Micriμm

Empowering Embedded Systems

μC/OS-II

μC/OS-III

for the

Renesas RL78 MCU

on the YRDKRL78G13 Evaluation Board

Application Note

AN-uCOS-II-uCOS-III-Renesas-YRDKRL78G13

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 and μC/OS-III

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 RL78 MCU available from Renesas.

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 and much more.

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

μC/OS-III is the newest Kernel that Micrium provides, which is designed to save time on embedded systems projects. Besides the inherited features from μC/OS-II, μC/OS-III provides an unlimited number of tasks to be created, as well as it allows multiple tasks to run at the same priority level.

Licensing

μC/OS-II and μC/OS-III are provided in source form for **FREE** evaluation, for educational use or for peaceful research. If you plan on using μC/OS-II and/or μC/OS-III 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 and μC/OS-III. The fact that the source is provided **DOES NOT** means 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.1.00	2011/09/20	PC	Initial Version
V.1.01	2011/10/03	PC	Added **μ**C/CPU, **μ**C/OS-II, and **μ**C/OS-III Port Sections
V.2.00	2011/11/08	PC	Separate App Note for architecture and evaluation board
V.2.01	2011/11/10	PC	Updated app code example to display message on LCD

Software Versions

This document may or may not have been downloaded as part of an executable file containing the code and projects described here. If so, then the versions of the Micrium 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	V2.92.01	
μC/OS-III	V3.02.00	
μC/CPU	V1.29.00	
μC/LIB	V1.36.00	

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-uCOS-II-uCOS-III-Renesas-YRDKRL78G13*, explains example code for using μC/OS-II and μC/OS-III on a YRDKRL78G13 evaluation board, as shown in Figure 1-1, which employs Renesas's 78K0-based RL78 microcontroller.

If this app note was downloaded in a packaged executable zip file, then it should have been found in the directory ***/Micrium/AppNotes/AN1xxx-RTOS/AN-uCOS-II-uCOS-III-Renesas-YRDKRL78G13*** 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.

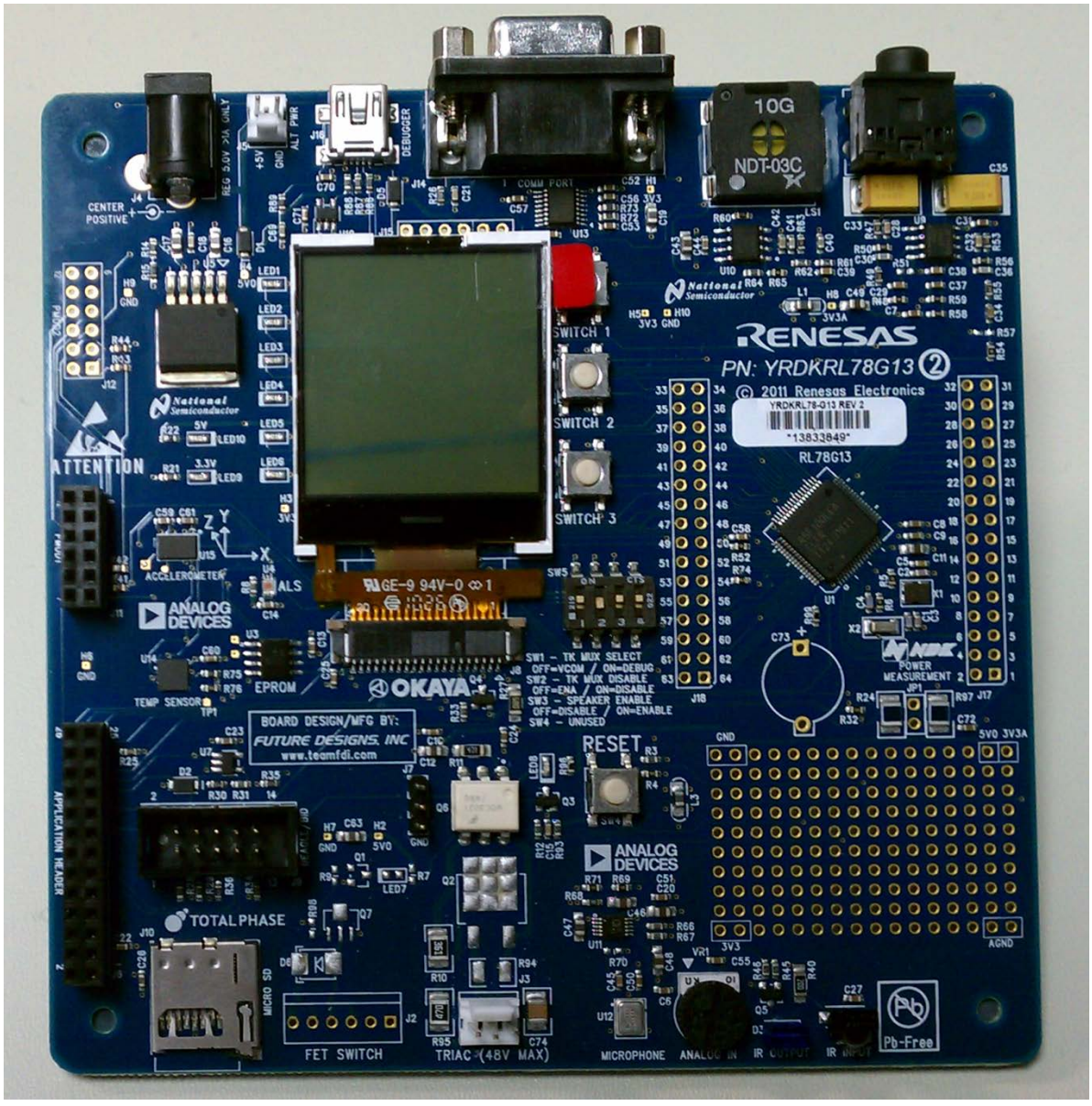

Figure 1-1. YRDKRL78G13 Evaluation Board

2. Getting Started

The following sections detail the necessary steps needed for use of the demonstration application described in this document, *AN-uCOS-II-uCOS-III-Renesas-YRDKRL78G13*. Note that μC/OS-II and μC/OS-III source code is provided with the downloaded example since the source code is available for evaluation purposes **only**.

2.01 Setting up the Hardware

Figure 2-1 shows the board layout of the YRDKRL78G13, ensure that a power supply is connected to the evaluation board and that the corresponding power source is selected. The project described in this document was loaded and debugged on the YRDKRL78G13 using the TK internal debugger. Ensure that the Switch Buttons on the board are configured the following way: 1,2, and 3 = ON; 2 = OFF.

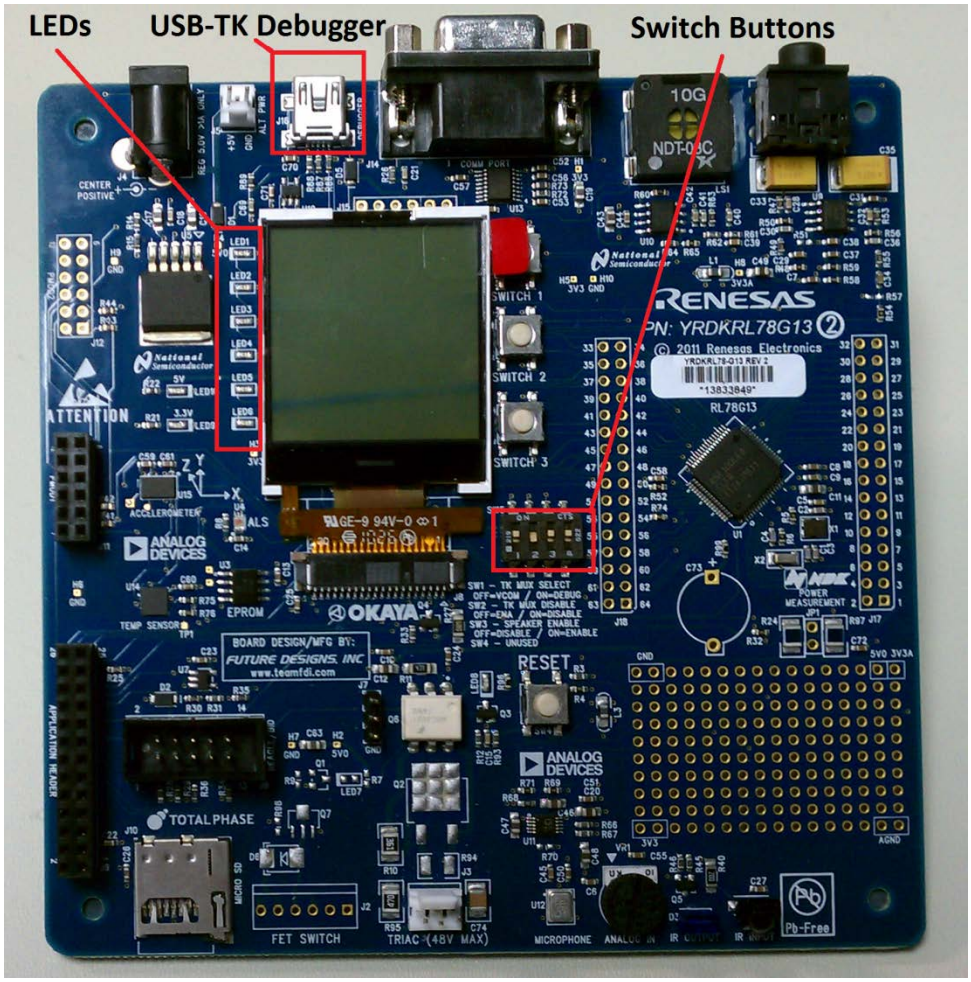

Figure 2-1. YRDKRL78G13 Peripherals

2.02 Opening and Viewing the Project

If the any of the example project were downloaded as part of a zip file (which should have been named *Micrium-uCOS-II-V2.92.01-RL78RDK-YRDKRL78G13* or *Micrium-uCOS-III-V3.02.00-RL78RDK-YRDKRL78G13*), then the code files referred to herein are located in the directory structure shown in Figure 2-2.

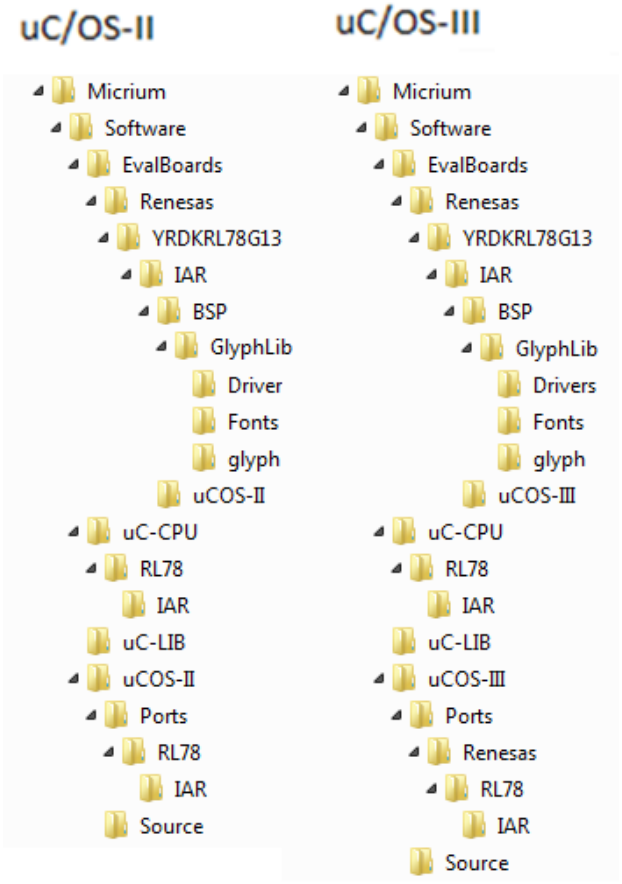

Figure 2-2, µC/OS-II **and** µC/OS-III **Directory Structure**

IAR workspace files (***.eww***), named ***uCOS-II.eww***, or ***uCOS-III.eww*** are located in the following directory.

/Micrium/Software/EvalBoards/Renesas/YRDKRL78G13/IAR/<OS>

Note that <OS> refers to the operating system you are using, namely µC/OS-II or µC/OS-III. To open this example project, first open IAR. Then, navigate to the project directory and select the workspace file, this can be done through either the "Embedded Workbench Startup" dialog box or through the "Open->Workspace" command located under the "File" menu. Additionally, the project can be opened through double clicking the workspace file inside Windows explorer.

2.03 Using the Project and Loading the Target

Once the project is opened and all the proper connections have been made (Section 2.01 and Section 2.02), the code can be built and loaded onto the target. To build the code, select the "Rebuild All" menu item from the "Project" menu.

To load the code onto the target, first, select the "Options" menu item from the "Project" menu and toggle to the "Debugger" tab. Ensure that the "TK" is selected for the Driver settings. Figure 2-3 displays the "Debugger" tab in the "Options" dialog box.

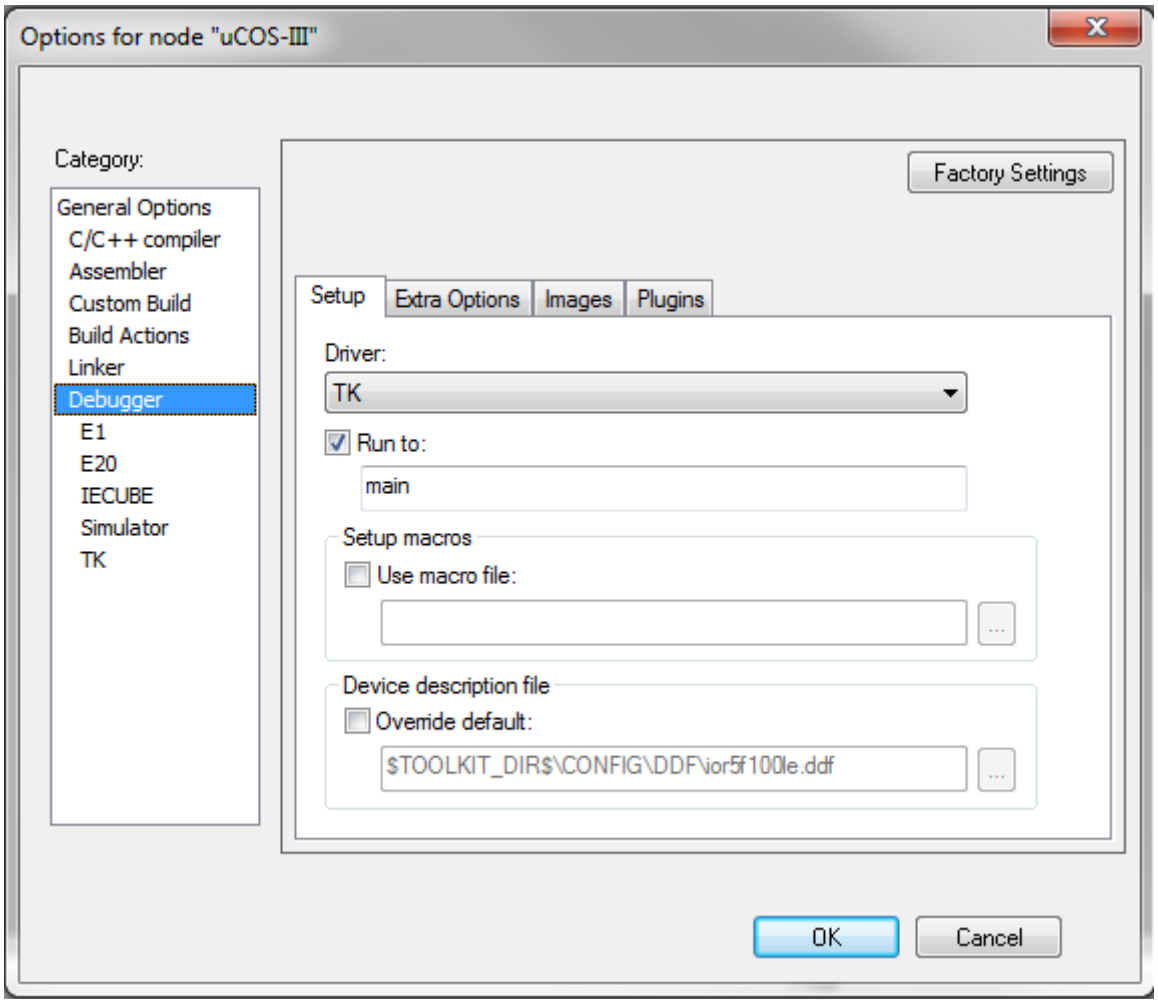

Figure 2-3, Emulator Mode Settings

Now that the debugger has been set, select the "Debug" menu item from the "Project" menu to load the project onto the target. A "TK Hardware Setup" screen might appear. This example project utilizes the default options, so click "OK". Once these steps have been completed, new options will be provided under the "Debug" menu, including debug commands such as: "Go", "Step In", "Step Over", and more.

3. Directories and Files

Application Notes

\Micrium\AppNotes\AN1xxx-RTOS\AN-uCOS-II-uCOS-III-Renesas-YRDKRL78G13

This directory contains this application note, AN-uCOS-II-uCOS-III-Renesas-YRDKRL78G13.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 or μC/OS-III 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\RL78\IAR

This directory contains the standard processor-specific files for the μC/OS-II 78K0R port using 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.

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

\Micrium\Software\uCOS-II\Source

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

- *os_core.c*
- *os_flag.c*
- *os_mbox.c*
- *os_mem.c*
- *os_mutex.c*
- *os_q.c*
- *os_sem.c*
- *os_task.c*
- *os_time.c*
- *os_tmr.c*
- *ucos_ii.h*

μC/OS-III Files

Micrium\Software\uCOS-III\Doc

This directory contains documentation for μC/OS-III.

Micrium\Software\uCOS-III\Ports\Renesas\RL78\IAR

This directory contains the standard processor-specific files for the μC/OS-III 78K0R port using 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.

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

Micrium\Software\uCOS-III\Source

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

- *os h*
- *os_cfg_app.c*
- *os_core.c*
- *os_dbg.c*
- *os_flag.c*
- *os_int.c*
- *os_mem.c*
- *os_msg.c*
- *os_mutex.c*
- *os_pend_multi.c*
- *os_prio.c*
- *os_q.c*
- *os_sem.c*
- *os_stat.c*
- *os_task.c*
- *os_tick.c*
- *os_time.c*
- *os_tmr.c*
- *os_type.h*
- *os_var.c*

μC/CPU Files

\Micrium\Software\uC-CPU

This directory contains the CPU definition file which declares #define constants for CPU alignment, endianess, and other generic CPU properties.

- *cpu_core.c*
- *cpu_core.h*
- *cpu_def.h*

\Micrium\Software\uC-CPU\RL78\IAR

This directory defines the Micriμm portable data types for 8-, 16-, and 32-bit signed and unsigned integers. These allow the code to be independent of processor and compiler word size definitions.

- *cpu.h*

μC/LIB Files

\Micrium\Software\uC-LIB

This directory provides `#defines` for useful constants and macros.

- *lib_ascii.c*
- *lib_ascii.h*
- *lib_def.h*
- *lib_math.c*
- *lib_math.h*
- *lib_mem.c*
- *lib_mem.h*
- *lib_str.c*
- *lib_str.h*

\Micrium\Software\uC-LIB\Doc

This directory contains the documentation for **μ**C/LIB.

Application Code

\Micrium\Software\EvalBoards\Renesas\YRDKRL78G13\IAR\<OS>

This directory contains the source code for the example application.

- ***app.c*** contains the test code for the example application including calls to the functions that start multitasking within **μ**C/OS-II or **μ**C/OS-III registers tasks with the kernel, and updates the user interface (LEDs). Furthermore, it uses Renesas Glyph library to display a message on the LCD. The initialization functions for supplementary installed modules are called from this file as well.
- ***app_cfg.h*** is a configuration file which specifies stack sizes and priorities for all user tasks as well as `#defines` for important global application constants.
- ***app_hooks.c*** contains application defined task hooks.
- ***cpu_cfg.h*** is the **μ**C/CPU configuration file.
- ***includes.h*** is the master include file used by the application.
- ***intrinsics.h*** contains the declaration of intrinsic functions for the RL78.
- ***ior5f100LE.h*** Declarations of SFR registers, interrupt and call vector addresses for RL78.
- ***ior5f100LE_ext.h*** Declarations of extended SFR registers for RL78.
- ***low_level_initialization.c*** *and* ***low_level_initialization.h*** basic microcontroller initialization.
- ***macros.h*** and ***types.h*** data type definitions.
- ***os_cfg.h*** is the **μ**C/OS-II or **μ**C/OS-III configuration file.
- ***uCOS-II.**** and ***uCOS-III.**** are IAR project files.

\Micrium\Software\EvalBoards\Renesas\YRDKRL78G13\IAR\BSP

This directory contains the Board Support Package for the YRDKRL78G13 evaluation board.

- ***bsp.c*** contains the board support package functions which initialize critical processor functions (e.g., the clock generation unit) and provide support for peripherals such as the LEDs.
- ***bsp.h*** contains constants and prototypes for functions which may be called by the user.
- ***cpu_bsp.c*** initializes, starts, and manipulates CPU timestamp timer.

\Micrium\Software\EvalBoards\Renesas\YRDKRL78G13\IAR\BSP\GlyphLib

This directory and its subfolders contains Renesas Glyph library. This pre-compiled library is used in the application code to display a message on the LCD.

4. μC/OS-II Application Code

The example application described in this appnote, *AN-uCOS-II-uCOS-III-Renesas-YRDKRL78G13*, is a demonstration of **μ**C/OS-II and **μ**C/OS-III for the Renesas RL78 processor on the YRDKRL78G13 evaluation board. The basic procedure for setting up and using each of these can be easily grasped 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.

Functions of interest located in app.c are:

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()`** performs multiple function calls to: initializes the modules used in the BSP and create an additional application task. It uses Renesas Glyph library to display a message on the LCD; then it enters an infinite loop which toggles the LEDs.

Listing 4-1, `main()`

```
int  main (void)
{                                                                       /* 1 */
#if (OS_TASK_NAME_EN > 0)
    CPU_INT08U  err;
#endif

#if (CPU_CFG_NAME_EN == DEF_ENABLED)
    CPU_ERR     cpu_err;
#endif

    BSP_PreInit();                                                      /* 2 */
    CPU_Init();                                                         /* 3 */
    Mem_Init();                                                         /* 4 */
    Math_Init();

#if (CPU_CFG_NAME_EN == DEF_ENABLED)
    CPU_NameSet((CPU_CHAR *)"TMPM366FD",
                (CPU_ERR  *)&cpu_err);
#endif

    OSInit();                                                           /* 5 */
    OSTaskCreateExt((void (*)(void *)) App_TaskStart,                   /* 6 */
                    (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_EN > 0)                                               /* 7 */
    OSTaskNameSet(APP_CFG_TASK_START_PRIO, "Start", &err);
#endif
    OSStart();                                                          /* 8 */
    return (1);
}
```

L4-1(1) As with most C applications, the code starts in `main()`.

L4-1(2) `Bsp_PreInit()` performs all the board support package initialization that does not request an OS service (such as I/O, LEDs, external memories, etc.).

L4-1(3) `CPU_Init()` initializes the **μ**C/CPU module. `CPU_NameSet()` sets the CPU Host Name.

L4-1(4) `Mem_Init()` Initializes the memory management module and `Math_Init()` initializes the mathematical module.

L4-1(5) `OS_Init()` must be called before creating a task or any other kernel object, as must be done with all **μ**C/OS-II applications.

L4-1(6) At least one task must be created. `OSTaskCreateExt()` will create the startup task `App_TaskStart()`.

L4-1(7) The startup task is given the name “Start”. Task names can be displayed during debugging using C-Spy with Micrium’s Kernel Awareness Plug-In.

L4-1(8) Finally multitasking under **μ**C/OS-II is started by calling `OSStart()`. The operating system will then begin executing `App_TaskStart()` since that is the highest-priority task created (both `OS_TaskStat()` and `OS_TaskIdle()` having lower priorities).

Listing 4-2, `App_TaskStart()`

```
static  void  App_TaskStart (void *p_arg)
{
    (void)p_arg;
    BSP_PostInit();                                                     /* 1 */

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

    APP_TRACE_INFO(("Creating Application Events...\n\r"));
    App_EventCreate();                                                  /* 3 */
    APP_TRACE_INFO(("Creating Application Tasks...\n\r"));
    App_TaskCreate();                                                   /* 4 */
                                                                    /* 5 */
    T_glyphError  err;
    err = GlyphOpen(&G_lcd, 0);
    if (err == GLYPH_ERROR_NONE)
    {
        GlyphNormalScreen(G_lcd);
        GlyphClearScreen(G_lcd);
        GlyphSetFont(G_lcd, GLYPH_FONT_LOGOS);
        GlyphSetXY (G_lcd, 6, 0);
        GlyphChar(G_lcd, 0);
        GlyphSetFont(G_lcd, GLYPH_FONT_5_BY_7);
        GlyphSetXY (G_lcd, 40, 16);
        GlyphString(G_lcd, "RL78 RDK", 8);
        GlyphSetFont(G_lcd, GLYPH_FONT_8_BY_8);
        GlyphSetXY (G_lcd, 20, 40);
        GlyphString(G_lcd, "MICRIUM", 7);
        GlyphSetFont(G_lcd, GLYPH_FONT_5_BY_7);
        GlyphSetXY (G_lcd, 30, 48);
        GlyphString(G_lcd, "uC/OS-II", 8);
    }

    while (DEF_TRUE) {                                                  /* 6 */
        BSP_LED_Toggle(0);
        OSTimeDlyHMSM(0, 0, 1, 0);                                      /* 7 */
    }
}
```

L4-2(1) `BSP_Init()` initializes the Board Support Package drivers that are related to the OS or use an OS service (semaphores, mutexes, queues, etc).

L4-2(2) `OSStatInit()` initializes **μ**C/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.

L4-2(3) `App_EventCreate()` creates all the μC/OS-II application events.

L4-2(4) `App_TaskCreate()` creates all the application tasks.

L4-2(5) Displays message on the LCD using Renesas Glyph library.

L4-2(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 on which the LEDs are toggled every second.

L4-2(7) All tasks managed by μC/OS-II should yield the CPU at some point, typically through a time delay.

5. μC/OS-III Application Code

This example code should serve as a beginning template for further **μ**C/OS-III use.

Functions of interest located in *app.c* are:

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()`** performs multiple function calls to: initializes the modules used in the BSP and create an additional application task. It uses Renesas Glyph library to display a message on the LCD; then it enters an infinite loop which toggles the LEDs.

Listing 5-1, `main()`

```
int  main (void)
{
    OS_ERR   os_err;
    CPU_Init();                                                          /* 1 */
    BSP_PreInit();                                                       /* 2 */
    OSInit(&os_err);                                                     /* 3 */
    OSTaskCreate((OS_TCB      *)&App_TaskStartTCB,                       /* 4 */
                 (CPU_CHAR    *)"Start",
                 (OS_TASK_PTR  )App_TaskStart,
                 (void        *)0,
                 (OS_PRIO      )APP_CFG_TASK_START_PRIO,
                 (CPU_STK     *)&App_TaskStartStk[0],
                 (CPU_STK_SIZE )APP_CFG_TASK_START_STK_SIZE_LIMIT,
                 (CPU_STK_SIZE )APP_CFG_TASK_START_STK_SIZE,
                 (OS_MSG_QTY   )0u,
                 (OS_TICK      )0u,
                 (void        *)0,
                 (OS_OPT       )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                 (OS_ERR      *)&os_err);

    OSStart(&os_err);                                                    /* 5 */
    (void)&os_err;
    return (0);
}
```

L5-1(1) `CPU_Init()` initialize **μ**C/CPU. **μ**C/CPU contains CPU related code to enable/disable interrupts, enter/exit to/from critical sections among other functions.

L5-1(2) The Board Support Package pre-initialization, Bsp_`PreInit()` function is called. This function will perform all the hardware initialization required before the OS is initialized. Most of the time this function will disable all interrupts to make sure the application does not get interrupted until it is fully initialized.

L5-1(3) `OS_Init()` must be created before creating a task or any other kernel objects, as this must be done with all **μ**C/OS-III applications.

L5-1(4) At least one task must be created. `OSTaskCreate()` will create the startup task.

L5-1(5) Finally multitasking under **μ**C/OS-III is started by calling `OSStart()`.
The operating system will then begin executing `AppTaskStart()` since that is the highest-priority task created (both `OS_TaskStat()` and `OS_TaskIdle()` having lower priorities).

Listing 5-2, `App_TaskStart()`

```
static  void  App_TaskStart (void *p_arg)
{
    OS_ERR  err;
    (void)p_arg;
    BSP_PostInit();                                                     /* 1 */
#if (OS_CFG_STAT_TASK_EN > 0u)
    OSStatTaskCPUUsageInit(&err);                                       /* 2 */
#endif
    App_TaskCreate();                                                   /* 3 */
    App_ObjCreate();
#ifdef CPU_CFG_INT_DIS_MEAS_EN
    CPU_IntDisMeasMaxCurReset();
#endif
#if (OS_CFG_STAT_TASK_EN > 0u)
    OSStatReset(&err);
#endif
                                                                        /* 4 */
    T_glyphError  _err;
    _err = GlyphOpen(&G_lcd, 0);
    if (_err == GLYPH_ERROR_NONE)
    {
        GlyphNormalScreen(G_lcd);
        GlyphClearScreen(G_lcd);
        GlyphSetFont(G_lcd, GLYPH_FONT_LOGOS);
        GlyphSetXY (G_lcd, 6, 0);
        GlyphChar(G_lcd, 0);
        GlyphSetFont(G_lcd, GLYPH_FONT_5_BY_7);
        GlyphSetXY (G_lcd, 40, 16);
        GlyphString(G_lcd, "RL78 RDK", 8);
        GlyphSetFont(G_lcd, GLYPH_FONT_8_BY_8);
        GlyphSetXY (G_lcd, 20, 40);
        GlyphString(G_lcd, "MICRIUM", 7);
        GlyphSetFont(G_lcd, GLYPH_FONT_5_BY_7);
        GlyphSetXY (G_lcd, 30, 48);
        GlyphString(G_lcd, "uC/OS-III", 9);
    }

    while (DEF_TRUE) {                                                  /* 5 */
       BSP_LED_Toggle(0);
       OSTimeDlyHMSM(0u, 0u, 0u, 500u,
                     OS_OPT_TIME_HMSM_STRICT,
                      &err);
    }
}
```

L5-2(1) The Board support package post-initialization `BSP_PostInit()` function is called. This function will initialize all the hardware drivers that require OS services such semaphores, queues, mutexes, etc.

L5-2(2) `OSStatTaskCPUUsageInit()` Initialize the μC/OS-III statistics task**.** The statistics task calculates the CPU usage and the stack usage if the option is enabled in the *os_cfg.h* file**.**

L5-2(3) `App_TaskCreate()` and `App_ObCreate()` creates the application's task and the application's kernel objects(e.g. semaphores, queues, mutexes, etc).

L5-2(4) Displays message on the LCD using Renesas Glyph library.

L5-2(5) Any task managed by μC/OS-III must either enter an infinite loop 'waiting' for event to occur or terminate itself. This task enters in an infinite loop on which the LEDs are toggled every 500ms.

6. Board Support Package (BSP)

The Board Support Package (BSP) provides functions to facilitate the user in porting his/her application code. Several functions within the *bsp.c* file, are intended to be called directly by the user, these functions are prototyped within *bsp.h*.

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

The user functions provided within the *bsp.c* file are divided into several categories, one for each module. Each module contains several local functions which may configure port pins for the respective module or handle other miscellaneous tasks which the user does not need to manage. With the exception of the functions associated with the timer module, the functions which will be discussed in this section will be limited to the global function which may be utilized by the user.

The global functions defined in *bsp.c* (and prototyped in *bsp.h*) may be divided into the following categories: BSP services, and LED services.

The BSP services functions provided are:

- **`BSP_Init()`** - this function is called by the application code after multitasking has begun, and should be called before any other BSP functions are used. `BSP_Init()` initializes the various modules which will be used in the application, including the timer used for **μ**C/OS-II or **μ**C/OS-III tick interrupt.
- **`BSP_IntDisAll()`**- this function is called to disable all interrupts, thereby preventing any interrupts until the processor is ready to handle them.
- **`BSP_IntDisAll()`**- this function enables all interrupts.
- **`BSP_Dly()`**- this function will create a short delay by performing a series of operations, for the delay amount specified in μs.

The BSP services provided are:

- **`LED_On()`** - this function will turn on the LED number that was given as its argument. If the LED number passed is 0, then this operation will be performed on all LEDs.
- **`LED_Off()`** - this function will turn off the LED number that was given as its argument. If the LED number passed is 0, then this operation will be performed on all LEDs.
- **`LED_Toggle()`** - this function will toggle the LED number that was given as its argument. If the LED number passed is 0, then this operation will be performed on all LEDs.

6.02 *cpu_bsp.c*

This file contains the CPU BSP functions. This includes the functions to initialize and start the CPU timestamp timer, to get the current CPU timestamp timer count value, and to convert a 32-/64-bit CPU timestamp from timer counts to microseconds.

6.03 Initialization Functions

Listing 6-1, `BSP_PostInit()`

```
void  BSP_PostInit (void)
{
    CLK_OscInit();                                                  /* 1 */
    TMR_TickInit();                                                 /* 2 */
}
```

L6-1(1) The clock source is initialized.

L6-1(2) The **μ**C/OS-II or **μ**C/OS-III tick interrupt source is initialized.

Listings 9-2 and 9-3 give the **μ**C/OS-II or **μ**C/OS-III timer tick initialization function, `Tmr_TickInit()`, the tick ISR handler, `Tmr_TickISR_Handler()`. These may serve as examples for initializing an interrupt and servicing that interrupt.

Listing 6-2, `Tmr_TickInit()`

```
void  TMR_TickInit (void)
{
    PER0          |=    0x01;                                       /* 1 */
    TPS0           =  0x0008;
    TMR00          =  0x0000;
    TDR00          = (BSP_DLY_CONST / 512) - 1;                    /* 2 */

    TOE0          &= 0x00FE;
    TO0           &= 0x00FE;
    TOM0          &= 0x00FE;
    TOL0          &= 0x00FE;

    PR01L_bit.no4  =      1;
    PR11L_bit.no4  =      1;

    TMIF00         =      0;                                       /* 3 */
    TMMK00         =      0;                                       /* 4 */
    TS0           |=   0x01;
}
```

L6-2(1) Power on Time Array Unit.

L6-2(2) Set interval timer.

L6-2(3) Clear TAU Ch. 0 interrupt request flag.

L6-2(4) Enable TAU Ch. 0 interrupt.

Listing 6-3, `Tmr_TickISR_Handler ()`

```
void  Tmr_TickISR_Handler (void)
{
                                                                    /* 1 */
}
```

L6-3(1) The interrupt is cleared automatically. You may insert any additional handling here.

Licensing

μC/OS-II and **μ**C/OS-III are provided in source form for **FREE** evaluation, for educational use or for peaceful research. If you plan on using **μ**C/OS-II and/or **μ**C/OS-III 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 **μ**C/OS-II and **μ**C/OS-III. 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

μC/OS-III, User's Manual
Micriμm
Micriμm Press, 2010
ISBN 978-0-9823375-9-2

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

Contacts

Micriμm
1290 Weston Road, Suite 306
Weston, FL 33326
USA

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

e-mail: sales@Micrium.com
WEB : 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