RTX下使用ADC，GPIO和USB Mouse

NuMicro® 32 位系列微控制器范例代码介绍

文件信息

代码简述	在 RTX 程序下执行 ADC，GPIO 和 USB Mouse。
BSP 版本	NUC240 Series BSP CMSIS V3.01.001
开发平台	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 功能介绍

1.1 简介

当微控制器程序需同时执行许多任务时，可使用实时操作系统(real-time operating system，RTOS)来管理这些任务。RTOS将CPU的处理时间切割成许多时隙 (time slots)，每个时隙进行不同的任务，并且在每个时隙之间进行任务的切换。因为任务间的切换在一秒钟内可以达到上百次，整体看起来像是这些任务是同时在执行。当程序比较复杂需要许多任务同时执行，就可使用RTOS协助建立和管理这些任务。

Keil MDK (Microcontroller Development Kit) 内建的CMSIS-RTOS，为底层 RTOS和上层应用程序提供一套标准API，包括Keil免费的RTX (Real-Time eXecutive) 核心。当其他RTOS也支持CMSIS-RTOS，透过共同标准的API，则可轻易将程序移植到其他RTOS平台。本范例代码利用Keil里提供的CMSIS-RTOS建立RTX核心的系统，然后执行USB、ADC、UART和GPIO任务。

1.2 原理

在RTX环境下，建立以下4个任务，并且同时执行：

- USB

用4个按键SW1~SW4作为输入，仿真控制鼠标的移动方向(上、下、左或右)。

- ADC

ADC进行电压转换，旋转可变电阻的旋钮可改变输入的电压值。

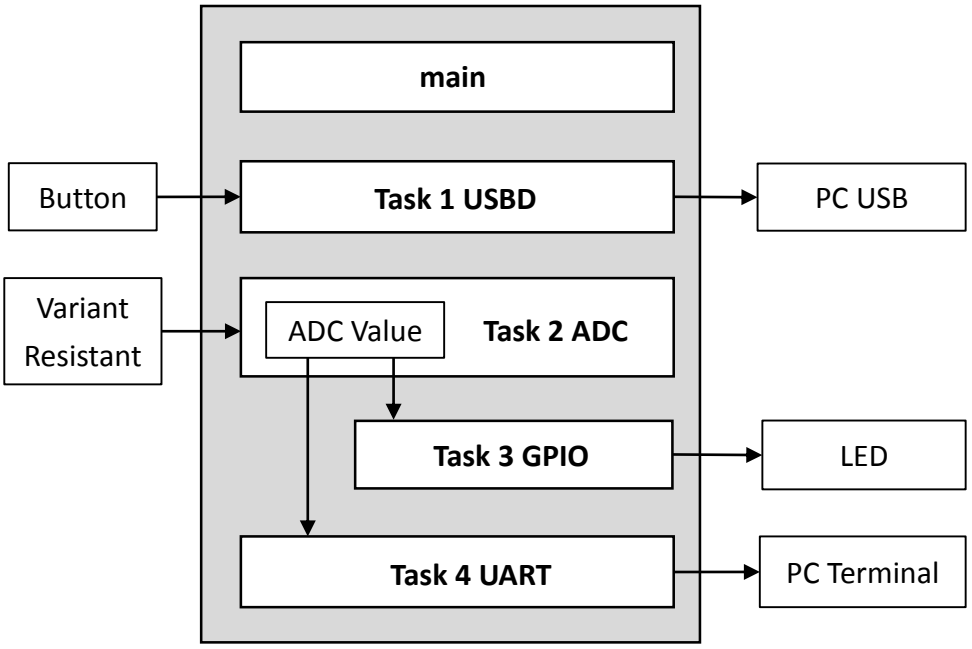
- UART

完成一次ADC电压转换，则打印出ADC的电压转换值。

- GPIO

根据当时的ADC电压转换值，当作LED闪烁速率的输入。

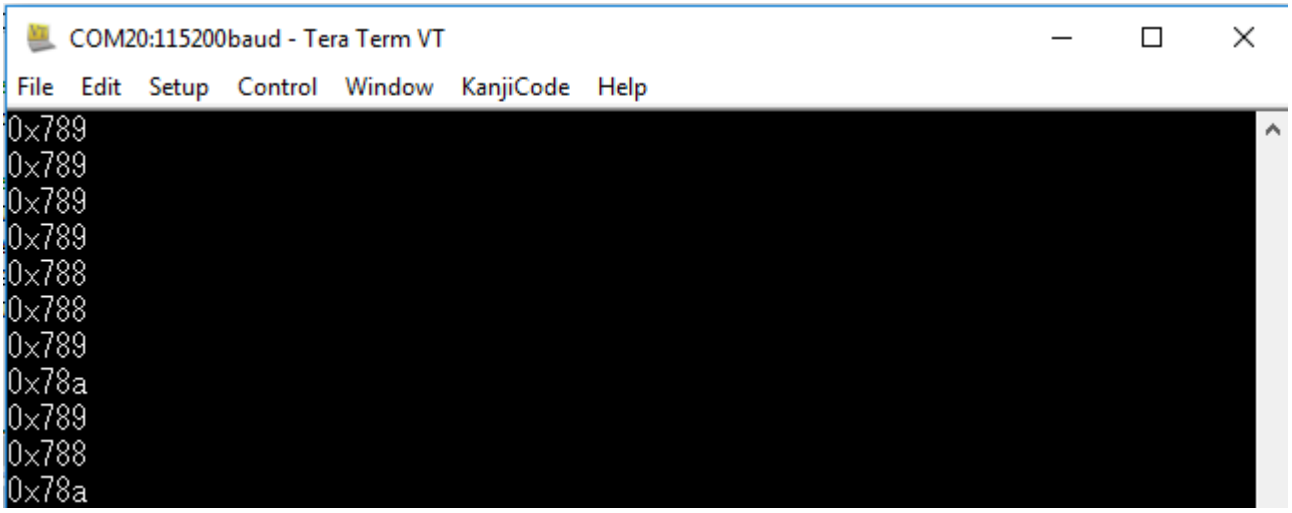
下图描述这4个同时执行的任务所需要的输入、输出周边和任务间的互动关系。



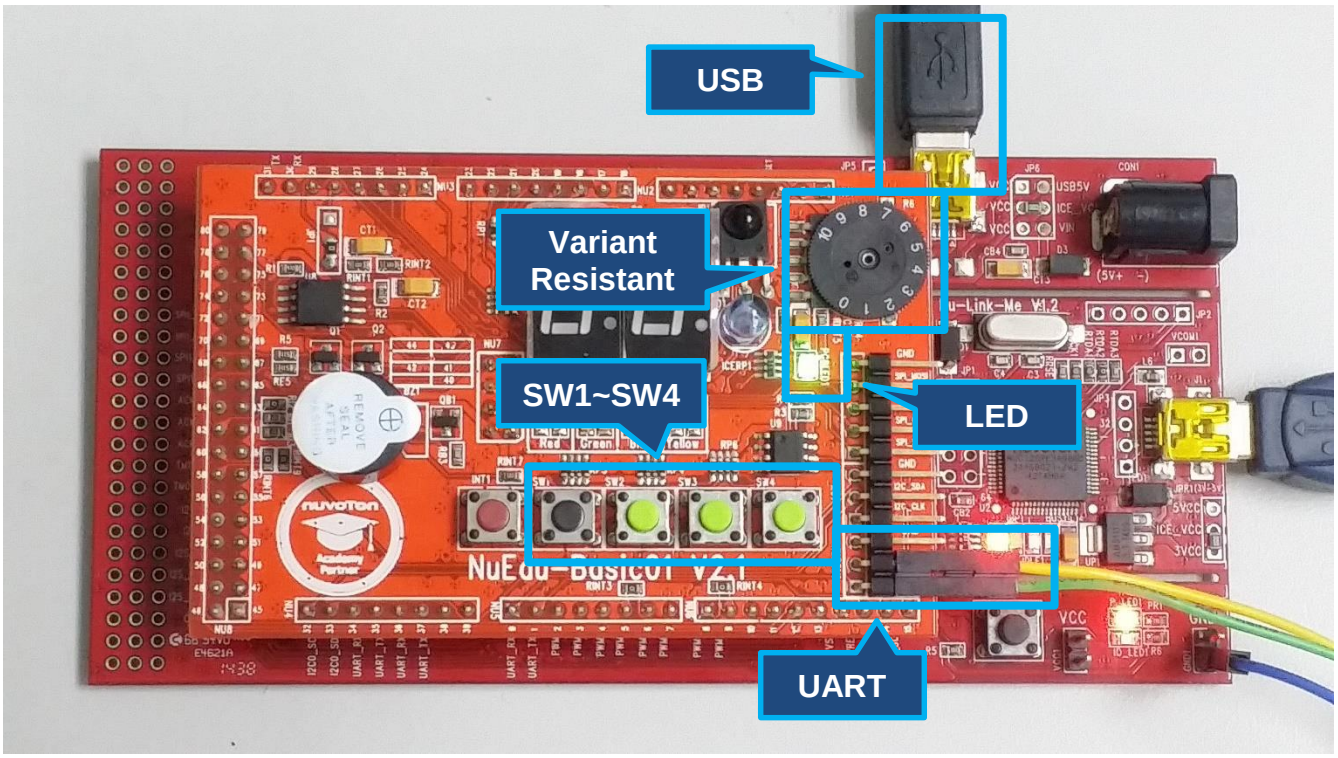
1.3 執行結果

將NUC240开发板的USB连接至计算机，按下开发版上的按钮SW1~SW4，可控制鼠标的上、下、左和往右移动。ADC持续对输入电压进行电压转换，转换出来的电压值做为控制LED闪烁速率的参数，旋转开发版上的可变电阻旋钮，改变ADC的输入电压值，可看到LED的闪烁速度改变。连接UART至NuBridge，开启计算机终端机软件，UART打印ADC的电压转换值。

计算机终端机显示UART打印讯息：



NUC240开发版上所用到的周边和位置：



2 代码介绍

2.1 RTX

Thread宣告

本范例代码有4个平行处理的任务，分别宣告其函式原型，并为每个任务宣告各自的ID，ID的数据格式为osThreadId。

```
/* Thread IDs */
osThreadId t_task1_USB;     // Declare a thread ID for task 1 (USBD mouse)
osThreadId t_task2_ADC;     // Declare a thread ID for task 2 (ADC conversion)
osThreadId t_task3_GPIO;    // Declare a thread ID for task 3 (GPIO LED control)
osThreadId t_task4_UART;    // Declare a thread ID for task 4 (UART print message)

/* Function declaration */
void task1_USB(void const *argument);
void task2_ADC(void const *argument);
void task3_GPIO(void const *argument);
void task4_UART(void const *argument);
```

使用osThreadDef macro宣告每个任务为thread，也可以用来设定任务的优先等级。

```
/* define task as thread function */
osThreadDef(task1_USB, osPriorityNormal, 1, 0);
osThreadDef(task2_ADC, osPriorityNormal, 1, 0);
osThreadDef(task3_UART, osPriorityNormal, 1, 0);
osThreadDef(task4_GPIO, osPriorityNormal, 1, 0);
```

Semaphore宣告

宣告作为任务间内部互相沟通的Semaphore，名字为Sem_AdcDone，以及宣告其ID变量。

```
/* Declare Semaphore */
osSemaphoreDef(Sem_AdcDone);

/* Semaphore ID */
osSemaphoreId Sem_AdcDone_id;
```

OS Kernel 初始化

呼叫osKernelInitialize函式，开始进行OS Kernel初始化。

```
    /* Initialize OS Kernel */
    printf("RTX Start ...\n");
    osKernelInitialize();
```

Semaphore建立

呼叫osSemaphoreCreate函式建立新的Semaphore，并将传回值指定给该Semaphore的ID。一开始没有任何token，当ADC完成一次电压转换，会释出一个token，UART则等ADC转换完成后，打印电压转换结果。

```
    /* Create Semaphore without token */
    Sem_AdcDone_id = osSemaphoreCreate(osSemaphore(Sem_AdcDone), 0);
```

Thread建立

用osThreadCreate函式为每个任务建立新的thread，传回值指定给该任务的ID。

```
    /* Create task and assign thread ID to t_task */
    t_task1_USB = osThreadCreate(osThread(task1_USB), NULL);
    t_task2_ADC = osThreadCreate(osThread(task2_ADC), NULL);
    t_task3_GPIO = osThreadCreate(osThread(task3_GPIO), NULL);
    t_task4_UART = osThreadCreate(osThread(task4_ UART), NULL);
```

开始执行OS Kernel

呼叫osKernelStart函式开始执行OS Kernel以及任务切换。

```
    /* Start the OS kernel and begin thread switching */
    osKernelStart();
```

2.2 USB

初始化按键，将相对应的GPIO (PC.13、PE.8、PC14和PC.15) 设定为input mode。

```
    /* Initial GPIO for key input */
    Initial_Key_Input();
```

在USB任务里，当End Point 2事件发生，检查目前是否有按键被按下，将数据写入USB SRAM。数据传回计算机USB控制鼠标的移动。

```
void task1_USB(void const *argument)
{
    while(1) {

        uint8_t *buf;

        if(g_u8EP2Ready) {
            buf = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP2));

            /* Update new report data */
            buf[0] = 0x00;
            buf[1] = 0x00;
            buf[2] = 0x00;
            buf[3] = 0x00;

            if(Get_Key_Input() & 0x01)
                buf[1] = 16;
            if(Get_Key_Input() & 0x02)
                buf[1] = (uint8_t)-16;
            if(Get_Key_Input() & 0x04)
                buf[2] = 16;
            if(Get_Key_Input() & 0x08)
                buf[2] = (uint8_t)-16;

            g_u8EP2Ready = 0;

            /* Set transfer length and trigger IN transfer */
            USBD_SET_PAYLOAD_LEN(EP2, 4);
        }
    }

}
```

2.3 ADC

初始化ADC，设定ADC时钟，多功能管脚，操作模式和开启电源。

```
    /* Open Volume Knob Device */
    Open_ADC_Knob();
```

在ADC任务里，延迟10ms后做一次ADC电压转换，得到的数值存到g_u32AdcValue，然后释出一个semaphore token。

```
void task2_ADC(void const *argument)
{
    while(1) {
        osDelay(10);                            /* delay 10 msec */
        g_u32AdcValue = Get_ADC_Knob();         /* Get Volume Knob data(0~4095) */
        osSemaphoreRelease(Sem_AdcDone_id);     /* Increase semaphore token */
    }
}
```

2.4 GPIO

RGB LED分别连接到GPIO (PA.12、PA.13和PA.14)。ADC转换出来的电压值g_u32AdcValue作为LED闪烁速率的输入，当转换出来的数值越大，表示LED的闪烁速率越快，延迟的时间越短，每次轮流点亮一个LED

```
void Write_LED_Color_Flash(uint32_t u32Speed)
{
    uint32_t i;
    static uint32_t u32LED_Count=0;
    volatile uint32_t *p32LED[LED_Color_Count] = {&LED_B, &LED_R, &LED_G};

    /* LED Delay Time Control */
    if(u32Speed>4096)  u32Speed = 4096;
    for(i=0; i<((4096-u32Speed)*500); i++);

    for(i=0; i<LED_Color_Count; i++) {
        if(u32LED_Count==i)
            *p32LED[i] = 0; //LED ON
        else
            *p32LED[i] = 1; //LED OFF
    }
```

```
    u32LED_Count++;
    if(u32LED_Count==LED_Color_Count)
        u32LED_Count = 0;
}

void task3_GPIO(void const *argument)
{
    while(1) {
        Write_LED_Color_Flash(g_u32AdcValue);
    }
}
```

2.5 UART

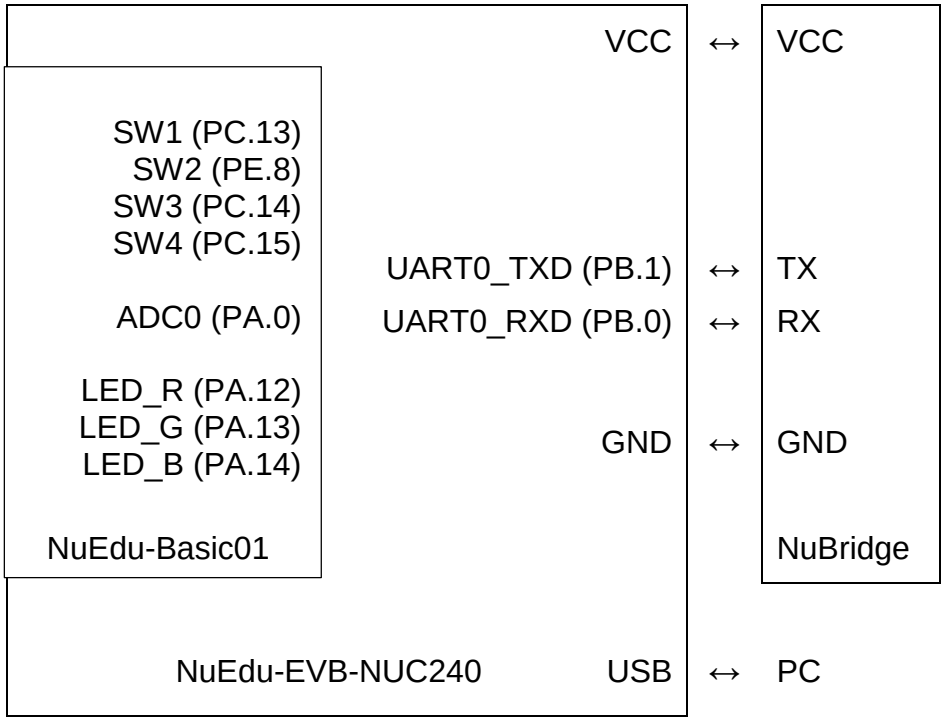
UART任务，等每一次ADC电压转换完成，释出一个token，打印转换出来的ADC电压值。

```
void task4_UART(void const *argument)
{
    while(1) {
        printf("0x%x\n\r", g_u32AdcValue);
        osSemaphoreWait(Sem_AdcDone_id, osWaitForever);     /* Wait for semaphore token */
    }
}
```

3 软件与硬件环境

- 软件环境
  - BSP 版本
    - NUC240 Series BSP CMSIS V3.01.001
  - IDE 版本
    - Keil uVersion 5.28
- 硬件环境
  - 电路组件
    - NuTiny-EVB-NUC240 V1.2
    - NuEdu-Basic01 V2.1
  - 示意图

NuEdu-EVB-NUC240开发版上接NuEdu-Basic01版子、NuBridge和USB。



4 目录信息

- EC_RTX_ADC_GPIO_USB _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 如何执行范例程序

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 NUC240_RTX_ADC_GPIO_USB.uvprojx。
2. 进入编译模式接口
   a. 编译
   b. 下载代码至内存
3. 进入除错模式接口
   a. 执行代码

6 修订纪录

Date	Revision	Description
Aug. 31, 2019	1.00	1. 初始发布.

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.