Use ADC, GPIO and USB Mouse in RTX

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This code uses ADC, GPIO and USB mouse in RTX.
BSP Version	NUC230_240 Series BSP CMSIS V3.01.001
Hardware	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 Function Description

1.1 Introduction

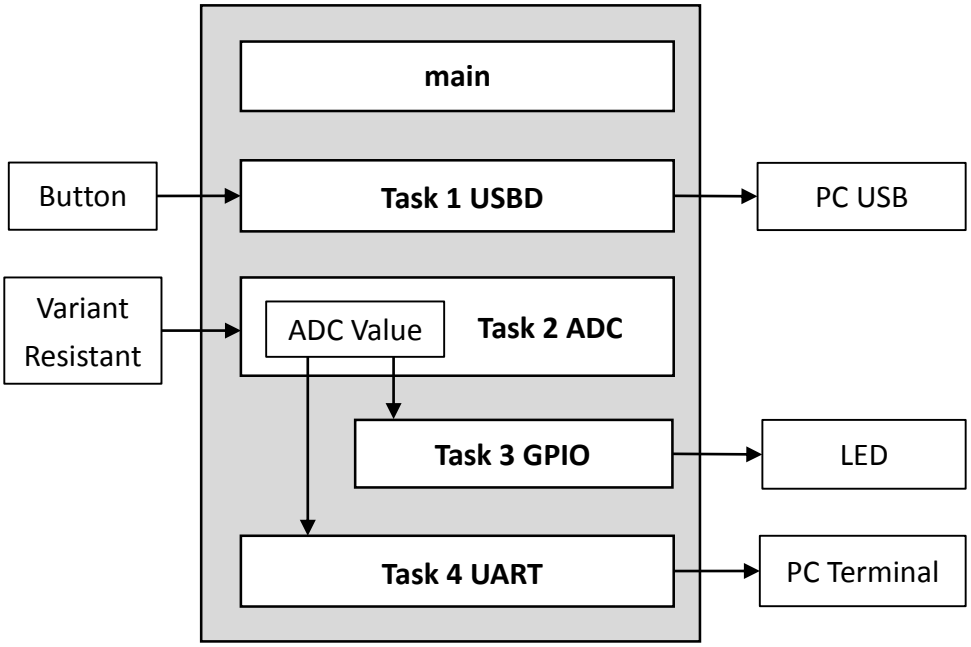
When a microcontroller application has to handle many tasks at the same time, we can use RTOS (real-time operating system) to help us manage these tasks. In RTOS system, CPU processing time is divided into a number of time slots. Each time slot processes different task and the switching between these tasks can be hundred times per second. It appears to that the application tasks are running simultaneously. When the application becomes more and more complex, we can use RTOS to help us manage these tasks.
CMSIS-RTOS in Keil MDK (Microcontroller Development Kit) provides a standard API for RTOS and application layer, includes the Keil free RTOS, RTX (Real-Time eXecutive) kernel. Once others RTOS can support CMSIS-RTOS, we can port our application to others RTOS platform easily. This sample code uses RTX kernel, which is provided by Keil CMSIS-MDK, to create a RTX kernel system, and then executes USB, ADC, UART and GPIO task.

1.2 Principle

Create the following 4 tasks and execute them simultaneously in RTX system:

- USB
  Use 4 keys (SW1~SW4) as input data to simulate mouse moving direction (up, down, left or right)
- ADC
  ADC performs voltage conversion. The input voltage can be changed by rotating variant resistance knob.
- UART
  UART prints ADC voltage conversion result after one ADC voltage conversion is finished.
- GPIO
  The ADC voltage conversion result is used as the input reference of LED blink frequency.

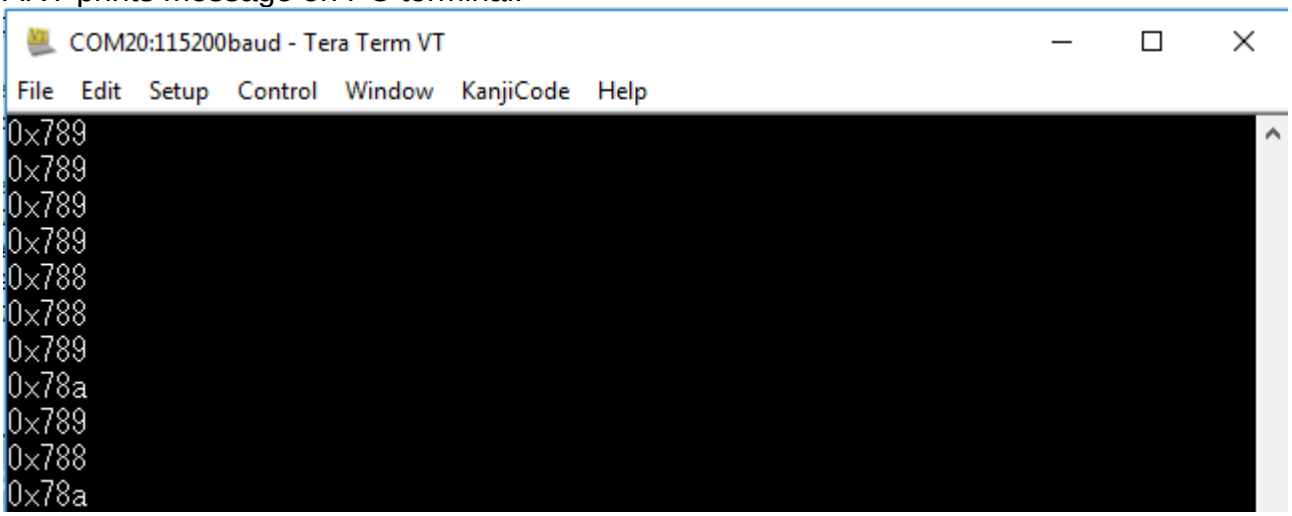
The following figure describes the input, output peripherals and internal communication between tasks.



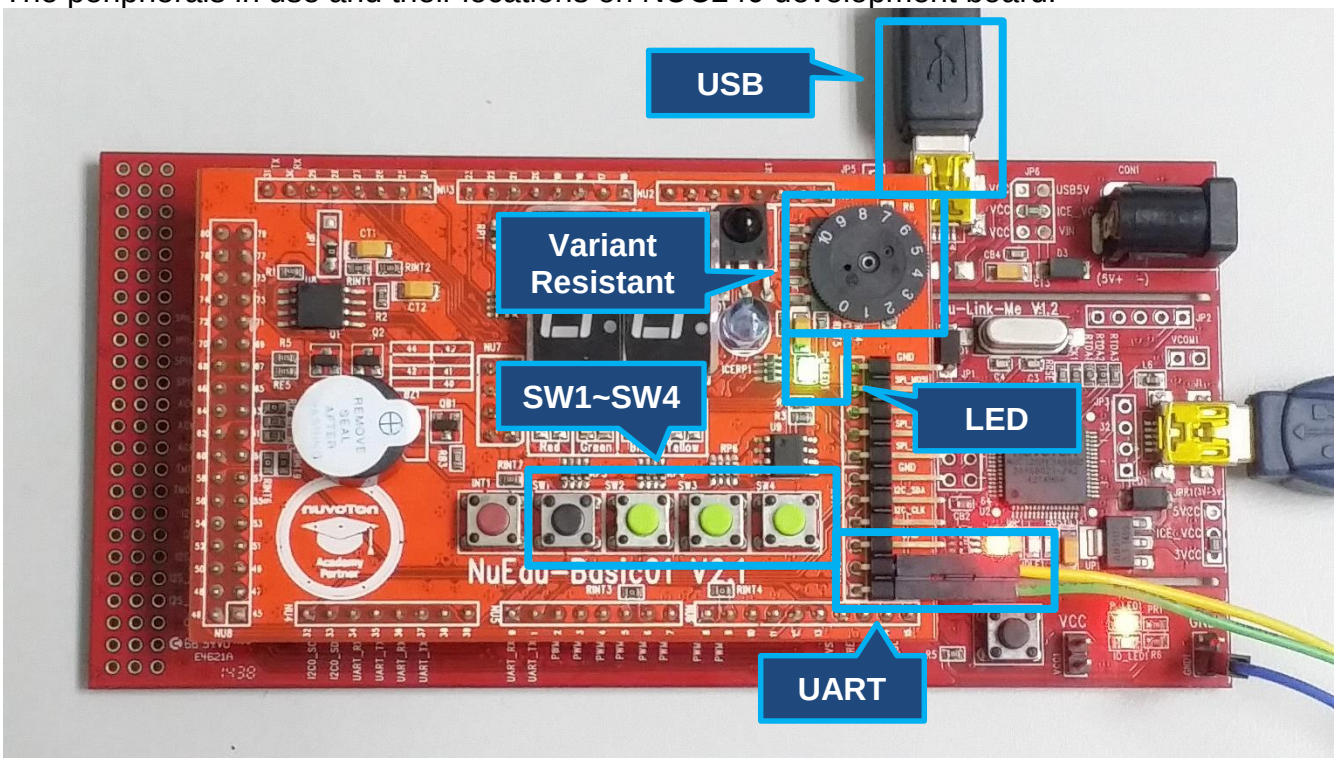
1.3 Demo Result

Connect NUC240 development board USB with computer. The keys (SW1~SW4) are used to control the USB mouse moving: up, down, left and right. ADC performs voltage conversion. The conversion result is used to control LED blink frequency. Rotating the variant resistant knob on development board can change ADC input voltage and LED blink frequency. On the PC terminal, UART prints the ADC voltage conversion result.

UART prints message on PC terminal.



The peripherals in use and their locations on NUC240 development board.



2 Code Description

2.1 RTX

Thread Declaration

This sample code has 4 tasks in parallel processing. Declare their function prototype and ID for each task. The data format of ID is 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);
```

Use osThreadDef macro to define each tasks as threads. This API also can be used to set task priority.

```
/* 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 Declaration

Semaphore is used for internal communication between tasks. Declare a semaphore whose name is Sem_AdcDone and declare its ID variable.

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

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

OS Kernel Initialization

Call osKernelInitialize function to start the OS kernel initialization.

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

```
    osKernelInitialize();
```

Semaphore Creation

Call osSemaphoreCreate function to create a new semaphore and assign the return value to the semaphore's ID. There is no token in the beginning. ADC releases one token when one voltage conversion is finished. UART waits for the token which indicates ADC conversion is done and prints the voltage conversion result.

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

Thread Creation

Use osThreadCreate function to create new thread for every task and assign the return value to their 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);
```

Start OS Kernel Execution

Call osKernelStart function to start OS Kernel execution and begin thread switching.

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

2.2 USB

Initialize GPIO for key input. Set corresponding GPIO (PC.13, PE.8, PC14 and PC.15) as input mode.

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

In USB task, when end point 2 event occurs, check if any key is pressed, write data in USB SRAM and wait for data transferring to USB to control USB mouse moving.

```
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

Initialize ADC, configure ADC clock, multiple function pin, operation mode and power-on ADC.

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

In ADC task, ADC converts voltage after 10ms delay. The conversion result is stored into g_u32AdcValur and then release one 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 is connected to GPIO (PA.12, PA.13 and PA.14). ADC voltage conversion result g_u32AdcValue is a reference input for LED blink frequency. The larger the converted value, the faster the LED blink frequency and the shorter the delay time. One LED is lighted on every time.

```
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

In UART task, UART waits for the token which was released after ADC voltage conversion is finished every time and prints the ADC voltage conversion result.

```
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 Software and Hardware Environment

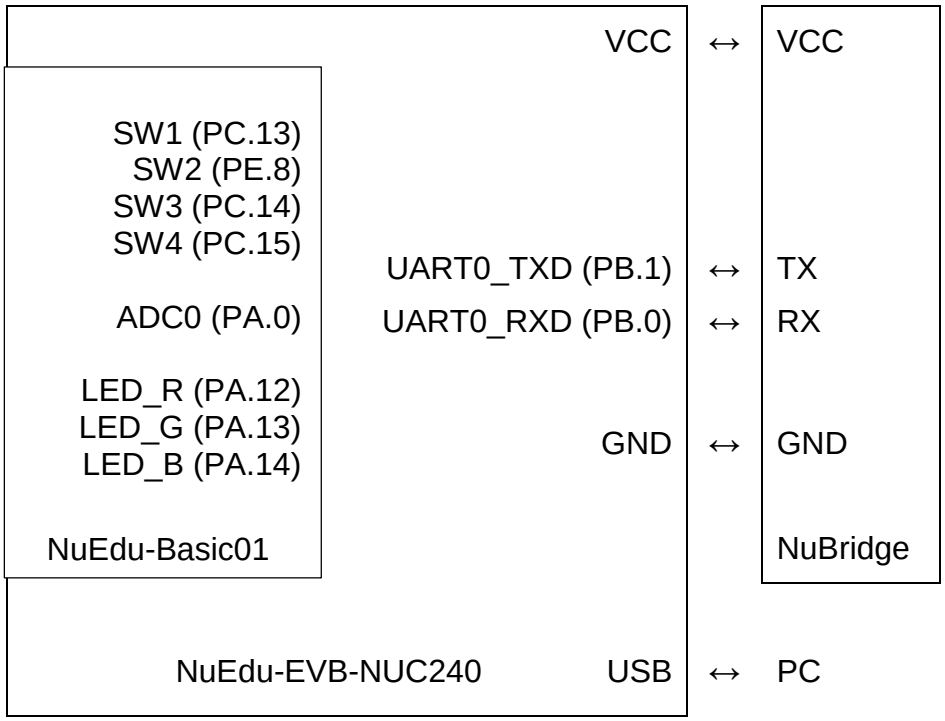
Software Environment

- BSP version
  - NUC230_240 Series BSP CMSIS V3.01.001
- IDE version
  - Keil uVersion 5.28

Hardware Environment

- Circuit components
  - NuEdu-EVB-NUC240 V2.1
  - NuEdu-Basic01 V2.1
- Diagram

Connect NuEdu-EVB-NUC240 development board with NuEdu-Basic01 board, NuBridge and USB.



4 Directory Information

- EC_NUC240_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 How to Execute Example Code

1. Browsing into sample code folder by Directory Information (section 4) and double click NUC240_RTX_ADC_GPIO_USB.uvprojx.
2. Enter Keil compile mode
   a. Build
   b. Download
   c. Start/Stop debug session
3. Enter debug mode
   a. Run

6 Revision History

Date	Revision	Description
Aug. 31, 2019	1.00	1. Initially issued.

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.