Micriμm

Empowering Embedded Systems

μC/OS-II

μC/Probe

and the

STMicroelectronics STM32 Processor

(Using the ST STM3210B-EVAL Evaluation Board, ST STM3210E-EVAL Evaluation Board, and the IAR STM32-SK Evaluation Board)

Application Note

AN-1320

www.Micrium.com

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

μC/OS-II is a preemptive, real-time, multitasking kernel. **μC/OS-II** has been ported to over 45 different CPU architectures.

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

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.1.01	2008/08/15	BAN	Updated.
V.1.00	2007/07/02	BAN	Initial version.

Software Versions

This document may or may not have been downloaded as part of an executable file, *Micrium-ST-uCOS-II-LCD--STM32.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	V2.86	
µC/Probe	V2.00	

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. Introduction

This document, ***AN-1320***, explains example code for using **μC/OS-II** and **μC/Probe** with the STMicroelectronics STM32 (Cortex-M3) processor on three different evaluation boards. The first is ST's STM3210B-EVAL evaluation board (shown in Figure 1-2); the second is IAR's STM32-SK evaluation board (shown in Figure 1-1); the third is ST's STM3210E-EVAL evaluation board (shown in Figure 1-3). The STM32 tested on these evaluation boards included a 128-kB flash and 20-kB SRAM and could operate at clock speeds as high as 72-MHz. Peripherals for several communications busses are provided, including UARTs, I^{2}C, SPI, CAN, and USB. Two twelve-channel ADCs, 3 general-purpose timers and up to 80 GPIOs round out the features on the chip.

All boards provide similar capabilities. Each has two RS-232 ports, one CAN port, one USB device port, an SD card slot (which, on the STM3210B-EVAL/STM3210E-EVAL, is a micro-SD card slot) and a 20-pin JTAG for debugging and loading the processor. The IAR STM32-SK has three user push buttons, one potentiometer, up to 16 LEDs (depending on other hardware usage) and a 2- x16-character LCD. The ST STM3210B-EVAL/STM3210E-EVAL has two user push buttons, one potentiometer, four LEDs, and a 240- x 320-pixel LCD.

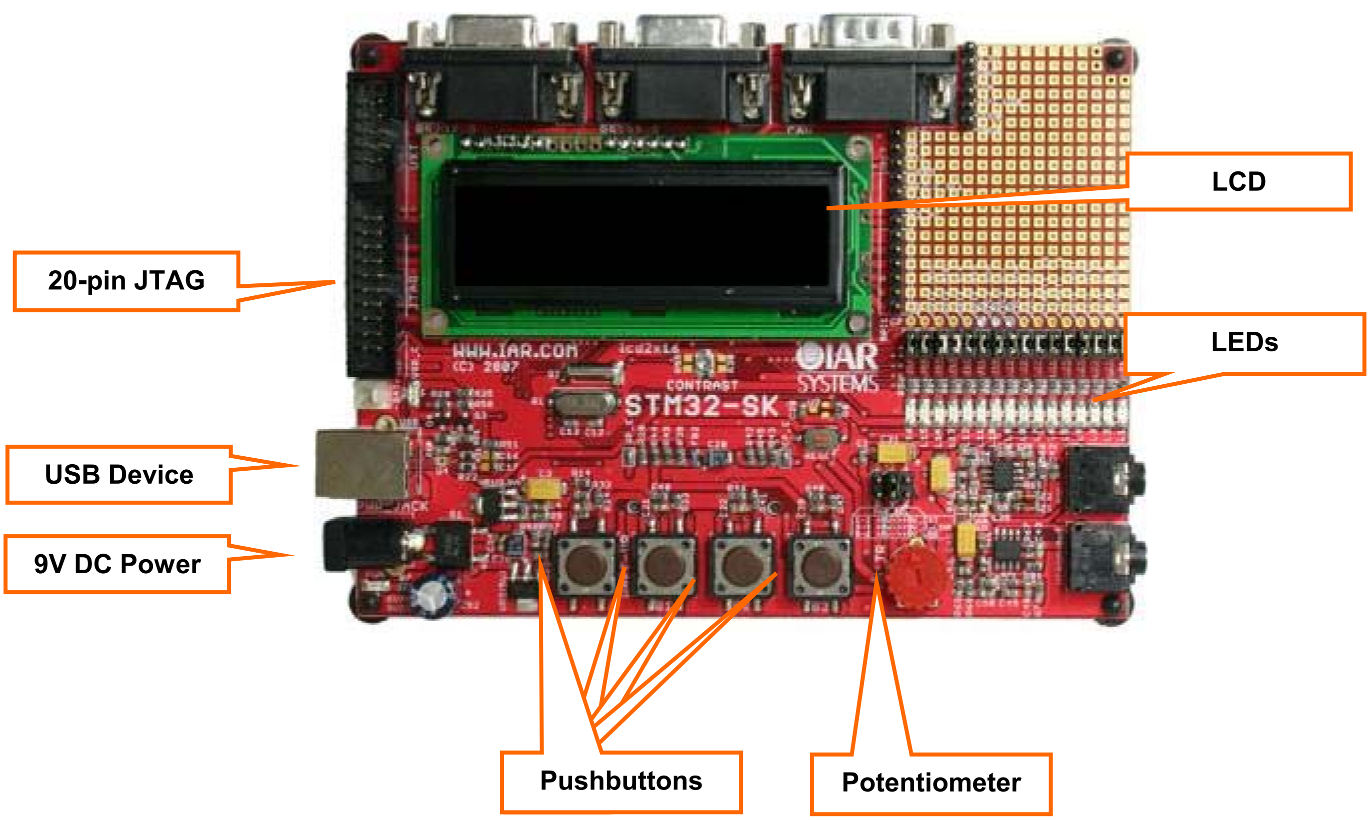


Figure 1-1. IAR STM32-SK Evaluation Board

STMicroelectronics provides a driver library for its STM32 processors, as it does for its ARM7 and ARM9 offerings. Each processor peripheral is represented by a family of functions intended to quickly acquaint the new user with the basic capability (and basic functional details) of that peripheral. Though the libraries may be adequate only as a reference when more complex requirements are faced, the example application detailed in this application note accomplishes much of its peripheral access through the driver library.

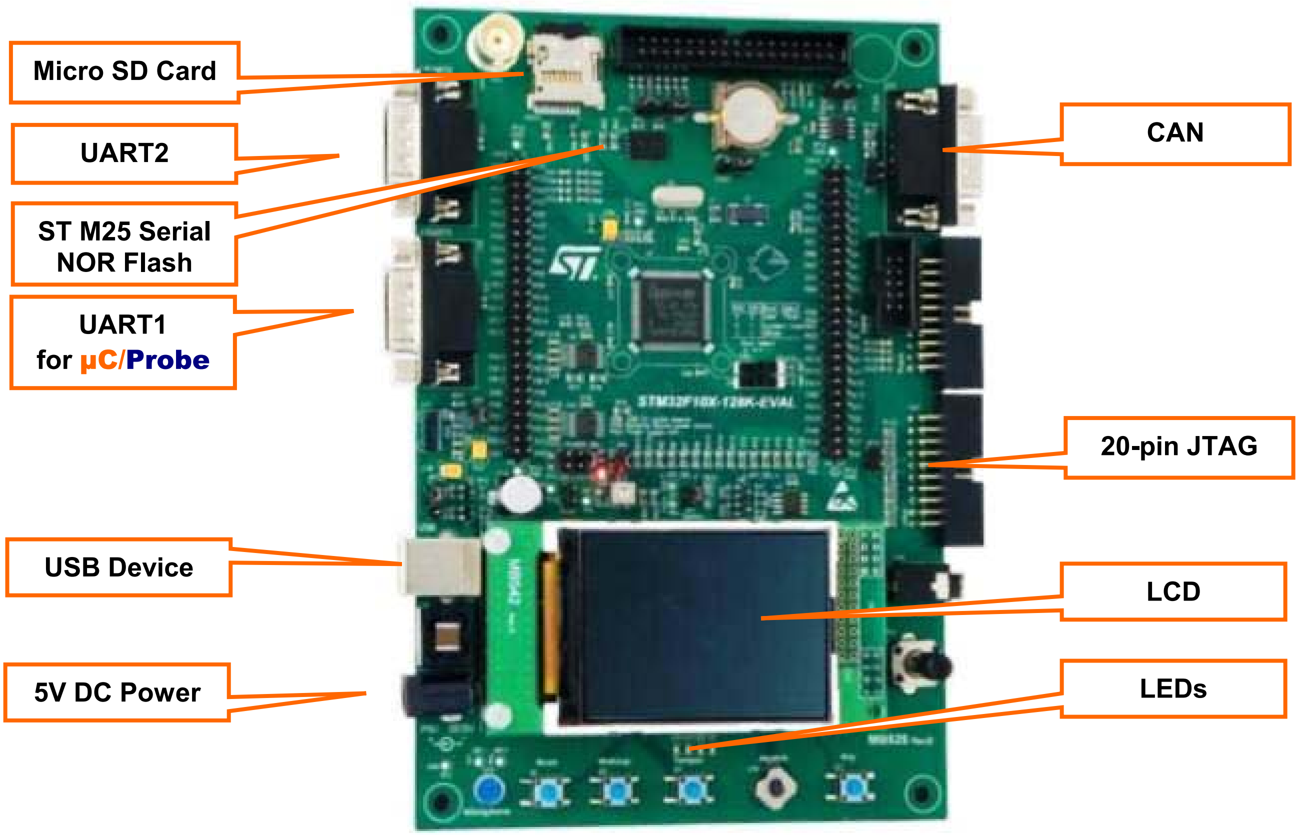
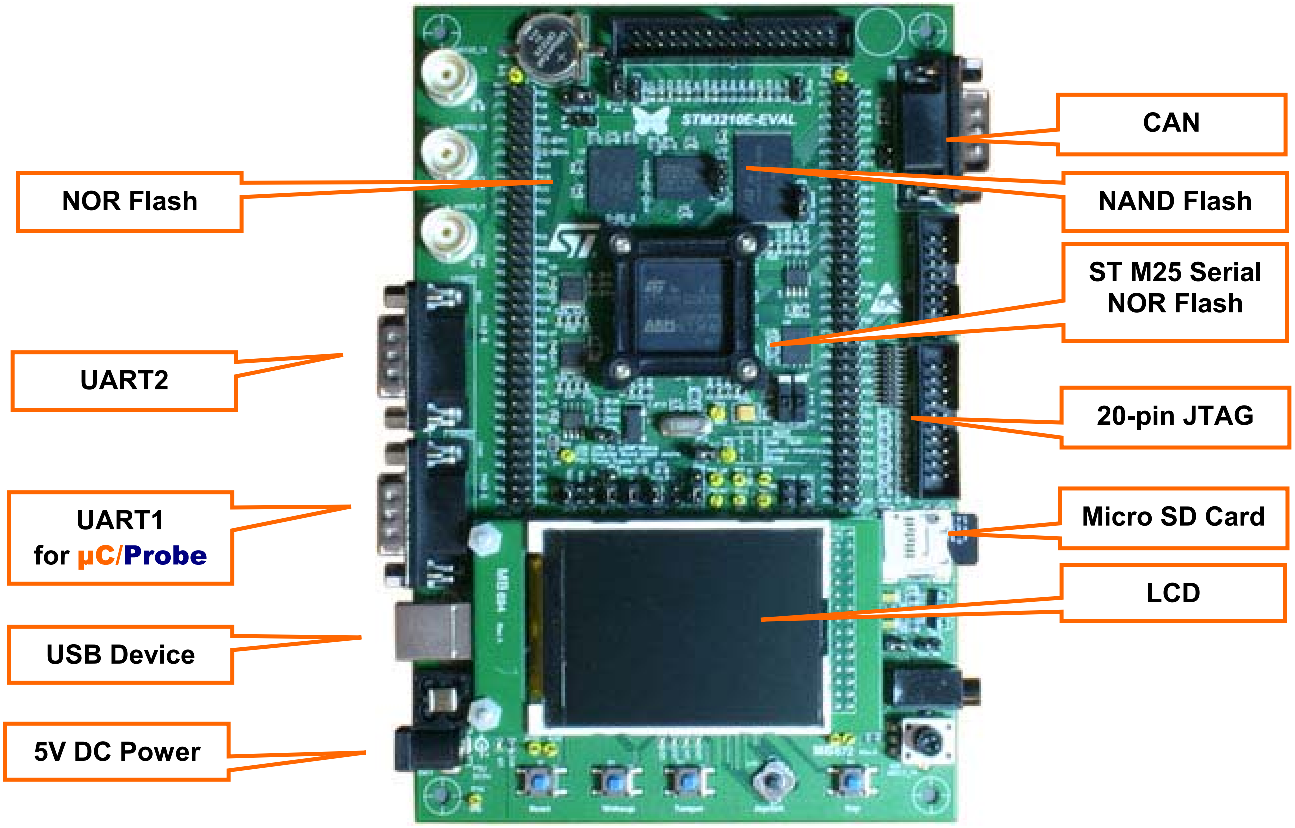
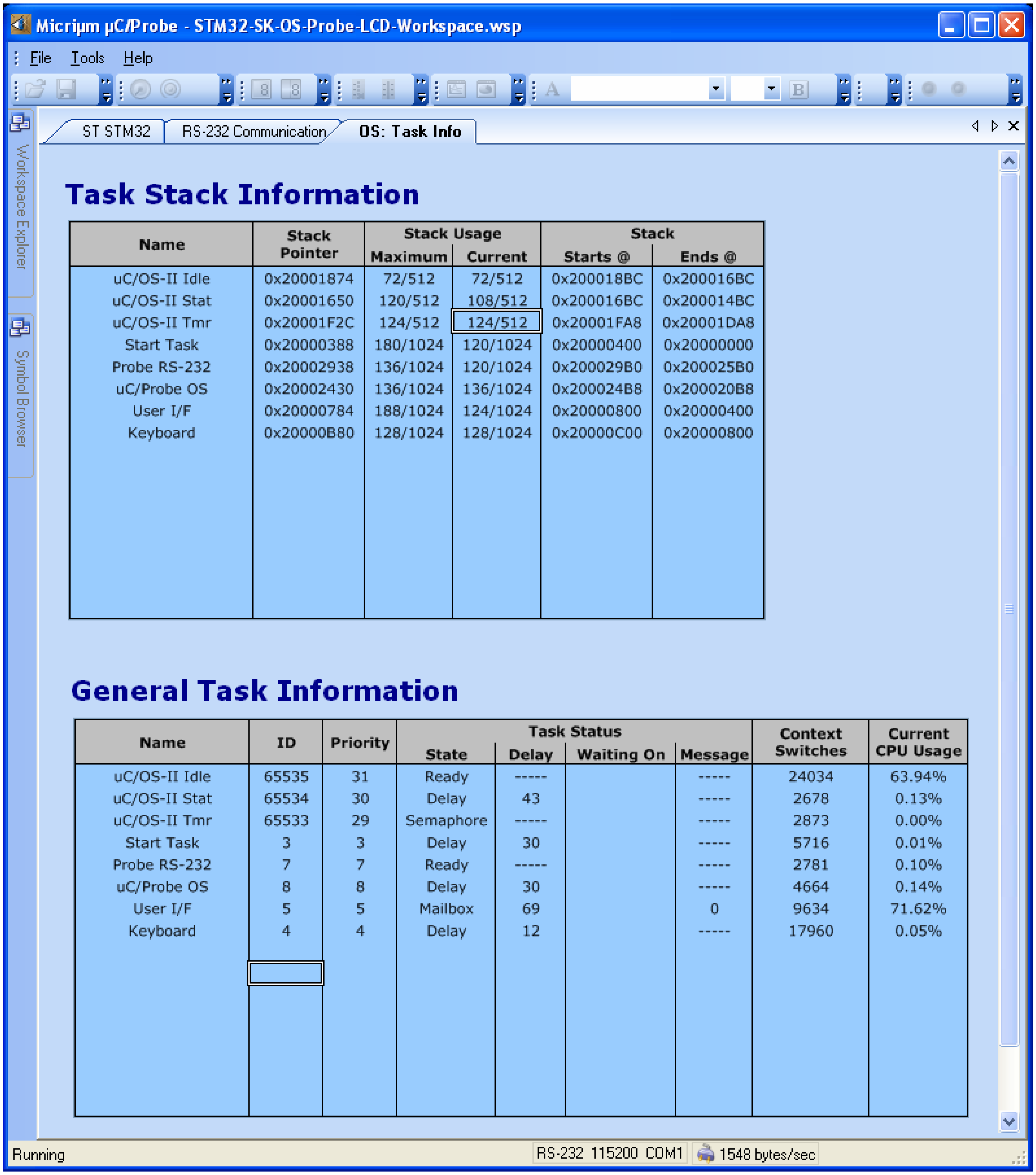


Figure 1-2. ST STM3210B-EVAL Evaluation Board

If this appnote was downloaded in a packaged executable zip file, then it should have been found in the directory ***/Micrium/Appnotes/AN1xxx-RTOS/AN1320-uCOS-II-ST-STM32*** and the code files referred to herein are located in the directory structure displayed in Section 2.02; these files are described in Section 3.



The executable zip also includes example workspaces for **µC/Probe**. **µC/Probe** is a Windows program which retrieves the value of variables form a connected embedded target and displays the values in an engineer-friendly format. It interfaces with the STM32 via RS-232. For more information, including instructions for downloading a trial version of the program, please refer to Section 6.



Task Stack Information

Name	Stack Pointer	Stack Usage		Stack	
		Maximum	Current	Starts @	Ends @
uC/OS-II Idle	0x20001874	72/512	72/512	0x200018BC	0x200016BC
uC/OS-II Stat	0x20001650	120/512	108/512	0x200016BC	0x200014BC
uC/OS-II Tmr	0x20001F2C	124/512	124/512	0x20001FA8	0x20001DA8
Start Task	0x20000388	180/1024	120/1024	0x20000400	0x20000000
Probe RS-232	0x20002938	136/1024	120/1024	0x200029B0	0x200025B0
uC/Probe OS	0x20002430	136/1024	136/1024	0x200024B8	0x200020B8
User I/F	0x20000784	188/1024	124/1024	0x20000800	0x20000400
Keyboard	0x20000B80	128/1024	128/1024	0x20000C00	0x20000800

General Task Information

Name	ID	Priority	Task Status				Context Switches	Current CPU Usage
			State	Delay	Waiting On	Message		
uC/OS-II Idle	65535	31	Ready	-----		-----	24034	63.94%
uC/OS-II Stat	65534	30	Delay	43		-----	2678	0.13%
uC/OS-II Tmr	65533	29	Semaphore	-----		-----	2873	0.00%
Start Task	3	3	Delay	30		-----	5716	0.01%
Probe RS-232	7	7	Ready	-----		-----	2781	0.10%
uC/Probe OS	8	8	Delay	30		-----	4664	0.14%
User I/F	5	5	Mailbox	69		0	9634	71.62%
Keyboard	4	4	Delay	12		-----	17960	0.05%

Figure 1-4. µC/Probe

2. Getting Started

The following sections step through the prerequisites for using the demonstration application described in this document, ***AN-1320***. First, the setup of the hardware will be outlined. Second, the use and setup of the IAR Embedded Workbench and the Keil µVision3 projects will be described. Thirdly, the steps to build the projects and load the application onto the board through a JTAG will be described. Lastly, instructions will be provided for using the example application.

2.01 Setting up the Hardware

The processors on all evaluation boards can be programmed and debugged through the 20-pin JTAG port using a JTAG emulator, such as a J-Link (which we used for the IAR projects) or a ULINK (which we used for the Keil projects).

All boards can use power from a standard DC converter. For the STM3210B-EVAL and STM3210E-EVAL, this should supply 5VDC; for the STM32-SK, this should supply 9VDC.

To use **µC/Probe** with the STM32, download and install the trial version of the program from the Micriµm website as discussed in Section 6. After programming your target with one of the included example projects, connect a RS-232 cable between your PC and the evaluation board, configure the RS-232 options (also covered in Section 6), and start running the program. The open data screens should update, as shown in Figure 1-2. The STM32-SK example application is configured to use UART2 (via the RS-232 port labeled “RS232_2”). The STM3210B-EVAL/STM3210E-EVAL example applications are configured to use UART1. Both are configured to operate at 115200 baud.

2.02 Directory Tree

If this file were downloaded as part of an executable zip file (which should have been named *Micrium-ST-uCOS-II-LCD-STM32.exe*), then the code files referred to herein are located in the directory structure shown in Figure 2-2. .

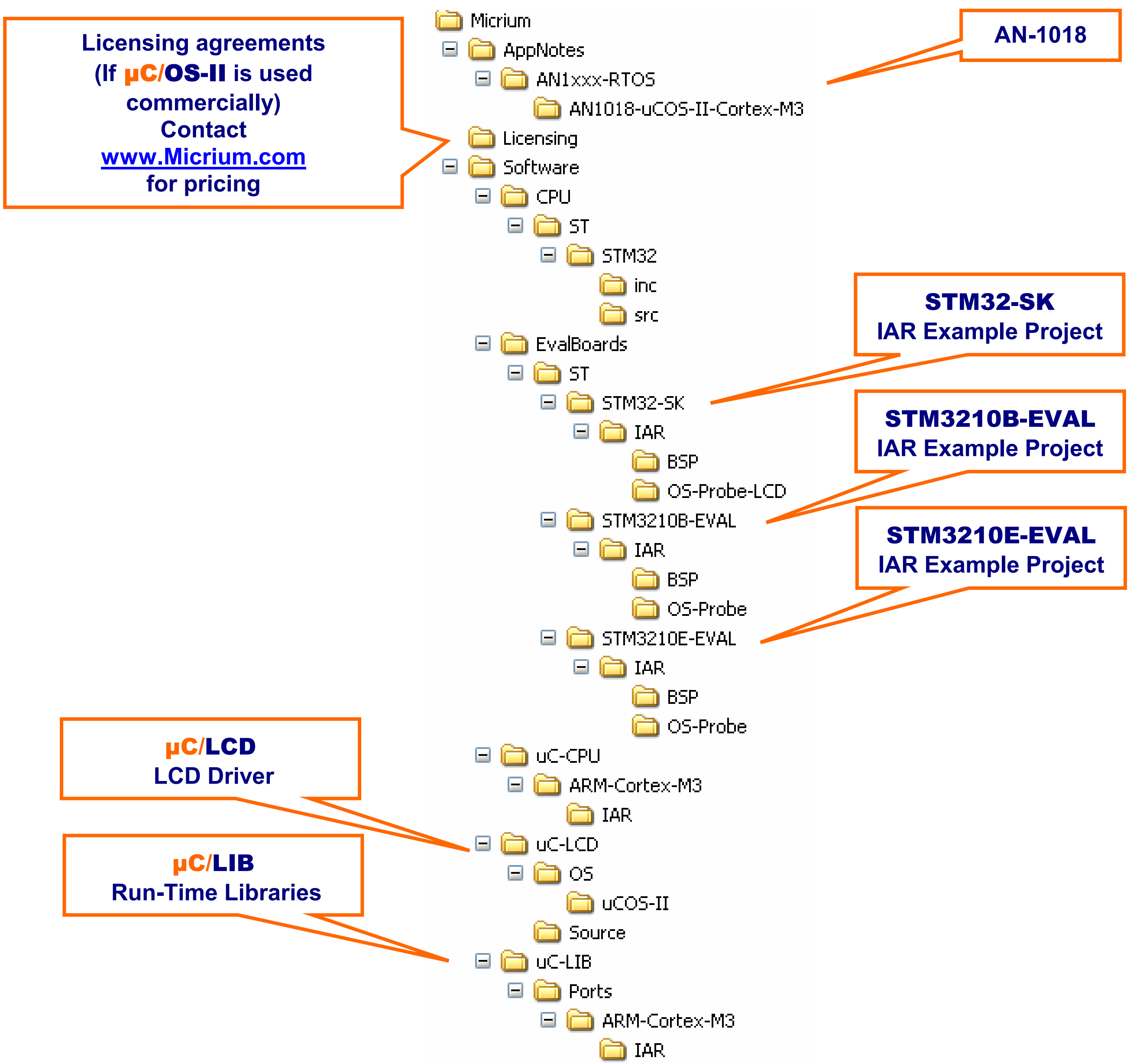


Figure 2-2. Directory Structure

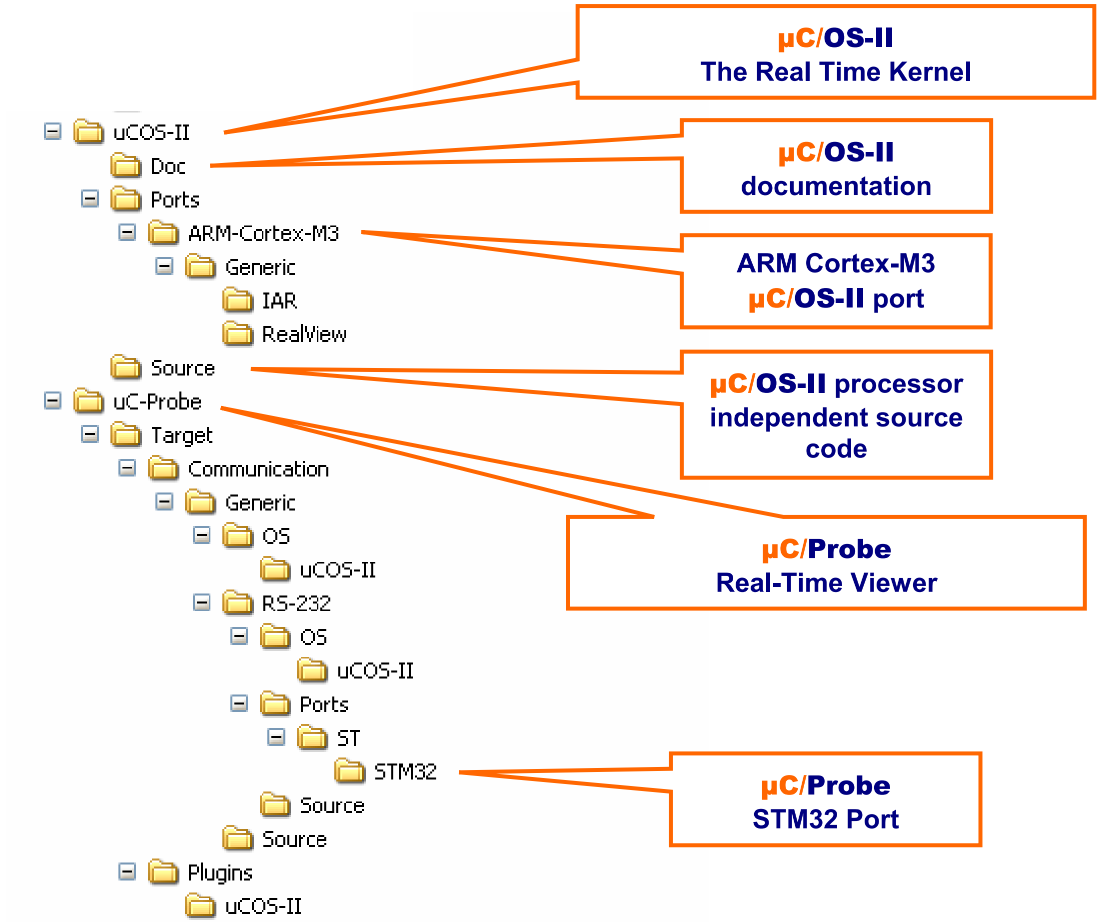


Figure 2-2. Directory Structure (continued)

2.03 STM32-SK IAR Project

An IAR project file named ***STM32-SK-OS-Probe-LCD.ewp*** is located in the directory (marked "STM32-SK IAR Example Project" in Figure 2-2):

/Micrium/Software/EvalBoards/ST/STM32-SK/IAR/OS- Probe-LCD

To view this example project, start an instance of IAR EW, and open the workspace file ***STM32-SK-OS-Probe-LCD.eww***. To do this, select the "Open" menu command under the "File" menu, select the "Workspace…" submenu command and select the workspace file after navigating to the project directory. The project tree shown in Figure 2-3 should appear. (In addition, the workspace should be openable by double-clicking on the file itself in a Windows explorer window.)

2.03.01 Project Options

Once the connections described in Section 2.01 are made between your PC and the IAR STM32-SK evaluation board, the code can be built and loaded onto the board. To build the code, select the "Rebuild All" menu item from the "Project" menu. To load the code through a JTAG debugger onto the connected evaluation board, select the "Debug" menu item from the "Project" menu.

2.03.02 µC/OS-II Kernel Awareness

When running the IAR C-Spy debugger, the **µC/OS-II** Kernel Awareness Plug-In can be used to provide useful information about the status of **µC/OS-II** objects and tasks. If the **µC/OS-II** Kernel Awareness Plug-In is currently enabled, then a “µC/OS-II” menu should be displayed while debugging. Otherwise, the plug-in can be enabled. Stop the debugger (if it is currently active) and select the “Options” menu item from the “Project” menu. Select the “Debugger” entry in the list box and then select the “Plugins” tab pane. Find the **µC/OS-II** entry in the list and select the check box beside the entry, as shown in Figure 2-5.

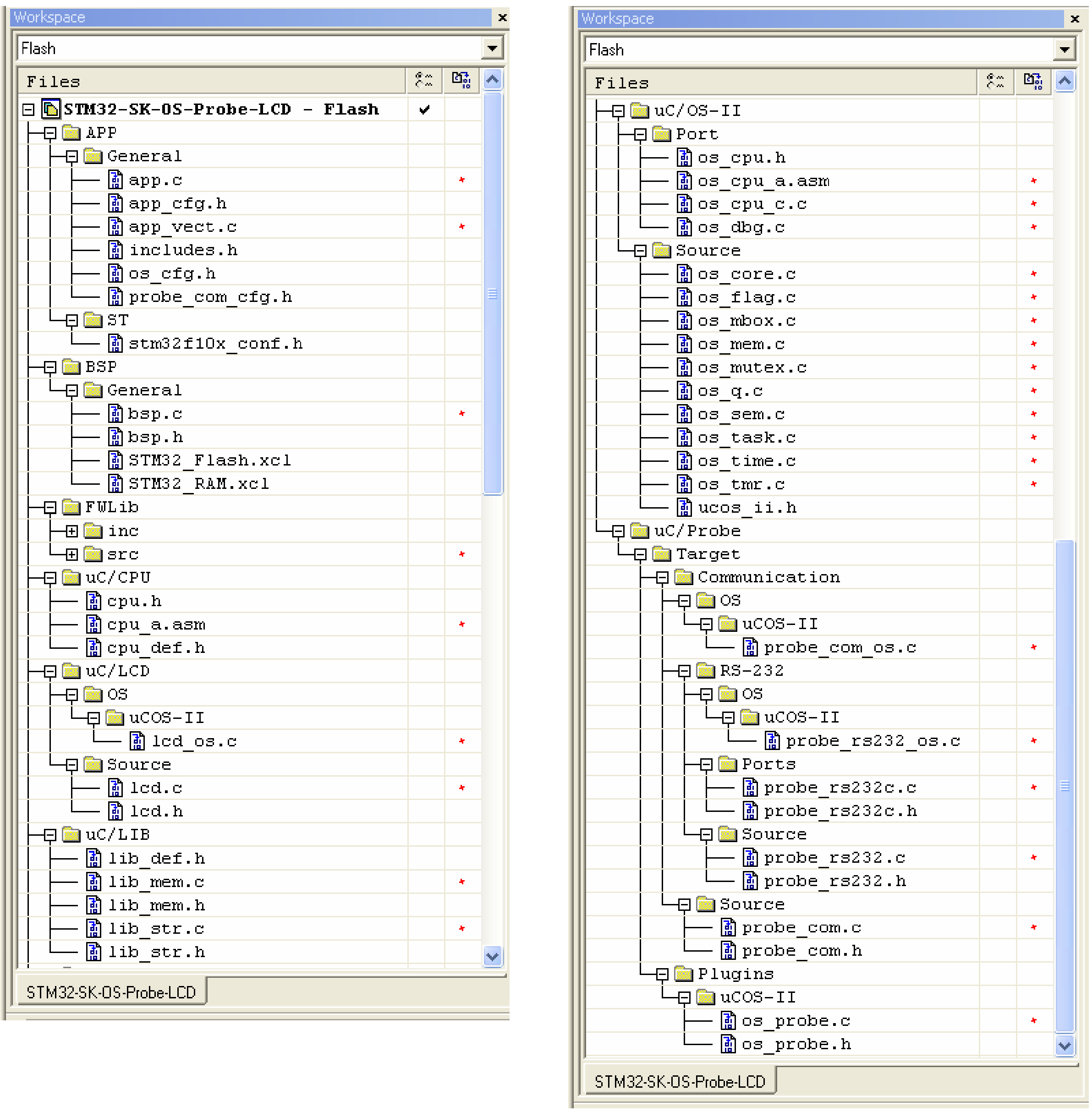


Figure 2-3. IAR EWARM Project Tree for *STM32-SK-OS- Probe-LCD.ewp*.

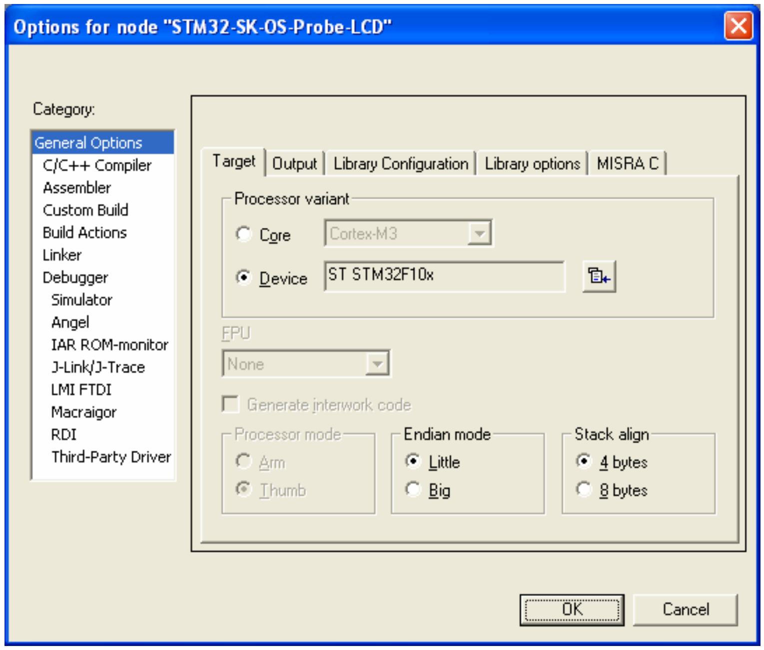


Figure 2-4. Project Options: General Options

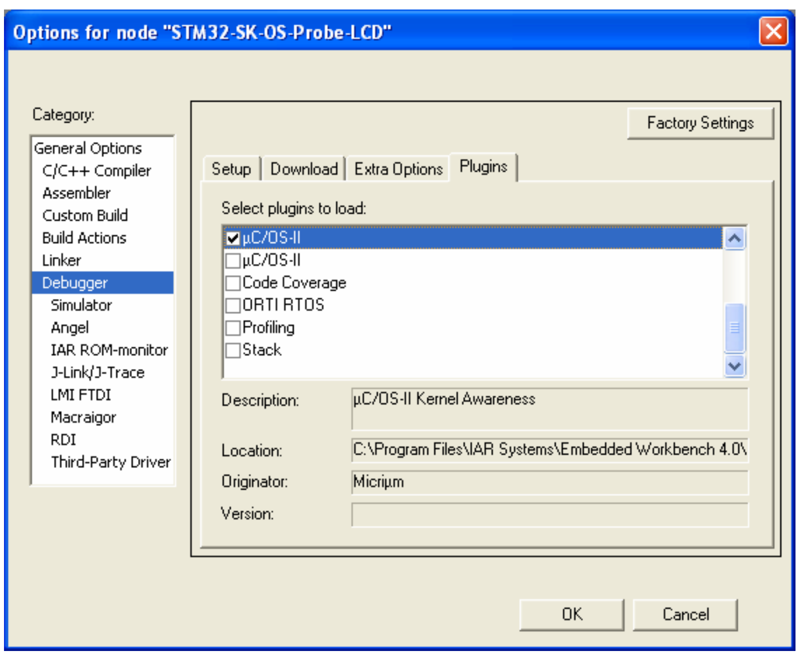


Figure 2-5. Enabling the µC/OS-II Kernel Awareness Plug-In

When the code is reloaded onto the evaluation board, the “µC/OS-II” menu should appear. Options are included to display lists of kernel objects such as semaphores, queues, and mailboxes, including for each entry the state of the object. Additionally, a list of the current tasks may be displayed, including for each task pertinent information such as used stack space, task status, and task priority, in addition to showing the actively executing task. An example task list for this project is shown in Figure 2-6.

Task List

	Name	Ref	Prio	State	Dly	Waiting On	Msg	Ctx Sw	Stk Ptr	Max%	Cur%	Max	Cur	Size	Starts @	Ends @
	Start Task	3	3	Dly	46			1334	20000188	35%	23%	180	120	512	20000200	20000000
	Keyboard	7	4	Dly	13			4188	20000580	25%	25%	128	128	512	20000600	20000400
	User I/F	6	5	Mbox	34	?		2234	20000384	34%	24%	176	124	512	20000400	20000200
	Probe Str	8	6	Dly	161			447	20000790	25%	21%	132	112	512	20000800	20000600
	Probe RS-232	4	7	Sem	0	Probe RS-232		367	20002594	13%	10%	140	108	1024	20002600	20002200
	uC/Probe OS	5	8	Dly	2			1091	20001E70	26%	26%	136	136	512	20001EF8	20001CF8
	uC/OS-II Tmr	2	29	Sem	0	OS-TmrSig		673	20001B6C	24%	24%	124	124	512	20001BE8	200019E8
	uC/OS-II Stat	1	30	Dly	2			626	20001290	23%	21%	120	108	512	200012FC	200010FC
>	uC/OS-II Idle	0	31	Ready	0			5359	200014B8	14%	13%	72	68	512	200014FC	200012FC

Figure 2-6. μC/OS-II Task List for *STM32-SK-OS- Probe-LCD.ewp.*

2.04 STM3210B-EVAL/STM3210E-EVAL IAR Project

An IAR project file named ***STM3210B-EVAL-OS-Probe.ewp*** is located in the directory (marked "STM32102B-EVAL IAR Example Project" in Figure 2-2):

/Micrium/Software/EvalBoards/ST/STM310B-EVAL/IAR/OS- Probe

To view this example project, start an instance of IAR EW, and open the workspace file ***STM3210B-EVAL-OS-Probe.eww***. To do this, select the "Open" menu command under the "File" menu, select the "Workspace…" submenu command and select the workspace file after navigating to the project directory. The project tree shown in Figure 2-7 should appear. (In addition, the workspace should be openable by double-clicking on the file itself in a Windows explorer window.)

The basically identical STM3210E-EVAL IAR project file named ***STM3210E-EVAL-OS-Probe.ewp*** is located in the directory (marked "STM32102E-EVAL IAR Example Project" in Figure 2-2):

/Micrium/Software/EvalBoards/ST/STM310E-EVAL/IAR/OS- Probe

The information in this section, though specifically addressing the STM3210B-EVAL, also applies to the STM3210B-EVAL.

2.04.01 Project Options

Once the connections described in Section 2.01 are made between your PC and the ST STM3210B-EVAL evaluation board, the code can be built and loaded onto the board. To build the code, select the "Rebuild All" menu item from the "Project" menu. To load the code through a JTAG debugger onto the connected evaluation board, select the "Debug" menu item from the "Project" menu.

2.04.02 μC/OS-II Kernel Awareness

When running the IAR C-Spy debugger, the **μC/OS-II** Kernel Awareness Plug-In can be used to provide useful information about the status of **μC/OS-II** objects and tasks. If the **μC/OS-II** Kernel Awareness Plug-In is currently enabled, then a "μC/OS-II" menu should be displayed while debugging. Otherwise, the plug-in can be enabled. Stop the debugger (if it is currently active) and select the "Options" menu item from the "Project" menu. Select the "Debugger" entry in the list box and then select the "Plugins" tab pane. Find the **μC/OS-II** entry in the list and select the check box beside the entry, as shown in Figure 2-9.

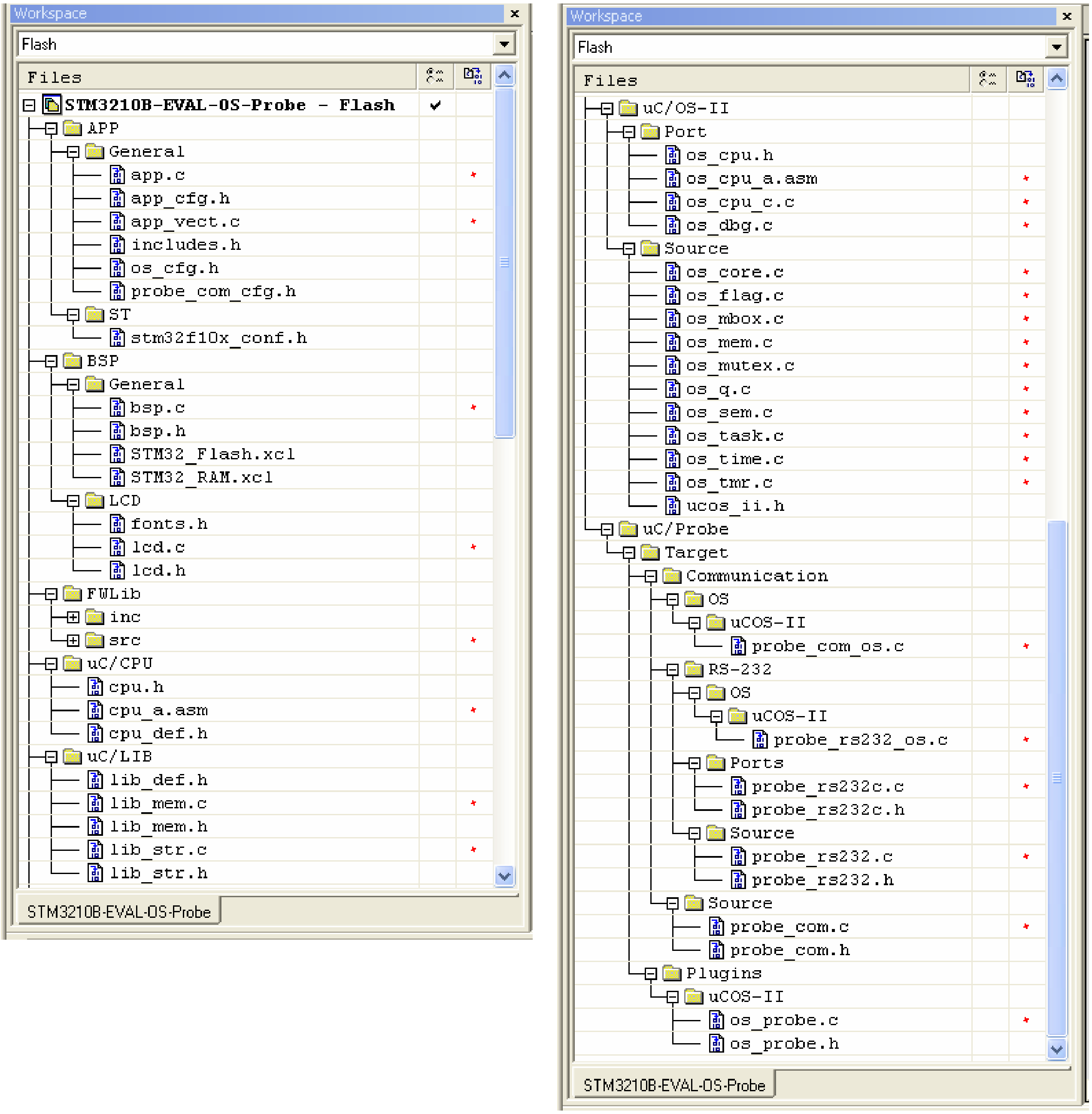


Figure 2-7. IAR EWARM Project Tree
for *STM3210B-EVAL-OS-Probe.ewp*.

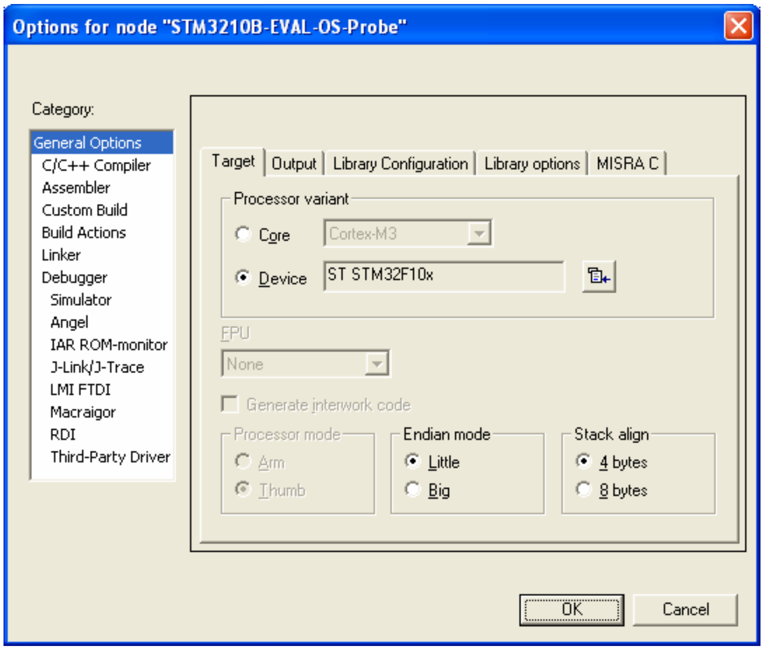


Figure 2-8. Project Options: General Options

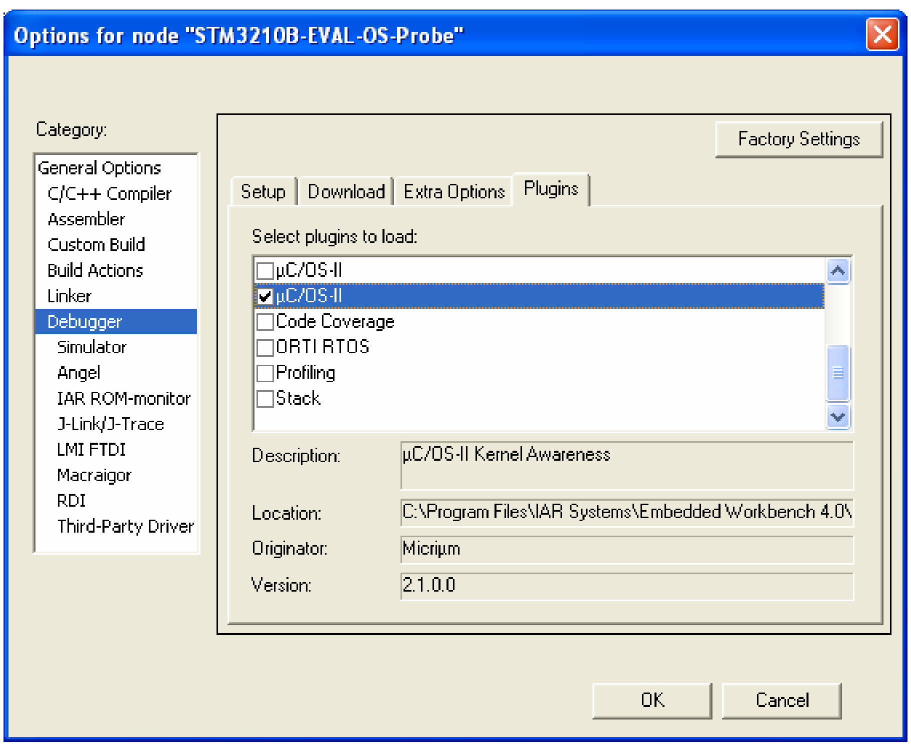


Figure 2-9. Enabling the μC/OS-II Kernel Awareness Plug-In

When the code is reloaded onto the evaluation board, the "μC/OS-II" menu should appear. Options are included to display lists of kernel objects such as semaphores, queues, and mailboxes, including for each entry the state of the object. Additionally, a list of the current tasks may be displayed, including for each task pertinent information such as used stack space, task status, and task priority, in addition to showing the actively executing task. An example task list for this project is shown in Figure 2-10.

Task List

	Name	Ref	Prio	State	Dly	Waiting On	Msg	Ctx Sw	Stk Ptr	Max%	Cur%	Max	Cur	Size	Starts @	Ends @
	Start Task	3	3	Dly	3			91	20000390	17%	11%	176	116	1024	20000404	20000004
	Keyboard	7	4	Dly	1			284	20000B8C	11%	11%	120	120	1024	20000C04	20000804
>	User I/F	6	5	Ready	0			215	20000740	20%	19%	212	196	1024	20000804	20000404
	Probe Str	8	6	Dly	33			5	20000F94	10%	10%	112	112	1024	20001004	20000C04
	Probe RS-232	4	7	Sem	0	Probe RS-232		1	20002ED0	11%	11%	120	120	1024	20002F48	20002B48
	uC/Probe OS	5	8	Ready	0			42	20002938	13%	13%	136	136	1024	200029C0	200025C0
	uC/OS-II Tmr	2	29	Ready	0			33	20002434	24%	24%	124	124	512	200024B0	200022B0
	uC/OS-II Stat	1	30	Ready	0			22	20001B58	21%	21%	108	108	512	20001BC4	200019C4
	uC/OS-II Idle	0	31	Ready	0			145	20001D80	14%	13%	72	68	512	20001DC4	20001BC4

Figure 2-10. µC/OS-II Task List for *STM3210B-EVAL-OS-Probe.ewp.*

2.05 Example Applications

Both the STM32-SK and STM3210B-EVAL example applications contain application tasks which respond to the push buttons, update the LED, and toggle the LEDs. In addition, either can be used with the Micriµm's real-time monitor, **µC/Probe**, as covered in Section 6.

2.05.01 STM32-SK Application Information

When the example application is started, a summary of the current **µC/OS-II** state is displayed on the LCD screen (as shown in Figure 2-11). Successive presses of PB1 will progress the LCD through a series of screens, shown in Figures 2-12 through 2-14. If the push button is pressed after Screen 4, shown in Figure 2-14 is reached, Screen 1 will again be shown.

Screen 2 displays the version of **µC/OS-II** currently running on the target and its tick timer frequency. Screen three shows the percent CPU usage and CPU clock speed. Two cumulative measures are shown in Screen 4: the number of ticks and the number of context switches that have occurred since **µC/OS-II** began running.

Pressing PB2 will toggle the LCD backlight, and the potentiometer controls the rate of movement of the LEDs.

Figure 2-11, Screen 1

Figure 2-12, Screen 2:
µC/OS-II Version Number and Tick Rate

Figure 2-13, Screen 3:
CPU Usage and CPU Speed

Figure 2-14, Screen 4:
Cumulative Ticks and Context Switches

2.05.01 STM3210B-EVAL/STM3210E-EVAL Application Information

When the example application is started, a summary of the current **µC/OS-II** state is displayed on the LCD screen (as shown in Figure 2-15). Pressing the push button labelled "Key" will cause the screen shown in Figure 2-16 to appear, which displays the list of **µC/OS-II** tasks.

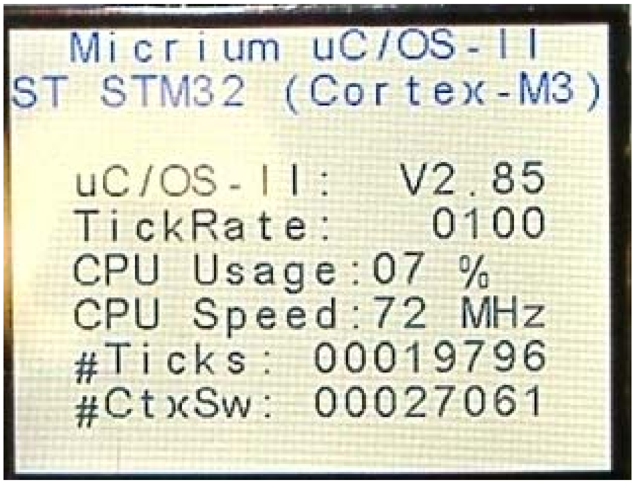


Figure 2-15, Screen 1

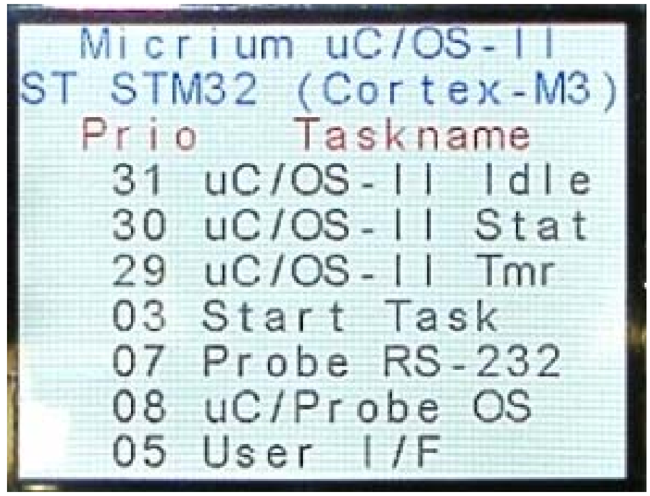


Figure 2-16, Screen 2:
μC/OS-II Task List

2.05.03 Additional Application Information

The project is configured so that code is loaded into Flash and the stacks and data are loaded into RAM, as shown in Table 2-1. The tasks that run in the example application are listed in Table 2-2.

Memory Range	Size	Segment(s)
`0x08000000-0x0001FFFF`	128 kB	Code (in Flash)
`0x20000000-0x20004FFF`	20 kB	Stacks and data (in RAM)

Table 2-1. Memory Setup

Task Name	Priority	Function
`AppTaskStart()` "Start Task"	3	Initializes **μC/TCP-IP**; toggles LEDs.
`AppTaskKbd()` "Keyboard"	4	Reads status of push buttons and joystick, passing new input to `AppTaskUserIF()`
`AppTaskUserIF()` "User I/F"	5	Updates the LCD.
"Probe RS-232"	7	Parses packets from **μC/Probe**.
"uC/Probe OS"	8	Updates CPU usage for **μC/Probe**.
"uC/OS-II Tmr"	29	Manages timers.
"uC/OS-II Stat"	30	Collect stack usage statistics.
"uC/OS-II Idle"	31	Executes when no other task is executing.

Table 2-2. Example Application Tasks

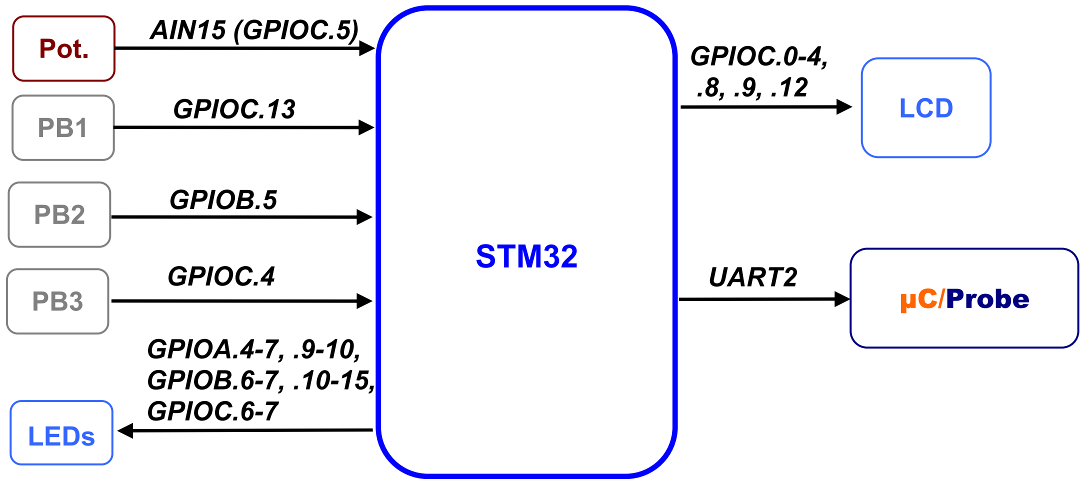


Figure 2-17. Example Application Hardware Use (STM32-SK)

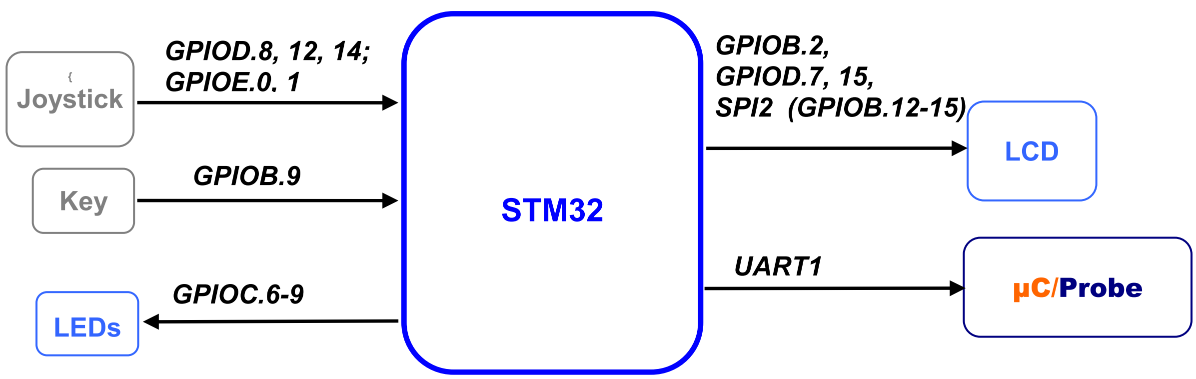


Figure 2-18. Example Application Hardware Use
(STM3210B-EVAL)

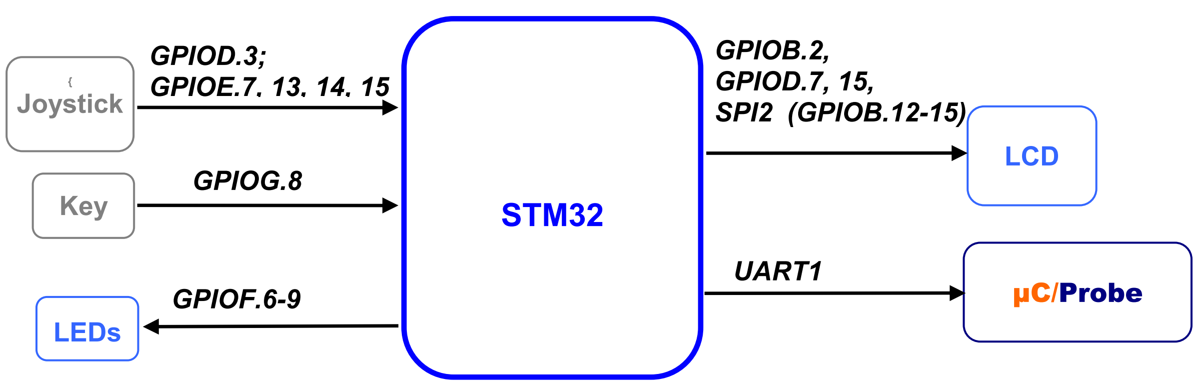


Figure 2-19. Example Application Hardware Use
(STM3210E-EVAL)

3. 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\AN1320-uCOS-II-ST-STM32

This directory contains this application note, *AN-1320.pdf*.

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-Cortex-M3\Generic\IAR
\Micrium\Software\uCOS-II\Ports\ARM-Cortex-M3\Generic\RealView

This directory contains the standard processor-specific files for the generic **μC/OS-II** ARM Cortex-M3 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_dbg.c*

The ARM Cortex-M3 port is described in application note *AN-1018* which is available from the Micriμm 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\ST\STM32

This directory contains ***probe_rs232c.c*** and ***probe_rs232c.h***, the STM32 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.

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-Cortex-M3\IAR
\Micrium\Software\uC-CPU\ARM-Cortex-M3\RealView

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 ARM Cortex-M3 processors which is used to enable and disable interrupts within the operating system. This code is called from C with `CPU_CRITICAL_ENTER()` and `CPU_CRITICAL_EXIT()`.

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()`, `memcpy()` and `memcmp()`. These functions are replaced by `Mem_Clr()`, `Mem_Set()`, `Mem_Copy()` and `Mem_Cmp()`, respectively. The reason we declared our own functions is for third party certification purposes.

The files ***lib_str.c*** and ***lib_str.h*** contain code to replace the standard library functions `str???()` with their equivalent `Str_???()` functions. Again, this is to simplify 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\ST\STM32-SK\IAR\OS-Probe-LCD

This directory contains the source code for the example application for the IAR STM32-SK evaluation board with IAR EWARM project files :

- ***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 LCD). ***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_vect_v5.c*** defines the ARM Cortex-M3 exception vector table for the application code. This vector table would be different for each application based on the interrupt service routines needed in that application.
- ***includes.h*** is the master include file used by the application.
- ***os_cfg.h*** is the **μC/OS-II** configuration file.
- ***probe_com_cfg.h*** is the **μC/Probe** target communications module configuration file.
- ***STM32-SK-OS-Probe-LCD-v5.**** are the IAR Embedded Workbench v5.11 project files.

\Micrium\Software\EvalBoards\ST\STM3210B-EVAL\IAR\OS-Probe

This directory contains the source code for the example application for the ST STM3210B-EVAL evaluation board with IAR EWARM project files :

- ***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 LCD). ***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_vect_v5.c*** defines the ARM Cortex-M3 exception vector table for the application code. This vector table would be different for each application based on the interrupt service routines needed in that application.
- ***includes.h*** is the master include file used by the application.
- ***os_cfg.h*** is the **μC/OS-II** configuration file.
- ***probe_com_cfg.h*** is the **μC/Probe** target communications module configuration file.
- ***STM3210B-EVAL-OS-Probe-v5.**** are the IAR Embedded Workbench v5.11 project files.

\Micrium\Software\EvalBoards\ST\STM3210E-EVAL\IAR\OS-Probe

This directory contains the source code for the example application for the ST STM3210E-EVAL evaluation board with IAR EWARM project files :

- ***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 LCD). ***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_vect_v5.c*** defines the ARM Cortex-M3 exception vector table for the application code. This vector table would be different for each application based on the interrupt service routines needed in that application.
- ***includes.h*** is the master include file used by the application.
- ***os_cfg.h*** is the **μC/OS-II** configuration file.
- ***probe_com_cfg.h*** is the **μC/Probe** target communications module configuration file.
- ***STM3210E-EVAL-OS-Probe-v5.**** are the IAR Embedded Workbench v5.11 project files.

\Micrium\Software\CPU\ST\STM32

This directory contains the ST driver library for the STM32 microprocessors.

- ***/src/*.c*** is the driver library source code.
- ***/inc/*.h*** are header files containing prototypes for the library API.

4. Application Code

The example application described in this appnote, *AN-1320*, is a simple demonstration of **µC/OS-II** and **µC/Probe** for the STMicroelectronics processors with IAR STM32-SK, ST STM3210B-EVAL and ST STM3210E-EVAL evaluation boards.. The basic procedure for setting up and using each of these can be gleaned from an inspection of the application code contained in *app.c*, which should serve as a beginning template for further use of these software modules. Being but a basic demonstration of software and hardware functionality, this code will make evident the power and convenience of **µC/OS-II** "The Real-Time Kernel" used on the STMicroelectronics STM32 processor without the clutter or confusion of a more complex example.

4.01 *app.c*

Five 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, `AppTaskStart()`, begins multitasking, and exits.

2. **`AppTaskStart()`**, after creating the user interface tasks, enters an infinite loop in which it blinks the LED on the board based on the state of the push buttons.

3. **`AppTaskUserIF()`** writes information to the LCD

4. **`AppTaskKbd()`** monitors the current state of the push button. When the push button is pressed, this task will post a message to **`AppTaskUserIF()`**, which will advance the LCD to the next screen.

```
int  main (void)                                                   /* Note 1 */
{
    CPU_INT08U  err;


    BSP_IntDisAll();                                               /* Note 2 */

    OSInit();                                                      /* Note 3 */

    OSTaskCreateExt((void (*)(void *)) App_TaskStart,                      /* Note 4 */
                    (void          * ) 0,
                    (OS_STK        * )&App_TaskStartStk[APP_TASK_START_STK_SIZE - 1],
                    (INT8U           ) APP_TASK_START_PRIO,
                    (INT16U          ) APP_TASK_START_PRIO,
                    (OS_STK        * )&App_TaskStartStk[0],
                    (INT32U          ) APP_TASK_START_STK_SIZE,
                    (void          * )0,
                    (INT16U          )(OS_TASK_OPT_STK_CLR | OS_TASK_OPT_STK_CHK));

#if (OS_TASK_NAME_SIZE > 11)                                       /* Note 5 */
    OSTaskNameSet(APP_TASK_START_PRIO, "Start Task", &err);
#endif

    OSStart();                                                     /* Note 6 */
}
```

Listing 4-1, `main()`

Listing 4-1, Note 1: As with most C applications, the code starts in `main()`.

Listing 4-1, Note 2: All interrupts are disabled to make sure the application does not get interrupted until it is fully initialized.

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

Listing 4-1, Note 4: 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 4-1, Note 5: As of V2.6x, you can now name **μC/OS-II** tasks (and other kernel objects) and display task names at run-time or with a debugger. In this case, the `AppTaskStart()` 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 4-1, Note 6: Finally multitasking under **μC/OS-II** is started by calling `OSSTart()`. **μC/OS-II** will then begin executing `AppTaskStart()` since that is the highest-priority task created (both `OS_TaskStat()` and `OS_TaskIdle()` having lower priorities).

```
static  void  AppTaskStart (void *p_arg)
{
    CPU_INT32U  i;
    CPU_INT32U  j;


    (void)p_arg;


    BSP_Init();                                           /* Note 1 */
    OS_CPU_SysTickInit();                                 /* Note 2 */

#if (OS_TASK_STAT_EN > 0)
    OSStatInit();                                         /* Note 3 */
#endif

#if ((APP_PROBE_COM_EN == DEF_ENABLED) || \               /* Note 4 */
     (APP_OS_PROBE_EN  == DEF_ENABLED))
    App_InitProbe();
#endif

    AppEventCreate()                                      /* Note 5 */
    AppTaskCreate();

    while (DEF_TRUE) {                                    /* Note 6 */

        /* Toggle LEDs */
   }
}
```

Listing 4-2, AppTaskStart()

Listing 4-2, Note 1: BSP_Init() initializes the Board Support Package—the I/Os, tick interrupt, etc. See Section 5 for details.

Listing 4-2, Note 2: OS_CPU_SysTickInit() initializes Cortex-M3 SysTick timer, which generates the **µC/OS-II** time tick.

Listing 4-2, 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 4-2, Note 4: App_InitProbe() initializes the **µC/Probe** plug-in for **µC/OS-II**, which maintains CPU usage statistics for each task, and the **µC/Probe** generic communication module, including 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 4-2, Note 5: The even used in the application is created, a mailbox that provides the communication between the two application tasks, App_TaskKbd() and App_TaskUserIF(). When the push button is pressed, App_TaskKbd() will send a message using App_UserIFMbox to App_TaskUserIF() containing the new state of the display. The two application tasks are then created.

Listing 4-2, Note 6: 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 an LED is toggled if one of the push buttons is pressed.

4.02 *app_vect.c* or *vector.s*

Either ***app_vect.c*** (on IAR EWARM) or ***vector.s*** (on Keil μVision/RVMDK) contains the exception vector table for the ARM Cortex-M3. We only filled in the first 64 entries because the STM32 only provides up to this many vectors. However, this table can contain up to 256 elements (pointers). *app_vect.c* or *vector.s* is part of the application code and not the BSP because the vector table can change based on how many interrupts the application has and the use of those interrupts.

4.03 *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 Jean Labrosse's book, ***μC/OS-II, The Real-Time Kernel, 2nd Edition***. *os_cfg.h* assumes you have **μC/OS-II** V2.83 or higher but also works with previous versions of **μC/OS-II**.

- **`OS_APP_HOOKS_EN`** is set to 1 so that the cycle counters in the `OS_TCB`s will be maintained.

- 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 31, allowing up to 32 total tasks.

- **`OS_MAX_TASKS`** determines the number of "application" tasks and is currently set to 16 allowing 7 more tasks to be added to the example code.

- **`OS_TICKS_PER_SEC`** is set to 100 Hz. This value can be changed as needed and the proper tick rate will be adjusted in *bsp.c* if you change this value. You would typically set the tick rate betweek 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.

5. 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 the STM32 (or its evaluation board). Though one file, *bsp.c*, contains some functions which are intended to be called directly by the user (all of which are prototyped in *bsp.h*), the other files serve the compiler (as with *STM32_Flash.xcl*).

5.01 IAR-Specific BSP Files

The BSP includes one file intended specifically for use with IAR tools: ***STM32_Flash.xcl***. This serves to define the memory map of the processor. If the example application is to be used with other toolchains, the services provided by this file must be replicated as appropriate.

Before the processor memories can be programmed, the compiler must know where code and data should be placed. IAR requires a linker command file, such as *STM32_Flash.xcl*, that provides directives to accomplish this. All data, stack and heap segments are placed in the 20-kB internal RAM between `0x20000000` and `0x20004FFF`. Code is loaded into the 128-kB Flash between `0x08000000` and `0x0801FFFF`.

5.02 BSP, *bsp.c* and *bsp.h*

The file *bsp.c* implements several global functions, each providing some important service, be that the initialization of processor functions for **μC/OS-II** to operate or the toggling of an LED. Several local functions are defined as well to perform some atomic duty, initializing the I/O for the LED or initialize the **μC/OS-II** tick timer. The discussion of the BSP will be limited to the discussion of the global functions that might be called from user code (and may be called from the example application).

The global functions defined in *bsp.c* (and prototyped in *bsp.h*) may be roughly divided into two categories: critical processor initialization and user interface services. Two functions constitute the former:

- **`BSP_Init()`** is called by the application code to initialize critical processor features (particularly the **μC/OS-II** tick interrupt) after multitasking has started (i.e., `OS_Start()` has been called). This function should be called before any other BSP functions are used. See Listing 5-1 for more details.
- **`BSP_IntDisAll()`** is called to disable all interrupts, thereby preventing any interrupts until the processor is ready to handle them.

For the IAR STM32-SK, eight functions provide access to the user interface components:

- **`BSP_LED_Toggle()`**,**`BSP_LED_On()`** and **`BSP_LED_Off()`** will toggle, turn on, and turn off (respectively) the LED corresponding to the ID passed as the argument (which can be between 1 and 16 for the STM32-SK). If an argument of 0 is provided, the appropriate action will be performed on all LEDs.
- **`BSP_PB_GetStatus()`** takes as its argument the ID (on the STM32-SK, 1 to 3) of a push button and returns `DEF_TRUE` if the push button is being pressed and `DEF_FALSE` if the push button is not being pressed.

- **BSP_LCD_LightToggle()**, **BSP_LCD_LightOn()** and **BSP_LCD_LightOff()** will toggle, turn on, and turn off (respectively) the LCD backlight.
- **BSP_ADC_GetStatus()** takes as its argument the ID (on the STM32-SK, 1) of a analog input and returns the current conversion value. On the STM32-SK, an ID of 1 corresponds to the potentiometer output.

For the ST ST3210B-EVAL/STM3210E-EVAL, six functions provide access to the user interface components:

- **BSP_LED_Toggle()**, **BSP_LED_On()** and **BSP_LED_Off()** will toggle, turn on, and turn off (respectively) the LED corresponding to the ID passed as the argument (which can be between 1 and 4 for the ST3210B-EVAL). If an argument of 0 is provided, the appropriate action will be performed on all LEDs.
- **BSP_PB_GetStatus()** takes as its argument the ID (on the ST3210B-EVAL/STM3210E-EVAL, 1) of a push button and returns `DEF_TRUE` if the push button is being pressed and `DEF_FALSE` if the push button is not being pressed.
- **BSP_ADC_GetStatus()** takes as its argument the ID (on the ST3210B-EVAL/STM3210E-EVAL, 1) of a analog input and returns the current conversion value. On the STM32-SK, an ID of 1 corresponds to the potentiometer output.
- **BSP_Joystick_GetStatus()** returns the status of the joystick.

5.03 Processor Initialization Function

```
void  BSP_Init (void)
{
                                                            /* Note 1 */
    RCC_DeInit();
    RCC_HSEConfig(RCC_HSE_ON);
    RCC_WaitForHSEStartUp();


    RCC_HCLKConfig(RCC_SYSCLK_Div1);
    RCC_PCLK2Config(RCC_HCLK_Div1);
    RCC_PCLK1Config(RCC_HCLK_Div2);
    RCC_ADCCLKConfig(RCC_PCLK2_Div6);
    FLASH_SetLatency(FLASH_Latency_2);
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
    RCC_PLLCmd(ENABLE);

    while (RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET) {
        ;
    }

    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);

    while (RCC_GetSYSCLKSource() != 0x08) {
        ;
    }

       /* Initialize board I/Os, ADCs, LCD, etc. */        /* Note 2 */

}
```

Listing 5-1, `BSP_Init()`

Listing 5-1, Note 1: The CPU clock (which is also the peripheral clock) is setup. With the settings here specified, the PLL clock will equal the on-board oscillator (with a frequency output of 8 MHz) divided by 1 and multiplied by 9, resulting in a 72 MHz clock.

Listing 5-1, Note 2: The peripherals for the user interface components are initialized. This includes setting up the I/O pins for the PBs and LEDs and (if applicable) initializing an LCD or ADC inputs.

6. 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.

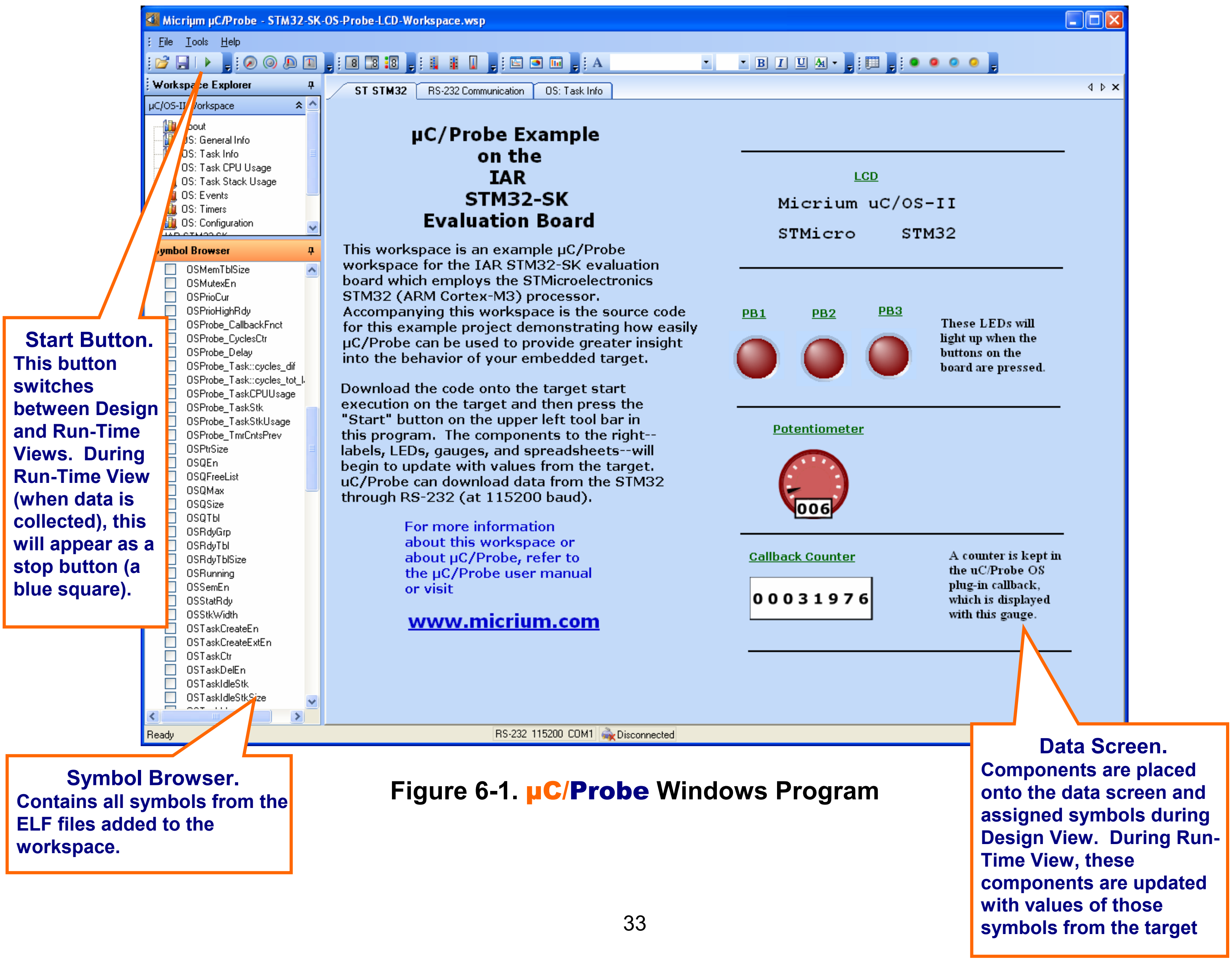


Figure 6-1. µC/Probe Windows Program

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

   You can also open one of the example workspaces that accompanies this appnote, which are located in the project directories.

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.) For IAR EWARM, this ELF 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*** unless you specify otherwise.

   In Keil μVision3, the ELF file will either be in the same directory as the ****.UV2*** project file or in a ***/rvmdk*** subdirectory and will have a ****.axf*** extension; the name will be the name configured in the "Output" tab of the target options dialog. For the workspaces accompanying this appnote, the name of the output is the same as the name of the workspace file.

   To load this ELF file, right-click on the symbol browser and choose "Add Symbols". Navigate to the file directory, select the file, and choose "OK".

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.

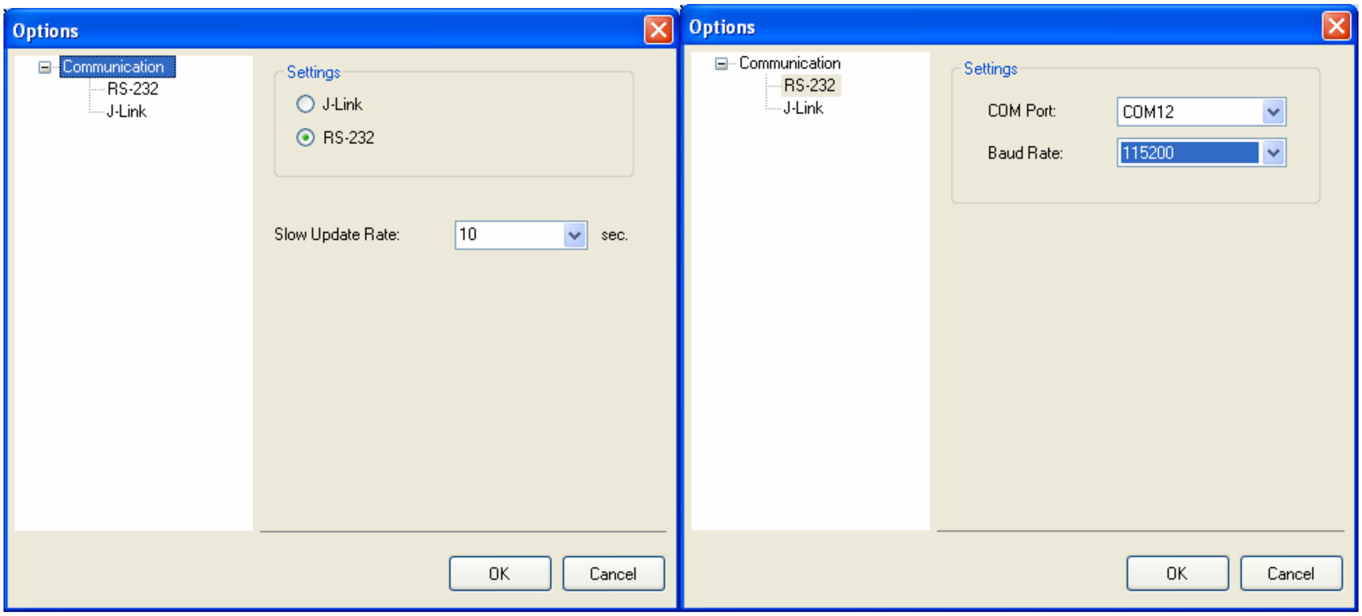


Figure 6.2. μC/Probe Options

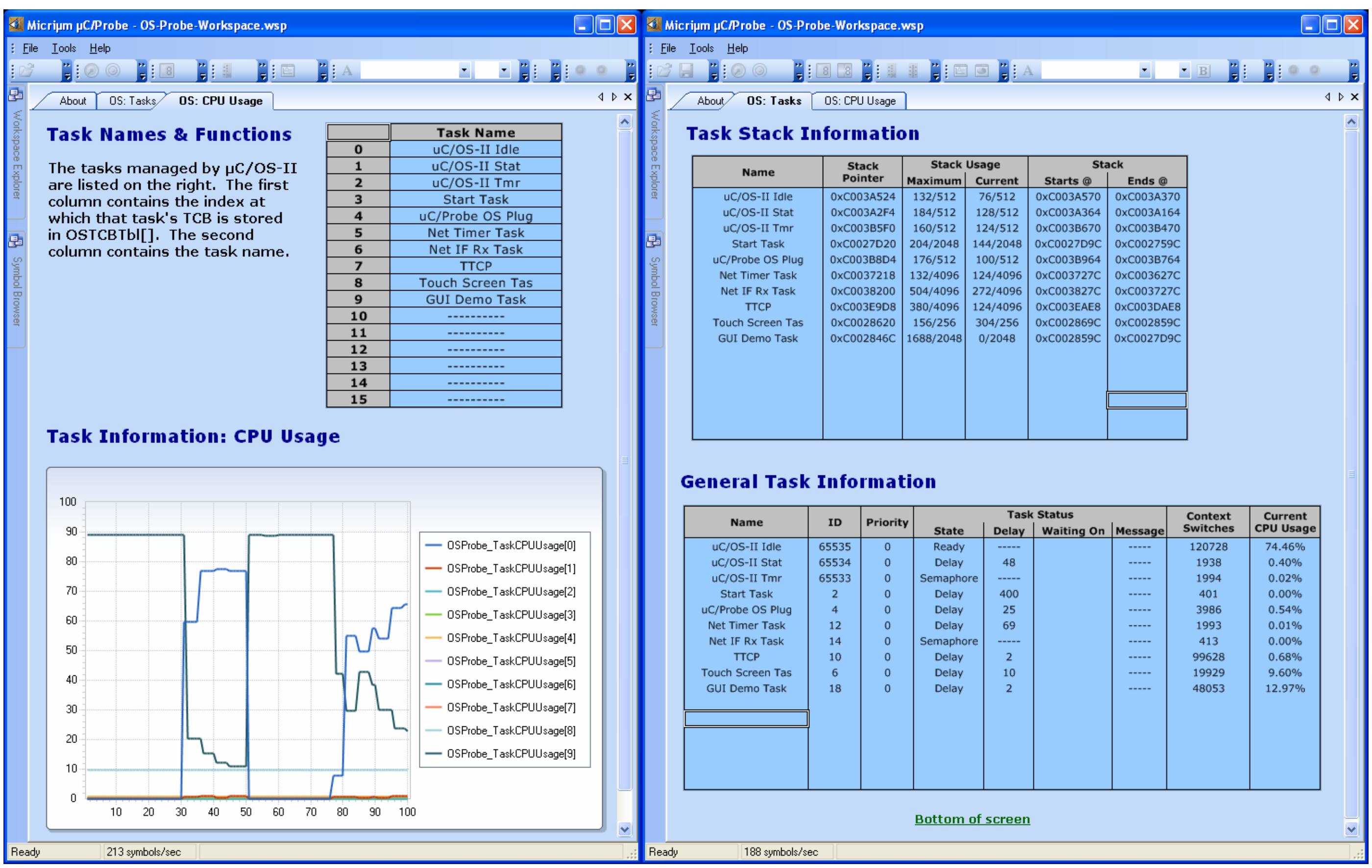


Figure 6-3. μC/Probe Run-Time: μC/OS-II Task Information

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 : http://www.IAR.com

Micriμm
949 Crestview Circle
Weston, FL 33327
USA

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

e-mail: Jean.Labrosse@Micrium.com
WEB : http://www.Micrium.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

ST Microelectronics
39, Chemin du Champ des Filles
C. P. 21
CH 1228 Plan-Les-Ouates
Geneva, Switzerland

+41 22 929 29 29
+41 22 929 29 00 (FAX)

WEB : http://www.st.com