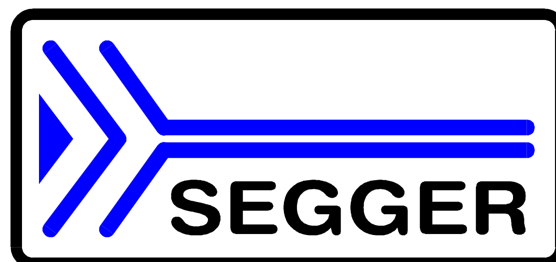
embOS

Real Time Operating System

CPU & Compiler specifics for
RENESAS M16C CPUs
and HEW workbench

Document Rev. 1



A product of SEGGER Microcontroller GmbH & Co. KG

www.segger.com

© 2008 SEGGER Microcontroller GmbH & Co. KG

Contents

Contents ... 3
1. About this document ... 4
    1.1. How to use this manual ... 4
2. Using *embOS* with RENESAS HEW ... 5
    2.1. Installation ... 5
    2.2. First steps ... 6
    2.3. The sample application Main.c ... 6
3. Using debugging tools to debug the application ... 8
    3.1. Debug the application using HEW M16C Simulator ... 8
    3.2. Using KD30 ROM Monitor ... 14
    3.3. Using PD30 / PC4701 / PC7501 in circuit emulator ... 17
    3.4. Using PD30Sim ... 18
4. Build your own application ... 19
    4.1. Required files for an *embOS* application ... 19
    4.2. Select a start project ... 19
    4.3. Add your own code ... 19
    4.4. Change memory model or library mode ... 19
5. M16C and NC30 specifics ... 21
    5.1. Memory models ... 21
    5.2. Available libraries ... 21
    5.3. Distributed project files ... 21
    5.4. M306N CPU specifics ... 21
    5.5. M16C/62P CPU specifics, PLL ... 22
    5.6. Startup file NCRT0.a30 ... 22
    5.7. Section and interrupt vector definition file SECT30.inc ... 22
6. Stacks ... 23
    6.1. Task stack for M16C ... 23
    6.2. System stack for M16C ... 23
    6.3. Interrupt stack for M16C ... 23
    6.4. Reducing the stack size ... 23
7. Interrupts ... 24
    7.1. What happens when an interrupt occurs? ... 24
    7.2. Defining interrupt handlers in "C" ... 24
    7.3. Interrupt vector table ... 24
    7.4. Interrupt-stack ... 24
    7.5. Fast interrupts with M16C ... 25
    7.6. Interrupt priorities ... 25
8. STOP / WAIT Mode ... 26
9. Technical data ... 27
    9.1. Memory requirements ... 27
10. Files shipped with *embOS* for NC30 compiler ... 27
11. Index ... 28

© 2008 SEGGER Microcontroller GmbH & Co. KG

1. About this document

This guide describes how to use ***embOS*** for M16C Real Time Operating System for the RENESAS M16C series of microcontroller using RENESAS NC30 compiler version 5.40 and HEW version 4.

1.1. How to use this manual

This manual describes all CPU and compiler specifics for ***embOS*** using M16C 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 M16C CPUs and NC30 compiler.

© 2008 SEGGER Microcontroller GmbH & Co. KG

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 M16C 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.

© 2008 SEGGER Microcontroller GmbH & Co. KG

2.2. First steps

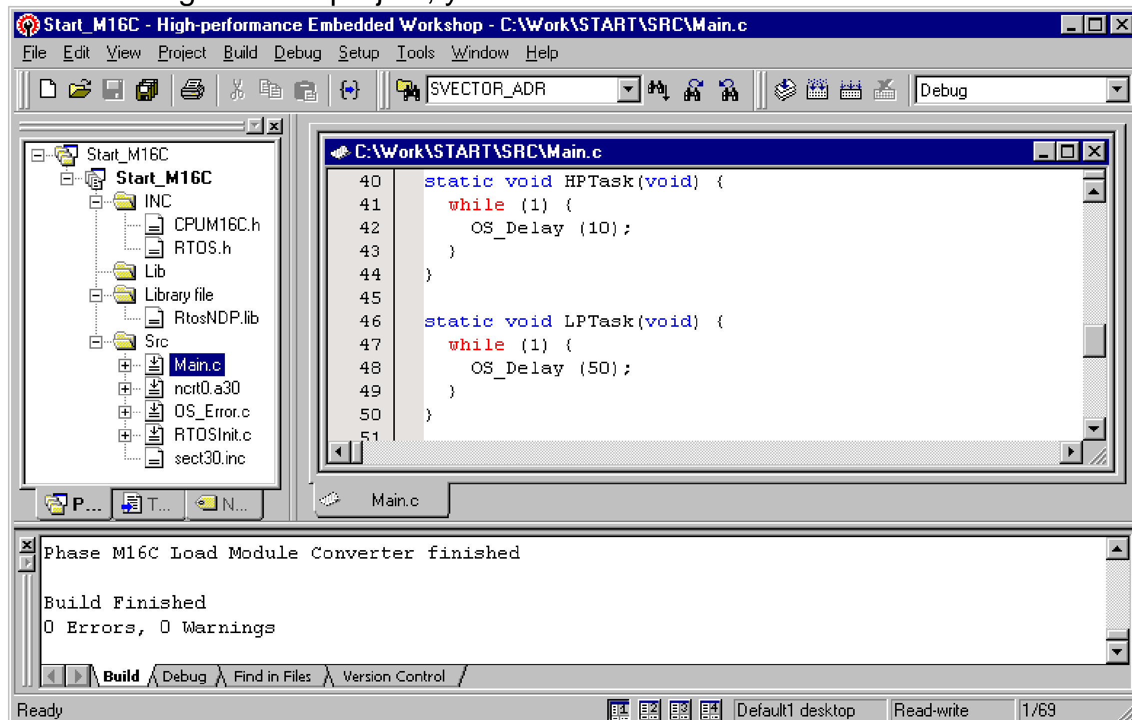
After installation of ***embOS*** (→ Installation) you are able to create your first multitasking application. You received a ready to go sample start project for M16C CPUs and it is a good idea to use this as a starting point of all your applications.

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_M16C_NC30_V540_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_M16C.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 startingpoint 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

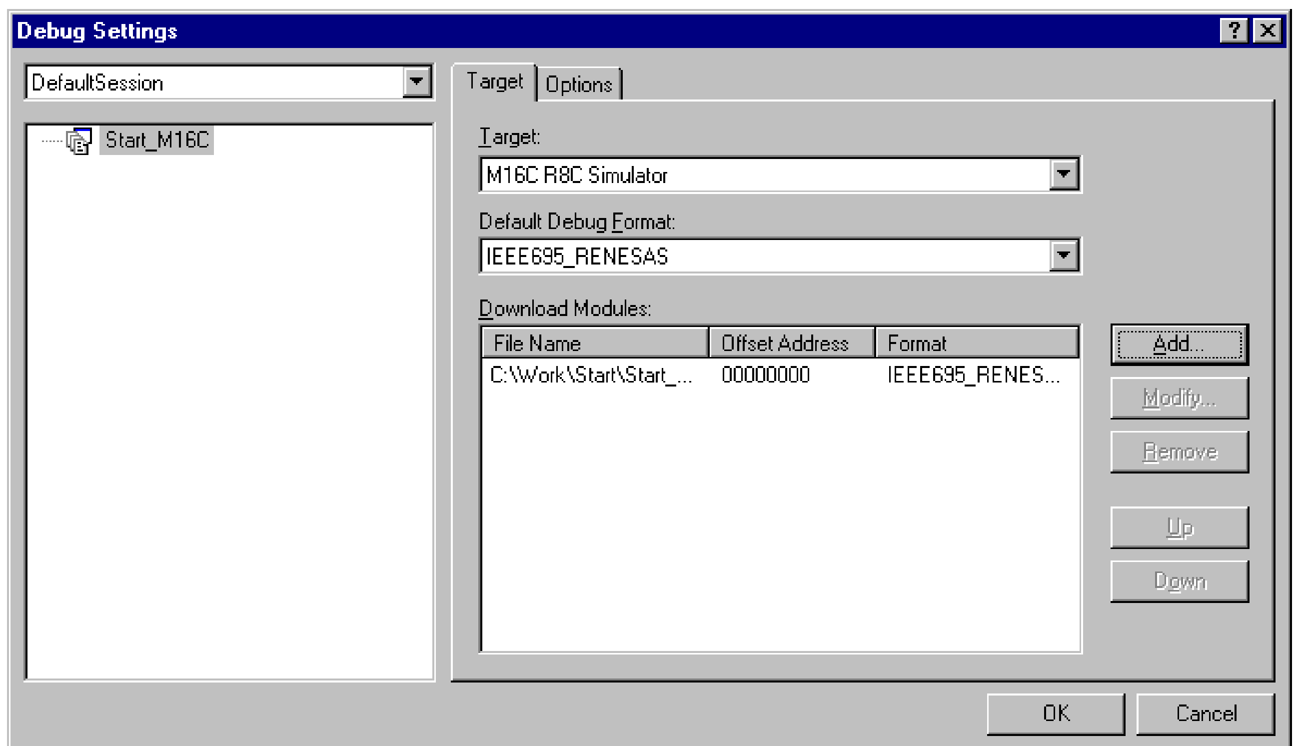
The *embOS* start project is configured to produce an “X30” output file which can directly be loaded into RENESAS in circuit emulator, the compatible simulator PD30Sim, may also be used with the ROM Monitor KD30 or even the HEW simulator.
The following chapter describes a sample session based on our sample application main.

3.1. Debug the application using HEW M16C Simulator

The easiest way to debug the start project is using the M16C Simulator which is included in Renesas HEW.
The distribution of embOS for M16C comes with a simple timer simulation script which can be used to simulate the embOS system timer.
After building the start project, the M16C Simulator can be selected as debugger by the following procedure:

- From The main menu choose “Debug -> Debug Settings”.
- In the “Debug Settings” dialog, select Target: M16C Simulator.
- In the “Debug Settings” dialog, select Default Debug Format: IEEE695_RENESAS.
  In the “Debug Settings” dialog, add the generated output of the start project to the list “Download Modules”. The output file of the sample start project for M16C is “Start\Start_M16C\debug\Start_M16C.x30”

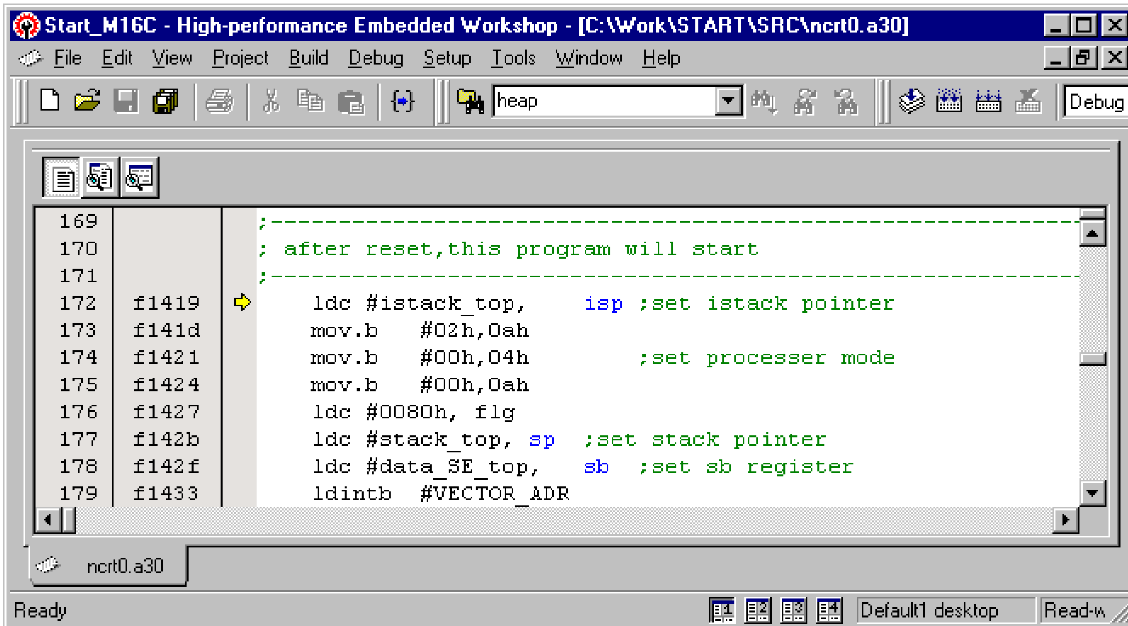
The dialog should look like follows:



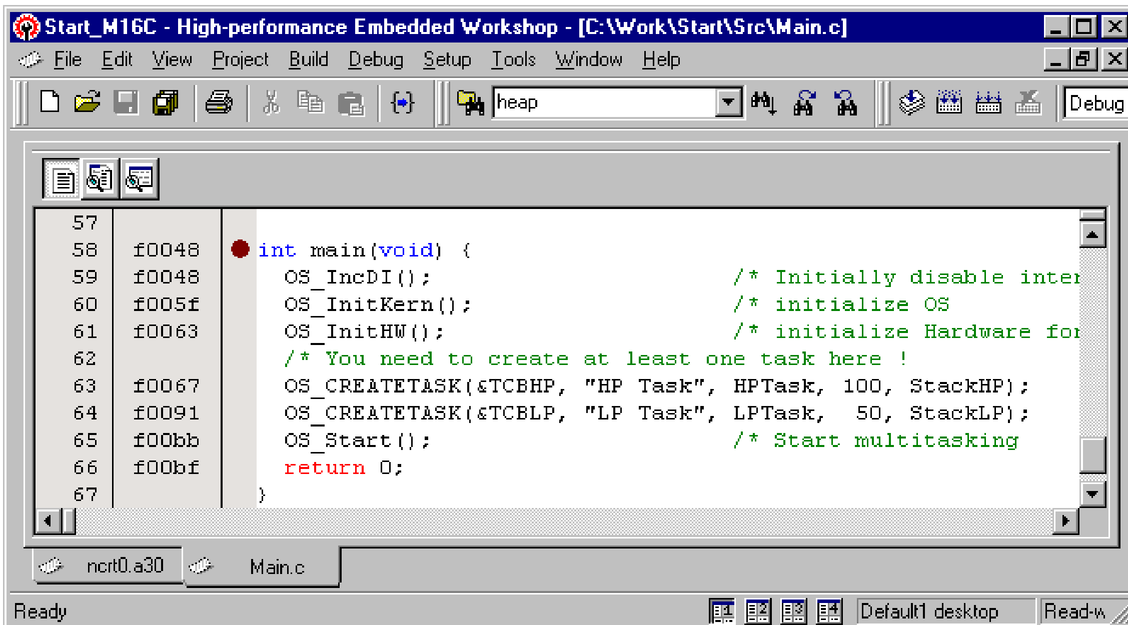
You will then have to setup the M16C Simulator, if not already done.
When you choose “Debug -> Connect” from the main menu, The “Init (M16C Simulator)” dialog appears.

You have to define a CPU in the “MCU” tab. To select a CPU, click the “Re-fer…” button.
For the Start_M16C sample, select the M16C6x.mcu file.
Depending on other options, the debugger then automatically loads the target file.
To be sure the right target is loaded, you may choose “Debug -> Download Modules” from the main menu and load the Start_M16C.x30 file.

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



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

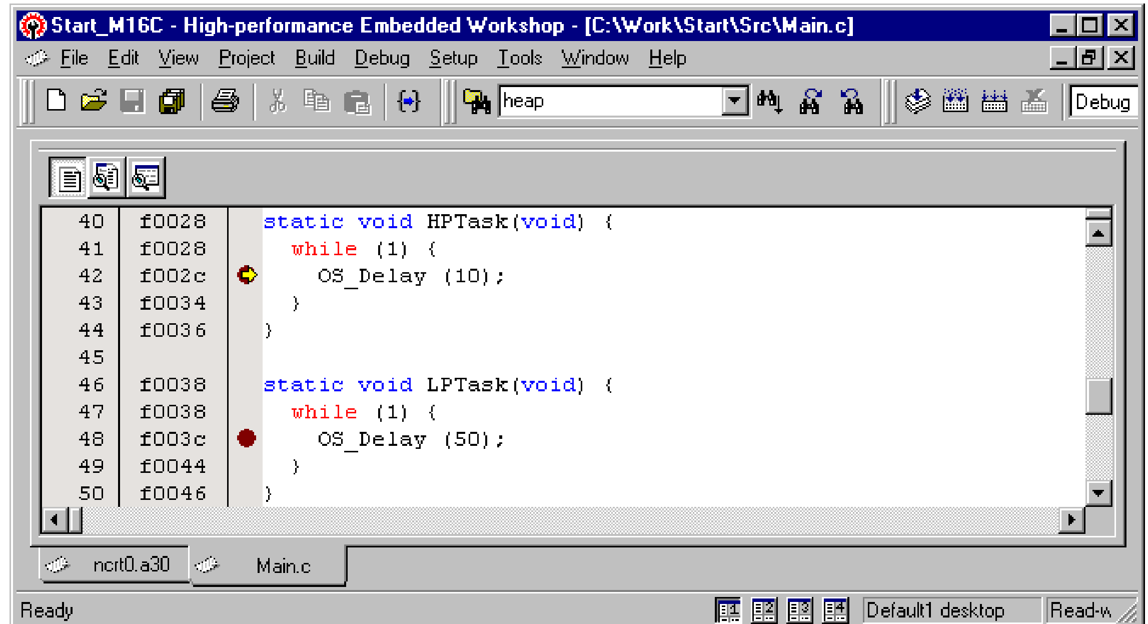


You may now step through the sample application.

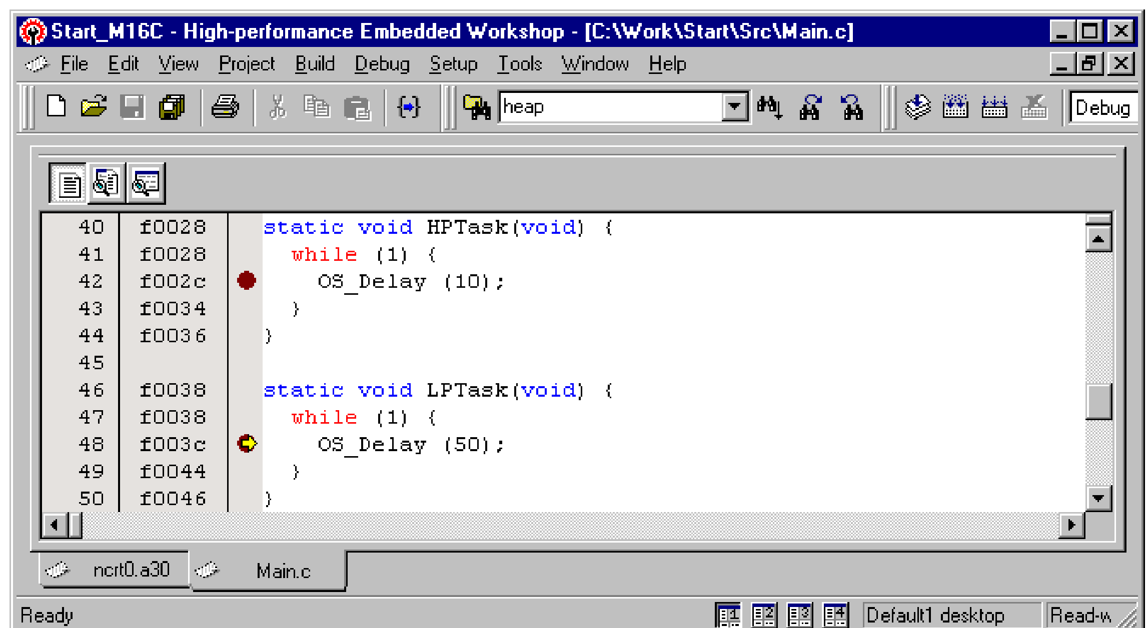
- `OS_IncDI()` Initially disables interrupts and prevents re-enabling them in `OS_InitKern()`.
- `OS_InitKern()` is part of the ***embOS*** Library; you can therefore only step into it in disassembly mode. It initializes the relevant OS-Variables.
- `OS_InitHW()` is part of RTOSInit.c and therefore part 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.

© 2008 SEGGER Microcontroller GmbH & Co. KG

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 maust not step over `OS_Start()`). In our small start program, `HPTask()` is the highest priority task and is therefore active:

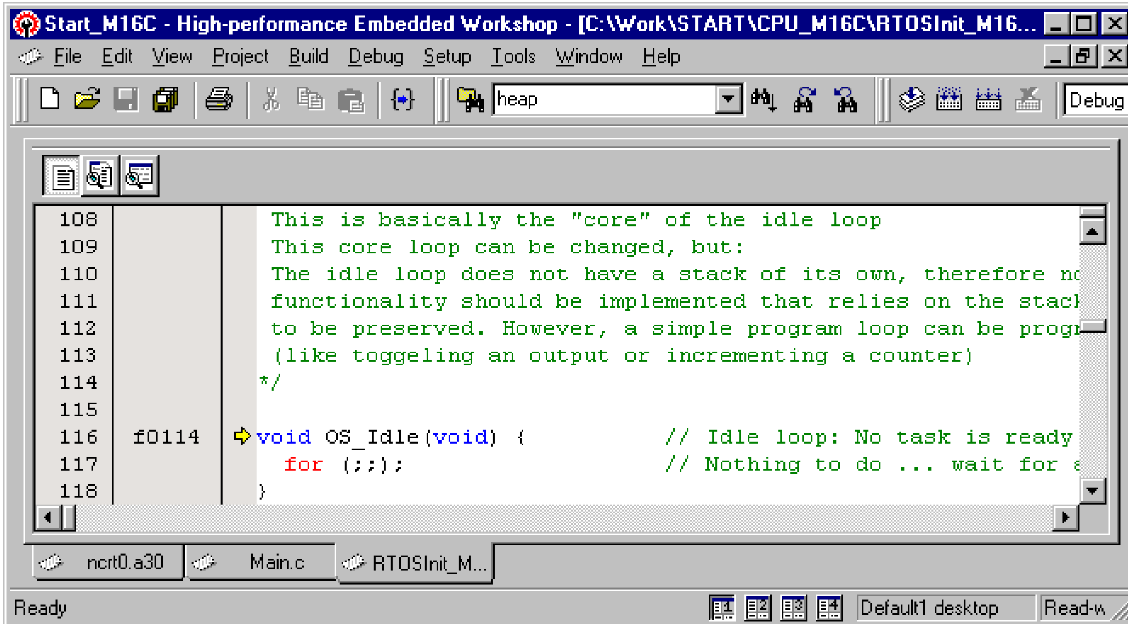


You should set a breakpoint in every task, as shown above. If you continue stepping, you will arrive in the task with the second highest priority:



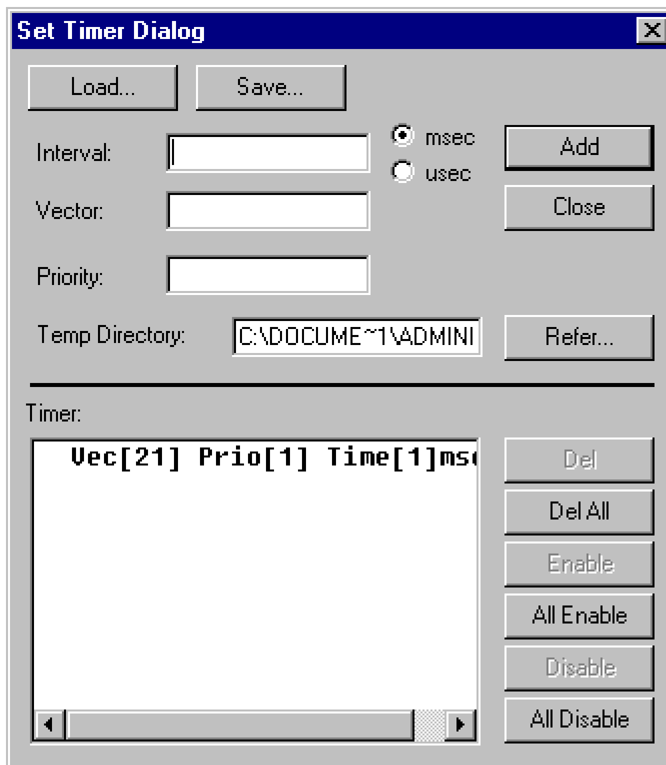
Continuing to step through the program, there is no other task ready for execution. ***embOS*** will therefore start the idle-loop, which is an endless loop which is always executed if there is nothing else to do (no task is ready, no interrupt routine or timer executing).
When you step into the `OS_Delay()`,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

© 2008 SEGGER Microcontroller GmbH & Co. KG



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, it is time to start the embOS timer simulation script:

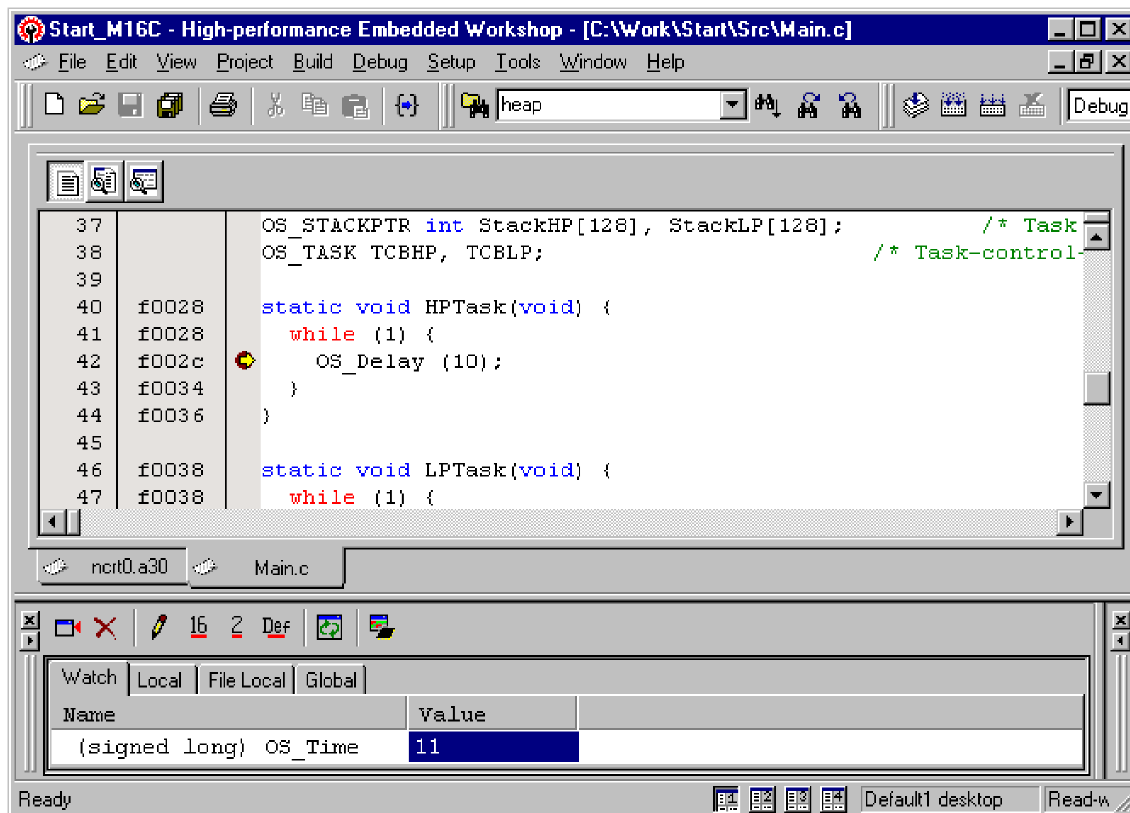
- From main menu choose View -> CPU -> I/O Timing Setting.
  The I/O Timing setting window opens.
- Select the Timer-symbol.
  The Set Timer Dialog opens
- Choose "Load..."
- Select the file Start\embOS_Timer.stm which is delivered with embOS.



The simple timer script generates interrupt 21 which is used for embOS timer

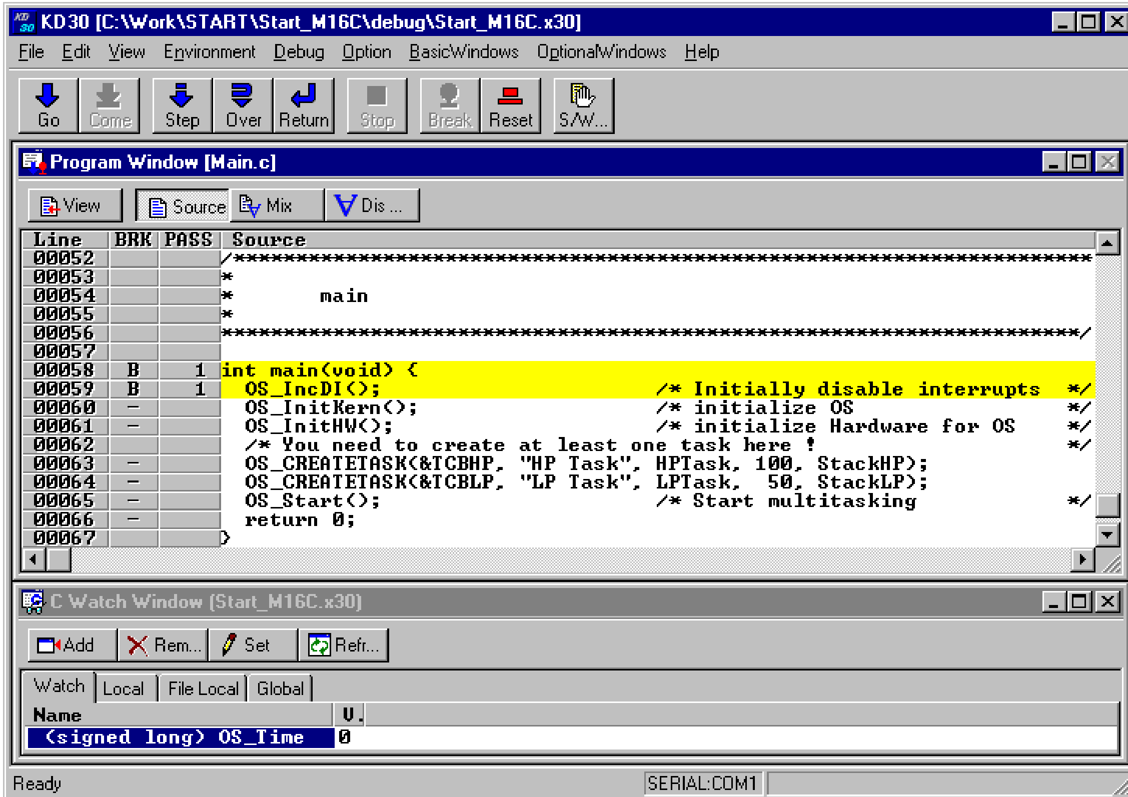
- Close the Set Timer dialog.
- Do not close the I/O Timing settings window, because the timer only runs as long as this window is open.

Now start the target CPU by “Debug -> Go” or press F5. The HP task will continue after the given delay of 10 ms:



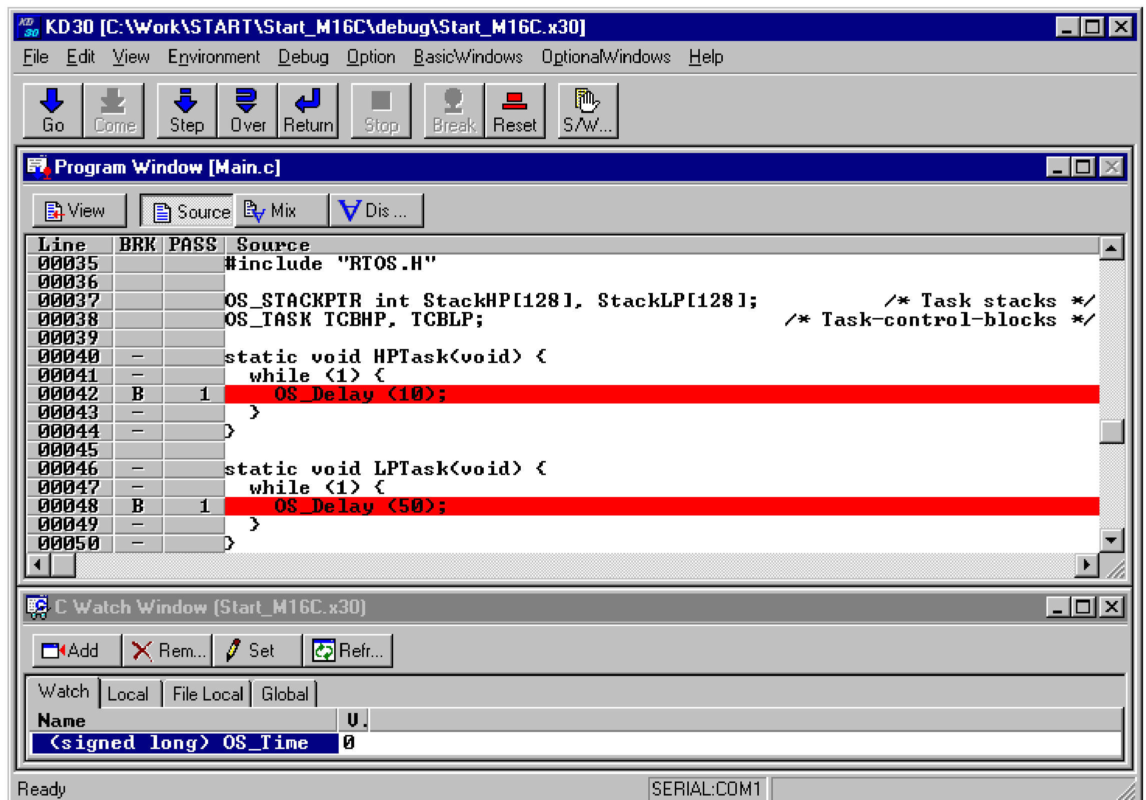
3.2. Using KD30 ROM Monitor

The distribution of ***embOS*** for M16C is prepared for usage of KD30 ROM monitor. Therefore two interrupt vectors are defined for UART1 per default.
This is done in 'sect30.inc'. Please check whether these vectors fit to your version of ROM monitor.
After starting KD30 debugger and downloading the Module "Start_M16C.x30" , you will usually see the startup code.
You should set a breakpoint at main and then press "Go" to arrive at main().

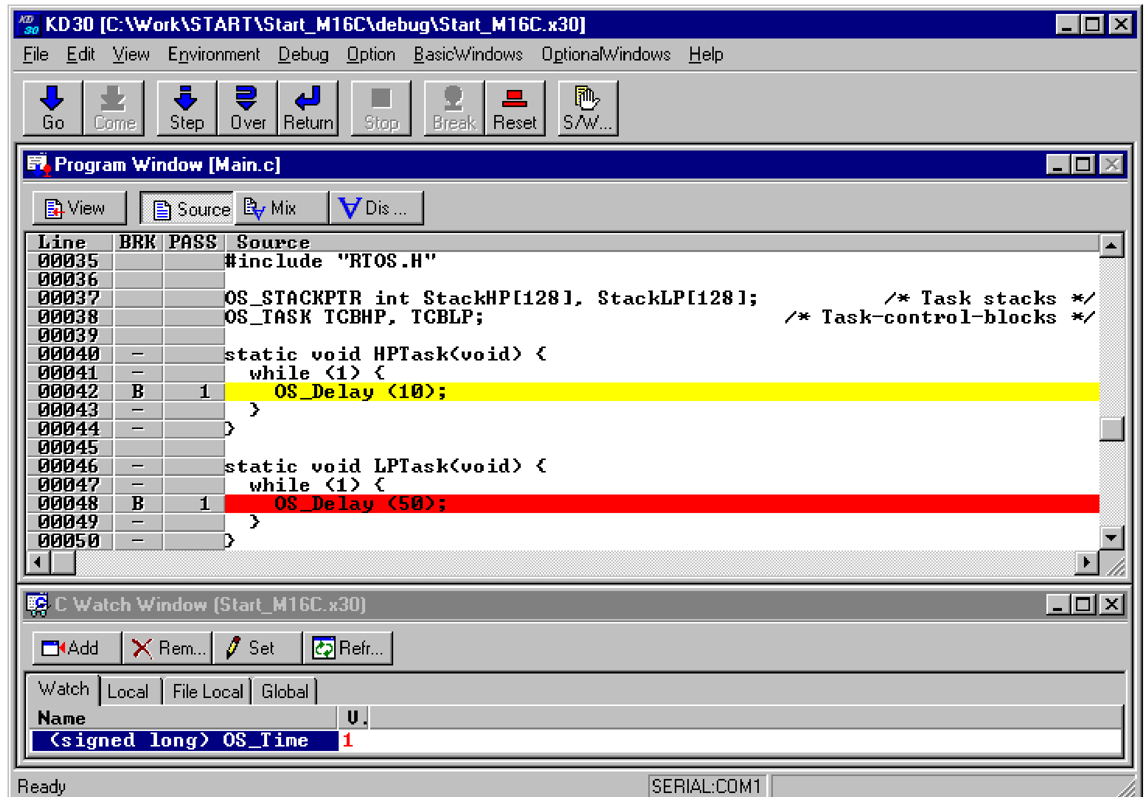


`OS_IncDI()` initially disables interrupts and informs ***embOS*** that interrupts should not be re-enabled during ***embOS*** function calls.
`OS_InitKern()` is part of the ***embOS*** Library; you can therefore only step into it in disassembly mode. It initializes the relevant OS-Variables and would enable interrupts if `OS_IncDI()` was not called before.
`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_COM_Init()` in `OS_InitHW()` is optional. It is required if embOSView shall be used. In this case it initializes the UART used for communication.
`OS_Start()` should be the last line in main, since it starts multitasking and does not return. Interrupts are re-enabled during execution of `OS_Start()`.

Before you step into `OS_Start()`, you should remove the breakpoint at `main()` and set two breakpoints in the two tasks as shown below:

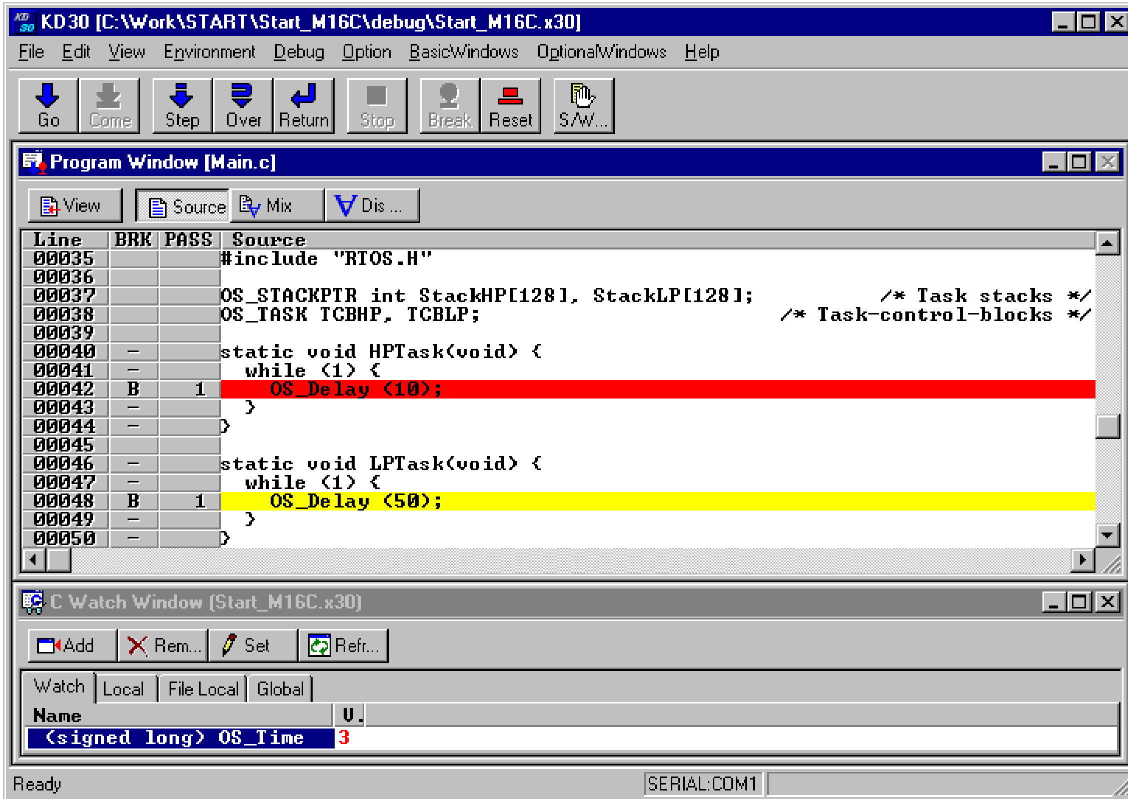


When you step over `OS_Start()`, the next line executed is already in the highest priority task created. (you may also step into `OS_Start()`, then stepping through the task switching process in disassembly mode). In our small start program, `HPTask()` is the highest priority task and is therefore active.

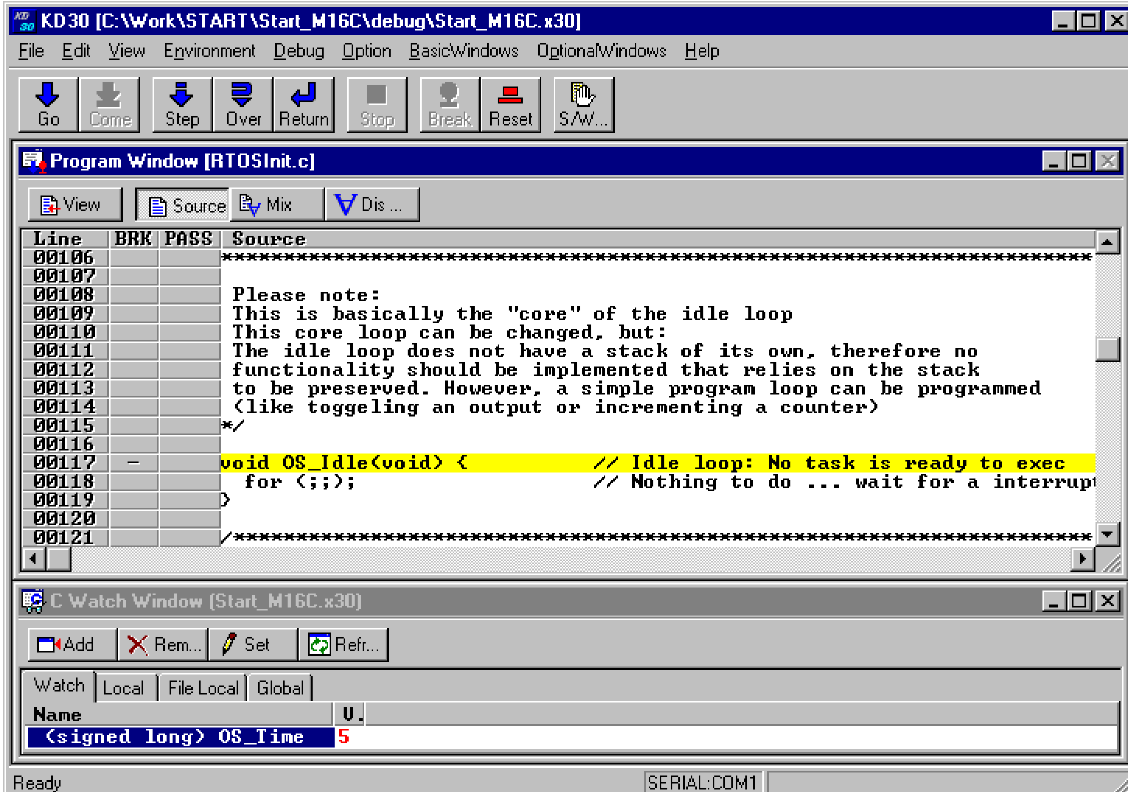


If you continue stepping, you will arrive in the task with the lower priority:

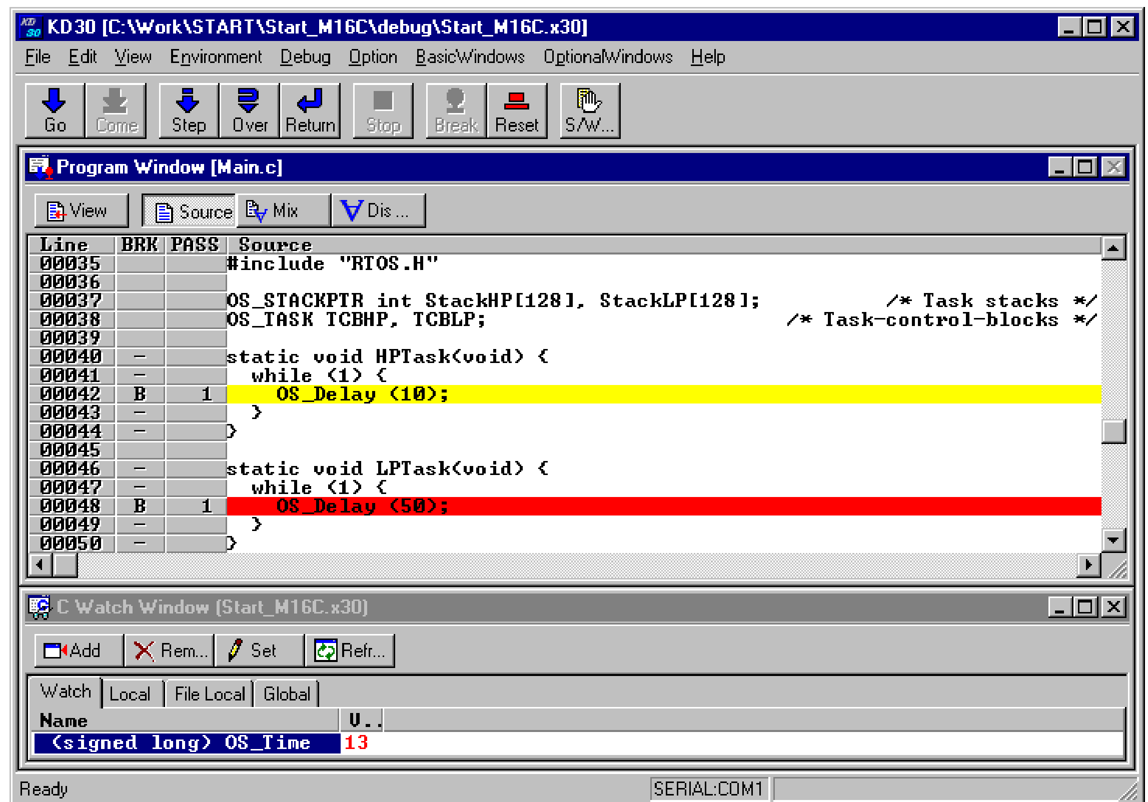
© 2008 SEGGER Microcontroller GmbH & Co. KG



Continuing to step through the program, there is no other task ready for execution. ***embOS*** will suspend `LPTask()` and switch to the idle-loop, which is always executed if there is nothing else to do (no task is ready, no interrupt routine or timer executing). `OS_Idle()` is found in RTOSInit.c:



If you set a breakpoint in one or both of our tasks, you will see that they continue execution after the given delay.
Coming from `OS_Idle()`, you should execute the 'Go' command:



Please note:
As the ROM monitor does not stop the timer when it reaches a breakpoint, the timer continues counting and produces an interrupt as soon as the next step is executed. This results in extra counts of the time variable OS_Time which is shown in the watch window.

Hints for debugging an *embOS* application with KD30 ROM monitor:

When ROM monitor stopped at a breakpoint, it may happen, that any interrupt activates a task switch while stepping through the program, as interrupts are enabled during stepping. This task switch can not be handled by KD30 and KD30 crashes.
To overcome this problem, you should open the register window and set the interrupt priority register of the CPU (IPL) to 6 immediately after the breakpoint was reached. This enables stepping without any task switches, as all *embOS* interrupts normally run with lower priorities.
When using M16C/62P CPUs, the ROM monitor may have set up the PLL of the CPU. This may cause problems during the call of `OS_InitHW()`, because standard initialisation of timer hardware does not initialise the PLL and sets CPUs clock mode register to a state which might not work with PLL enabled.
This may require modification of `OS_InitHW()`.

3.3. Using PD30 / PC4701 / PC7501 in circuit emulator

You may use the same output file “Start_M16C.x30” with an in circuit emulator. Debugging our sample application should look similar to the sample described above.

3.4. Using PD30Sim

The PD30 simulator may be used to examine or debug your application. The peripherals of M16C CPUs are not simulated automatically. To simulate interrupts, you have to load an I/O script file.
The ***embOS*** distribution contains the I/O script file ‘`PD30_SimInt21.scr`’ which simulates timer A0 interrupt, which is normally used as ***embOS*** timer interrupt.
The interrupt simulation should be started after the call of `OS_InitHW()`.
To start the simulation, proceed as follows:

- Open the I/O script window by menu ‘Optional Window | IO Window’.
- From the menu in the IO Window, choose ‘Load’
- Open the I/O script file ‘PD30_SimInt21.scr’ found in the start directory

Please note:

The timer interrupt simulation starts immediately, but stops as soon as the IO Window is closed. Therefore the IO Window may be minimized, but must not be closed during the debugging session.

4. 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.

4.1. 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.

4.2. Select a start project

embOS comes with one start project for an M16C60 CPU and one start project for an M16C62P CPU. The start projects were built and tested for standard CPUs. For various CPU variants there may be modifications required.

4.3. 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\.

4.4. 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.

© 2008 SEGGER Microcontroller GmbH & Co. KG

Finally check project options about target CPU data / memory model settings and compiler settings according library mode used. Refer to chapter 5 about the library naming conventions to select the correct library.

© 2008 SEGGER Microcontroller GmbH & Co. KG

5. M16C and NC30 specifics

5.1. Memory models

embOS supports all memory models that RENESAS NC30 C-Compiler supports.
For M16C there are two memory models available:

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

5.2. Available libraries

The files for M16C to use are:

Memorymodel	Library type	Library	define
Near	Release	RTOSNR	OS_LIBMODE_R
Near	Stack-check	RTOSNS	OS_LIBMODE_S
Near	Stack-check + Profiling	RTOSNSP	OS_LIBMODE_SP
Near	Debug	RTOSND	OS_LIBMODE_D
Near	Debug + Profiling	RTOSNDP	OS_LIBMODE_DP
Near	Trace + Debug	RTOSNDT	OS_LIBMODE_DT
Far	Release	RTOSFR	OS_LIBMODE_R
Far	Stack-check	RTOSFS	OS_LIBMODE_S
Far	Stack-check + Profiling	RTOSFSP	OS_LIBMODE_SP
Far	Debug	RTOSFD	OS_LIBMODE_D
Far	Debug + Profiling	RTOSFDP	OS_LIBMODE_DP
Far	Trace + Debug	RTOSFDT	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.

5.3. Distributed project files

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

5.4. M306N CPU specifics

The M16C/6N group of CPUs require modification affecting RTOSInit.c. The hardware initialization routines and default settings in RTOSInit.c were designed for M16C/62 CPUs.
M306N CPUs have additional prescalers that are activated per default after reset and divide the peripheral clock for timer and UART by two.
This results in wrong settings for ***embOS*** timer tick and baudrate for UART used for embOSView.

© 2008 SEGGER Microcontroller GmbH & Co. KG

There are different solutions to correct these settings.

As far as possible, you should not modify RTOSInit.c, as this has the disadvantage, that this modifications have to be tracked, when you update to a newer version of ***embOS***.

You may reprogram the Peripheral function clock select register (PCLKR) at address 0x025E to disable the prescaler for timer and UART, before calling `OS_InitHW()`. This could be done during your own target specific hardware initialization. The protection register bit 0 has to be set to enable modification of PCLKR
When PCLKR is left unchanged (reset value = 0x00), CPUs internal timer A0 and UART clock is derived from CPU clock divided by two.
If you do not want to disable (reprogram) the prescaler for UART or timer, you may define different values for `OS_PCLK_TIMER` and `OS_PCLK_UART` as compiler / project option without any changes in RTOSInit.c
`OS_PCLK_TIMER` is the frequency of CPUs internal peripheral clock used for the timer. Calculations of timer reload value is derived from this define. Without modification or override, it is defined to OS_FSYS.
`OS_PCLK_UART` is the frequency of CPUs internal peripheral clock used for UARTs. Calculations of baudrate generator value is derived from this define. Without modification or override, it is defined to OS_FSYS.

5.5. M16C/62P CPU specifics, PLL

M16C/62P CPUs come with a built in PLL.
The initialization routine `OS_InitHW()` for timer initialization is written for generic M16C CPUs and can also be used for M16C/62P CPUs.
If you want to use the PLL, you will have to modify the initialization sequence for CPU clock mode setting in `OS_InitHW()`.
You might also have to modify the clock mode initialization when you decide to use the KD30 ROM monitor for debugging. If the monitor is set up to initialize the PLL, the system may stop working when the CPU clock mode is modified during `OS_InitHW()`.

5.6. Startup file NCRT0.a30

embOS. comes with a modified startup file for M16C. Minor modifications are required; they are documented in this file.

5.7. 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.

© 2008 SEGGER Microcontroller GmbH & Co. KG

6. Stacks

6.1. Task stack for M16C

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 M16C CPU.
As M16C 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 M16C, this minimum stack size is about 42 bytes in the near memory model.

6.2. System stack for M16C

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 ncrt0.a30 start up file.
A good value for the system stack is typically about 80 to 200 bytes.

6.3. Interrupt stack for M16C

The M16C 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 M16C 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 M16CC.
The stack used as interrupt stack is the one defined at startup. Its size is defined as `ISTACKSIZE` in the ncrt0.a30 start up file.

6.4. 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.

© 2008 SEGGER Microcontroller GmbH & Co. KG

7. Interrupts

7.1. 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.

7.2. 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();
}
```

7.3. 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.

7.4. Interrupt-stack

Since the M16C 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.

© 2008 SEGGER Microcontroller GmbH & Co. KG

7.5. Fast interrupts with M16C

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.

7.6. 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!

© 2008 SEGGER Microcontroller GmbH & Co. KG

8. 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.

© 2008 SEGGER Microcontroller GmbH & Co. KG

9. Technical data

9.1. 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	---	---

10. Files shipped with *embOS* for NC30 compiler

embOS for M16C 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_M16C_NC30_V540” and contains the following files:

Directory	File	Explanation
Start\	`Start_M16C.hws`	Start workspace for HEW 4
Start\	`PD30_SimInt21.scr`	IO script file for ***embOS*** timer interrupt simulation under PD30Sim
Start\Start_M16c\	`*.*`	Project files for HEW
Start\INC\	`RTOS.h`	***embOS*** API header file. To be included in any file using ***embOS*** functions
Start\INC\	`CPUM16C.h`	SFR definition file for M16C CPU
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\Src\	`ncrt0.a30`	Startup file, modified for use with ***embOS***
Start\Src\	`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.

11. Index

F
Fast interrupt 25
I
Installation 5
Interrupt priority 25
Interrupt stack 23
Interrupt vector table.................... 24
Interrupt, fast................................ 25
Interrupts...................................... 24
Interrupt-stack.............................. 24
ISTACKSIZE............................... 23
K
KD30 14
M
M16C/62P.................................... 22
M306N...21
Memory models............................21
Memory requirements 27
N
NCRT0.a30 22
O
OS_Error() 19
OS_PCLK_UART........................22
OS_PLCK_TIMER......................22
P
PD30 17
PD30Sim......................................18
PLL 22
R
ROM Monitor 14
S
SECT30.inc 22
Stacks 23
Stacks, interrupt stack...................23
Stacks, system stack......................23
Stacks, task stacks 23
STACKSIZE 23
Startup file....................................22
Stop-mode 26
System stack 23
T
Task stacks 23
Technical data...............................27
W
Wait-mode....................................26

© 2008 SEGGER Microcontroller GmbH & Co. KG