embOS

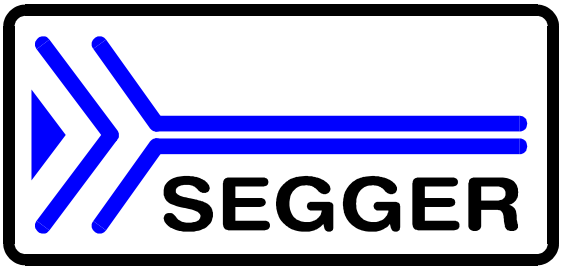
Real Time Operating System

CPU & Compiler specifics for

RENESAS R8C CPUs

and HEW workbench

Document Rev. 1



A product of Segger Microcontroller Systeme GmbH

www.segger.com

Contents

1. About this document

This guide describes how to use ***embOS*** for R8C Real Time Operating System for the RENESAS R8C series of microcontroller using RENESAS NC30 compiler version 5.40 and HEW workbench version 4.

1.1. How to use this manual

This manual describes all CPU and compiler specifics for ***embOS*** using R8C CPUs with NC30 compiler. Before actually using ***embOS***, you should read or at least glance through this manual in order to become familiar with the software.
Chapter 2 gives you a step-by-step introduction, how to install and use ***embOS*** using RENESAS High-performance Embedded Workshop HEW. If you have no experience using ***embOS***, you should follow this introduction, even if you do not plan to use RENESAS HEW, because it is the easiest way to learn how to use ***embOS*** in your application.
Most of the other chapters in this document are intended to provide you with detailed information about functionality and fine-tuning of ***embOS*** for the R8C CPUs and NC30 compiler.

2. Using *embOS* with RENESAS HEW

2.1. Installation

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

In order 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, please keep all files in their respective sub directories. Make sure the files are not read only after copying.
If you received a zip-file, please extract it to any folder of your choice, preserving the directory structure of the zip-file.

Assuming that you are using RENESAS HEW to develop your application, no further installation steps are required. You will find a prepared sample workspace and sample start project for R8C CPU, which you should use and modify to write your application. So follow the instructions of the next chapter ‘First steps’.

You should do this even if you do not intend to use RENESAS HEW for your application development in order to become familiar with *embOS.*

embOS does in no way rely on RENESAS HEW, it may be used without the workbench using batch files or a make utility without any problem.

2.2. First steps

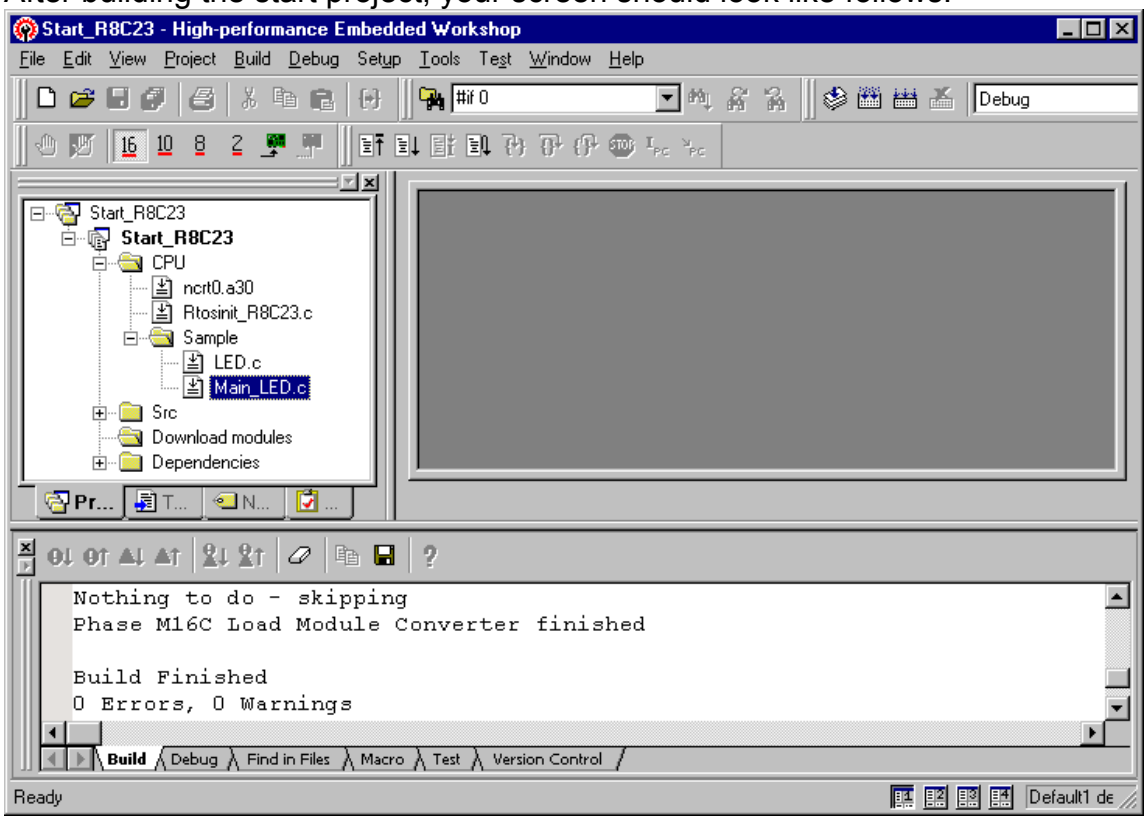
After installation of ***embOS*** (→ Installation) you are able to create your first multitasking application. You received a ready to go sample start project for R8C CPUs and it is a good idea to use this as a starting point of all your applications.

Your ***embOS*** distribution contains everything you need for NC30 compiler version 5.40 and HEW version 4. If you have to use compiler version 5.3 or lower and RENESAS Tool manager, the sample start workspace can not be used.

For NC30 compiler version 5.40 and HEW version 4 which is explained in this manual, you should:

- Create a work directory for your application, for example c:\work
- Copy all files and subdirectories from the folder 'embOS_R8C_HEW' from your ***embOS*** distribution into your work directory.
- Clear the read only attribute of all files in the new 'Start'-folder in your working directory.
- Open the folder 'Start'
- Open the start workspace 'Start_R8C*.hws'. (e.g. by double clicking it)
- Build the start project

After building the start project, your screen should look like follows:



2.3. The sample application Main.c

The following is a printout of the sample application main.c. It is a good starting-point for your application.
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         *
**********************************************************************
*                                                                    *
*       (C) 2006   SEGGER Microcontroller Systeme GmbH               *
*                                                                    *
*       www.segger.com     Support: support@segger.com               *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File    : Main.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 */

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

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

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

int 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;
}
```

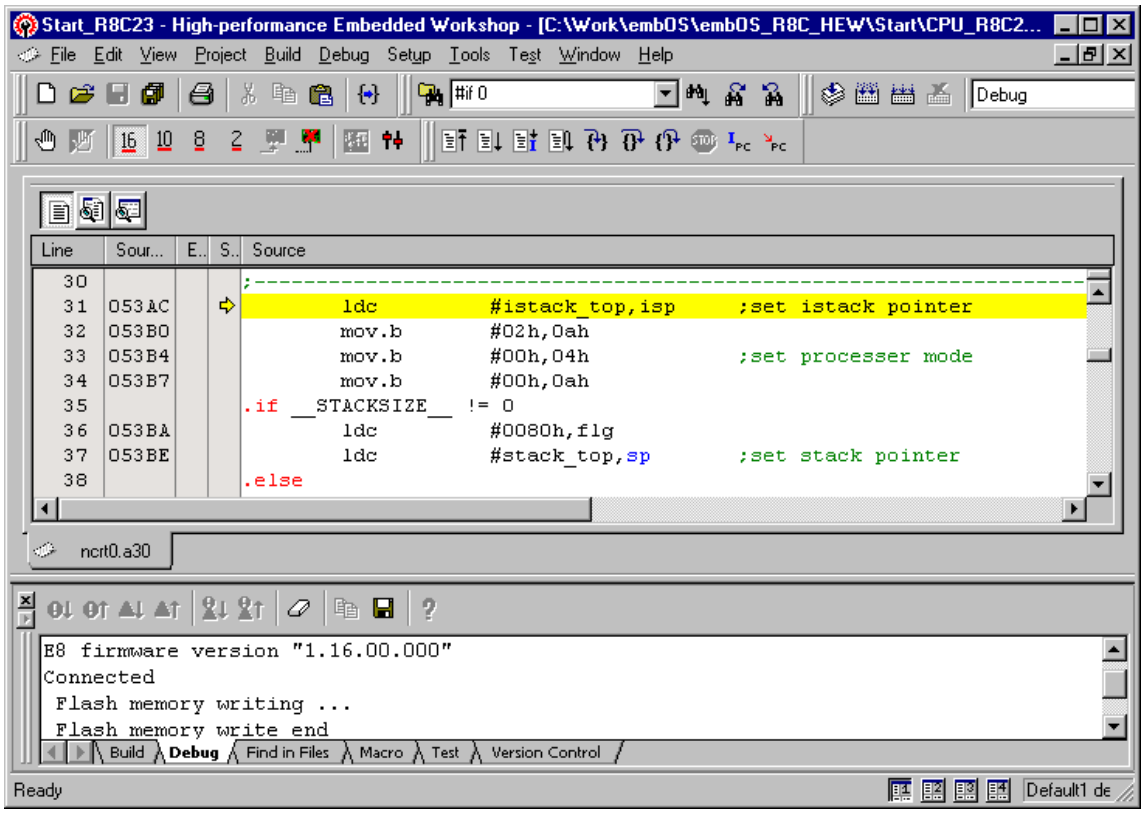
3. Using debugging tools to debug the application

The ***embOS*** start project is configured to produce an output file which can directly used with the E8 debugger.
The following chapter describes a sample session based on our sample application MainLed.c.

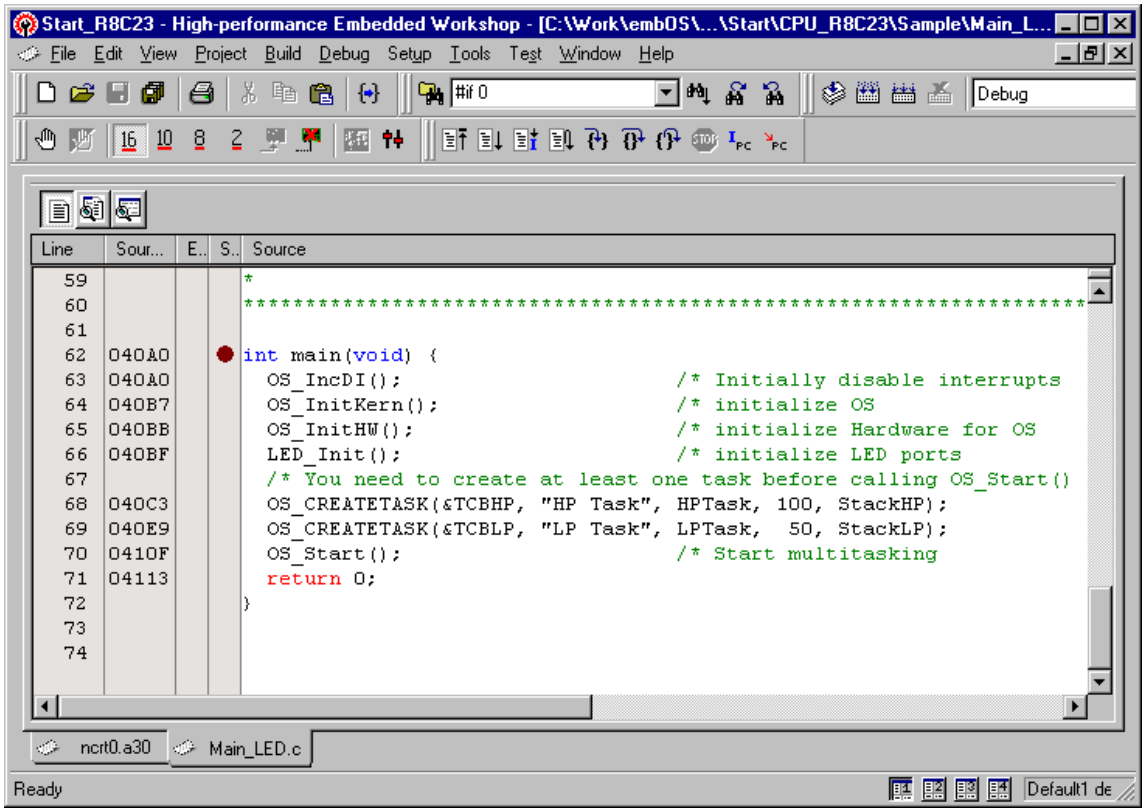
3.1. Debug the application using Renesas E8 debugger

The easiest way to debug the start project is using the E8 debugger. Please ensure that you have select the build session “Debug” and the debug session “Debug_E8”.
When you choose “Debug -> Connect” from the main menu, The “Init (R8C debugger)” dialog appears. Depending on other options, the debugger then automatically loads the target file. If not please choose “Debug -> Download Modules” from the main menu.

The Debugger will load the file and show the startup code:



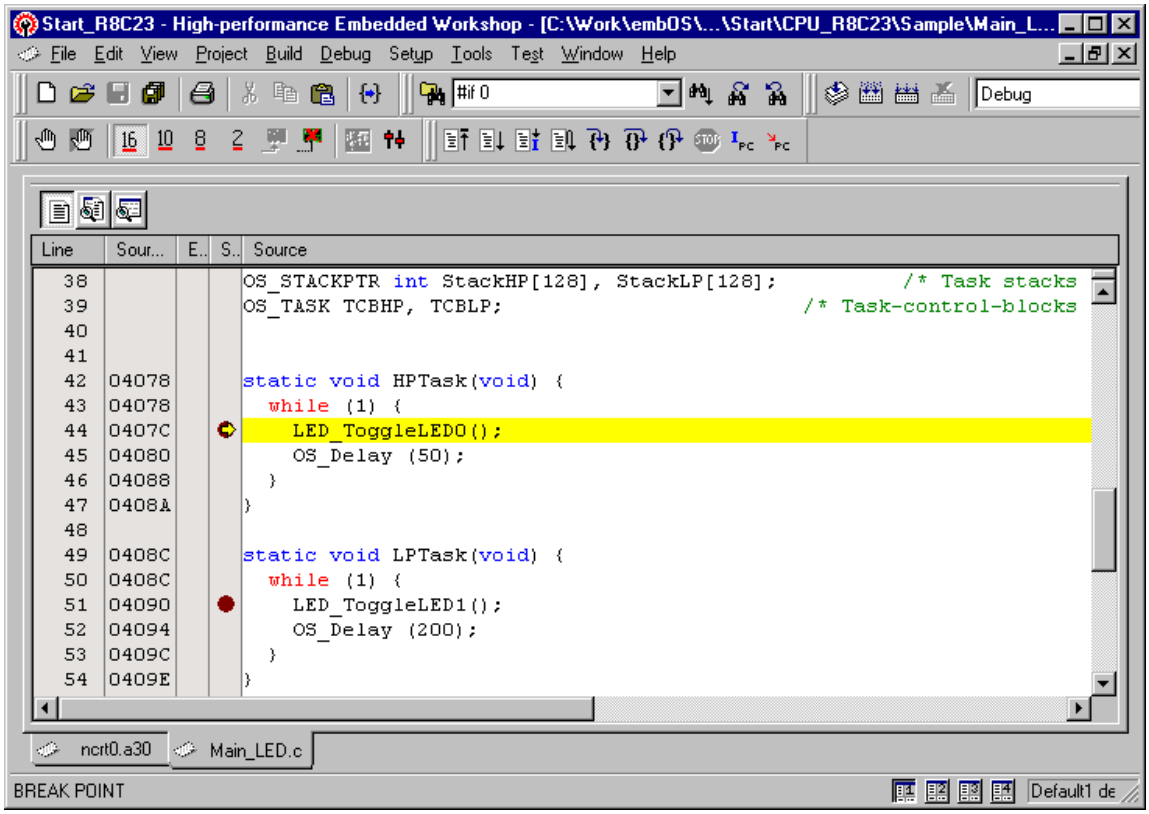
You should open or select the MainLed.c file and set a breakpoint at main()
When you then start the CPU by “Debug -> Go” or just press F5, the debugger stops at main. Alternatively, you may step through the startup code to get there:



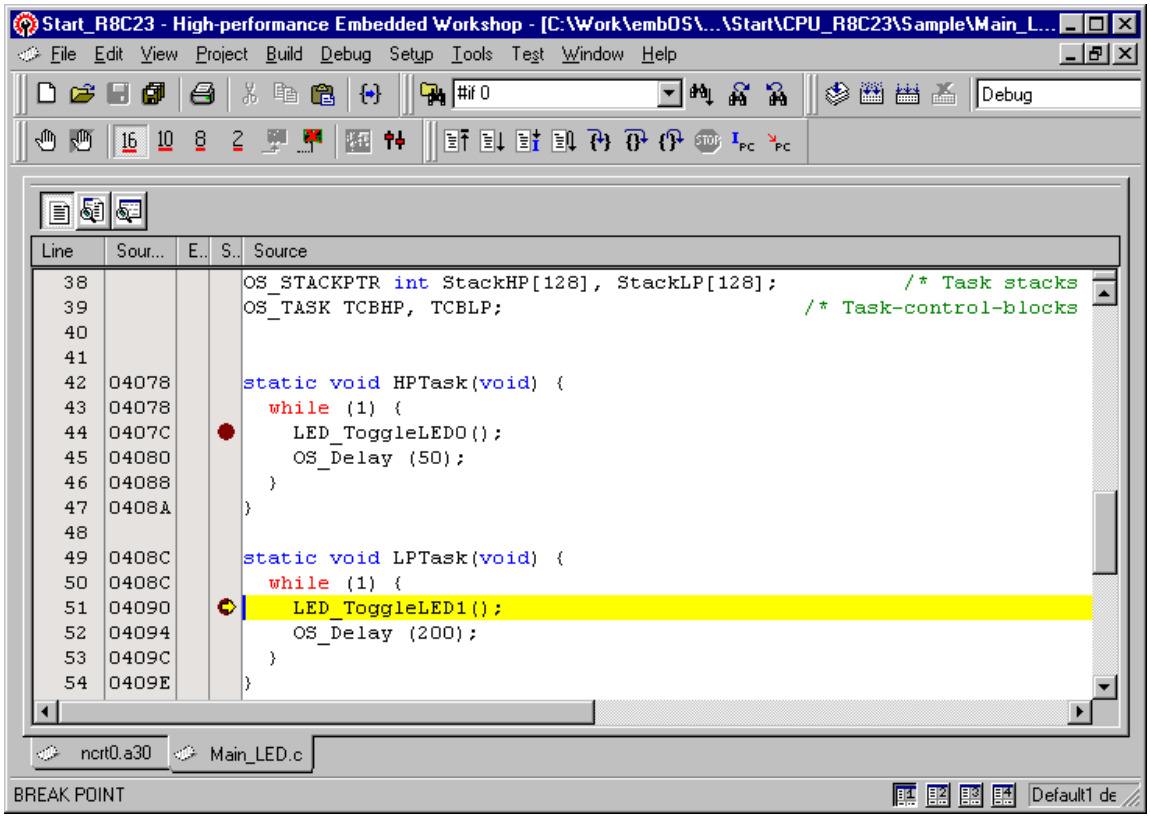
You may now step through the sample application.

- `OS_IncDI()` Initially disables interrupts and prevents re-enabling them in `OS_InitKern()`.
- `OS_InitKern()` is part of the ***embOS*** Library; you can therefore only step into it in disassembly mode. It initializes the relevant OS-Variables.
- `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, since it starts multitasking and does not return. `OS_Start()` automatically enables interrupts.

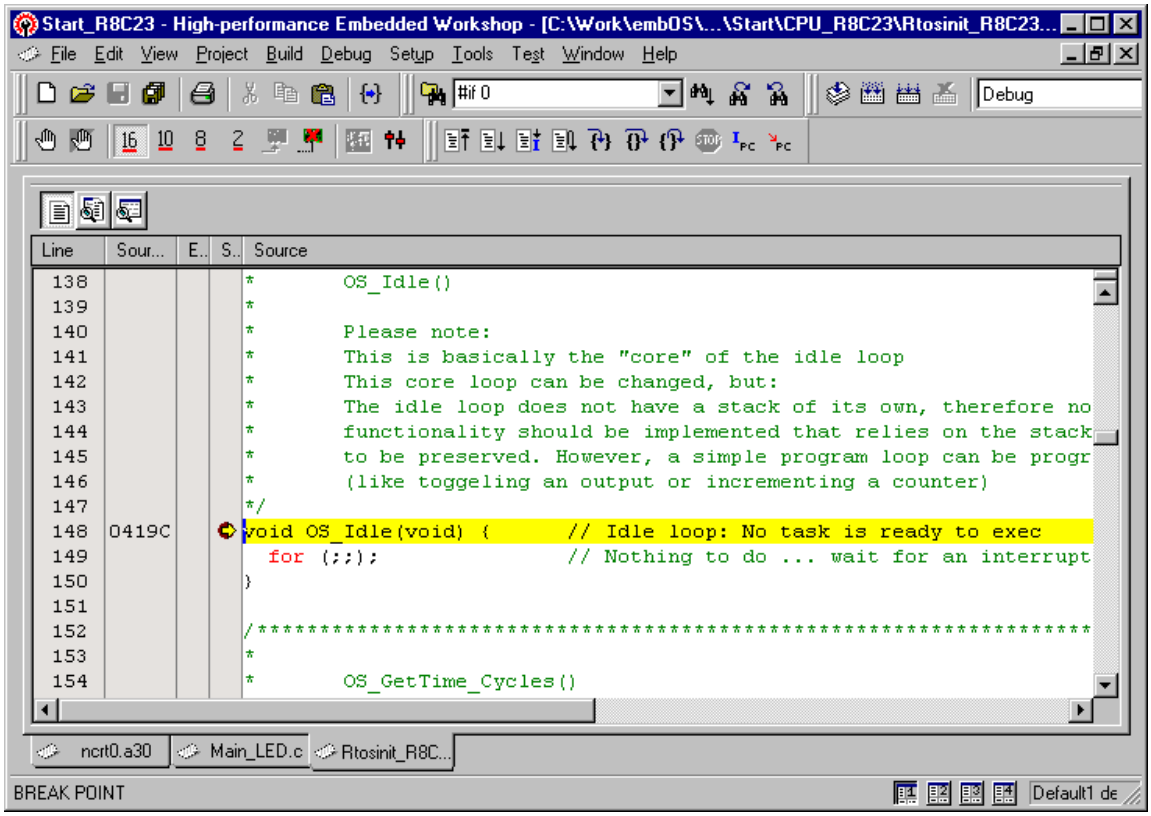
When you step into `OS_Start()`, the next line executed is already in the highest priority task created. (you may also use disassembly mode to get there of course, then stepping through the task switching process, but you must not step over `OS_Start()`). In our small start program, `HPTask()` is the highest priority task and is therefore active:



You should set a breakpoint in every task, as shown above. If you continue stepping, you will arrive in the task with the second highest priority:

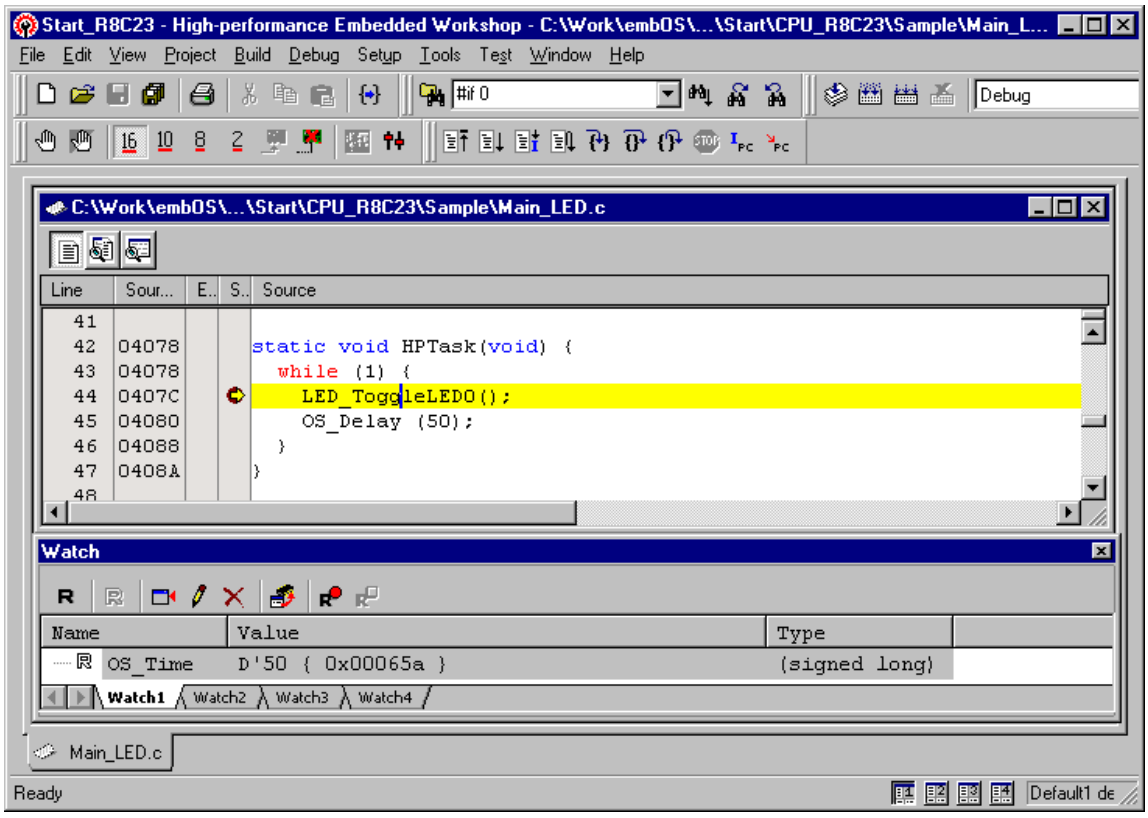


Continuing to step through the program, there is no other task ready for execution. ***embOS*** will therefore start the idle-loop, which is an endless loop which is always executed if there is nothing else to do (no task is ready, no interrupt routine or timer executing).
When you step into the `OS_Delay()`, you will arrive there:



If you set a breakpoint in one or both of our tasks, you will see that they continue execution after the given delay. You may open the watch window to display the ***embOS*** time variable OS_Time, which shows how much time has expired in the target system.

Now start the target CPU by “Debug -> Go” or press F5. The HP task will continue after the given delay of 50 ms:



3.2. Build your own application

To build your own application, you should start with a sample start project. This has the advantage, that all necessary files are included and all settings for the project are already done.

3.3. 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 sub folder 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 subfolder CPU_*\.
  It contains hardware dependent initialization code for ***embOS*** timer and optional UART for embOSView.
- **OS_Error.c** from subfolder Src\.
  It contains the ***embOS*** runtime error handler `OS_Error()`which is used in stack check or debug builds.
- One ***embOS*** **library** from the Lib\ subfolder
- **ncrt0.a30** from subfolder Src\.
  This is the startup code which is modified to be used with ***embOS***.
- **sect30.inc** from subfolder Src\.
  This is the interrupt vector table file which is setup to be used with ***embOS***.

When you decide to write your own startup code, please 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 except `OS_incDI()` are called.

3.4. Select a start project

embOS comes with one start project for an R8C23 (R5F21237) CPU. The start project was built and tested for standard CPUs. For various CPU variants there may be modifications required.

3.5. Add your own code

For your own code, you may add a new group to the project.
You should then modify or replace the main.c source file in the subfolder src\.

3.6. Change memory model or library mode

For your application you may have to choose an other data- / memory-model. 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.
Therefore you have to replace the ***embOS*** library in your project or target:

- Replace the library by modifying the linker settings.

Finally check project options about target CPU data / memory model settings and compiler settings according library mode used. Refer to chapter 4about the library naming conventions to select the correct library.

4. R8C and NC30 specifics

4.1. Memory models

embOS supports all memory models that RENESAS NC30 C-Compiler supports.
For R8C there is one memory models available:

Model	Code	Data
Near	far (20 bits always)	near (16 bits)

4.2. Available libraries

The files for R8C to use are:

Library type	Library	define
Release	RTOSR	OS_LIBMODE_R
Stack-check	RTOSS	OS_LIBMODE_S
Stack-check + Profiling	RTOSSP	OS_LIBMODE_SP
Debug	RTOSD	OS_LIBMODE_D
Debug + Profiling	RTOSDP	OS_LIBMODE_DP
Trace + Debug	RTOSDT	OS_LIBMODE_DT

When using RENESAS HEW, please check the following points:

- The memory model is set as option for your compiler
- One ***embOS*** library is added to your project (under Project Options | Linker settings)
- The appropriate define is set as compiler option for your project.

4.3. Distributed project files

The distribution of ***embOS*** contains one start project which is set up for the near memory model.

4.4. Startup file NCRT0.a30

embOS. comes with a modified startup file for R8C. Minor modifications are required; they are documented in this file.

4.5. Section and interrupt vector definition file SECT30.inc

This file was modified to export information about stack sizes. Also ***embOS*** interrupts are defined in this file. All modifications are documented in this file.

5. Stacks

5.1. Task stack for R8C

Every ***embOS*** task has to have its own stack. Task stacks can be located in any RAM memory location that can be used as stack by the R8C CPU.
As R8C CPUs have a 16bit stack pointer only, this may be any RAM located from 0x0000..0xFFFF.
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 R8C, this minimum stack size is about 42 bytes in the near memory model.

5.2. System stack for R8C

The system stack size required by ***embOS*** is about 40 bytes (65 bytes in. profiling builds) 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 also use the system-stack, the actual stack requirements depend on the application.
The stack used as system stack is the one defined at startup. Its size is defined as `__STACKSIZE __` in the nc_define.inc file.
A good value for the system stack is typically about 80 to 200 bytes.

5.3. Interrupt stack for R8C

The R8C CPU has been designed with multitasking in mind; it has 2 stack-pointers, the USP and the ISP. The U-Flag selects the active stack-pointer. During execution of a task or timer, the U-flag is set thereby selecting the user-stack-pointer. If an interrupt occurs, the R8C clears the U-flag and switches to the interrupt-stack-pointer automatically this way. The ISP is active during the entire ISR (interrupt service routine). This way, the interrupt does not use the stack of the task and the task-stack-size does not have to be increased for interrupt-routines. Additional stack-switching as for other CPUs is therefore not necessary for the R8C.
The stack used as interrupt stack is the one defined at startup. Its size is defined as `__ISTACKSIZE__` in the nc_define.inc file.

5.4. Reducing the stack size

The stack check libraries check the used stack of every task and the system and interrupt stack also. Using embOSView, the total size and used size of any stack can be examined. This may be used to analyze stack requirements and to reduce the stack sizes, if RAM space is a problem in your application.

6. Interrupts

6.1. What happens when an interrupt occurs?

- The CPU-core receives an interrupt request
- As soon as the interrupts are enabled and the processor interrupt priority level is below or equal to the interrupt priority level, the interrupt is executed
- the CPU switches to the Interrupt stack
- the CPU saves PC and flags on the stack
- the IPL is loaded with the priority of the interrupt
- the CPU jumps to the address specified in the vector table for the interrupt service routine (ISR)
- ISR : save registers
- ISR : user-defined functionality
- ISR : restore registers
- ISR: Execute REIT command, restoring PC, Flags and switching to User stack
- For details, please refer to the RENESAS users manual.

6.2. Defining interrupt handlers in "C"

Routines defined with the keywords `#pragma INTERRUPT` automatically save & restore the registers they modify and return with REIT.
For a detailed description on how to define an interrupt routine in "C", refer to the NC30 C-Compiler's user's guide.

Example

"Simple" interrupt-routine

```
#pragma INTERRUPT OS_ISR_tx
void OS_ISR_tx(void) {
  OS_EnterNestableInterrupt();  //  We will enable interrupts
  OS_OnTx();
  OS_LeaveNestableInterrupt();
}
```

6.3. Interrupt vector table

The interrupt vectors may be defined in “C” when using RENESAS HEW 4 and new NC30 compiler.
The distribution of ***embOS*** uses the old style of interrupt vector table definition using an assembly include file which is included in the startup file.
embOS comes with a prepared and modified section definition file sect30.inc which should be used and modified for your needs.

6.4. Interrupt-stack

Since the R8C CPUs have a separate stack pointer for interrupts, there is no need for explicit stack-switching in an interrupt routine. The routines `OS_EnterIntStack()` and `OS_LeaveIntStack()` are supplied for source compatibility to other processors only and have no functionality.

6.5. Fast interrupts with R8C

Instead of disabling interrupts when ***embOS*** does atomic operations, the interrupt level of the CPU is set to 4. Therefore all interrupts with level 5 or above can still be processed.
These interrupts are named *Fast interrupts*. You must not execute any ***embOS*** function from within a *fast interrupt* function.

6.6. Interrupt priorities

With introduction of *Fast interrupts*, interrupt priorities useable for interrupts using ***embOS*** API functions are limited.

- Any interrupt handler using ***embOS*** API functions has to run with interrupt priorities from 1 to 4. These ***embOS*** interrupt handlers have to start with `OS_EnterInterrupt()` or `OS_EnterNestableInterrupt()` and must end with `OS_LeaveInterrupt()` or `OS_LeaveNestableInterrupt()`.
- Any *Fast interrupt* (running at priorities from 5 to 7) must not call any ***embOS*** API function. Even `OS_EnterInterrupt()` and `OS_LeaveInterrupt()` must not be called.
- Interrupt handler running at low priorities (from 1 to 4) not calling any ***embOS*** API function are allowed, but must not re-enable interrupts!

The priority limit between *embOS* interrupts and Fast interrupts is fixed to 4 and can only be changed by recompiling *embOS* libraries!

7. STOP / WAIT Mode

Usage of the wait instruction is one possibility to save power consumption during idle times. If required, you may modify the `OS_Idle()` routine, which is part of the hardware dependent module RtosInit.c.

The stop-mode works without a problem; however the real-time operating system is halted during the execution of the stop-instruction if the timer that the scheduler uses is supplied from the internal clock. With external clock, the scheduler keeps working.

8. Technical data

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 values in the table are for the near memory model and release build library.

Short description	ROM [byte]	RAM [byte]
Kernel	approx.1670	27
Add. Task	---	19
Add. Semaphore	---	4
Add. Mailbox	---	11
Add. Timer	---	11
Power-management	---	---

9. Files shipped with *embOS* for NC30 compiler

embOS for R8C and NC30 compiler is shipped for compiler version 5.40 and projects for HEW version 4.
This version of ***embOS*** is located in Folder “embOS_R8C_HEW” and contains the following files:

Directory	File	Explanation
Start\	`Start_R8C23.hws`	Start workspace for HEW 4
Start\Start_MR8C23\	`*.*`	Project files for HEW
Start\INC\	`RTOS.h`	***embOS*** API header file. To be included in any file using ***embOS*** functions
Start\Lib	`*.lib`	***embOS*** libraries
Start\Src\	`main.c`	Frame program to serve as a start
Start\CPU_*\	`RTOSInit_*.c`	Hardware dependant functions used by ***embOS***
Start\Src	`OS_Error.c`	The ***embOS*** error handler, used called on runtime error occurrence in debug builds.
Start\ CPU_*\	`ncrt0.a30`	Startup file, modified for use with ***embOS***
Start\ CPU_*\	`sect30.inc`	Section definition file and interrupt vector table, modified for use with ***embOS***

embOSView and the manuals are found in the root directory of the distribution.

10. Index