

**NUC1XX Cortex M0 Series
uC/OS-II
Application Note**

Table of Contents

1. Introduction.....	3
1.1 Introduction.....	3
2. User Method.....	4
2.1 Patch usage	4
3. Code Section---SampleNUC100.....	5
3.1 specific #define constants, macros, typedefs and Variable	5
3.2 main function	5
3.3 Execution Result	7
4. Revision History	8

1. Introduction

1.1 Introduction

This document describes the Nuvoton port for uC/OS-II to the Cortex-M0 processor and how to use the patch that the Nuvoton supplied to port.

The patch source code for uC/OS-II is provided in source form along with the Cortex-M0 ports under the Keil uVision3 toolchain.

Note: We only supply the patch for uC/OS-II, the uC/OS-II independent source code and licensing agreements should contact www.Micrium.com.

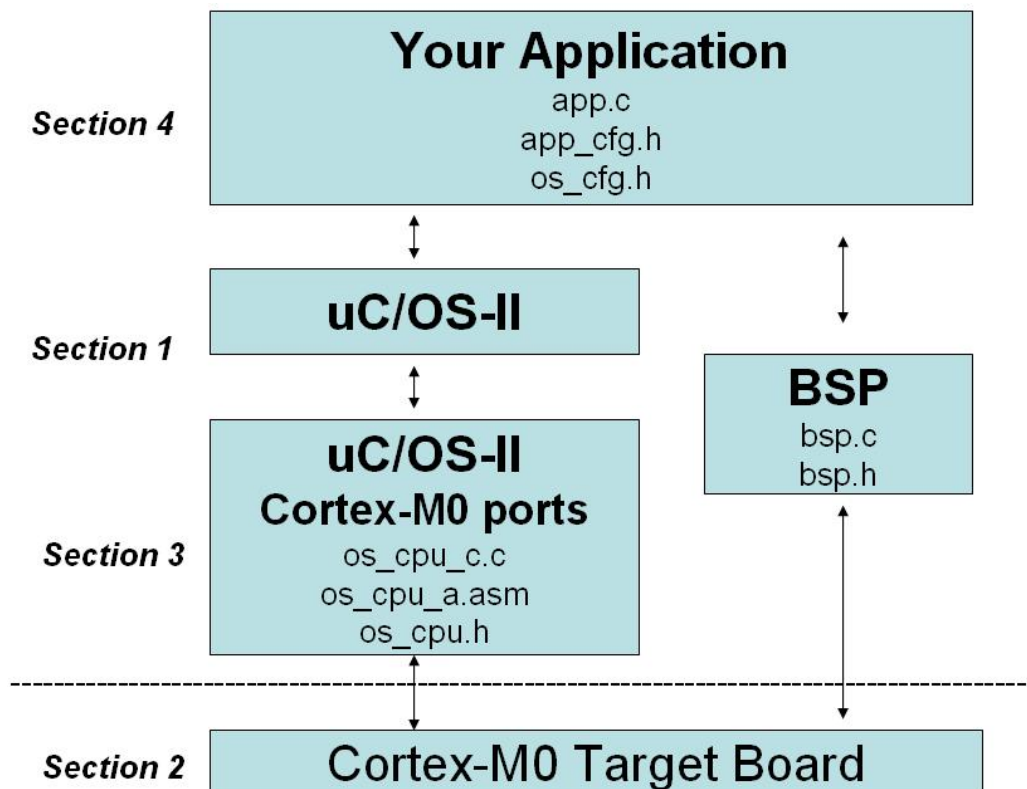


Figure 1-1. Code Block Diagram

2. User Method

2.1 Patch usage

1. Download the uC/OS-II independent source code from the www.Micrium.com (Version 2.86)
2. Put the code into the Micrium\Software\uCOS-II\Source directory.
3. In the directory \Micrium\Software\Evalboards\NUVOTON\NUC100\RealView\OS-Probe, there is the Sample named SampleNUC100 of Keil.

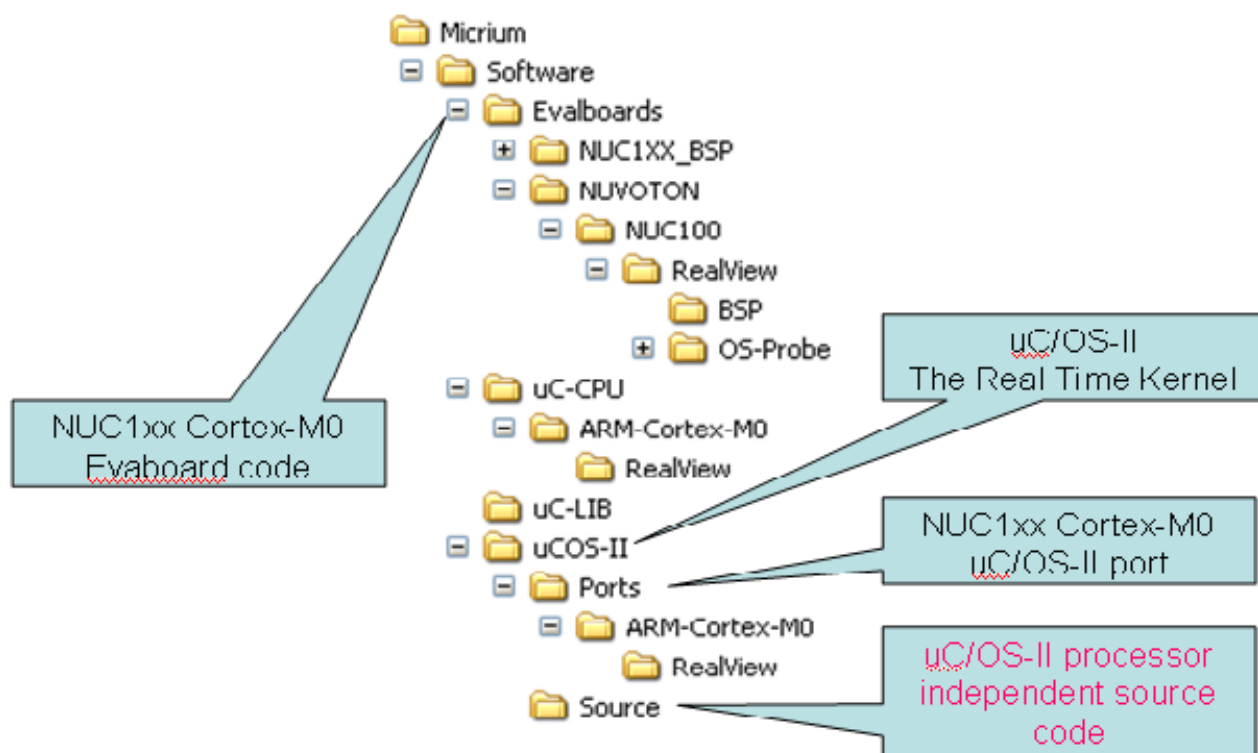


Figure 2-1. NUC1XX Cortex-M0 uC/OS-II Directory Structure

The source code for the uC/OS-II port for the Cortex-M0 with ARM Keil uVision3 is found in the following directory:

\Micrium\Software\uCOS-II\Ports\ARM-Cortex-M0\RealView

The port consists of four files:

- os_cpu.h contains processor and implementation specific #define constants, macros, and typedefs.
- os_cpu_a.asm contains several simple assembly languages functions.
- os_cpu_c.c contains several simple C-language functions, including the function that initializes the task stacks.
- os_dbg.c was added to allow a kernel-aware debugger to extract information about uC/OS-II and its configuration.

3. Code Section---SampleNUC100

3.1 specific #define constants, macros, typedefs and Variable

Define some macros and variables in the app_cfg.h and app.c
As following:

```
#define OS_TASK_TMR_PRIO 26//The priority of the Timer management task

//#define SEM_TEST
#define THREAD_TEST //We use this macro for test sample.
//#define MBOX_TEST
//#define FLAG_TEST
//#define MALLOC_TEST
//#define MEM_TEST
//#define MUTEX_TEST
//#define QUEUE_TEST
//#define TMR_TEST.
```

```
#ifdef THREAD_TEST
OS_STK SYS_Task_Stack[STACKSIZE];
#define SYS_Task_Prio 21
void SYS_Task(void *Id);

OS_STK Task1_Stack[STACKSIZE];
void Task1(void *Id);
#define Task1_Prio 22

OS_STK Task2_Stack[STACKSIZE];
void Task2(void *Id);
#define Task2_Prio 20
#endif
```

3.2 main function

The main function is in the app.c.

```

int main (void)
{
    CPU_INT08U err;
    int i;
    UNLOCKREG();
    SYSCLK->PWRCON.OSC22M_EN = 1;//default is enabled
    SYSCLK->PWRCON.XTL12M_EN = 1;
    SYSCLK->CLKSEL0.HCLK_S = 0;//using 12M as HCLK src
    Delay(100);
    SystemFrequency = 12000000;

    BSP_INIT_UART0();           //init the uart0
    OSInit();
    UNLOCKREG();

    SysTick_Config(120000);

                                /* Initialize "uC/OS-II, The Real-Time Kernel". */

#ifdef THREAD_TEST
    OSTaskCreate(SYS_Task, (void *)0, (OS_STK *)&SYS_Task_Stack[STACKSIZE-1], SYS_Task_Prio);
    OSTaskCreate(Task2, (void *)0, (OS_STK *)&Task2_Stack[STACKSIZE-1], Task2_Prio);
    OSTaskCreate(Task1, (void *)0, (OS_STK *)&Task1_Stack[STACKSIZE-1], Task1_Prio);
#endif
    OSStart();                  /* Start multitasking (i.e. give control to uC/OS-II). */
}

```

```

#ifdef THREAD_TEST
void Task1(void *Id) {
    for(;;) {
        printf("Task1 is running\n"); OSTimeDly(100);
    }
}

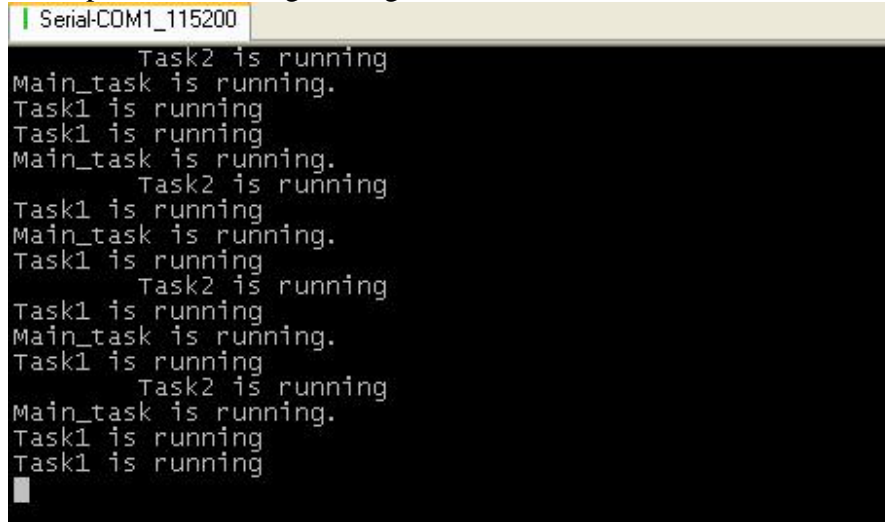
void Task2(void *Id) {
    for (;;) {
        printf("\tTask2 is running\n"); OSTimeDly(200);
    }
}

void SYS_Task(void *Id) {
    for (;;) {
        printf("Main_task is running.\n"); OSTimeDly(150);
    }
}
#endif

```

3.3 Execution Result

The UART0 would output the following message:

A screenshot of a serial terminal window. The title bar at the top reads "Serial-COM1_115200". The terminal area has a black background with white text. The output consists of several lines of status messages, including "Task2 is running", "Main_task is running.", "Task1 is running", and "Task1 is running" repeated multiple times, indicating a sequence of task executions.

```
Serial-COM1_115200
Task2 is running
Main_task is running.
Task1 is running
Task1 is running
Main_task is running.
Task2 is running
Task1 is running
Main_task is running.
Task1 is running
Task2 is running
Task1 is running
Main_task is running.
Task1 is running
Task2 is running
Main_task is running.
Task1 is running
Task1 is running
```

4. Revision History

Version	Date	Description
V1.00	Apr 2, 2010	• Created