# embOS

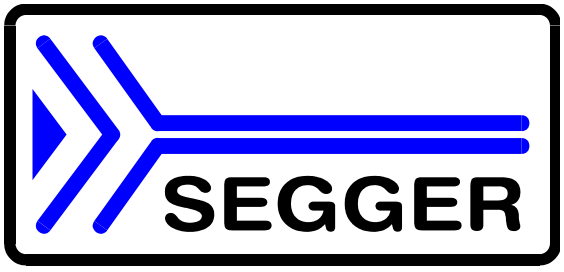
Real Time Operating System

## CPU & Compiler specifics for SO8 core

## using Freescale Codewarrior workbench®

Software version 3.80g
Document: UM01012
Revision: 1

Date: June 30, 2009




**A product of SEGGER Microcontroller GmbH & Co. KG**

**www.segger.com**

## Disclaimer

Specifications written in this document are believed to be accurate, but are not guaranteed to be entirely free of error. The information in this manual is subject to change for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, SEGGER MICROCONTROLLER GmbH & Co. KG (the manufacturer) assumes no responsibility for any errors or omissions. The manufacturer makes and you receive no warranties or conditions, express, implied, statutory or in any communication with you. The manufacturer specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

## Copyright notice



## Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

## Contact address

SEGGER Microcontroller Systeme GmbH & Co. KG

In den Weiden 11
D-40721 Hilden

Germany

Tel. +49 2103-2878-0
Fax.+49 2103-2878-28
Email: support@segger.com
Internet: *http://www.segger.com*

## Software and manual versions

This manual describes the current software version. If any error occurs, inform us and we will try to assist you as soon as possible.
Contact us for further information on topics or routines not yet specified.

Print date: June 30, 2009

| Software | Manual | Date | By | Description |
|---|---|---|---|---|
| 3.80g | 1 | 090625 | TS | First version |

# About this document

## Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler)
- The C programming language
- The target processor
- DOS command line.

If you feel that your knowledge of C is not sufficient, we recommend The C Programming Language by Kernighan and Richie (ISBN 0-13-1103628), which describes the standard in C-programming and, in newer editions, also covers the ANSI C standard.

## How to use this manual

The intention of this manual is to give you a CPU- and compiler-independent introduction to embOS and to be a reference for all embOS API functions.

For a quick and easy startup with embOS, refer to Chapter 2 in the *CPU & Compiler Specifics manual* of embOS documentation, which includes a step-by-step introduction to using embOS.

## Typographic conventions for syntax

This manual uses the following typographic conventions:

| Style | Used for |
|---|---|
| Body | Body text. |
| `Keyword` | Text that you enter at the command-prompt or that appears on the display (that is system functions, file- or pathnames). |
| `Parameter` | Parameters in API functions. |
| `Sample` | Sample code in program examples. |
| *Reference* | Reference to chapters, tables and figures or other documents. |
| **GUIElement** | Buttons, dialog boxes, menu names, menu commands. |
| **Emphasis** | Very important sections |

**Table 1.1: Typographic conventions**

**SEGGER Microcontroller GmbH & Co. KG** develops and distributes software development tools and ANSI C software components (middleware) for embedded systems in several industries such as telecom, medical technology, consumer electronics, automotive industry and industrial automation.

SEGGER's intention is to cut software development-time for embedded applications by offering compact flexible and easy to use middleware, allowing developers to concentrate on their application.

Our most popular products are emWin, a universal graphic software package for embedded applications, and embOS, a small yet efficient real-time kernel. emWin, written entirely in ANSI C, can easily be used on any CPU and most any display. It is complemented by the available PC tools: Bitmap Converter, Font Converter, Simulator and Viewer. embOS supports most 8/16/32-bit CPUs. Its small memory footprint makes it suitable for single-chip applications.

Apart from its main focus on software tools, SEGGER develops and produces programming tools for flash microcontrollers, as well as J-Link, a JTAG emulator to assist in development, debugging and production, which has rapidly become the industry standard for debug access to ARM cores.

**Corporate Office:**
*http://www.segger.com*

**United States Office:**
*http://www.segger-us.com*

## EMBEDDED SOFTWARE (Middleware)

### emWin
**Graphics software and GUI**

emWin is designed to provide an efficient, processor- and display controller-independent graphical user interface (GUI) for any application that operates with a graphical display. Starterkits, eval- and trial-versions are available.

### embOS
**Real Time Operating System**

embOS is an RTOS designed to offer the benefits of a complete multitasking system for hard real time applications with minimal resources. The profiling PC tool embOSView is included.

### emFile
**File system**


emFile is an embedded file system with FAT12, FAT16 and FAT32 support. emFile has been optimized for minimum memory consumption in RAM and ROM while maintaining high speed. Various Device drivers, e.g. for NAND and NOR flashes, SD/MMC and CompactFlash cards, are available.

### USB-Stack
**USB device stack**

A USB stack designed to work on any embedded system with a USB client controller. Bulk communication and most standard device classes are supported.

## SEGGER TOOLS

### Flasher
**Flash programmer**

Flash Programming tool primarily for microcontrollers.

### J-Link
**JTAG emulator for ARM cores**

USB driven JTAG interface for ARM cores.

### J-Trace
**JTAG emulator with trace**

USB driven JTAG interface for ARM cores with Trace memory. supporting the ARM ETM (Embedded Trace Macrocell).

### J-Link / J-Trace Related Software

Add-on software to be used with SEGGER's industry standard JTAG emulator, this includes flash programming software and flash breakpoints.

# Table of Contents



# Chapter 1

# Using embOS with Freescale Codewarrior

This chapter describes how to start with and use embOS for SO8 and the Codewarrior compiler. You should follow these steps to become familiar with embOS for SO8 together with Freescale Codewarrior®.

# 1.1 Installation

embOS is shipped on CD-ROM or as a zip-file in electronic form.

To install it, proceed as follows:

If you received a CD, copy the entire contents to your hard-drive into any folder of your choice. When copying, keep all files in their respective sub directories. Make sure the files are not read only after copying. If you received a zip-file, extract it to any folder of your choice, preserving the directory structure of the zip-file.

Assuming that you are using the Freescale Codewarrior project manager to develop your application, no further installation steps are required. You will find a prepared sample start application, which you should use and modify to write your application. So follow the instructions of section *First steps* on page 11.

You should do this even if you do not intend to use the project manager for your application development to become familiar with embOS.

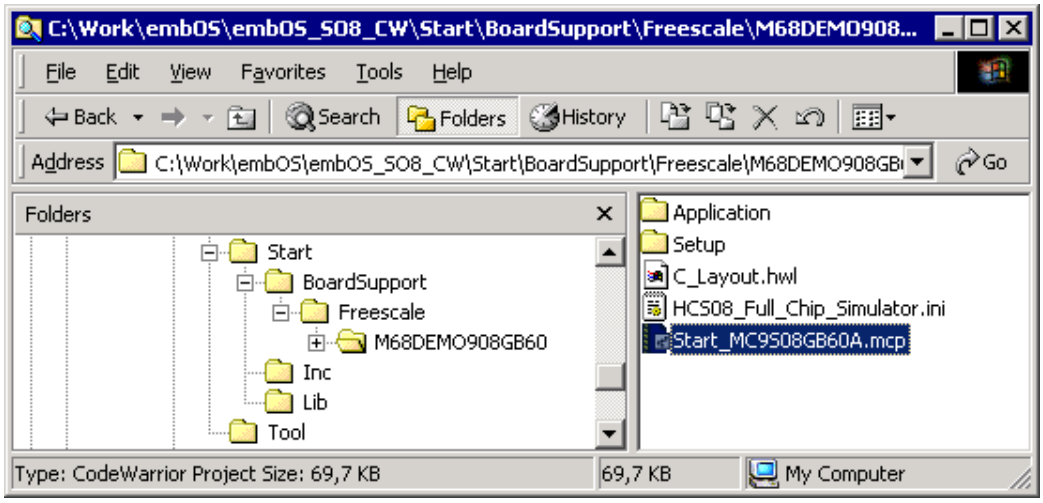
If you will not work with the project manager, you should: Copy either all or only the library-file that you need to your work-directory. This has the advantage that when you switch to an updated version of embOS later in a project, you do not affect older projects that use embOS also. embOS does in no way rely on Freescale Codewarrior project manager, it may be used without the project manager using batch files or a make utility without any problem.

## 1.2 First steps

After installation of embOS you can create your first multitasking application. You received several ready to go sample start workspaces and projects and every other files needed in the subfolder **Start**. It is a good idea to use one of them as a starting point for all of your applications. The subfolder **BoardSupport** contains the workspaces and projects which are located in manufacturer- and CPU-specific subfolders.

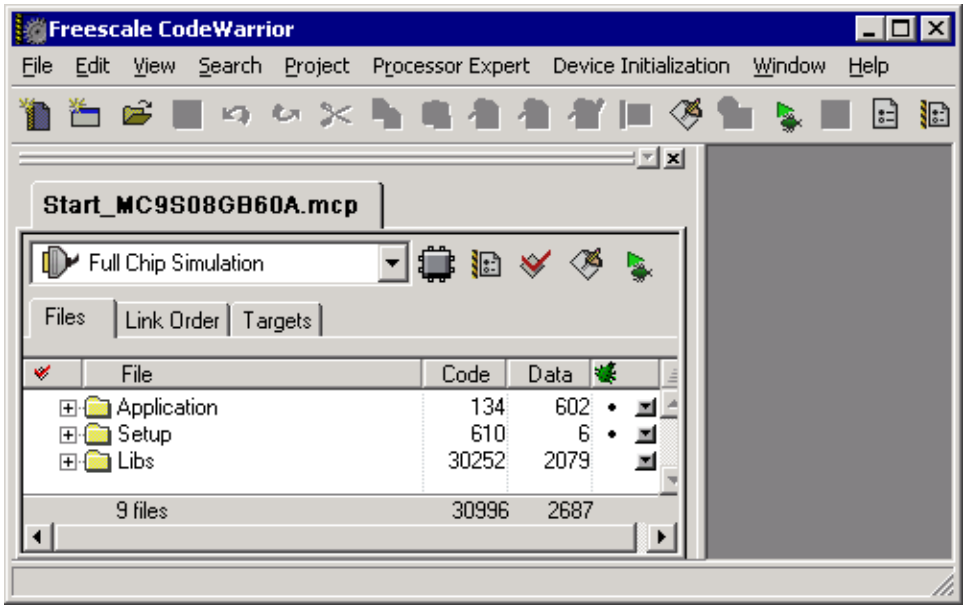
For the first step, you may use the Freescale Codewarrior simulator:




To get your new application running, you should proceed as follows:

- Create a work directory for your application, for example `c:\work`.
- Copy the whole folder **Start** which is part of your embOS distribution into your work directory.
- Clear the read-only attribute of all files in the new **Start** folder.
- Open the sample workspace. **Start\BoardSuppord\Freescale\M68DEMO908GB60\ Start_MC9S08GB60A.mcp** with the Freescale Codewarrior project manager (for example, by double clicking it).
- Build the project. It should be build without any error or warning messages.

After generating the project of your choice, the screen should look for example like this:



For additional information you should open the `ReadMe.txt` file which is part of every specific project. The ReadMe file describes the different configurations of the project and gives additional information about specific hardware settings of the supported eval boards, if required.

## 1.3 The example application Start_2Tasks.c

The following is a printout of the example application `Start_2Tasks.c`. It is a good starting point for your application. (Note that the file actually shipped with your port of embOS may look slightly different from this one.)

What happens is easy to see:

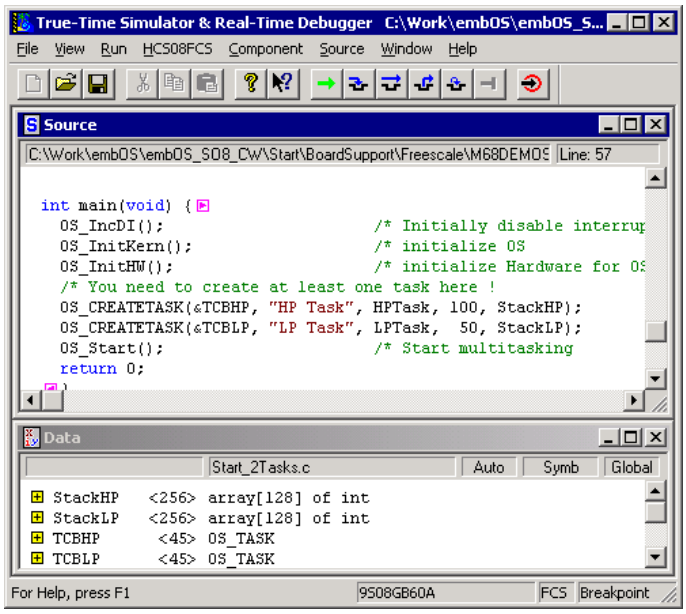
After initialization of embOS; two tasks are created and started.
The two tasks are activated and execute until they run into the delay, then suspend for the specified time and continue execution.

```
/*********************************************************
* SEGGER MICROCONTROLLER SYSTEME GmbH
* Solutions for real time microcontroller applications
**********************************************************
File : Main.c
Purpose : Skeleton program for embOS
--------- END-OF-HEADER ---------------------------------*/

#include "RTOS.H"

OS_STACKPTR int Stack0[128], Stack1[128]; /* Task stacks */
OS_TASK TCB0, TCB1;                       /* Task-control-blocks */



void HPTask(void) {
  while (1) {
    OS_Delay (10);
  }
}

void LPTask(void) {
  while (1) {
    OS_Delay (50);
  }
}

/*********************************************************
*
* main
*
**********************************************************/

void main(void) {
  OS_IncDI();                               /* Initially disable interrupts */
  OS_InitKern();                            /* Initialize OS */
  OS_InitHW();                              /* Initialize Hardware for OS */
  /* You need to create at least one task here ! */
  OS_CREATETASK(&TCB0, "HP Task", HPTask, 100, Stack0);
  OS_CREATETASK(&TCB1, "LP Task", LPTask, 50, Stack1);
  OS_Start();                               /* Start multitasking */
}
```

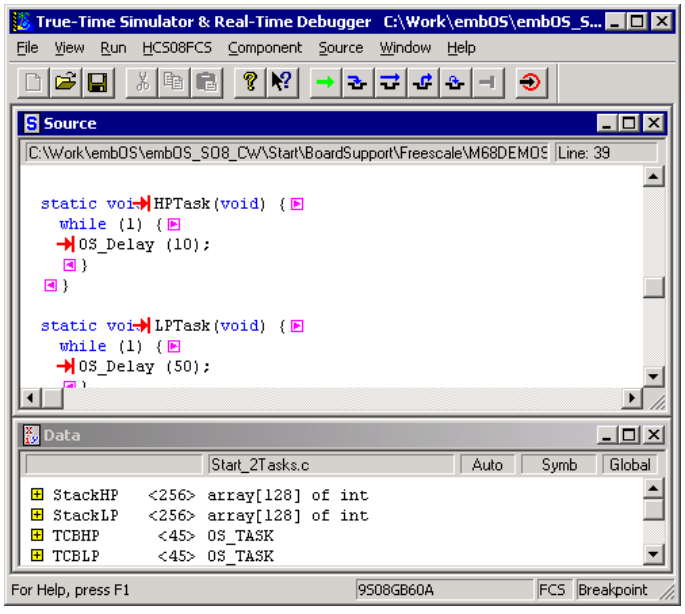
# 1.4 Stepping through the sample application using the simulator

When starting the simulator, you will see the `main` function (see example screenshot below). Now you can step through the program. `OS_IncDI()` initially disables interrupts.
`OS_InitKern()` is part of the embOS library and written in assembler; you can therefore only step into it in disassembly mode. It initializes the relevant OS variables. Because of the previous call of `OS_IncDI()`, interrupts are not enabled during execution of `OS_InitKern()`.
`OS_InitHW()` is part of `RTOSInit_*.c` and therefore part of your application. Its primary purpose is to initialize the hardware required to generate the timer-tick-interrupt for embOS. Step through it to see what is done.

`OS_Start()` should be the last line in main, because it starts multitasking and does not return.



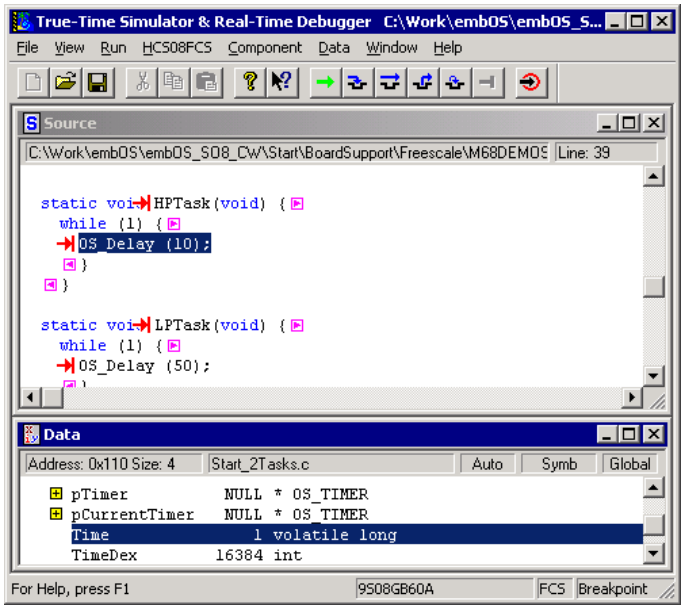

Before you step into `OS_Start()`, you should set two breakpoints in the two tasks as shown below.



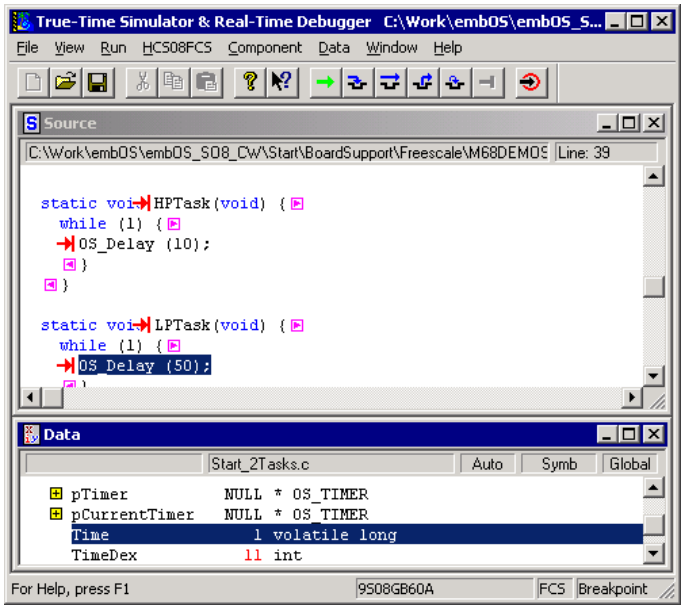

As `OS_Start()` is part of the embOS library, you can step through it in disassembly mode only.

Click **GO**, step over `OS_Start()`, or step into `OS_Start()` in disassembly mode until you reach the highest priority task.

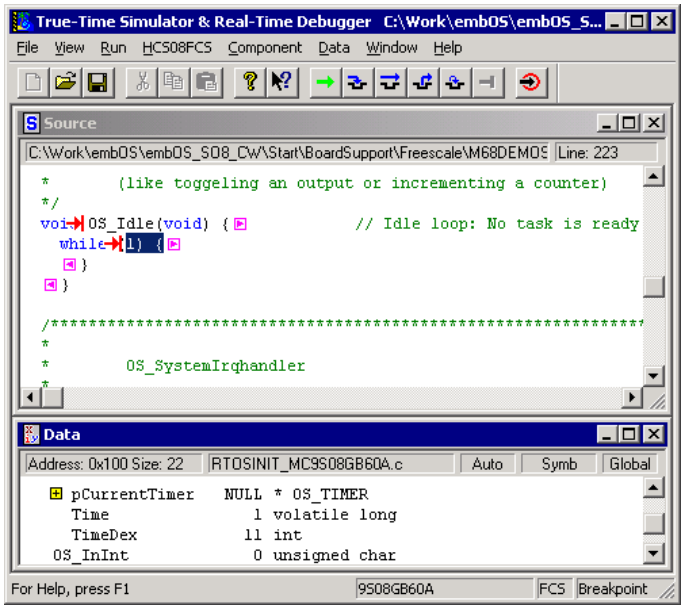



If you continue stepping, you will arrive in the task that has lower priority:



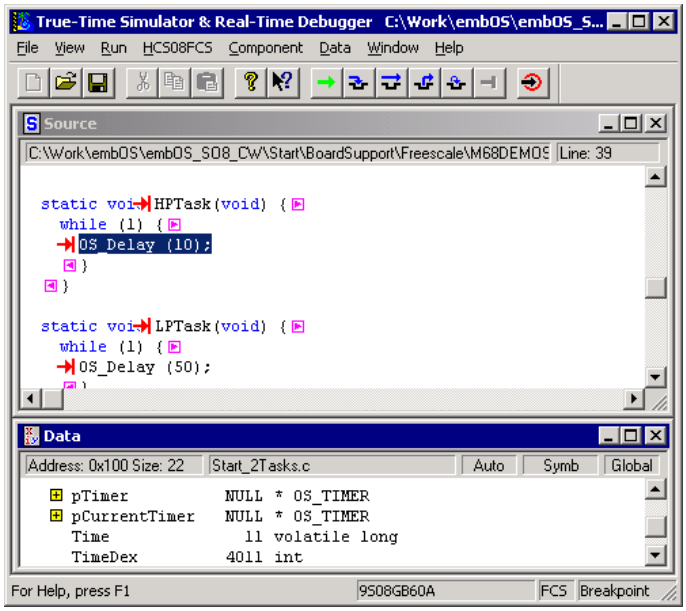

Continue to step through the program, there is no other task ready for execution. embOS will therefore start the idle-loop, which is an endless loop which is always executed if there is nothing else to do (no task is ready, no interrupt routine or timer executing).

You will arrive there when you step into the `OS_Delay()` function in disassembly mode. `OS_Idle()` is part of `RTOSInit*.c`. You may also set a breakpoint there before you step over the delay in `LPTask`.




If you set a breakpoint in one or both of our tasks, you will see that they continue execution after the given delay.

As can be seen by the value of embOS timer variable `OS_Time`, shown in the **Watch** window, `HPTask` continues operation after expiration of the 10 ms delay.



# Chapter 2

# Build your own application

This chapter provides all information to setup your own embOS project.

# 2.1 Introduction

To build your own application, you should always start with one of the supplied sample workspaces and projects. Therefore, select an embOS workspace as described in First steps on page 9 and modify the project to fit your needs. Using a sample project as starting point has the advantage that all necessary files are included and all settings for the project are already done.

## 2.2 Required files for an embOS application

To build an application using embOS, the following files from your embOS distribution are required and have to be included in your project:

- `RTOS.h` from subfolder `Inc\`.
  This header file declares all embOS API functions and data types and has to be included in any source file using embOS functions.
- `RTOSInit_*.c` from one target specific **BoardSupport\<Manufacturer>\<MCU>** subfolder.
  It contains hardware-dependent initialization code for embOS timer and optional UART for embOSView.
- One embOS library from the subfolder `Lib\`.
- `OS_Error.c` from one target specific subfolder **BoardSupport\<Manufacturer>\<MCU>**.
  The error handler is used if any library other than Release build library is used in your project.
- Additional low level init code may be required according to CPU.

When you decide to write your own startup code or use a `__low_level_init()` function, ensure that non-initialized variables are initialized with zero, according to C standard. This is required for some embOS internal variables.
Your `main()` function has to initialize embOS by call of `OS_InitKern()` and `OS_InitHW()` prior any other embOS functions are called.
You should then modify or replace the `Start_2Task.c` source file in the subfolder `Application\`.

## 2.3 Change library mode

For your application you might want to choose another library. For debugging and program development you should use an embOS-debug library. For your final application you may wish to use an embOS-release library or a stack check library.

Therefore you have to select or replace the embOS library in your project or target:

- If your selected library is already available in your project, just select the appropriate configuration.
- To add a library, you may add a new `Lib` group to your project and add this library to the new group. Exclude all other library groups from build, delete unused `Lib` groups or remove them from the configuration.
- Check and set the appropriate `OS_LIBMODE_*` define as preprocessor option and and modify the OS_Config.h file accordingly.

# Chapter 3

# SO8 core specifics

# 3.1 CPU modes

embOS supports nearly all memory and code model combinations that Freescale Codewarrior C/C++ Compiler supports.

# 3.2 Available libraries

embOS for SO8 for Freescale Codewarrior compiler is shipped with different prebuilt libraries

The libraries are named as follows:

`osso8_<LibMode>.lib`

| Parameter | Meaning | Values |
|---|---|---|
| libmode | Specifies the library mode | XR: Extreme Release |
| | | R: Release |
| | | S: Stack check |
| | | D: Debug |
| | | SP: Stack check + profiling |
| | | DP: Debug + profiling |
| | | DT: Debug + profiling + trace |

**Table 3.1: Naming conventions for prebuild libraries**

## Example

`osso8_dp.lib` is the library for a project using SO8 core and debug and profiling features of embOS.

# Chapter 4

# Stacks

## 4.1 Task stack for SO8

Every embOS task has its own individual stack which can be located in any memory area. The stack-size required is the sum of the stack-size of all routines plus basic stack size. The basic stack size is the size of memory required to store the registers of the CPU plus the stack size required by embOS-routines. For the SO8 cpu, the minimum task stack size is about 20 bytes.

## 4.2 System stack for SO8

The embOS system executes in supervisor mode. The minimum system stack size required by embOS is about 80 bytes (stack check & profiling build). However, because the system stack is also used by the application before the start of multi-tasking (the call to `OS_Start()`), and because software-timers and C level interrupt handlers also use the system-stack, the actual stack requirements depend on the application.

The size of the system stack can be changed by modifying `CSTACK` in your linker command file project.prm.

# Chapter 5

# Interrupts

# 5.1 What happens when an interrupt occurs?

- The CPU-core receives an interrupt request.
- As soon as the interrupts are enabled, the interrupt is executed.
- Saving the CPU registers on the stack.
- Setting the I bit in the CCR to mask further interrupts.
- Fetching the interrupt vector for the highest-priority interrupt that is currently pending.
- Filling the instruction queue with the first three bytes of program information starting from theaddress fetched from the interrupt vector locations
- Run the interrupt routine
- Return from interrupt

# 5.2 Defining interrupt handlers in C

Routines preceded by the keyword __*interrupt* save & restore the temporary registers and all registers they modify onto the stack and return with RTI.

The interrupt handler used by embOS are implemented in the CPU specific RTOSInit_*.c file.

### Example

Simple interrupt routine:

```
__interrupt void OS_SystemIrqhandler(void) {
  TPM1C0SC &= ~0x80;
#if (ALLOW_NESTED_INTERRUPTS == 1)
  OS_EnterNestableInterrupt();
#else
  OS_EnterInterrupt();
#endif
  OS_TICK_Handle();
#if (ALLOW_NESTED_INTERRUPTS == 1)
  OS_LeaveNestableInterrupt();
#else
  OS_LeaveInterrupt();
#endif
}
```

Any interrupt handler must call OS_EnterInterrupt()/OS_EnterNestableInterrupt() and OS_LeaveInterrupt()/OS_LeaveNestableInterrupt().

## 5.3 Interrupt Vector Table

The interrupt vector table is defined in the file Rtosinit.c. All unassigned interrupt table entries calls the default handler *_OS_DefaultInterrupt*.

```
__interrupt static void _OS_DefaultInterrupt(void) {
  while (1);
}
```

If you use an additional interrupt please set the interrupt handler in the interrupt vector table.

# Chapter 6

# Technical data

# 6.1 Memory requirements

These values are neither precise nor guaranteed but they give you a good idea of the memory-requirements. They vary depending on the current version of embOS. Using SO8, the minimum ROM requirement for the kernel itself is about 2.500 bytes.

In the table below, which is for release build, you can find minimum RAM size requirements for embOS resources. Note that the sizes depend on selected embOS library mode.

| embOS resource | RAM [bytes] |
|---|---|
| Task control block | 20 |
| Resource semaphore | 7 |
| Counting semaphore | 4 |
| Mailbox | 11 |
| Software timer | 9 |

**Table 6.1: embOS memory requirements**

# Chapter 7

# Files shipped with embOS

## List of files shipped with embOS

| Directory | File | Explanation |
|---|---|---|
| root | *.pdf | Generic API and target specific documentation. |
| root | Release.html | Version control document. |
| root | embOSView.exe | Utility for runtime analysis, described in generic documentation. |
| Start\ BoardSupport\ | | Sample workspaces and project files for Freescale Codewarrior, contained in manufacturer specific sub folders. |
| Start\Inc | RTOS.h BSP.h | Include file for embOS, to be included in every C-file using embOS functions. |
| Start\Lib | os*.r79 | embOS libraries |
| \Application | *.c | Sample applications |
| \Setup | OS_Error.c | embOS runtime error handler used in stack check or debug builds. |
| \Setup | Start08.c | Startup code |
| \Setup | RTOSINIT_MC9S08GB60A.c | embOS cpu specific hardware routines |
| \Setup | BSP.c | Board support package |
| \Setup | project.prm | Linker command file |

**Table 7.1: Files shipped with embOS**

Any additional files shipped serve as example.

# Index

