Micriμm

Empowering Embedded Systems

μC/OS-II

and

The STMicroelectronics STR711 Microcontrollers

(Using the CrossWorks for ARM toolchain)

Application Note

AN-1711B

www.Micrium.com

Table Of Contents

1.00 Introduction

This document describes example code for using **μC/OS-II** with the STMicroelectronics STR71x Microcontrollers. To test the code, we used a evaluation board which contains a STMicroelectronics STR711 (ARM7TDMI) microcontroller. The simplified block diagramm of a generic evaluation board is shown in Figure 1-1.

This example uses the **μC/OS-II** port described in AN-1014, which allows you to run the STR71x either in ARM or Thumb mode.

We also ported **μC/OS-View** to this board (see Section 1.01, **μC/OS-View**). If you did not purchase **μC/OS-View** from Micriμm, you can 'disable' it by removing the **μC/OS-View** files from the build.

We used Rowley Associates CrossWorks for ARM to demonstrate the examples, but other toolchains can be used. In fact, you only need the evaluation version of CrossWorks for ARM to run the example code.

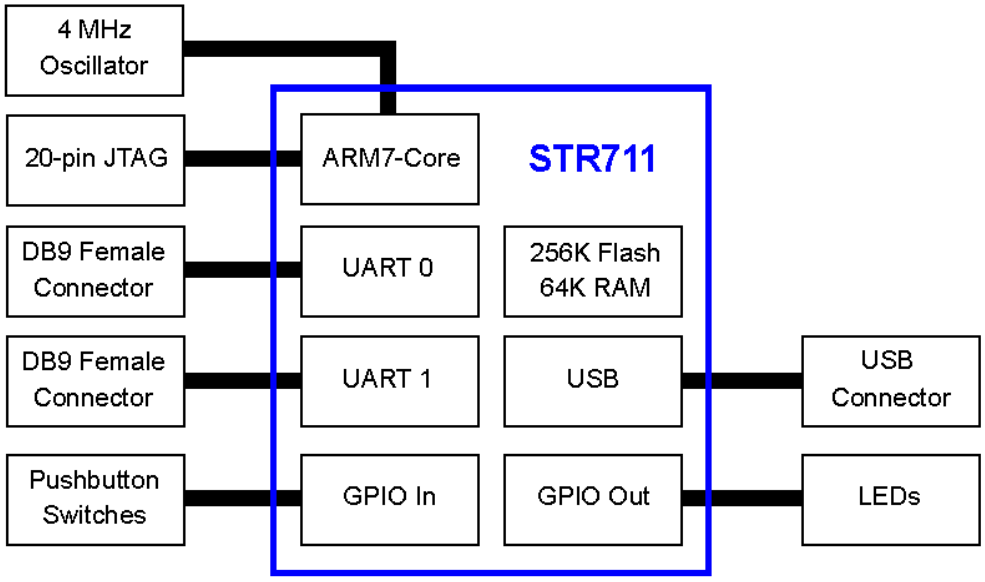

Figure 1-1, The Block Diagram

The 4 LEDs are connected to P0.4, P0.5, P06 and P0.7, the buttons to P0.15 and P1.9.

1.01 μC/OS-View

The application code described in this application note allows you to connect a Windows-based PC to your target and display run-time information about your target in a Window as shown in Figure 1-2. This is done via an add-on module called **μC/OS-View**.

Note that you can 'disable' **μC/OS-View** by removing the **μC/OS-View** files from the build and setting OS_VIEW_MODULE to 0 in os_cfg.h. You would need to do this if you didn't purchase **μC/OS-View** from Micriμm.

μC/OS-View is a combination of a Microsoft Windows application program and code that resides in your target system (in this case, the STR711 Evaluation Board). The Windows application connects with your system via an RS-232C serial port (we used UART0 of the STR711). The Windows application allows you to '**View**' the status of your tasks which are managed by **μC/OS-II**.

μC/OS-View allows you to view the following information from a **μC/OS-II** based product:

The address of the TCB of each task (up to 63 tasks)
The name of each task (up to 63 tasks)
The status (Ready, delayed, waiting on event) of each task
The number of ticks remaining for a timeout or if a task is delayed
The amount of stack space used and left for each task
The percentage of CPU time each task relative to all the tasks
The number of times each task has been 'switched-in'
The execution profile of each task
More.

μC/OS-View also allows you to send commands to your target and allow your target to reply back and display information in a 'terminal window'.

μC/OS-View is licensed on a per-developer basis. In other words, you are allowed to install **μC/OS-View** on multiple PCs as long as the PC is used by the same developer. If multiple developers are using **μC/OS-View** then each needs to obtain their own copy. Contact Micriμm for pricing information.

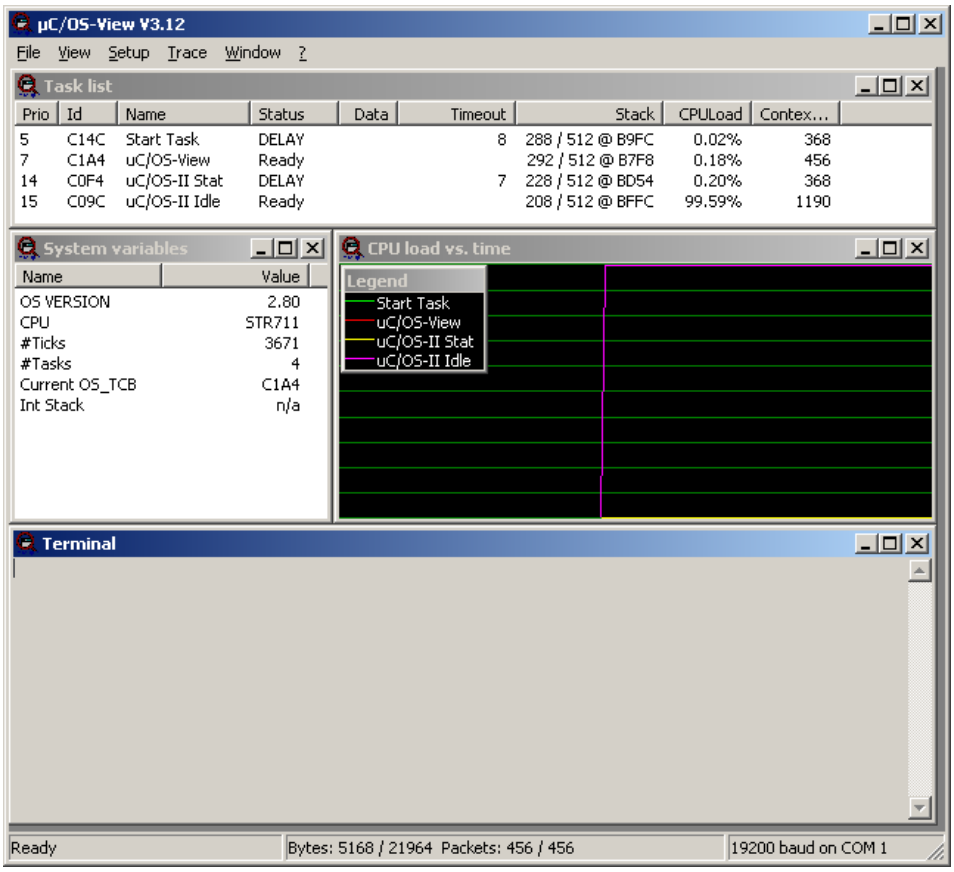

Figure 1-2, μC/OS-View Windows' 'Viewer'

1.02 Directories and Files

The code and documentation of the port are placed in a directory structure according to “AN-2002, μC/OS-II Directory Structure”. Specifically, the files are placed in the following directories:

μC/OS-II:

\Micrium\Software\uCOS-II\Source

This directory contains the processor independent code for **μC/OS-II**. The version used is 2.80 or higher.

\Micrium\Software\uCOS-II\Ports\ARM

This directory is the main directory for ARM7 ports.

\Micrium\Software\uCOS-II\Ports\ARM\Generic

This directory is used to place ‘generic’ ARM7 ports (i.e. ports that can be used with any target board).

\Micrium\Software\uCOS-II\Ports\ARM\Generic\Rowley

This directory contains the standard processor specific files for a **μC/OS-II** port assuming the Rowley Associates toolchain (CrossWorks for ARM). In fact, these files could easily be modified to work with other toolchains. However, you would place the modified files in a different directory. Specifically, this directory contains the following files:

```
os_cpu.h
os_cpu_a.s
os_cpu_c.c
os_dbg_c
```

`os_dbg.c` is included to provide additional information to Kernel Aware debuggers.

The port can work in either ARM or Thumb mode. The port is fully described in application note AN-1014 which is available from the Micriμm web site. The files are:

```
AN-1014.PDF
AN-1014-PPT.PDF
```

μC/OS-View:

\Micrium\Software\uCOSView\Source

This directory contains the processor independent code for **μC/OS-View**. The version used was 1.20. This directory contains the following files:

```
os_view.c
os_view.h
```

\Micrium\Software\uCOSView\Ports\ARM7\STR71x\Rowley

This directory is the main directory for **μC/OS-View** ARM7 ports specifically for STMicroelectronics STR71x series of microcontrollers.

Application Code:

\Micrium\Software\EvalBoards\ST\STR711\Rowley\Ex1

This directory is the directory that contains the source code for Example #1 running on a STR711 evaluation board. This directory contains:

```
app.c
app_cfg.h
includes.h
os_cfg.h
ST_STR711SK_Rowley_Ex1.*
str711.h
threads.js
```

`app.c` contains the test code, and `app_cfg.h` contains application specific configuration information, such as task priorities and stack sizes configuration. `includes.h` contains a master include file used by the application, and `os_cfg.h` is the **μC/OS-II** configuration file. `str711.h` is the header file for the STR711 and `threads.js` is the Plug-In for the CrossWorks debugger. `ST_STR711SK_Rowley_Ex1.*` are the CrossWorks project files.

\Micrium\Software\EvalBoards\ST\STR711\Rowley\BSP

This directory contains the Board Support Package for the STR711 evaluation board.

\Micrium\Software\EvalBoards\ST\STR711\Doc

This directory is the directory that contains the documentation for the STR711 test code.

1.03 Rowley Associates CrossWorks for ARM

We used the Rowley Associates CrossWorks for ARM 1.5 (build 2) to test the STR711 example. You can of course use **μC/OS-II** with other tools. Figure 1-3 shows the project tree in the CrossStudio.

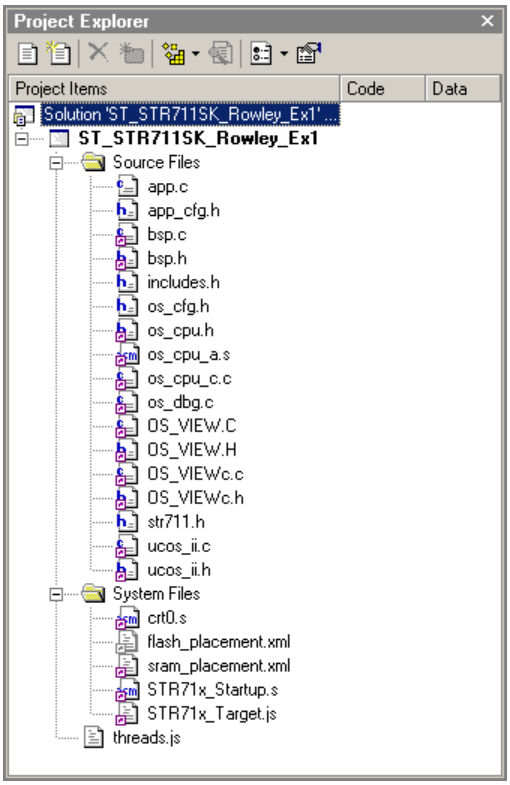

Figure 1-3, CrossWorks Project

Figure 1-4 shows all the tasks created in the STR711 example. Each task can be assigned a name, you can also see the priority and the state.

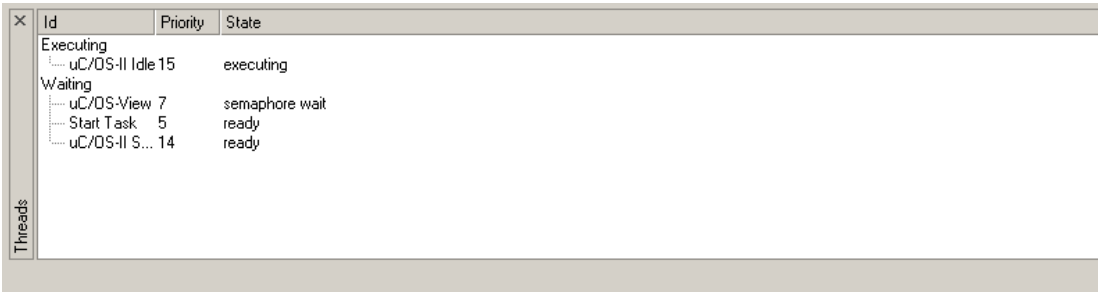

Figure 1-4, CrossWorks debugger Plug-In (threads.js) for μC/OS-II , Task List

2.00 Test Code

As mentioned in the previous section, the test code for this board is found in the following directory and will be briefly described:

```
\Micrium\Software\EvalBoards\ST\STR711\Rowley\Ex1
```

These files in this directory are:

```
app.c
app_cfg.h
includes.h
os_cfg.h
ST_STR711SK_Rowley_Ex1.*
str711.h
threads.js
```

The test code works either in ARM or Thumb mode. In fact, you can simply select ARM or Thumb ***Processor Mode*** (see Figure 2-1) and 'rebuild' the code and it will run just as well.

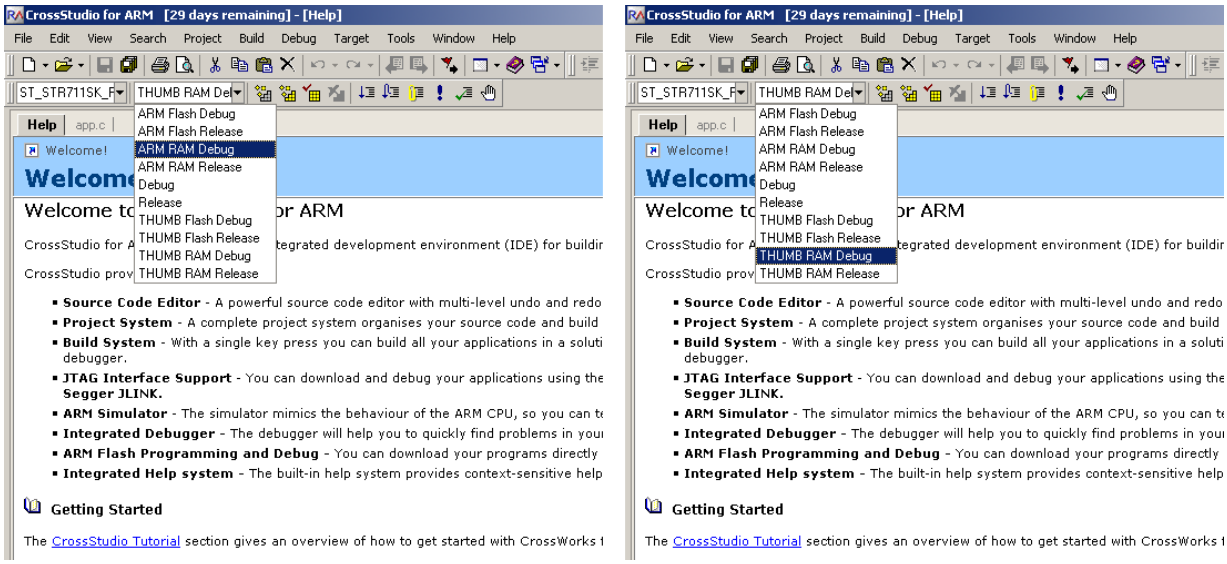

Figure 2-1, Building either ARM (left) or Thumb (right) mode code

As mentioned in AN-1014, **μC/OS-II** must run in SVC mode. Therefore the preprocessor definition SUPERVISOR_START must be set (see Figur 2-2).

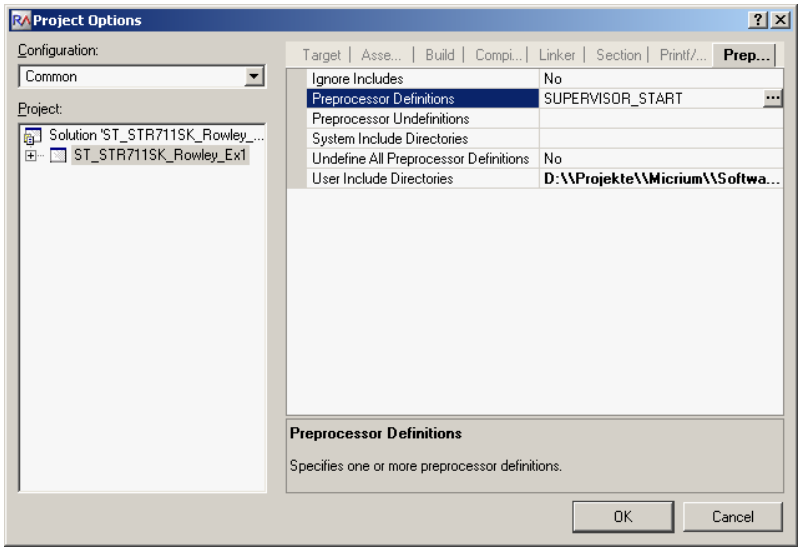

Figure 2-2, Preprocessor Definitions, SUPERVISOR_START

Depending from where you have installed **μC/OS-II** you must set the user include directories. In Figure 2-3 **μC/OS-II** was installed in `D:\Projekte`.

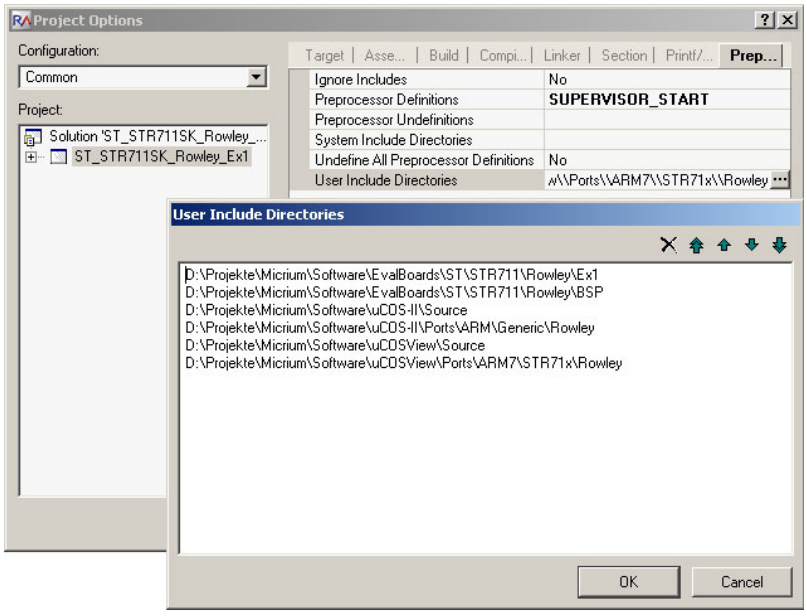

Figure 2-3, User Include Directories

2.01 Test Code, app.c

app.c demonstrates some of the capabilities of **μC/OS-II**. The code doesn't really do anything useful except create an application task that blinks the 4 user LEDs on the evaluation board.

Listing 2-1, main()

```
void main (void)                                                    (1)
{
    INT8U  err;

    BSP_IntDisAll();                                                (2)

    OSInit();                                                       (3)

    OSTaskCreateExt(AppTaskStart,                                   (4)
                    (void *)0,
                    (OS_STK *)&AppTaskStartStk[APP_TASK_START_STK_SIZE - 1],
                    APP_TASK_START_PRIO,
                    APP_TASK_START_PRIO,
                    (OS_STK *)&AppTaskStartStk[0],
                    APP_TASK_START_STK_SIZE,
                    (void *)0,
                    OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

#if OS_TASK_NAME_SIZE > 13
    OSTaskNameSet(APP_TASK_START_PRIO, "Startup", &err);            (5)
#endif

#if OS_TASK_NAME_SIZE > 14
    OSTaskNameSet(OS_IDLE_PRIO, "uC/OS-II Idle", &err);
#if OS_TASK_STAT_EN > 0
    OSTaskNameSet(OS_STAT_PRIO, "uC/OS-II Stat", &err);
#endif
#endif

    OSStart();                                                      (6)
}
```

L2-1(1) As with most C applications, the code starts in `main()`.

L2-1(2) We start off by calling a BSP function (see `bsp.c`) that will disable all interrupts. We do this to ensure that initialization doesn't get interrupted in case we do a 'warm restart'.

L2-1(3) As with all **μC/OS-II** applications, you need to call `OSInit()` before creating any tasks or other kernel objects.

L2-1(4) We then create at least one task (in this case we used `OSTaskCreateExt()` to specify additional information about the task to **μC/OS-II**). It turns out that **μC/OS-II** creates one and possibly two tasks in `OSInit()`. As a minimum, **μC/OS-II** creates an idle task (`OS_TaskIdle()` which is internal to **μC/OS-II**) and `OS_TaskStat()` (if you set `OS_TASK_STAT_EN` to `1` in `OS_CFG.H`). `OS_TaskStat()` is also an internal task in **μC/OS-II**.

L2-1(5) As of V2.6x, you can now name **μC/OS-II** tasks (and other kernel objects) and be able to display task names at run-time or with a debugger. In this case, we name our first task as well as the two internal **μC/OS-II** tasks.

L2-1(6) We finally start **μC/OS-II** by calling `OSStart()`. **μC/OS-II** will then start executing `AppStartTask()` since that's the highest priority task created.

Listing 2-2, AppTaskStart()

```
static  void  AppStartTask (void *p_arg)
{
    p_arg = p_arg;

    BSP_Init();                                                 (1)

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

#if OS_VIEW_MODULE > 0
    OSView_Init(19200);                                         (3)
    OSView_TerminalRxSetCallback(AppTerminalRx);                (4)
    OSView_RxIntEn();                                           (5)
#endif

    LED_Off(0);                                                 (6)
    while (TRUE) {
        for (i = 1; i <= 4; i++) {                              (7)
            LED_On(i);
            OSTimeDlyHMSM(0, 0, 0, 100);
            LED_Off(i);
        }
        for (i = 1; i <= 4; i++) {
            LED_On(5 - i);
            OSTimeDlyHMSM(0, 0, 0, 100);
            LED_Off(5 - i);
        }
    }
}
```

L2-2(1) `BSP_Init()` is called to initialize the Board Support Package – the I/Os, the tick interrupt, and so on. `BSP_Init()` will be discussed in the next section.

L2-2(2) `OSStatInit()` is used to initialize **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 also, performs stack checking for all the tasks that have been created with `OSTaskCreateExt()` with the stack checking option set. Stack checking is useful to have since it gives you warning about possible stack overflow problems.

L2-2(3) `OSView_Init()` is called to initialize the **μC/OS-View** module. Here we need to specify the baud rate of the RS-232C port connecting the the **μC/OS-View** 'viewer'.

L2-2(4) `OSView_TerminalRxSetCallback()` allows you to specify the name of a function that will be called by **μC/OS-View** when characters are typed on the 'Terminal Window' of the **μC/OS-View** viewer.

L2-2(5) `OSView_RxIntEn()` simply enables receive interrupts from the UART used for **μC/OS-View**.

L2-2(6) `LED_Off()` is a BSP function that is used to turn off LEDs on the evaluation board. Passing `0` as an argument specifies to turn off ALL the user LEDs on the board.

L2-2(7) The task then enters an infinite loop. This task simply turns on and then off each LED on the evaluation board one after the other from left to right and then from right to left. Each LED is turned on for 100 mS.

The LEDs are a modification of the original evaluation board, and connected to P0.4 to P0.7.

2.02 Test Code, app_cfg.h

This file is used to establish the task priorities of each of the tasks in your application as well as the stack size for those tasks. The reason this is done here is to make it easier to configure task priorities for your entire application. In other words, you can set the task priorities of all your tasks in one place.

2.03 Test Code, includes.h

`includes.h` is a 'master' header file that contains `#include` directives to include other header files. This is done to make the code cleaner to read and easier to maintain.

2.04 Test Code, os_cfg.h

This file 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 the **μC/OS-II** book. `os_cfg.h` assumes you have **μC/OS-II** V2.80 or higher but also works with previous versions of **μC/OS-II**.

2.05 Test Code, ST_STR711SK_Rowley_Ex1.*

These files are CrossWorks project files.

2.06 Test Code, str711.h

`str711.h` is the header file for the STR711.

2.07 Test Code, threads.js

`threads.js` is the 'Plug-In' for the CrossWorks debugger.

3.00 Board Support Package (BSP)

BSP stands for Board Support Package and provides functions to encapsulate common I/O access functions in order to make it easier for you to port your application code. In fact, you should be able to create other applications using the STR711 board and reuse these functions, thus saving you a lot of time.

The BSP performs the following functions:

- Determine the STR711's CPU clock and peripheral frequencies
- Initialize the interrupt vector table
- Configure the I/Os for the board
- Read the status of the board's push buttons
- Control the board's LEDs
- Handle interrupts
- **μC/OS-View** timer functions
- Handling of **μC/OS-II**'s tick timer
- Set up the EIC (Enhanced Interrupt Controller)

The BSP for the STR711 evaluation board is found in the follow directory.

```
\Micrium\Software\EvalBoards\ST\STR711\Rowley\BSP
```

The BSP files are:

```
bsp.c
bsp.h
```

3.01 Board Support Package, bsp*.*

We will not be discussing every aspect of the BSP but only cover topics that require special attention.

Your application code must call `BSP_Init()` to initialize the BSP. `BSP_Init()` in turn calls other functions as needed.

Listing 3-1, BSP_Init()

```
void BSP_Init (void)
{
    PRCCU_PLL1CR                       = 0x00000070;              (1)
    PRCCU_CFR                         |= 0x00000003;

    BSP_IRQ_VECTOR_ADDR                = 0xE59FF018;              (2)
    BSP_IRQ_ISR_ADDR                   = (INT32U)OS_CPU_IRQ_ISR;

    BSP_FIQ_VECTOR_ADDR                = 0xE59FF018;
    BSP_FIQ_ISR_ADDR                   = (INT32U)OS_CPU_FIQ_ISR;

    BSP_UNDEF_INSTRUCTION_VECTOR_ADDR = 0xEAFFFFFE;               (3)
    BSP_SWI_VECTOR_ADDR                = 0xEAFFFFFE;
    BSP_PREFETCH_ABORT_VECTOR_ADDR     = 0xEAFFFFFE;
    BSP_DATA_ABORT_VECTOR_ADDR         = 0xEAFFFFFE;
    BSP_FIQ_VECTOR_ADDR                = 0xEAFFFFFE;

    BSP_Set_CPU_ClkFreqPeripheral();                              (4)

    while (BSP_Peripheral_Clk1_Freq != BSP_CPU_FREQ) {            (5)
        BSP_Set_CPU_ClkFreqPeripheral();
    }

    BSP_IO_Init();                                                (6)

    EIC_Init();                                                   (7)

    LED_Init();                                                   (8)

    Tmr_TickInit();                                               (9)
}
```

L3-1(1) The STR711's PLL is set up to produce a CPU clock frequency of 32 MHz. Assumed we have a CK input frequency of 4 MHz.

L3-1(2) At location `0x00000018` we 'force' the opcode for `LDR PC,[PC,#0x18]` such that when the CPU recognizes an IRQ interrupt, it will load the contents of location `0x00000038` into the PC (i.e. the address of `OS_CPU_IRQ_ISR()`). The same applies for the FIQ, except that we jump to `OS_CPU_FIQ_ISR()`.

L3-1(3) We then use instructions that loop to themselves for the other exceptions.

L3-1(4) `BSP_Set_CPU_ClkFreqPeripheral()` reads the appropriate STR711 registers to determine the frequency at which the board's peripherals are running.

L3-1(5) After modifying registers in the Power, Reset, Clock, and Control Unit, several CPU cycles must elapse before a new clock frequency takes effect. Therefore, we continuously call `BSP_Set_CPU_ClkFreqPeripheral()` until it indicates that the desired CPU frequency is

being used.

L3-1(6) We then call `BSP_IO_Init()` to initialize the I/O ports.

L3-1(7) We then call `EIC_Init()` to initialize the interrupt controller.

L3-1(8) We initialize the LED services provided by the BSP. At this point, your application can call `LED_On()`, `LED_Off()` or `LED_Toggle()` to turn on, off and toggle (respectively) the board's LEDs.

L3-1(9) We then call `Tmr_TickInit()` which will initialize Timer #0 to generate interrupts for the **μC/OS-II** clock tick. The code for this function is described below.

Listing 3-2, Tmr_TickInit()

```
void Tmr_TickInit (void)
{
    INT8U  err;


    BSP_Tmr0_Rst_Value = (BSP_Peripheral_Clk2_Freq / (BSP_TMR0_PRESCALER + 1)) /
                         OS_TICKS_PER_SEC;                                          (1)

    err                = BSP_VectSet((INT16U)BSP_TMR0_INT,(BSP_PFNCT)Tmr_TickISR_Handler);

    if (err == BSP_VECT_SET) {
        EIC->SIR[BSP_TMR0_INT] |= 0x00000001;                                       (2)
    }
    EIC_IER   = 1 << BSP_TMR0_INT;

    TIM0_CR2  = 0x4000 | BSP_TMR0_PRESCALER;
    TIM0_CR1  = 0x8040;
    TIM0_OCAR = BSP_Tmr0_Rst_Value;
    TIM0_CNTR = 0xFFF0;
}
```

L3-2(1) A reset value for Timer #0 is calculated so that `OS_TICKS_PER_SEC` tick interrupts will be received each second.

L3-2(2) A pointer to `Tmr_TickISR_Handler()`, the handler for the tick interrupt, is passed to `BSP_VectSet()`. `BSP_VectSet()` will place the pointer in a table of interrupt handlers that is accessed whenever an interrupt occurs. A priority for the tick interrupt is also set.

When Timer #0 issues an interrupt, the processor vectors to `OS_CPU_IRQ_ISR()` which then calls `OS_CPU_IRQ_ISR_Handler()` (see `bsp.c`). `OS_CPU_IRQ_ISR_Handler()` reads the EIC to obtain an index to the table of interrupt handlers set up by `BSP_VectSet()`. When a tick interrupt occurs, `Tmr_TickISR_Handler()`, which is shown in Listing 3-3, will be called.

Listing 3-3, Tmr_TickISR_Handler()

```
void Tmr_TickISR_Handler (void)
{
    TIM0_SR  &= ~0x4000;                        (1)
    EIC_IPR   = 0x00000001;                     (2)
    TIM0_OCAR = BSP_Tmr0_Rst_Value;             (3)
    TIM0_CNTR = 0xFFF0;                         (4)

    OSTimeTick();                               (5)
}
```

L3-3(1) The timer's interrupt pending bit is cleared.

L3-3(2) The timer's interrupt pending bit within the EIC is also cleared.

L3-3(3) The timer is reset to the value that was calculated in `Tmr_TickInit()`.

L3-3(4) A write to the timer's counter register restarts the timer.

L3-3(5) `OSTimeTick()` is called to handle the **μC/OS-II** clock tick.

You should note that ALL of your ISRs should be written as 'void MyISR(void)' functions as shown. Refer to **AN-1014** for details.

References

μC/OS-II, The Real-Time Kernel, 2nd Edition
Jean J. Labrosse
R&D Technical Books, 2002
ISBN 1-57820-103-9

Contacts

Rowley Associates Limited
8 Silver Street
Dursley
Gloucestershire
GL11 4ND
UNITED KINGDOM
+44 (0)1453 547916
+44 (0)1453 544068 (FAX)
e-mail: enquiries@rowley.co.uk
WEB: www.rowley.co.uk

R&D Books, Inc.
1601 W. 23rd St., Suite 200
Lawrence, KS 66046-9950
USA
(785) 841 1631
(785) 841 2624 (FAX)
e-mail: rushorders@cmpbooks.com
WEB: www.cmpbooks.com

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

Validated Software
Lafayette Business Park
2590 Trailridge Drive East, Suite 102
Lafayette, CO 80026
USA
+1 303 531 5290
+1 720 890 4700 (FAX)
e-mail: Sales@ValidatedSoftware.com
WEB: www.ValidatedSoftware.com