Micriμm

Empowering Embedded Systems

μC/OS-II

μC/Probe

and the
NXP LPC17xx Processors
on the IAR LPC1768-SK development Board

Application Note

AN-1080

About Micriμm

Micriμm provides high-quality embedded software components in the industry by way of engineer-friendly source code, unsurpassed documentation, and customer support. The company's world-renowned real-time operating system, the Micriμm **μC/OS-II**, features the highest-quality source code available for today's embedded market. Micriμm delivers to the embedded marketplace a full portfolio of embedded software components that complement **μC/OS-II**. A TCP/IP stack, USB stack, CAN stack, File System (FS), Graphical User Interface (GUI), as well as many other high quality embedded components. Micriμm's products consistently shorten time-to-market throughout all product development cycles. For additional information on Micriμm, please visit www.micrium.com.

About μC/OS-II

Thank you for your interest in **μC/OS-II**. **μC/OS-II** is a preemptive, real-time, multitasking kernel. **μC/OS-II** has been ported to over 45 different CPU architectures and now, has been ported to the NXP LPC17xx processors

μC/OS-II is small yet provides all the services you would expect from an RTOS: task management, time and timer management, semaphore and mutex, message mailboxes and queues, event flags a much more.

You will find that **μC/OS-II** delivers on all your expectations and you will be pleased by its ease of use.

Licensing

μC/OS-II is provided in source form for **FREE** evaluation, for educational use or for peaceful research. If you plan on using **μC/OS-II** in a commercial product you need to contact Micriμm to properly license its use in your product. We provide ALL the source code with this application note for your convenience and to help you experience **μC/OS-II**. The fact that the source is provided **DOES NOT** mean that you can use it without paying a licensing fee. Please help us continue to provide the Embedded community with the finest software available. Your honesty is greatly appreciated.

About μC/Probe

μC/Probe is a Windows application that allows a user to display the value (at run-time) of virtually any variable or memory location on a connected embedded target. The user simply populates **μC/Probe**'s graphical environment with gauges, tables, graphs, and other components, and associates each of these with a variable or memory location. Once the application is loaded onto the target, the user can begin **μC/Probe**'s data collection, which will update the screen with variable values fetched from the target.

μC/Probe retrieves the values of global variables from a connected embedded target and displays the values in an engineer-friendly format. The supported data-types are: booleans, integers, floats and ASCII strings.

μC/Probe can have any number of 'data screens' where these variables are displayed. This allows to logically group different 'views' into a product.

A 30-day trial version of **μC/Probe** is available on the Micriμm website:

http://www.micrium.com/products/probe/probe.html

Manual Version

If you find any errors in this document, please inform us and we will make the appropriate corrections for future releases.

Version	Date	By	Description
V.1.00	2009/09/19	FT	Initial version.

Software Versions

This document may or may not have been downloaded as part of an executable file, *Micrium-NXP-uCOS-II-LPC1768-SK.exe*, containing the code and projects described here. If so, then the versions of the Micriμm software modules in the table below would be included. In either case, the software port described in this document uses the module versions in the table below

Module	Version	Comment
μC/OS-II	V2.89	
μC/Probe	V2.3	

See Also

In addition to the **μC/OS-II**, **μC/FS**, **μC/USB-Device**, **μC/USB-Host**, **μC/USB-OTG**, **μC/TCP-IP** have been ported to the LPC17xx processors.

Document Conventions

Numbers and Number Bases

- Hexadecimal numbers are preceded by the "0x" prefix and displayed in a monospaced font. Example: `0xFF886633`.
- Binary numbers are followed by the suffix "b"; for longer numbers, groups of four digits are separated with a space. These are also displayed in a monospaced font. Example: `0101 1010 0011 1100b`.
- Other numbers in the document are decimal. These are displayed in the proportional font prevailing where the number is used.

Typographical Conventions

- Hexadecimal and binary numbers are displayed in a monospaced font.
- Code excerpts, variable names, and function names are displayed in a monospaced font. Functions names are always followed by empty parentheses (e.g., `OS_Start()`). Array names are always followed by empty square brackets (e.g., `BSP_Vector_Array[]`).
- File and directory names are always displayed in an italicized serif font. Example: */Micrium/Sofware/uCOS-II/Source/*.
- A bold style may be layered on any of the preceding conventions—or in ordinary text—to more strongly emphasize a particular detail.
- Any other text is displayed in a sans-serif font.

Table of Contents

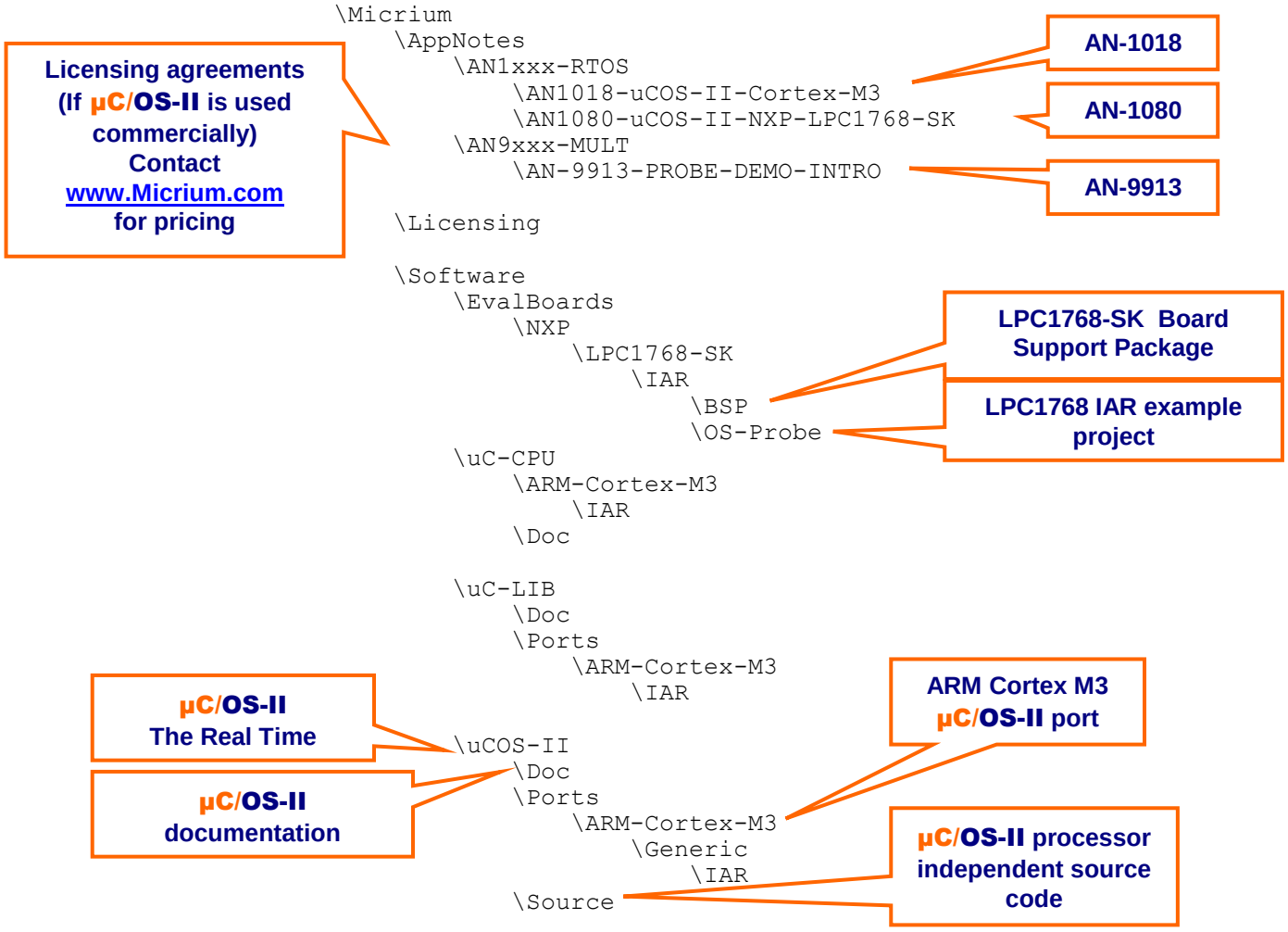
1. Getting Started.

The following sections step trough the prerequisites for using the demonstration application described in this document, ***AN-1080.*** First, installation of software and the setup of the hardware will be outlined. Second, the use and setup of the IAR embedded Workbench. Thirdly, the steps to build the projects and load the application onto the board trough JTAG will be described. Lastly, instructions will be provided for using the example application.

1.01 Installing the Micirum Software

The source code for **μC/OS-II** is provided in source form along with IAR Embedded Workbench for ARM project files that allow you to run **μC/OS-II** on the IAR LPC1768-SK development board. To install the software, simply run the self-extracting executable. ***Micrium-NXP-uCOS-II-LPC1768-SK.exe.***

You will be prompted to accept the simple terms of the licensing agreement. If you answer 'Yes', the software will be installed on your PC under the ***\Micrium*** directory from the root as shown in Figure 1-1



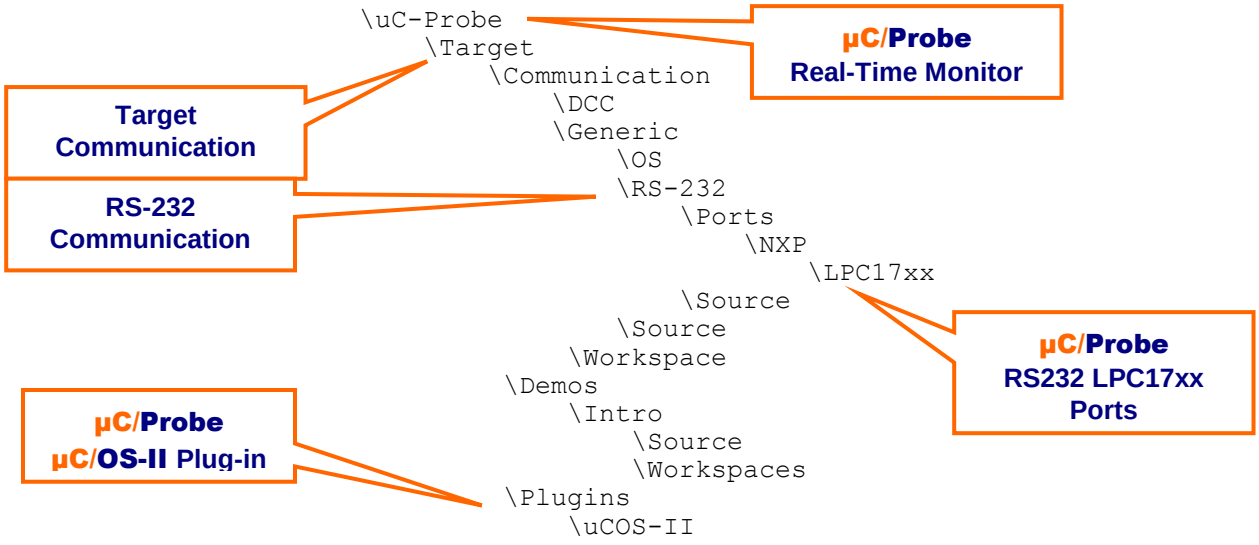


Figure 1-1. Directory Structure

1.02 Setting up the Hardware

The processor can be programmed and debugged trough the 20-pin JTAG port using a JTAG emulator, such as J-Link.

The board can be power up from a standard 5v DC converter, J-Link or the USB connector.
The Power select jumper (PWR_SEL) will determine the power supply used.

To use **μC/Probe** with the LPC1768-SK, download and install the trial version of the program from the Micrium website as discussed in section 5. After programming your target with one of the included projects, connect a RS-232 cable between the board and your PC, configure RS-232 options, and start running the program.

1.03 Opening the Examples Projects

1.03.01 IAR Example Project

To view the IAR example project, start an instance of IAR Embedded Workbench, and open:

- ***LPC1768--OS-Probe.ewp***, located in
  /Micrium/Software/EvalBoards/NXP/LPC1768-SK /IAR/OS-Probe folder.

To do this, use the *Add Existing Project...* menu command under the *Project* menu:

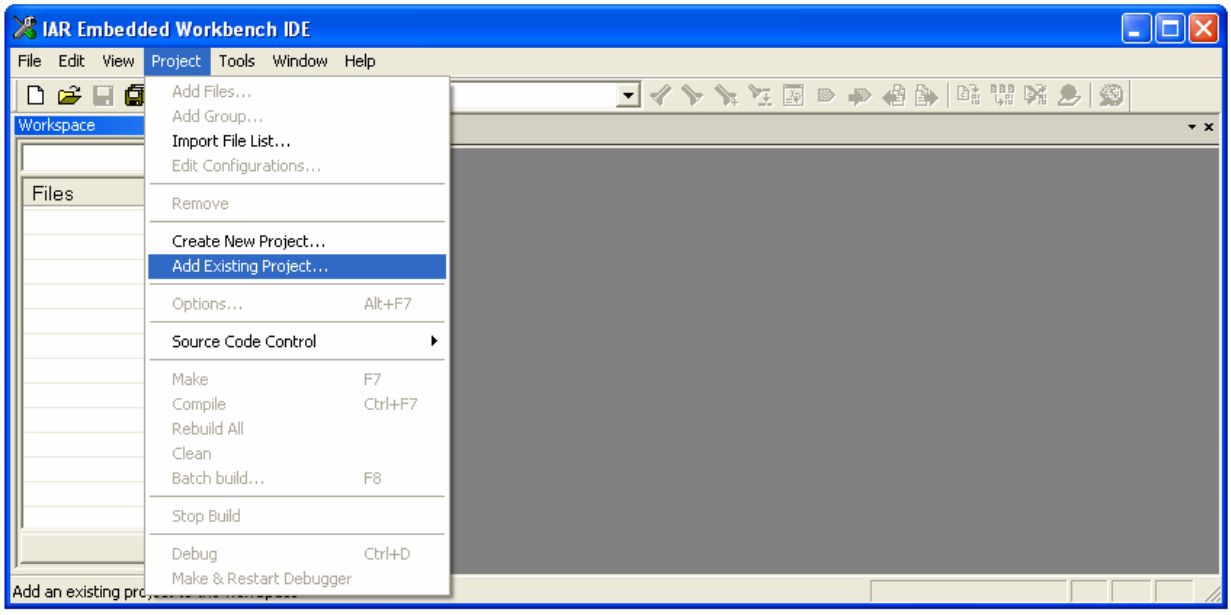


Figure 1-2. IAR EW. Opening an existing project

IAR EWARM Versions

Be certain to open the proper project for your version of EWARM. The NXP LPC1768 examples project was built using **EWARM ver. 5.4**

1.03.02 IAR μC/OS-II Kernel Awareness.

The **μC/OS-II** Kernel Awareness plug-in will allow you to examine information about system objects while using the C-Spy debugger. To gain access to this feature, enable the plug-in by right-clicking on the project name in the work space browser and choosing *Options…* Then, select the "Debugger" entry in the list box, and the "Plug-in" tab pane. Find the **μC/OS-II** entry in the list and, finally, select the check box beside the entry. Make sure you select the correct plug-in for the correct version of **μC/OS-II.**

- "uC/OS-II for version 2.86 and earlier" for **μC/OS-II** version 2.86 and earlier
- "uC/OS-II" for **μC/OS-II** version 2.87 and above.

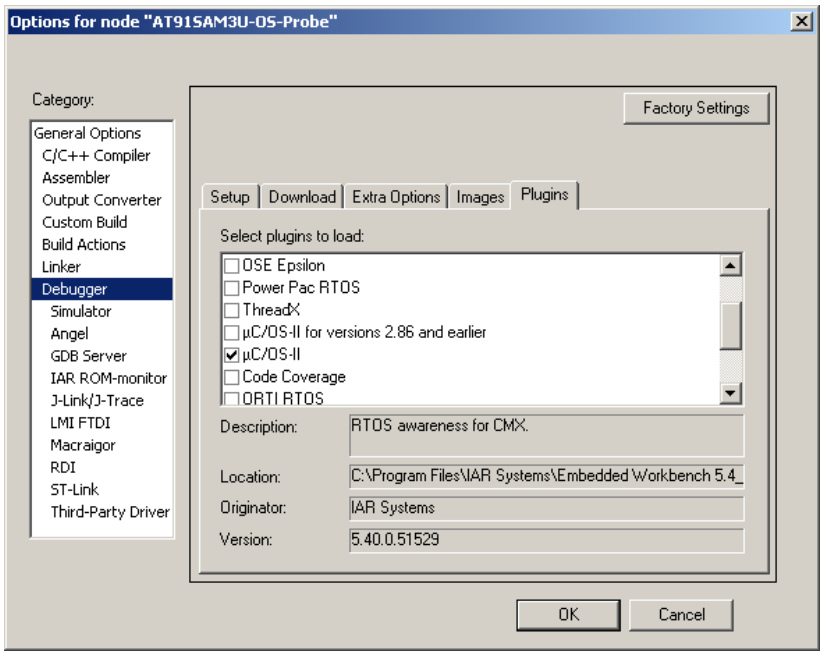


Figure 1-3. Enabling the μC/OS-II Kernel Awareness Plug-In

1.03.03 IAR Project Options

The IAR project configurations allow you to compile, link and load the software in different ways to the target. The following configuration is available in the IAR project.

- FLASH: This project option is configured to load the code into the Internal 256Kb Internal Flash.

1.04 Running the Example Applications

The example project includes a basic demonstration of **μC/OS-II** and **μC/Probe.** The evaluation board components are labeled in the figure 1-4

Once the program is loaded onto the target, the LEDs will start blinking.

The system state will be output to the color LCD display, the joystick (toggle left/right) can be used to move the output to a new item.

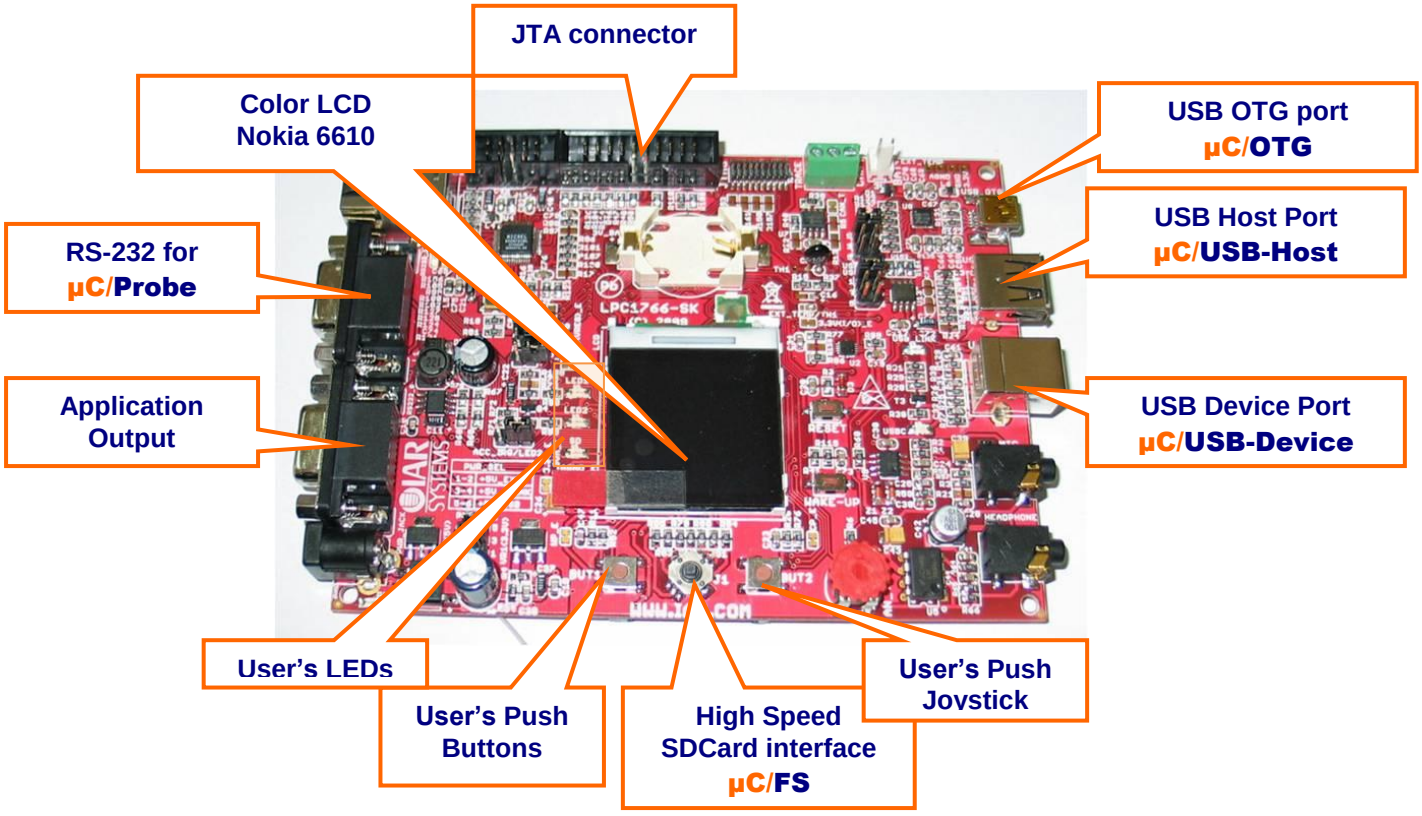


Figure 1-4. IAR LPC1768-SK Development Board

The RS232 port labeled "RS232 for **μC/Probe**" is used for **μC/Probe** (at 115200 baud), which allows you to view (in real-time) the value of any variables in the target system.

2. Directories and Files

Application Notes

\Micrium\AppNotes\AN1xxx-RTOS\AN1018-uCOS-II-Cortex-M3

This directory contains ***AN-1018.pdf***, the application note describing the ARM-Cortex-M3 port for **μC/OS-II**.

\Micrium\AppNotes\AN1xxx-RTOS\AN1080--uCOS-II-NXP-LPC1768-SK

This directory contains this application note, ***AN-1080.pdf***.

\Micrium\AppNotes\AN9xxx-MULT\AN-9913-PROBE-DEMO-INTRO

This directory contains this application note, ***AN-9913.pdf*** describing the introductory demo for **μC/Probe**

Licensing Information

\Micrium\Licensing

Licensing agreements are located in this directory. Any source code accompanying this appnote is provided for evaluation purposes only. If you choose to use **μC/OS-II** in a commercial product, you must contact Micriμm regarding the necessary licensing.

μC/OS-II Files

\Micrium\Software\uCOS-II\Doc

This directory contains documentation for **μC/OS-II**.

\Micrium\Software\uCOS-II\Ports\ARM\Generic\IAR

This directory contains the standard processor-specific files for the generic **μC/OS-II** ARM port assuming the IAR toolchain. These files could easily be modified to work with other toolchains (i.e., compiler/assembler/linker/locator/debugger); however, the modified files should be placed into a different directory. The following files are in this directory:

- *os_cpu.h*
- *os_cpu_a.asm*
- *os_cpu_c.c*
- *os_dcc.c*
- *os_dbg.c*

With this port, **μC/OS-II** can be used in either ARM or Thumb mode. Thumb mode, which drastically reduces the size of the code, was used in this example, but compiler settings may be switched (as discussed in Section 2.30) to generate ARM-mode code without needing to change either the port or the application code. The ARM/Thumb port is described in application note *AN-1014* which is available from the Micrium web site.

\Micrium\Software\uCOS-II\Source

This directory contains the processor-independent source code for **μC/OS-II**.

μC/Probe Files

\Micrium\Software\uC-Probe\Communication\Generic

This directory contains the **μC/Probe** generic communication module, the target-side code responsible for responding to requests from the **μC/Probe** Windows application (including requests over RS-232).

\Micrium\Software\uC-Probe\Communication\Generic\Source

This directory contains ***probe_com.c*** and ***probe_com.h***, the source code for the generic communication module.

\Micrium\Software\uC-Probe\Communication\Generic\OS\uCOS-II

This directory contains ***probe_com_os.c***, which is the **μC/OS-II** port for the **μC/Probe** generic communication module.

\Micrium\Software\uC-Probe\Communication\Generic\Source\RS-232

This directory contains the RS-232 specific code for **μC/Probe** generic communication module, the target-side code responsible for responding to requests from the **μC/Probe** Windows application over RS-232

\Micrium\Software\uC-Probe\Communication\Generic\Source\RS-232\Source

This directory contains ***probe_rs232.c*** and ***probe_rs232.h***, the source code for the generic communication module RS-232 code.

\Micrium\Software\uC-Probe\Communication\Generic\Source\RS-232\Ports\NXP\LPC17xx

These directories contain ***probe_rs232c.c*** and ***probe_rs232c.h***, the NXP LPC17xx port for the RS-232 communications.

\Micrium\Software\uC-Probe\Communication\Generic\Source\RS-232\OS\uCOS-II

This directory contains ***probe_rs232_os.c***, which is the **μC/OS-II** port for the **μC/Probe** RS-232 communication module.

\Micrium\Software\uC-Probe\Demos\Intro\Source

This directory contains ***probe_demo_intro.c***, which contains a self-explanatory introductory demo showing how to use **μC/Probe** (consult the application note ***AN-9913***)

μC/CPU Files

\Micrium\Software\uC-CPU

This directory contains ***cpu_def.h***, which declares `#define` constants for CPU alignment, endianness, and other generic CPU properties.

\Micrium\Software\uC-CPU\ARM\IAR

This directory contains ***cpu.h*** and ***cpu_a.s***. ***cpu.h*** defines the Micriμm portable data types for 8-, 16-, and 32-bit signed and unsigned integers (such as `CPU_INT16U`, a 16-bit unsigned integer). These allow code to be independent of processor and compiler word size definitions. *cpu_a.s* contains generic assembly code for ARM7 and ARM9 processors which is used to enable and disable interrupts within the operating system. This code is called from C with `OS_ENTER_CRITICAL()` and `OS_EXIT_CRITICAL()`.

μC/LIB Files

\Micrium\Software\uC-LIB

This directory contains ***lib_def.h***, which provides `#defines` for useful constants (like `DEF_TRUE` and `DEF_DISABLED`) and macros.

The files ***lib_mem.c*** and ***lib_mem.h*** contain code to replace the standard library functions `memclr()`, `memset()`, `memcopy()` and `memcmp()`. These functions are replaced by `Mem_Clr()`, `Mem_Set()`, `Mem_Copy()` and `Mem_Cmp()`, respectively.

The files ***lib_str.c*** and ***lib_str.h*** contain code to replace the standard library functions `str???()`, with the equivalent `Str_???()` functions.

The files ***lib_str.c*** and ***lib_str.h*** contain code to replace the standard library functions `str???()`, with the equivalent `Str_???()` functions.

The files ***lib_ascii.c*** and ***lib_ascii.h*** contain code to replace the standard library character classification and case conversion functions & macros such as `tolower()`, `toupper()`, `isalpha()`, `isdigit()`, `etc`. These functions are replaced with `ASCII_ToLower()`, `ASCII_ToUpper()`,`ASCII_IsAlpha()`and `ASCII_IsDig()`.

The files ***lib_math.c*** and ***lib_math.h*** contain code to replace the standard mathematics functions such as `rand()`, `srand()`, `etc`. These functions are replaced with `Math_Rand()`, `Math_RandSetSeed()`.

The reason Micium declare its own function of for third party certification for avionics and medical use

\Micrium\Software\uC-LIB\Doc

This directory contains the documentation for **μC/LIB**.

Application Code

\Micrium\Software\EvalBoards\NXP\LPC1768-SK\IAR\OS-Probe

This directory contains the source code the example application:

- ***app.c*** contains the test code for the example application including calls to the functions that start multitasking within **μC/OS-II**, register tasks with the kernel, and update the user interface (the LEDs and the push buttons).
- ***app_cfg.h*** is a configuration file specifying stack sizes and priorities for all user tasks and `#define`s for important global application constants.
- ***app_probe.c/h*** contain code to initialize **μC/Probe**,
- ***app_hooks.c/h*** contain code for the **μC/OS-II** application hooks.
- ***app_vect.c*** contain the initialization code for the NXP LPC17xx processor
- ***includes.h*** is the master include file used by the application.

- ***os_cfg.h*** is the **μC/OS-II** configuration file.
- ***cpu_cfg.h*** is the **μC/CPU** configuration file.
- ***probe_com_cfg.h*** is the **μC/Probe** configuration file.
- ***LPC1768-OS-Probe.**** are the IAR Embedded Workbench project files for the IAR LPC1768-SK board.

\Micrium\Software\EvalBoards\NXP\LPC1768 \IAR\BSP

This directory contains the Board Support Package and chip support package for the IAR LPC1768-SK development board and LPC1768 processor.

- ***bsp.c /h*** contain generic BSP functions which initialize critical processor functions (e.g., the PLL) and provide support for peripherals such as the push button and LEDs.
- ***bsp_int.c/h*** contain routines to install ISRs and enable/disable interrupt sources.
- ***bsp_pmc.c/h*** Contain basic function to enable, disable and retrieve clock frequency information from the peripheral and system clocks.
- ***bsp_ser.c/h*** Provide simple serial interface for tracing functionality.
- ***bsp_gpio.c/h*** Contain basic functionality to configure and manipulate I/Os pins.

3. Application Code

The example application described in this appnote, ***AN-1080***, is a simple demonstration of **μC/OS-II** and **μC/OS-Probe** for the NXP LPC1768 processors on the IAR LPC1768-SK developments board.

3.01 *app.c*

Four functions of interest are located in *app.c*:

1. **`main()`** is the entry point for the application, as it is with most C programs. This function initializes the operating system, creates the primary application task, `App_TaskStart()`, begins multitasking, and exits.

2. **`App_TaskStart()`**, after creating the application events and tasks, enters an infinite loop in which it blinks the LEDs.

3. **`App_TaskKbd()`** polls the user inputs—Board's Joystick—and, if new input is detected, places a message in a mailbox for `App_TaskUserIF()`.

4. **`App_TaskUserIF()`**,Outputs the state of the system based on the display state passed to it by `App_TaskKbd()`.

```
int  main (void)                                                    /* Note 1 */
{
#if (OS_TASK_NAME_EN > 0)
    CPU_INT08U  err;
#endif
#if (CPU_CFG_NAME_EN == DEF_ENABLED)
    CPU_ERR     cpu_err;
#endif

    CPU_Init();                                                     /* Note 2 */

#if (CPU_CFG_NAME_EN == DEF_ENABLED)
    CPU_NameSet((CPU_CHAR *)"LPC1768",
                (CPU_ERR  *)&cpu_err);
#endif

    CPU_IntDis();                                                   /* Note 3 */

    OSInit();                                                       /* Note 4 */

    OSTaskCreateExt((void (*)(void *)) App_TaskStart,               /* Note 5 */
                    (void           *) 0,
                    (OS_STK         *)&AppTaskStartStk[APP_CFG_TASK_START_STK_SIZE - 1],
                    (INT8U           ) APP_CFG_TASK_START_PRIO,
                    (INT16U          ) APP_CFG_TASK_START_PRIO,
                    (OS_STK         *)&AppTaskStartStk[0],
                    (INT32U          ) APP_CFG_TASK_START_STK_SIZE,
                    (void           *) 0,
                    (INT8U           )(OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));

#if (OS_TASK_NAME_EN > 0)                                           /* Note 6 */
    OSTaskNameSet(APP_CFG_TASK_START_PRIO, (CPU_INT08U *)"Startup", &err);
#endif

    OSStart();                                                      /* Note 7 */

    return (1);
}
```

Listing 3-1, `main()`

Listing 3-1, Note 1: As with most C applications, the code starts in `main()`.

Listing 3-1, Note 2: `CPU_Init()` initialize the **μC/CPU** module. `CPU_NameSet()` set the CPU Host Name

Listing 3-1, Note 3: `CPU_IntDis()` Disable all the interrupts. All interrupts are disabled to make sure the application does not get interrupted until is fully initialized.

Listing 3-1, Note 4: `OSInit()` must be called before creating a task or any other kernel object, as must be done with all **μC/OS-II** applications.

Listing 3-1, Note 5: At least one task must be created (in this case, using `OSTaskCreateExt()` to obtain additional information about the task). In addition, **μC/OS-II** creates either one or two internal tasks in `OSInit()`. **μC/OS-II** always creates an idle task, `OS_TaskIdle()`, and will create a statistic task, `OS_TaskStat()` if you set `OS_TASK_STAT_EN` to 1 in *os_cfg.h*.

Listing 3-1, Note 6: You can name **μC/OS-II** tasks (and other kernel objects) and display task names at run-time or with a debugger. In this case, the `App_TaskStart()` is given the name “Start Task”. Because C-Spy can work with the Kernel Awareness Plug-In available from Micriμm, task names can be displayed during debugging.

Listing 3-1, Note 7: Finally multitasking under **μC/OS-II** is started by calling `OSStart()`. **μC/OS-II** will then begin executing `App_TaskStart()` since that is the highest-priority task created (both `OS_TaskStat()` and `OS_TaskIdle()` having lower priorities).

```
static  void  App_TaskStart (void *p_arg)
{
    (void)p arg;


    BSP_Init();                                                 /* Note 1 */

    BSP_OS_TmrTickInit(OS_TICKS_PER_SEC);                       /* Note 2 */

#if (OS TASK STAT EN > 0)
    OSStatInit();                                               /* Note 3 */
#endif

    Mem_Init();                                                 /* Note 4 */
    Math Init();

    BSP_SerInit(115200);                                        /* Note 5 */
    APP TRACE INFO(("\n\n\r"));

#if (APP_CFG_PROBE_COM_MODULE_EN == DEF_ENABLED) || \
    (APP CFG PROBE OS PLUGIN EN  == DEF ENABLED)
    App_ProbeInit();                                            /* Note 6 */
#endif

    APP_TRACE_INFO(("Creating Application Events...\n\r"));
    App_EventCreate();                                          /* Note 7 */


    APP TRACE INFO(("Creating Application Tasks...\n\r"));
    App_TaskCreate();

    while (DEF_TRUE) {                                          /* Note 8 */
        BSP_LED_Toggle(0);
        OSTimeDlyHMSM(0, 0, 0, 100);
    }
}
```

Listing 3-2, `App_TaskStart ()`

Listing 3-2, Note 1: `BSP_PostInit()` initializes the Board Support Package drivers that are related to the OS or use a OS service (semaphores, mutexes, queues, etc)

Listing 3-2, Note 2: `BSP_OS_TmrTickInit()` Initializes the tick interrupt

Listing 3-2, Note 3: `OSStatInit()` initializes **μC/OS-II**'s statistic task. This only occurs if you enable the statistic task by setting `OS_TASK_STAT_EN` to 1 in *os_cfg.h*. The statistic task measures overall CPU usage (expressed as a percentage) and performs stack checking for all the tasks that have been created with `OSTaskCreateExt()` with the stack checking option set.

Listing 3-2, Note 4: `Mem_Init()` initializes the **μC/LIB** memory management module. `Mem_Math()` initializes the **μC/LIB** mathematical module.

Listing 3-2, Note 5: **`BSP_Ser_Init()`** Initializes the RS-232 communication port at 115200.

Listing 3-2, Note 6: If **μC/OS-Probe** is enabled, then the module's initialization procedure **`App_ProbeInit()`** is called. **`App_ProbeInit()`** calls `OSProbe_Init()` which initializes the

μC/Probe plug-in for **μC/OS-II**, which maintains CPU usage statistics for each task, `ProbeCom_Init()` that initializes the **μC/Probe** generic communication module and `ProbeRS232_Init()` that initializes the RS-232 communication module. After these have been initialized, the **μC/Probe** Windows program will be able to download data from the processor. For more information, see Section 6.

Listing 3-2, Note 7: **`App_EventCreate()`** Creates all the application uC/OS-II events and **`App_TaskCreate()`** creates all the application tasks.

Listing 3-2, Note 8: Any task managed by **μC/OS-II** must either enter an infinite loop 'waiting' for some event to occur or terminate itself. This task enters an infinite loop in which it toggles the LEDs .

3.02 *os_cfg.h*

The file ***os_cfg.h*** is used to configure **μC/OS-II** and defines the maximum number of tasks that your application can have, which services will be enabled (semaphores, mailboxes, queues, etc.), the size of the idle and statistic task and more. In all, there are about 60 or so `#define` that you can set in this file. Each entry is commented and additional information about the purpose of each `#define` can be found in

Task sizes for the Idle (**`OS_TASK_IDLE_STK_SIZE`**), statistics **`OS_TASK_STAT_STK_SIZE`**) and timer (**`OS_TASK_TMR_STK_SIZE`**) task are set to 128 `OS_STK` elements (each is 4 bytes) and thus each task stack is 512 bytes. If you add code to the examples make sure you account for additional stack usage.

- **`OS_DEBUG_EN`** is set to 1 to provide valuable information about **μC/OS-II** objects to IAR's C-Spy through the Kernel Awareness plug-in. Setting `OS_DEBUG_EN` to 0 should some code space (though it will not save much).

- **`OS_LOWEST_PRIO`** is set to 63, allowing up to 64 total tasks.

- **`OS_MAX_TASKS`** determines the number of "application" tasks and is currently set to 20 allowing 13 more tasks to be added to the example code.

- **`OS_TICKS_PER_SEC`** is set to 1000 Hz. This value can be changed as needed and the proper tick rate will be adjusted when the `BSP_OS_TmrTickInit()` is called. if you change this value. You would typically set the tick rate between 10 and 1000 Hz. The higher the tick rate, the more overhead **μC/OS-II** will impose on the application. However, you will have better tick granularity with a higher tick rate.

4. Board Support Package (BSP)

The Board Support Package (BSP) provides functions to encapsulate common I/O access functions and make porting your application code easier. Essentially, these files are the interface between the application and LPC1768-SK board.

4.01 BSP, *bsp_xxx.c* and *bsp_xxx.h* files

Figure 4-1 shows the relationship between the BSP's functions list and the most important components on the Processor/development boards

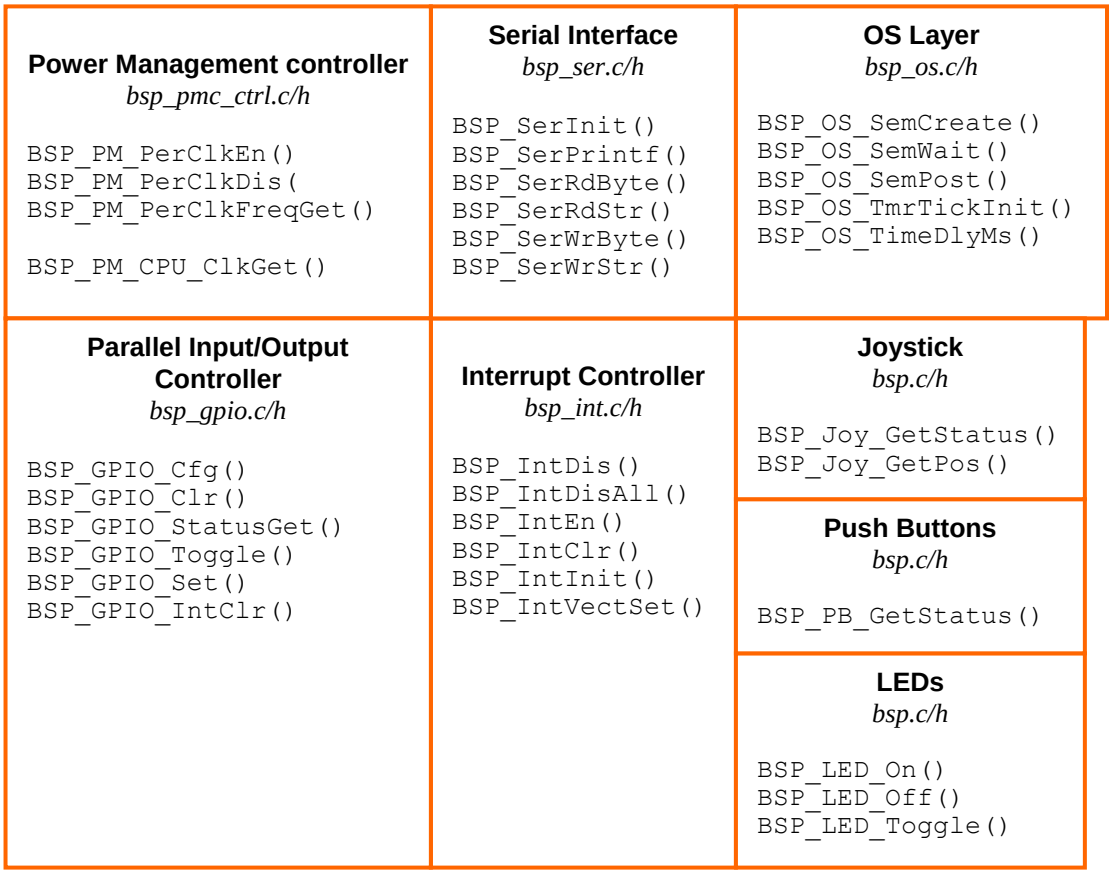


Figure 4-1. BSP's Functions List for the LPC1768-SK

4.02 Board Support Package Configuration

The serial port used to output the system state can be configured at compile-time using the following #define:

BSP_CFG_SER_COMM_SEL	**BSP_SER_COMM_UART_00** **BSP_SER_COMM_UART_01**	Defines the serial port used to output the system state.

4.03 Tick Interrupt code.

Listings 5-2 gives the **μC/OS-II** timer tick initialization function, `BSP_OS_TmrTickInit()`.

```
void  BSP_OS_TmrTickInit (CPU_INT32U  tick_per_sec)
{
    CPU_INT32U  cnts;
    CPU_INT32U  cpu_freq;

    cpu_freq = BSP_PM_CPU_FreqGet(BSP_SYS_CLK_ID_MCLK);         /* Note 1 */

    cnts     = (cpu_freq / tick_rate);                          /* Note 2 */

    OS_CPU_SysTickInit(cnts);
}
```

Listing 5-2, `BSP_OS_TmrTickInit()`

The **μC/OS-II** ARM Cortex M3 port uses the SysTick timer. On the NXP LPC17xx processors the SysTick clock is the CPU clock.

Listing 5-2, Note 1: Get the CP clock frequency.

Listing 5-2, Note 2: Calculate the reload value.

Listing 5-2, Note 3: **`OS_CPU_SysTickInit()`** initialize the SysTick timer with the number of SysTick counts between two OS tick interrupts.

5. μC/Probe

μC/Probe is a Windows program which retrieves the values of global variables from a connected embedded target and displays the values in a engineer-friendly format. To accomplish this, an ELF file, created by the user's compiler and containing the names and addresses of all the global symbols on the target, is monitored by **μC/Probe**. The user places components (such as gauges, labels, and charts) into a Data Screen in a **μC/Probe** workspace and assigns each one of these a variable from the Symbol Browser, which lists all symbols from the ELF file. The symbols associated with components placed on an open Data Screen will be updated after the user presses the start button (assuming the user's PC is connected to the target).

μC/Probe currently interfaces with a target processor with a RS-232. A small section of code resident on the target receives commands from the Windows application and responds to those commands. The commands ask for a certain number of bytes located at a certain address, for example, "Send 16 bytes beginning at 0x0040102C". The Windows application, upon receiving the response, updates the appropriate component(s) on the screens with the new values.

Figure 5-1. μC/Probe Windows Program

To use **μC/Probe** with the example project (or your application), do the following:

1. **Download and Install μC/Probe.** A trial version of **μC/Probe** can be downloaded from the Micriμm website at

   http://www.micrium.com/products/probe/probe.html

2. **Open μC/Probe**. After downloading and installing this program, open the example **μC/Probe** workspace for **μC/OS-II**, named ***OS-Probe-Workspace.wsp***, which should be located in your installation directory at

   /Program Files//Micrium/uC-Probe/Target/Plugins/uCOS-II/Workspace

3. **Connect Target to PC**. Currently, **μC/Probe** can use RS-232 to retrieve information from the target. You should connect a RS-232 cable between your target and computer.

4. **Load Your ELF File**. The example projects included with this application note are already configured to output an ELF file. (If you are using your own project, please refer to Appendix A of the **μC/Probe** user manual for directions for generating an ELF file with your compiler.) This file should be in

   /<Project Directory>/<Configuration Name>/exe/

   where <***Project Directory***> is the directory in which the IAR EWARM project is located (extension *.ewp) and <***Configuration Name***> is the name of the configuration in that project which was built to generate the ELF file and which will be loaded onto the target. The ELF file will be named

   <***Project Name***>***.elf***

   in EWARM v4.4x and

   <***Project Name***>***.out***

   in EWARM v5.1x unless you specify otherwise. To load this ELF file, right-click on the symbol browser and choose “Add Symbols”.

5. **Configure the RS-232 Options**. In **μC/Probe**, choose the “Options” menu item on the “Tools” menu. A dialog box as shown in Figure 6-2 (left) should appear. Choose the “RS-232” radio button. Next, select the “RS-232” item in the options tree, and choose the appropriate COM port and baud rate. The baud rate for the projects accompanying this appnote is 115200.

6. **Start Running**. You should now be ready to run **μC/Probe**. Just press the run button () to see the variables in the open data screens update. Figure 6-3 displays two screens in the **μC/OS-II** workspace which display detailed information about each task’s state.

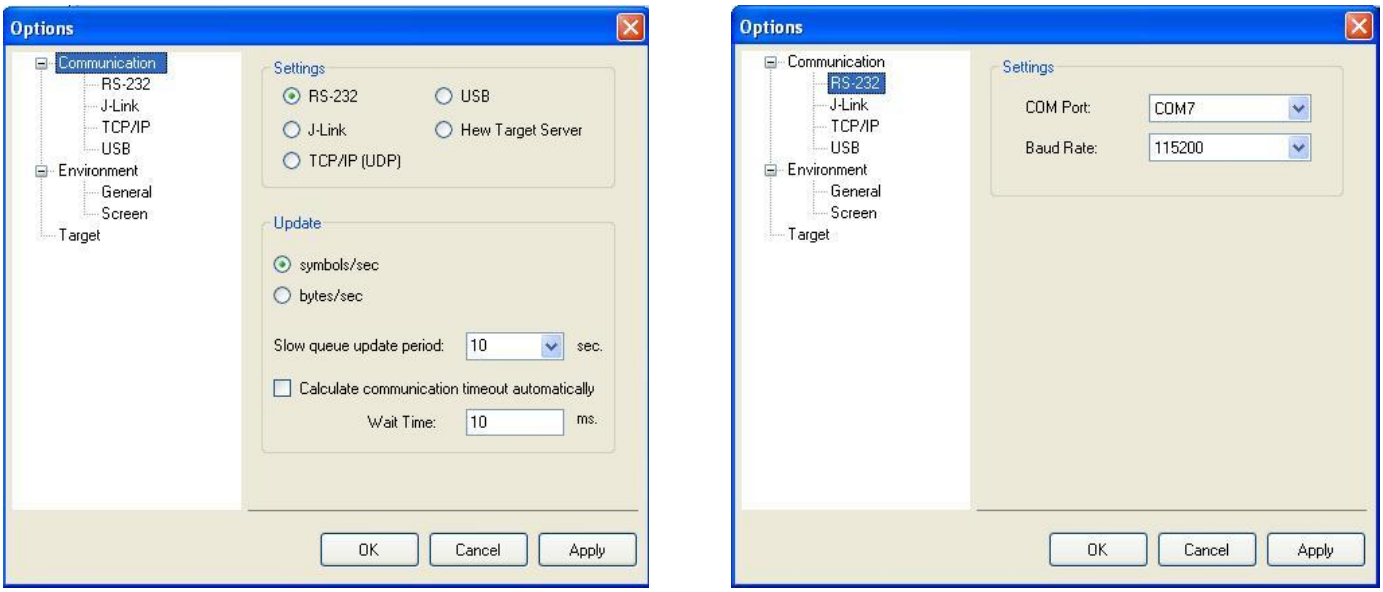


Figure 5.2. μC/Probe Options

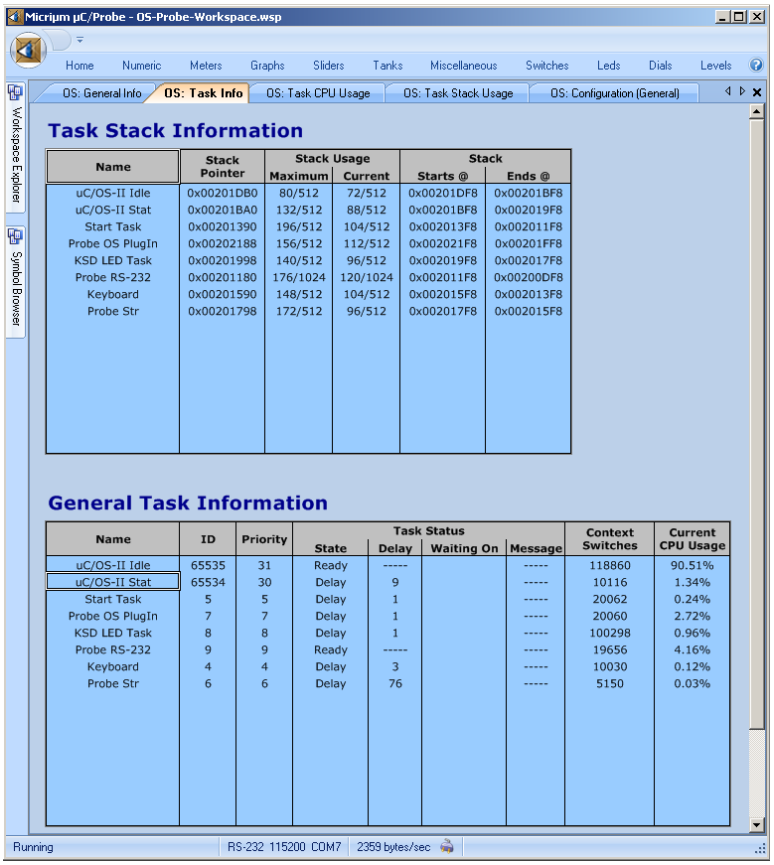


Figure 5-3. μC/Probe Run-Time: μC/OS-II Task Information

Licensing

μC/OS-II is provided in source form for **FREE** evaluation, for educational use or for peaceful research. If you plan on using **μC/OS-II** in a commercial product you need to contact Micriμm to properly license its use in your product. We provide **ALL** the source code with this application note for your convenience and to help you experience **μC/OS-II**. The fact that the source is provided does **NOT** mean that you can use it without paying a licensing fee. Please help us continue to provide the Embedded community with the finest software available. Your honesty is greatly appreciated.

References

μC/OS-II, The Real-Time Kernel, 2nd Edition
Jean J. Labrosse
R&D Technical Books, 2002
ISBN 1-57820-103-9

Embedded Systems Building Blocks
Jean J. Labrosse
R&D Technical Books, 2000
ISBN 0-87930-604-1

Contacts

IAR Systems
Century Plaza
1065 E. Hillsdale Blvd
Foster City, CA 94404
USA

+1 650 287 4250
+1 650 287 4253 (FAX)

e-mail: Info@IAR.com
WEB : www.IAR.com

CMP Books, Inc.
1601 W. 23rd St., Suite 200
Lawrence, KS 66046-9950
USA

+1 785 841 1631
+1 785 841 2624 (FAX)

e-mail: rushorders@cmpbooks.com
WEB : http://www.cmpbooks.com

Micriμm
949 Crestview Circle
Weston, FL 33327
USA

+1 954 217 2036
+1 954 217 2037 (FAX)

e-mail: support@micrium.com
WEB : www.micrium.com