

Mode-Dynamic Task Allocation and Scheduling for an Engine Management Real-Time System Using a Multicore Microcontroller

Jin Seo Park
Infineon Technologies Korea Co., Ltd.

Jens Harnisch
Infineon Technologies AG

Michael Deubzer
Timing-Architects Embedded Systems GmbH

Keum Cheol Jeong
Hyundai Motor Co.

Patrick Leteinturier
Infineon Technologies AG

Il hong Suh
Hanyang Univ.

ABSTRACT

A variety of methodologies to use embedded multicore controllers efficiently has been discussed in the last years. Several assumptions are usually made in the automotive domain, such as static assignment of tasks to the cores. This paper shows an approach for efficient task allocation depending on different system modes. An engine management system (EMS) is used as application example, and the performance improvement compared to static allocation is assessed.

The paper is structured as follows: First the control algorithms for the EMS will be classified according to operating modes. The classified algorithms will be allocated to the cores, depending on the operating mode. We identify mode transition points, allowing a reliable switch without neglecting timing requirements. As a next step, it will be shown that a load distribution by mode-dependent task allocation would be better balanced than a static task allocation. All EMS tasks being applied in this paper serve for a 4 cylinder gasoline torque based system including engine low level drivers. The Infineon AURIX microcontroller is used, featuring three cores. Performance evaluation is done by simulation on the chosen abstraction level.

The last section of the document will address proposals for future work to measure and reduce the behavioral differences between the simulation and the implementation on the real target.

CITATION: Park, J., Harnisch, J., Deubzer, M., Jeong, K. et al., "Mode-Dynamic Task Allocation and Scheduling for an Engine Management Real-Time System Using a Multicore Microcontroller," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 7(1):2014, doi:10.4271/2014-01-0257.

INTRODUCTION

The introduction of multicore systems is able to reach a higher performance, which is not possibly easily now by only increasing the clock rate. As we know, Higher clock rates will lead to an exponentially increasing in power dissipation, that causes thermal issues and as well as EMV problems.

Furthermore, an exceptional high design efforts are necessary in designing higher clock rate devices. Surely in the future, clock rates will still be increased to some degree; however, the main key to achieve more performance will rather be multicore systems.

On one hand, multicore system may generate some challenges for software development. This could be due to additional synchronization and data exchanging needed between cores, leading to certain software overhead. On the other hand, multicore systems can also offer an option to better structure the software, for example to have time driven tasks on one core, and event driven tasks on another core, or to have the software assigned onto the different cores by ASIL level.

When a multicore microcontroller is chosen for a general control system, usually tasks are most efficiently allocated to cores after the calculation of worst case response time of each task, considering the task execution time. In existing schedule designs, the operating modes are usually not considered. The applied Engine Management System (EMS), one of powertrain systems, features the following modes: IG-On, Cranking, Idle, Part load and Wide Open Throttle (WOT). If those modes are properly considered in the load balancing across the cores, this may lead to relaxed performance requirements for the microcontroller, and eventually to lower system costs. This paper suggests a model-based scheduling design for multicore processor systems.

TASKS FOR ENGINE MANAGEMENT SYSTEM [EMS]

Torque Based Engine Management System

Torque based engine control is a control algorithm to manage the increasing system complexity. A torque-based engine control architecture uses a central torque demand variable to control the regulating qualities as shown in Figure 1. This torque demand variable is the result of the coordination of all torque requests throughout the vehicle. Therefore, the system manages the whole process of prioritizing the torque demands of the different vehicle subsystems such as electronic stability or transmission control.

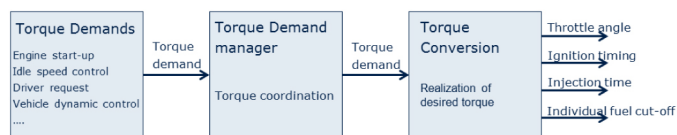


Figure 1. Torque flow in torque based EMS, aligned on [2].

Operating Modes for Engine Control

Generally, a torque-based EMS has 5 operating modes as shown below and in Figure 2.

1. Ignition(IG) key-On mode: Engine off state
2. Cranking mode: Engine starting state
3. Idle mode: a throttle valve is not opened (the driver does not kick the acceleration pedal)
4. Part load mode: a throttle valve is partially opened
5. Wide Open Throttle (WOT) mode: a throttle valve is 'wide open' (the driver kicks and acceleration pedal fully)

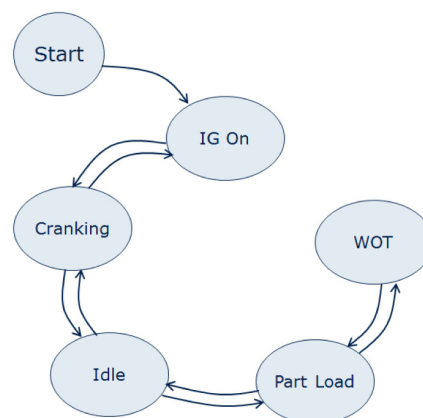


Figure 2. State Diagram for Operating Mode of Engine Management System.

This paper considers only general operation, and not failure cases. Therefore, mode switching driven by a failure case is not considered. IG-on mode and Cranking mode are driven by the engine operating status. Idle mode, Part load mode and WOT mode are driven by the throttle valve position. See Figure 3.

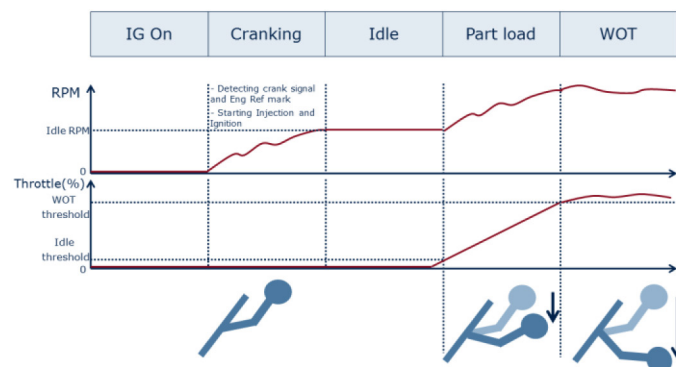


Figure 3. Possible scenario for Operating Mode transition.

Task Scheduling

Tasks are implemented as schedule-sequences that contain a sequence of runnables to be executed in the specified order and at a given priority level upon occurrence of a certain activation event (Figure 4). For the task scheduling in an EMS, a combination of preemptive and cooperative scheduling is used. And to assure analyzability of the system, usually static task scheduling is used.

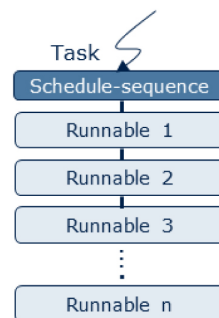


Figure 4. Sequence of runnables called within a Task, [3].



Documents

Document Sharing Hub

Task can generally categorize into two; Event-driven and Timing-driven. Event driven tasks usually have short latency requirements which requiring preemptive scheduling while timing driven tasks can be scheduled cooperatively. Figure 5 shows the hierarchical scheduler concept where the cooperative scheduler is subordinated under the preemptive scheduler. The cooperative scheduler is treated as a single task at the lowest priority level of the preemptive scheduler. In existing operating systems, like OSEK VDX [9], realization of different preemptive groups are configured by task group parameter.

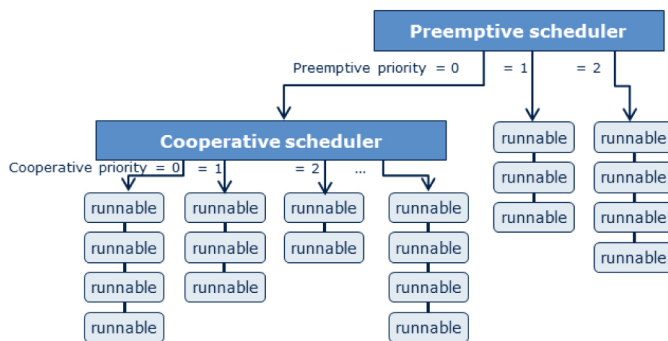


Figure 5. Hierarchy of Scheduler in an EMS system, aligned to [3].

For the EMS system, major tasks selected in this paper are shown in Table 1.

Table 1. Major tasks in EMS.

Scheduling Type	Task Name	Period	Priority	
			preemptive	cooperative
Preemptive	T_Crank	every 6 degree of engine	3	-
	T_Eng Syn	every 180 degree of engine	1	-
	T_1ms	1ms	2	-
Cooperative	T_10ms	10ms	-	6
	T_20ms	20ms	-	5
	T_50ms	50ms	-	4
	T_100ms	100ms	-	3
	T_200ms	200ms	-	2
	T_1000ms	1000ms	-	1

CONSIDERATION FOR TASK ALLOCATION AND SCHEDULING

Beyond the prioritization problem, as seen in multicore systems, the scheduling decision has become an additional dimension for optimization. This means the choice of assigning execution of task to core, also known as allocation problem. In Table 2, a categorization of the prioritization and allocation problem is given. Migration indicates the dynamic behavior regarding task allocation during runtime. No migration means all instances of a

task execute on the same core, job-migration means different instances can execute on different cores, however a task instance once started at a core has to stay on the core, section-fix migration means dedicated areas of a task instance may execute on different cores and full migration means a task instance can migrate after preemption to any arbitrary core. Regarding priority value, a scheduler is classified in the following types. A task-fix priority scheduler assigns all instances of a task to the same priority, a job-fix priority may assign different instances different priorities, a section-fix priority may assign different predefined areas a different priority and a dynamic priority scheduler is able to change the priority at any time. The interruption type defines when a task can be preempted. Cooperative tasks have defined preemption points. Additionally there are non-preemptive and fully preemptive tasks. A work-conserving algorithm newer set a core to idle when there is an active task, a non-work-conserving scheduler may also set a core to idle when there are tasks active. This might lead to a better load balancing. In this paper, we focus only on static prioritization of task.

Table 2. Categorization of allocation (i.e. Migration), prioritization, preemption type and work conserving property.

		1	2	3	4
Migration	M	No(local scheduling)	Job-	Section	Full (global scheduling)
Priority	P	Task-fix	Job-fix	Section-fix	Dynamic
Interruption Type	I	No preemptive	cooperative	preemptive	
Work Conserving	W	Work-Conserving	Non Work-Conserving		

Regarding allocation, scheduling can be divided in the following two general groups; Dynamic task and Static task:

Dynamic Task Allocation

Dynamic task allocation scheduling, i.e. global scheduling, allows a core to select a task from the set of all tasks, even if a task has been allocated to a core; that is, one core can take an allocated task from another core. With global scheduling, a core may select only tasks allocated to other cores, thus increasing the number of tasks allocated to it (assuming that other cores do not choose its tasks).

Dynamic task allocation has the problem of overhead when come to the context migration of an executing task from one core to another core. However, a dynamic execution load can be handled more flexible.

Static Task Allocation

Static task allocation scheduling, i.e. local scheduling, requires each core to select only from its own set of allocated task, thus selection of tasks from other cores are not possible. Static task