OSEK realtime operating system

osCAN TMS470

User's manual

Author:	Dipl.-Inform. Joachim Stehle, Dipl.-Inform. Marcel Läzer, Dipl.-Inf. Ulf Vatter
Date:	2008-04-17
Document version:	3.08
State:	Released

Contents

1 Overview

This documentation describes the implementation specific part of the OSEK[1] operating system for the Texas Instruments TMS470 microcontroller family. In the documentation a processor of the family may be referred as TMS470.

The common part of all *osCAN* implementations is described in the separate document /osCAN/.

The implementation is based on the OSEK-OS-specification 2.2.1 described in the document „OSEK/VDX Operating System“Version 2.2.1 /OSEK OS/. This documentation assumes that the reader is familiar with the OSEK specification.

OSEK is a registered trademark of Siemens AG.

[1] (Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug, Open Systems and their interfaces to the Electronic in motor vehicles)

2 Related Documents

Abbreviation	File Name	Description
/OSEK OS/		OSEK/VDX Operating System Specification 2.2.1 [2]
/osCAN/	TechnicalReference_osCAN3.pdf	User manual of Vector *osCAN*
/osCAN_HW/	Osektms470.pdf	User manual of Vector *osCAN*; Hardware specific part (This document)
/OSEK COM/		OSEK/VDX Communication – Version: 2.2.2 [2]
/OIL/		OIL: OSEK Implementation Language – Version: 2.3 [2]

Table 1: Documents

[2] This document is available in PDF-format on the Internet at the OSEK/VDX homepage (http://www.osek-vdx.org)

3 Overview of OSEK properties

3.1 Overview of properties

OSEK OS specification:	2.2.1
OSEK COM specification:	2.2.2
OSEK OIL specification:	2.3
Conformance class:	BCC1, BCC2, ECC1, ECC2
Scheduling policy:	full-, non- and mixed- preemptive
Scheduling points:	depending on scheduling policy
Maximum number of tasks:	65535
Maximum number of events per task:	32
Maximum number of activations per task:	255
Maximum number of priorities:	8192
Maximum number of counters:	256
Maximum number of alarms:	32767
Maximum number of resources:	8192
Maximum number of resources locked simultaneously:	255
Maximum number of ISRs:	limited by the hardware
Maximum number of messages:	see OSEK intertask communication in /osCAN/
Status levels:	STANDARD and EXTENDED
OSEK-COM:	CCCA, CCCB
Nested Interrupts:	Possible
Interrupt level resource handling	Available
ORTI	Available (V2.0 and V2.1)
Procedure Support:	Available (without stack optimization)

Supported C-Compiler/Linker:	Green Hills Multi2000 v4.0.2
	Green Hills Multi2000 v5.0.2
	TI CCS v2.21 (Compiler v2.52)
	TI CCS v3.1 (Compiler v4.1.3 in "`–abi=tiabi`" mode)
	TI CCS v3.3 (Compiler v4.4.4 in "`–abi=tiabi`" mode)
Supported CPUs:	TI TMS470R1
	TI TMS470R1_SE with IEM support
	PL2002EX
	TI TMS470PVF24x
	TI TMS470PVF34x
	TMS470PVF145B

4 Installation

The installation is described in the common document /osCAN/. The TMS470 specific files are described below.

4.1 OIL-Configurator

The OIL-configurator is a general tool for different OSEK-implementations. The implementation specifc parts are the codegenerator and the OIL-implementation files for the codegenerator.

- OIL-Configurator root\OILTOOL
- OIL-Implementation files root\OILTOOL\GEN
- Codegenerator root\OILTOOL\GEN

4.1.1 OIL-Implementation files

The implementation specific files will be copied onto the local harddisk. The OIL-tool has knowledge about these files through the INI-file OILGEN.INI (the correct path is set by the installation program).

CPU	Implementation file	Standard object file	Description
TMS470R1	TMS470R1.I23	TMS470R1.S23	Source code version
TMS470PVF24x	TMS470PVF24.I23	TMS470PVF24.S23	Source code version
TMS470PVF34x	TMS470PVF34.I23	TMS470PVF34.S23	Source code version
TMS470PVF145B	TMS470PVF24.I23	TMS470PVF24.S23	Source code version
TMS470PVFx	TMS470PVF_GENERIC.I23	TMS470PVF_GENERIC.S23	Source code version
TMS470R1SE (with IEM)	TMS470R1_SE.I23	TMS470R1_SE.S23	Source code version
PL2002EX (with IEM)	TMS470_PL2002EX.I23	TMS470_PL2002EX.S23	Source code version

5 OIL-Attributes

Before an application can be compiled all static OSEK objects have to be generated. This is done by defining all objects in the OIL-configurator and by starting the codegenerator. Some of the attributes of an OSEK object are standard for all OSEK implementations; some are specific for each implementation.

5.1 OIL-configurator

The OIL-configurator is a Windows based program, which is used to configure an OSEK application. The OIL-configurator reads and writes OIL-Files (OSEK Implementation Language). The usage of the OIL-configurator is described in the online help of the OIL-configurator.

The OIL-configurator has separate property sheets for each OSEK object. Each object has several standard attributes, which are defined in the OIL specification. The standard attributes are described in the document /osCAN/. The TMS470 specific attributes are described below.

5.2 OS

The OS-object can only be defined once. The OS-object controls general aspects of the operating system.

Property	Description
SystemStackSize	System stack size in bytes
CpuFrequency	CPU Frequency in MHz. **Note**: the TMS470PVF knows various clock domains, here the RTICLK is the one of interest.
TimerClockDivider	Preload value for RTI prescaler (PRELD).
SystemTimer	RTI compare interrupt used for system timer or UserDefined.
SystemTimerIntPrio	Priority (CIM/VIM vector number) used for system timer. **Note**: This attribute is not available for TMS470R1. **Note:** On TMS470PVF24x/ TMS470PVF24x AUTO is allowed to use the default priority of the selected comparator.
ThumbMode	If set to TRUE, function calls for mixed ARM (32bit)/THUMB (16bit) functions are generated. See section 10.1
IRQStackOffset	Size of IRQ stack is extended by this value (bytes). Use this

	attribute to add stack for category 1 interrupts.
IRQMaxNesting	Maximum interrupt nesting depth. **Note**: System timer interrupt is enabling nested interrupts.Therefore IRQMaxNesting must be set at least to 2 if any other IRQ interrupt is used by the application.
EnableIntNesting	If set to TRUE, interrupt nesting is automatically enabled for all category 2 interrupts. **Note**: This attribute is not available for TMS470R1.
SupervisorStackSize	Supervisor stack size (for SWI) in bytes.
StartupStackSize	Size of stack used during startup. StartOS() and PreTaskHook() are using startup stack. If startup stack fits into biggest task stack, both stacks are sharing memory. Otherwise a separate startup stack is created.
NotUsingRES_SCHEDULER	If set to TRUE, the generation of resource RES_SCHEDULER is suppressed. This will reduce the number resources and priorities.
Compiler	The supported compiler can be chosen.
EmulatorVersion	If set to TRUE, the interrupt mapping of the emulator version is used (GIOA mapped to IEM44 and GIOB mapped to IEM45). **Note**: This attribute is available for implementation TMS470_PL2002EX only.
ORTIDebugSupport	The TMS470 implementation supports ORTI debug information (ORTI-file) if this attribute is selected.
ORTIDebugLevel	ORTI_20: Support of ORTI 2.0 (no official standard) ORTI_21_STANDARD: Support of ORTI 2.1, no overhead ORTI_21_Additional: Support of ORTI 2.1, additional features, requires some additional runtime and memory.
GHSDebugSupport	If set to TRUE, additional information for Green Hills OS aware debugging is generated. ROM size is increased when enabling GHSDebugSupport. This option is only useful for the Green Hills compiler version.
UseRegistersForOS	If set to true, the CPU registers R5 and R6 are used for global variable register allocation for some frequently used system variables. If enabled, all modules have to include *osek.h* to ensure that these registers are not used by the application. See 11.1 for more details.

	Note: This option is currently implemented in the TI compiler version only.
UDEFHandler	Name of handler for undefined instruction interrupt or AUTO for default handler.
SWIHandler	Name of handler for software interrupt or AUTO for default handler. Attention: If an SWI handler is provided by the application, there will be a jump (no call!) to the SWI handler.
PABTHandler	Name of handler for prefetch abort interrupt or AUTO for default handler.
DABTHandler	Name of handler for data abort interrupt or AUTO for default handler.
FIQHandler	Name of handler for FIQ interrupt or AUTO for default handler.
NoFIQVector	If set to TRUE, the FIQ vector is not generated into the interrupt vector table. The application must provide an own FIQ vector or handler.

Table 2: TMS470 specific attributes of OS

5.3 Event

Events in the OSEK operating system are always implemented as bits in bit-fields. The user could use bit-masks like ‘0x0001’ but to achieve portability between different OSEK implementation he should use event names, which are mapped by the codegenerator to the defined bits. TheTMS470 implementation allows 32 events per task.

The size of bit fields needed to store set event bits or bits the task is waiting for are calculated. The OSEK type *EventMaskType* is adapted accordingly (to either uint16 or uint32). The task, which receives the highest number of events determines the size of *EventMaskType*.

5.4 ISR

Property	Description
InterruptTarget	Defines whether the interrupt shall be handled as either IRQ or FIQ.
InterruptSource	The interrupt source can be chosen. All interrupt service routines have to be defined in the OIL-Configurator.
InterruptPriority	Priority (CIM/VIM vector number) of the interrupt. **Note**: This attribute is not available for TMS470R1. **Note**: On TMS470PVF2xx/ TMS470PVF3xx AUTO is allowed to use the default priority.
UseSpecialFunc-tionName	This attribute allows to change the ISR function name. (The default function name equals the ISR name.) This attribute makes it possible to handle the interrupt requests of different interrupt sources in one single ISR function. (see also /osCAN/)
FunctionName	When UseSpecialFunctionName is TRUE this attribute will be visible. A function name can be specified here which will be the ISR name.

Table 3: TMS470 specific attributes of ISR

6 System generation

The system generation process is described in the document /osCAN/ which is common to all implementations. The following section describes the TMS470 specific parts of this generation process.

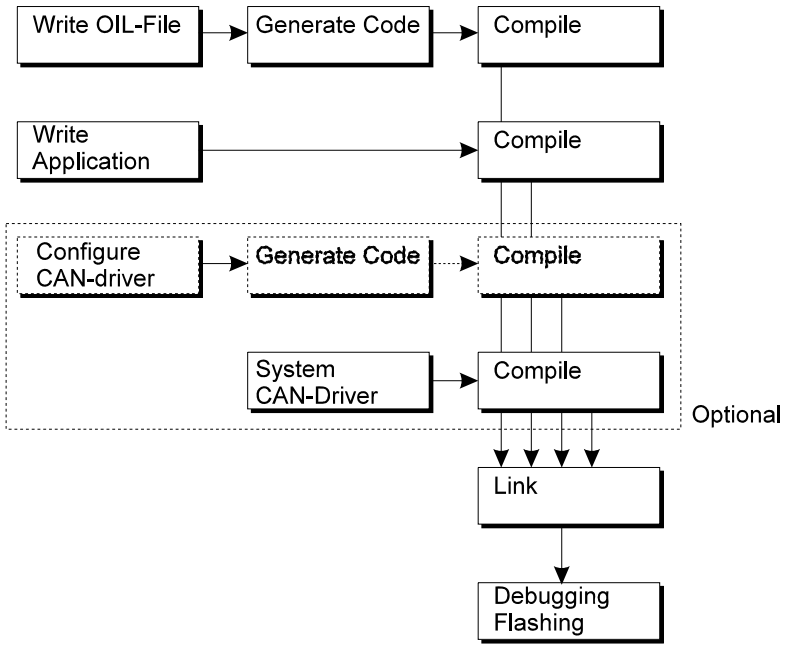

Figure 1: System generation

6.1 Codegenerator

The following files are generated by the codegenerator GENTMS470.EXE.

- tcb.c
- tcb.h
- msg.c
- msg.h
- proctask.c
- intvect.arm (GHS compiler)
- intvect.asm (TI compiler)
- tcb.inc
- *OILFileName*.ort
- *OILFileName*.htm
- *OILFileName*.lst

The files tcb.c, tcb.h, msg.c, msg.h, proctask.c, *OILFileName*.ort, *OILFileName*.lst and *OILFileName*.htm are described in the document /osCAN/.

The file tcb.inc is used to configure the assembler functions (used for TI compiler only).

The modules intvect.asm (TI version) and intvect.arm (Green Hills version) are containing the interrupt vector table and wrapper functions for interrupt routines of category 2.

6.2 Timebase

In the TMS470 implementation it is possible to use one of the 2 (4 on TMS470PVF24x) compare interrupts of the RTI timer for the system timer. The timer can be selected in the OIL-configurator with the OS attribute *SystemTimer*. The prescaler factor can also be selected.

6.2.1 User defined SystemTimer

In addition it is possible to configure *UserDefined* for the *SystemTimer*. If *UserDefined* is selected the counter API will be generated for the system timer and the system timer can then be triggered by any cyclic interrupt.

Example:

RTI compare Interrupt COMP1 should be used for *SystemTimer* but it is necessary to use COMP1 also for the application.

OIL-Configurator:

- Select UserDefined as SystemTimer
- Insert an ISR of category 2 for COMP1
- Select UseGeneratedFastAlarm
- Select your TickTime (e.g.1000)

The following functions are generated by the Codegenerator:

- TickType GetCounterValueSystemTimer(void);
- StatusType InitCounterSystemTimer(TickType ticks);
- void CounterTriggerSystemTimer(void);

Insert the following code in your application:

```
void StartupHook(void)
{
   /* my initialisation code in the StartupHook */
   ...

   /* initialize Timer */
   MyTimerInit();

   /* call generated function to reset the counter of the SystemTimer */
   InitCounterSystemTimer(0);
}
```

```
ISR(MyTimerISR)
{
   /* ISR is called every 500 microseconds by Timer */

   static uint8 n=0;
   n++;

   /* my application code in the ISR */
   ...


   /* TickTime is 1000 micro seconds, but ISR is called every 500 */
   if (n==2)
   {
      /* call generated function every 1000 micro seconds */
      CounterTriggerSystemTimer();
      n=0;
   }
}
```

Remark: The configured *TickTime* in the OIL-Configurator must be the same as the cyclic call time of the generated function *CounterTriggerSystemTimer*.

6.2.2 Range of alarms

Depended of the *TickTime* and the selected alarm management (*UseGeneratedFastAlarm*) the range of the alarms is different.

Generated fast alarm management:

Range: 0 .. TickTime (µs) * 2^{32}-1

Standard alarm management:

Range: 0 .. TickTime (µs) * 2^{31}-1

6.2.3 Selection of the TickTime

As shorter the *TickTime* as bigger is the interrupt load, which is produced by the System Timer ISR. Therefore the *TickTime* should be chosen as big as possible (greatest unique divider of all alarm times), if there are no additional requirements of the accuracy.

Attention:

If the *TickTime* is too short and there are a lot of other interrupts, which can interrupt the system timer ISR, the alarm management can be delayed longer than the *TickTime*. The OS does not detect this.

It is not allowed to disable interrupts longer than the TickTime. If interrupts are disabled longer than the *TickTime* the alarm management could be handled wrong. This error is *not* detected by the operating system. Note, that this also concerns Hooks, since interrupts are always disabled within them.

7 Stack handling

The general stack handling is described in the common document /osCAN/. The implementation of the hardware dependent stacks is described below.

7.1 Interrupt stack and system stack

osCAN TMS470 uses two special stacks for interrupt handling: IRQ stack and system stack.

Each activation of an interrupt causes a switch to stack pointer LR_{IRQ}. To support nested interrupts with subroutines, the interrupt entry code of category 2 will switch to system stack. The whole user part of an interrupt of category 2 will use the system stack.

Attention:
IRQ interrupts of category 1 will not switch to system stack. Initially there are 9 words of stack reserved for every interrupt nesting level (configured by OS attribute IRQMaxNesting). If an interrupt of category 1 will use more stack, stack size must be extended by OS attribute IRQStackOffset.

7.2 Supervisor Stack

The supervisor stack is used for the SWI. The OS initializes the SVC stack pointer during StartOS().

If SWI is used before the call of StartOS, the SVC stack pointer must be initialized by the application. The following symbols are provided by osCAN to allow initialization of the SVC stack pointer:

osSupervisorStack	Start address (lowest address) of memory array for supervisor stack. Label osSupervisorStack is globally available.
osdSupervisorSize	Size of supervisor stack in byte. osdSupervisorStackSize is defined in tcb.h.

The SVC stack pointer must be initialized to the highest address of the system stack:

Initial SP_{SVC} = osSupervisorStack + osdSupervisorSize

7.3 Startup stack

Function *StartOS()* (including *StartupHook()*) is running on startup stack. If the startup stack fits into the biggest task stack, *StartOS()* is running on the biggest task stack. A separate startup stack is only created if the size of the biggest task stack is smaller than the startup stack size configured by the user (see 5.2, attribute StartupStackSize).

7.4 Idle state

During idle state (while no task is in RUNNING state), no stack is used by this OSEK implementation.

7.5 Get system stack usage

Using the function *osGetSystemStackUsage* allows to get the maximum system stack usage in byte since StartOS.

Prototype:

```
uint16 osGetSystemStackUsage(void);
```

Argument: none

Return value: Maximum stack usage (bytes) of interrupt level stack since StartOS().

Note: It is required to set *WithStackCheck* = TRUE to use this functions.

8 Interrupt handling

8.1 IRQ interrupts

IRQ interrupts can be used for category 1 and 2 interrupts. All IRQ interrupts must be configured in the ISR section of the OIL configurator.

There is only 1 IRQ primary vector available (at address 0x18). An interrupt dispatch function is calling the IRQ handlers all IRQ interrupt sources (CIM or VIM in nonvectored mode).

8.2 FIQ

FIQ interrupts can be used for category 1 interrupts only. FIQ interrupts are not configured in the OIL-configurator. The FIQ interrupt handler must be provided by the application.

Note for TMS470PVF24x: All FIQ interrupts **must** be configured in the ISR section of the OIL configurator because any unused VIM priority (no ISR configured for it) is initialized to an unused channel in StartOS. The REQMASK bits are initialized only for the SystemTimer interrupt. The FIQ handler has to be provided by the application.

8.3 Category 1 interrupts

Interrupts of category 1 are not allowed to use API-functions, vice versa these routines can be programmed without restrictions and are independent from the kernel. The programming conventions depend on the used compiler / assembler.

Category 1 interrupts can be enabled before call of StartOS(). If interrupts of category 1 and 2 cannot be disabled separately, all interrupts must be disabled.

Note: Some OSEK API functions for interrupt handling are allowed inside category 1 interrupts. Although category 1 interrupts can be enabled before the call of StartOS the application must ensure that no API function is called before StartOS!

C-Language interrupt definition for TI compiler:

```
#pragma INTERRUPT(ISR1,IRQ)
void ISR1 (void)
{
   /* ISR code */
}
```

C-Language interrupt definition for Green Hills compiler:

```
__interrupt void ISR1(void);
#pragma intvect ISR1 0x18
__interrupt void ISR1 (void)
{
  /* ISR code */
}
```

Note for Green Hills compiler: Compile with option –no_auto_interrupt_table to avoid conflicts with the vector table generated by the OSEK implementation. Category 1 interrupt functions must be compiled for 32bit ARM mode!

Refer the compiler manual for more details about writing ISR's in C or assembly language.

8.4 Category 2 interrupts

For category 2 interrupts all interrupt entry and exit code is automatically generated.

8.5 Nested interrupts

The TMS470 doesn't support interrupt levels. By default the I bit in CPSR register is set and all interrupts are disabled after entering an IRQ. By clearing the I-Bit in the CPSR nested interrupts are possible. OIL attribute IRQMaxNesting must be set to the maximum nesting level in the application.

As each interrupt routine (category 2) could activate a higher priority task (higher in priority as the task which was interrupted by the first interrupt) the interrupt routine on the lowest level (which will terminate at last) has to call the scheduler. The housekeeping of nested interrupts is done by the operating system. The *osCAN* TMS470 supports nested interrupts, but there are some important restrictions.

Category 1 interrupts must always have higher priority than category 2 interrupts. As the TMS470 does not support interrupt levels, it is not allowed to clear the I-Bit while a category 1 IRQ interrupt is processed. Because FIQ interrupts have higher priority than IRQ interrupts, this restriction does not apply for category 1 FIQ interrupts.

In category 2 interrupts it is allowed to clear the I-Bit and nested interrupts are possible.

8.5.1 Automatic nesting

Note: Automatic interrupt nesting is available for implementations TMS470R1_SE and TMS470_PL2002EX only.

When automatic interrupt nesting is enabled, the (generated) interrupt entry code for all category 2 interrupts will clear the I-Bit in the CPSR register. Before clearing the I-Bit, all interrupts with equal and lower priority (equal and higher CIM entry number) will be disabled in the REQMASK register of the interrupt controller. The (generated) interrupt exit code will restore the previous state of the REQMASK register.

Note: Changes of the REQMASK register inside an interrupt category 2 handler will be overwritten by the interrupt exit code.

If the interrupt nesting depth has reached the maximum nesting depth as defined by attribute IRQMaxNesting, nesting will not be enabled.

8.6 Interrupt vectors

The codegenerator started by the OIL-configurator generates module intvect.arm / intvect.asm, containing all IRQ interrupt vectors. The vector numbers of all interrupt routines are taken from the OIL-definition. All IRQ interrupt routines have to be defined in the OIL-configurator, even those of category 1.

The ARM core vector table must be linked properly, see section 15.

For TMS470PVF24x and TMS470PVF34x, currently only VIM legacy mode is supported

8.6.1 SWI

The SWI is used by the OS for task switching and to provide access to registers available in privileged modes only. The SWI can be called with one 16 bit argument in ARM mode and with one 8 bit argument in THUMB mode. Argument range 0x80 to 0xFF is reserved for the OS. For all other arguments there is a jump (no call!) to an SWI handler provided by the application.

8.6.2 Unhandled exception

All interrupt vectors which are not assigned to an interrupt routine will generate a call to an ‘unhandled exception‘-handler. The handler calls, if enabled, the ErrorHook-Routine with the error-code *osdErrUEUnhandledException*.

8.7 Interrupt sources

The application is allowed to enable and disable all interrupt sources beside the source used for system timer. This OSEK implementation does not make any presumptions about the state of the Interrupt Mask Register REQMASK (e.g. there are no shadow registers in the operating system). For implementations TMS470R1_SE and TMS470_PL2002EX see also 8.5.1 for the usage of REQMASK when automatic interrupt nesting is enabled.

8.8 IEM support

Note: Support for the IEM (Interrupt Expansion Module) is currently available for implementations TMS470R1_SE and TMS470_PL2002EX only.

The IEM support will automatically initialize the IEM at startup. Each IEM entry will be connected to a CIM entry according to the configured priority.

Example: If priority 5 is configured for IEM entry 30, the IEM entry 30 will be connected to the CIM entry number 5.

Multiple IEM entries can be connected to one CIM entry. The interrupt dispatching code will be generated by the operating system. If multiple interrupts on one CIM entry occur simultaneously, the interrupt handler for the lowest IEM entry number is called first.

Note: CIM entry sharing is allowed for category 1 interrupts or category 2 interrupts. It is **not** allowed to mix category 1 interrupts and category 2 interrupts on the same CIM entry.

9 Implementation specific behaviour

9.1 API functions

9.1.1 DisableAllInterrupts

The function *DisableAllInterrupts* disables all interrupts by setting the I and F bits in CPSR. The old value of the bits is stored. **Remark**: Nested calls are **not** possible.

9.1.2 EnableAllInterrupts

The function *EnableAllInterrupts* restores the contents of the I and F bits (saved by *DisableAllInterrupts)*. **Remark**: Nested calls are **not** possible.

9.1.3 SuspendOSInterrupts

The function *SuspendOSInterrupts* disables all IRQ interrupts by setting the I bit in CPSR. The old value of the I bit is saved. **Remark**: Nested calls are possible.

9.1.4 ResumeOSInterrupts

The function *ResumeOSInterrupts* restores the content of the I bit (saved by *SuspendOSInterrupts)*. **Remark**: Nested calls are possible.

9.1.5 SuspendAllInterrupts

The function *SuspendAllInterrupts* disables all interrupts by setting the I and F bits in CPSR. The old value of the bits is stored. **Remark**: Nested calls are possible.

9.1.6 ResumeAllInterrupts

The function *ResumeAllInterrupts* restores the content of the I and F bits (saved by *SuspendAllInterrupts)*. **Remark**: Nested calls are possible.

9.1.7 GetResource

osCAN TMS470 supports the extension of the resource concept for interrupt levels. If a resource is shared between tasks and ISRs or between ISRs, the *GetResource* system call will disable IRQ interrupts.

9.1.8 ReleaseResource

ReleaseResource is the counterpart of *GetResource*. After calling *ReleaseResource* the IRQ interrupt flag is restored. Resources are assigned to interrupt service routines by means of the RESOURCE attribute in the OIL configurator.

9.2 Hook routines

The context where hook routines are called are implementation specific and are described below. All Hook routines are called with disabled interrupts.

9.2.1 ErrorHook

The *ErrorHook* is called if an error occurs inside an API-function. Dependent of the call level of the API-function, the *ErrorHook* runs either on task stack in the CPU mode of the task (called on task level) or on system stack in system mode (called on interrupt level).

9.2.2 PreTaskHook

The *PreTaskHook* is running on the task stack of the task that is in the RUNNING state. PreTaskHook is always executed in system mode.

9.2.3 PostTaskHook

The *PostTaskHook* is running on the task stack of the task that is in the RUNNING state. If the task is leaving the RUNNING state because of calling WaitEvent, TerminateTask or Chaintask, the PostTaskHook is running in the CPU mode of the task, otherwise in system mode.

9.2.4 StartupHook

The *StartupHook* is running in system mode on startup stack and is called in *StartOS*.

9.2.5 ShutdownHook

The *ShutdownHook* is called in *ShutdownOS*. Dependent of the call level of *ShutdownOS*, the *ShutdownHook* runs either on task stack in the CPU mode of the task (called on task level) or on system stack in system mode (called on interrupt level). If *ShutdownOS* is called in *StartupHook*, the *ShutdownHook* runs on system stack.

10 CPU modes

10.1 ARM/Thumb mode

osCAN TMS470 supports plain 32 bit ARM mode and a mixed 16/32 mode using Thumb code.

If Thumb code is used, the OIL attribute ThumbMode has to be set TRUE in the OS section. All OSEK source code written in C must be compiled for Thumb mode with this configuration. With ThumbMode set to TRUE, all Task functions must be compiled for Thumb mode, while functions called from the task functions might also be compiled for ARM mode.

An SWI handler provided by the application must always be compiled for ARM mode.

10.2 Task privileges

TI compiler version:	All tasks are running in system mode.
Green Hills compiler version:	All tasks are running in user mode.

11 Optimization

11.1 Registers reserved for OS

This optimization is currently available for the TI compiler version only.

By setting the OIL attribute UseRegistersForOS to TRUE, the CPU registers R5 and R6 can be reserved for the operating system. With this configuration two frequently used system variables will be located in fixed registers for fast access. In addition, it is not necessary to save and restore these registers during context switches.

Note: When UseRegistersForOS is set to TRUE, all application code must be compiled with register r5 and r6 reserved! This also applies for all libraries used. All files containing the statement *include "osek.h"* will automatically use the proper configuration. Refer to the TI documentation of the library build utility MK470 for recompilation of the runtime support libraries (rts*.lib).

Note: ORTI debugging does not work properly if UseRegistersForOS is set to TRUE because of the missing debug information for global register variables!

Refer the TI compiler documentation for more information about "Global Register Variables".

11.2 TASK pragmas

This optimization is available for the TI compiler version only.

The TI compiler provides a TASK pragma. This pragma can be used for all task functions used by the application. The name of a task function is the name inside the OSEK TASK macro with "func" appended.

Example:

```
#pragma TASK(MyTaskfunc)

TASK(MyTask)
{
   …
}
```

If TASK pragma is assigned to a task function, the compiler will not generate unnecessary code for register saving and restoring at start and end of the task function. This will save some runtime and some stack memory.

Note: If a task function with the TASK pragma assigned is left without the call of TerminateTask(), this error will not be detected!

11.3 Startup stack

If the startup stack fits into any task stack, the task stack will be used for startup. Otherwise a separate task stack will be created.

For the exceptional case that the size of the startup stack is bigger than the size of the largest task stack, increasing the size of the largest task stack to startup stack size will avoid the creation of a separate startup stack. This will save the RAM used for the separate startup stack.

12 Application startup code

12.1 Stack pointer

The application startup code must initialize all stack pointers except those for IRQ mode and SWI mode. If any of these modes is already used before call of StartOS, the related stack pointers must be initialized before.

See also chapter 7 for details on stack handling.

12.2 Startup mode

When calling StartOS(), the CPU must be in supervisor or system mode. This is the reset default and should not be changed by the application startup code.

12.3 Reset Vector

The reset vector must be set properly: a section containing a single jump to the startup code must be linked to address 0.

13 Error handling

13.1 osCAN TMS470 error numbers

The OSEK specification defines several error numbers, which are returned by the API-functions. A certain error number has different meanings with different API-functions. The user has to know the API-function to interpret correctly the error number. The implementation independent error codes are described in /osCAN/.

The following OSEK error codes are TMS470 specific.

In addition to the OSEK error numbers all *osCAN* implementations provide unique error numbers for an exact error description. All error numbers are defined as a 16 bit value. The error numbers are defined in the headerfile 'osekerr.h' and are defined according to the following syntax:

```
0xgfee
  ||+--- consecutive error number
  |+---- number of function in the function group
  +----- number of function group
```

The error numbers common to all *osCAN* implementations are described in the document /osCAN/. The implementation specific error numbers have a function group number >= 0xA000 and are described below.

Error types:

Type	Description
OSEK	OSEK error. After calling *ErrorHook*, the program is continued.
Assertion	System assertion error. After calling *ErrorHook* the operating system is shut down. Assertion checking is enabled by setting the attribute *OSInternalChecks* to *Additional* and the attribute *STATUS* to *EXTENDED* in the OIL-configurator.
Syscheck	System error: After calling *ErrorHook* the operating system is shut down. Refer to the specific error for a description how to enable or disable error checking.

13.1.1 Error numbers of group: TMS470 specific

Group (A) contains the functions:

API-function	Abbreviation	function number
osSystemStackOverflow	YO	1
interrupt	HW	2

Error numbers of the group:

Name of error	Error	Type	Reason
osdErrYOStackOverflow	0xA101	assertion	Called if system stack overflow is detected
osdErrHWDefVect	0xA201	warning	Call of default interrupt vector 0.

Table 45: Error numbers of group: TMS470 specific

14 Version and Variant coding

The version and the variant are coded into the generated binary or HEX-file. The user has the possibility to read version and variant using an emulator, or if the electronic control unit is accessible via the CCP-protocol via the CAN-bus.

The version and variant information is written into a structure, which is defined in OSEK.H.

```
typedef struct
{
   uint8 ucMagicNumber1;     /* magic number) */
   uint8 ucMagicNumber2;     /* defined as uint8 for independency of */
   uint8 ucMagicNumber3;     /* byte order                           */
   uint8 ucMagicNumber4;

   uint8 ucSysVersionMaj;    /* version of operating system, Major */
   uint8 ucSysVersionMin;    /* version of operating system, Minor */
   uint8 ucGenVersionMaj;    /* version of code generator */
   uint8 ucGenVersionMin;    /* version of code generator */
   uint8 ucSysVariant;       /* general variant coding    */
   uint16 ucSpecVariant
} osVersionVariantCodingType;
```

The common part of the structure is described in the document /osCAN/. The member ucSpecVariant contains the following TMS470 specific information.

Bits	Meaning	Possible values
0	Thumb mode	0 = disabled; 1 = enabled
1	ORTI support	0 = disabled; 1 = enabled
2	Register optimization (UseRegistersForOS)	0 = disabled; 1 = enabled
4..7	Derivative	0 = TMS470R1 1 = TMS470R1_SE 2 = TMS470_PL2002EX 3 = TMS470PVF24x or TMS470PVF34x
8..15	ORTI version	20 = ORTI 2.0 21 = ORTI 2.1

Table 67: Bit-definitions of the variant coding

15 Section linking

All osCAN code and data is located in separate sections:

Section name	*Description*
.osarmvect	ARM core vector table, must be linked to 0x04 (except in case of using a flash boot loader; refer to its manual)
.osvtable	IRQ vector table used by interrupt dispatcher
.oscode	OSEK code area
.osconst	OSEK constant area
.osdata	OSEK variables to be placed into RAM
.osstack	OSEK stack areas

Note: Sections .oscode, .osconst, .osdata and .osstack are available in the Green Hills compiler version only.

Note: The reset vector (a jump to the startup code) must be linked to address 0

Note: When using the the TI tool chain, ISR code must be linked nearby section osarmvect. Otherwise, the assembler will fail with the error “intvect.asm: branchtarget not reachable”

Kernel-aware debugging

15.1 ORTI

The osCAN TMS470 operating system provides support for kernel-aware debugging via the ORTI interface. To use this feature *ORTIDebugSupport* must be enabled in the OIL Configurator and the level has to be set (*ORTIDebugLevel*). For detailed information please refer to your debugger's manual.

15.2 Green Hills debugger support

The Green Hills toolchain MULTI2000 provides special debugging support for osCAN. Set OIL attribute GHSDebugSupport to TRUE to enable Green Hills debugging support. Refer manual "Using MULTI 2000 with osCAN" for details.

Note: This feature is only useful in the Green Hills compiler version.

16 List of files (source code version)

16.1 Modules

The TMS470 implementation consist of the following files:

Subdirectory SRC:

osek.c	System initialization, scheduler, interrupt control
osekalrm.c	Alarm related functions
osekcom.c	OSEK COM, intertask communication
osekerr.c	Error handling
osekevnt.c	Event handling
osekrsrc.c	Resource handling
osektask.c	Task management
osektime.c	Timer interrupt routine and alarm management
osektrac.c	Internal Trace Module
osekasm.arm	Parts of OS written in assembler. This file is used by the Green Hills compiler version only.
Osekasm.asm	Parts of OS written in assembler. This file is used by the TI compiler version only.
Osekint.arm	Interrupt handling parts of the OS written in assembler. This file is used by the Green Hills compiler version only.
Osekint.asm	Interrupt handling parts of the OS written in assembler. This file is used by the TI compiler version only.

Subdirectory INCLUDE:

osek.h	main header, has to be included in every application module
osekasrt.h	included by osek.h, macro-definitions for assertion handling
osekerr.h	included by osek.h, definitions of all error-numbers
osekcom.h	header for OSEK COM, has to be included in the application modules

osekext.h	OSEK header for internal functions
vrm.h	included by all headers and system modules, Vector release management
emptymac.h	empty API hook macros
testmac1.h	user API hook macros (testmac1.h contains Macros for ORTI debug support)
testmac2.h	user API hook macros (testmac2.h contains Macros for internal Trace debug support)
testmac3.h	user API hook macros
testmac4.h	user API hook macros

17 Resource-Requirements (TI compiler version)

Note: All tasks running in system mode!

17.1 RAM- Requirements (fixed kernel size)

17.1.1 RAM Requirement, System:

Standard Status	Extended Status
26 Bytes	27 Bytes

17.1.2 RAM Requirement, Tasks:

N = Number of Tasks

Task Type	CC	Standard Status	Extended Status
Basic	BCC1 / ECC1	N*5 Bytes	N*5 Bytes
Basic	BCC2 / ECC2	N*6 Bytes	N*6 Bytes
Extended	ECC1	N*7 Bytes	N*7 Bytes
Extended	ECC2	N*8 Bytes	N*8 Bytes

17.2 ROM-Requirements (fixed kernel size)

Variant	ARM mode (32bit)		Thumb mode (16bit)	
	Standard Status	Extended Status	Standard Status	Extended Status
BCC1	3928 Bytes	4848 Bytes	3244 Bytes	3948 Bytes
BCC2	4380 Bytes	5332 Bytes	3560 Bytes	4280 Bytes
ECC1	4396 Bytes	5768 Bytes	3540 Bytes	4608 Bytes
ECC2	4988 Bytes	6384 Bytes	3952 Bytes	5028 Bytes

The code sizes include all API functions of the OSEK standard, but do not contain the code of OSEK-COM intertask-communication.

Note: By switching off the API functions not used by the application, a significant amount of ROM might be saved.

17.3 Time-Requirements

MCU: TMS470R1, 30 MHz
Compiler optimization options: -o2
All times in µs

Abbreviations:
BT Basic Task
ET Extended Task
MA multiple activation

17.3.1 OSEK OS

17.3.1.1 BCC1

	ARM mode (32bit)		Thumb mode (16bit)	
	Standard Status	**Extended Status**	**Standard Status**	**Standard Status**
ActivateTask (BT without taskswitch)	3.4	3.6	3.6	4.0
ActivateTask (BT with taskswitch)	8.2	8.5	8.4	9.0
TerminateTask	7.8	8.6	8.2	9.2
ChainTask (BT)	8.4	9.2	8.9	9.8
Schedule (without taskswitch)	2.8	3.8	3.4	4.6
Schedule (with taskswitch)	7.1	8.1	7.5	8.8
GetTaskID	0.4	0.4	0.5	0.5
GetTaskState	0.7	1.0	0.8	1.0
GetResource	3.6	5.1	4.2	5.9
ReleaseResource (without taskswitch)	4.8	5.8	5.5	6.6
ReleaseResource (with taskswitch)	9.0	10.0	9.5	10.7
GetAlarmBase	0.6	0.8	0.6	0.9
GetAlarm	2.8	2.9	3.0	3.4
SetRelAlarm	2.6	2.8	3.0	3.1
SetAbsAlarm	2.8	3.1	3.3	3.7
CancelAlarm	2.0	2.3	2.3	2.6
GetActiveApplicationMode**	0.0	0.0	0.0	0.0
StartOS*	20.6	21.2	24.8	25.3
EnableAllInterrupts	0.6	0.6	0.7	0.7
DisableAllInterrupts	0.5	0.5	0.6	0.6
ResumeOSInterrupts	0.9	0.9	1.0	1.0
SuspendOSInterrupts	0.8	0.8	1.0	1.0
ResumeAllInterrupts	0.9	0.9	1.0	1.0
SuspendAllInterrupts	0.8	0.8	1.0	1.0

Time depends on number of tasks
* only one application mode configured

17.3.1.2 BCC2

	ARM mode (32bit)		Thumb mode (16bit)	
	Standard Status	Extended Status	Standard Status	Standard Status
ActivateTask (BT without taskswitch)	4.5	4.8	5.1	5.2
ActivateTask (BT with taskswitch)	9.4	9.8	9.9	10.1
ActivateTask (BT without taskswitch, MA)	4.5	4.8	4.9	5.1
TerminateTask	9.5	10.5	10.2	11.2
TerminateTask (MA)	8.6	9.6	9.3	10.4
ChainTask (BT)	11.5	12.5	12.7	13.9
ChainTask (BT, MA)	11.1	12.1	12.0	13.2
Schedule (without taskswitch)	3.1	4.1	3.7	4.9
Schedule (with taskswitch)	7.4	8.4	7.8	9.1
GetTaskID	0.4	0.4	0.5	0.5
GetTaskState	0.7	1.0	0.8	1.0
GetResource	3.7	5.1	4.3	6.0
ReleaseResource (without taskswitch)	5.1	6.0	5.8	6.9
ReleaseResource (with taskswitch)	9.3	10.3	9.9	11.1
GetAlarmBase	0.6	0.8	0.6	0.9
GetAlarm	2.8	2.9	3.0	3.4
SetRelAlarm	2.5	2.9	3.0	3.1
SetAbsAlarm	2.8	3.1	3.3	3.7
CancelAlarm	2.0	2.3	2.3	2.6
GetActiveApplicationMode**	0.0	0.0	0.0	0.0
StartOS*	26.8	27.3	32.7	33.2
EnableAllInterrupts	0.6	0.6	0.7	0.7
DisableAllInterrupts	0.5	0.5	0.6	0.6
ResumeOSInterrupts	0.9	0.9	1.0	1.0
SuspendOSInterrupts	0.8	0.8	1.0	1.0
ResumeAllInterrupts	0.9	0.9	1.0	1.0
SuspendAllInterrupts	0.8	0.8	1.0	1.0

Time depends on number of tasks

* only one application mode configured

17.3.1.3 ECC1

	ARM mode (32bit)		Thumb mode (16bit)	
	Standard Status	Extended Status	Standard Status	Standard Status
ActivateTask (BT without taskswitch)	3.5	3.8	3.8	4.3
ActivateTask (BT with taskswitch)	8.3	8.6	8.6	9.2
ActivateTask (ET without taskswitch)	3.6	3.8	4.1	4.5
ActivateTask (ET with taskswitch)	8.4	8.7	8.9	9.5
TerminateTask	7.8	8.6	8.1	9.1
ChainTask (BT)	8.6	9.5	9.3	10.4
ChainTask (ET)	8.4	9.3	9.0	10.1
Schedule (without taskswitch)	2.8	3.8	3.5	4.7
Schedule (with taskswitch)	7.1	8.1	7.6	8.9
GetTaskID	0.4	0.4	0.5	0.5
GetTaskState	0.7	1.0	0.8	1.0
GetResource	3.6	5.0	4.2	5.9
GetResource (ResourceScheduler)	3.6	5.0	4.2	5.9
ReleaseResource (without taskswitch)	4.8	5.7	5.4	6.5
ReleaseResource (with taskswitch)	9.0	10.0	9.4	10.6
SetEvent (without taskswitch)	2.6	3.2	2.9	3.6
SetEvent (with taskswitch)	9.6	10.1	9.9	10.7
ClearEvent	2.0	2.5	2.4	3.3
GetEvent	0.6	1.4	0.8	1.9
WaitEvent (without taskswitch)	1.8	2.8	2.0	3.5
WaitEvent (with taskswitch)	9.2	10.2	9.6	11.1
GetAlarmBase	0.6	0.8	0.6	0.9
GetAlarm	2.8	2.9	3.0	3.4
SetRelAlarm	2.5	2.9	3.0	3.1
SetAbsAlarm	2.8	3.1	3.3	3.7
CancelAlarm	2.0	2.3	2.3	2.6
GetActiveApplicationMode**	0.0	0.0	0.0	0.0
StartOS*	22.7	23.3	27.7	28.2
EnableAllInterrupts	0.6	0.6	0.7	0.7
DisableAllInterrupts	0.5	0.5	0.6	0.6
ResumeOSInterrupts	0.9	0.9	1.0	1.0
SuspendOSInterrupts	0.8	0.8	1.0	1.0
ResumeAllInterrupts	0.9	0.9	1.0	1.0
SuspendAllInterrupts	0.9	0.9	1.0	1.0

Time depends on number of tasks

* only one application mode configured

17.3.1.4 ECC2

	ARM mode (32bit)		Thumb mode (16bit)	
	Standard Status	Extended Status	Standard Status	Standard Status
ActivateTask (BT without taskswitch)	4.8	5.0	5.3	5.5
ActivateTask (BT with taskswitch)	9.5	9.9	10.1	10.3
ActivateTask (ET without taskswitch)	4.9	5.2	5.6	5.7
ActivateTask (ET with taskswitch)	9.6	10.0	10.4	10.7
ActivateTask (BT without taskswitch, MA)	4.6	4.9	5.2	5.3
TerminateTask	9.6	10.6	10.2	11.2
TerminateTask (MA)	8.6	9.6	9.4	10.5
ChainTask (BT)	11.6	12.6	13.3	14.3
ChainTask (ET)	11.4	12.4	13.0	14.0
ChainTask (BT, MA)	11.2	12.2	12.5	13.6
Schedule (without taskswitch)	3.1	4.1	3.8	5.0
Schedule (with taskswitch)	7.4	8.4	7.9	9.2
GetTaskID	0.3	0.4	0.5	0.5
GetTaskState	0.7	1.0	0.8	1.0
GetResource	3.7	5.1	4.3	6.0
ReleaseResource (without taskswitch)	5.1	6.0	5.7	6.8
ReleaseResource (with taskswitch)	9.3	10.3	9.8	11.0
SetEvent (without taskswitch)	2.6	3.2	2.9	3.6
SetEvent (with taskswitch)	10.6	11.2	11.2	12.0
ClearEvent	2.0	2.5	2.4	3.3
GetEvent	0.6	1.4	0.8	1.9
WaitEvent (without taskswitch)	1.8	2.9	2.0	3.5
WaitEvent (with taskswitch)	10.6	11.8	11.1	12.7
GetAlarmBase	0.6	0.8	0.6	0.9
GetAlarm	2.8	2.9	3.0	3.4
SetRelAlarm	2.6	2.8	3.0	3.1
SetAbsAlarm	2.8	3.1	3.3	3.7
CancelAlarm	2.0	2.3	2.3	2.6
GetActiveApplicationMode**	0.0	0.0	0.0	0.0
StartOS*	29.8	30.3	36.6	37.1
EnableAllInterrupts	0.6	0.6	0.7	0.7
DisableAllInterrupts	0.5	0.5	0.6	0.6
ResumeOSInterrupts	0.9	0.9	1.0	1.0
SuspendOSInterrupts	0.8	0.8	1.0	1.0
ResumeAllInterrupts	0.9	0.9	1.0	1.0
SuspendAllInterrupts	0.8	0.8	1.0	1.0

Time depends on number of tasks
* only one application mode configured

17.3.1.5 Small sample application

The time measurements for the tables above were made with typical OSEK configurations, containing 7-13 tasks, resources, alarms, preemptive and non preemptive tasks. For smaller applications, a better performance can be achieved.

The following table shows some API execution times for an example application with 4 extended tasks, no resources, full preemptive tasks only:

	ECC1 ARM mode (32bit) Standard Status
ActivateTask (BT without taskswitch)	3.5
ActivateTask (BT with taskswitch)	7.7
TerminateTask	6.0
GetTaskID	0.4
GetTaskState	0.7
SetEvent (without taskswitch)	2.6
SetEvent (with taskswitch)	9.2
ClearEvent	1.9
GetEvent	0.6
WaitEvent (without taskswitch)	1.8
WaitEvent (with taskswitch)	7.4
GetAlarmBase	0.6
GetAlarm	2.8
SetRelAlarm	2.6
SetAbsAlarm	2.8
CancelAlarm	2.0
EnableAllInterrupts	0.6
DisableAllInterrupts	0.5
ResumeOSInterrupts	0.9
SuspendOSInterrupts	0.8
ResumeAllInterrupts	0.9
SuspendAllInterrupts	0.9

17.3.2 OSEK COM

All measurements have been made with ACTION=NONE (no notification) in the With-Copy configuration (subattribute WITHOUTCOPY of ACCESSOR is set to FALSE).

17.3.2.1 CCCA

	ARM mode (32bit)		Thumb mode (16bit)	
	Standard Status	Extended Status	Standard Status	Standard Status
StartCOM	2.6	2.6	3.0	3.0
StopCOM	0.4	0.4	0.5	0.5
SendMessage (of 1byte unqueued message)	4.5	4.7	4.8	5.1
SendMessage (of 8byte unqueued message)	5.9	6.1	6.6	6.9
ReceiveMessage (of 1byte unqueued message)	4.6	4.9	5.2	5.6
ReceiveMessage (of 8byte unqueued message)	6.4	6.7	7.3	7.6

17.3.2.2 CCCB

	ARM mode (32bit)		Thumb mode (16bit)	
	Standard Status	Extended Status	Standard Status	Standard Status
StartCOM	5.2	5.2	5.7	5.7
StopCOM	2.4	2.4	2.7	2.6
GetMessageResource	2.8	3.1	3.2	3.5
ReleaseMessageResource	2.6	2.8	3.0	3.4
GetMessageStatus (queued message)	2.1	2.4	2.6	3.0
GetMessageStatus (unqueued message)	1.1	1.4	1.4	1.7
SendMessage (of 1byte queued message)	6.9	7.0	8.4	8.7
SendMessage (of 8byte queued message)	8.4	8.6	10.3	10.6
ReceiveMessage (of 1byte queued message)	6.6	6.8	7.7	8.0
ReceiveMessage (of 8byte queued message)	7.6	7.7	8.8	9.2

18 Resource-Requirements (Green Hills compiler version)

Note: All tasks running in user mode! Measurements for implementation TMS470R1.

18.1 RAM- Requirements (fixed kernel size)

18.1.1 RAM Requirement, System:

Standard Status	Extended Status
26 Bytes	27 Bytes

18.1.2 RAM Requirement, Tasks:

N = Number of Tasks

Task Type	CC	Standard Status	Extended Status
Basic	BCC1 / ECC1	N*5 Bytes	N*5 Bytes
Basic	BCC2 / ECC2	N*6 Bytes	N*6 Bytes
Extended	ECC1	N*7 Bytes	N*7 Bytes
Extended	ECC2	N*8 Bytes	N*8 Bytes

18.2 ROM-Requirements (fixed kernel size)

Variant	ARM mode (32bit)		Thumb mode (16bit)	
	Standard Status	Extended Status	Standard Status	Extended Status
BCC1	5328 Bytes	9752 Bytes	3796 Bytes	4638 Bytes
BCC2	5932 Bytes	10356 Bytes	4212 Bytes	5066 Bytes
ECC1	6176 Bytes	12764 Bytes	4180 Bytes	5432 Bytes
ECC2	6796 Bytes	13380 Bytes	4636 Bytes	5896 Bytes

The code sizes include all API functions of the OSEK standard, but do not contain the code of OSEK-COM intertask-communication.

Note: By switching off the API functions not used by the application, a significant amount of ROM might be saved.

18.3 Time-Requirements

MCU: TMS470R1, 30 MHz
Compile options: optimized for speed
All times in µs

Abbreviations:
BT Basic Task
ET Extended Task
MA multiple activation

18.3.1 OSEK OS

18.3.1.1 BCC1

	ARM mode (32bit)		Thumb mode (16bit)	
	Standard Status	**Extended Status**	**Standard Status**	**Extended Status**
ActivateTask (BT without taskswitch)	6.0	6.1	6.3	6.5
ActivateTask (BT with taskswitch)	11.7	11.9	12.2	12.4
TerminateTask	12.5	13.6	13.5	14.5
ChainTask (BT)	12.9	14.1	14.1	15.1
Schedule (without taskswitch)	6.4	7.6	7.5	8.6
Schedule (with taskswitch)	11.4	12.6	12.4	13.7
GetTaskID	0.7	0.7	1.0	1.0
GetTaskState	0.8	1.2	1.2	1.4
GetResource	7.0	8.8	7.6	9.3
ReleaseResource (without taskswitch)	8.2	9.3	9.3	10.2
ReleaseResource (with taskswitch)	13.2	14.3	14.3	15.4
GetAlarmBase	0.7	1.0	1.0	1.3
GetAlarm	5.3	5.4	6.2	6.3
SetRelAlarm	5.2	5.3	5.9	6.0
SetAbsAlarm	5.5	5.6	6.2	6.3
CancelAlarm	4.6	4.7	5.4	5.5
GetActiveApplicationMode**	0.0	0.0	0.0	0.0
StartOS*	19.5	20.0	20.9	21.4
EnableAllInterrupts	1.8	1.8	2.1	2.1
DisableAllInterrupts	1.6	1.6	2.0	2.0
ResumeOSInterrupts	2.1	2.1	2.5	2.5
SuspendOSInterrupts	2.0	2.0	2.3	2.3
ResumeAllInterrupts	2.1	2.1	2.5	2.5
SuspendAllInterrupts	2.0	2.0	2.3	2.3

*Time depends on number of tasks
** only one application mode configured

18.3.1.2 BCC2

	ARM mode (32bit)		Thumb mode (16bit)	
	Standard Status	**Extended Status**	**Standard Status**	**Extended Status**
ActivateTask (BT without taskswitch)	7.2	7.3	7.7	7.9
ActivateTask (BT with taskswitch)	12.8	13.1	13.5	13.8
ActivateTask (BT without taskswitch, MA)	7.1	7.2	7.6	7.8
TerminateTask	14.4	15.3	15.7	16.7
TerminateTask (MA)	13.2	14.2	14.6	15.7
ChainTask (BT)	15.8	16.9	17.5	18.6
ChainTask (BT, MA)	15.4	16.8	16.9	18.2
Schedule (without taskswitch)	6.7	7.9	7.8	8.9
Schedule (with taskswitch)	11.7	12.9	12.8	14.1
GetTaskID	0.7	0.7	1.0	1.0
GetTaskState	0.8	1.2	1.2	1.4
GetResource	7.2	9.0	7.8	9.5
ReleaseResource (without taskswitch)	8.5	9.6	9.6	10.6
ReleaseResource (with taskswitch)	13.5	14.6	14.7	15.7
GetAlarmBase	0.7	1.0	1.0	1.3
GetAlarm	5.3	5.4	6.2	6.3
SetRelAlarm	5.2	5.3	5.9	6.0
SetAbsAlarm	5.5	5.6	6.2	6.3
CancelAlarm	4.6	4.7	5.4	5.5
GetActiveApplicationMode**	0.0	0.0	0.0	0.0
StartOS*	23.3	23.8	25.0	25.5
EnableAllInterrupts	1.8	1.8	2.1	2.1
DisableAllInterrupts	1.6	1.6	2.0	2.0
ResumeOSInterrupts	2.1	2.1	2.5	2.5
SuspendOSInterrupts	2.0	2.0	2.3	2.3
ResumeAllInterrupts	2.1	2.1	2.5	2.5
SuspendAllInterrupts	2.0	2.0	2.3	2.3

- Time depends on number of tasks

** only one application mode configured

18.3.1.3 ECC1

	ARM mode (32bit)		Thumb mode (16bit)	
	Standard Status	Extended Status	Standard Status	Extended Status
ActivateTask (BT without taskswitch)	6.1	6.3	6.5	6.6
ActivateTask (BT with taskswitch)	11.8	12.1	12.3	12.6
ActivateTask (ET without taskswitch)	6.4	6.5	7.0	7.2
ActivateTask (ET with taskswitch)	11.9	12.2	12.5	12.7
TerminateTask	12.5	13.6	13.5	14.6
ChainTask (BT)	13.1	14.3	14.1	15.2
ChainTask (ET)	13.0	14.2	14.2	15.3
Schedule (without taskswitch)	6.4	7.6	7.3	8.4
Schedule (with taskswitch)	11.4	12.6	12.3	13.6
GetTaskID	0.7	0.7	1.0	1.0
GetTaskState	0.8	1.2	1.2	1.4
GetResource	7.0	8.8	7.6	9.3
ReleaseResource (without taskswitch)	8.2	9.3	9.4	10.3
ReleaseResource (with taskswitch)	13.2	14.3	14.4	15.4
SetEvent (without taskswitch)	5.0	5.6	5.7	6.2
SetEvent (with taskswitch)	13.4	14.0	13.9	14.3
ClearEvent	4.9	5.5	5.6	6.2
GetEvent	0.7	2.0	1.1	1.8
WaitEvent (without taskswitch)	4.6	6.2	5.2	6.4
WaitEvent (with taskswitch)	13.9	15.4	14.8	16.0
GetAlarmBase	0.7	1.0	1.0	1.3
GetAlarm	5.3	5.4	6.2	6.3
SetRelAlarm	5.2	5.3	5.9	6.0
SetAbsAlarm	5.5	5.6	6.2	6.3
CancelAlarm	4.6	4.7	5.4	5.5
GetActiveApplicationMode**	0.0	0.0	0.0	0.0
StartOS*	20.6	21.1	22.2	22.6
EnableAllInterrupts	1.8	1.8	2.1	2.1
DisableAllInterrupts	1.6	1.6	2.0	2.0
ResumeOSInterrupts	2.1	2.1	2.5	2.5
SuspendOSInterrupts	2.0	2.0	2.3	2.3
ResumeAllInterrupts	2.1	2.1	2.5	2.5
SuspendAllInterrupts	2.0	2.0	2.3	2.3

- Time depends on number of tasks

** only one application mode configured

18.3.1.4 ECC2

	ARM mode (32bit)		Thumb mode (16bit)	
	Standard Status	Extended Status	Standard Status	Extended Status
ActivateTask (BT without taskswitch)	7.3	7.5	8.3	8.0
ActivateTask (BT with taskswitch)	13.0	13.2	14.1	13.9
ActivateTask (ET without taskswitch)	7.6	7.7	8.8	8.6
ActivateTask (ET with taskswitch)	13.1	13.3	14.2	14.1
ActivateTask (BT without taskswitch, MA)	7.2	7.4	8.2	7.9
TerminateTask	14.4	15.3	15.8	16.7
TerminateTask (MA)	13.2	14.2	14.6	15.7
ChainTask (BT)	16.0	17.0	17.5	18.7
ChainTask (ET)	15.9	16.9	17.6	18.8
ChainTask (BT, MA)	15.5	16.9	17.0	18.5
Schedule (without taskswitch)	6.7	7.9	7.7	8.8
Schedule (with taskswitch)	11.7	12.9	12.6	13.9
GetTaskID	0.7	0.7	1.0	1.0
GetTaskState	0.8	1.2	1.2	1.4
GetResource	7.2	9.0	7.8	9.5
ReleaseResource (without taskswitch)	8.5	9.6	9.7	10.6
ReleaseResource (with taskswitch)	13.5	14.6	14.7	15.8
SetEvent (without taskswitch)	5.1	5.7	5.9	6.3
SetEvent (with taskswitch)	14.2	14.8	14.8	15.2
ClearEvent	4.9	5.5	5.6	6.2
GetEvent	0.7	2.0	1.1	1.8
WaitEvent (without taskswitch)	4.6	6.2	5.3	6.4
WaitEvent (with taskswitch)	15.4	16.8	16.4	17.6
GetAlarmBase	0.7	1.0	1.0	1.3
GetAlarm	5.3	5.4	6.2	6.3
SetRelAlarm	5.2	5.3	5.9	6.0
SetAbsAlarm	5.5	5.6	6.2	6.3
CancelAlarm	4.6	4.7	5.4	5.5
GetActiveApplicationMode**	0.0	0.0	0.0	0.0
StartOS*	27.3	27.9	30.5	31.0
EnableAllInterrupts	1.8	1.8	2.1	2.1
DisableAllInterrupts	1.6	1.6	2.0	2.0
ResumeOSInterrupts	2.1	2.1	2.5	2.5
SuspendOSInterrupts	2.0	2.0	2.3	2.3
ResumeAllInterrupts	2.1	2.1	2.5	2.5
SuspendAllInterrupts	2.0	2.0	2.3	2.3

- Time depends on number of tasks

** only one application mode configured

18.3.2 OSEK COM

All measurements have been made with ACTION=NONE (no notification) in the With-Copy configuration (subattribute WITHOUTCOPY of ACCESSOR is set to FALSE).

18.3.2.1 CCCA

	ARM mode (32bit)		Thumb mode (16bit)	
	Standard Status	**Extended Status**	**Standard Status**	**Extended Status**
StartCOM	5.1	5.1	5.7	5.7
StopCOM	0.3	0.3	0.4	0.4
SendMessage (of 1byte unqueued msg)	6.8	7.0	7.5	7.9
SendMessage (of 8byte unqueued msg)	7.1	7.4	7.8	8.2
ReceiveMessage (of 1byte unqueued msg.)	8.2	8.5	8.9	9.3
ReceiveMessage (of 8byte unqueued msg.)	8.6	8.8	9.2	9.6

18.3.2.2 CCCB

	ARM mode (32bit)		Thumb mode (16bit)	
	Standard Status	**Extended Status**	**Standard Status**	**Extended Status**
StartCOM	6.4	6.4	7.3	7.3
StopCOM	1.4	1.4	1.8	1.8
GetMessageResource	5.2	5.4	5.8	6.2
ReleaseMessageResource	4.9	5.2	5.6	5.9
GetMessageStatus (queued msg.)	2.5	2.7	2.7	3.1
GetMessageStatus (unqueued msg.)	1.2	1.5	1.4	1.7
SendMessage (of 1byte queued msg.)	9.5	9.7	9.7	10.0
SendMessage (of 8byte queued msg.)	9.8	10.1	10.0	10.3
ReceiveMessage (of 1byte queued msg.)	9.5	9.8	10.2	10.6
ReceiveMessage (of 8byte queued msg.)	9.8	10.1	10.5	10.9

19 History

Version	Date	Author	Description
2.95	2004-01-29	Se	Creation
3.00	2004-01-30	Se	First released version (Green Hills compiler version)
3.01	2004-03-24	Se	First release for TI compiler version.
3.02	2004-07-26	Se	New implementation TMS470R1_SE with support for IEM and automatic interrupt nesting.
3.03	2004-10-05	Se	New implementation TMS470_PL2002EX.
3.04	2006-02-08	Lam	New implementation TMS470PVF24x (beta).
3.05	2006-06-29	Lam	Comment on TMS470PVF clock domains.
3.06	2007-05-21	Vr	Described attribute InterruptTarget , Osek support email address, remark about linking the reset vector, remark about VIM legacy mode, new implementation file for PVF344, remark about limiting Hook running times, more about startup
3.07	2008-01-30	Se	New GHS compiler version New ISR attribute UseSpecialFunctionName
3.08	2008-04-17	Se	New TI compiler version New generic implementation