

Micrium

Empowering Embedded Systems

μC/Probe Demo Version

on the
IAR Kickstart Development Kits

Application Note

AN-9913

www.Micrium.com

About Micrium

Micrium 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 Micrium **µC/OS-II**, features the highest-quality source code available for today's embedded market. Micrium 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. Micrium's products consistently shorten time-to-market throughout all product development cycles. For additional information on Micrium, please visit www.micrium.com.

About µC/OS-II

Thank you for your interest in **µC/OS-II**. **µC/OS-II** is a preemptive, real-time, multitasking kernel. **µC/OS-II** has been ported to over 45 different CPU architectures and now, has been ported to the Atmel AT91SAM9RL-EK evaluation board which includes the ARM-based Atmel AT92SAM9RL64 processor.

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

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

Licensing

µC/OS-II is provided in source form for **FREE** evaluation, for educational use or for peaceful research. If you plan on using **µC/OS-II** in a commercial product you need to contact Micrium 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 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 Micrium website:

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

Manual Version

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

Version	Date	By	Description
V.1.00	2008/07/07	FT	Initial version.

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/Software/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

Table of Contents	6
1. Introduction	7
2. µC/Probe	8
2.01 Installing and running µC/Probe	8
2.02 µC/Probe Example Demo Code <code>ksd.c</code>	9
2.03 µC/Probe Example Demo Structure	12
2.03.01 Numeric and Meter Components:	12
2.03.02 Dial and Level Component.	13
2.03.03 Graph Component.	13
2.03.04 LED Components.	14
2.03.05 Spreadsheets.	15
2.03.06 Image Component.	16
Licensing	18
References	18
Contacts	18

1. Introduction

This document AN-9913 explains how to use the demo version of **μC/Probe** with the IAR kickstart development kits. The demo version of **μC/Probe** has the following limitations:

- Data can only be retrieved from RS-232C or J-LINK interfaces.
- Only **15** symbols can be read/sent to the target.
- A watermark "Micrium **μC/Probe** Demo Version © 2008" is set in the data screen background.

The demo workspace is intended to work with any of the IAR kickstart development kits, for more information about the IAR development kits please visit the IAR website at:

<http://www.iar.com/website1/1.0.1.0/16/1/index.php>

2. µC/Probe

The following sections step through the **µC/Probe** example demo for the IAR kickstart developments kits. First, the setup of **µC/Probe** will be outlined. Second, the source code of demo will be explained. Lastly, instructions will be provided for using the demo.

2.01 Installing and running µC/Probe

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

1. **Download and Install µC/Probe.** A demo version of **µC/Probe** can be downloaded from the Micrium 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 named *OS-Probe-IAR-Kickstart-Demo-Workspace.wsp*, which should be located in your project folder.

/Micrium/Software/EvalBoards/< IAR Kickstart Development Kit Directory >/OS-Probe

where *<IAR Kickstart Development Kit Directory>* is the directory for the specific IAR kickstart development kit board.

1. **Connect Target to PC.** Currently, the demo version of **µC/Probe** can use only RS-232 and J-LINK to retrieve information from the target. You should connect a RS-232 cable between your target and computer.
2. **Load Your ELF File.** The example projects included with this application note are already configured to output an ELF file. (If you are using your own project, please refer to Appendix A of the **µC/Probe** user manual for directions for generating an ELF file with your compiler.) This file should be in

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


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

<Project Name>.elf

in EWARM v4.4x and

<Project Name>.out

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

3. **Configure the RS-232 Options.** In **µC/Probe**, choose the “Options” menu item on the “Tools” menu. 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.
4. **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.

2.02 **µC/Probe** Example Demo Code `ksd.c`

The code used for the IAR Kickstart Demo (KSD) demonstrating all the capabilities provided by **µC/Probe** is found in the file `ksd.c` located at:

Micrium\Software\uC-Probe\Target\Demo\KSD\Source\ksd.c
Micrium\Software\uC-Probe\Target\Demo\KSD\Source\ksd.h

Basically, `ksd.c` implements two single functions:

1. **KSD_Init()** Initialize all the KSD module internal variables and creates the `KSD_TaskLED` task. This is the only function called from the application code `app.c` if you want to integrate this demo to you project.
2. **KSD_TaskLED()**, it is a periodic task in which the LEDs for the **µC/Probe** example data screen 4 (see section 2.03.04) are blinked based on the state of the Switch component .

```
static CPU_INT32U KSD_Probe_Dial; /* Note 1 */
static CPU_INT32U KSD_Probe_Picture_Ix;

static CPU_BOOLEAN KSD_Probe_LED_Sel; /* Note 2 */
static CPU_BOOLEAN KSD_Probe_LED1;
static CPU_BOOLEAN KSD_Probe_LED2;
static CPU_BOOLEAN KSD_Probe_LED3;

static OS_STK KSD_TaskLEDStk[KSD_CFG_TASK_LED_STK_SIZE]; /* Note 3 */
```

Listing 2-1, `ksd.c` Global variables

Listing 2-1, Note 1: `KSD_Probe_Dial` and `KSD_Probe_Dial` are not used, they are only created for the **µC/Probe** example data screen 2 and 6 (see section 2.03.02 and 2.03.06).

Listing 2-1, Note 2: `KSD_Probe_LED_Sel` holds the value of the Switch component in the **µC/Probe** example data screen 4 `KSD_Probe_LED1`, `KSD_Probe_LED2`, `KSD_Probe_LED3` are the variables attached to the LED components.

Listing 2-1, Note 3: `KSD_TaskLEDStk` is the stack for the `KSD_TaskLED()` task.

```

void KSD_Init (void)
{
    CPU_INT08U  os_err;

    (void)&KSD_Probe_Dial;                                /* Note 1 */
    (void)&KSD_Probe_LED_Sel;
    (void)&KSD_Probe_LED1;
    (void)&KSD_Probe_LED2;
    (void)&KSD_Probe_LED3;
    (void)&KSD_Probe_Picture_Ix;

    #if (OS_TASK_CREATE_EXT_EN > 0)                        /* Note 2 */
    #if (OS_STK_GROWTH == 1)
        os_err = OSTaskCreateExt( KSD_TaskLED,
                                   (void *)0,
                                   &KSD_TaskLEDStk[KSD_CFG_TASK_LED_STK_SIZE - 1],
                                   KSD_CFG_TASK_LED_PRIO,
                                   KSD_CFG_TASK_LED_ID,
                                   &KSD_TaskLEDStk[0],
                                   KSD_CFG_TASK_LED_STK_SIZE,
                                   (void *)0,
                                   OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);
    #else
        os_err = OSTaskCreateExt( KSD_TaskLEDSEL,
                                   (void *)0,
                                   &KSD_TaskLEDSELStk[0],
                                   KSD_CFG_TASK_LED_SEL_PRIO,
                                   KSD_CFG_TASK_LED_SEL_ID,
                                   &KSD_TaskLEDSELStk[KSD_CFG_TASK_LED_SEL_STK_SIZE - 1],
                                   KSD_CFG_TASK_LED_SEL_STK_SIZE,
                                   (void *)0,
                                   OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);
    #endif
    #else
    #if (OS_STK_GROWTH == 1)
        os_err = OSTaskCreate( KSD_TaskLED,
                                (void *)0,
                                &KSD_TaskLEDStk[KSD_CFG_TASK_LED_STK_SIZE - 1],
                                KSD_CFG_TASK_LED_PRIO);
    #else
        os_err = OSTaskCreate( KSD_TaskLED,
                                (void *)0,
                                &KSD_TaskLEDStk[0],
                                KSD_CFG_TASK_LED_PRIO);
    #endif
    #endif

    #if (OS_TASK_NAME_SIZE > 12)                            /* Note 3 */
        OSTaskNameSet(KSD_CFG_TASK_LED_PRIO, (CPU_INT08U *)"KSD LED Task", &os_err);
    #endif
}

```

Listing 2-2, KSD_Init()

Listing 2-2 Note 1: Since most of the variables are not used in the `ksd.c` module, this will avoid the warning messages.

Listing 2-1, Note 2: Creates the KSD_TaskLED() task. KSD_CFG_TASK_LED_STK_SIZE and KSD_CFG_TASK_LED_PRIO must be defined in app_cfg.h

Listing 2-1, Note 3: Set the name for the KSD_TaskLED() task.

```
static void KSD_TaskLED (void *p_arg)
{
    CPU_INT32U  ctr;

    ctr          = 0;
    KSD_Probe_LED1 = 0;
    KSD_Probe_LED2 = 0;
    KSD_Probe_LED3 = 0;

    while(DEF_TRUE) {
        if(KSD_Probe_LED_Sel == DEF_OFF) {
            KSD_Probe_LED1 = DEF_OFF;
            KSD_Probe_LED2 = DEF_ON;
        } else {
            KSD_Probe_LED1 = DEF_ON;
            KSD_Probe_LED2 = DEF_OFF;
        }
        /* Note 1 */

        if (ctr > 100) {
            if (KSD_Probe_LED3 == DEF_OFF) {
                KSD_Probe_LED3 = DEF_ON;
            } else {
                KSD_Probe_LED3 = DEF_OFF;
            }
            ctr = 0;
        } else {
            ctr++;
        }
        /* Note 2 */

        OSTimeDlyHMSM(0, 0, 0, 10);
        /* Note 3 */
    }
}
```

Listing 2-3, KSD_TaskLED ()

Listing 2-3, Note 1: 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 LED1 and LED2 are toggled based on the value of the Switch component.

Listing 2-3, Note 2: LED3 is toggled at 1 Hz.

Listing 2-3, Note 3: The task is delayed for 10 m

2.03 μC/Probe Example Demo Structure

This demo will explain you all the **μC/Probe** components and how you can easily integrate them in your target.

The example workspace *OS-Probe-IAR-Kickstart-Demo-Workspace.wsp* is a self-explanatory demo where all the **μC/Probe** components are explained. You can use this demo example to start you own **μC/Probe** project.

The demo workspace is divided in 6 different screens where every aspect of **μC/Probe** is explained in detail:

2.03.01 Numeric and Meter Components:

μC/Probe can display data using a Numeric or a Meter component. In this first screen the value of `OSCPUsage` is being displayed using a Meter and a Numeric component as shown in figure 4.1.

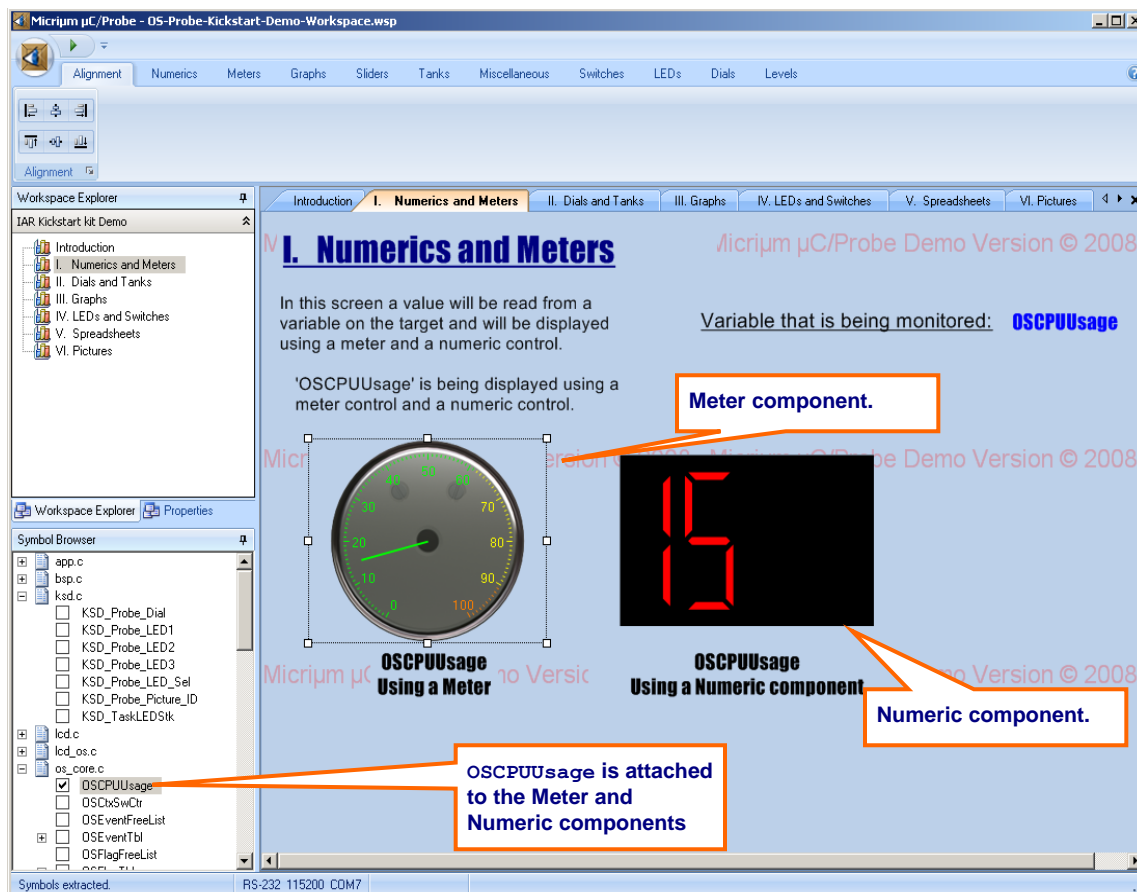


Figure 2-1. **μC/Probe**, Numeric and Meter components.

2.03.02 Dial and Level Component.

Figure 4.2 shows how values can be injected to the target using the Dial component in **μC/Probe**.

In this screen a numeric value is being sent to the variable `KSD_Probe_Dial` using a Dial component. The Dial component is attached to the `KSD_Probe_Dial` symbol.

At the same time the value of `KSD_Probe_Dial` is being read from the target using a Level component. The Level component is also attached to the `KSD_Probe_Dial` symbol.

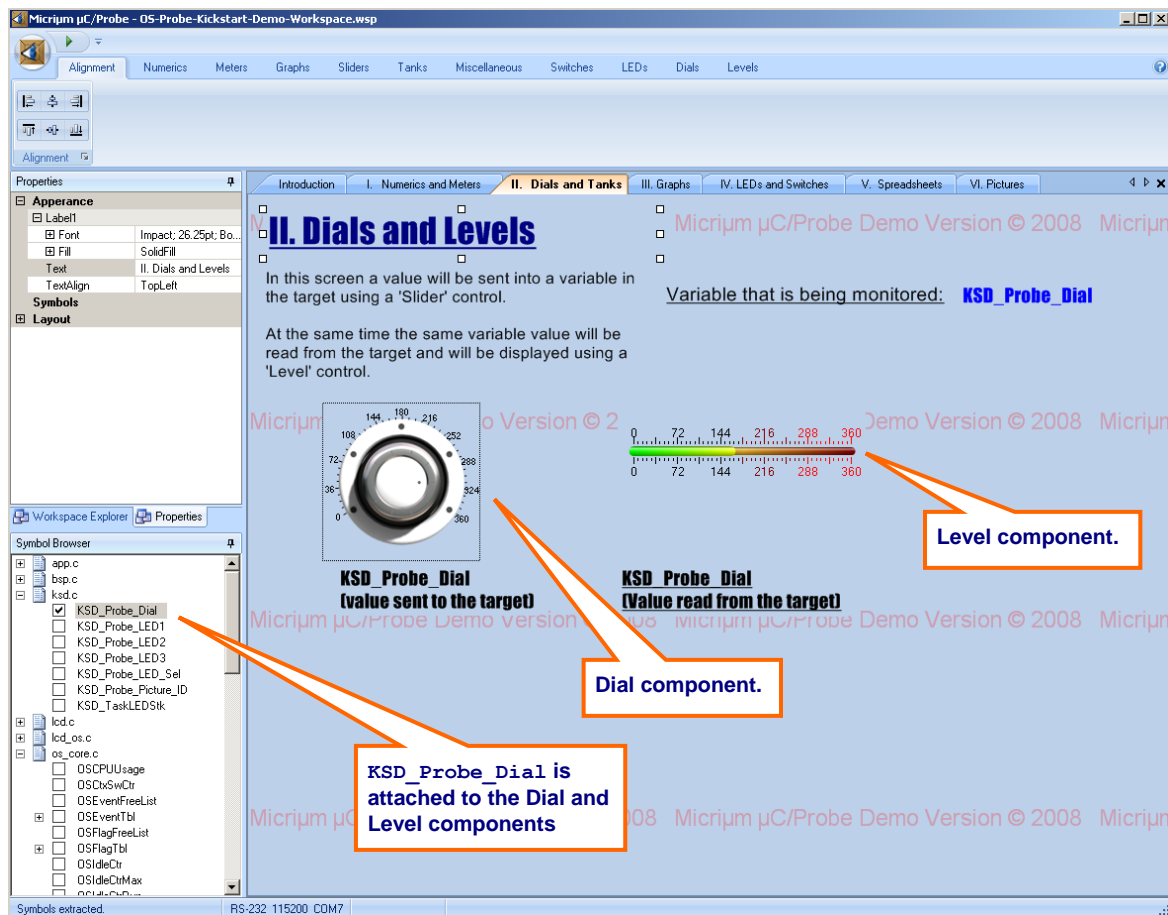


Figure 2-2. **μC/Probe**, Dial and Level component.

2.03.03 Graph Component.

With **μC/Probe** a variable can be monitored over a period of time. Figure 4-3 shows the example screen for the Graphs components, in this screen the value of `OSCPUUsage` is being graphed using the Graphs components.

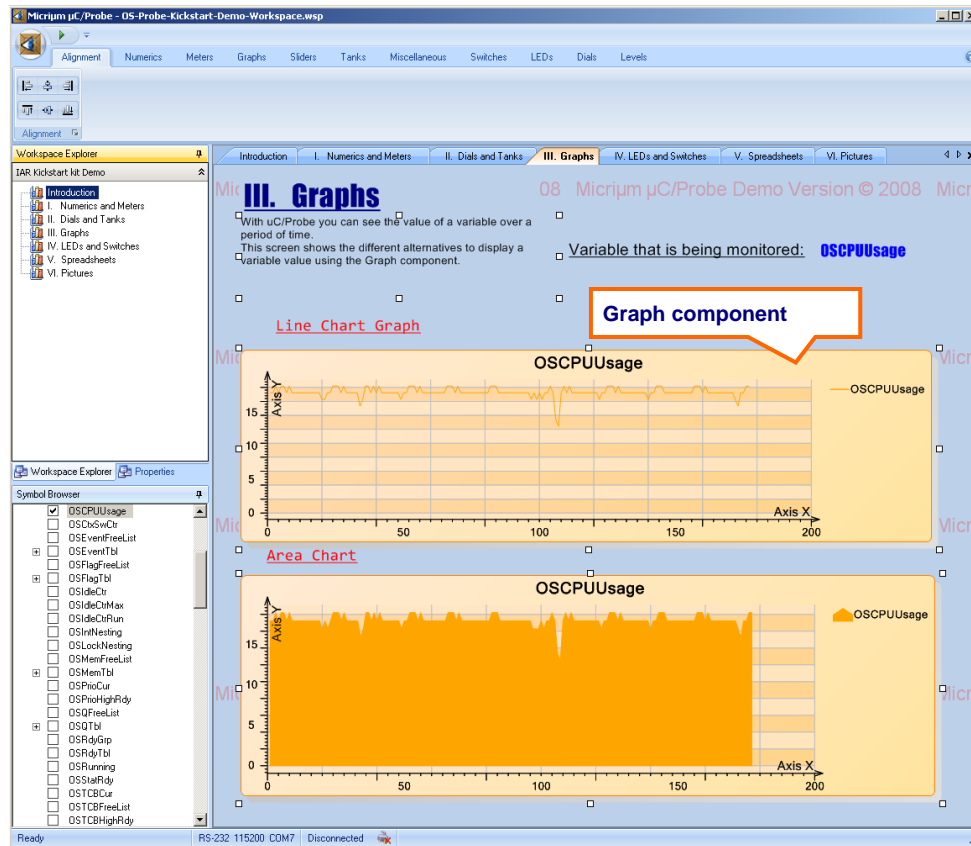


Figure 2-3. μC/Probe, Graph component.

2.03.04 LED Components.

In this screen three LEDs are being displayed using the LED component. The value of LED 1 and LED 2 will be controlled by the Switch component and the value of LED 3 is being toggled at 1 Hertz.

The figure 4.4 shows the LED component example screen and the relationship between the **μC/Probe** components and the variables on the target.

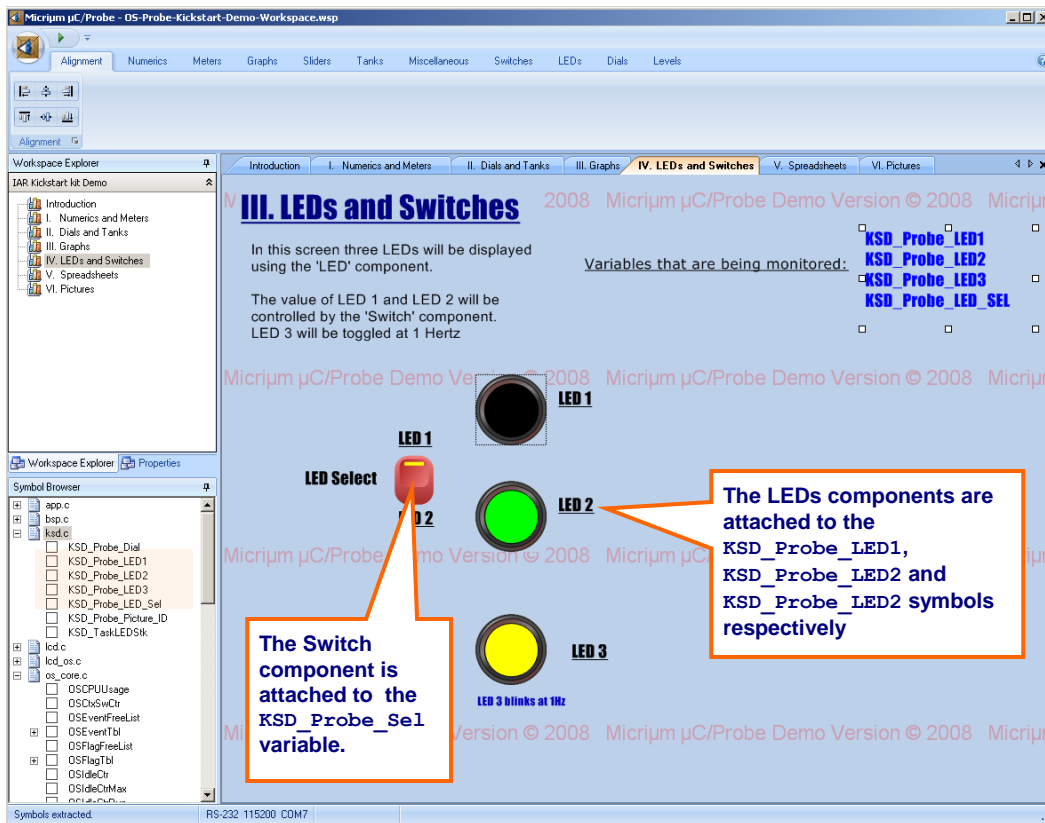


Figure 2-4. **μC/Probe**, LED and Switch components.

2.03.05 Spreadsheets.

The spreadsheet component offers a convenient and flexible interface for organizing symbols and using symbol values in calculations. The component interface mirrors Microsoft Excel, with the same formula conventions, a compatible function library, and similar management features.

The example screen shown in Figure 4-5 shows a digital clock using the value of OSTime and OSTickPerSec symbols from **μC/OS-II**.

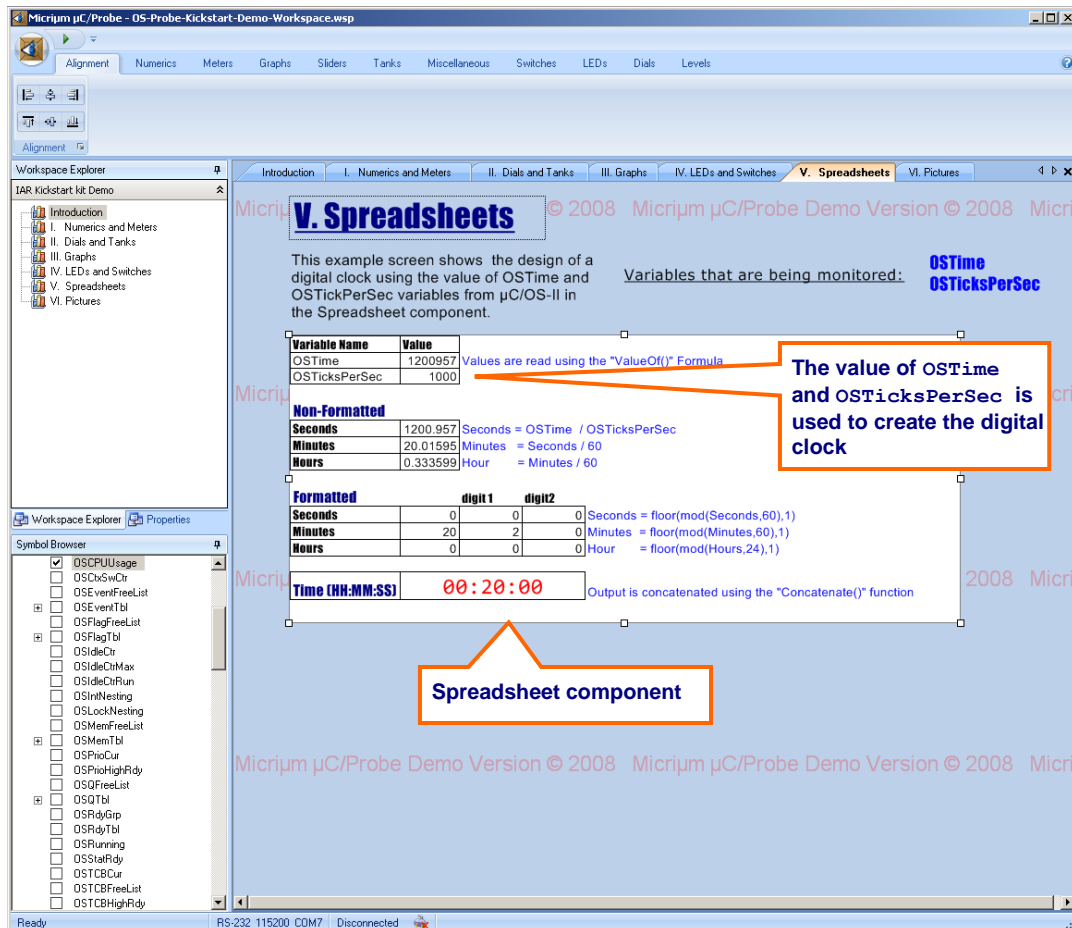


Figure 2-5. μC/Probe, Spreadsheet component.

2.03.06 Image Component.

The picture component allows you to select an image from a set of images based on the value of a symbol. In this example screen the image index is sent to the symbol KSD_Probe_Picture_Ix in the target using the Slider component as shown in Figure 4.

You can add any set of images using the **ImageWrap Collection Editor** from the Component's properties as shown in Figure 4

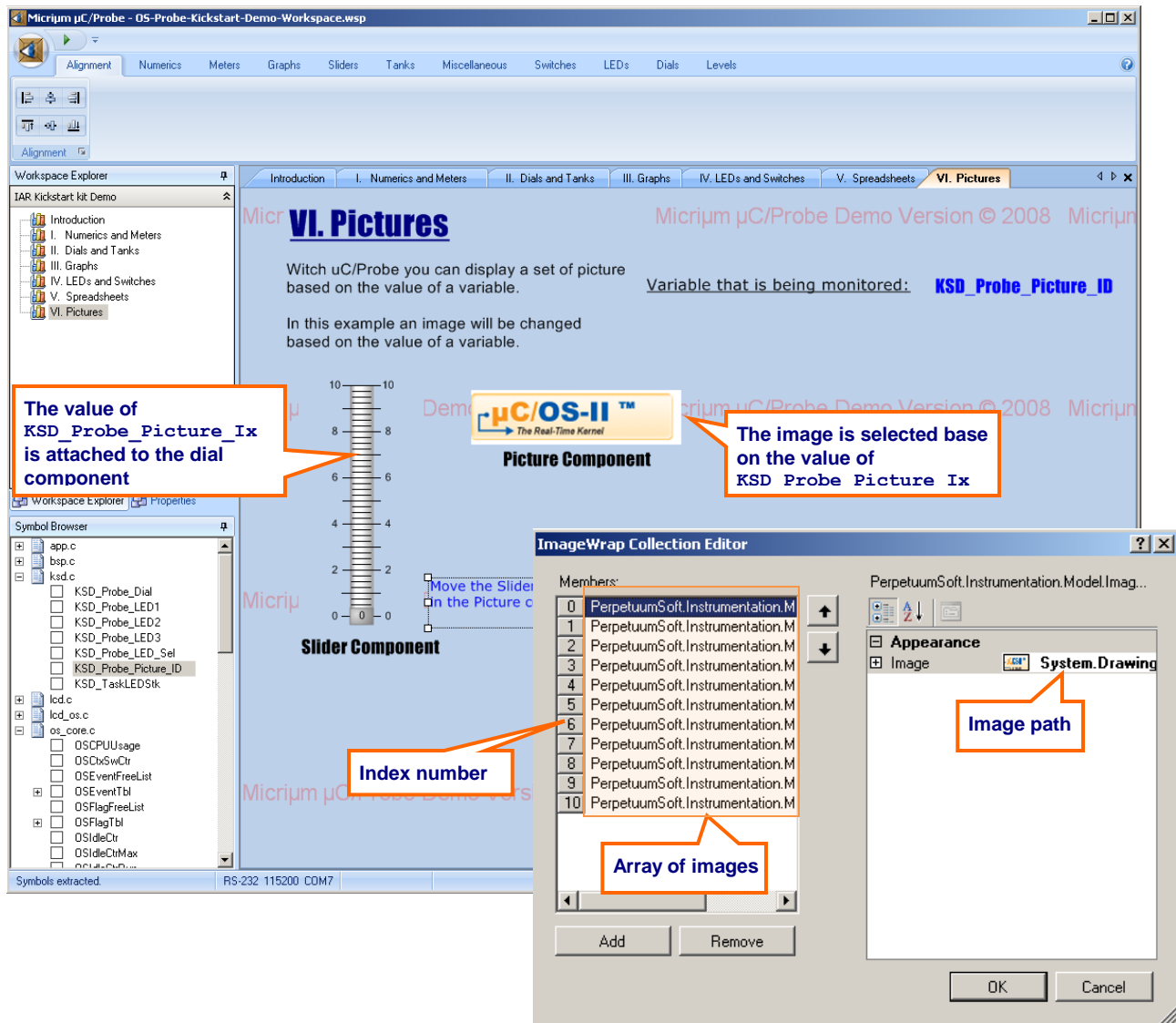


Figure 2-6. **μC/Probe**, Image component.

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 Micrium to properly license its use in your product. We provide **ALL** the source code with this application note for your convenience and to help you experience **μC/OS-II**. The fact that the source is provided does **NOT** mean that you can use it without paying a licensing fee. Please help us continue to provide the Embedded community with the finest software available. Your honesty is greatly appreciated.

References

μC/OS-II, The Real-Time Kernel, 2nd Edition

Jean J. Labrosse
R&D Technical Books, 2002
ISBN 1-57820-103-9

Embedded Systems Building Blocks

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

Contacts

IAR Systems

Century Plaza
1065 E. Hillsdale Blvd
Foster City, CA 94404
USA
+1 650 287 4250
+1 650 287 4253 (FAX)
e-mail: Info@IAR.com
WEB : www.IAR.com

CMP Books, Inc.

1601 W. 23rd St., Suite 200
Lawrence, KS 66046-9950
USA
+1 785 841 1631
+1 785 841 2624 (FAX)
e-mail: rushorders@cmpbooks.com
WEB : <http://www.cmpbooks.com>

Micrium

949 Crestview Circle
Weston, FL 33327
USA
+1 954 217 2036
+1 954 217 2037 (FAX)
e-mail: Jean.Labrosse@Micrium.com
WEB : www.Micrium.com

Atmel

2325 Orchard Parkway
San Jose, CA 95131
USA
+1 408 441 0311
+1 408 487 2500
WEB : www.atmel.com