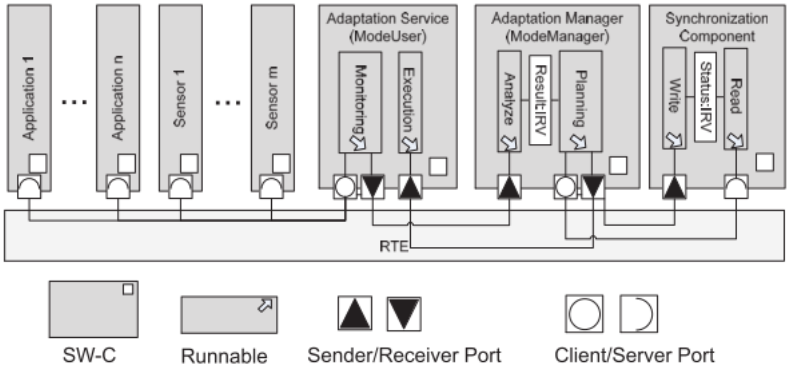
} **while** ($ok == 0$);
return FEASIBLE;
}

4.3 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.

4.4 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.

4.5 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.

4.6 Towards the design of certifiable mixed-criticality systems.

4.7 On the effective use of Fault injection for the assessment of AUTOSAR

Key Idea

AUTOSAR safety standard ISO26262 strongly recommends usage of the fault injection standards, but no definite mechanism exists for enforcing the same. Representation of the faults using the standard fault models e.g. bit flips and data type based corruption are not sufficient to model

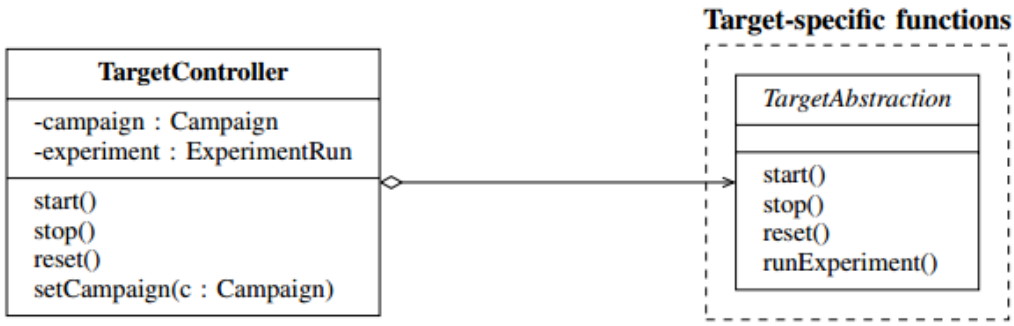


Figure 4.1: Abstract Target representation of GRINDER

real time behavior. The existing Fault Injection framework GRINDER is extended to support AUTOSAR and an assessment is provided.

Width and scope

Provide open source and ready to use framework to do Fault Injection. Dependability assessment on existing AUTOSAR timing monitor safety mechanism for identifying deficiencies and guidelines for derivation of special fault models, injection mechanisms and locations based on abstract AUTOSAR and ISO26262 fault models. Eariler approach mentioned include: Lanigan et.al. and Baugarten et al. First approach being based on VECTOR CaNoe and second approach used anotated SWC Component to introduce fault ports. Another approach is pre implementation testing using UML or AADL, in this failure level are assumed and the effect on model analysed. Vedder et. al extended property based testing to AUTOSAR. Here automated test cases were generated from a pre specified property files.

Hardware based FI: Modeling hardware error such as CAN bus failure, or NVRAM failure through corrupted CRC. Hardware based FI fails to handle component interaction and dependability property. Software based FI: e.g. BeSafe framework to intercept SWC calls and fuzzing error models to check resilience of SWC. GOOFI-2 to evaluate bit flip cases and MODIFI to check model at Simulink level.
Note: Simultaneous fault models with multiple points of failure completely missing from AUTOSAR standards.

Experimental approach

- **Configuration**: Target is instrumented with injectors and detectors.
- **Execution:** The target system is executed till the injection of fault and the perturbation data is successfully completed.
- **Evaluation:** The logs and traces are collected. GRINDER extended to AUTOSAR by providing a target specific implementation of TargetAbstraction Class with which GRINDER interacts during FI.

A general overview of the architecture is given below.

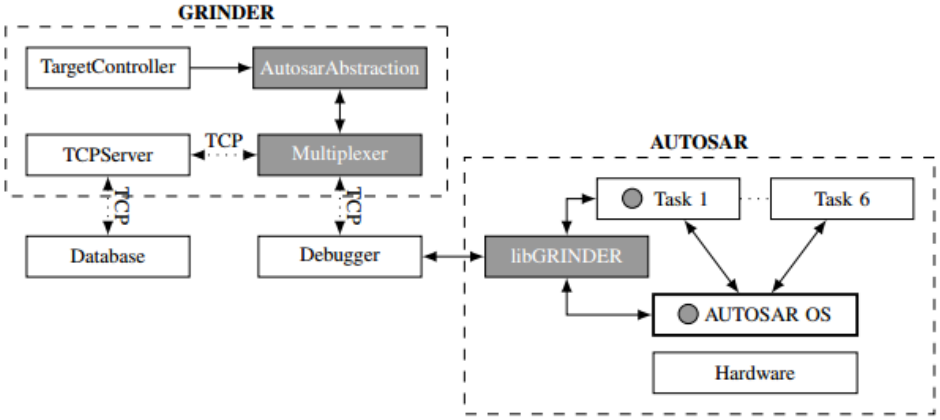


Figure 4.2: GRINDER new arch.

A case study is further done on Adaptive Cruise Control module.

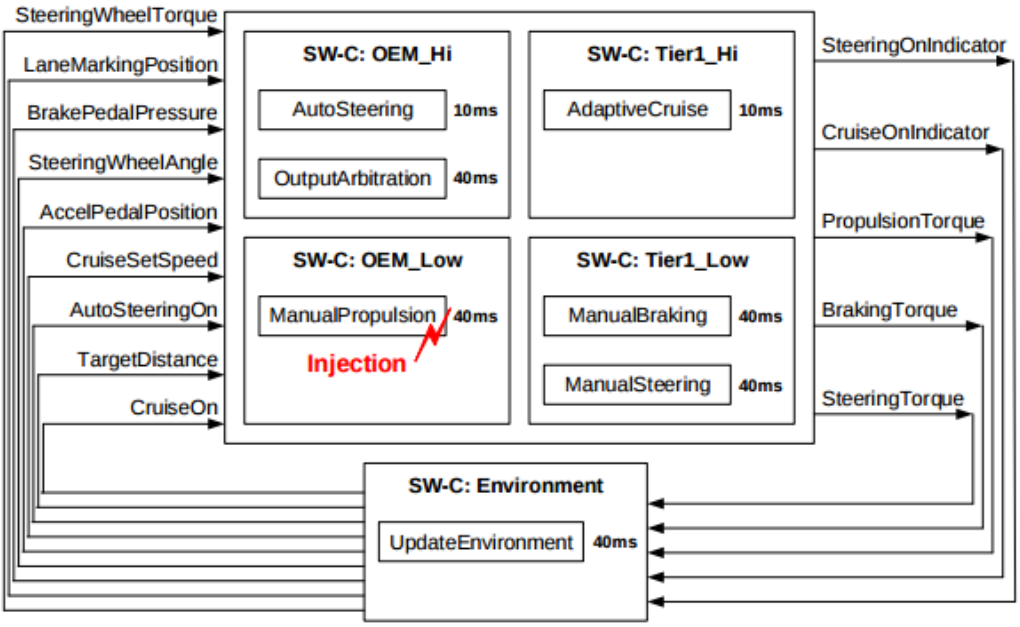


Figure 4.3: ABS Module architecture

Four different scenarios tested:

- Task timing error: Assess the correctness of the error detection and error mitigation of execution time monitoring. Analyze the propagation of the error with and without the presence of the timing errors.
- Interaction between the execution time monitoring and resource lock timing monitoring. Assess correctness and robustness of the mechanisms.

The task selection for monitoring ABS is shown below:

Task name	Priority	Runnable(s)
OEM_Hi_10ms	100	AutoSteering
Tier1_Hi_10ms	90	AdaptiveCruise
OEM_Hi_40ms	80	OutputArbitration
Environment_40ms	70	UpdateEnvironment
OEM_Low_40ms	60	ManualPropulsion
Tier1_Low_40ms	50	ManualBraking, ManualSteering

Figure 4.4: ACC task setup

Conclusion

Detailed description of scenarios, A good framework to test the mechanism of fault injection and different failure scenarios. Can be used with possible change to JTAG interface to FTDI interface. Can be implemented as part of the eclipse plugin as as schedulability and testing mechanism.

Links

http://www1.deeds.informatik.tu-darmstadt.de/External/PublicationData/1/edcc-2015.pdf

4.8 Adaptive Runtime Shaping for Mixed-Criticality Systems

Key Idea

Presents an approach to adaptively shape the inflow workload of low-critical tasks based on actual demand of high critical tasks on runtime. Compared to the shaping of the offline bound low-critical tasks event delay is reduced and system utilization is improved.

4.8.1 Width and Scope

Main contributions are:

- An adaptive scheme for shaping the low critical workload.
- A light weight mechanism with complexity of $O(m.log(n))$ to refine the shaping bound.
- Experimental results to show the efficiency.

This work build on three main approaches:

- Real-Time interface analysis: Connecting real time interface design and calculus.
- Workload prediction: Real time calculus models task activation as event, arrival curves originating from network calculus provide an upper and lower bound on number of arrival events. Prediction method based on historical arrival data. Arrival curve predicted by several stair case functions for tighter prediction.
- Runtime shaping: Shapers often used in regulating packets in network. Greedy mechanism for shaping in real time systems. FPGA based shaping mechanism.

Experimental Approach

Task generation using UUnifast mechanism. Experiments done on MATLAB Based on RTC Tool box(Theele et.al), mainly three different shaping mechanisms are compared.

- Shaping by offline computed bound.
- Shaping by backward derivation online.
- Shaping by proposed lightweight scheme.

Conclusion

TMD

Links

http://www6.in.tum.de/Main/Publications/Biao2015EMSOFT.pdf

4.9 Deadline Analysis of AUTOSAR OS Periodic Tasks in the Presence of Interrupts

Key Idea

Provide an abstract framework to determine if the given periodic task will miss its deadline in the presence of interrupts. Rather than bounding interrupts for given time. Proposes a mechanism to bind the timing calculation to maximum number of interruption allowed to a given task from interrupts.(e.g. The task will meet deadline as long as the number of interrupts is at most n.)

Width and Scope

Provides a mean to abstract representation of the maximum number of interruption allowed to a task. Assumptions part of the approach:

- Tasks are periodic and are implicit in nature with deadline equal to period.
- Any task activated by ISR2 will execute and terminate before the end of the ISR.
- Resource locking and blocking to the tasks due to it are not considered.

Mentions two independent healthiness concept: Task that is not interrupted and tasks that are interrupted.

$$\mathbf{TTI}(T_j, l) =_{df} \begin{cases} \lfloor \frac{l}{De(T_j)} \rfloor \times (ET(T_j) + IT(T_j)) + (ET(T_j) + IT(T_j)) \\ \qquad \text{if } l \bmod De(T_j) \geq (ET(T_j) + IT(T_j)) \\ \lfloor \frac{l}{De(T_j)} \rfloor \times (ET(T_j) + IT(T_j)) + l \bmod De(T_j) \\ \qquad \text{if } l \bmod De(T_j) < (ET(T_j) + IT(T_j)) \end{cases}$$

Figure 4.5: Timing Bound formula

Experimental Approach

Experiments done on Mathematica. Total slack available then split to interrupt times.

Conclusion

Calculate the slack, split it in terms of ISR time.That's main idea.

Link

http://haslab.uminho.pt/jff/files/2013-deadlineanalysisautosar_os.pdf

4.10 Mitigating Timing Error Propagation in Mixed-Criticality Automotive Systems

Key Idea

ISO26262 stipulates freedom from interferences, i.e, error should not propagate from low to high criticality tasks. Different from indirect protection of the critical tasks, approach provides direct low overhead protection to high critical tasks by introducing the concept of preemption budget.

Width and Scope

Introduces the concept of Preemption Budget(PB) and its specifies the maximum amount of time for which a critical task can be preempted. This is very much similar to the concept of Adaptive Mixed Criticality scheduling with deferred Preemption(AMC-DP), which specifies a minimum amount of time for which the task has to be run without any preemption. Timer based approach, For critical task that are active in the run queue timer is started to limit the earliest expiring preemption time. Once the preemption time is depleted the task is moved up the run queue and made to execute for rest of its budget. Handles Transient error runthrough (A mechanism similar to tolerence, where tasks are allowed to stretch the budget). Transient error run through is allowed under PB due to the slack available.

Experiments

Study of Transient and Permanent fault cases for ACC tasks. PBM Budget monitoring implemented under AUTOSAR, measurement made for static code size increase due to the implementation and the overhead incurred due to the budget monitoring.

Conclusion

An alternate take on AMC-DP and Tolerance limit based approach implemented and evaluated to show low overhead.

4.11 A Novel heuristic Algorithm For Mapping AUTOSAR Runnables To Tasks.

Key Idea

Runnables represents the internal behavior of SwC in AUTOSAR and are the smallest pieces of the code to be scheduled. All runnables are triggered in response to an event such as timing event, data receiving or operations invoking the runnables. Key question is deciding upon how many tasks are required to represent a set of runnables, how to define priority and how to define activation offset and execution order of the tasks.
Prominent methods of grouping the runnables to tasks are:

- **Periodic Solution**: Runnables with same period are assigned to one task. WCET of the task is sum of the WCET of all the runnables mapped to it.
- **Multiple Periodic Solution**: In this approach period of the task is the shortest period among different runnables mapped to it.
- **Arbitrary Periodic Solution**: In this approach runnables with different approach are mapped to same task using their activation offset. period of the task is GCD of all the runnables mapped to it.

For APS and MPS system schedulability may be guaranteed using Rate Monotonic Period Assignment(RMPA) or Deadline Monotonic Period Assignment(DMPA). Currently multiple approach to mapping exists, including use of MILP, Simulated Annealing and Genetic algorithm based approach.

Width and Scope

Using Audsley's approach to sort out the tasks among different schedule frames with constraint of width of the frame and the deadline of each of the runtimes.

Conclusion

Neat way, but practicality is limited, as usual runtimes of tasks are grouped to common schedule tables as there functionalities are important.

4.12 Scheduling Algorithms and OS Support for RTS.

4.12.1 Key Idea

An overview of the scheduling support in real time systems and subsequent use cases in commercial of the shelf RTOS. Three main type of scheduling paradigms are discussed:

- Static table driven approaches.
- Static priority driven approaches.
- Dynamic planning based approaches.
- Dynamic best effort approaches.

Discussed are table driven scheduling, priority based preemptive scheduling, cyclic scheduling. Cyclic scheduling resembles the schedule table approach used in autosar based systems. Here tasks are assigned one of a set of harmonic periods. If purely priority driven approach is used, by using task deadlines as priorities, and without any planning, task would be preempted at any time. So unless the deadline is reached or task completes, whichever comes first, It is not possible to know if the task constraints are met. So worst case performance analysis of such a task system is necessary. When tasks are acted upon by parameters other than priority like resource requirements heuristics based approach on branch and bound are necessary.

4.12.2 Ideas

Priority categorization of I/O bound and CPU bound tasks in the context on non-realtime systems.

- In addition to being intuitive, static priority assignment forgoes re-computation of the priority at runtime.
- Similar to EDF, another dynamic approach to scheduling is based on laxity, or least laxity first scheduling.
- Although feasibility checking on schedulability is made easier by preemption, most schedulability analysis ignores the dispatch cose associated with scheduling, which can be significant depending upon the task characteristics.
- Dynamic planning based algorithm: OCBP can be considered a variant of this approach. In this approach a task is guaranteed by constructing a plan for the all the tasks that are available at its possession at the given time.
- Different mechanisms to dynamic planning algorithms exist: Focused addressing, bidding algorithm, flexible algorithm or a combination of bidding/focused algorithm.
- Dynamic best effort algorithm: Used in most of the multicore real time systems. Example being earliest deadline first and the least laxity first algorithms.

4.12.3 Comclusion

Paper is representation of work in the area of scheduling and operating system support for the same.

4.13 Resource Kernels: A Resource-Centric Approach to Real-Time and Multimedia systems.

Main Idea

Design and development of resource kernels. Resource kernels provide timely and predictable access to the multiple system resources, using of wealth of scheduling mechanism running underneath the request APIs.
Main design goals:

- Timeliness of the resources. Admission control policy shold be provided.
- Efficient resource utilization. Have admission control policy with predictable results.
- Enforcement and policy. Resource kernel must enforce the usage of resources such that one application does not abuse the resource usage and hurt other applications.

- Resource kernel must provide access to multiple resource types including processor cycles, disk bandwidth, network bandwidth, communication buffers and virtual memory.
- Portability and automation. Same resource usage can be used on multiple platforms and tuning of the parameters should be automated.
- Upward compatibility with the fielded operating systems.

Concepts

The resource kernel gets its name from its resource centric view and its ability to:

- Apply a uniform resource model for sharing multiple resources.
- Take resource usage specification from application.
- Guarantee resource allocations at admission time.
- Scheduling contending activities based on well defined scheduling scheme.
- Ensure timeliness by dynamically monitoring resource usage and enforcing the actual usage limits.

Some of the APIs to support this usage requirements are exposed which are: *Create, Request, Modify, Notify, Set Attribute, Bind, Get Usage*.
Resource reservations are first class entities and need to be invoked through system calls. Thus enforced by the kernel.

Practical Issues

Using different resources together, is an impractical problem to find an optimal solution to. The problem of scheduling concurrent resources on multiple resources is NP-complete.
One approach to this problem is resource decouping, where each of the resource involved are scheduled independent of each other. For example usage of multiple resources as is case of FFMpeg or similar system involving disk, network of cpu cycles can be split across pipelines and scheduled independently.
Processor codependency is an approach to this solution where two independent resources like disk and network have a common intermediary in the form of CPU.

4.14 On Scheduling Tasks with Quick Recovery from Failure

Proposes a dynamic programming algorithm that ensure that backup or contingency schedules can be embedded in original primary schedule so that hard real time deadlines continues to be met in the face of upto maximum number of processor failure.

Key Idea

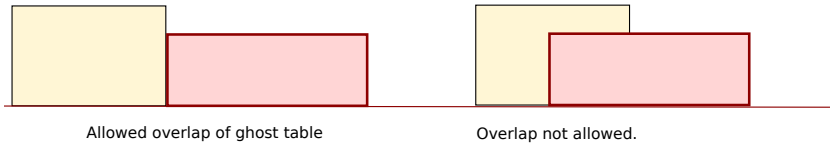
The main aim of this approach is to create a limp mode schedule mechanism, where in spite of failure to few available processor nodes, system is able to meet its real time constraints. The optimization criterion uses cost function, which corresponds to the feedback delay incurred in the control network.
Tasks are run periodically and each instance is known as version. Multiple copies of versions are called *clones*. There are two types of clones: primary and ghost. A ghost clone is a backup copy which remains dormant until it is activated to take the place of corresponding primary or another ghost whose processor has failed.

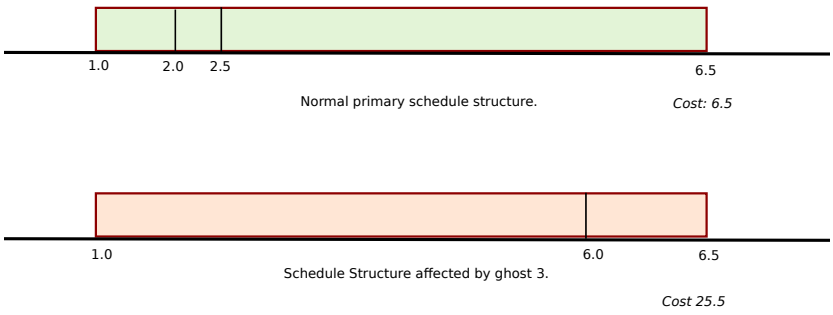
Concepts

An algorithm P is considered consisting of two components: $P1$ and $P2$. P1 finds the optimal allocation of tasks to processor and also optimal schedule by calling P2 which is an optimum scheduler for uniprocessor systems.

Problem Statement

Develop and algorithm Q, that can be used in conjunction with P1 and P2 to obtain an optimal schedule when enough ghosts are incorporated into the schedule to handle N_{sus} processor failures. Schedules designed are considered to be locally preemptive and globally non-preemptive to reduce interconnect overloads.





Scheduling mechanism

Dynamic programming to find the slack holes existing with in the primary schedules. Now do compactness to generate extended holes to schedule ghosts, such that compactness allows the primary tasks to meet there deadlines. In a way similar to the elastic scheduling approach where each task is compacted with restriction being the elasticity coefficient.

conclusion

A partitioned schedule table driven approach with slack stealing across the cores in case of failure. Its main difference from traditional slack stealing is that, the alternate schedule tables are kept on other processors to be activated in case of failure of the primary schedule. And the activation of the

ghost table should not in any way act upon the primary table on the given core.

4.15 On the Scheduling of Mixed-Criticality Real-Time Task Sets

Main Idea

This paper introduces the concept of Zero slack scheduling, In zero slack scheduling the task consist of two execution modes: normal and critical. When a task is unable to meet its budget demands in normal mode, it switches to critical mode at the last instance at which it is able to meet the demand in critical mode.

4.16 Schedule Table Generation for Time-Triggered Mixed Criticality Systems: Jens Theis et.al

Main Idea

Heuristic search algorithm for developing schedule tables of different criticalities.

- Builds on previous work on implementing mixed criticality in Time Triggered domain.
- Two different schedule tables are maintained in the previous approach and priority of each job based on its criticality level.
- The priority ordering is inflexible resulting in total utilization which is lower than EDF.
- Use of mode change schedulers to accommodate two different criticality levels(Changing operational modes in the context of pre-run-time scheduling.).
- Two schedule tables are constructed and switching from Lo to High is possible at every point in time called *Switch through property.*
- Each job consists of low and high mode states: J_{lo} *and* J_{hi}. High critical definition consists of J_{hi} and δJ_{hi}.
- Precedence constraint introduced to prevent δJ_{hi} from getting scheduled before J_{lo}.
- Granularity of the Scheduler is schedule table width and hence preemptible at these boundaries.
- Each scheduling decision for both tables represented as an edge in search tree, based on iterative deepening.
- Feasible schedule table design done using backtracking based tree search.
- Concept of *leeway* is introduced as heuristic function to aid backtracking mechanism. For low criticality jobs it is the difference between the current time and its deadline. For high criticality jobs it the difference between deadline of the job in low crit mode and current time, reduced by the δJ_{hi} of all jobs that could executed between current time and deadline of δJ_{hi} of current job.
- During tree searching for viable schedule backtrack to previous node for a negative leeway.

Model

- **Low crit Table**: Pick Job with earliest deadline that is released –> Consider for leeway –> Backtrack if necessary.
- **High Crit Table**: Based on decision of low crit table, pick a Job. Pick a δJ_{b}ased on precedence constraint (J_{lo} before respective δJ_{lo}). In case a δJ missed its deadline backtrack.

Backtracking Heuristics:

Backtracking step involves swapping current node with a node higher up in tree. And is based on:

- Release time should be later than release time of Job i.
- The leeway of possible swap slot must be greater than or equal to difference in number of slots between current slot and candidate of the swapping slot.
- After swapping recalculate the leeways of these tasks.

- If precedence constraint is changed or scheduled demand of δJ is changed, start new schedule decision from swapped position.

4.17 Using Dual Priority Scheduling to Improve the Resource utilization in nMRPA Microcontrollers.

Main Idea

Handling of tasks when the system enters a state that is different from normal running state for which certification of the system has been done.

- Uses three different queues to separate the executing tasks, viz., run queue, interrupt queue and long task queue.
- tasks are picked first from active queue, then interrupt queue and from long task queue in that order.
- Over a heuristically determined number of periods, if same task is executing, it is moved to long task queue.
- Main aim of the approach to bring flexibility to the static scheduler.

4.18 Time-Triggered Mixed-Critical Scheduler: Dario Socci et.al

Main Idea

A scheduling approach on single core when the exact arrival time are known a priori. This paper proposes a generalization of the Single Time Table Per Mode Scheduling proposed by Baruah et.al .

- Baruah et.al proposed time triggered version of OCBP. The scheduling algorithm uses one static table per criticality mode also known as *Single Time Table Per Mode*(**STTM**).
-

4.19 Mixed-Criticality Scheduling in Time Triggered Legacy Systems

Main Idea

A method to add handling of criticality changes to existing schedule tables for legacy TT systems.

- A simple online mechanism that executes the jobs according the existing schedule table, manages a change of criticality and continues to execute the high criticality job set.
- Method based on slot shifting. Table and constraints are analysed and unused resources and leeways are determined. This corresponds to spare capacity.
- The *spare capacity* is used to provide flexibility and handle firm aperiodic tasks at runtime.
- In the offline phase of the slot shifting complex precedence constraints are resolved.
- Task scheduling is considered in terms of time slots or instances. The main idea of scheduling is based on utilizing slack available in each such slot. Slack calculated as:
  $SC^{LO}(I_i) = |I_i| - \Sigma\, C_{LO} + min(sc^{LO}(I_{i+1}, 0))$
  $SC^{HI}(I_i) = |I_i| - \Sigma\, C_{HI} + min(sc^{HIs}(I_{i+1}, 0))$

Online Decision Process

Three separate run queue are maintained: One for low criticality tasks, one for high and third one consisting of all the tasks. During scheduling, for overrun of slack for a given slot is compensated from the immediate next slot, and recovered during idle slot.

- When the low critical slack for current slot is greater than or equal to zero($s_{lo} >= 0$), schedule tasks with earliest deadline from the runqueue.
- If $s_{lo} < 0$, Task with earliest deadline from the high crit runqueue.
- $s_{hi} < 0$, indicates possible failure of schedule.

4.20 Integrated Time- and Event-Triggered Scheduling: Overhead analysis on ARM Architecture.

Main Idea

Analyses the slot shifting approach to integrate time triggered and event triggered scheduling of Real Time Systems.

- Implements a slot shifting algorithm and analyses the cost of adding flexibility to offline schedules due to addition of aperiodic tasks.
- Three important issues are considered:
  - While measuring runtimes, avoid distortions caused by other processes executing in the system or caused by interrupts of the hardware systems.
  - The operating system can be a source of distortion in measurement, as processes running in parallel can trigger actions on the bus and can result in unpredictable cache behaviours.
  - A suitable platform is required to reproduce runtime information.
- MPARM hardware simulator is used for the experiments.

Model and Notes

- Background discusses some previous approaches in this direction, to assign aperiodic task along with periodic tasks.
- Discusses usage of server task to reserve bandwidth for aperiodic task. Other server algorithms are noted.Advantage being better service for aperiodic tasks at the cost of pessimistic reservation of the bandwidth.
- Constant bandwidth servers are similarly mentioned, which in addition to providing service to aperiodic tasks maintains temporal isolation.
- Slack stealing algorithm, acting as master procrastinator with periodic tasks to service aperiodic tasks, But hampered by excessing computation costs.

Slot Shifting Algorithm

- Aims for predictable flexibility on top of an offline schedule.
- Consists of an offline and online component.
- **Offline Mode**:
  - Create distributed schedule table with precedence.
  - Determine earliest start time and latest finishing time for all tasks.
  - On each node, deadlines of all tasks are sorted and offline schedule is divided into disjoint intervals(In AUTOSAR world: Schedule Table.), with each interval consisting of tasks with same deadline.
  - Slack in system defined as:
    $sc(I_j) = |I_j| - \Sigma\ WCET(T_i) + min(sc(I_{j+1}), 0)$, Where I_j is the effective width of time interval.
  - So slack is interval width decreased by sum of WCET of tasks belonging to given interval and amount of slot/slack lent to next time interval.
- **Online Mode**:
  - For each new slot(strict, periodically repeating, time-interval. Roughly equivalent to time-tick cost + a non-preemptible width.), scheduler invoked on each node independently. Each slot having scheduling cost(t_s) and execution budget(t_e).
  - Each schedule point, check for aperiodic task arrival, **check for total slack available till deadline of aperiodic task, if it meets the task demand run the aperiodic task else reject.**
  - Similarly for an idle instance, run aperiodic task, decrease from current time interval.

Implementation

- *task_list* and *interval_list*: interval list maintains grouped tasks and trigger events.
- Timer ISR to run the decision algorithm on aperiodic tasks.
- Aperiodic tasks arrived handled in earliest deadline first.
- Accommodating aperiodic task done either by running in background or recreating the tables till the aperiodic task deadline.(Done only if task passes acceptance test.)

Experiment

- Simulator based test bed.
- logging in shared memory and dump to interface to reduce overhead.
- Experiment 1: Overhead of slot shifting online algorithm.
- Experiment 2: Overhead of online splitting of schedule table.
- Experiment 3: Optimizing splitting overhead by aligning aperiodic task to existing time interval boundary.
- Experiment 4: Scalability on multicore.
- Aggregated histogram plot on logging overhead, ISR overhead, schedule list updation overhead, acceptance test overhead, polling for aperiodic task overhead.
- Memory overhead analysis.

4.21 RTOS support for mixed time-triggered and event-triggered task sets

Main Idea

Extend an off the shelf RTOS(In this case μC/OS-II) with time triggered implementation and slot shifting to support time-triggered and event triggered tasks.
Tasks executed based on earliest deadline first and aperiodic tasks accommodated if it passes a predefined acceptance test(As is the case in above paper using slack).

- Implements dynamic slot shifting.
- Additionally implements an enhanced mechanism for slack reclamation from periodic tasks.(Based on the fact that periodic task may finish before there specified WCET, thus proving more slack than computed based on an offline algorithm.)

Some key differentiation of slot shifting from other approaches

- Similar to the reservation based approach, distinguishing characteristics being:
  - Lack of temporal isolation between tasks.
  - Tasks can execute over multiple budget(or Time Intervals, or Schedule Tables as is case with AUTOSAR).
- Compared to hierarchical scheduling, provisioned spare capacity neither follows a periodic service pattern nor provides a constant service rate. Disjunct timing intervals created rather than regular intervals as in case of hierarchical approach.

Model and Notes

- New structure introduced similar to Task Control Block(TCB) - Interval Control Block(ICB), to store offline specified task intervals with tasks, count, start time, end time, spare capacity etc.
- Resource management done at tick ISR.
- TCB extended to keep track of execution budget assigned/remaining and ICB to which the task belongs to.
- Experiment section involves comparison of overhead in slot shifting with respect to Fixed Priority Scheduling and Earliest Deadline First Scheduling.

5. Automotive control systems

5.1 Power-Steering Control Architecture for Automatic Driving

A two layered control architecture to automatically moving the steering wheel is presented.

Overview

- Two layer control architecture, first layer is designed to calculate the position of the steering wheel at any time based on fuzzy logic.
- The second layer is a classic control layer that moves the steering wheel to the position predicted by the first layer, monitored by Real-Time Kinematic Differential Global Positioning System(RTK-DGPS).
- Comparison is done for the performance of the implementation to human driver.

Key points

- There are two ways to implement automatic steering control, Imitating a driver or using dynamic models of car and control methods based on linear control theory.
- the main contribution of this work is the combined use of a PID and a fuzzy controller in a cascade-control scheme, and its application to regulate the steering wheel of a mass-produced vehicle.
- The sensor input comes from GPS receiver at a frequency of 10Hz.
- Each measured value is compared with reference GPS trajectory. Two input variables are gathered from this comparison: the lateral and angular errors.
- A Fuzzy control ingests these two inputs to generate target steering turning command.
- Fuzzy control performs three main functionalities: fuzzification, inference, defuzzification.
- Two different driving conditions are handled. Straight road and curved road.
- Straight road steering angles are considered short and fast reactive while curve is considered higher steering angles and slower reaction.
- Steering angle is limited to 2.5% on straight road. While in curve mode the angle is not limited.

- Center-of-area method is used for defuzzification.
- Each crisp value, which is left or right steering output is multiplied by square of weight and averaged over sum of weights.
- The weights are calculated using Mamdani's rule inference method.
- The trajectory control defined above is fed to a PID controller that is tuned on the basis of Zeigler-Nichols method and further tuned experimentally, proportional, integral and derivative gains were adjusted to yield an overdamped closed-loop response.
- Proportional, integral and derivative gains were set to 20, 50 and 1000 respectively.

Pseudo codes

Algorithm 1 First layer Fuzzy Controller

```
1: procedure FUZZY_RULE()
2:     if Angular_Error == Left || Lateral_Error == Left then return Right
3:     if Angular_Error == Right ||Lateral_Error == Right then return Left
```

reference

Paper Link:

5.2 Predictive Active Steering Control for Autonomous Vehicle Systems

A Model Predictive Control (MPC) approach for controlling an Active Front Steering system in an autonomous vehicle is presented.

5.2.1 overview

- Early work on active safety focused on improving longitudinal dynamics part of the motion, in particular, on more effective braking system(ABS) or traction control system(TCS). ABS increases the braking efficiency by avoiding the lock of braking wheels. TCS prevents wheel from slipping at the same time improves vehicle steerability and stability by maximising the tractive and lateral forces.
- This was followed by work on vehicle stability control system known under different acronyms such as Electronic Stability Program(ESP), Vehicle Stability Control(VSC), Interactive Vehicle Dynamics(IVD), Dynamic Stability Control(DSC) etc.
- Other subsystem that are being investigated involves 4 wheel steering, active steering, active suspension, active differential etc.
- Focus on control of the yaw and lateral vehicle dynamics via active front steering.
- The control input is the front steering angle and the goal is to follow the desired trajectory or target as close as possible while fulfilling various constraints reflecting vehicle physical limits and design requirements.
- The future desired trajectory is known only over finite horizon at each time step. This is done in the spirit of Model Predictive Control.
- Two different formulation of Active Front Steering Model Predictive Control is presented and analyzed.
- Non Linear Vehicle model to predict future evolution of system. A non linear optimization problem is solved at each computation instance.
- Second approach uses a sub-optimal MPC Controller based on successive on line linearization of the non-linear model. Approach also known as LTV(Linear Time Varying Model.)

subsectionKey Ideas

- Bicycle model for the vehicle.
- Control system consisting of trajectory planning - low level control system and actuator system.
- Guidance and Navigation Control System - Trajectory-Mode Generator + Trajectory-Mode Replanning + Low Level Control System.
- Trajectory-Replanner :- Receding Horizon Control design.
- Vehicle-Model :- Rear centered kinematic model with acceleration, steering, speed, steering rate and rollover constraints.
- Computation heavy non-linear optimization model.

5.2.2 Conclusion

Too complex a model to be used for analysis of the scheduling patterns.

6. Proposed Design

6.1 Scheduler Design

Table 6.1: MC Approaches in Time Triggered Systems.

STTM Based MC in Time Triggered Systems	data	data	data	data
MC in Time Triggered Legacy Systems	data	data	data	data
MC In Time Triggered Single and Multicore	data	data	data	data
Towards Runtime Adaptation in AUTOSAR				

7. Test and Measurement Methodology

7.1 Approaches to test and measurement under AUTOSAR and ISO26262

This section is a overview of approaches to time measurement and testing while being compliant to AUTOSAR standards.

FTQ/FWQ

The FTQ/FWQ benchmarks measure hardware and software interference or 'noise' on a node from the applications perspective.
FWQ(Fixed Work Quanta) and FTQ(Fixed Time Quanta) runs on each core and hardware thread within a single node via pthreads. FWQ continuously measure the time taken to execute a fixed amount of code. FTQ repetitively works for a fixed amount of time and measure the amount of work done in the given time period.
Due to the fixed work approach of FWQ, the data samples can be used to compute useful statistics (mean, standard deviation and kurtosis) of the scaled noise (sample time minus the minimum work time and scaled by the minimum work time).
Due to the fixed time quanta approach of FTQ, the work data can be processes with a Fast Fourier Transform (FFT) in order to determine the temporal frequency of software interference. This is an extremely useful tool to find sources of periodic interference such as scheduling intervals, the regular operation of daemons, etc.

7.2 Time measurements on TI TMS-570-LS3137

The main approaches to logging under ti hercules platform are discussed here.

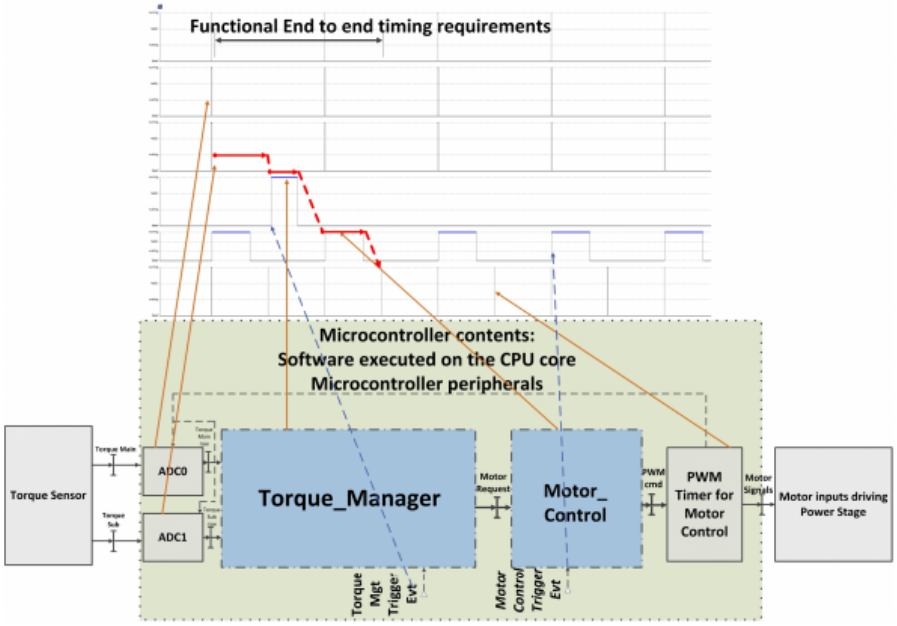
7.3 A Practical Approach to the Simulation of Safety-critical Automotive Control Systems considering Complex Data Flows

overview

- Issue of determining end to end latencies in systems composed of functional and dysfunctional modes of operations.
- Depending on required safety level the complexity of implementation varies.
- Development consists of finding the real time bottlenecks, verifying and deploying, and as such is complex.
- Aims to provide virtual integration of system early in development stages.

Main points

- Important task is planning execution of tasks and ISRs.
- Model based simulation of various execution scenarios to determine the worst case overhead of task executions.
- Event chain used to specify a sequence of execution and data flow, subject to RT requirement.



- Initially data flow between hardware and software components at system level were modeled using flow port concept of *SysML*.
- Formalized model analyzed using INCRON tool-suite for:
  - Deadline Violation.
  - End to End latency.
  - Start to Start jitter.
  - CPU peak load in certain averaging intervals.
  - Response time distribution.
- Aspects of system configuration to be considered for optimizing event chains:
  - Periods of the time-triggered tasks.
  - Number and decomposition of the processes.
  - Execution order.
  - Core affinity.
- Iterative design flow defined. Consists of:
  - Specify structural system architecture.
  - Identify interaction between application and basic software.

- Specify the dynamic system architecture comprising of tasks and their basic timing properties.
- Specify functional data and control flow.
- Perform safety analysis and identify dysfunctional control flow for all relevant faults.
- Define, Create and run simulation on timing analysis tool like chronSIM.
- Extract timing properties and define the architecture.

Reference

https://hal.archives-ouvertes.fr/hal-01291352/document

8. In-text Elements

8.1 New section

Bibliography