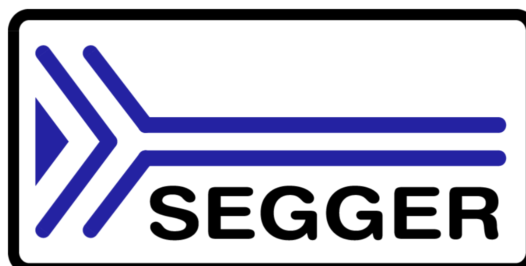
embOS

Real-Time
Operating System

CPU & Compiler
specifics for Freescale
HCS12 using
CodeWarrior compiler

Document: UM01038
Software version 3.86f
Revision: 0
Date: July 27, 2012



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 (SEGGER) assumes no responsibility for any errors or omissions. SEGGER makes and you receive no warranties or conditions, express, implied, statutory or in any communication with you. SEGGER specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

Copyright notice

You may not extract portions of this manual or modify the PDF file in any way without the prior written permission of SEGGER. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

© 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG, Hilden / Germany

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 GmbH & Co. KG

In den Weiden 11
D-40721 Hilden

Germany

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

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: July 27, 2012

Software	Revision	Date	By	Description
3.86f	0	120727	TS	First version.

© 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

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

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

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.
`Sample comment`	Comments in programm examples.
Reference	Reference to chapters, sections, tables and figures or other documents.
GUIElement	Buttons, dialog boxes, menu names, menu commands.
Emphasis	Very important sections.

Table 2.1: Typographic conventions

© 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

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 micro controllers, 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.

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.

embOS/IP
TCP/IP stack
embOS/IP a high-performance TCP/IP stack that has been optimized for speed, versatility and a small memory footprint.

emFile
File system
emFile is an embedded file system with FAT12, FAT16 and FAT32 support. Various Device drivers, e.g. for NAND and NOR flashes, SD/MMC and Compact-Flash cards, are available.

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

SEGGER TOOLS

Flasher
Flash programmer
Flash Programming tool primarily for micro controllers.

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.

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

Table of Contents

1 Using embOS for Freescale HCS12/HCS12X 9

1.1 Installation 10
1.2 First steps 11
1.3 The example application Start_2Tasks.c 12
1.4 Stepping through the sample application 13

2 Build your own application 17

2.1 Introduction 18
2.2 Required files for an embOS for HCS12 18
2.3 Change library mode 18
2.4 Select another CPU 18

3 HCS12/HCS12X CPU specifics 21

3.1 CPU modes 22
3.2 Available libraries 22

4 Compiler specifics 25

4.1 Standard system libraries 26

5 Stacks 27

5.1 Task stack for HCS12/HCS12X 28
5.2 System stack for HCS12/HCS12X 28
5.3 Interrupt stack for HCS12/HCS12X 28

6 Interrupts 29

6.1 What happens when an interrupt occurs? 30
6.2 Defining interrupt handlers in C 30
6.3 Interrupt vector table 30
6.4 Zero latency interrupts 30

7 STOP / WAIT Mode 33

7.1 Introduction 34

8 Technical data 35

8.1 Memory requirements 36

9 Files shipped with embOS 37

© 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

Chapter 1

Using embOS for Freescale HCS12/HCS12X

This chapter describes how to start with and use embOS for Freescale HCS12/HCS12X cores and the CodeWarrior compiler. You should follow these steps to become familiar with embOS.

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

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 CodeWarrior project manager to develop your application, no further installation steps are required. You will find a lot of prepared sample start projects, 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.

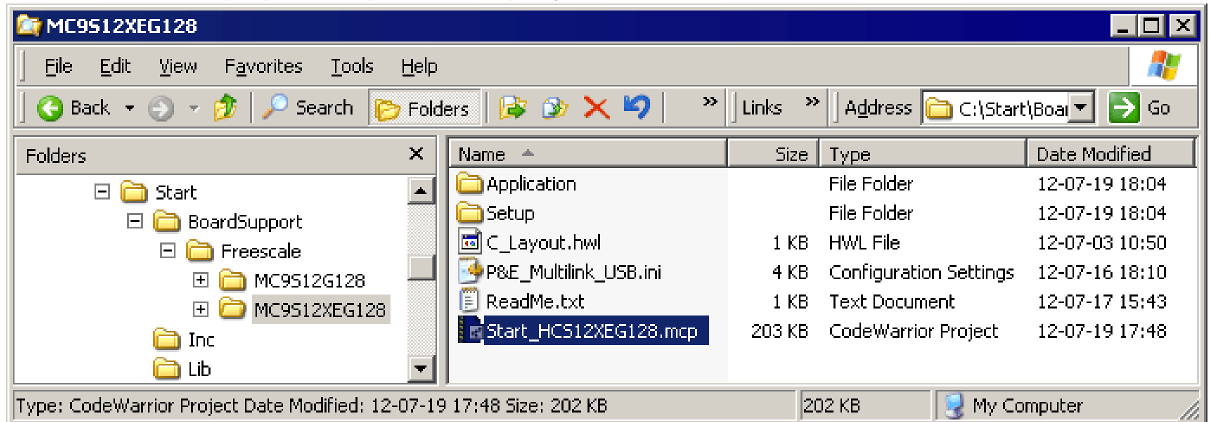
If you will not work with the embedded workbench, you should: Copy either all or only the library-file that you need to your work-directory. This has the advantage that when you switch to an updated version of embOS later in a project, you do not affect older projects that use embOS also. embOS does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

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

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.

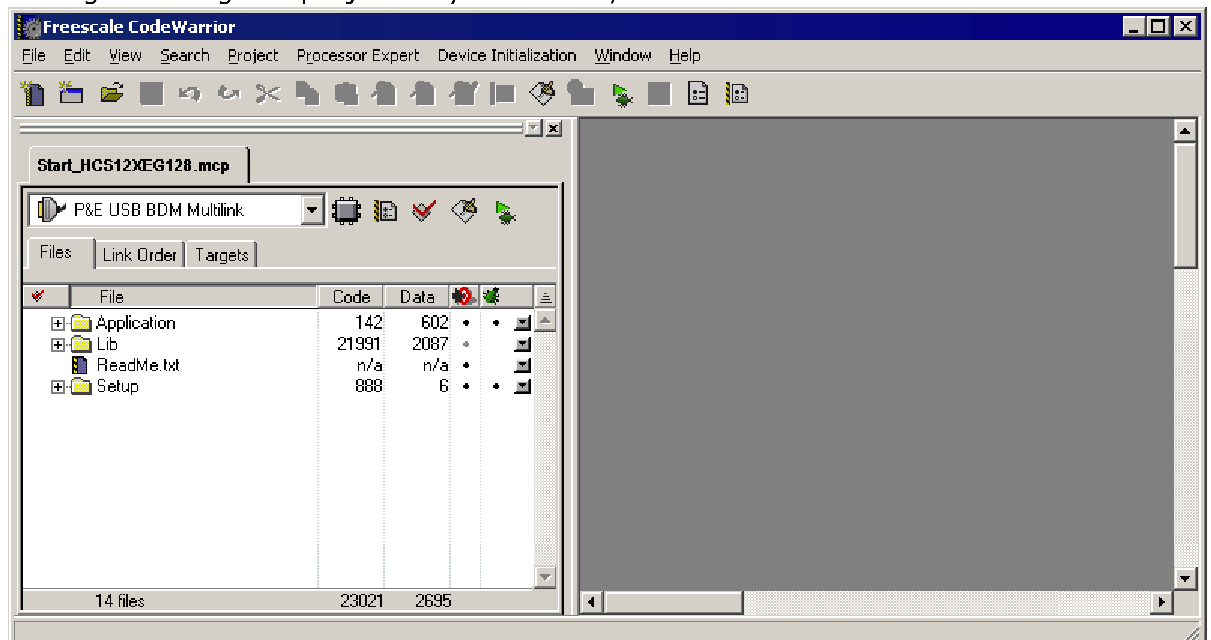
For the first step, you may use the project for MC9S12XEG128 CPU:



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 project **Start\BoardSupport\Freescale\MC9S12XEG128** with the 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 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.

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

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 & Co.KG
* Solutions for real time microcontroller applications
***********************************************************
File    : Start2Tasks.c
Purpose : Skeleton program for embOS
--------- END-OF-HEADER ---------------------------------*/

#include "RTOS.h"

OS_STACKPTR int StackHP[128], StackLP[128]; /* Task stacks */
OS_TASK TCBHP, TCBLP;                       /* 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(&TCBHP, "HP Task", HPTask, 100, StackHP);
  OS_CREATETASK(&TCBLP, "LP Task", LPTask,  50, StackLP);
  OS_Start();                        /* Start multitasking */
  return 0;
}
```


© 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

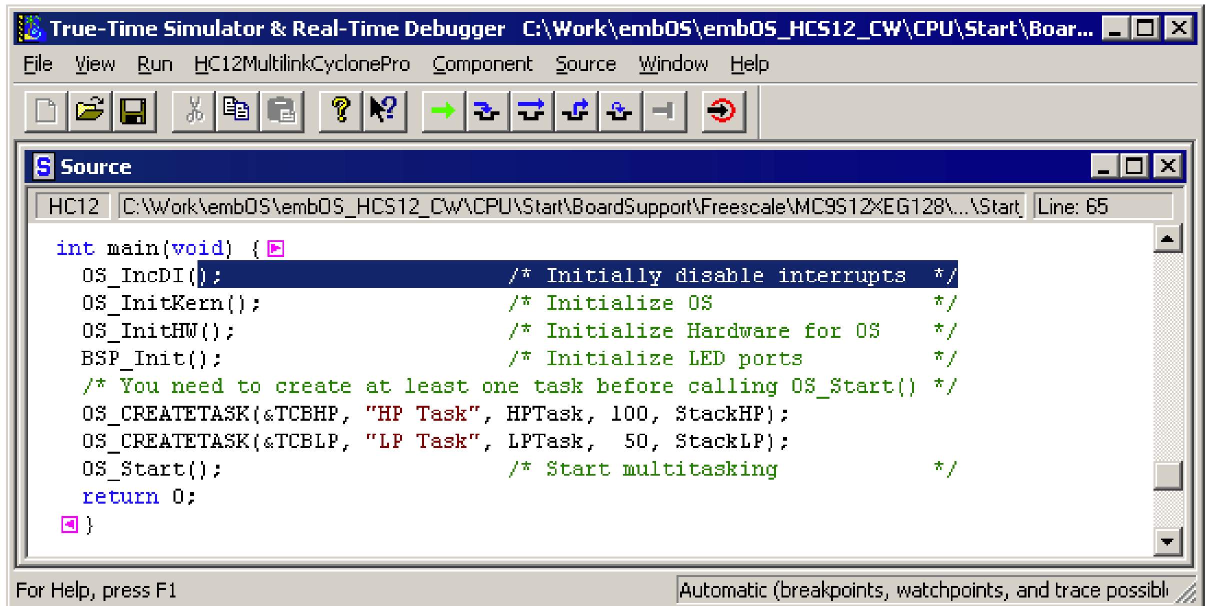
1.4 Stepping through the sample application

When starting the debugger, you will see the `main` function (see example screenshot below). Now you can step through the program. `OS_IncDI()` initially disables interrupts.

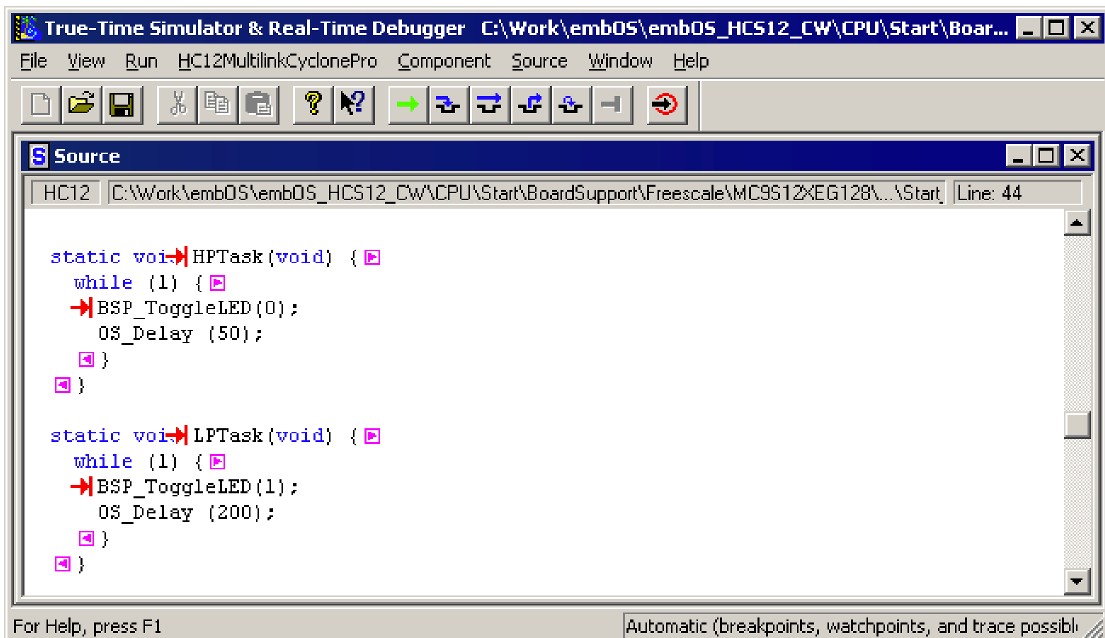
`OS_InitKern()` is part of the embOS library; you can therefore only step into it in disassembly mode. It initializes the relevant OS variables. Because of the previous call of `OS_IncDI()`, interrupts are not enabled during execution of `OS_InitKern()`.

`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.



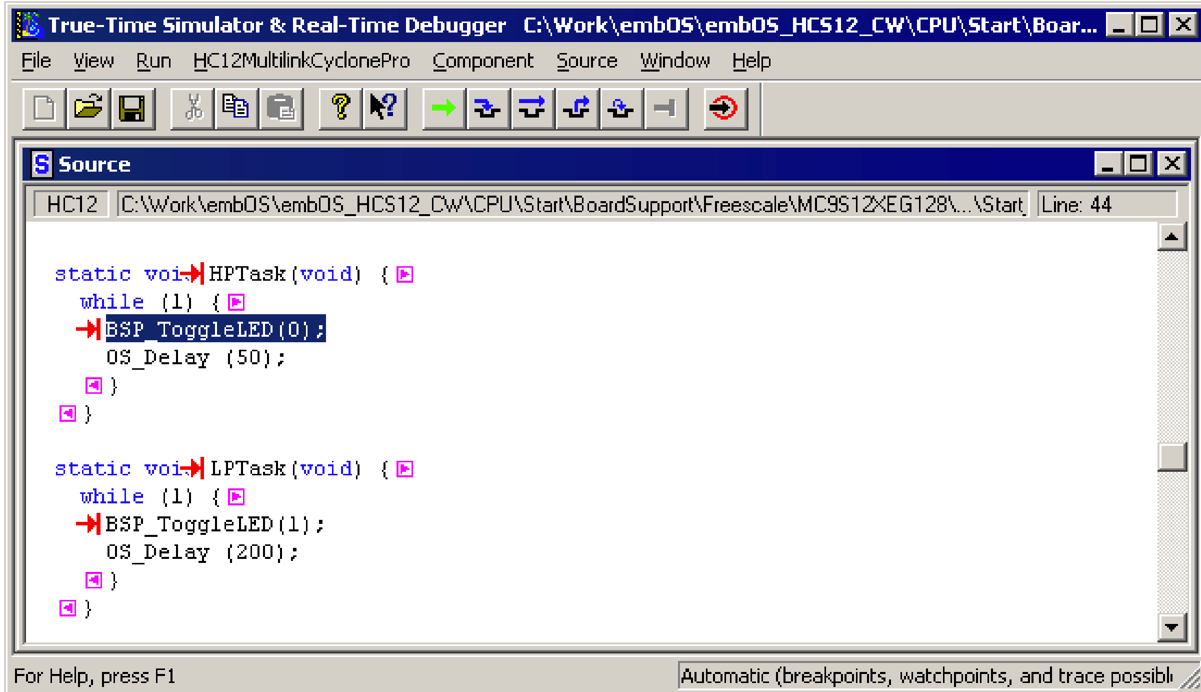
Before you step into `OS_Start()`, you should set two breakpoints in the two tasks as shown below.



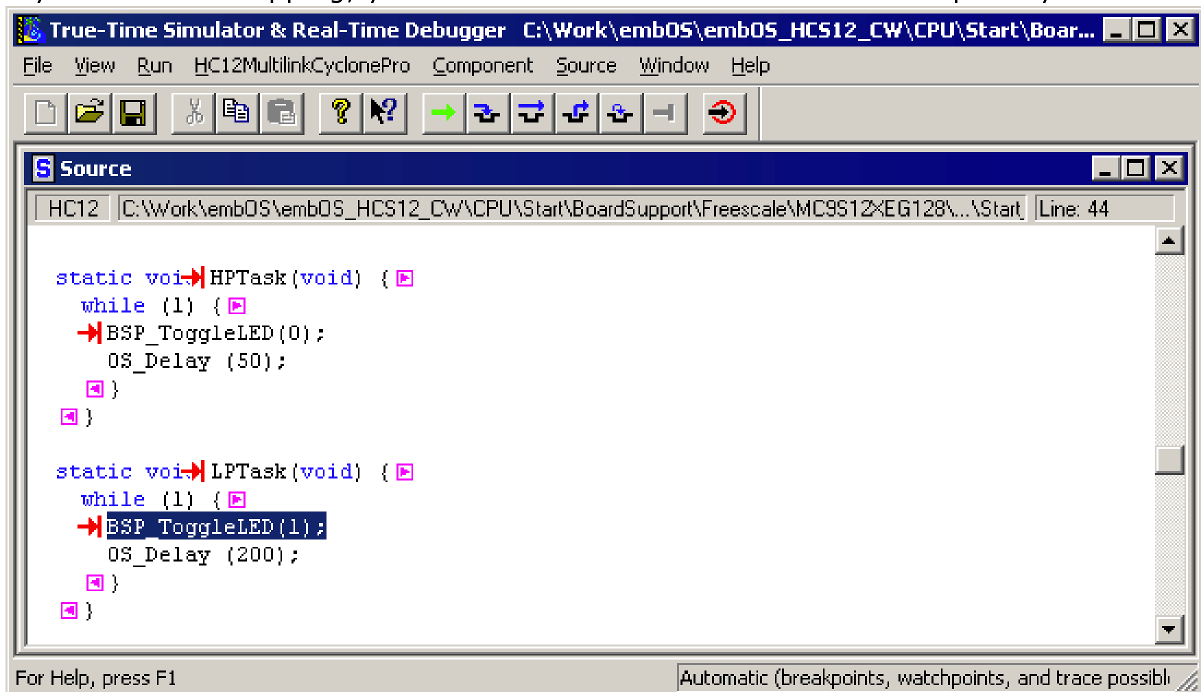
As `OS_Start()` is part of the embOS library, you can step through it in disassembly mode only.

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

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



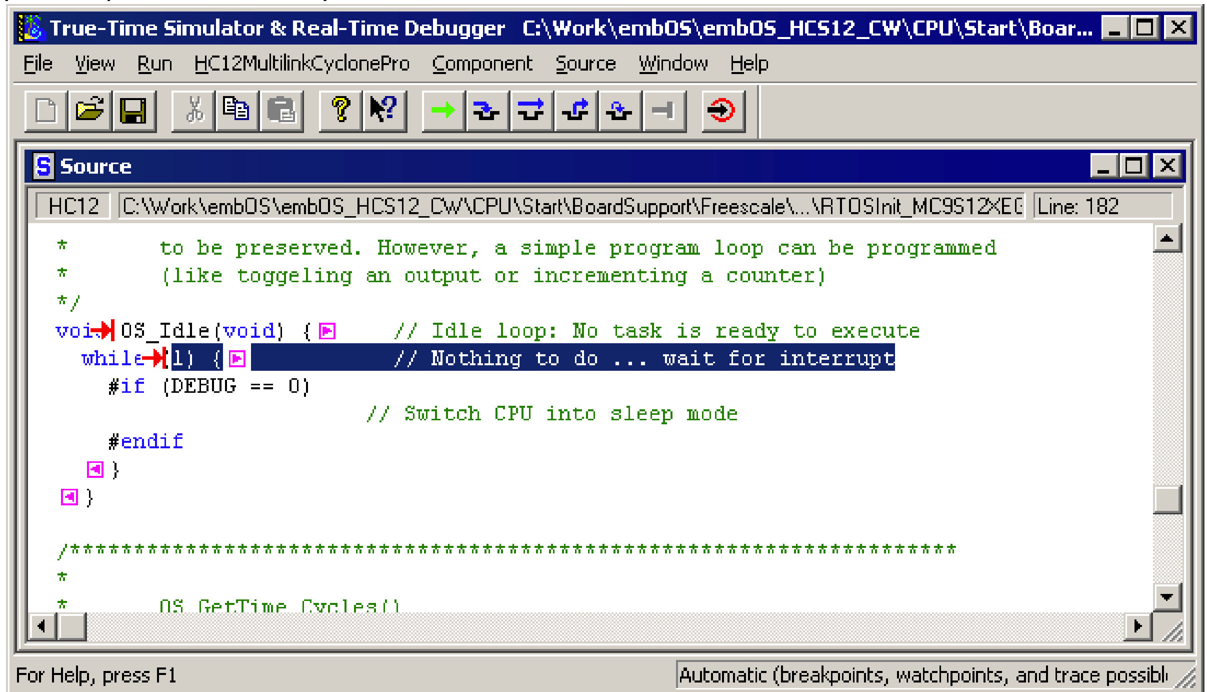
If you continue stepping, you will arrive in the task that has lower priority:



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).

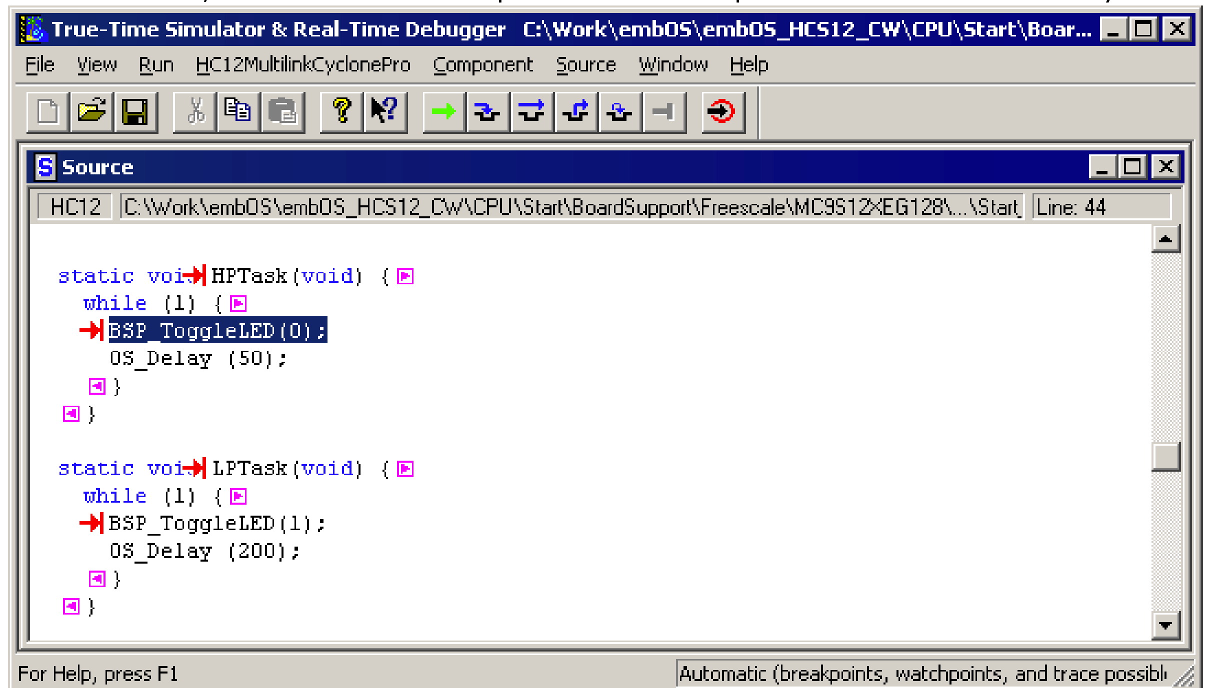
 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

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_Global.Time, shown in the Watch window, `HPTask` continues operation after expiration of the 50 ms delay.



 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

Chapter 2

Build your own application

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

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

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 for HCS12

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>\Setup** subfolder.
  It contains hardware-dependent initialization code for embOS. It initializes the system timer, timer interrupt and optional communication for embOSView via UART or JTAG.
- One embOS library from the subfolder `Lib\`.
- OS_Error.c from one target specific subfolder **BoardSupport\<Manufacturer>\<MCU>\Setup**.
  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 this startup code initializes all non-initialized variables are initialized with zero, according to C standard. This is required for some embOS internal variables.
Also ensure, that `main()` is called with the CPU running in supervisor mode.
Your `main()` function has to initialize embOS by a 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/or modify the OS_Config.h file accordingly.

2.4 Select another CPU

embOS contains CPU-specific code for various HCS12/HCS12X CPUs. Manufacturer- and CPU specific sample start workspaces and projects are located in the subfolders of the **BoardSupport** folder. To select a CPU which is already supported, just select the appropriate workspace from a CPU specific folder.

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

If your HCS12/HCS12X CPU is currently not supported, examine all `RTOSInit` files in the CPU-specific subfolders and select one which almost fits your CPU. You may have to modify `OS_InitHW()`, `OS_COM_Init()`, and the interrupt service routines for embOS timer tick and communication to embOSView.

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

Chapter 3

HCS12/HCS12X CPU specifics

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

3.1 CPU modes

embOS for HCS12/HCS12X supports nearly all memory and code model combinations that CodeWarrior compiler supports. Large memory model is not supported.

3.2 Available libraries

embOS for HCS12 and CodeWarrior compiler comes with 28 different libraries, one for each CPU core / memory model / library mode combination. The library names follow the naming convention for the system libraries from CodeWarrior.

© 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

3.2.1 Naming conventions for prebuilt libraries

embOS HCS12 for CodeWarrior is shipped with different prebuilt libraries with different combinations of the following features:

- CPU core - Core
- Memory model - MemModel
- Library mode - LibMode

The libraries are named as follows:

os<Core><MemModel>i_<Libmode>.a

Parameter	Meaning	Values
Core	Specifies the CPU mode.	x: HCS12X core _: HCS12 core
MemModel	Specifies the memory model	s: Small memory model b: Banked memory model
LibMode	Specifies the library mode	xr: Extreme Release
		r: Release
		s: Stack check
		sp: Stack check + profiling
		d: Debug
		dp: Debug + profiling
		dt: Debug + profiling + trace

Example

osxbi_DP.lib is the library for a project using a HCS12X core, banked memory model with debug and profiling support.

© 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

Chapter 4

Compiler specifics

4.1 Standard system libraries

embOS for HCS12 and CodeWarrior compiler may be used with standard CodeWarrior system libraries for most of all projects. For heap management, embOS delivers its own thread safe functions which may be used. These functions are described in the embOS generic manual.

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

Chapter 5

Stacks

© 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

5.1 Task stack for HCS12/HCS12X

Each task uses its individual stack. The stack poiter is initialized and set every time a task is activated by the scheduler. The stack-size required for a task is the sum of the stack-size of all routines plus a basic stack size plus size used by exceptions.

The basic stack size is the size of memory required to store the registers of the CPU plus the stack size required by calling embOS-routines.

For the HCS12/HCS12X CPUs, this minimum basic task stack size is about 64 bytes. Because any function call uses some amount of stack and every exception also pushes at least 9 bytes for HCS12 and 10 bytes for HCS12X onto the current stack, the task stack size has to be large enough to handle exceptions too. We recommend at least 256 bytes stack as a start.

5.2 System stack for HCS12/HCS12X

The minimum system stack size required by embOS is about 128 bytes (stack check & profiling build) However, since the system stack is also used by the application before the start of multitasking (the call to OS_Start()), and because software-timers and 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 the STACKSIZE define in your *.prm linker file.
We recommend a minimum stack size of 256 bytes for the system stack.

5.3 Interrupt stack for HCS12/HCS12X

The HCS12/HCS12X core has no separate interrupt stack pointer. Interrupts are executed on the current stack which could be task stack or system stack.

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

Chapter 6

Interrupts

The HCS12(X) core supports up to 128 separate interrupt sources. The real number of interrupt sources depends on the specific target CPU.

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

6.1 What happens when an interrupt occurs?

- The CPU-core receives an interrupt request form the interrupt controller.
- As soon as the interrupts are enabled, the interrupt is accepted and executed.
- The CPU pushes temporary registers and the return address onto the current stack.
- The CPU jumps to the vector address from the vector table
- The interrupt handler is processed.
- The interrupt handler ends with a return from interrupt
- The CPU restores the temporary registers and return address from the stack and continues the interrupted function.

6.2 Defining interrupt handlers in C

Interrupt handlers for HCS12/HCS12X cores are written as normal C-functions which do not take parameters and do not return any value. The interrupt function must be marked with the key word *__interrupt*. Since the vector table entries are 16Bit wide the interrupt handler functions must be in the near area.
Interrupt handler which call an embOS function need a prolog and epilog function as described in the generic manual and in the examples below.

Example

Simple interrupt routine:

```
#pragma CODE_SEG __NEAR_SEG NON_BANKED
__interrupt void __near OS_ISR_Tick (void) {
  PITTF = 0x01u; // Reset interrupt request flag
  OS_EnterNestableInterrupt();
  OS_TICK_Handle();
  OS_LeaveNestableInterrupt();
}
#pragma CODE_SEG DEFAULT
```

6.3 Interrupt vector table

The interrupt vector table is located in a C source file. All interrupt handler function addresses have to be inserted in the vector table.

6.4 Zero latency interrupts

6.4.1 Zero latency interrupts with HCS12X

Zero interrupt latency is supported with HCS12X but not with HCS12.

Instead of disabling interrupts when embOS does atomic operations, the interrupt level of the CPU is set to 4. Therefore all interrupt priorities higher than 4 can still be processed. You must not execute any embOS function from within a zero latency interrupt function.

6.4.2 OS_SetFastIntPriorityLimit(): Set the interrupt priority limit for zero latency interrupts

The interrupt priority limit for fast interrupts is set to 4 by default. This means, all interrupts with higher priority from 4 up to the maximum CPU specific priority will never be disabled by embOS.

Description

OS_SetFastIntPriorityLimit() is used to set the interrupt priority limit between zero latency interrupts and lower priority embOS interrupts.

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

Prototype

```
void OS_SetFastIntPriorityLimit(unsigned int Priority)
```

Parameter	Meaning
Priority	The highest value useable as priority for embOS interrupts. All interrupts with higher priority are never disabled by embOS. Valid range: 1 <= Priority <= 7

Return value

NONE.

Add. information

To disable zero latency interrupts at all, the priority limit may be set to the highest interrupt priority supported by the CPU, which is 7.

To modify the default priority limit, *OS_SetFastIntPriorityLimit()* should be called before embOS was started.
In the default projects, *OS_SetFastIntPriorityLimit()* is not called. The start projects use the default zero latency interrupt priority limit.
Any interrupts running above the zero latency interrupt priority limit must not call any embOS function.

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

Chapter 7

STOP / WAIT Mode

7.1 Introduction

In case your controller does support some kind of power saving mode, it should be possible to use it also with embOS, as long as the timer keeps working and timer interrupts are processed. To enter that mode, you usually have to implement some special sequence in the function `OS_Idle()`, which you can find in embOS module `RTOSInit.c`.

Chapter 8

Technical data

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

8.1 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. 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	18
Resource semaphore	4
Counting semaphore	4
Mailbox	12
Software timer	12

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

Chapter 9

Files shipped with embOS

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

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 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??_*.lib	embOS libraries for CodeWarrior compiler
Start\BoardSupport\..\Setup	OS_Error.c	embOS runtime error handler used in stack check or debug builds.
Start\BoardSupport\...\Setup\	*.*	CPU specific hardware routines for various CPUs.

Any additional files shipped serve as example.

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

Index

C

CPU modes 22

I

Installation 10
interrupt handlers 30
Interrupt stack 28
Interrupt vector table 30
Interrupts 29

L

libraries 22

M

Memory requirements 36

O

OS_SetFastIntPriorityLimit() 30

S

Stacks 27
Syntax, conventions used 5
System stack 28

T

Task stack 28

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG

 © 2010 - 2012 SEGGER Microcontroller GmbH & Co. KG