RTX 即时操作系统

NuMicro® 32 位系列微控制器范例代码介绍

文件信息

代码简述	介绍 RTX (Real-Time eXecutive)及时操作系统的多重时序运作原理
BSP 版本	NUC230/240 Series BSP v3.01.002
开发平台	NuTiny–EVB–NUC240–LQFP100 V1.0

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

Keil RTX (Real-Time eXecutive)是一个专为ARM及Cortex M系列处理器开发的即时操作系统。它提供了一个可以让使用者建立多重任务同步执行的程序软件的环境，同时也能帮助使程序代码更加结构化和便于维护。此范例程式则演示了在RTX环境下使用者如何建立任务(task)并执行及时操作系统的各项功能。

1.2 原理

在此专案中包含了多个范例程式，主要有以下几个分类：

- Task 的建立、启动、执行与终止，演示多重 Task 的执行。
- Event 的执行，演示多重 Task 以等待 event 的机制交替执行。
- Semaphore 的建立与执行，演示多重 Task 以等待 Semaphore 的机制交替执行。
- Mutex 的建立以及执行，演示多重 Task 以等待 Mutex 释放的机制交替执行。
- Mailbox 的建立以及执行，演示两个 Task 透过 Mailbox 传送与接收讯息。
- Timer 的建立，演示建立 Timer 并确认 Timer 是否成功建立。

1.3 执行结果

执行RTX_Blinky专案，在UART视窗显示执行结果。如图1-1所示：

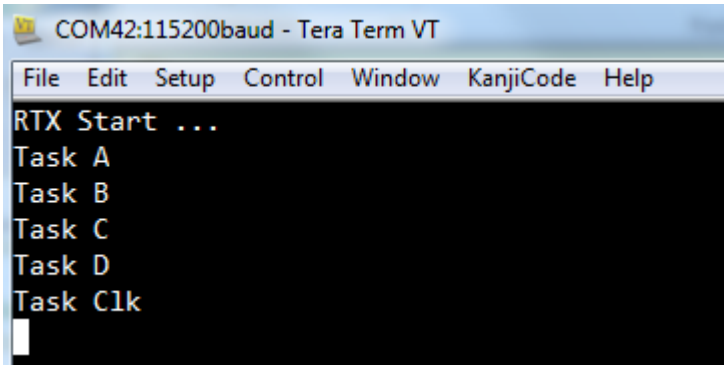


图 1-1 RTX_Blinky 范例程式执行结果

承上，执行RTX_Blinky专案，NUC240版上相对应的LED灯会有闪烁的现象。如图1-2中黄色框中所示：

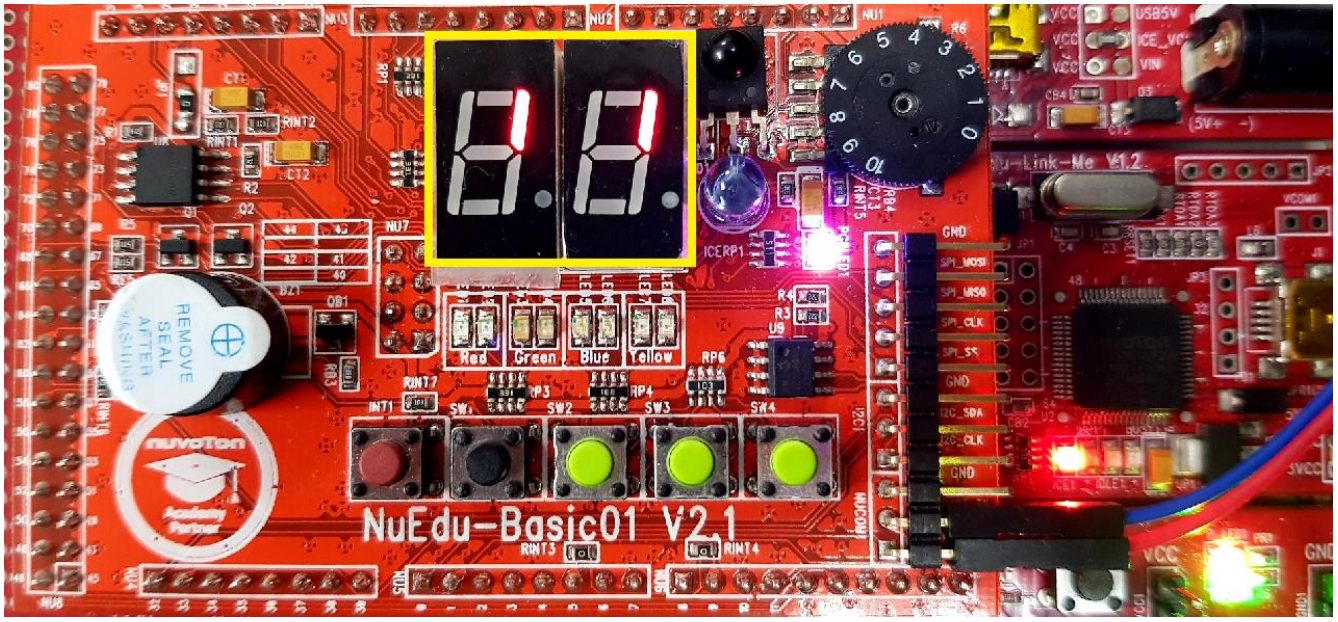


图 1-2 RTX_Blinky范例程式执行时，NUC240版LED之闪烁

执行RTX_Delay专案，在UART视窗显示执行结果。如图1-3所示：

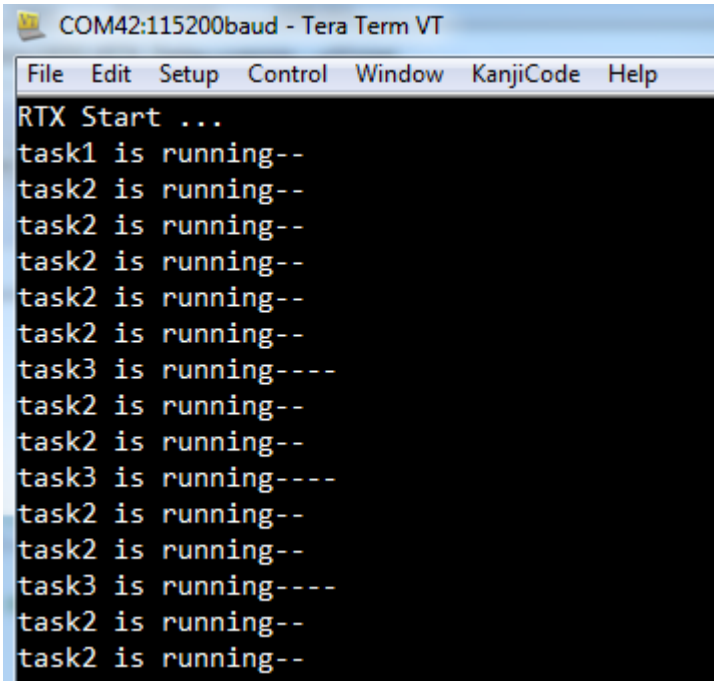


图 1-3 RTX_Delay范例程式执行结果

執行RTX_Event_Interrupt專案，在UART視窗显示执行结果。如图1-4所示：

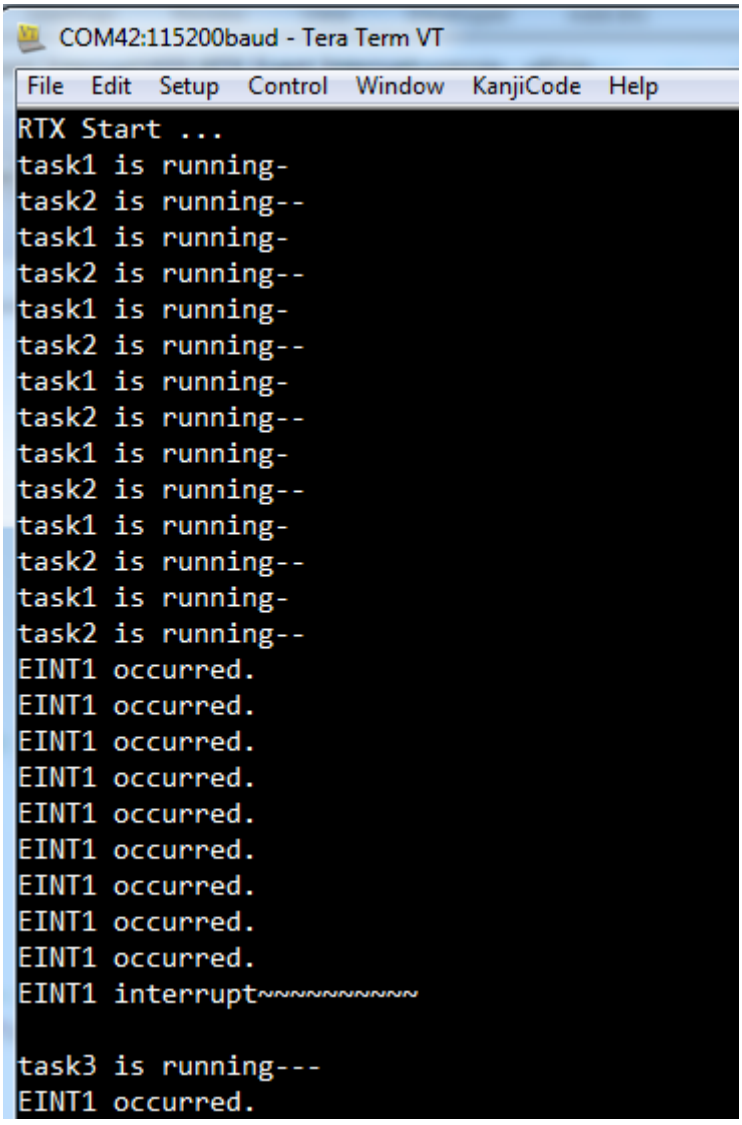


图 1-4 RTX_Event_Interrupt范例程式执行结果

執行RTX_Event專案，在UART視窗显示执行结果。如图1-5所示：

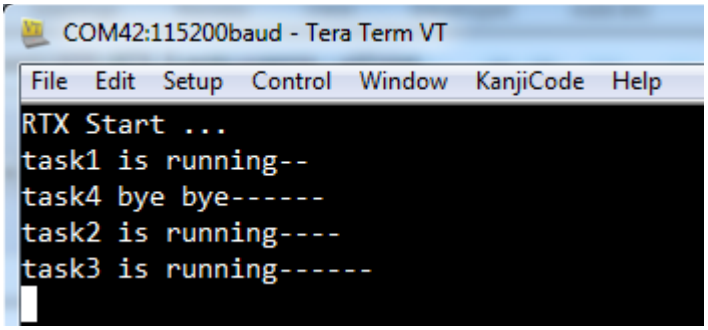


图 1-5 RTX_Event 范例程式执行结果

執行RTX_Idle專案，在UART視窗显示执行结果。如图1-6所示：

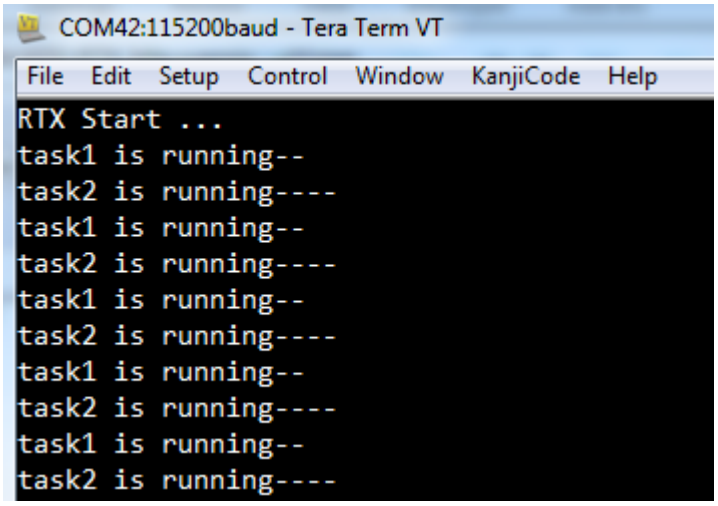


图 1-6 RTX_Idle 范例程式执行结果

執行RTX_Mailbox專案，在UART視窗显示执行结果。如图1-7所示：

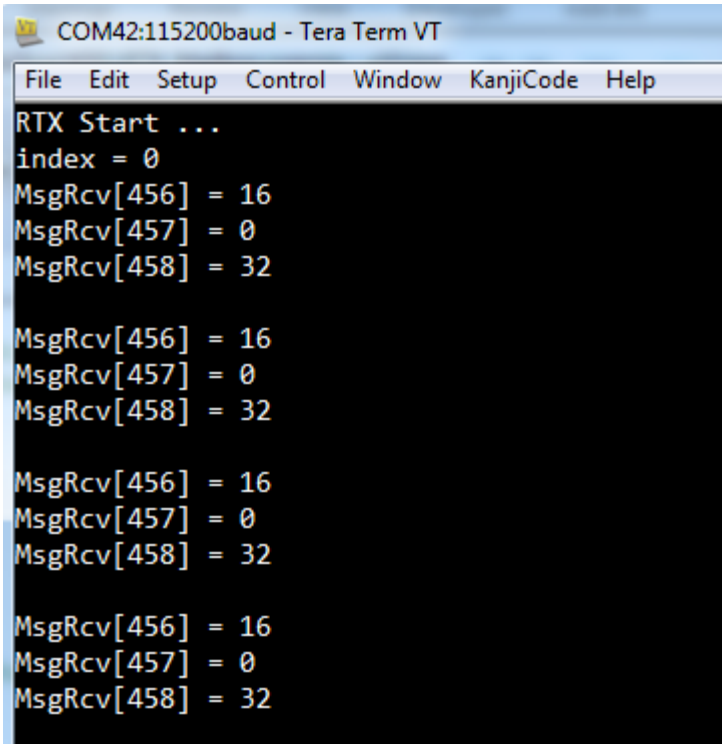


图 1-7 RTX_Mailbox 范例程式执行结果

執行RTX_Multiple_Instance專案，在UART視窗显示执行结果。如图1-8所示：

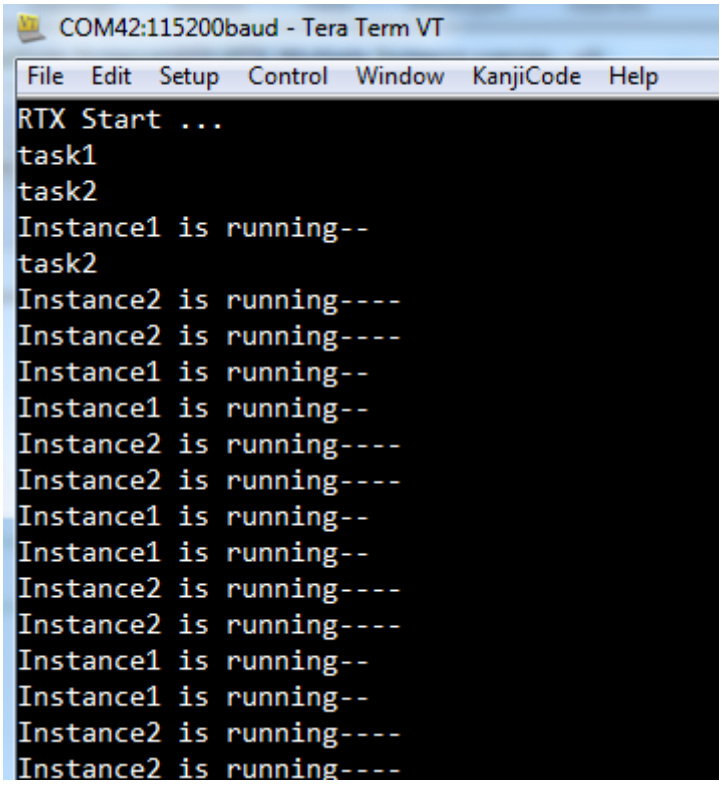


图 1-8 RTX_Multiple_Instance 范例程式执行结果

執行RTX_Mutex專案，在UART視窗显示执行结果。如图1-9所示：

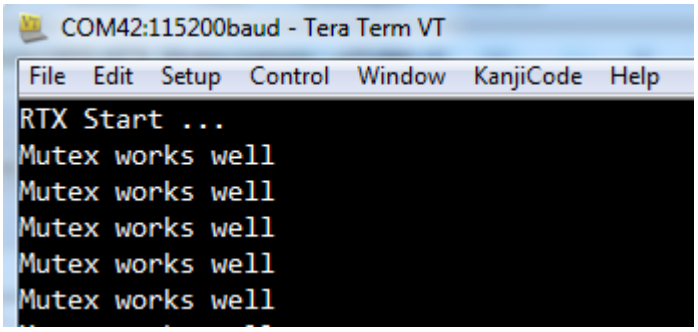


图 1-9 RTX_Mutex 范例程式执行结果

執行RTX_Priority_Inv專案，在UART視窗显示执行结果。如图1-10所示：

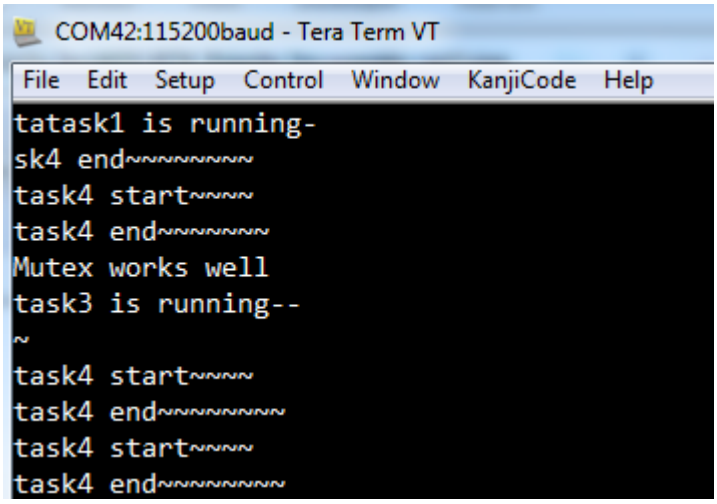


图 1-10 RTX_Priority_Inv范例程式执行结果

執行RTX_Semaphore_Barrier專案，在UART視窗显示执行结果。如图1-11所示：

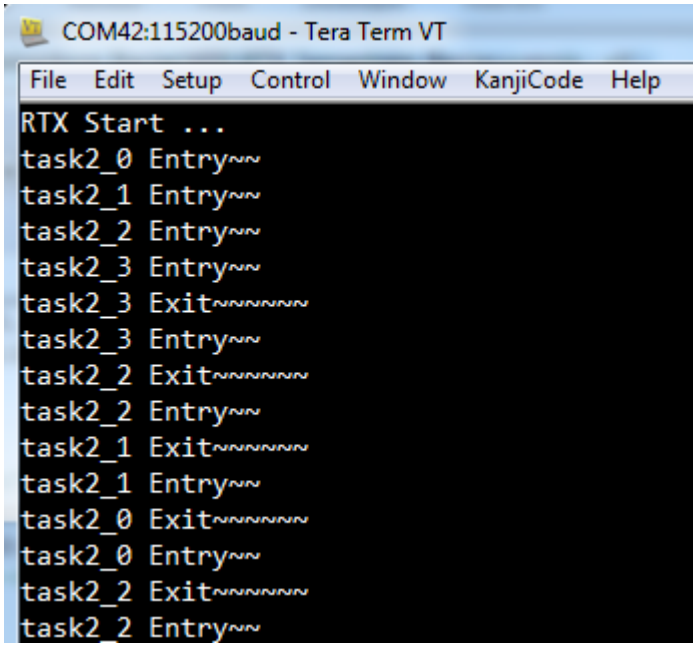


图 1-11 RTX_Semaphore_Barrier范例程式执行结果

執行RTX_Semaphore_Multiplex專案，在UART視窗显示执行结果。如图1-12所示：

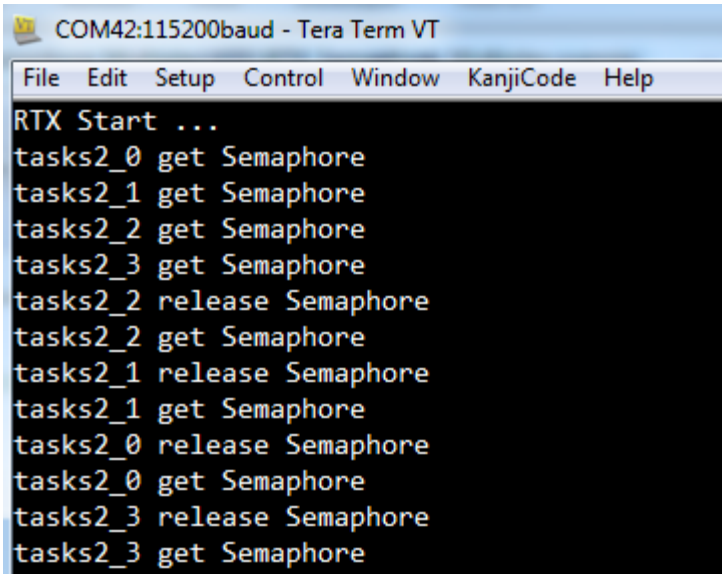


图 1-12 RTX_Semaphore_Multiplex范例程式执行结果

執行RTX_Semaphore_Rendezvous專案，在UART視窗显示执行结果。如图1-13所示：

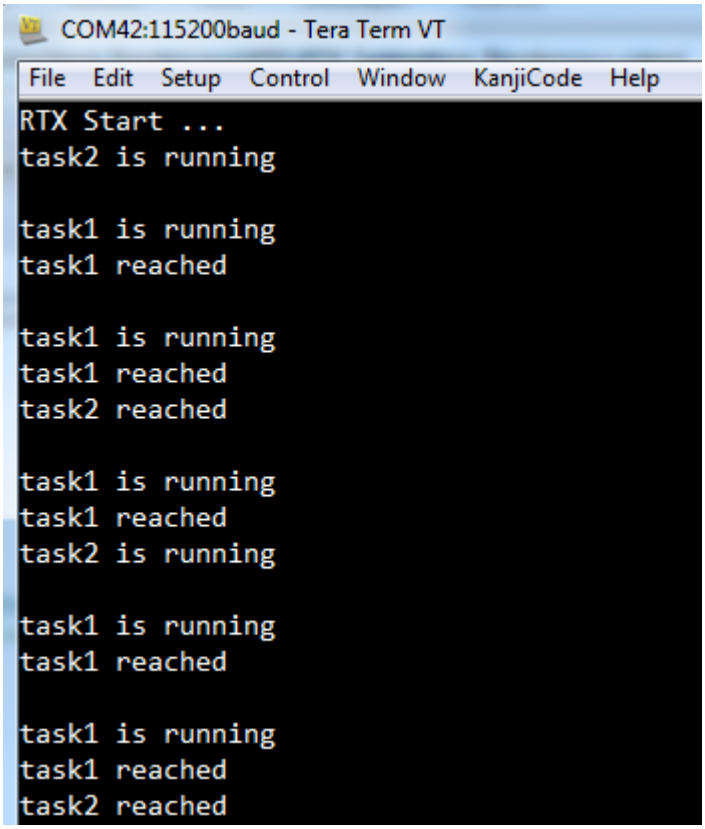


图 1-13 RTX_Semaphore_Rendezvous 范例程式执行结果

執行RTX_Semaphore_Synchronization專案，在UART視窗显示执行结果。如图1-14所示：

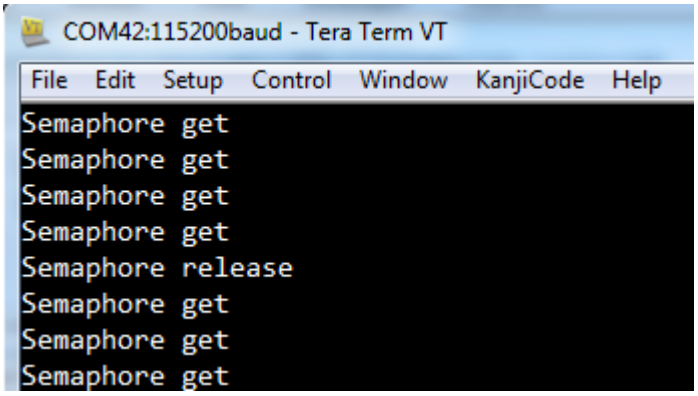


图 1-14 RTX_Semaphore_Synchronization 范例程式执行结果

執行RTX_Timer專案，在UART視窗显示执行结果。如图1-15所示：

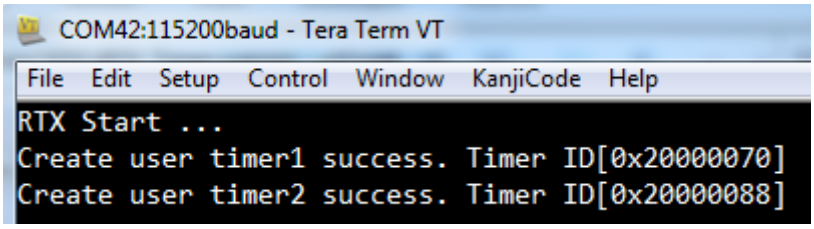


图 1-15 RTX_Timer 范例程式执行结果

2 代码介绍

以下为 Task 建立、启动的流程：

```
osThreadId t_taskA;                         /* assigned task id of task: TaskA */
osThreadId t_taskB;                         /* assigned task id of task: TaskB */
osThreadId t_taskC;                         /* assigned task id of task: TaskC */
osThreadId t_taskD;                         /* assigned task id of task: TaskD */
osThreadId t_taskClk;                       /* assigned task id of task: TaskClk */

void TaskA(void const *argument);
void TaskB(void const *argument);
void TaskC(void const *argument);
void TaskD(void const *argument);
void TaskClk(void const *argument);

/* define task as thread function */
osThreadDef(TaskA, osPriorityNormal, 1, 0);
osThreadDef(TaskB, osPriorityNormal, 1, 0);
osThreadDef(TaskC, osPriorityNormal, 1, 0);
osThreadDef(TaskD, osPriorityNormal, 1, 0);
osThreadDef(TaskClk, osPriorityNormal, 1, 0);

int main(void)
{

    /* Create task and assign thread ID to t_task */
    t_taskA = osThreadCreate(osThread(TaskA), NULL);
    t_taskB = osThreadCreate(osThread(TaskB), NULL);
    t_taskC = osThreadCreate(osThread(TaskC), NULL);
    t_taskD = osThreadCreate(osThread(TaskD), NULL);
    t_taskClk  = osThreadCreate (osThread(TaskClk), NULL);
    osSignalSet (t_taskA, 0x0001);

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

/*----------------------------------------------------------------------------
 *        Task 1 'TaskA': Phase A output
 *---------------------------------------------------------------------------*/
```

```
void TaskA (void const *argument) {
    printf("Task A\n");
    for (;;) {
        osSignalWait (0x0001, 0xffff);    /* wait for an event flag 0x0001    */
        LED_On (LED_A);
        signal_func (t_taskB);            /* call common signal function      */
        LED_Off(LED_A);
    }
}

/*----------------------------------------------------------------------------
 *        Task 2 'TaskB': Phase B output
 *---------------------------------------------------------------------------*/
void TaskB (void const *argument) {
    printf("Task B\n");
    for (;;) {
        osSignalWait (0x0001, 0xffff);    /* wait for an event flag 0x0001    */
        LED_On (LED_B);
        signal_func (t_taskC);            /* call common signal function      */
        LED_Off(LED_B);
    }
}

/*----------------------------------------------------------------------------
 *        Task 3 'TaskC': Phase C output
 *---------------------------------------------------------------------------*/
void TaskC (void const *argument) {
    printf("Task C\n");
    for (;;) {
        osSignalWait (0x0001, 0xffff);    /* wait for an event flag 0x0001    */
        LED_On (LED_C);
        signal_func (t_taskD);            /* call common signal function      */
        LED_Off(LED_C);
    }
}

/*----------------------------------------------------------------------------
 *        Task 4 'TaskD': Phase D output
 *---------------------------------------------------------------------------*/
void TaskD (void const *argument) {
    printf("Task D\n");
```

```
    for (;;) {
        osSignalWait (0x0001, 0xffff);    /* wait for an event flag 0x0001    */
        LED_On (LED_D);
        signal_func (t_taskA);            /* call common signal function      */
        LED_Off(LED_D);
    }
}

/*----------------------------------------------------------------------------
 *        Task 5 'TaskClk': Signal TaskClk
 *---------------------------------------------------------------------------*/
void TaskClk (void const *argument) {
    printf("Task Clk \n");
    for (;;) {
        osSignalWait (0x0100, 0xffff);    /* wait for an event flag 0x0100    */
        LED_On (LED_CLK);
        osDelay (8);                      /* delay 8 TaskClk ticks            */
        LED_Off(LED_CLK);
    }
}
```

以下为 Event 执行的流程：

```
void task1 (void const *argument)
{
    while(1)
    {
        osSignalWait(0x0005,0xffff);   /* Wait for an event to be set, no time out */

        printf("task1 is running--\n");
        osDelay(50);

        osSignalSet(t_task2, 0x0001);   /* Set an event for task 2. Each task has 16 event
flags */
        osSignalSet(t_task3, 0x0001);   /* Set an event in task 3 */
    }
}

void task2 (void const *argument)
{
    while(1)
```

```
    {
        osSignalWait(0x0001,0xffff);            /* Wait for an event */

        printf("task2 is running----\n");
        osDelay(50);

        osSignalSet(t_task3, 0x0002);           /* Set an event in Task 3 */
    }
}

void task3 (void const *argument)
{
    while(1)
    {
        osSignalWait(0x0003,0xffff);            /* Wait for two event flags to be set */

        printf("task3 is running------\n");
        osDelay(10);

        osSignalSet(t_task1, 0x0001);           /* Trigger task 1 to start over */
    }
}
```

以下为 Semaphore 建立以及执行的流程：

```
osSemaphoreDef_t *SemMultiplex;
osSemaphoreId SemMultiplexId;

uint32_t g_au32Param[6] = {0, 1, 2, 3, 4, 5};

void task1 (void const *argument)
{
    /* Create semaphore with semaphore num = 6 */
    SemMultiplexId = osSemaphoreCreate(SemMultiplex, 6);

    /* Create 6 sub-tasks of task 2 */
    t_task2_0 = osThreadCreate(osThread(task2), &g_au32Param[0]);
    t_task2_1 = osThreadCreate(osThread(task2), &g_au32Param[1]);
    t_task2_2 = osThreadCreate(osThread(task2), &g_au32Param[2]);
    t_task2_3 = osThreadCreate(osThread(task2), &g_au32Param[3]);
    t_task2_4 = osThreadCreate(osThread(task2), &g_au32Param[4]);
    t_task2_5 = osThreadCreate(osThread(task2), &g_au32Param[5]);
```

```
    /* Delete task1 itself */
    osThreadTerminate(t_task1);

}

void task2 (uint32_t *argument)
{
    while(1)
    {
        /* Wait here until you aquire a semaphore    */
        osSemaphoreWait(SemMultiplexId, 0xffff);
        printf("tasks2_%d get Semaphore\n", *argument);
        osDelay(100);
        osSemaphoreRelease(SemMultiplexId);
        printf("tasks2_%d release Semaphore\n", *argument);
    }
}
```

以下为 Mutex 建立以及执行的流程：

```
osMutexDef_t mutex;
osMutexId mutexId;
uint32_t g_u32CriticalRes1, g_u32CriticalRes2;

void task1 (void const *argument)
{
    /* Initilise the Mutex */
    mutexId = osMutexCreate(&mutex);

    g_u32CriticalRes1 = 0;
    g_u32CriticalRes2 = 0;

    while(1)
    {
        /* Wait here until the mutex is aquired */
        osMutexWait(mutexId, 0xffff);

        g_u32CriticalRes1++;
        /* Before Mutex release, the procees can't be interrupted */
```

```
        /* The value of g_u32CriticalRes1 and g_u32CriticalRes2 should be the same*/
        osDelay(50);
        g_u32CriticalRes2++;

        osMutexRelease(mutexId);
    }
}

void task2 (void const *argument)
{
    while(1)
    {
        /* Wait here until the mutex is aquired */
        osMutexWait(mutexId, 0xffff);

        if(g_u32CriticalRes1 != g_u32CriticalRes2)
        {
            printf("Mutex error\n");
            while(1);
        }
        else
        {
            printf("Mutex works well\n");
        }

        g_u32CriticalRes1++;
        g_u32CriticalRes2++;

        osMutexRelease(mutexId);
    }
}
```

以下为 Mailbox 建立以及执行的流程：

```
osPoolId poo1Id;
osMailQId mailId;

msg_t message[2];
uint8_t g_u8Idx;

osMailQDef_t mailbox[16];
```

```
osPoolDef_t  mpool[16];

void task1 (void const *argument)
{
    msg_t *mptr;                                /* Pointer to the mailbox send buffer */
    uint32_t i;


    mailId = osMailCreate (mailbox, t_task1);
    poo1Id = osPoolCreate(mpool);

    for(g_u8Idx = 0; g_u8Idx < 2; g_u8Idx++)
    {
        message[g_u8Idx].msg_id = g_u8Idx;
        for(i = 0; i < 3; i++)
        {
            message[g_u8Idx].msg[i] = g_u8Idx*3 + i;
        }
    }

    g_u8Idx = 0;
    while(1)
    {
        g_u8Idx %= 2;

        osPoolAlloc(poo1Id);                    /* Set the pointer to a mailbox buffer */
        mptr = (msg_t *)mpool;
        *mptr = message[g_u8Idx];               /* Write data to the mailbox buffer    */
        osMailPut(mailId, mptr);                /* Send a message through mailbox      */

        printf("index = %d\n", g_u8Idx++);
        osDelay(50);
    }
}

void task2 (void const *argument)
{
    msg_t MsgRcv,*rptr;                         /* Pointer to the mailbox receive buffer */
    uint32_t i;

    while(1)
```

```
    {
        osMailGet(mailId, 0xffff);           /* Wait for a new message in mailbox */
        MsgRcv = *rptr;
        osMailFree(mailId, rptr);            /* Release the mailbox receive buffer */

        for(i = 0; i < 3; i++)
        {
            printf("MsgRcv[%d] = %d\n", MsgRcv.msg_id*3 + i, MsgRcv.msg[i]);
        }
        printf("\n");
    }
}
```

以下为 Timer 的建立的流程：

```
osTimerId Timer1_Id;
osTimerId Timer2_Id;

void Timer1_Callback (void const *arg)
{

}

void Timer2_Callback (void const *arg)
{

}
osTimerDef (Timer1, Timer1_Callback);
osTimerDef (Timer2, Timer1_Callback);

void task1 (void const *argument)
{
    /* Create Timer 1*/
    Timer1_Id = osTimerCreate (osTimer(Timer1), osTimerPeriodic, NULL);
    /* Check if Timer 1 create success*/
    if (Timer1_Id == NULL)
    {
        printf ("Failed to create user timer1.\n");
    }
    else
    {
        printf ("Create user timer1 success. Timer ID[0x%x]\n", (uint32_t)Timer1_Id);
    }
```

```
    /* Create Timer 1*/
    Timer2_Id = osTimerCreate (osTimer(Timer2), osTimerPeriodic, NULL);
    /* Check if Timer 1 create success*/
    if (Timer2_Id  == NULL)
    {
        printf ("Failed to create user timer2.\n");
    }
    else
    {
        printf ("Create user timer2 success. Timer ID[0x%x]\n", (uint32_t)Timer2_Id);
    }

    while(1)
    {
        osDelay(20);
    }

}
```

3 软件与硬件环境

- 软件环境
  - BSP 版本
    - NUC230/240 Series BSP CMSIS v3.01.002
  - IDE 版本
    - Keil uVersion 5.28
- 硬件环境
  - 电路组件
    - NuTiny–EVB–NUC240–LQFP100 V1.0
  - 示意图
    - 将 UART TX（PB.1）pin 脚连接到 PC UART RX，以显示示例代码的执行结果于 PC 上

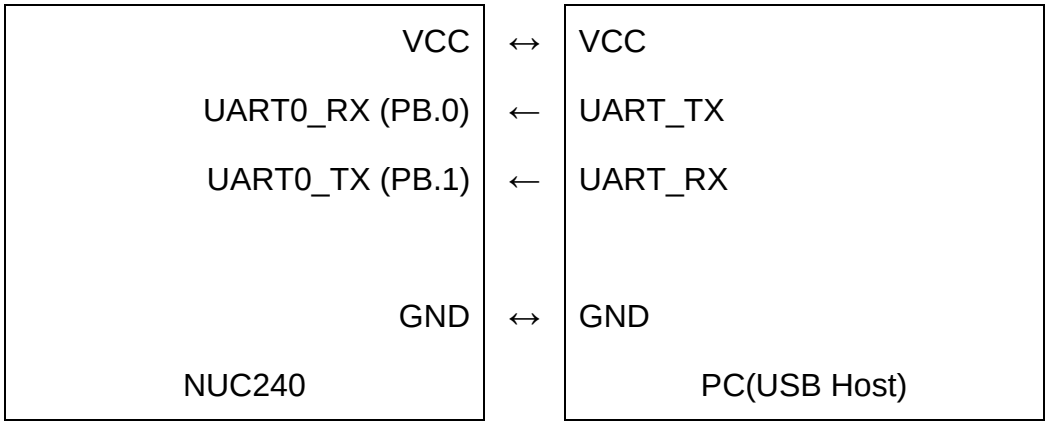


图 3-1 NUC240 和 PC连接图

4 目录信息

- EC_NUC240_RTX_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
    - 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 RTX_xxx.uvproj。
2. 进入编译模式接口
   a. 编译
   b. 下载代码至内存
   c. 进入 / 离开除错模式
3. 进入除错模式接口
   a. 执行代码

6 修订纪录

Date	Revision	Description
Sep. 18, 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.