Micriμm

Empowering Embedded Systems

μC/OS-II

μC/Probe

and the

NXP® LPC2478

(Using the IAR LPC2478 SK board)

Application Note

AN-1479

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.

About µC/Probe Demo Version

µC/Probe is a Windows application that allows a user to display and change 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 grouping different 'views' into a product.

This **µC/Probe** demo version can only retrieve information from RS-232C or J-LINK interfaces and is limited up to 15 symbols.

The demo version of **µC/Probe** is available on the Micriµm website:

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

About µC/Probe Full Version

The full version of **µC/Probe** allows you to use a TCP/IP is a Windows application that allows a user to display and change 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.

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	2008/10/22	FT	Initial version.

Software Versions

This document may or may not have been downloaded as part of an executable file, *Micrium-NXP-uCOS-II-LPC2478-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	V2.86	
μC/Probe	V2.20	

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-1479***, explains example code for using **μC/OS-II** and **μC/OS-Probe** with the IAR LPC2478-SK Development board, as shown in Figure 1-1, which employs NXP's ARM7TDMI-based LPC2478 microcontroller. The processor includes 512 kB on-chip flash memory and 64-kB SRAM in addition to dedicated SRAM for the EMAC and DMA peripherals. Additionally, the chip includes serial interfaces such as an internal 10/100 EMAC, USB device and host (with support for an external OTG transceiver), two CAN channels, a SPI controller, two SSP controllers, four UARTs, and several I^2C and I^2S interfaces . Additionally, the chip has a SD/MMC card interface, many general purpose I/O pins, and a 10-bit A/D converter.

The IAR LPC2478-SK board includes the following peripherals:

- LPC2478 device
- LCD 3.5" 320x200 24bit color TFT with backlight and touch screen
- MP3 decoder DSP + codec VS1002D
- 3-axis digital accelerometer with 11 bit accuracy
- 64M SDRAM
- USB host connector
- USB device connector
- IrDA transceiver
- PS2 keyboard connector
- 100 Mbit Ethernet
- CAN driver and connector
- RS232 with ICSP control
- SD/MMC card connector
- JTAG connector
- MICTOR TRACE connector
- Reset button
- 2 user buttons
- Trim pot
- UEXT connector
- Audio IN
- Audio OUT
- RTC battery
- RoHS

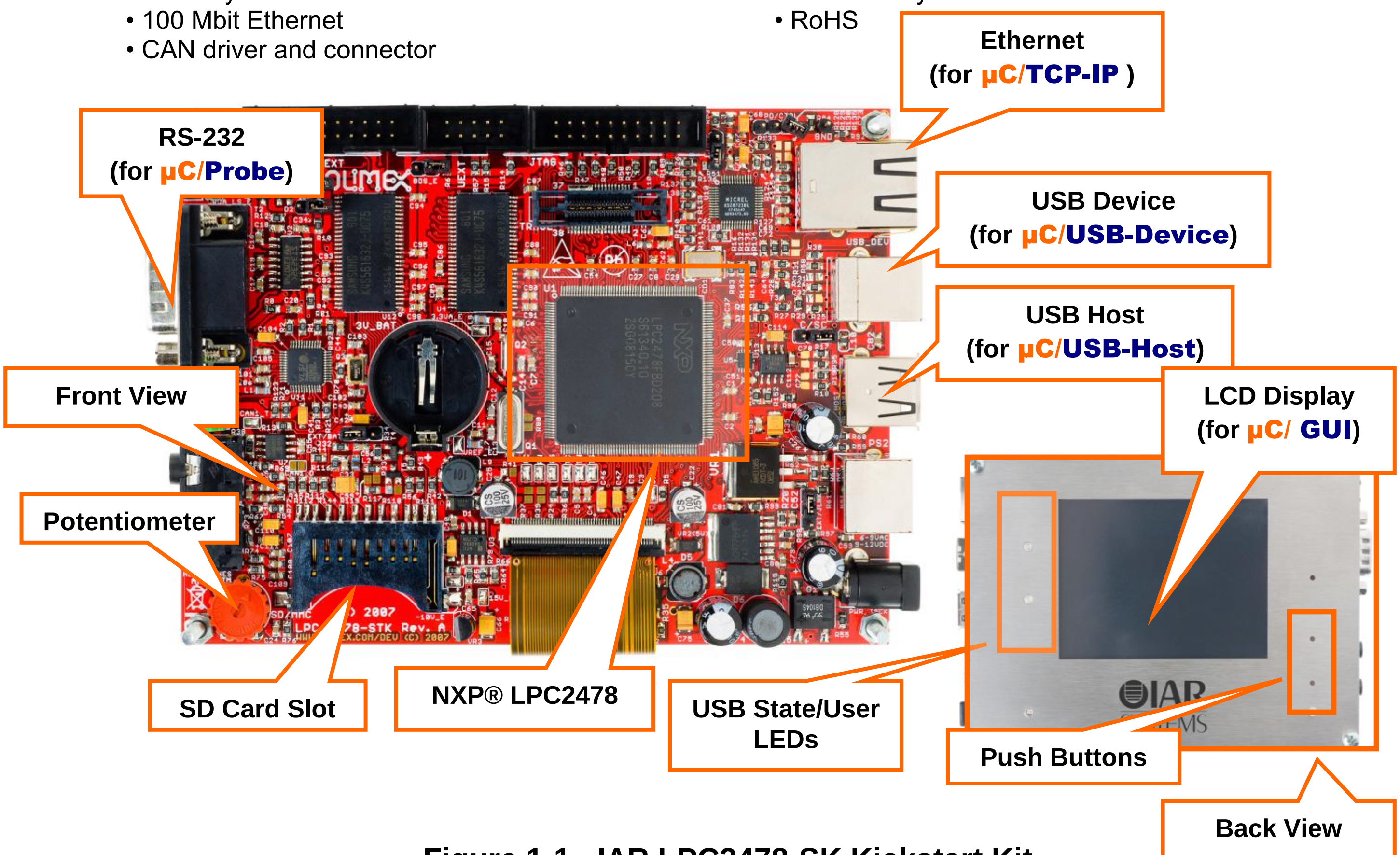


Figure 1-1. IAR LPC2478-SK Kickstart Kit

If this appnote was downloaded in a packaged executable zip file, then it should have been found in the directory ***Micrium/AppNotes/AN1xxx-RTOS/AN1479-uCOS-II-NXP-LPC2478-SK*** 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 IAR LPC2478 via RS-232C. For more information, including instructions for downloading a trial and the demo version of the program, please refer to Section 6.

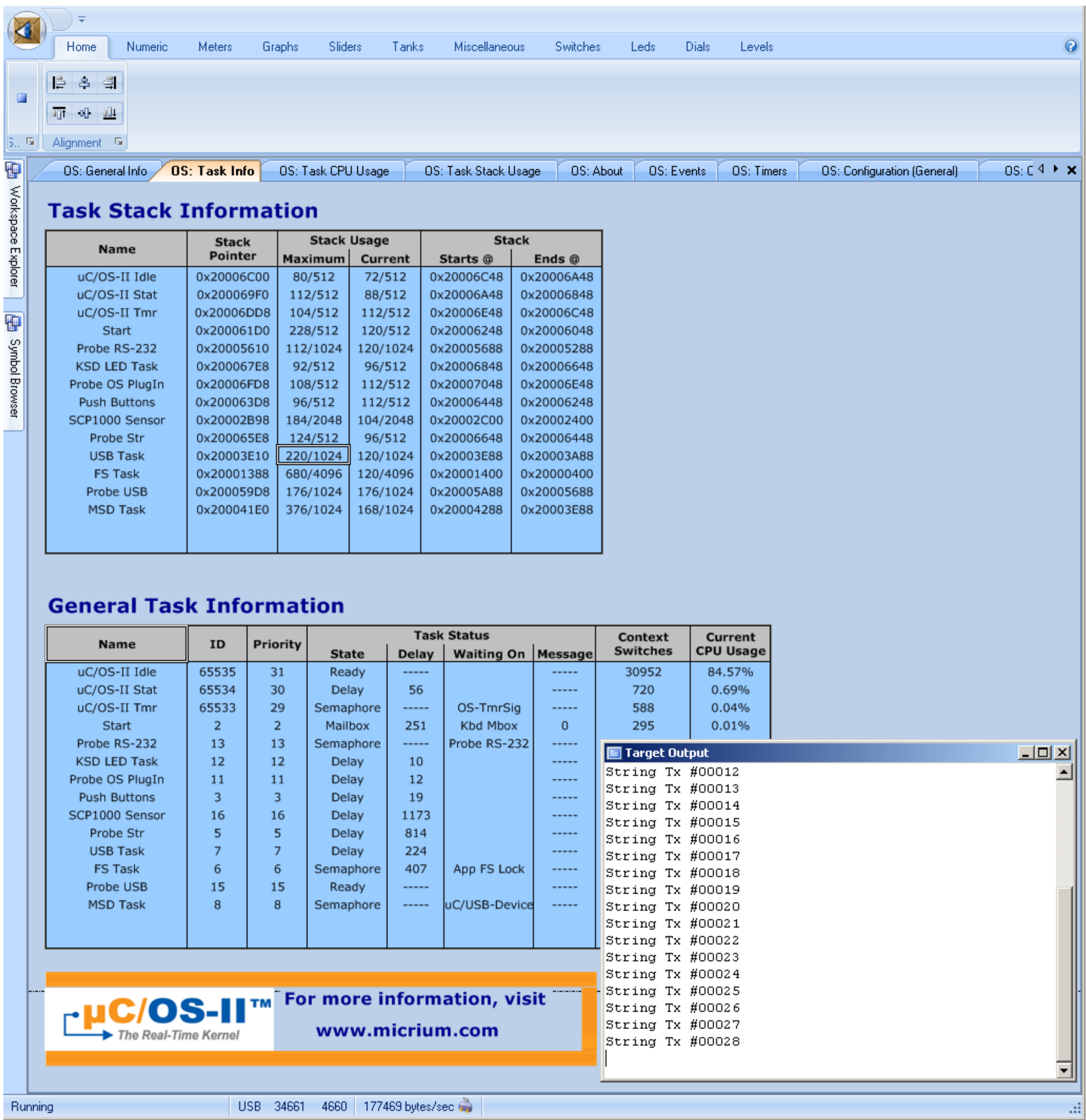


Figure 1-2. μC/Probe (with Target Output Window)

2. Getting Started

The following sections step through the prerequisites for using the demonstration application described in this document, ***AN-1479*** .First, the setup of the hardware will be outlined. Second, the use and setup of the IAR Embedded Workbench project 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

2.01.01 Powering the board.

The IAR LPC2478-SK board can be power up using three different sources:

- 6- 9V AC External power adapter.
- 9-12V DC External power adapter.
- Through J-Link.

2.01.02 Using μC/Probe

If **μC/Probe** is being used then connect the RS-232 cable to the port labeled "RS-232 for **μC/Probe**" in Figure 1-1.

2.02 Directory Tree

If this file were downloaded as part of an executable zip file (which should have been named *Micrium-NXP-uCOS-II-LPC2478-SK.exe*) then the code files referred to herein are located in the directory structure shown in Figure 2-2.

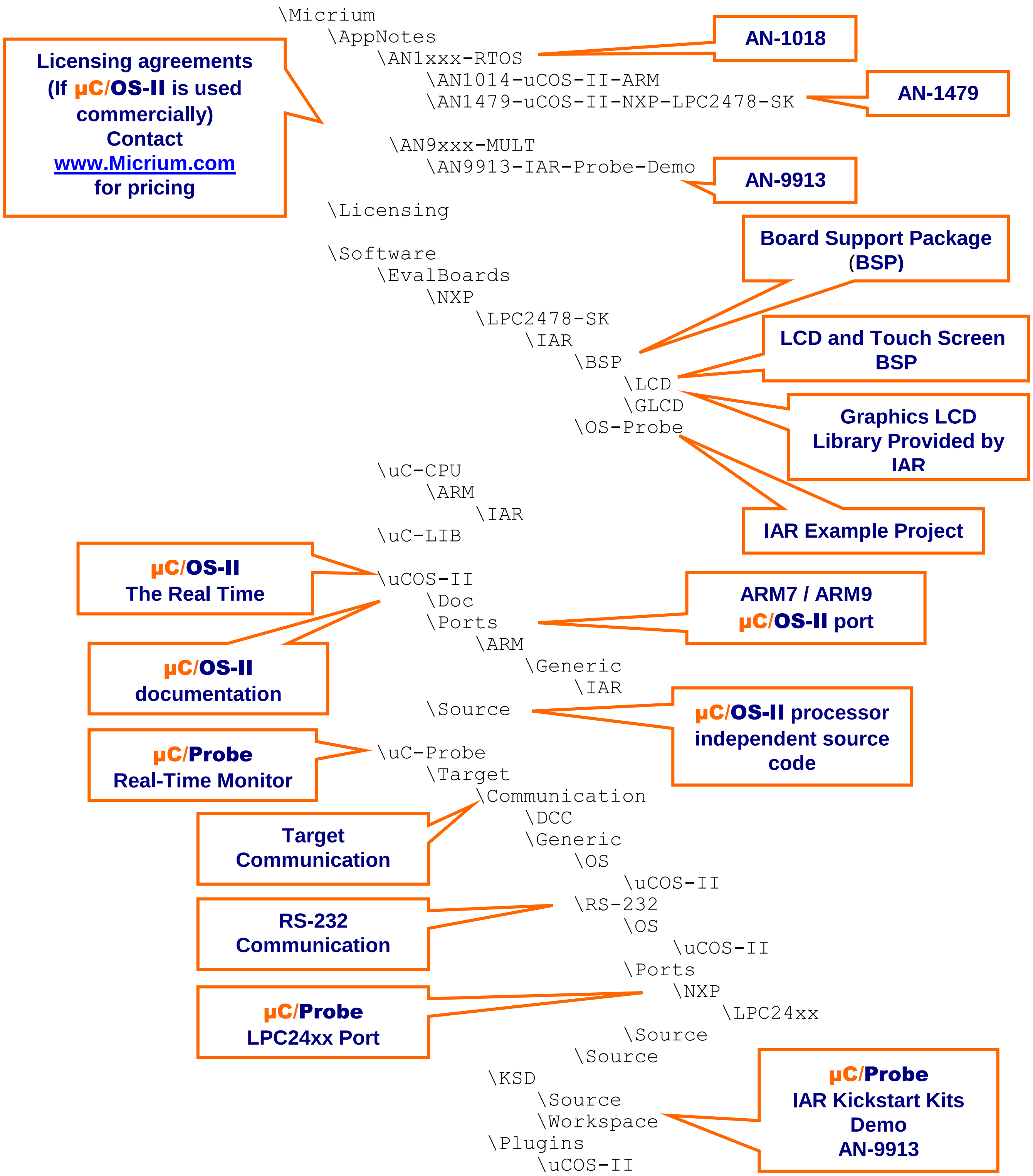


Figure 2-1. Directory Structure

2.03 Using the IAR Projects

An IAR projects is located in the directory marked "IAR Example Project " in Figure 2-1:

\Micrium\Software\EvalBoards\NXP\IAR-LPC2478\IAR\LPC2478-SK

The example project, ***LPC2478-SK-OS-Probe-v5-2.ewp***, is intended for EWARM v5.2x. To view this example, start an instance of IAR EWARM v5.2x, and open the workspace file ***LPC2478-SK-OS-Probe-v5-2.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.

2.03.01 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 2-4.

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 2-5.

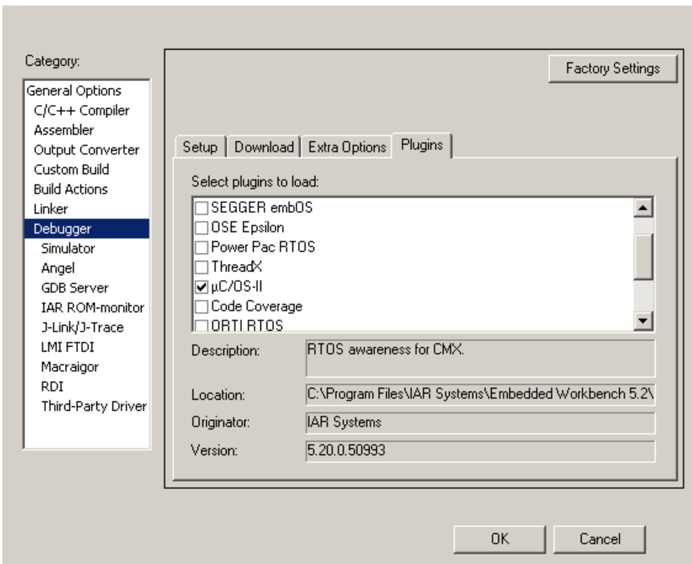
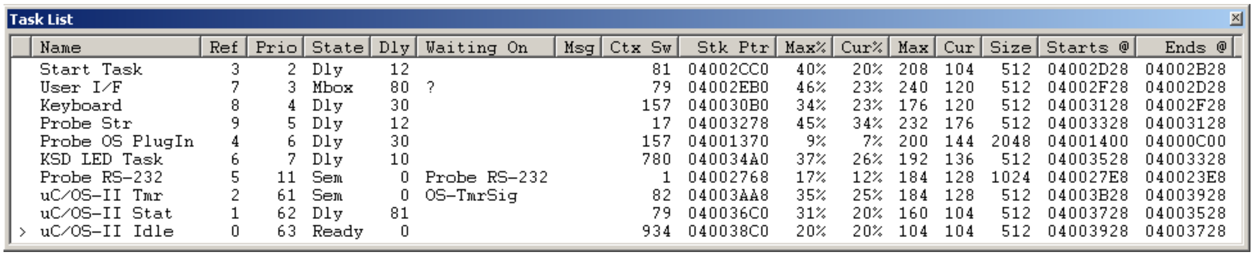


Figure 2-2. Enabling the μC/OS-II Kernel Awareness Plug-In



Task List

	Name	Ref	Prio	State	Dly	Waiting On	Msg	Ctx Sw	Stk Ptr	Max%	Cur%	Max	Cur	Size	Starts @	Ends @
	Start Task	3	2	Dly	12			81	04002CC0	40%	20%	208	104	512	04002D28	04002B28
	User I/F	7	3	Mbox	80	?		79	04002EB0	46%	23%	240	120	512	04002F28	04002D28
	Keyboard	8	4	Dly	30			157	040030B0	34%	23%	176	120	512	04003128	04002F28
	Probe Str	9	5	Dly	12			17	04003278	45%	34%	232	176	512	04003328	04003128
	Probe OS PlugIn	4	6	Dly	30			157	04001370	9%	7%	200	144	2048	04001400	04000C00
	KSD LED Task	6	7	Dly	10			780	040034A0	37%	26%	192	136	512	04003528	04003328
	Probe RS-232	5	11	Sem	0	Probe RS-232		1	04002768	17%	12%	184	128	1024	040027E8	040023E8
	uC/OS-II Tmr	2	61	Sem	0	OS-TmrSig		82	04003AA8	35%	25%	184	128	512	04003B28	04003928
	uC/OS-II Stat	1	62	Dly	81			79	040036C0	31%	20%	160	104	512	04003728	04003528
>	uC/OS-II Idle	0	63	Ready	0			934	040038C0	20%	20%	104	104	512	04003928	04003728

Figure 2-3. µC/OS-II Task List.

2.04 Example Applications

Once the program is loaded onto the target, the LEDs will begin blinking. The system state will be updated using the LCD display. There are several screens showing information related to the hardware and **µC/OS-II** as shown in Figure 2-4. To move to another item and the LCD the user should touch the 'Next' or the 'Prev' buttons.

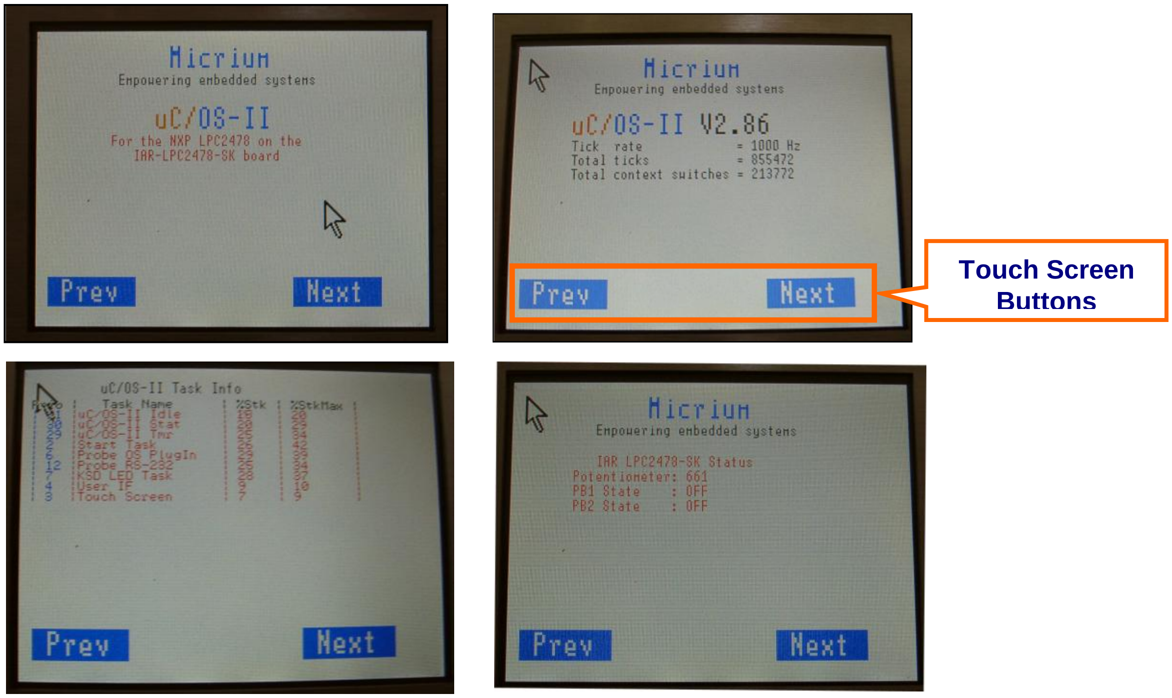


Figure 2-4. IAR LPC2478-SK LCD Output

Including the **µC/OS-II** system tasks, the example application includes several tasks, as listed in Table 2-1.

Task Name	Priority	Function
`App_TaskStart()` "Start Task"	2	Starts **μC/OS-Probe**; reads ADCs, blinks LEDs.
`App_TaskTouchScr()` "Touch Screen"	3	Monitors the state of the Touch Screen. Additionally, this task draws the cursor using the LPC2478 Cursor Controller capabilities and sends messages to the `App_TaskUserIF()`.
`App_TaskUserIF()` "User IF"	4	Outputs the system state through the LCD display
"Probe OS PlugIn"	6	Updates CPU usage for **μC/Probe**.
"KSD LED Task"	7	IAR Kickstart Kits Demo for the demo version of **μC/Probe.**
"Probe RS-232"	7	Parses packets from **μC/Probe**.
"uC/OS-II Tmr"	29	Manages **μC/OS-II** timers
"uC/OS-II Stat"	30	Collect stack usage statistics
"uC/OS-II Idle"	31	Executes when no other task is executing.

3. Directories and Files

Application Notes

\Micrium\AppNotes\AN1xxx-RTOS\AN1014-uCOS-II-ARM

This directory contains ***AN-1014.pdf***, the application note describing the ARM port for **μC/OS-II**, and ***AN-1014-PPT.pdf***, a supplement to *AN-1014.pdf*.

\Micrium\AppNotes\AN1xxx-RTOS\AN1479-uCOS-II-NXP-LPC2478-SK

This directory contains this application note, ***AN-1479.pdf***.

\Micrium\AppNotes\AN9xxx-MULT\AN-9913-IAR-Probe-Demo

This directory contains this application note, ***AN-9913.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\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\LPC24xx

This directory contains ***probe_rs232c.c*** and ***probe_rs232c.h***, the LPC24xx 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\Target\Demo\KSD\Source

This directory contains ***ksd.c*** and ***ksd.h,*** the source code for the IAR Kickstart kits demo example for the demo version of **µC/Probe**.

\Micrium\Software\uC-Probe\Target\Demo\KSD\Workspace

This directory contains ***OS-Probe-Kickstart-Demo-Workspace.wsp*** which is the generic **µC/Probe** workspace the IAR Kickstart kits demo example for the demo version of **µC/Probe**.

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.

\Micrium\Software\uC-LIB\Doc

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

Application Code

\Micrium\Software\EvalBoards\NXP\LPC2478-SK \IAR\OS-Probe

This directory contains the soruce code the **μC/OS-II** and **μC/Probe** 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, the ADC, the LCD 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.
- ***includes.h*** is the master include file used by the application.
- ***os_cfg.h*** is the **μC/OS-II** configuration file.
- ***LPC2478-SK-OS-Probe--Workspace.wsp*** is an example **μC/Probe** workspace.
- ***LPC2478-SK-OS-Probe-v5-2-v5-2.**** are the IAR EWARM v5.2x project files.

\Micrium\Software\EvalBoards\NXP\LPC2478-SK\IAR\BSP

This directory contains the Board Support Package for the IAR LPC2478 SK Kickstart Kit:

- ***bsp.c*** contains the board support package functions which initialize critical processor functions (e.g., the PLL) and provide support for peripherals such as the push buttons and LEDs.
- ***bsp.h*** contains prototypes for functions that may be called by the user.
- ***cstartup.s*** is the IAR EWARM v5.2x startup file. This file performs critical processor initialization (such as the initialization of task stacks), readying the platform to enter `main()`.
- ***LPC2478_Flash.icf*** is a IAR EWARM v5.xx linker file which contains information about the placement of data and code segments in the processor's memory map.
- ***LPC2478_Flash.mac*** contains instructions that are executed prior to loading code onto the processor.

4. Application Code

The example application described in this appnote, *AN-1479*, is a simple demonstration of **μC/OS-II** and **μC/Probe** for the NXP® LPC2478 processor on the IAR LPC2478-SK evaluation board. 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 NXP® LPC2478 processor without the clutter or confusion of a more complex example.

4.01 *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, `AppTaskStart()`, begins multitasking, and exits.

2. **`App_TaskStart()`**, after creating the user interface tasks, enters an infinite loop in which it blinks the LEDs on the board,

3. **`App_TaskTouchScr(),`** Monitors the state of the Touch Screen and draws the cursor pointer using the LPC2478 Cursor Controller Hardware.

4. **`App_TaskUserIF()`**,Outputs the state of the system based on the display state passed to it b**y** `App_TaskTouchScr()`**.**

```
void  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_CFG_TASK_START_STK_SIZE - 1],
                    (INT8U          ) APP_CFG_TASK_START_PRIO,
                    (INT16U         ) APP_CFG_TASK_START_PRIO,
                    (OS_STK        *)&App_TaskStartStk[0],
                    (INT32U         ) APP_CFG_TASK_START_STK_SIZE,
                    (void          *) 0,
                    (INT16U         )(OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));
#if OS_TASK_NAME_SIZE > 13                                         /* Note 5 */
    OSTaskNameSet(APP_CFG_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 `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 4-1, Note 6: 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)
{
    CPU_INT32U  i;
    CPU_INT32U  j;
    CPU_INT08U  err;


    (void)p_arg;

    BSP_Init();                                            /* Note 1 */

#if (OS_TASK_STAT_EN > 0)
    OSStatInit();                                          /* Note 2 */
#endif

#if (APP_CFG_PROBE_COM_MODULE_EN == DEF_ENABLED)           /* Note 3 */
    App_ProbeInit();
#endif

    App_UserIF_Scr = APP_USER_IF_SCR_FIRST;

    App_TaskCreate();                                      /* Note 4 */
    App_EventCreate();

    BSP_LED_Off(0);

    while (DEF_TRUE) {                                     /* Note 5 */
        BSP_LED_On(1);
        OSTimeDlyHMSM(0, 0, 0, 100);
        BSP_LED_Off(1);
        OSTimeDlyHMSM(0, 0, 0, 100);
        BSP_LED_On(2);
        OSTimeDlyHMSM(0, 0, 0, 100);
        BSP_LED_Off(2);
        OSTimeDlyHMSM(0, 0, 0, 100);

     }
}
```

Listing 4-2, `App_TaskStart()`

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: `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 3: `App_ProbeInit()` initialize **μC/Probe**. This function calls `OSProbe_Init()` which initializes the **μC/Probe** plug-in for **μC/OS-II**, which maintains CPU usage statistics for each task. `ProbeCom_Init()` which initializes the **μC/Probe** generic communication module, `ProbeRS232_Init()` which initializes the RS-232 communication module and `KSD_Init()` which initializes the IAR Kickstart kit demo (KSD) for the demo version of **μC/Probe.** (see AN-9913). 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 4: `App_TaskCreate()` Creates all the application task. `App_EventCreate()` creates all the application events, in there, a mailbox is created. When the either the 'Next' or "Prev' button are pressed the `App_TaskTouchScr()` will send a message to `App_TaskUserIF()` with the value of the state of the user interface , changing the LCD output.

Listing 4-2, Note 9: 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 the LEDs are toggled.

4.02 *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 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 63, allowing up to 64 total tasks.
- **`OS_MAX_TASKS`** determines the number of "application" tasks and is currently set to 10.
- **`OS_TICKS_PER_SEC`** is set to 1000 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 IAR LPC2478-SK. 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 *cstartup.)*.

5.01 BSP, *bsp.c*, *bsp_lcd.c, bsp_touchscr* and *bsp.h*

The file *bsp.c bsp_lcd.c* and *bsp_touchscr.c* implement 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 several categories:

Critical Processor Initialization and Clock information:

- **`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.
- **`BSP_CPU_ClkFreq()`** returns the CPU clock frequency in Hz.
- **`BSP_CPU_PclkFreq()`** returns the peripheral clock frequency in Hz based on the ID of the peripheral.

LEDs Functions:

- **`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 If an argument of 0 is provided, the appropriate action will be performed on all LEDs.

Push Buttons Functions:

- **`BSP_PB_GetStatus ()`** returns the status of the board's push buttons corresponding the ID passed as the argument.

ADC Functions:

- **`BSP_ADC_GetStatus ()`** returns the status of the ADC corresponding the ADC channel passed as the argument

Serial Interface Functions:

- **BSP_Ser_Init()** Initializes the serial port UART 0
- **BSP_Ser_WrByte()** and **BSP_Ser_WrStr ()** writes a byte and a string (respectively) to the serial port UART 0
- **BSP_Ser_RdByte()** and **BSP_Ser_RdStr ()** reads a byte and a string (respectively) to the serial port UART 0
- **BSP_Ser_Printf()** write a formatted C string to the serial port.

LCD Interface Functions:

- **BSP_LCD_Init()**, **BSP_LCD_TurnOn()** and **BSP_LCD_TurnOff()** will initializes, turn 0n and turn off the LCD display.

  These functions will not be necessary if the GLCD library is used for access the LCD. These function are needed to be used with other Micrium's products (e.g. **μC/GUI)**

Touch Screen Interface Functions:

- **BSP_TouchScr_Init()**, initializes the Touch Screen hardware.
- **BSP_TouchScr_MeasureX()**, this function measure the touch screen position in the x-axis. The value returned is in terms of the ADC resolution (e.g. For 10 bits the values will be between `0x000` and `0x3FF`)
- **BSP_TouchScr_MeasureY()**, this function measure the touch screen position in the Y-axis. The value returned is in terms of the ADC resolution (e.g. For 10 bits the values will be between `0x000` and `0x3FF`)
- **BSP_TouchScr_Convert()**.Convert the Touch screen coordinates from ADC values to Pixels.

The Touch Screen functions **BSP_TouchScr_MeasureX()**, **BSP_TouchScr_MeasureY()** and **BSP_TouchScr_Convert()** receives a pointer to the `BSP_TOUCH_SCR_STATUS` structure defined as:

```
typedef struct bsp_touch_scr_status {

    CPU_INT16U    TouchScrX;
    CPU_INT16U    TouchScrY;
    CPU_BOOLEAN   TouchScrIsPressed;

} BSP_TOUCH_SCR_STATUS;
```

Where `TouchScrX` and `TouchScrY` are the coordinates and `TouchScrIsPressed` specified if the Touch Screen has been pressed form the last conversion function call.

```
void  BSP_Init (void)
{
    BSP_PLL_Init();                                                /* Note 1 */

    BSP_RTC_Init();

    BSP_MAM_Init();

    BSP_LED_Init();

    BSP_PB_Init();

    BSP_ADC_Init();

    BSP_VIC_Init();

    BSP_Tmr_TickInit();                                             /* Note 2 */
}
```

Listing 5-1, `BSP_Init()`

Listing 5-1, Note 1: All the Peripherals are initialized.

Listing 5-1, Note 2: The **μC/OS-II** tick interrupt source is initialized.

Listings 5-2 and 5-3 give the **μC/OS-II** timer tick initialization function, `BSP_Tmr_TickInit()`, the tick ISR handler, `BSP_Tmr_TickISR_Handler()`. These may serve as examples for initializing an interrupt and servicing that interrupt.

```
static  void  BSP_Tmr_TickInit (void)
{
    CPU_INT32U  pclk_freq;


    VICINTSELECT      &= ~(1 << VIC_TIMER0);
    VICVECTADDR4       =  (CPU_INT32U)BSP_Tmr_TickISR_Handler;          /* Note 1 */
    VICVECTPRIORITY4   = 15;
    VICINTENABLE       =  (1 << VIC_TIMER0);

    pclk_freq          =   BSP_CPU_PclkFreq(BSP_PCLK_TIMER0);
    rld_cnts           =   pclk_freq / OS_TICKS_PER_SEC;                /* Note 2 */
    T0TCR              =  (1 << 1);
    T0TCR              =   0;
    T0PC               =   0;

    T0MR0              =   rld_cnts;                                    /* Note 2 */
    T0MCR              =   3;

    T0CCR              =   0;
    T0EMR              =   0;
    T0TCR              =   1;
}
```

Listing 5-2, `BSP_Tmr_TickInit()`

Listing 5-2, Note 1: The tick ISR handler is programmed into the Vectored Interrupt controller and the interrupt is enabled.

Listing 5-2, Note 2: The number of counts per tick is calculated

Listing 5-2, Note 3: The calculated re-load value is programmed into the Timer Match 0, the timer interrupt is enabled and the timer is started

```
void  BSP_Tmr_TickISR_Handler (void)
{

    T0IR  = 0xFF;                                              /* Note 1 */
    OSTimeTick().                                              /* Note 2 */
}
```

Listing 5-3, `BSP_Tmr_TickISR_Handler()`

Listing 5-3, Note 1: The timer 0 interrupt is cleared.

Listing 5-3, Note 2: `OSTimeTick()` informs **μC/OS-II** of the tick interrupt.

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).

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.

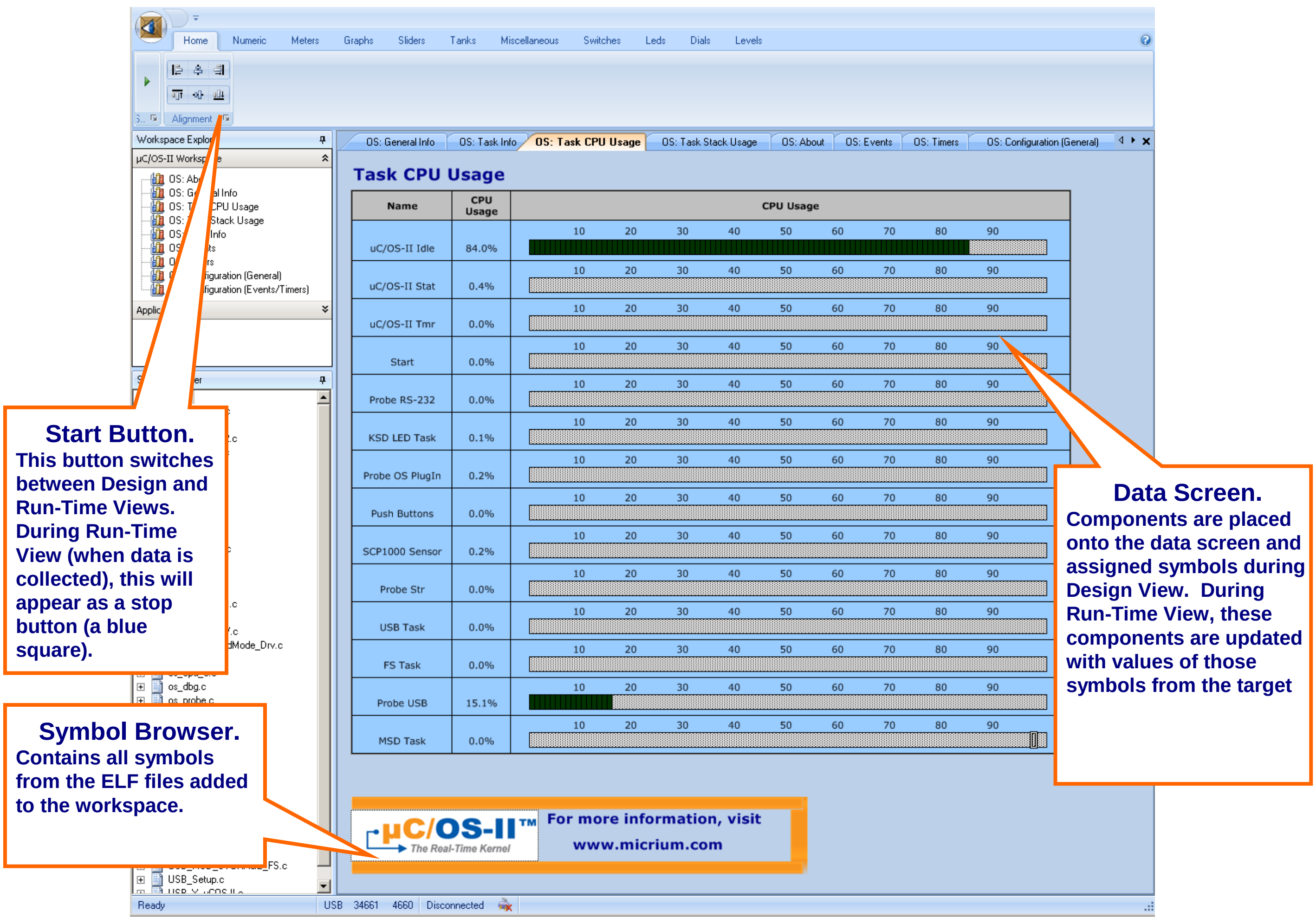


Figure 6-1. μC/Probe Windows Program

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

> **IAR Kickstart Kits Users**
>
> If this development board is part of the IAR Kickstart Kit a demo version of **μC/Probe** is already included in the installation CD. Please refer to the application note **AN-9913** for more details in how to use the demo version of **μC/Probe** with the IAR Kickstart kits.

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

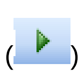
   <Project Name>.elf

   in EWARM v4.4x and

   <Project Name>.out

   in EWARM v5.xx 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 the **μC/OS-II** workspace which displays detailed information about each task's state.

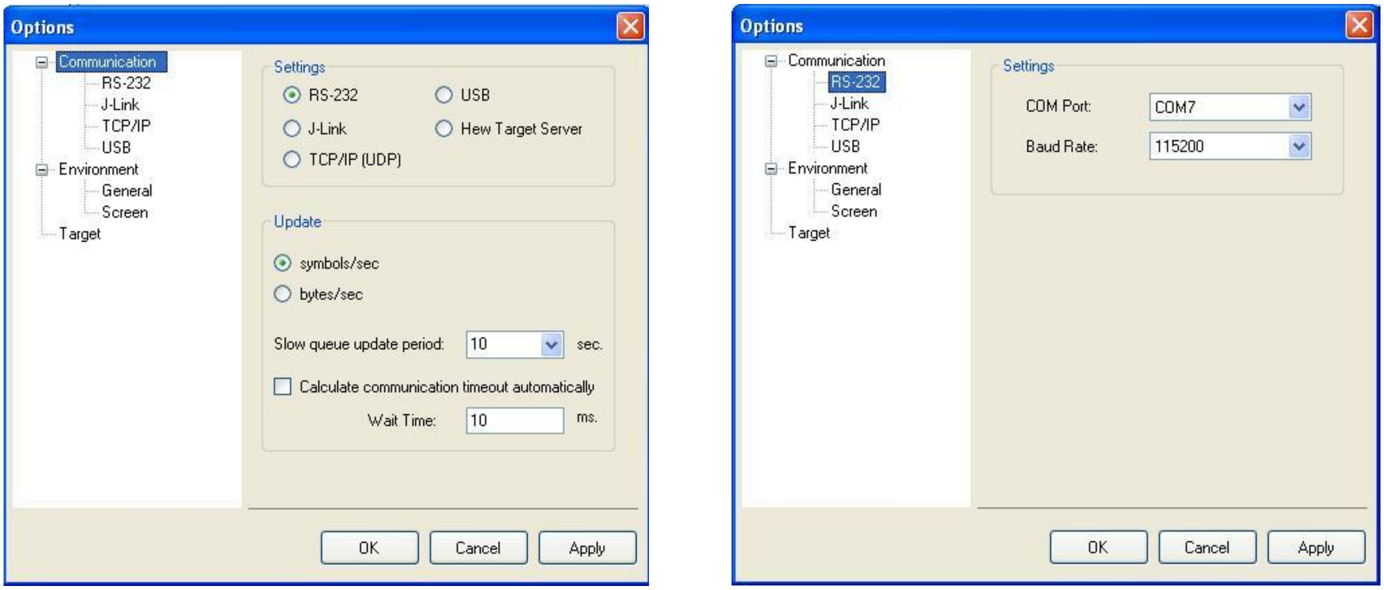


Figure 6.2. µC/Probe Options

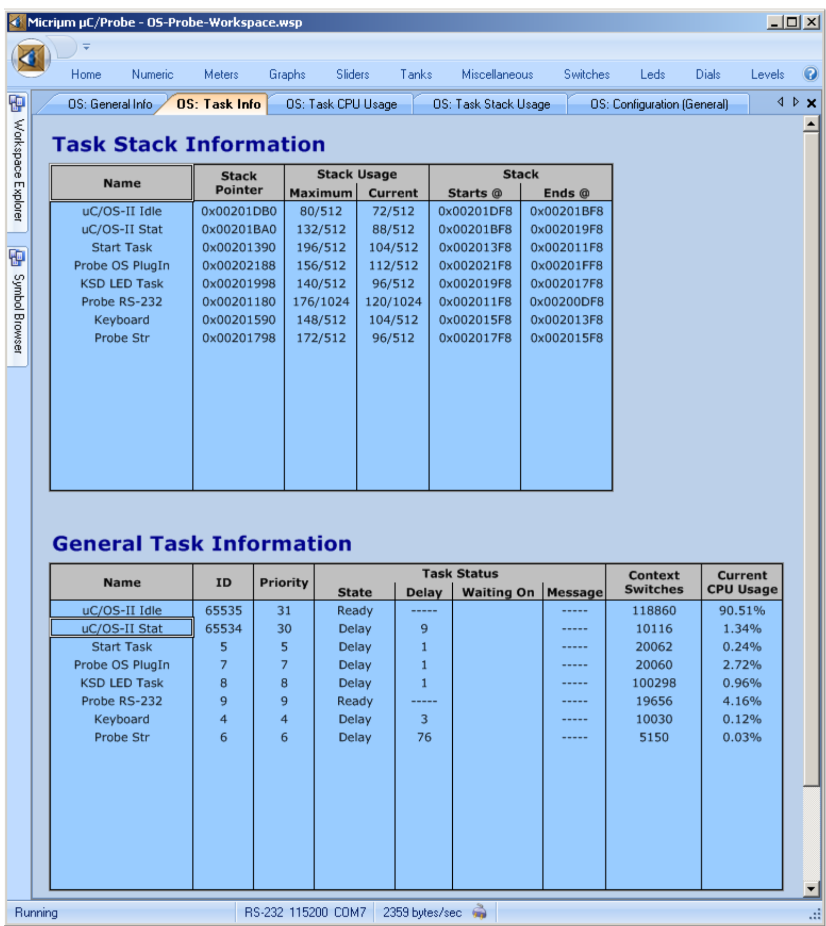


Figure 6-3. µC/Probe Run-Time: µC/OS-II Task Information

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
1290 Weston Road, Suite 306
Weston, FL 33326 U.S.A.

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

WEB : http://www.Micrium.com