TTC BLE SDK
RTOS Introduction

Version: V1.0

Shenzhen Shengrun Technology Co.,Ltd

5th Dec,2016

Version	Revised Date	Revisionist	Reviewer	Modified content
1.0	2016-12-05	郭高亮	张眼	First release

Content

1.TI RTOS Overview.....1
1.1 Semaphore.....1
1.2 Task and Event Handing (Task & Event).....2
1.3 Timing Task.....4
1.4 Messege.....7
1.5 Power down save.....10
1.6 Dynamic memory management.....10
2.Contact us.....12

1. TI RTOS Overview

TI-RTOS is a preemptive, multi-threaded real-time operating system. Can perform hardware interrupt, software interrupt, task, sleep, and priority decrease. The following are the six concepts of semaphore, task, timing task, message, power-down storage and dynamic memory management, and a brief description of the TTC_CC2640_SDK project.

1.1 Semaphore

In TI-RTOS, semaphores are used to synchronize the operation of two tasks, such as coordinating the access of the application to the shared resource by the two tasks of the BLE protocol stack. The task can be awakened by placing a semaphore. To TTC_CC2640_SDK project serial demo, for example, explain the use of semaphores.

(1) Define thread semaphore (no need to define again):

```
<TTCBlePeripheralTask.c>
static ICall_Semaphore sem;          //Thread semaphore, used on waking up
thread
```

(2) Pass the semaphore address & semaphore to the serial port initialization function:

```
<TTCBlePeripheralTask.c>
#ifdef TTCDRIVER_UART
   TTCDriverDemoUARTInit(&TTCBlePeripheralTaskCls,&sem,&appMsgQueue);
#endif //TTCDRIVER_UART
```

(3) After the serial port initialization is successful, the defined semaphore pointer UartSem points to the semaphore defined in TTCBlePeripheralTask.c.

```
<TTCDriverUARTDemo.c>
static ICall_Semaphore * UartSem;
-------------------------------------------------------------------------
void TTCDriverDemoUARTInit(TTCSdkClass_t *appCallbacks,
                           ICall_Semaphore * sem,
                            Queue_Handle * appMsgQueue){
        //…
        UartSem = sem;
        //…
    }
```

(4) Semaphore_post()function Set the Semaphore

```
<TTCDriverUARTDemo.c>
static void TTCSDKDriverUARTSetEvent(UArg arg){
```

```
    events |= arg;
    Semaphore_post(*UartSem); //set the semaphore, wake up thread
  }
```

(5) ICall_wait()Wait the Semaphore to wake up thread

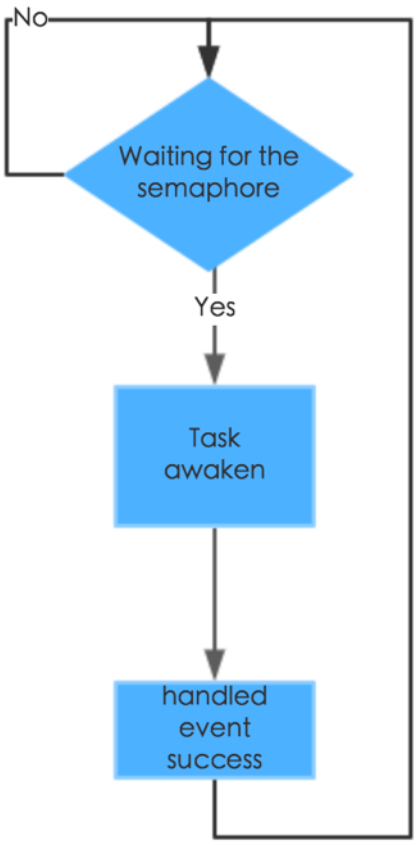
```
<TTCBlePeripheralTask.c>
static void TTCBlePeripheralTaskFxn(UArg a0, UArg a1){
    TTCBlePeripheralTaskInit();
    for (;;){
        ICall_Errno errno = ICall_wait(ICALL_TIMEOUT_FOREVER);
        //…
    }
```

1.2 Task and Event Handing (Task & Event)

The system has and only one task at all times task is executing or it may be an empty task. After each task is executed, the processor goes to the task with the highest priority in the ready state. When the priorities are equal, they are executed according to the principle of "preemptive first execution".

In the slave project, the BLE protocol stack has the highest priority and the lowest priority of the application task. Most of the task is in a suspended state. Once the semaphore is set, the task will be awakened and the corresponding event will be handled, as shown in the following figure.

All of the events in the application that need to be processed can be handled in an existing task without creating a new task. Take the serial demo of TTC_CC2640_SDK as an example to explain the use of event handling in application tasks.



(1) Define event variables:

```
<TTCDriverUARTDemo.c>
static u16 events;                                  // Local event
#define TTCBLE_SDK_UART_EVN    0x0001               //Serial event
```

(2) Timing task callback set the event TTCBLE_SDK_UART_EVN, pay attention to the event macro definition for the bit field mode, and 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, and so on, not repeatable.

```
<TTCDriverUARTDemo.c>
Util_constructClock(&uClock,
                    TTCDriverDemoUARTClockHandler,    //callback function
                    1000, 0, true,
                    TTCBLE_SDK_UART_EVN);           // callback function,
serial event
--------------------------------------------------------------------------
static void TTCDriverDemoUARTClockHandler(UArg arg){
    TTCSDKDriverUARTSetEvent(arg);
}
--------------------------------------------------------------------------
static void TTCSDKDriverUARTSetEvent(UArg arg){     //arg 为 TTCBLE_SDK_UART_EVN
    events |= arg;                                   // Set up serial event
```

```
    Semaphore_post(*UartSem);                    //wake up thread
}
```

(3) There is a set of events, but also need to have the appropriate event handle function TTCSDKDriverUARTEvent()

```
<TTCDriverUARTDemo.c>
void TTCSDKDriverUARTEvent(void){
    if(events & TTCBLE_SDK_UART_EVN){            // Query whether serial
event has occurred
        events &= ~TTCBLE_SDK_UART_EVN;          //Clear the serial event
        TTCDriverUartWrite(&uartHandle,"TTCDriverUART Test\r\n",
                           strlen("TTCDriverUART Test\r\n"));
        Util_startClock(&uClock);
    }
}
```

(4) When the event is set, the semaphore is also set. After the thread wakes up, calls TTCSDKDriverUARTEvent ()

```
<TTCBlePeripheralTask.c>
static void TTCBlePeripheralTaskFxn(UArg a0, UArg a1){
    TTCBlePeripheralTaskInit();
    for (;;){
        ICall_Errno errno = ICall_wait(ICALL_TIMEOUT_FOREVER);

         #ifdef TTCDRIVER_UART
           TTCDriverUartEvent();
           TTCSDKDriverUARTEvent();              //handle the serial event
         #endif //TTCDRIVER_UART
        //…
     }
}
```

1.3 Timing Task

Timing tasks can start the corresponding event to process. When the timer arrives, the software interrupt is triggered and the timer task callback function is entered. For example, the TTCSdkSetEvent () function can be called in the callback function to set the corresponding event and semaphore. To TTC_CC2640_SDK project using serial demo, for example, explain the use of timing tasks.

(1) Create a timed task that defines a timed clock variable uClock:

```
<TTCDriverUARTDemo.c>
```

```
Clock_Struct uClock;
```

(2) Initialize the timed task in the serial initialization function TTCDriverDemoUARTInit ()

```
<TTCDriverUARTDemo.c>
Util_constructClock(&uClock,                        // software timers
                    TTCDriverDemoUARTClockHandler, // Callback function name
                    1000,                           // timing time(ms)
                    0,                              // Single cycle timing
task
                    true,                           // Immediately start up
after initialization
                    TTCBLE_SDK_UART_EVN);           // Callback function band
parameter
```

Note: This timer task immediately after initialization, to call the callback function after 1000ms when the parameters, such as TTCDriverDemoUARTClockHandler (TTCBLE_SDK_UART_EVN); see the following entry parameter description.

< util.c> Util_constructClock Prototype, corresponding to entry parameter description:

```
Clock_Handle Util_constructClock(Clock_Struct *pClock,
                                 Clock_FuncPtr clockCB,
                                 uint32_t clockDuration,
                                 uint32_t clockPeriod,
                                 uint8_t startFlag,
                                 UArg arg)
```

- pClock: the clock variable address
- clockCB: When timing time arrives, enter this timing task callback function
- clockDuration: Timing time (ms) for the first start of a scheduled task
- clockPeriod: Repeat cycle (ms), 0 is not repeated, non-0 is repeated.
  (1) When the value is 0, the timer task is single cycle. After the timer task is turned on, the clockDuration setting is reached and the callback function is called and stopped.
  (2) When the value is not zero, the timer task is the repetition period. After the timer task is turned on, the first timing time is clockDuration, the subsequent timing time is clockPeriod, and the timing task is automatically executed periodically.
- startFlag: Open the logo immediately
  (1) true: no need to call open function, immediately open;

(2) false: will not open itself, if you want to start the timing task, you need to call the open function;

- arg: Callback function parameter

(3) After the timing task starts to reach timing time, enter the callback function and set the TTCBLE_SDK_UART_EVN event (see section 1.2 for event handling).

```
<TTCBlePeripheralTask.c>
static void TTCDriverDemoUARTClockHandler(UArg arg){  //串口事件
    TTCSDKDriverUARTSetEvent(arg);
}
```

(4) Restart timing task, method one:

```
<TTCDriverUARTDemo.c>
void TTCSDKDriverUARTEvent(void){
    if(events & TTCBLE_SDK_UART_EVN){
        events &= ~TTCBLE_SDK_UART_EVN;
        TTCDriverUartWrite(&uartHandle,"TTCDriverUART Test\r\n",
                           strlen("TTCDriverUART Test\r\n"));
        //restart timing task,timing time:1000ms
        Util_startClock(&uClock);
    }
}
```

(5) Restart timing task, methods two (timing time can be modified):

```
<TTCDriverUARTDemo.c>
void TTCSDKDriverUARTEvent(void){
    if(events & TTCBLE_SDK_UART_EVN){
        events &= ~TTCBLE_SDK_UART_EVN;
        TTCDriverUartWrite(&uartHandle,"TTCDriverUART Test\r\n",
                           strlen("TTCDriverUART Test\r\n"));
        // restart timing task,timing time modified to 2000ms
        Util_restartClock(&uClock, 2000);
    }
}
```

(6) Stop timing task, the timer task can be stopped at any time

```
Util_stopClock(&uClock);
```

1.4 Messege

When the events need to deal with the strict timing requirements, you can use the timing task processing; and when the event must be in accordance with the order of execution, the message was a better way, because the message queue in accordance with the "first in first out" principle, as shown below. You can send a message in Task A and process the message in Task B.

The "put" operation in the figure above can be done via the TTCSdkTaskEnqueueMsg () function, which is "pushed in" the message queue; The "get" operation can be implemented by the Util_dequeueMsg () function, which takes the message first in the message queue and performs the corresponding event handling in the TTCBlePeripheralTaskProcessAppMsg () function.

The application task has created a corresponding message queue, no need to create a new message queue. To TTC_CC2640_SDK project serial demo, for example, how to send a message to deal with the message.

(1) Define a thread message handle:

```
<TTCBlePeripheralTask.c>
static Queue_Struct appMsg;                    // Message structure
static Queue_Handle appMsgQueue;               // Message handle
```

(2) Create a message queue:

```
<TTCBlePeripheralTask.c>
static void TTCBlePeripheralTaskInit(void){
    ICall_registerApp(&selfEntity, &sem);
    appMsgQueue = Util_constructQueue(&appMsg);  // create a message queue
    //…
    }
```

(3) Register the message function and pass the message handle address & appMsgQueue into the serial port initialization function:

```
<TTCBleSDKConfig.h>
typedef struct {                               //TTC SDK Public function
class
    TTCSdkSetEvent_t          pfnTTCSdkSetEvent;
    TTCSdkTaskEnqueueMsg_t    pfnTTCSdkTaskEnqueueMsg;
    TTCSdkDriverCB_t          pfnTTCSdkDriverCB;
```

```
    TTCSdkBleCB_t               pfnTTCSdkBleCB;
}TTCSdkClass_t;
-------------------------------------------------------------------------
<TTCBlePeripheralTask.c>
TTCSdkClass_t TTCBlePeripheralTaskCls = {
    TTCSdkSetEvent,
    TTCSdkTaskEnqueueMsg,
    TTCSdkDriverCB,
};
-------------------------------------------------------------------------
#ifdef TTCDRIVER_UART
   TTCDriverDemoUARTInit(&TTCBlePeripheralTaskCls, &sem, &appMsgQueue);
#endif //TTCDRIVER_UART
```

(4) After the Define serial application events;Message handle pointer UartMsgQueue;Callback function pointer UartCallbacks,and the the serial port initiazation success, the UartMsgQueue pointer points to the message handler defined in TTCBlePeripheralTask.c appMsgQueue. The UartCallbacks pointer points to TTCBlePeripheralTask.c defines the TTCBlePeripheralTaskCls.

```
<TTCBleSDKConfig.h>
#define TTCSDK_MSG_APP_UART_EVENT       0x000C    //TTCSDK serial application
event
-------------------------------------------------------------------------
<TTCDriverUARTDemo.c>
static Queue_Handle * UartMsgQueue = NULL;       // message handle pointer
static TTCSdkClass_t * UartCallbacks = NULL;     // Callback registration
-------------------------------------------------------------------------
void TTCDriverDemoUARTInit(TTCSdkClass_t *appCallbacks,
                           ICall_Semaphore * sem,
                           Queue_Handle * appMsgQueue){
    if(appCallbacks == NULL || sem == NULL || appMsgQueue == NULL){
       return;
    }
    UartSem       = sem;
    UartMsgQueue  = appMsgQueue;
    UartCallbacks = appCallbacks;
    //…
  }
```

(5) Use the callback message and set the semaphore, The message "pressed into" the message queue

```
<TTCDriverUARTDemo.c>
-------------------------------------------------------------------------------
void TTCSDKDriverUARTEvent(void){
    if(events & TTCBLE_SDK_UART_EVN){
        events &= ~TTCBLE_SDK_UART_EVN;
        TTCDriverDemoUARTSendMsg(TTCSDK_MSG_APP_UART_EVENT, 0, NULL);
    }
}
-------------------------------------------------------------------------------
static void TTCDriverDemoUARTSendMsg(uint16_t event,
                                     uint16_t len, uint8_t *pData){
    if ( UartCallbacks != NULL &&
        UartCallbacks->pfnTTCSdkTaskEnqueueMsg != NULL){
        UartCallbacks->pfnTTCSdkTaskEnqueueMsg(*UartSem,
                                               *UartMsgQueue,
                                               event,
                                               NULL,
                                               NULL);
    } else {
    }
  }
  Note: The actual call is TTCSdkTaskEnqueueMsg () through callback;
```

(6) Wake up the thread, remove the message from the message queue, and handle it

```
<TTCBlePeripheralTask.c>
static void TTCBlePeripheralTaskFxn(UArg a0, UArg a1){
  //…
  if (errno == ICALL_ERRNO_SUCCESS){
      TTCBlePeripheralProcessStack(selfEntity);         // Protocol stack
message processing
      while (!Queue_empty(appMsgQueue)){                //receiving messege
         TTCMsg_t *pMsg = (TTCMsg_t *)Util_dequeueMsg(appMsgQueue);
         if (pMsg){
              TTCBlePeripheralTaskProcessAppMsg(pMsg); //handle the message
              ICall_free(pMsg);                        // Release memory
           }
        }
   }
  //…
}
```

```
------------------------------------------------------------------------
static void TTCBlePeripheralTaskProcessAppMsg(TTCMsg_t *pMsg){
    switch (pMsg->hdr.event){
       //…
        case TTCSDK_MSG_APP_UART_EVENT:{
             // TTCSDK handle the serial port application event
             }break;
       //…
    }
}
```

1.5 Power down save

SNV is part of CC2640 internal flash. Depending on the configuration of the SDK project, the size of the SNV can be 0 or 4Kbyte. It can be used to store binding information or store a small amount of user data to save data.

Osal_snv_read () and osal_snv_write () are respectively read and write functions of SNV. The required parameters are NV ID, length and data pointer.

```
Function prototype:
uint8 osal_snv_read( osalSnvId_t id,      //NV ID
                     osalSnvLen_t len,    //data length
                     void *pBuf)          //Read data pointer

uint8 osal_snv_write( osalSnvId_t id,     //NV ID
                      osalSnvLen_t len,   //data length
                      void *pBuf)         //write data pointer
parameter description
(1) NV ID: range: 0x80~0x8F
    #define BLE_NVID_CUST_START 0x80
    #define BLE_NVID_CUST_END   0x8F
(2) Data length: at most 252 byte
Data pointer
```

1.6 Dynamic memory management

In order to flexible and reasonable use of memory, the application can dynamically apply for memory, after use must release the applied memory, or lead to memory leak! According to the SDK default settings, the default size of the cache is 2672Byte, the user application memory recommended no more than 700Byte; to dynamically apply for more memory space, please modify the macro HEAPMGR_SIZE size (Project-> Options-> C / C ++ Complier-> Defined Symbols).

The following is a brief example of memory application, memory release:

(1) When sending a message, the ICall_malloc () function requests the cache for the message data:

```
static u8 TTCSdkTaskEnqueueMsg(ICall_Semaphore sem,
                                Queue_Handle queueHandle,
                                u16 event,
                                u16 state,
                                void * pValue){
      u8 status = FALSE;
      TTCMsg_t *pMsg;
      if ((pMsg = ICall_malloc(sizeof(TTCMsg_t)))) {
          pMsg->hdr.event = event;
          pMsg->hdr.state = state;
          pMsg->pValue = pValue;
          status = Util_enqueueMsg(queueHandle, sem, (u8*)pMsg);
      }
      return status;
}
```

(2) From the message queue to remove the message, the message data is used, call ICall_free () release memory:

```
static void TTCBlePeripheralTaskFxn(UArg a0, UArg a1){
  //…
  if (errno == ICALL_ERRNO_SUCCESS){
      TTCBlePeripheralProcessStack(selfEntity);          // Protocol stack
message processing
      while (!Queue_empty(appMsgQueue)){                 // Receive the
message
         TTCMsg_t *pMsg = (TTCMsg_t *)Util_dequeueMsg(appMsgQueue);
         if (pMsg){
              TTCBlePeripheralTaskProcessAppMsg(pMsg); //Process the message
              ICall_free(pMsg);                          // Release memory!!!
           }
        }
   }
  //…
}
```

2. Contact us

深圳市昇润科技有限公司
ShenZhen ShengRun Technology Co.,Ltd.
Tel: 0755-86233846 Fax: 0755-82970906

Website：www.tuner168.com

Alibaba Shop：http://shop1439435278127.1688.com

E-mail: marketing@tuner168.com

地址：广东省深圳市南山区西丽镇龙珠四路金谷创业园 B 栋 6 楼 601-602
ADD: the 6F, BBlock of jingu Pioneer Park, longzhu 4th Road, Nanshan District, Shenzhen, China