在Keil RTX下以多工的方式存取EEPROM

NuMicro® 32 位系列微控制器範例代碼介紹

文件資訊

代碼簡述	本代碼示範在 Keil RTX 程式下以多工的方式存取 EEPROM。
BSP 版本	NUC230_240 Series BSP CMSIS V3.01.001
開發平台	NuEdu-EVB-NUC240 V2.1 NuEdu-Basic01 V2.1

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1 功能介紹

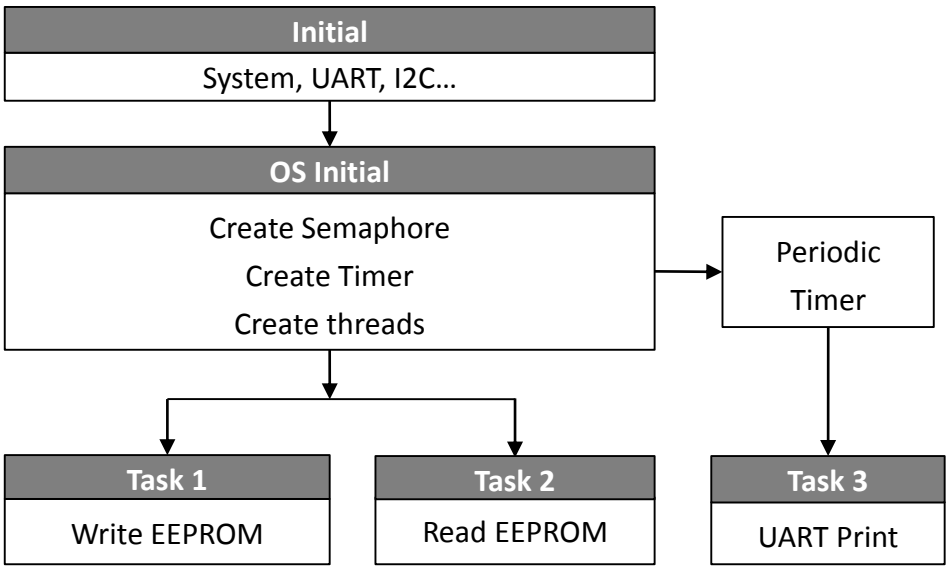
1.1 簡介

當微控制器程式需同時執行許多任務時，可使用實時操作系統(real-time operating system，RTOS)來管理這些任務。RTOS將CPU的處理時間切割成許多時隙 (time slots)，每個時隙進行不同的任務，並且在每個時隙之間進行任務的切換。因為任務間的切換在一秒鐘內可以達到上百次，整體看起來像是這些任務是同時在執行。當程式比較複雜需要許多任務同時執行，就可使用RTOS協助建立和管理這些任務。

Keil MDK (Microcontroller Development Kit) 內建的CMSIS-RTOS，為底層 RTOS和上層應用程式提供一套標準API，包括Keil免費的RTX (Real-Time eXecutive) 核心。當其他RTOS也支援CMSIS-RTOS，透過共同標準的API，則可輕易將程式移植到其他RTOS平台。本範例代碼利用Keil裡提供的CMSIS-RTOS建立RTX核心的系統，進行EEPROM的讀寫。

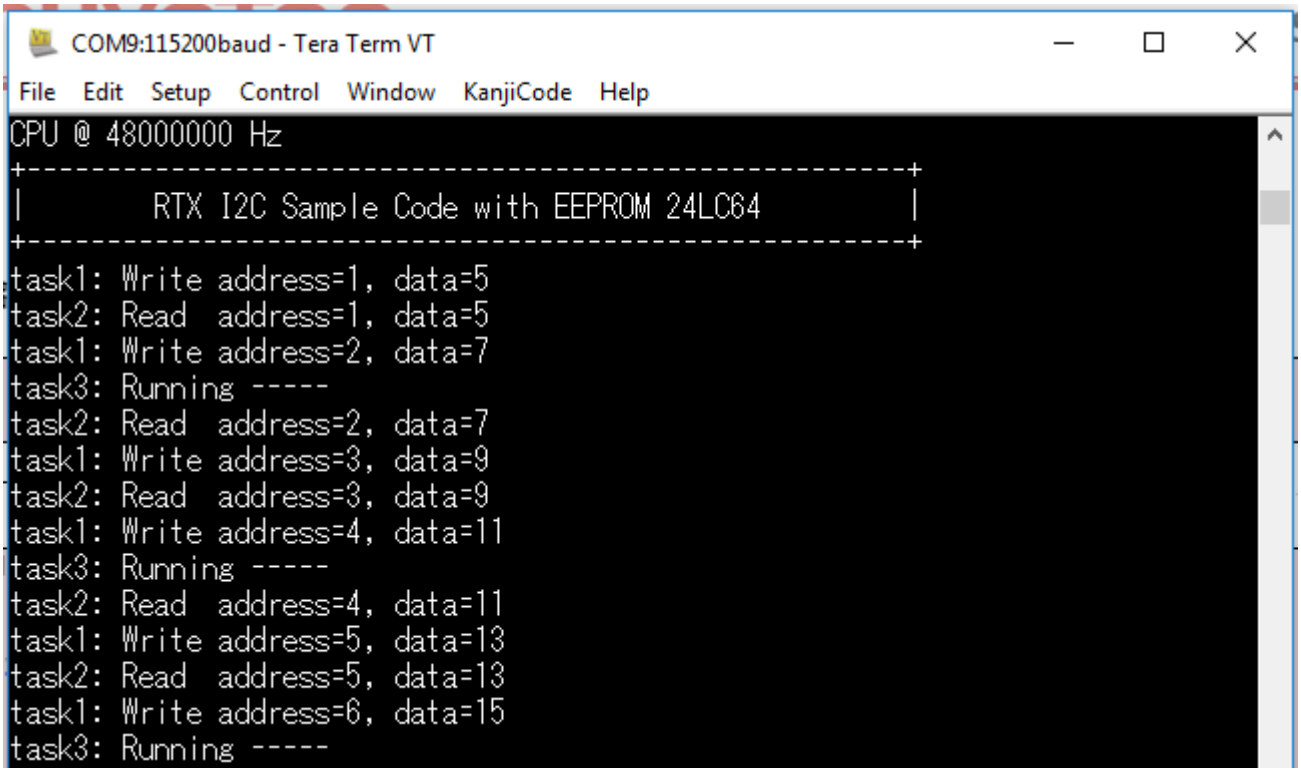
1.2 原理

程式首先做初始化，包括打印訊息的UART和存取EEPROM的I2C。建立一個Semaphore、一個Timer和兩個Thread。兩個Thread分別是對EEPROM的讀寫：其中一個 Thread先寫入資料，釋放一個Semaphore token；另外一個任務則等待完成EEPROM寫入的Semaphore token，再讀取EEPROM的資料。設定週期性的Timer，定時觸發UART打印訊息。

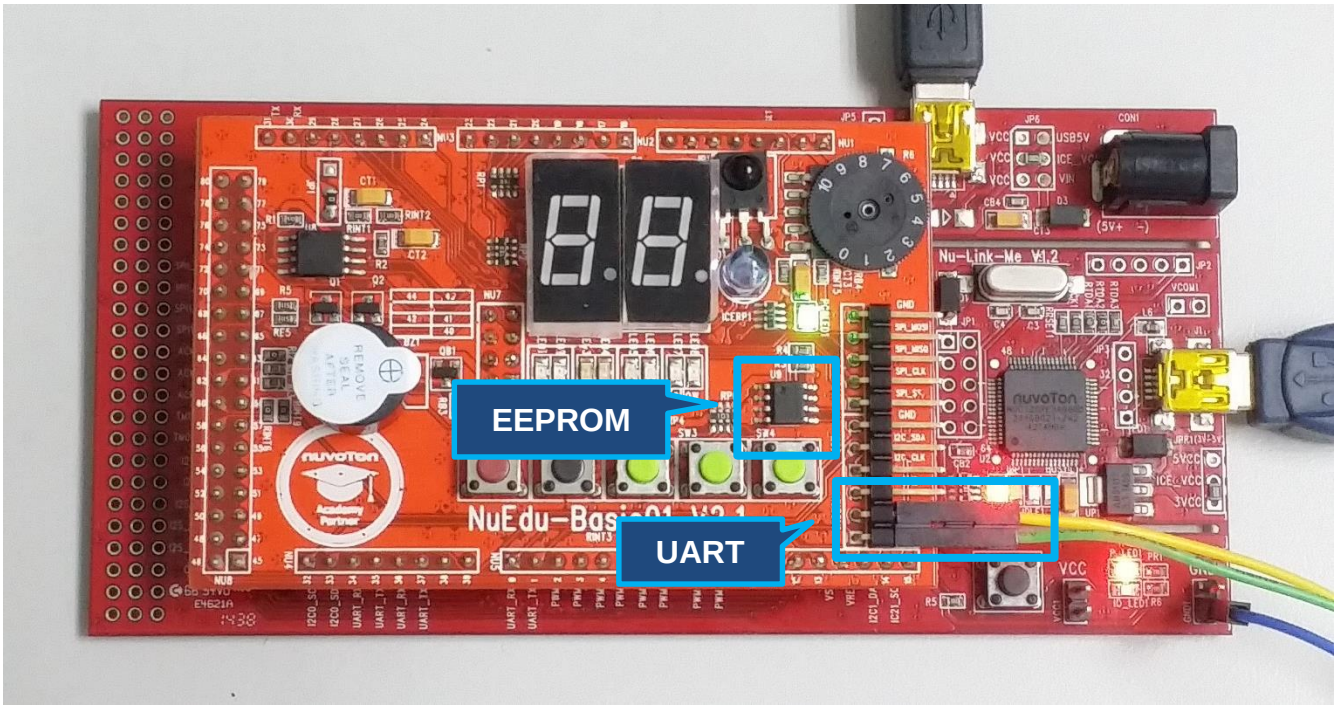


1.3 執行結果

NUC240開發板透過NuBridge連接到電腦，開啟電腦終端機，顯示UART打印的訊息。



下圖為NUC240開發版上EEPROM和UART的位置。



2 代碼介紹

2.1 RTX

Thread

本範例代碼有兩個平行處理的任務，分別宣告其函式原型，並為每個任務宣告各自的ID(t_task1和t_task2)。使用osThreadDef宣告每個任務為thread，也可以用來設定任務的優先等級。

```
/* Thread IDs */
osThreadId t_task1;     // Declare a thread ID for task 1 (EEPROM Write)
osThreadId t_task2;     // Declare a thread ID for task 2 (EEPROM Read)

/* Function Declaration */
void task1_EEPROMwrite(void const *argument);
void task2_EEPROMread(void const *argument);

/* Thread Definition with function, priority, and stack requirements */
osThreadDef(task1_EEPROMwrite, osPriorityNormal, 1, 0);
osThreadDef(task2_EEPROMread, osPriorityNormal, 1, 0);
```

在主程式中使用osThreadCreate函式為每個任務建立新的thread，傳回值指定給該任務的ID。

```
    /* Create task and assign thread IDs */
    t_task1 = osThreadCreate(osThread(task1_EEPROMwrite), NULL);
    t_task2 = osThreadCreate(osThread(task2_EEPROMread), NULL);
```

Semaphore

宣告作為任務之間內部互相溝通的Semaphore，名字為Sem_Arrive，以及宣告其ID變數，Sem_Arrive_id。

```
/* Semaphore ID */
osSemaphoreId Sem_Arrive_id;

/* Semaphore Declaration */
osSemaphoreDef(Sem_Arrive);
```

主程式中呼叫osSemaphoreCreate函式建立新的Semaphore，並將傳回值指定給該Semaphore的ID (Sem_Arrive_id)，設定可用的token數目為1。

```
    /* Create Semaphore without token */
```

```
    Sem_Arrive_id = osSemaphoreCreate(osSemaphore(Sem_Arrive), 1);
```

Timer

宣告一個Timer ID名字為Timer_Delay_id。定義這個Timer的名字為Timer_Delay，time-out時會觸發task3_UARTprint函式。

```
/* Timer ID */
osTimerId Timer_Delay_id;

/* osTimer Declaration */
osTimerDef(Timer_Delay, task3_UARTprint);
```

主程式中呼叫osTimerCreate函式建立新的Timer，操作在週期的模式，沒有傳入參數，並將傳回值指定給該Timer的ID(Timer_Delay_id)。呼叫osTimerStart函式開始Timer的計數，時間為50ms。每50ms就會觸發task3_UARTprint函式執行。

```
    /* Create Timer in periodic mode and without argument */
    Timer_Delay_id = osTimerCreate(osTimer(Timer_Delay), osTimerPeriodic, (void *)NULL);

    /* Start Timer with period 50ms */
    osTimerStart(Timer_Delay_id, 50);
```

task3_UARTprint函式打印程式執行中訊息。

```
void task3_UARTprint(void const *argument)
{
    /* UART print message */
    printf("task3: Running -----\n");
}
```

OS Kernel 初始化和執行

呼叫osKernelInitialize函式，開始進行OS Kernel初始化。

```
    /* Initialize RTOS */
    osKernelInitialize();
```

呼叫osKernelStart函式開始執行OS Kernel以及任務切換。

```
    /* Start the RTOS */
    osKernelStart();
```

2.2 EEPROM

開啟I2C，設定多功能管腳、時鐘、傳輸速率和中斷。

```
    /* Initial I2C */
    I2C_EEPROM_Init();
```

第一個任務裡等有空的Semaphore token，呼叫I2C_EEPROM_Write函式寫入資料，然後釋出一個Semaphore token。

```
void task1_EEPROMwrite(void const *argument)
{
    while(1) {

        /* Wait for semaphore token */
        osSemaphoreWait(Sem_Arrive_id, osWaitForever);

        /* Delay 10 msec */
        osDelay(10);

        /* Access EEPROM address from 0~10 */
        u32Addr++;
        if (u32Addr>10)
            u32Addr=0;

        /* Write data */
         printf("task1: Write address=%d, data=%d \n\r",u32Addr,(u32Addr*2+3));
         I2C_EEPROM_Write(u32Addr,(u32Addr*2+3));

        /* Release semaphore token */
        osSemaphoreRelease(Sem_Arrive_id);
    }
}
```

第二個任務，等有空的Semaphore token，呼叫I2C_EEPROM_Read函式讀取資料，檢查資料數值，然後釋出Semaphore token。

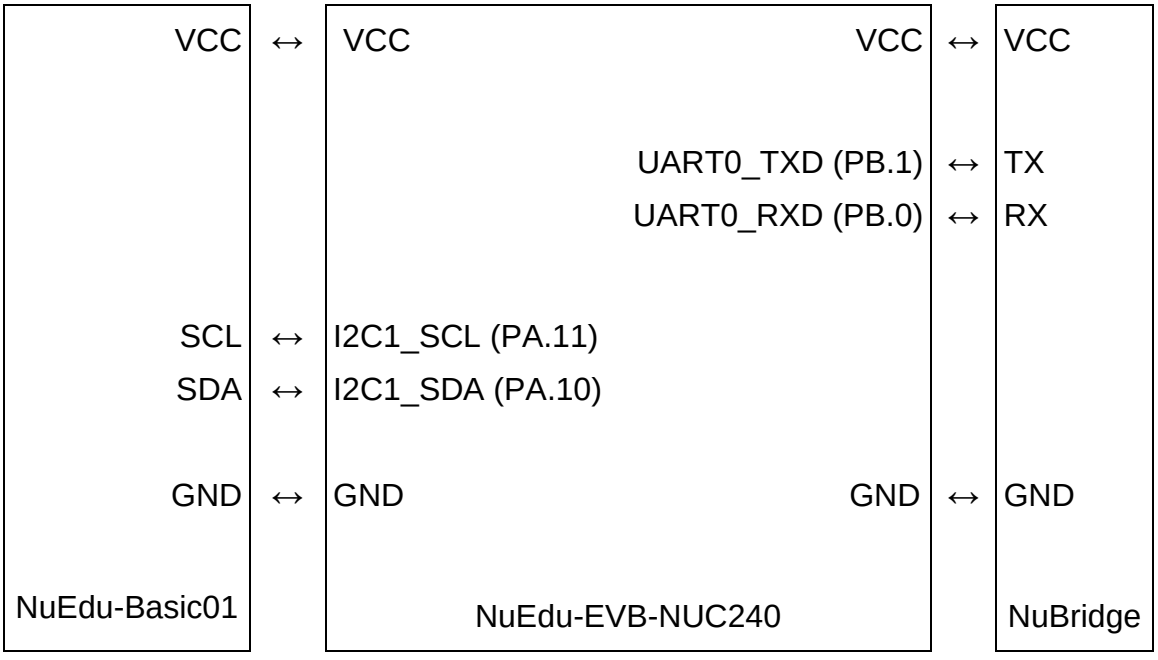
```
void task2_EEPROMread(void const *argument)
{
    uint32_t u32Data;

    while(1)
```

```
    {
        /* Wait for semaphore token */
        osSemaphoreWait(Sem_Arrive_id, osWaitForever);

        /* Read data */
        u32Data = I2C_EEPROM_Read(u32Addr);
        printf("task2: Read  address=%d, data=%d \n\r",u32Addr,u32Data);

        /* Check data is correct or not */
         if(u32Data!=(u32Addr*2+3)){
             printf("I2C Byte Write/Read Failed, Data 0x%x\n", u32Data);
             while(1);
         }

        /* Release semaphore token */
        osSemaphoreRelease(Sem_Arrive_id);

    }
}
```

3 軟體與硬體環境

- **軟體環境**
  - BSP 版本
    - NUC230_240 Series BSP CMSIS V3.01.001
  - IDE 版本
    - Keil uVersion 5.28
- **硬體環境**
  - 電路元件
    - NuEdu-EVB-NUC240 V2.1
    - NuEdu-Basic01 V2.1
    - NuBridge
  - 示意圖

NuEdu-EVB-NUC240 開發版上接 NuEdu-Basic01 版子和 NuBridge。



4 目錄資訊

- EC_NUC240_RTX_EEPROM_V1.00
  - Library — Sample code header and source files
    - CMSIS — Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
    - Device — CMSIS compliant device header file
    - NuEdu — Library for NuEdu-SDK-NUC240 board
    - StdDriver — All peripheral driver header and source files
  - SampleCode
    - ExampleCode — Source file of example code

5 如何執行範例程式

1. 根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，雙擊 NUC240_RTX_EEPROM.uvprojx。
2. 進入編譯模式介面
   a. 編譯
   b. 下載代碼至記憶體
   c. 進入 / 離開除錯模式
3. 進入除錯模式介面
   a. 執行代碼

6 修訂紀錄

Date	Revision	Description
Sep. 30, 2019	1.00	1. 初始發布.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.