March 2009.

OSEK/RTOS & OSEK turbo Introduction

Christian Michel Sendis
Field Applications Engineer
OSEK/VDX Introduction

OSEK/VDX™

- Began in French and German automotive industries
- Steering committee includes BMW, Daimler Chrysler, Robert Bosch, Siemens, Volkswagen and several other prominent automotive companies.
- Is now the standard for most Operating Systems used in the Automotive Industry.
- Standards-based: ISO Standard 17356

The name OSEK/VDX:

“Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug”
(“Open systems and corresponding interfaces for automotive electronics”)
“Vehicle Distributed eXecutive”
Different standards defined in OSEK/VDX

There are three main standards in OSEK/VDX:

- **OS (Operating System version 2.2)** – Provides a common API
- **COM (Communication version 2.2.2)** – Provides the standard interfaces and protocols for data exchange
- **NM (Network Management version 2.5.1)** – Provides the standard functionality to ensure proper operation of in-vehicle networks

OIL- (OSEK Implementation Language version 2.3) – Provides system configuration and object description (OS and internal COM)

ORTI (OSEK Real Time Interface) – provides debuggers with OS Aware information

Typical OSEK implementation uses OS, OIL, and subset of the other components
OSEK/VDX OS Introduction

Features that influenced the architectural choices when designing the OSEK Standard:

- Scalability
- Portability of software
- Configurability
- Statically allocated OS
OSEK OS Executes in a Single ECU

OSEK OS SERVICES

- Task Management, Scheduler
- ISR Management
- Resource Management
- Counters
- Alarms
- Events
- Communication
- Error Handling, Hook Routines

ECU

Memory
MCU
I/O
Power
Tasks in OSEK/VDX OS
Tasks in OSEK/VDX OS

A task provides a frame for executing functions. Tasks will execute concurrently and asynchronously.

OSEK provides two kinds of tasks:

Basic tasks - no waiting state. Synchronization possible only at start and end of task

Extended Tasks – can use the call to WaitEvent(). Event Synchronization possible several times inside the task.

The termination of a task instance only happen when a task terminates itself (to simplify the OS, no explicit task kill primitives are provided).
Task state transitions

- **Waiting**
  - Transition: Wait
  - Transition: Release

- **Ready**
  - Transition: Preempt
  - Transition: Start

- **Running**
  - Transition: Terminate

- **Suspended**
  - Transition: Activate

**Notes:**
- Basic & Extended tasks
- Extended tasks only
Task activation

A task may be activated (transferred from suspended to ready state) by:

- the OS, at system start-up
- another task (service calls: ActivateTask, ChainTask)
- an ISR (service call: ActivateTask)
- an alarm expiration
- sending a message

An extended task is transferred from waiting to ready state (released) by setting an event

Tasks states changes are triggered by the application and the scheduler
Task Termination

- **Tasks must terminate with one of the following API calls**
  - Terminate Itself (with call to TerminateTask())
  - Terminate itself and activate a different task (with call to ChainTask())

- No “Kill Task ” service is provided. This simplifies the OS.
Scheduling policies in OSEK/VDX OS
Reminder of terms related to scheduling

Scheduling
The processor assignment of the tasks is called scheduling. All tasks which have the state **ready** may assign the processor (if tasks were activated or an event occurred).

**Non preemptive scheduling (cooperative)**
Tasks cannot be preempted by other tasks during their execution.
With the preemptive scheduling tasks with high priority delay tasks with lower priority.

**Preemptive scheduling**
Tasks may be preempted by other tasks during runtime.
A low priority task does not cause a delay for a higher priority task if preemptive scheduling is used. ⇒ fast reaction time of the more important/critical tasks

**Static scheduling**
The processor assignment of the tasks with static scheduling has a predefined sequence. The sequence of the task execution is determined at compile time.

**Dynamic Scheduling**
The decision which task is executed is determined at run time. The scheduler adapts to the current task situation.

OSEK uses priority-based dynamic scheduling
OSEK Scheduling Policy

OSEK scheduler will select the highest priority task from the list of ready tasks. It will select the oldest task if more than one of same high priority exists.
Scheduling Policy

- **High** priority tasks are executed first.
- **Low** priority tasks are executed after high priority tasks.

**FIFO list of ready tasks**

Tasks are added to a FIFO list according to their priority.

**CPU**

Tasks are executed on the CPU in the order they are added to the FIFO list.

**Scheduler active**

The scheduler is active when a task is ready to be scheduled.

**Processor time**

Tasks are executed during processor time.

- **CPU**
- **Scheduler active**
OSEK can have 4 different preemption policies:

The preemption policy can be chosen by the user:

Four preemption policies:
- Non preemptive
- Full preemptive
- Mixed preemptive
- Groups of tasks (cooperative)

**Application behavior is specified by task priority and scheduling policy**
Non preemptive scheduling

Task n

Task 2

Task 1

Latency time

Task activated

Scheduler running

Task terminated

Task terminated

Higher Priority

Lower Priority
Full preemptive scheduling

Task n
- Suspended

Task 2
- Suspended
- Running
- Suspended

Task 1
- Running
- Ready
- Running
- Suspended

Task preempted
Task activated
Task terminated

Higher Priority

Lower Priority
Groups of tasks / Cooperative scheduling

Task 1

Task 2

Task 3

Res1

Groups of tasks / Cooperative scheduling
Conformance Classes
OSEK/VDX OS Conformance Classes

Conformance classes exist to allow partial implementations of the standard along pre-defined lines.

The conformance classes specify different requirements for the following attributes:

- Multiple requesting task activations (only one activation or more than one)
- Task types (basic tasks only or basic and extended tasks)
- Number of tasks per priority (one or more than one)
<table>
<thead>
<tr>
<th>Task Characteristic</th>
<th>BCC1</th>
<th>BCC2</th>
<th>ECC1</th>
<th>ECC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic tasks</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Extended tasks</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>One task per priority</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Multiple tasks per priority</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>No multiple activations</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Multiple activations for basic tasks</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Interrupts in OSEK
Interrupt Processing

Nested interrupts are possible
Interrupt Service Routines have highest priority in the system

Interrupts are the key part of any real-time operating system
2 categories of ISRs

Category 1 *)

ISR (ISR_NAME)
{
    /* code without OS calls except interrupt */
}

Category 2

ISR (ISR_NAME)
{
    /* code with OS calls */
}

*) Usage of ISR macro is OSEKturbo specific
Resource management OS services
Resource management

When using resource management

- Two tasks/ISRs cannot “own” the same resource at the same time
- Priority inversion cannot occur while resources are used
- Deadlocks do not occur as a result of using resources
- Tasks accessing resources never enter a Wait state

OSEK Priority ceiling protocol implemented as a resource management discipline

Predefined resource RES_SCHEDULER

- Resource with ceiling priority higher than all tasks

Concurrent access coordination of shared resources

- Task A wants Resrc B
- Task B wants Resrc A

Deadlock
Priority Inversion problem (without OSEK)

- **Task n**: Suspended
- **Task 3**: Suspended → Running → Wait on release resource → Running
  - Resource denied
- **Task 2**: Suspended → Ready → Running → Suspended
  - Task terminated
- **Task 1**: Running → Ready → Running → Ready
  - Task preempted
  - Occupy resource
  - Release resource
OSEK resource types

Standard resource
  • OSEK Priority Ceiling Protocol

Linked resource
  • Has the same properties like another existing resource (priority, type)

Internal resource
  • The internal resource is not visible for the user.
  • The internal resource has the same behavior as the standard resource (OSEK Priority Ceiling Protocol etc.).
  • Groups of tasks is realized through the use of internal resources (Cooperative scheduling).
OSEK Priority Ceiling Protocol

- **Task n**
  - Suspended

- **Ceiling Priority**
  - Running
  - Running
  - Task 2
  - Suspended
  - Ready
  - Running
  - Running
  - Suspened
  - Task 1
  - Running
  - Ready
  - Running

- **Lower Priority**
  - Request resource
  - Task preempted

- **Actions**
  - Release resource
  - Task terminated
  - Request resource
OSEK Summary

- System elements are statically defined
  - All tasks execute out of Flash, EEPROM, or other non-volatile memory
  - No dynamic task allocation
  - No dynamic memory allocation or heap used
  - Fixed stack size
  - Benefits
    - The required resources are fixed quantities
    - System architecture is simplified
- Task priorities are fixed
  - Simplifies task scheduler
  - Scheduler can temporarily adjust task’s priority level to resolve deadlocks
- Provides set of OS APIs
  - Offers services for:
    - Task management and synchronization
    - Data exchange
    - Resource control
    - Interrupt handling
  - Uses C-style syntax
OSEKturbo
OSEKturbo Summary

- Freescale offers an efficient, scalable, reliable implementation of OSEK/VDX called OSEKturbo.
- High quality, efficient, reliable and scalable OSEK/VDX™ compliant Operating System
- Can be used in any embedded application that needs a small footprint RTOS (occupies less than 2K of memory on most controllers)
- Supports deterministic scheduling (timing analysis) to uncover timing problems early in the design phase
- Works seamlessly with CodeWarrior Development Studio with RTOS Aware debugger functionality
- Supports industry leading compilers CodeWarrior, Diab, Cosmic, IAR and more
- Support for HC(S)08, HCS12(X), DSC and Automotive Power Architectures (MPC5xx, 55xx, MPC5200 and MAC7x00) from Freescale.
- Industry Leading Support, Training and Services are available to support your customers.
OSEKturbo Benefits

- The smallest and fastest, fully-certified OSEK implementation available
- Multiple MCU support
- High speed performance and low RAM usage
- Support for all conformance classes available
- ORTI for debugging (debugging standard interface)
- Integration to Codewarrior
- Builder tool for easy configuration of OSEK applications
Building an OSEKturbo application (make)

OSEK Builder

Application configuration file (OIL)

Debug interface file (ORTI)

System Generator

 Compiler/Linker

Executable file

- Make tool
- CodeWarrior tools & related files
- OSEK components, tools & related files
- User written / defined

Sysgen source code

User’s source code

OSEK OS source code
Building an OSEK turbo application (IDE)

OSEK Builder

Call for edit OIL

CodeWarrior Project

Application configuration file (OIL)

Sysgen source code

User’s source code

OSEK OS source code

Debug interface file (ORTI)

Call for compile OIL

System Generator

Call for compile C code

Compiler/Linker

Executable file

Call for compile OIL

User written / defined

OSEK components, tools & related files

CodeWarrior tools & related files

User written / defined

Building an OSEK turbo application (IDE)
OSEK Implementation Language - OIL

OIL_VERSION = "2.3";
IMPLEMENTATION OSEKturbo_OS12_2_2_1_59 {
  OS {
    ENUM WITH_AUTO [BCC1,ECC1] CC = AUTO;
    ENUM [STANDARD,EXTENDED] STATUS;
    UINT32 [0,1,2,4] DEBUG_LEVEL = 0;
    BOOLEAN STARTUPHOOK;
    BOOLEAN ERRORHOOK;
  }
  COUNTER {
    UINT32 [ 0x0 .. 0xffff ] MINCYCLE;
    UINT32 [ 0x1 .. 0xffff ] MAXALLOWEDVALUE;
    UINT32 TICKSPERBASE;
  }
  ...
};

CPU cpuname {
  OS osname {
    CC = ECC1;
    STATUS = EXTENDED;
    STARTUPHOOK = TRUE;
    ERRORHOOK = TRUE;
  }
  TASK InitTask {
    PRIORITY = 30;
    SCHEDULE = NON;
    AUTOSTART = TRUE{
      APPMODE = appmode;
    }
    ACTIVATION = 1;
  }
  COUNTER SYSTE MTIMER {
    MAXALLOWEDVALUE = 0xFFFF;
    TICKSPERBASE = 10;
    MINCYCLE = 0;
  }
  ...
};
Builder V2.3, workspace
Builder 2.3, tasks
At www.freescale.com

Most of your problems solved in less than 24 hours!

Use the Freescale Web Technical Support!

Enter your request details here!

Email: support@freescale.com