# CY8CKIT-044_OS3

**Licensing is required when using any Micriμm software, regardless of the state of the software (Library or Full Source). This project is only meant for example purposes. For projects using Library versions of the software, please contact our Sales office to obtain the Full Source version at +1 (954) 217-2036.**

## CY8CKIT-044 Example Project Read-Me

The provided example project for which this Read-Me was made utilizes the Cypress CY8CKIT-044 (CY8C4247AZI) evaluation board from the PSoC4 M-Series Family. The MCU found on this development board conforms with the ARM_Cortex_M0 architecture.



## Project Download

| | |
|---|---|
| Download Link | Micrium_CY8CKIT-044_OS3.zip |

## Toolchain IDE Versions

| IDE/Toolchain | Version |
|---|---|
| Cypress PSoC Creator | 3.3 |

## Micriμm Product Versions

| Product | Version |
|---|---|
| µC/CPU | 1.30.02 |
| µC/LIB | 1.38.01 |
| µC/OS-III | 3.05.01 |
| µC/Probe | 3.6.15.900 |

## Hardware Setup

1. Have the board connected via the **PSoC Prog (USB)** into the board debugging input (**J6**).
2. PSoC Prog USB will supply Power.

## Loading & Running The Project on the Board

**Make sure to open the example project workspace using the mentioned IDE(s) version or newer.**

## Cypress PSoC Creator™

1. Click on *File–>Open–>Project/Workspace...*
2. Navigate to the directory where the workspace is located: *$\Micrium\Examples\Cypress\CY8CKIT-044\OS3\PSoC\OS3.cywrk*
3. Click *Open*.
4. For safety, clean the project by clicking on *Build–>Clean* (if available).
5. Compile the project by clicking on *Build–>Build*.
    a. If Generated_Source is not included, then PSoC Creator will create "Generated Source Code" based on the TopDesign.cysch file.
6. Make sure your hardware setup (as previously described) is correct.
7. Download the code to the board by clicking on *Debug–>Debug*.
8. Run the project by clicking on *Debug–>Resume Execution*. To stop the project from running click *Debug–>Stop Debugging*.

## µC/OS-III

```
void  main (void)
{
    ...
    OSInit(&os_err);                                        /* Initialize uC/OS-III
*/        (1)

 ...
    OSTaskCreate(&AppTaskStartTCB,                          /* Create the start task
*/        (2)
                 "App Task Start",
                  AppTaskStart,
                  0,
                  APP_CFG_TASK_START_PRIO,
                 &AppTaskStartStk[0],
                  APP_CFG_TASK_START_STK_SIZE / 10u,
                  APP_CFG_TASK_START_STK_SIZE,
                  0u,
                  0u,
                  0,
                 (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                 &os_err);
    OSStart(&os_err);                                       /* Start multitasking
*/        (3)
}

static  void  AppTaskStart (void *p_arg)
(4)
{
    ....


    while (DEF_TRUE) {                                      /* Task body, always as an
infinite loop.      */        (5)
     ...
(6)

        OSTimeDlyHMSM( 0u, 0u, 0u, 500u,
(7)
                       OS_OPT_TIME_HMSM_STRICT,
                      &os_err);
    }
}
```

Listing - app.c

(1)
OSInit() initializes uC/OS-III and must be called prior to calling OSStart(), which actually starts multitasking.

(2)
OSTaskCreate() creates a task to be managed by uC/OS-III. Tasks can be created either prior to the start of multitasking or by a running task. In this case, the task "AppStartTask" gets created.

(3)
OSStart() starts multitasking under uC/OS-III. This function is typically called from the startup code but after calling OSInit().

(4)
AppTaskStart is the startup task created in (2).

(5)
A task must be written as an infinite loop and must not return.

(6)

In most examples, there is hardware dependent code such as LED blink, etc.

(7)
OSTimeDlyHMSM() allows AppTaskStart to delay itself for a user-specified amount of time (500ms in this case). Rescheduling always occurs when at least one of the parameters is nonzero. Placing a break-point here can ensure that uC/OS-III is running, it should get hit periodically every 500 milliseconds.

For more information please refer to **uC/OS-III Users' Guide**.

# µC/Probe

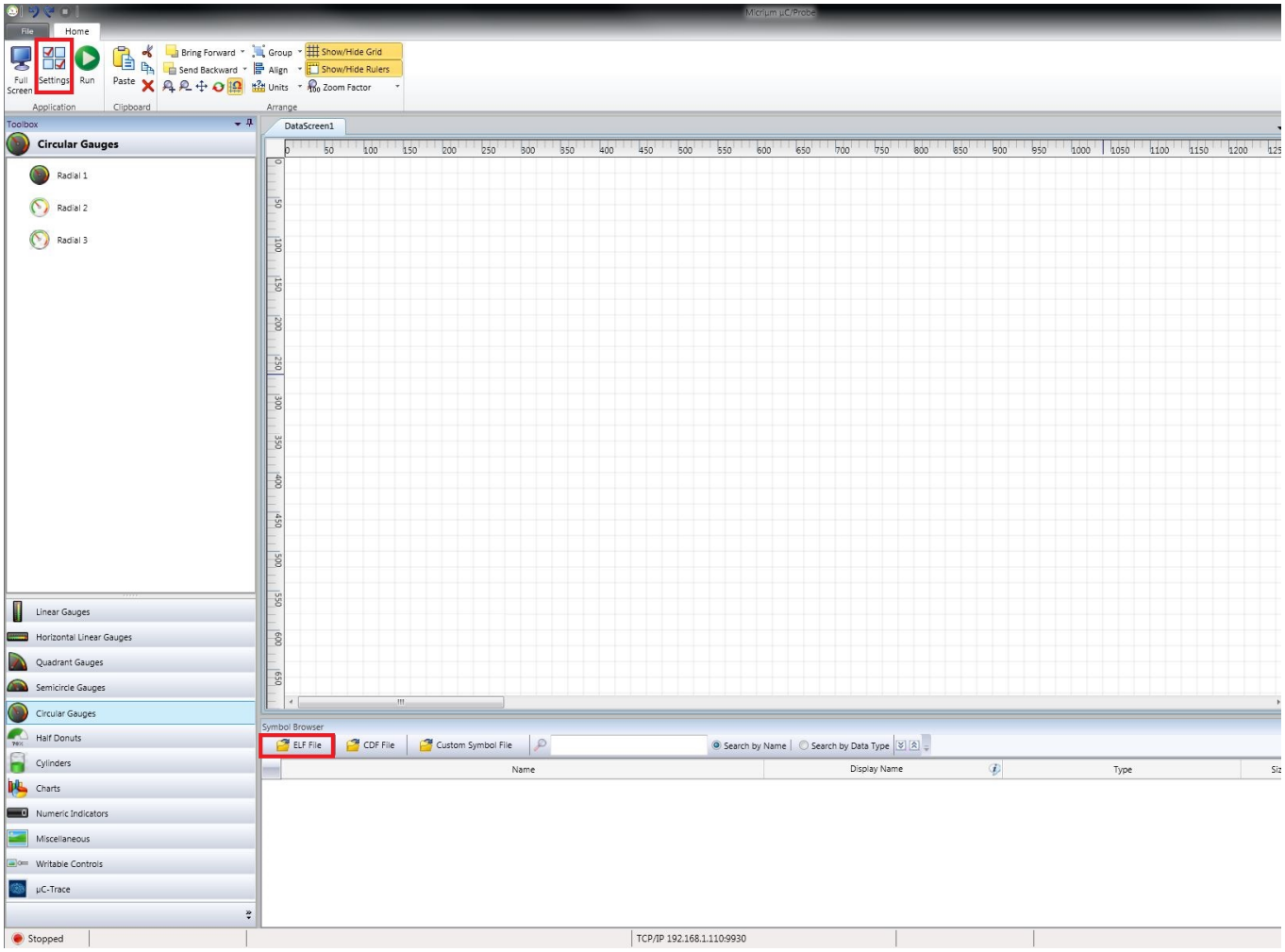
µC/Probe, is a Micriμm Windows™ application to graphically view the internals of any embedded system. This example project includes a pre-configured µC/Probe workspace that can be found at:

*$\Micrium\Examples\Cypress\CY8CKIT-044\OS3\<IDE>\OS3.wspx*

> ⚠ Please compile the project (as described earlier in this document) prior to opening a pre-configured µC/Probe workspace.

In order for µC/Probe to display symbols, an **ELF file** that is generated by the compiler is required. After the example project has compiled, look for the ELF file that is usually found inside the compiler auto-generated binaries folder.

The following image shows where the **ELF file** (highlighted in **RED**) button is found to search for the project's ELF file.




If creating a new µC/Probe workspace, you must configure µC/Probe with the proper communication protocol used in your project. The following communication protocols are currently available for this example project:

## Running with Cypress PSoC Programmer

When running an example project that interfaces with µC/Probe via Cypress' PSoC Programmer, there is no additional set-up necessary other than to configure µC/Probe's settings to "Cypress PSoC Prog".

In µC/Probe's settings, under the *Communication* tab, select *Cypress PSoC Prog* under the *Interfaces* section and configure the *Port* the PSoC Programmer that suites your project's needs. Along with the Cypress PSoC Programmer settings, µC/Probe also allows you to change the endianness of the device, how to receive statistics, and the rate at which the data collection is done.

The following image illustrates how the settings should look: