密级：公开资料

TTC BLE SDK 说明

文档版本：V1.6

深圳市昇润科技有限公司

2017 年 01 月 16 日

版本	修订日期	修订人	审稿人	修订内容
1.0	2016-06-13	廖健焜	张眼	初稿
1.1	2016-12-05	徐凯翔	张眼/廖健焜	1.新增 SDK 驱动说明 2.修改部分原有 SDK API 说明
1.2	2016-12-09	徐凯翔	张眼/廖健焜	1.修改GPIO使用示例的添加步骤 2.修改了目录里某些条目的名称
1.3	2016-12-13	徐凯翔	张眼/廖健焜	1.增加生产测试说明部分
1.4	2017-01-06	郭高亮/ 廖健焜	廖健焜	1.增加 TI-RTOS 简介 2.增加 OAD 操作说明 3.修改程序空间计算 4.新增主机 API
1.5	2017-01-12	郭高亮	廖健焜	1.修改 OAD bin 文件生成方法
1.6	2017-01-16	郭高亮		1.去除 TI-RTOS 简介 2.增加 TTC_BLE 二维码

目　　录

1. TTC SDK 简介

TTC SDK 是由我司提供的 CC2640 快速开发库。旨在让开发人员不再需要将大量精力放在蓝牙调试方面，只需将精力放在对 CC2640 功能上的开发。TTC SDK 提供了蓝牙参数设置、蓝牙数据收发、蓝牙状态处理等 API，同时也提供了测试程序，开发人员无需再设计测试程序。

使用 TTC SDK 能适配我司提供的 TTC-BLE 软件，方便调试数据收发，并且支持数据加密解密功能。能极大的缩短 CC2640 的开发周期。

1.1. CC2640 SDK 从机开源特征

SDK 中 SimpleBLEPeripheral 工程中包含有 3 种配置，配置区别以及空间计算如下表。

工程	OAD	测试程序	ROM(byte)			RAM(byte)		
			总量	用量	剩余	总量	用量	剩余
0（SNV=1 BOND）	无	无	65536	26602	38934	17388	10612	6776
		有		35439	30097		12189	5199
	片内	无	45056	32453	12603	17408	10876	6532
		有		41495	3561		12457	4951
	片外	无	61440	39222	22218	17388	12507	4881
		有		47626	13814		14080	3308
1（SNV=1 NO BOND）	无	无	69632	26602	43030	17592	10612	6980
		有		35439	34193		12189	5403
	片内	无	49152	32453	16699	17488	10876	6612
		有		41495	7657		12457	5031
	片外	无	65536	39222	26314	17592	12507	5085
		有		47626	17910		14080	3512
2（SNV=0 NO BOND）	无	无	73728	26602	47126	17596	10612	6984
		有		35439	38289		12189	5407
	片内	无	53248	32453	20795	17488	10876	6612
		有		41495	11753		12457	5031
	片外	无	69632	39222	30410	17596	12507	5089
		有		47626	22006		14080	3516

1.2. 公共特征

- 蓝牙服务 UUID：1000

蓝牙通道	UUID	通道特性	功能概述
UUID1	1001	Write_NoRsp/Read/Notify	蓝牙数据接收
UUID2	1002	Read/Notify	蓝牙数据发送
UUID3	1003	Write_NoRsp	寄存器写数据
UUID4	1004	Read	寄存器读数据
UUID5	1005	Write_NoRsp/Read	选定寄存器

1.3. TTC SDK 优势

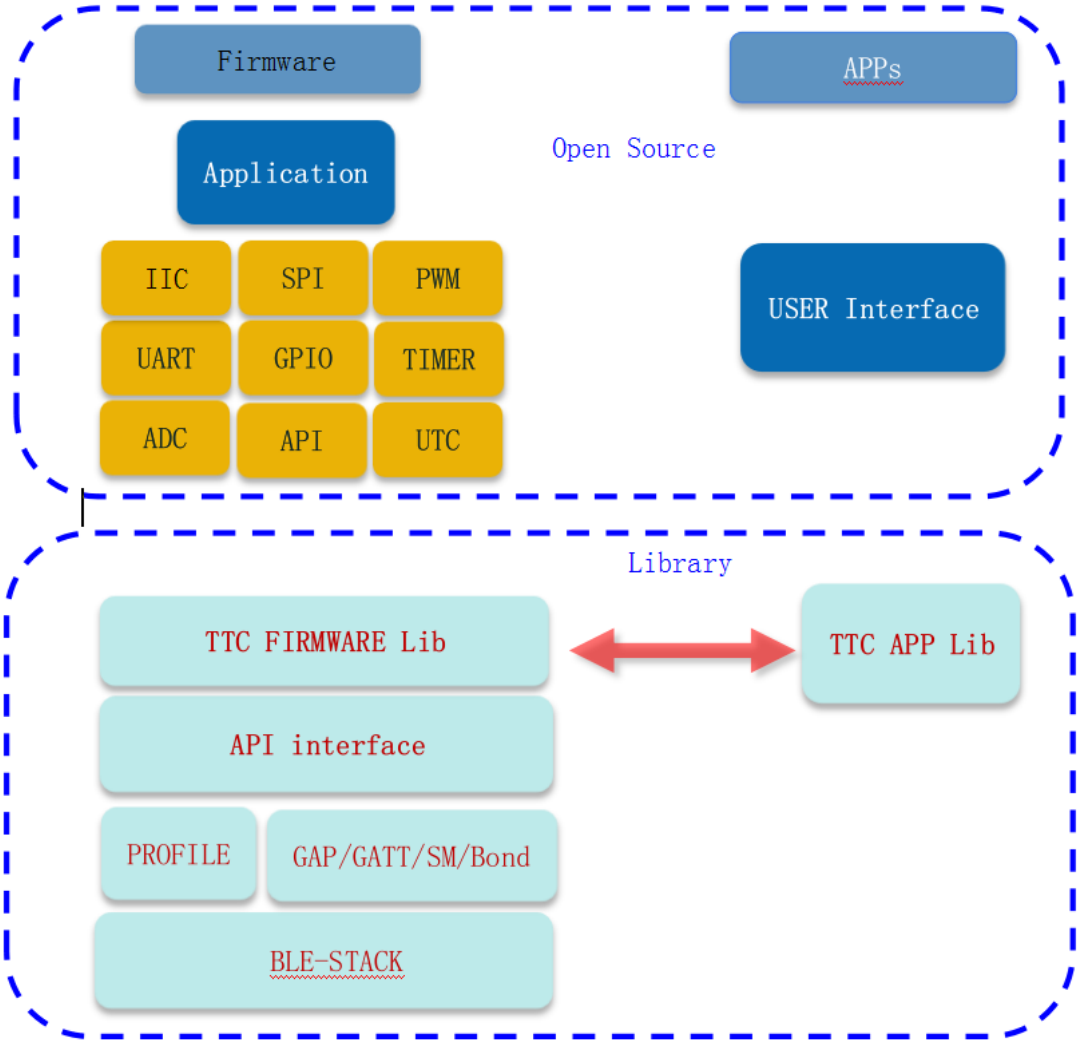
- 完整的蓝牙解决方案(IC+固件+ APP+云端)
- 简单的蓝牙设置以及轻松更新固件
- 类似串口数据收发的蓝牙交互模式
- 完整的 SDK 以及工具提供（SDK 就绪）
- 快速启动时间（RTOS < 500ms ， OSAL < 500ms）
- 超低功耗特性，CC2640 低至 1.5uA 睡眠电流，可用电池供电
- 数据支持 AES 加密解密
- 完整的蓝牙参数验证，适应安卓、IOS 两大平台，用户无需烦恼蓝牙参数适配问题
- 完整的测试方案提供，SDK 已包含测试程序，用户无需设计蓝牙测试程序
- 配套的专业测试架、SDK 包、开发调试工具、DEMO 板
- 减少由于蓝牙导致设备工作异常的情况出现

1.4. TTC SDK 解决问题

- 蓝牙固件
- 双平台蓝牙开发 SDK
- CC2640 驱动，已有驱动如下：

  ADC
  SPI
  UART
  IIC
  Timer（包含 PWM/定时/输入捕获）
  GPIO
  WatchDog
  Wechat

2. SDK 结构图


Firmware
APPs
Application
Open Source
IIC
SPI
PWM
USER Interface
UART
GPIO
TIMER
ADC
API
UTC
Library
TTC FIRMWARE Lib
TTC APP Lib
API interface
PROFILE
GAP/GATT/SM/Bond
BLE-STACK

3. TTC SDK API 说明

3.1. TTC SDK 蓝牙部分文件说明

TTC SDK 共有 5 个头文件

- TTCBleSDKConfig.h :TTC SDK 工程配置头文件
- TTCBleDevInfoService.h:TTC SDK 设备信息服务头文件
- TTCBlePeripheral.h:TTC SDK 从机线程头文件
- TTCBlePeripheralProcess.h:TTC SDK 从机处理头文件
- TTCBleProfile.h:TTC SDK 处理蓝牙数据头文件
- TTCBleCentral.h:TTC SDK 主机线程头文件
- TTCBleCentralProcess.h:TTC SDK 主机处理头文件

注意：

在函数说明上写了“请勿调用该函数，该函数为 SDK 功能。”。请用户不要自行调用或删除该函数。

3.2. 蓝牙初始化 API(TTCBlePeripheralProcess.h)

通过以下结构体可以直接设置蓝牙的参数。并将该结构体传递到函数 TTCBlePeripheralInit(TTCBlePeripheralInitConf_t * config)中。

蓝牙从机参数初始化结构体

```
typedef struct{
    TTCData_t             advData;          //广播数据
    TTCData_t             scanRspData;      //扫描回应数据
    TTCData_t             attDevName;       //设置通用设备特性名称
    u16                   maxConnInterval;  //最大连接间隔
    u16                   minConnInterval;  //最小连接间隔
    u16                   advOffTime;       //通过设置为零,设备将进入等待状态被发现后 30.72
                                            //秒,又不会被广告直到设置为 TRUE
    u8                    advEnable;        //蓝牙广播使能
    u8                    updateParEnable;  //参数更新使能
    u16                   updateParDelay;   //参数更新延迟(默认 6)
    u16                   slaveLatency;     //从机跳过的回应包包数
    u16                   connTimeout;      //连接超时
    u8                    txPower;          //TX
    u16                   advInterval;      //广播间隔
    u16                   rssiReadPeriod;   //RSSI 刷新周期
    u8                    encryptEnable;    //加密使能
    u8                    advNoticeEnable;  //广播回调使能
    TTCPeripheralCBFxn_t  CB;               //回调函数
    TTCSdkClass_t         *appCB;           //SDK 通用回调类
    ICall_Semaphore       *sem;             //线程信号量
    Queue_Handle          *queueHandle;     //消息句柄
    ICall_EntityID        *entity;          //任务 ID
}TTCBlePeripheralInitConf_t;
```

```
/*******************************************************************************
【函    数】  TTCBlePeripheralInit(TTCBlePeripheralInitConf_t * config)
【概    述】  蓝牙从机初始化
【入口参数】  config:蓝牙参数配置初始化结构体
【返回参数】  无
【说    明】  无
*******************************************************************************/
void TTCBlePeripheralInit(TTCBlePeripheralInitConf_t * config);
```

蓝牙主机参数初始化结构体

```
typedef struct{
    u16             scanDuration;           //扫描持续时间(单位 ms)
    u16             scanInterval;           //扫描间隔
    u16             scanWindows;            //扫描窗口
    u16             rssiReadPeriod;         //RSSI 获取周期(暂不开放)
    u8              encryptEnable;          //加密使能
    TTCBleCentralScanType_t  scanResultType;  //扫描结果类型(可根据需要设置所需要的返回数据
                                            //内容 设置类型参考 TTCBleCentralScanType_t )
    u8              filterType;             //过滤类型(可以设置需要过滤的广播类型
                                            //设置类型参考 @defgroup GAP_ADTYPE_DEFINES )
    u8              maxScanResult;          //设置最大扫描个数
    const TTCData_t  filterData;            //设置过滤广播数据内容
    const TTCData_t  attDevName;            //设置通用设备特性名称
    TTCSdkClass_t    *appCB;                //SDK 通用回调类
    ICall_Semaphore  *sem;                  //线程信号量
    Queue_Handle     *queueHandle;          //消息句柄
    ICall_EntityID   *entity;               //任务 ID
}TTCBleCentralInitConf_t;
/*******************************************************************************
【函    数】  TTCBleCentralInit(TTCBleCentralInitConf_t * config)
【概    述】  蓝牙主机初始化
【入口参数】  config : 蓝牙参数配置初始化结构体
【返回参数】  无
【说    明】  无
*******************************************************************************/
extern TTCDriverInfo_t TTCBleCentralInit(TTCBleCentralInitConf_t * config);
```

3.3.蓝牙参数及相关蓝牙操作 API(TTCBlePeripheral.h)

```
/*******************************************************************************
【函    数】  TTCBlePeripheralSetParameter(u16 param, uint8_t len, void *pValue)
【概    述】  从机参数设置
【入口参数】  param:参数，参数内容请参考 GAPROLE_PROFILE_PARAMETERS
              len:参数长度
              pValue:参数内容
【返回参数】  无
【说    明】  无
*******************************************************************************/
bStatus_t TTCBlePeripheralSetParameter(u16 param, uint8_t len, void *pValue);

/*******************************************************************************
【函    数】  TTCBlePeripheralGetParameter(u16 param, void *pValue)
【概    述】  从机获取参数
【入口参数】  param:参数，参数内容请参考 GAPROLE_PROFILE_PARAMETERS 参数说明
              *pValue:参数内容
【返回参数】  无
【说    明】  无
*******************************************************************************/
bStatus_t TTCBlePeripheralGetParameter(u16 param, void *pValue);

/*******************************************************************************
【函    数】  TTCBlePeripheralGAPRoleTerminateConnection(void)
【概    述】  断开蓝牙连接
【入口参数】  无
【返回参数】  无
【说    明】  无
*******************************************************************************/
bStatus_t TTCBlePeripheralGAPRoleTerminateConnection(void);

/*******************************************************************************
【函    数】  TTCBlePeripheralGAPRoleTerminateConnection(void)
【概    述】  断开蓝牙连接
【入口参数】  无
【返回参数】  无
【说    明】  无
*******************************************************************************/
bStatus_t TTCBlePeripheralGAPRoleTerminateConnection(void);
```

```
/*******************************************************************************
【函    数】  TTCBlePeripheralGAPRoleSendUpdateParam(u16 minConnInterval,
                                                    u16 maxConnInterval,
                                                    u16 latency,
                                                    u16 connTimeout,
                                                    u8 handleFailure);
【概    述】  发送蓝牙参数更新请求
【入口参数】  minConnInterval:最小连接间隔
              maxConnInterval:最大连接间隔
              latency:蓝牙从机延迟（允许跳过回应包包数）
              connTimeout:连接超时
              handleFailure:更新失败操作。请参考 GAPROLE_FAILED_UPDATE 参数说明
【返回参数】  无
【说    明】  无
*******************************************************************************/
bStatus_t TTCBlePeripheralGAPRoleSendUpdateParam(u16 minConnInterval,
                                                 u16 maxConnInterval,
                                                 u16 latency,
                                                 u16 connTimeout,
                                                 u8 handleFailure);
```

3.4.蓝牙数据发送相关 API（TTCBleProfile.h）

```
/*******************************************************************************
【函    数】  TTCBleProfileSetParameter( u8 param,u16 len, void *value )
【概    述】  设置蓝牙通道数据
【入口参数】  param:设置设备信息服务参数，填入的参数如下
                                    TTCBLE_PROFILE_CHAR1（暂不开放）
                                    TTCBLE_PROFILE_CHAR2
                                    TTCBLE_PROFILE_CHAR3（暂不开放）
                                    TTCBLE_PROFILE_CHAR4（暂不开放）
                                    TTCBLE_PROFILE_CHAR5（暂不开放）
              len:发送数据长度
              value:数据
【返回参数】  无
【说    明】  无
*******************************************************************************/
extern bStatus_t TTCBleProfileSetParameter( u8 param,u16 len, u8 * value );
```

3.5.蓝牙主机相关操作 API（TTCBleCentralPorcess.h）

```
/*****************************************************************************
【函    数】  TTCBleCentralProcEstablishLink(u8 scanResIndex)
【概    述】  简易模式调用连接函数
【入口参数】  scanResIndex:返回扫描结果的索引号
【返回参数】  TRUE:调用成功，当前处于正在连接状态
              FALSE:调用失败
【说    明】  调用失败原因：1.SDK 中扫描结果为空
                          2.扫描索引超出扫描结果总数
*****************************************************************************/
extern bool TTCBleCentralProcEstablishLink(u8 scanResIndex);
/*****************************************************************************
【函    数】  TTCBleCentralProcTerminateLink(u8 connHandle)
【概    述】  简易模式调用断开函数
【入口参数】  connHandle:当前的连接句柄
【返回参数】  TRUE:调用成功，当前处于正在断开状态
              FALSE:调用失败
【说    明】  调用失败原因：1.连接句柄错误
                          2.当前未处于连接状态
*****************************************************************************/
extern bool TTCBleCentralProcTerminateLink(uint16_t connHandle);
/*****************************************************************************
【函    数】  TTCBleCentralProcStartDiscovery(void)
【概    述】  简易模式调用扫描广播函数
【入口参数】  无
【返回参数】  TRUE:调用成功，当前处于正在扫描状态
              FALSE:调用失败
【说    明】  调用失败原因：1.当前设备未准备好
                          2.可能处于正在连接或正在断线状态
              如简易模式不能满足需求，
              请自行调用 TTCBleCentral.h 中的 TTCBleCentralStartDiscovery
*****************************************************************************/
extern bool TTCBleCentralProcStartDiscovery(void);
/*****************************************************************************
【函    数】  TTCBleCentralProcWriteData(u16 connHandle,u8 *wBuf,u8 len)
【概    述】  简易模式调用数据发送函数
【入口参数】  connHandle : 连接句柄
              wBuf       : 发送数据
              len        : 数据长度
【返回参数】  TRUE:调用成功，当前处于正在发送数据状态
```

FALSE:调用失败

【说　　明】　调用失败原因：1.当前设备未连接
2.连接句柄错误
3.数据过长
4.主机仍未就绪
5.发送缓存为空

**********/

```
extern bool TTCBleCentralProcWriteData(u16 connHandle,u8 *wBuf,u8 len);
```

3.6.SDK 线程消息说明

在 TTCBleSDKConfig.h 头文件中定义了如下的消息事件（请勿修改）

```
#define TTCSDK_MSG_GET_BLE_STATE_EVENT      0x0001                  //蓝牙状态事件
#define TTCSDK_MSG_GET_BLE_DATA_EVENT       0x0002                  //蓝牙数据事件
#define TTCSDK_MSG_REFRESH_RSSI_EVENT       0x0003                  //刷新 RSSI 值
#define TTCSDK_MSG_GET_BLE_PARAM_EVENT      0x0004                  //获取协商后的蓝牙参数
#ifdef FEATURE_OAD
#define TTCSDK_MSG_OAD_EVENT                0x0005                  //片外 OAD 升级事件
#endif //FEATURE_OAD
#define TTCSDK_MSG_DRIVER_UART_EVENT        0x0006                  //TTCSDK 驱动事件 UART
#define TTCSDK_MSG_DRIVER_SPI_EVENT         0x0007                  //TTCSDK 驱动事件 SPI
#define TTCSDK_MSG_DRIVER_I2C_EVENT         0x0008                  //TTCSDK 驱动事件 IIC
#define TTCSDK_MSG_DRIVER_UTC_EVENT         0x0009                  //TTCSDK 驱动事件 UTC
#define TTCSDK_MSG_DRIVER_PWM_EVENT         0x000A                  //TTCSDK 驱动事件 PWM
#define TTCSDK_MSG_BLE_WECHAT_EVENT         0x000B                  //TTCSDK 蓝牙微信事件
#define TTCSDK_MSG_GET_BLE_SCAN_RES_EVENT   0x000C                  //蓝牙获取扫描结果事件
#define TTCSDK_MSG_GET_CONINFO_EVENT        0x000D                  //蓝牙获取连接信息事件
```

在 TTCBlePeripheralTask.c 中对消息进行接收处理。

```
/**********************************************************************************
```

【函　　数】　TTCBlePeripheralTaskProcessAppMsg(TTCMsg_t *pMsg)

【概　　述】　线程消息处理函数

【入口参数】　pMsg:消息数据

【返回参数】　无

【说　　明】　请注意内存的释放

```
**********************************************************************************/
static void TTCBlePeripheralTaskProcessAppMsg(TTCMsg_t *pMsg);
```

在 TTCBleCentralTask.c 中对消息进行接收处理。

```
/**********************************************************************************
```

【函　　数】　TTCBleCentralTaskProcessAppMsg(TTCMsg_t *pMsg)

【概　　述】　线程消息处理函数

【入口参数】　pMsg ： 消息数据

【返回参数】　无

【说　　明】　请注意内存的释放

```
*****************************************************************************/
static void TTCBleCentralTaskProcessAppMsg(TTCMsg_t *pMsg)
```

3.6.1. TTCSDK_MSG_GET_BLE_STATE_EVENT 蓝牙状态处理

该事件会对蓝牙的状态进行处理，开发人员如需根据蓝牙状态进行应用，可在此 TTCSDK_MSG_GET_BLE_STATE_EVENT 事件中进行处理。

注意：请不要删除 TTCBlePeripheralProcessStateChangeEvt((gaprole_States_t)pMsg->hdr.state)(从机部分)

TTCBleCentralStateChange((gaprole_States_t)pMsg->hdr.state);(主机部分)

3.6.2. TTCSDK_MSG_GET_BLE_DATA_EVENT 蓝牙接收数据处理

开发人员可在主从机模式下分别可以通过以下函数处接收并处理蓝牙数据

TTCBlePeripheralTaskGetBleData(TTCMsg_t * TTCMsg)

TTCBleCentralStateChange(TTCMsg_t * TTCMsg)

TTCMsg->hdr.state 携带的内容表示蓝牙通道。TTCData->param 表示蓝牙数据是否正常。在开启加密的情况下，若加密数据有误，将不会对数据进行解密，并且 TTCData->param 携带的参数为 TTCSDK_ERR_ENCRYPT_DATA，表示数据错误。其他情况下 TTCData->param 携带的参数为 TTCSDK_NOERR_DATA。消息传递蓝牙数据结构体为

```
typedef struct{
    u16         len         ;//长度
    u16         param       ;//参数
    u8 *        pValue      ;//数据内容
}TTCData_t;
```

3.6.3. TTCSDK_MSG_REFRESH_RSSI_EVENT 连接状态下 RSSI 值读取

该事件用于处理设备与主机连接后的获取到的 RSSI 值。开发人员可在 TTCBl-ePeripheralSetParameter 函数中对 RSSI 的刷新周期进行设置，设置 RSSI 刷新周期参数为 GAPROLE_RSSI_READ_RATE。当设置值为 0 时，将不会刷新 RSSI 值。当设置值不为 0 时，将会以设置值为周期，定期的更新 RSSI 值。消息传递的内容为 s8(signed char)类型数据。

注意：该 RSSI 值只会在连接状态下刷新。

3.6.4. TTCSDK_MSG_GET_BLE_PARAM_EVENT 协商后的蓝牙参数

该事件用于告知开发人员主从机蓝牙参数协商后的蓝牙参数。开发人员可在该事件中获取到最终协商的蓝牙参数。消息传递蓝牙参数的结构体为

```
typedef struct{
    u16     connInterval            ;//连接间隔
    u16     connSlaveLatency        ;//从机跳过回应包包数
    u16     connTimeout             ;//连接超时
}TTCBleParamUpdate_t;
```

3.6.5. TTCSDK_MSG_GET_BLE_SCAN_RES_EVENT 获取扫描结果事件

该事件用于获取主机扫描的结果。扫描结果结构体为

```
typedef struct{
    u8   MAC[6];              //扫描到的设备 MAC 地址
    u8   addrType;            //匹配地址类型
    s8   rssi;                //RSSI 值
    u8   advDataLen;          //广播数据长度
    u8   scanDataLen;         //扫描回应数据长度
    u8   *advData;            //广播数据内容
    u8   *scanData;           //扫描回应数据内容
}TTCBleCentralScanResult_t;
```

3.6.6. TTCSDK_MSG_GET_CONINFO_EVENT 获取连接信息事件

该事件用于获取主机连接信息。连接信息结构体为

```
typedef struct{
    u16  connHandle;          //连接句柄
    u8   connDevAddr[6];      //当前连接的设备 MAC 地址
}TTCBleCentralConnInfo_t;
```

3.7. 设备信息服务参数设置（TTCBleDevInfoService.h）

/***

【函　　数】 TTCBleDevInfoSetParameter(u8 param, u8 len, void *value)

【概　　述】 设置设备信息服务参数

【入口参数】 param:设置设备信息服务参数，填入的参数如下

DEVINFO_SYSTEM_ID
DEVINFO_SERIAL_NUMBER
DEVINFO_FIRMWARE_REV
DEVINFO_HARDWARE_REV
DEVINFO_SOFTWARE_REV
DEVINFO_MANUFACTURER_NAME
DEVINFO_11073_CERT_DATA
DEVINFO_PNP_ID

len: 数据长度，对应参数的数据长度如下

DEVINFO_SYSTEM_ID_LEN	8（固定值，长度必须等于 8 否则无效）
DEVINFO_SERIAL_NUMBER_LEN	21
DEVINFO_FIRMWARE_REV_LEN	21
DEVINFO_HARDWARE_REV_LEN	21
DEVINFO_SOFTWARE_REV_LEN	21
DEVINFO_MANUFACTURER_NAME_LEN	21
DEVINFO_11073_CERT_DATA	自定义
DEVINFO_PNP_ID_LEN	7（固定值，长度必须等于 7 否则无效）

value:设置数据

【返回参数】 无

【说　　明】　无

**********/

```
extern bStatus_t TTCBleDevInfoSetParameter( u8 param, u8 len, void *value );
```

/**********

【函　　数】　TTCBleDevInfoGetParameter(u8 param, void *value)

【概　　述】　获取设备信息服务参数

【入口参数】　param:设置参数，填入的参数如下

DEVINFO_SYSTEM_ID

DEVINFO_SERIAL_NUMBER

DEVINFO_FIRMWARE_REV

DEVINFO_HARDWARE_REV

DEVINFO_SOFTWARE_REV

DEVINFO_MANUFACTURER_NAME

DEVINFO_11073_CERT_DATA

DEVINFO_PNP_ID

value:读取数据

【返回参数】　无

【说　　明】　无

**********/

```
extern bStatus_t TTCBleDevInfoGetParameter( u8 param, void *value );
```

3.8. 应用线程公共回调函数说明

在 TTCBleSDKConfig.h 中声明了公共回调函数（请勿修改）

/**********

【函　　数】　(*TTCSdkSetEvent_t)(ICall_Semaphore sem,

u16 * events,

UArg arg)

【概　　述】　线程置事件

【入口参数】　sem　　：线程信号量

events ：事件源

arg　　：标记事件源事件

【返回参数】　无

【说　　明】　该函数封装到 TTCBlePeripheralTaskClass_t 中，为公用函数

**********/

```
typedef void (*TTCSdkSetEvent_t)(ICall_Semaphore sem,
                                 u16 * events,
                                 UArg arg);
```

/**********

【函　　数】　(*TTCSdkTaskEnqueueMsg_t)(ICall_Semaphore sem,

```
                                    Queue_Handle queueHandle,
                                    u16 event,
                                    u16 state,
                                    void * pValue)
```

【概　　述】　线程置事件

【入口参数】　sem　　: 信号量

queueHandle : 消息句柄

event　　: 消息携带的事件

state　　: 消息携带的状态

pValue　　: 消息携带的数据

【返回参数】　无

【说　　明】　该函数封装到 TTCBlePeripheralTaskClass_t 中，为公用函数请勿修改

```
*****************************************************************************/
typedef u8 (*TTCSdkTaskEnqueueMsg_t)(ICall_Semaphore sem,
                                     Queue_Handle queueHandle,
                                     u16 event,
                                     u16 state,
                                     void * pValue);
/*****************************************************************************
```

【函　　数】　(*TTCSdkDriverCB_t)(TTCDriverType_t driverType,

u32 driverState,

u8 * buffer,

u16 len)

【概　　述】　SDK 驱动回调

【入口参数】　driverType　: 驱动类型

driverState : 驱动状态

buffer　　: 数据缓存

len　　: 数据长度

【返回参数】　无

【说　　明】　该函数封装到 TTCBlePeripheralTaskClass_t 中，用户可根据使用情况自行添加处理。

注意：严禁在此函数调用涉及到中断的函数。

```
*****************************************************************************/
typedef void (*TTCSdkDriverCB_t)(TTCDriverType_t driverType,
                                 u32 driverState,
                                 u8 * buffer,
                                 u16 len);

/*****************************************************************************
```

【函　　数】　(*TTCSdkBleCB_t)(TTCBleType_t bleType,

u8　　param,

u16　　*len,

```
                                u8          *pValue)
```

【概　　述】 BLE 相关回调

【入口参数】 bleType ： BLE 相关操作类型

param ： 参数

len ： 数据长度

pValue ： 数据

【返回参数】 无

【说　　明】 该函数封装到 TTCBlePeripheralTaskClass_t 中，用户可根据使用情况自行添加处理。

注意：严禁在此函数调用涉及到中断的函数。

```
*****************************************************************************/
typedef void (*TTCSdkBleCB_t)(TTCBleType_t bleType,
                              u8           param,
                              u16          *len,
                              u8           *pValue);
typedef struct {
    TTCSdkSetEvent_t            pfnTTCSdkSetEvent;
    TTCSdkTaskEnqueueMsg_t      pfnTTCSdkTaskEnqueueMsg;
    TTCSdkDriverCB_t            pfnTTCSdkDriverCB;
    TTCSdkBleCB_t               pfnTTCSdkBleCB;
}TTCSdkClass_t;
```

以上回调函数原型在 TTCBlePeripheralTask.c 中，开发人员可用回调置起线程事件和发送消息。

注意：TTCSdkDriverCB_t 暂不开放

4. TTC SDK 驱动 API 说明

4.1. GPIO 说明

CC2640 拥有丰富的 GPIO 资源，满足各种开发需求。最多可提供 31 个 GPIO 供开发人员使用，支持多种配置比如上下拉、开漏、推挽输出等。每个 GPIO 都可以配置中断功能，中断的方式也可以灵活配置，比如上升沿中断、下降沿中断、上升下降沿都中断等。另外每个 GPIO 都可以任意映射片内的外设资源，比如 PWM 输出口，ADC 输入口等。

4.1.1. GPIO API 说明

4.1.1.1 TTCDriverIOOpen()

```
/*****************************************************************************
【函    数】  TTCDriverIOOpen(PIN_Handle * pinHandle,
                              PIN_State * pinState,
                              const PIN_Config pinList[])
【概    述】  打开 IO 口功能
【入口参数】  pinHandle : 控制 IO 句柄
              pinState  : 控制 IO 状态
              pinList   : IO 口配置
【返回参数】  MANAGER_INFO_REQUEST_IO_SUCCESS：打开 IO 口成功
              MANAGER_INFO_REQUEST_IO_FAILED：打开 IO 口失败
【说    明】  若打开 IO 口失败，请检查是否有外设使用了配置中的 IO 口。
*****************************************************************************/
extern TTCBleSDKManagerInfo_t TTCDriverIOOpen(PIN_Handle * pinHandle,
                                              PIN_State * pinState,
                                              const PIN_Config pinList[]);
```

说明：若想使用 GPIO，最先调用的一定是这个函数，功能是为一个或多个 GPIO 分配资源，同时返回一个句柄，GPIO 的其他所有操作都是根据这个句柄来进行的，在未获取到有效的句柄前，不能操作 GPIO。

4.1.1.2 TTCDriverIOAdd()

```
/*****************************************************************************
【函    数】  TTCDriverIOAdd(PIN_Handle * pinHandle,
                             PIN_Config addIO)
【概    述】  添加 IO 口到句柄中
【入口参数】  pinHandle : 控制 IO 句柄
              addIO     : 添加的 IO 口及配置
【返回参数】  MANAGER_INFO_REQUEST_IO_SUCCESS：添加 IO 口成功
```

MANAGER_INFO_REQUEST_IO_FAILED：添加 IO 口失败

【说　　明】　若打开 IO 口失败，请检查是否有外设使用了该 IO 口。

**********/

```
extern TTCBleSDKManagerInfo_t TTCDriverIOAdd(PIN_Handle * pinHandle,
                                             PIN_Config addIO);
```

说明:此 API 用于向已经 Open 的一个 GPIO 句柄里添加一个新的 GPIO,使用时只需提供指定的且有效的 GPIO 句柄以及新增的 GPIO 的配置信息即可，添加完成之后即可对新增的 GPIO 进行操作。

4.1.1.3 TTCDriverIORemove()

/**********

【函　　数】　TTCDriverIORemove(PIN_Handle * pinHandle,
PIN_Id removeIO)

【概　　述】　从句柄中移除 IO 口

【入口参数】　pinHandle ： 控制 IO 句柄

removeIO　： 移除的 IO 口

【返回参数】　MANAGER_INFO_RELEASE_IO_SUCCESS：添加 IO 口成功

MANAGER_INFO_RELEASE_IO_FAILED：添加 IO 口失败

【说　　明】　若打开 IO 口失败，请检查是否有外设使用了配置中的 IO 口。

**********/

```
extern TTCBleSDKManagerInfo_t TTCDriverIORemove(PIN_Handle * pinHandle,
                                                PIN_Id removeIO);
```

说明：此 API 用户从一个已经 Open 的 GPIO 句柄里删除一个 GPIO，删除完成之后该 GPIO 即可做其他用途。

4.1.1.4 TTCDriverIOGetInputValue()

/**********

【函　　数】　TTCDriverIOGetInputValue(PIN_Id pinId)

【概　　述】　读取 IO 口的输入值

【入口参数】　pinId ： 需要获取的 IO 口

【返回参数】　1：高电平

0：低电平

【说　　明】　无

**********/

```
extern u8 TTCDriverIOGetInputValue(PIN_Id pinId);
```

说明：我们可以通过此 API 来获取具备输入功能的 GPIO 口的电平状态。

4.1.1.5 TTCDriverIOGetOutputValue()

```
/*****************************************************************************
【函    数】  TTCDriverIOGetOutputValue(PIN_Id pinId)
【概    述】  读取 IO 口的输出值
【入口参数】  pinId：需要获取的 IO 口
【返回参数】  1：高电平
              0：低电平
【说    明】  无
*****************************************************************************/
extern u8 TTCDriverIOGetOutputValue(PIN_Id pinId);
```

说明：我们可以通过此 API 来获取具备输出功能的 GPIO 口的电平输出状态，一个典型的例子就是用于 GPIO 口的电平翻转输出实验。

4.1.1.6 TTCDriverIOSetOutputVaule()

```
/*****************************************************************************
【函    数】  TTCDriverIOSetOutputVaule(PIN_Handle * pinHandle,
                                        PIN_Id pinId,
                                        u8 val)
【概    述】  设置 IO 口值
【入口参数】  pinHandle ：控制 IO 句柄
              pinId     ：需要控制的 IO 口
              val       ：想要输出的值
【返回参数】  MANAGER_INFO_CONTROL_IO_FAILED：  IO 口设置失败
              MANAGER_INFO_CONTROL_IO_SUCCESS：IO 口设置成功
              MANAGER_INFO_IO_NOT_ALLOCATED：该句柄中未包含该 IO 口
              MANAGER_INFO_IO_HANDLE_ERROR：句柄错误
【说    明】  无
*****************************************************************************/
extern TTCBleSDKManagerInfo_t TTCDriverIOSetOutputVaule(PIN_Handle * pinHandle,
                                                        PIN_Id pinId,
                                                        u8 val);
```

说明：这个 API 可以为一个具备输出功能的 GPIO 设置输出电平。

4.1.1.7 TTCDriverIOSetConfig()

```
/*****************************************************************************
【函    数】  TTCDriverIOSetConfig(PIN_Handle * pinHandle,
                                   PIN_Config bmMask,
                                   PIN_Config pinCfg)
```

```
【概      述】  设置 IO 口功能
【入口参数】  pinHandle：控制 IO 句柄
              bmMask：标记 IO 口操作
              pinCfg：IO 口配置
【返回参数】  MANAGER_INFO_CONTROL_IO_FAILED：  IO 口设置失败
              MANAGER_INFO_CONTROL_IO_SUCCESS：IO 口设置成功
              MANAGER_INFO_IO_NOT_ALLOCATED：该句柄中未包含该 IO 口
              MANAGER_INFO_IO_HANDLE_ERROR：句柄错误
【说      明】  无
*******************************************************************************/
TTCBleSDKManagerInfo_t TTCDriverIOSetConfig(PIN_Handle * pinHandle,
                                            PIN_Config bmMask,
                                            PIN_Config pinCfg);
```

说明：此 API 是用于为 GPIO 设置一些特殊的功能配置的，比如中断功能的配置等，具体的使用方法请参考下一小节的使用实例。

4.1.1.8 TTCDriverIOGetConfig()

```
/*******************************************************************************
【函      数】  TTCDriverIOGetConfig(PIN_Id pinId)
【概      述】  读取 IO 口配置
【入口参数】  pinId ： 需要获取配置的 IO 口
【返回参数】  IO 口的配置值
【说      明】  无
*******************************************************************************/
extern PIN_Config TTCDriverIOGetConfig(PIN_Id pinId);
```

说明：此 API 用于获取指定 GPIO 的配置信息，根据这些信息我们就可以知道这个 GPIO 具备哪些功能。

4.1.1.9 TTCDriverIORegisterIntCallBack()

```
/*******************************************************************************
【函      数】  TTCDriverIORegisterIntCallBack(PIN_Handle *pinHandle,
                                               PIN_IntCb pCb)
【概      述】  设置 IO 口回调函数
【入口参数】  pinHandle ： 控制 IO 句柄
              pCb       ： 中断回调函数
【返回参数】  MANAGER_INFO_CONTROL_IO_FAILED：  IO 口设置失败
              MANAGER_INFO_CONTROL_IO_SUCCESS：IO 口设置成功
              MANAGER_INFO_IO_NOT_ALLOCATED：该句柄中未包含该 IO 口
```

MANAGER_INFO_IO_HANDLE_ERROR：句柄错误

【说　　明】　无

**********/

```
extern TTCBleSDKManagerInfo_t TTCDriverIORegisterIntCallBack(PIN_Handle *pinHandle,
                                                             PIN_IntCb pCb);
```

说明：此 API 用于为一个 GPIO 设置中断回调函数，具体的使用方法请参考下一小节的使用实例。

4.1.2.GPIO 使用示例

```
/*****************************************************************************
```

【文　　件】　TTCDriverGPIODemo.c

【概　　述】　TTC SDK GPIO 示例代码

【编　　写】　SDK 工作小组

【修　　订】　SDK 工作小组

【修订日期】　2016/12/01

【版　　本】　V1.0.0

【说　　明】

功能说明：

配置 IOID_9 为输出口，配置 IOID_1 为输入口且具备中断功能，IOID_1 每拉低一次，IOID_9 的状态就翻转一次。

添加的步骤：

1、添加对应的头文件　TTCDriverGPIODemo.h

2、在工程中的 Opions/C/C++ Compiler/Defined symbols 中定义 TTCDRIVER_GPIO

3、在主线程里的 static void TTCSDKDriverInit(void)函数中添加如下代码。

TTCDriverDemoIOInit(KeyPressHandler);

注意：若是使用 "透传开发套件" 调试定时器时，请将套件上除了 3.3V 之外的所有跳帽都拔掉.

```
*****************************************************************************/
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Queue.h>
#include <ti/drivers/PIN.h>
#include <ti/drivers/pin/PINCC26XX.h>
#include "TTCSDKBoard.h"
#include "TTCBleSDKConfig.h"
#include "TTCBleSDKManager.h"
#include "TTCDriverGPIODemo.h"

/*****************************************************************
 * 本地变量
 */
```

```
PIN_Handle ioTestHandle;
PIN_State  ioTestState;

PIN_Config ioTestConfig[] = {
  PIN_TERMINATE
};

// Key debounce clock
static Clock_Struct keyChangeClock;

// Value of keys Pressed
static uint8_t keysPressed;

// Pointer to application callback
keysPressedCB_t appKeyChangeHandler = NULL;

/*********************************************************************
 * 本地函数声明
 */
static void TTCDrierDemoIOIsrCallback(PIN_Handle handle, PIN_Id pinId);
static void TTCDriverDemoIOChangeClockHandler(UArg a0);


/*****************************************************************************
【函    数】  TTCDriverDemoIOInit(keysPressedCB_t appKeyCB)
【概    述】  TTCDriver 驱动初始化
【入口参数】  appKeyCB
【返回参数】  无
【说    明】  本函数配置 IOID_9 为输出口，配置 IOID_1 为输入口且具备中断功能
              IOID_1 每拉低一次，IOID_9 的状态就翻转一次。
*****************************************************************************/
void TTCDriverDemoIOInit(keysPressedCB_t appKeyCB){
  TTCBleSDKManagerInfo_t err;
  err = TTCDriverIOOpen(&ioTestHandle,&ioTestState,(const PIN_Config *)ioTestConfig);
  //一个 IO 组打开成功
  if(err != MANAGER_INFO_REQUEST_IO_SUCCESS){
    return;
  }

  //往一个组里添加一个 io，添加之后就可以根据这个共同的句柄来操作这个新增加的 io
  err = TTCDriverIOAdd(&ioTestHandle,IOID_9 |
                                     PIN_GPIO_OUTPUT_EN |
```

```
                                        PIN_INPUT_DIS |
                                        PIN_GPIO_HIGH);

  if(err != MANAGER_INFO_REQUEST_IO_SUCCESS){
    return;
  }

//控制这个新增加的 IO 输出低电平
  err = TTCDriverIOSetOutputVaule(&ioTestHandle,IOID_9,0);

  //表示操作不成功
  if(err != MANAGER_INFO_CONTROL_IO_SUCCESS){
    return;
  }


  //再添加一个 io，配置为上拉输入
  err = TTCDriverIOAdd(&ioTestHandle,IOID_1 | PIN_GPIO_OUTPUT_DIS | PIN_INPUT_EN | PIN_PULLUP);
  if(err != MANAGER_INFO_REQUEST_IO_SUCCESS){
    return;
  }
  //注册一个 io 中断回调函数
  err = TTCDriverIORegisterIntCallBack(&ioTestHandle,TTCDrierDemoIOIsrCallback);
  if(err != MANAGER_INFO_CONTROL_IO_SUCCESS){
    return;
  }

  //配置 IOID_1 的中断功能
  err = TTCDriverIOSetConfig(&ioTestHandle,PIN_BM_IRQ,IOID_1 | PIN_IRQ_NEGEDGE);
  if(err != MANAGER_INFO_CONTROL_IO_SUCCESS){
    return;
  }
  //如果系统开启了省电功能 则还需要为此 io 配置唤醒功能
  #ifdef POWER_SAVING

    //配置 IOID_1 的中断唤醒功能
    err=TTCDriverIOSetConfig(&ioTestHandle,
                             PINCC26XX_BM_WAKEUP,IOID_1|PINCC26XX_WAKEUP_NEGEDGE);

    if(err != MANAGER_INFO_CONTROL_IO_SUCCESS){
      return;
    }
```

```
  #endif

  // 设置按键处理函数(消抖处理)
  Util_constructClock(&keyChangeClock,                    //
                      TTCDriverDemoIOChangeClockHandler,
                      200,                                //200mS
                      0,                                  //循环的周期为 0(即不循环定时)
                      false,                              //先不启动这个定时器
                      0);                                 //不需要传输参数
  //注册用户的按键处理函数
  appKeyChangeHandler = appKeyCB;
}

/*********************************************************************************
【函    数】  TTCDiver_Inter_Callback(PIN_Handle handle, PIN_Id pinId)
【概    述】  io 中断回调函数
【入口参数】  handle：句柄
              pinId：IO 口
【返回参数】  无
【说    明】  中断回调处理函数
*********************************************************************************/
static void TTCDrierDemoIOIsrCallback(PIN_Handle handle, PIN_Id pinId){
  Util_startClock(&keyChangeClock);   //开启这个软件定时器(相当于延时)
}
/*********************************************************************************
【函    数】  TTCDriverDemoIOChangeClockHandler(UArg a0)
【概    述】
【入口参数】  a0：事件
【返回参数】  无
【说    明】  无
*********************************************************************************/
static void TTCDriverDemoIOChangeClockHandler(UArg a0){
  (void)a0;
  if(TTCDriverIOGetInputValue(IOID_1) == 0){   //判断 io 的电平
    keysPressed |= 0x01;
  }else{
    keysPressed &= ~0x01;
  }
  if (appKeyChangeHandler != NULL){
    (*appKeyChangeHandler)(keysPressed);    //调用 用户的处理函数
  }
}
```

```
/*****************************************************************************
【函    数】  KeyPressHandler(uint8_t keys)
【概    述】  用户的按键处理函数
【入口参数】  keys：按键码
【返回参数】  无
【说    明】  无
*****************************************************************************/
void KeyPressHandler(u8 keys){
  if(keys & 0x01){
      TTCBleSDKManagerInfo_t err;
      //取反 IOID_9 脚的状态
      err= TTCDriverIOSetOutputVaule(&ioTestHandle,
                                     IOID_9,!TTCDriverIOGetOutputValue(IOID_9));
      //表示操作成功
      if(err == MANAGER_INFO_CONTROL_IO_SUCCESS){
      }
  }
}
```

4.2 UART 说明

CC2640 的 UART 具有如下特点：

1、具备可编程的波特率发生器，最高速率高达 3 Mbps。
2、具备独立的 32×8 发送（TX）和 32×12 接收（RX） FIFO 缓冲器，可以减少 CPU 的中断处理动作。
3、具备开始、停止和奇偶校验的标准异步通信位。
4、支持 CTS 和 RTS 功能。
5、使用 uDMA 传输数据。
6、具备可编程的硬件流控制。

4.2.1 UART API 说明

4.2.1.1 TTCDriverUartEvent()

```
/*****************************************************************************
【函    数】  TTCDriverUartEvent(void)
【概    述】  UART 事件处理
【入口参数】  无
【返回参数】  无
【说    明】  无
*****************************************************************************/
extern void TTCDriverUartEvent(void);
```

注意：此 API 是与 SDK 相关的一个函数，必须由用户添加到应用程序的线程中，否则无法传输 UART 数据。

4.2.1.2 TTCDriverUartInitDefaultParam()

```
/*****************************************************************************
【函    数】  void TTCDriverUartInitDefaultParam(Uart_Handle * uartHandle)
【概    述】  设置 UART 默认参数
【入口参数】  uartHandle : UART 句柄
【返回参数】  无
【说    明】  无
*****************************************************************************/
extern void TTCDriverUartInitDefaultParam(Uart_Handle * uartHandle);
```

说明：初始化 UART 句柄，在进行 UART 初始化操作前应该先调用此 API 将其初始化一下。

4.2.1.3 TTCDriverUartInit()

```
/*****************************************************************************
【函    数】  TTCDriverUartInit(TTCSdkClass_t *appCallbacks,
                                TTCSdkUartCB_t uartCB,
                                Uart_Handle * uartHandle,
                                TTCDriverUartParams_t param)
【概    述】  初始化 UART
【入口参数】  appCallbacks : 注册回调
              uartCB       : 串口回调注册
              uartHandle   : UART 句柄
              param        : UART 参数
【返回参数】  初始化失败原因请参考 TTCDriverInfo_t
【说    明】  若初始化失败 UART 会返回初始化失败原因。
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverUartInit(TTCSdkClass_t *appCallbacks,
                                         TTCSdkUartCB_t uartCB,
                                         Uart_Handle * uartHandle,
                                         TTCDriverUartParams_t param);
```

说明：初始化 UART,注意这里的回调函数，实际应用时不要在回调函数里发送数据。

4.2.1.4 TTCDriverUartClose()

```
/*****************************************************************************
【函    数】  TTCDriverUartClose(Uart_Handle * uartHandle)
【概    述】  关闭 UART
【入口参数】  uartHandle : UART 句柄
【返回参数】  请参考 TTCDriverInfo_t
【说 明】  若初始化失败 UART 会返回初始化失败原因。
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverUartClose(Uart_Handle * uartHandle);
```

说明：关闭 UART。

4.2.1.5 TTCDriverUartWrite()

```
/*****************************************************************************
【函    数】  TTCDriverUartWrite(Uart_Handle * uartHandle,
                                u8 *buffer,
                                u16 len)
```

```
【概    述】  向 UART TX 缓存写入数据并发送
【入口参数】  uartHandle : UART 句柄
              buffer     : 数据缓冲区
              size       : 数据长度
【返回参数】  发送失败原因请参考 TTCDriverInfo_t
【说    明】  如果当前 Wakeup 引脚处于拉高状态
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverUartWrite(Uart_Handle * uartHandle,
                                          u8 *buffer,
                                          u16 len);
```

说明：发送数据。

4.2.2 UART 使用示例

```
/*******************************************************************************
【文    件】  TTCDriverUARTDemo.c
【概    述】  TTC SDK UART 示例代码
【编    写】  SDK 工作小组
【修    订】  SDK 工作小组
【修订日期】  2016/12/02
【版    本】  V1.0.0
【说    明】
```

功能说明：配置了一个 UART，开启之后会周期性的打印"TTCDriverUART Test\r\n"；
若接收到"TTCDriverUART Test\r\n",则会翻转 IOID_9 脚的输出状态。

添加的步骤：

1、添加对应的头文件 TTCDriverUARTDemo.h

2、在工程中的 Opions/C/C++ Compiler/Defined symbols 中定义 TTCDRIVER_UART

3、在 TTCBlePeripheralTaskFxn()函数里添加

```
#ifdef TTCDRIVER_UART
    TTCDriverUartEvent();                    //SDK 的
    TTCSDKDriverUARTEvent();                 //用户的
#endif //TTCDRIVER_UART
```

4、在主线程里的 static void TTCSDKDriverInit(void)函数中添加如下代码。

```
#ifdef TTCDRIVER_UART
    TTCDriverDemoUARTInit(&sem,&appMsgQueue);
#endif
*******************************************************************************/
#ifdef  TTCDRIVER_UART
/*******************************************************************
 * 头文件
```

```
 */
#include <string.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Queue.h>
#include <ti/drivers/PIN.h>
#include <ti/drivers/pin/PINCC26XX.h>
#include <ti/drivers/UART.h>
#include <ti/drivers/uart/UARTCC26XX.h>
#include "TTCSDKBoard.h"
#include "TTCBleSDKConfig.h"
#include "TTCBleSDKManager.h"
#include "TTCDriverUART.h"
#include "TTCDriverUARTDemo.h"

/*********************************************************************
 * 常量及宏定义
 */
#define TTCBLE_SDK_UART_EVN                   0x0001


/*********************************************************************
 * 本地变量
 */
static TTCDriverInfo_t UartErrCode;

Uart_Handle uartHandle;
Clock_Struct uClock;

PIN_Handle uLedHandle;
PIN_State  uLedState;

PIN_Config uLedConfig[] = {
    IOID_9 | PIN_GPIO_OUTPUT_EN  | PIN_INPUT_DIS  |  PIN_GPIO_HIGH,
    PIN_TERMINATE
};

static ICall_Semaphore * UartSem;                    //线程信号量，用于唤醒线程
static Queue_Handle * UartMsgQueue;                  //消息句柄
static u16 events;                                   //本地事件
```

```
/*********************************************************************
 * 驱动硬件属性表声明 注意请勿删除
 */
extern const UARTCC26XX_HWAttrs uartCC26XXHWAttrs[];


/*********************************************************************
 * 本地函数声明
 */
static void TestUartCB(TTCDriverUartState_t uartState,u8 * buffer,u16 len);
static void TTCSDKDriverUARTSetEvent(UArg arg);
static void TTCDriverDemoUARTClockHandler(UArg arg);


/*****************************************************************************
【函    数】  TTCDriverDemoUARTClockHandler(UArg arg)
【概    述】  线程置事件
【入口参数】  arg    : 标记事件源事件
【返回参数】  无
【说    明】
*****************************************************************************/
static void TTCDriverDemoUARTClockHandler(UArg arg){
    TTCSDKDriverUARTSetEvent(arg);
}

/*****************************************************************************
【函    数】  TTCSDKDriverUARTSetEvent(UArg arg)
【概    述】  标记事件并唤醒线程
【入口参数】  arg : 标记事件
【返回参数】  无
【说    明】  无
*****************************************************************************/
static void TTCSDKDriverUARTSetEvent(UArg arg){
    events |= arg;
    Semaphore_post(*UartSem);
}

/*****************************************************************************
【函    数】  TestUartCB(TTCDriverUartState_t uartState,u8 * buffer,u16 len)
【概    述】  UART 回调函数
【入口参数】  uartState:UART 的状态
              Buffer:数据存储地址
```

```
                Len:数据的长度
【返回参数】   无
【说    明】   无
*******************************************************************************/
static void TestUartCB(TTCDriverUartState_t uartState,u8 * buffer,u16 len){

    if(uartState == TTCDRIVER_UART_RX_DATA_STATE){                //接收
        if(memcmp(buffer,"TTCDriverUART Test\r\n",len) == 0){
              TTCDriverIOSetOutputVaule(&uLedHandle,
                                        IOID_9,
                                        ~TTCDriverIOGetOutputValue(IOID_9));

         //注意这里不能直接调用发送函数,应该通过置事件或者发消息的方式来进行数据的回应等动作
        }
    }else if(uartState == TTCDRIVER_UART_TX_DONE_STATE){         //发送完毕

    }
}

/*******************************************************************************
【函    数】   TTCSDKDriverUARTEvent(void)
【概    述】   Demo UART 事件处理
【入口参数】   无
【返回参数】   无
【说    明】   处理本地的事件
*******************************************************************************/
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"));

        Util_startClock(&uClock);
    }
}

/*******************************************************************************
【函    数】   TTCDriverDemoUARTInit(TTCSdkClass_t *appCallbacks,
                                    ICall_Semaphore * sem,
                                    Queue_Handle * appMsgQueue)
【概    述】   UART 示例初始化化
```

```
【入口参数】  appCallbacks：注册回调
              sem          ：信号量
              appMsgQueue  ：消息句柄
【返回参数】  无
【说    明】  无
*******************************************************************************/
void TTCDriverDemoUARTInit(TTCSdkClass_t *appCallbacks,
                           ICall_Semaphore * sem,
                           Queue_Handle * appMsgQueue){
    if(appCallbacks == NULL || sem == NULL || appMsgQueue == NULL){
       return;
    }
    UartSem      = sem;
    UartMsgQueue = appMsgQueue;

    TTCDriverUartInitDefaultParam(&uartHandle);
    uartHandle.sem         = UartSem;
    uartHandle.queueHandle = UartMsgQueue;
    const TTCDriverUartParams_t uartParam = {
        .uartName     = CC2650_UART0,
        .baudRate     = 115200,
        .dataLength   = UART_LEN_8,
        .stopBits     = UART_STOP_ONE,
        .parityType   = UART_PAR_NONE,
        .wakeUpPin    = Board_UART_WAKEUP,
        .intPin       = Board_UART_INT,
        .uartRxBufLen = 100,
        .uartTxBufLen = 100,
        .uartHWAttr   = &uartCC26XXHWAttrs[CC2650_UART0],
    };

    UartErrCode = TTCDriverUartInit(appCallbacks,
                                    &TestUartCB,
                                    &uartHandle,
                                     uartParam);

     UartErrCode=TTCDriverUartWrite(&uartHandle,
                                    "TTCDriverUART Test\r\n",
                                     strlen("TTCDriverUART Test\r\n"));

     if(UartErrCode != TTCDRIVER_TX_DATA_SUCCESS){
       return ;
```

```
    }

  //当测试定时器的输入捕获时   这里需要设置为假 即不打开这个 GPIO 以防出现 GPIO 冲突
    #if 0
      TTCBleSDKManagerInfo_t Err;
      Err = TTCDriverIOOpen(&uLedHandle,&uLedState,(const PIN_Config *)uLedConfig);
    #endif

    #if 1   //当测试其他驱动时   这里需要设置为假 保证打印的效果

      Util_constructClock(&uClock,          //周期性的打印 "TTCDriverUART Test\r\n"
                          TTCDriverDemoUARTClockHandler,
                          1000,
                          0,
                          true,
                          TTCBLE_SDK_UART_EVN);
    #endif
}

/*****************************************************************************
【函    数】  u8 TTCDriverVal2Str(uint32 val,uint8 * buf)
【概    述】  将数字转换为字符串
【入口参数】  val : 待转换的数字
              buf : 转换的结果存储区
【返回参数】  字符串的长度
【说    明】  无
*****************************************************************************/
u8 TTCDriverVal2Str(u32 val,u8 * buf){
    u8  len = 0,i=0;
    u32 nr = 0;
    if(buf != NULL){
        nr = val;
        do{                   //计算数据的长度
            len++;
            nr /= 10;
        }while(nr);

        for(i=0;i<len;i++){
            buf[len-i-1]  = val%10 + 0x30;
            val /= 10;
        }
    }
```

```
    return len;
}

#endif //TTCDRIVER_UART
```

4.3 定时器说明

CC2640 拥有 8 个 16 bit 的定时器，每个定时器都可单独配置成不同的模式使用。支持可编程的计数方式，另外它支持同时启动 1 个以上的定时器，适用一些特殊的使用场合。

4.3.1 定时器 API 说明

4.3.1.1 TTCDriverTimerInit()

```
/*****************************************************************************
【函    数】  TTCDriverTimerInit(TTCDriverTimerInit_t timerInitParam,
                                 Timer_Handle * timerHandle)
【概    述】  定时器初始化
【入口参数】  appCallbacks   : 注册回调
              timerInitParam : 定时器初始化参数
              timerHandle    : 定时器句柄
【返回参数】  请参考附录 A TTCDriverInfo_t
【说    明】  若 Timer_Handle 未初始化则不可以使用
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerInit(TTCDriverTimerInit_t timerInitParam,
                                          Timer_Handle * timerHandle);
```

说明：定时器初始化。

4.3.1.2 TCDriverTimerRegisterIntCallBack()

```
/*****************************************************************************
【函    数】  TCDriverTimerRegisterIntCallBack(Timer_Handle * timerHandle,
                                               TTCSdkTimerCB_t pCB)
【概    述】  定时器设置中断函数回调
【入口参数】  timerHandle : 定时器句柄
              pCB         : 设置定时器中断回调函数
【返回参数】  请参考 TTCDriverInfo_t
【说    明】  若 Timer_Handle 未初始化则不可以使用
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerRegisterIntCallBack(Timer_Handle * timerHandle,
                                                         TTCSdkTimerCB_t pCB);
```

说明：注册各种中断回调函数，比如定时中断回调函数，输入计数中断回调函数，输入捕获中断回调函数等。

4.3.1.3 TTCDriverTimerCounterSetParam()

```
/*****************************************************************************
【函    数】  TTCDriverTimerCounterSetParam(Timer_Handle * timerHandle,
                                            TTCDriverTimerCounterParam_t param,
                                            bool enable)
【概    述】  定时器计数参数设置
【入口参数】  timerHandle : 定时器句柄
              param       : 计数参数
              enable      : 是否使能
【返回参数】  请参考 TTCDriverInfo_t
【说    明】  若 Timer_Handle 未初始化则不可以使用
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerCounterSetParam(Timer_Handle * timerHandle,
                                                     TTCDriverTimerCounterParam_t param,
                                                     bool enable);
```

说明：定时器的定时功能配置，包括定时的周期等。

4.3.1.4 TTCDriverTimerEdgCountSetParam()

```
/*****************************************************************************
【函    数】  TTCDriverTimerEdgCountSetParam(Timer_Handle * timerHandle,
                                             TTCDriverTimerEdgeCountParam_t param,
                                             PIN_Id edgeCountPin,
                                             bool enable)
【概    述】  定时器边沿计数模式参数设置
【入口参数】  timerHandle  : 定时器句柄
              param        : 计数参数
              edgeCountPin : 输入捕获引脚
              enable       : 是否使能
【返回参数】  请参考 TTCDriverInfo_t
【说    明】  若 Timer_Handle 未初始化则不可以使用
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerEdgCountSetParam(Timer_Handle * timerHandle,
                                                      TTCDriverTimerEdgeCountParam_t param,
                                                      PIN_Id edgeCountPin,
                                                      bool enable);
```

说明：定时器的外部输入计数功能配置，包括计数的个数(即计多少个数就产生中断)，溢出值等。

4.3.1.5 TTCDriverTimerEdgTimingSetParam()

```
/*****************************************************************************
【函    数】 TTCDriverTimerEdgTimingSetParam(Timer_Handle * timerHandle,
                                            TTCDriverTimerEdgeTimingParam_t param,
                                            PIN_Id edgeTimerPin,
                                            bool enable)
【概    述】 定时器边沿定时模式参数设置
【入口参数】 timerHandle  : 定时器句柄
             param        : 边沿定时参数
             edgeTimerPin : 输入捕获引脚
             enable       : 是否使能
【返回参数】 请参考 TTCDriverInfo_t
【说    明】 若 Timer_Handle 未初始化则不可以使用
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerEdgTimingSetParam(Timer_Handle * timerHandle,
                                                       TTCDriverTimerEdgeTimingParam_t param,
                                                       PIN_Id edgeTimerPin,
                                                       bool enable);
```

说明：定时器的输入捕获功能配置，包括捕获的边沿、计数的方向等。

4.3.1.6 TTCDriverTimerPwmSetParam()

```
/*****************************************************************************
【函    数】 TTCDriverTimerPwmSetParam(Timer_Handle * timerHandle,
                                       TTCDriverTimerPwmParams_t param,
                                       PIN_Id pwmPin,
                                       bool enable)
【概    述】 定时器 PWM 模式参数设置
【入口参数】 timerHandle : 定时器句柄
             param       : 边沿定时参数
             pwmPin      : PWM 输出引脚
             enable      : 是否使能
【返回参数】 请参考 TTCDriverInfo_t
【说    明】 若 Timer_Handle 未初始化则不可以使用。
             pwmPin 若不使用需要设置成 PIN_UNASSIGNED。
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerPwmSetParam(Timer_Handle * timerHandle,
                                                 TTCDriverTimerPwmParams_t param,
                                                 PIN_Id pwmPin,
```

bool enable);

说明：定时器的 PWM 配置，包括周期和占空比，还有有效电平的极性。

4.3.1.7 TTCDriverTimerStop()

```
/*****************************************************************************
【函    数】  TTCDriverTimerStop(Timer_Handle * timerHandle)
【概    述】  定时器停止
【入口参数】  timerHandle : 定时器句柄
【返回参数】  请参考 TTCDriverInfo_t
【说    明】  1.若 Timer_Handle 未初始化则不可以使用
              2.本函数不会清除相关配置，当要开启时可以直接调用 TTCDriverTimerStart 再次运行
              3.如果定时器为 PWM 模式，那么调用本函数则会让 PWM 输出引脚输出所设置的闲置电平
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerStop(Timer_Handle * timerHandle);
```

说明：暂时关闭一个定时器。

4.3.1.8 TTCDriverTimerStart()

```
/*****************************************************************************
【函    数】  TTCDriverTimerStart(Timer_Handle * timerHandle)
【概    述】  定时器启动
【入口参数】  timerHandle : 定时器句柄
【返回参数】  请参考 TTCDriverInfo_t
【说    明】  若 Timer_Handle 未初始化则不可以使用
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerStart(Timer_Handle * timerHandle);
```

说明：开启一个定时器。

4.3.1.9 TTCDriverTimerClose()

```
/*****************************************************************************
【函    数】  TTCDriverTimerClose(Timer_Handle * timerHandle)
【概    述】  定时器关闭
【入口参数】  timerHandle : 定时器句柄
【返回参数】  请参考 TTCDriverInfo_t
【说    明】  1.若 Timer_Handle 未初始化则不可以使用
```

2. 调用本函数后，如果要再运行定时器，则需要再次调用设置函数。调用 TTCDriverTimerStart 无效

```
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerClose(Timer_Handle * timerHandle);
```

说明：直接关闭一个定时器，若要再次开启则需要重新配置。

4.3.1.10 TTCDriverTimerSync()

```
/*****************************************************************************
【函    数】 TTCDriverTimerSync
【概    述】 选择同步启动的定时器
【入口参数】 timerHandle : 定时器句柄
             syncTimer   : 选择需要同步启动的定时器
                           CC2650_TIMER0_A_SYNC
                           CC2650_TIMER0_B_SYNC
                           CC2650_TIMER1_A_SYNC
                           CC2650_TIMER1_B_SYNC
                           CC2650_TIMER2_A_SYNC
                           CC2650_TIMER2_B_SYNC
                           CC2650_TIMER3_A_SYNC
                           CC2650_TIMER3_B_SYNC
【返回参数】 无
【说    明】 请在相关定时器启动后调用该函数。
*****************************************************************************/
extern void TTCDriverTimerSync(Timer_Handle * timerHandle,CC2650_TimerSync syncTimer);
```

说明：用于同步启动多个定时器，比如两个定时器都用于 PWM 输出，让其一同开启，保证输出信号同时输出。

4.3.2 定时器 使用示例

```
/*****************************************************************************
【文    件】  TTCDirverTimerDemo.c
【概    述】  TTC SDK Timer 示例代码
【编    写】  SDK 工作小组
【修    订】  SDK 工作小组
【修订日期】  2016/12/01
【版    本】  V1.0.0
【说    明】
```

功能说明：

1、配置任意一个定时器输出一路 PWM 信号，并让这路 PWM 在 Board_PWM0-Board_PWM7 脚之间来回切换输出。

2、为任意一个定时器配置定时中断功能，在中断回调函数里翻转 Board_PWM1 脚的输出状态

3、配置定时器 CC2650_TIMER0_A 从 Board_PWM0 脚输出 PWM 信号；

配置指定的定时器为输入计数模式，信号从 Board_PWM2 脚输入，在计数中断回调函数里翻转 Board_PWM3 脚的输出状态。

4、配置定时器 CC2650_TIMER0_A 从 Board_PWM0 脚输出 PWM 信号；

配置指定的 定时器 TimerName 为输入捕获模式，信号从 Board_PWM2 脚输入，在捕获中断回调函数里翻转 Board_PWM7，同时若是开启了 UART 驱动，则会通过 UART 打印两次捕获的时间差（此差值就是输入信号两个边沿之间时间）.

添加的步骤：

1、添加对应的头文件 TTCDirverTimerDemo.h

2、在工程中的 Opions/C/C++ Compiler/Defined symbols 中定义 TTCDRIVER_TIMER。

3、在 TTCBlePeripheralTaskFxn()函数里添加

```
#ifdef TTCDRIVER_TIMER
    TTCSDKDriverDemoTimerEvent();
#endif
```

4、在主线程里的 static void TTCSDKDriverInit(void)函数中添加如下代码。

注意：每次只能测试一种情况

```
#ifdef TTCDRIVER_TIMER
    TTCDriverDemoTimerInit(&sem,&appMsgQueue,CC2650_TIMER0_A);
#endif
```

注意:当想使用 "透传开发套件" 调试定时器时,请将套件上除了 3.3V 之外的所有跳帽都拔掉，另外，如果是用逻辑分析仪来抓波形的话，请适当设置抓取的频率。

```
*****************************************************************************/
#ifdef TTCDRIVER_TIMER
/*********************************************************************
 * 头文件
 */
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Queue.h>
#include <ti/drivers/PIN.h>
#include <ti/drivers/pin/PINCC26XX.h>
#include "TTCSDKBoard.h"
#include "TTCBleSDKConfig.h"
#include "TTCBleSDKManager.h"
#ifdef TTCDRIVER_UART
#include <ti/drivers/UART.h>
#include <ti/drivers/uart/UARTCC26XX.h>
```

```
#include "TTCDriverUART.h"
#include "TTCDriverUARTDemo.h"
#endif //TTCDRIVER_UART
#include "TTCDriverTimer.h"
#include "TTCDriverTimerDemo.h"
/*********************************************************************
 * 常量及宏定义
 */
#define TTCBLE_SDK_TIME_EVN                    0x0001


/*********************************************************************
 * 本地变量
 */
static Timer_Handle  Handle[4][2] = { 0 };       //8 个句柄
static TTCDriverInfo_t TimErrCode;
Clock_Struct TimerClock;
Clock_Struct CatTimerClock;

PIN_Handle EdGeHandle;
PIN_State  EdGeState;

PIN_Config EdGeConfig[] = {

  Board_PWM1 | PIN_GPIO_OUTPUT_EN  | PIN_INPUT_DIS  |  PIN_GPIO_HIGH,
  Board_PWM3 | PIN_GPIO_OUTPUT_EN  | PIN_INPUT_DIS  |  PIN_GPIO_HIGH,
  Board_PWM7 | PIN_GPIO_OUTPUT_EN  | PIN_INPUT_DIS  |  PIN_GPIO_HIGH,
  PIN_TERMINATE
};

u32 crt[2] = {0};

static ICall_Semaphore * TimSem;                                        //线程信号量,
用于唤醒线程
static Queue_Handle * TimMsgQueue;                                       //消息句柄
static u16 events;                                                       //本地事件

/*********************************************************************
 * 本地函数声明
 */
static void TTCDriverDemoTimerClockHandler(UArg arg);
static void TTCDriverDemoTimerSetEvent(UArg arg);
```

```
static TTCDriverInfo_t TTCSDKDriver_PWMInit(CC2650_TimerName TimerName,uint8 brd);
static TTCDriverInfo_t TTCSDKDriver_Inter_Init(CC2650_TimerName TimerName);
static TTCDriverInfo_t TTCSDKDriver_Cat_Init(CC2650_TimerName TimerName,uint8 brd);
static TTCDriverInfo_t TTCSDKDriver_Count_Init(CC2650_TimerName TimerName,uint8 brd);
Static TTCDriverInfo_tTCSDKDriver_Callback_Register(CC2650_TimerName TimerName,
                                                    TTCSdkTimerCB_t func);
static TTCBleSDKManagerInfo_t TTCSDKDriver_Inst_IO_Init(PIN_Config * config);
static void TTCDriveTimerFuncIsrCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue);
static void TTCDriveTimerFuncCountCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue);
static void TTCDriveTimerFuncCaptureCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue);
static void TTCDriverSinglePWMSignalTestOther(CC2650_TimerName TimerName);
static void TTCDriverSinglePWMSignalTest(ICall_Semaphore *,Queue_Handle *,CC2650_TimerName );
static void TTCDriverSinglePWMSignalDispose(CC2650_TimerName TimerName);
Static void TTCDriverSingleInterSignalTest(ICall_Semaphore * ,
                                           Queue_Handle * ,
                                           CC2650_TimerName );
static void TTCDriverSingleCountSignalTest(ICall_Semaphore * ,
                                           Queue_Handle * ,
                                           CC2650_TimerName );
static void TTCDriverSingleCatSignalTest(ICall_Semaphore *,Queue_Handle *,CC2650_TimerName );

/*****************************************************************************
【函    数】  TTCDriverDemoTimerClockHandler(UArg arg)
【概    述】  线程置事件
【入口参数】  arg    : 标记事件源事件
【返回参数】  无
【说    明】
*****************************************************************************/
static void TTCDriverDemoTimerClockHandler(UArg arg){
    TTCSDKDriverTIMSetEvent(arg);
}
/*****************************************************************************
【函    数】  TTCDriverDemoTimerSetEvent(UArg arg)
【概    述】
【入口参数】  arg：事件标志
【返回参数】  无
【说    明】  无
*****************************************************************************/
static void TTCDriverDemoTimerSetEvent(UArg arg){
    events |= arg;
    Semaphore_post(*TimSem);
}
```

```
/*******************************************************************************
【函    数】  TTCDriverDemoTimerInit(ICall_Semaphore * sem,
                                    Queue_Handle * appMsgQueue,
                                    CC2650_TimerName TimerName)
【概    述】  指定定时器初始化
【入口参数】  sem:主线程的信号量
              appMsgQueue:主线程的队列
              TimerName:定时器名称
【返回参数】  参见附录 A 里的 TTCDriverInfo_t
【说    明】
*******************************************************************************/
static TTCDriverInfo_t TTCDriverDemoTimerInit(ICall_Semaphore * sem,
                                              Queue_Handle * appMsgQueue,
                                              CC2650_TimerName TimerName){
    uint8 i = 0,j = 0;
    if(sem != NULL && appMsgQueue != NULL){
        TTCDriverTimerInit_t timerInitParam = {
            .sem          = sem,
            .queueHandle  = appMsgQueue,
            .timerName    = (CC2650_TimerName)(TimerName),
        };
        i = TimerName/2;
        j = TimerName%2;

        TimErrCode = TTCDriverTimerInit(timerInitParam,&Handle[i][j]);

        if(TimErrCode != TTCDRIVER_INIT_SUCCESS){          //初始化不成功
            return TimErrCode;
        }
    }else{                                                 //输入参数 不正确
        return TTCDRIVER_INIT_SYSPARA_FAILED;
    }
    return TTCDRIVER_INIT_SUCCESS;
}
/*******************************************************************************
【函    数】  TTCSDKDriver_PWMInit(CC2650_TimerName TimerName,uint8 brd)
【概    述】  指定定时器的 PWM 输出初始化
【入口参数】  nTmr ：  定时器编号
              brd ：   输出的引脚编号
【返回参数】  参见附录 A 里的 TTCDriverInfo_t
【说    明】  注意：本函数应该在 TTCSDKDriver_TIM_Init() 之后执行
*******************************************************************************/
```

```
static TTCDriverInfo_t TTCSDKDriver_PWMInit(CC2650_TimerName TimerName,uint8 brd){

    uint8 i=0,j=0;
    TTCDriverTimerPwmParams_t param = {
        48000,                              //48000000/48000 = 1000Hz
        12000,
        PWM_OUTPUT_INVERTED,
        false,
        1                                   //空闲电平
    };
    i = TimerName/2;
    j = TimerName%2;

    TimErrCode = TTCDriverTimerPwmSetParam(&Handle[i][j],      //定时器的句柄
                                           param,              //PWM 的参数
                                           brd,                //指定 PWM 信号的输出脚
                                           false);             //立即使能定时器 false  true

    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){                  //PWM 初始化不成功
        return TimErrCode;
    }
    TTCDriverTimerStart(&Handle[i][j]);
    return TTCDRIVER_INIT_SUCCESS;
}

/*******************************************************************************
【函    数】  TTCSDKDriver_Inter_Init(CC2650_TimerName TimerName)
【概    述】  配置指定定时器的中断功能
【入口参数】  TimerName：定时器编号
【返回参数】  参见附录 A 里的 TTCDriverInfo_t
【说    明】  注意：本函数应该在 TTCSDKDriver_TIM_Init() 之后执行
*******************************************************************************/
static TTCDriverInfo_t TTCSDKDriver_Inter_Init(CC2650_TimerName TimerName){
    uint8 i,j;
    TTCDriverTimerCounterParam_t counterParam = {

        .counterMode = CC2650_TIMER_PERIODIC_MODE,    //周期性的
        .countValue  = 24000,                         //48000000 /  (24000*2)  = 1000Hz
        .upCountMode = false,                         //
        .prescaler   = 1                              //
    };
```

```
    i = TimerName/2;
    j = TimerName%2;

    TimErrCode = TTCDriverTimerCounterSetParam(&Handle[i][j],
                                               counterParam,
                                               true);        //直接开启
    return TimErrCode;
}

/*****************************************************************************
【函    数】  TTCSDKDriver_Cat_Init(CC2650_TimerName TimerName,uint8 brd)
【概    述】  指定定时器的输入捕获初始化
【入口参数】  nTmr ：  定时器编号
              brd ：   输出的引脚编号
【返回参数】  参见附录 A 里的 TTCDriverInfo_t
【说    明】  注意：本函数应该在 TTCSDKDriver_TIM_Init() 之后执行
*****************************************************************************/
static TTCDriverInfo_t TTCSDKDriver_Cat_Init(CC2650_TimerName TimerName,uint8 brd){
    uint8 i,j;

    TTCDriverTimerEdgeTimingParam_t param = {
        .countValue                 = 65535,
        .upCountMode                = 1,                        //递增计数
        .edgeMode                   = TIMER_POSITIVE_EDGE,      //上升沿捕获
    };

    i = TimerName/2;
    j = TimerName%2;

    TimErrCode = TTCDriverTimerEdgTimingSetParam(&Handle[i][j],
                                                 param,
                                                 brd,          //设置输入口
                                                 true);
    return TimErrCode;
}
/*****************************************************************************
【函    数】  TTCSDKDriver_Count_Init(CC2650_TimerName TimerName,uint8 brd)
【概    述】  指定定时器的输入计数初始化
【入口参数】  nTmr ：  定时器编号
              brd ：   输出的引脚编号
【返回参数】  参见附录 A 里的 TTCDriverInfo_t
【说    明】  注意：本函数应该在 TTCSDKDriver_TIM_Init() 之后执行
```

```
*****************************************************************************/
static TTCDriverInfo_t TTCSDKDriver_Count_Init(CC2650_TimerName TimerName,uint8 brd){
    uint8 i,j;
    TTCDriverTimerEdgeCountParam_t EdgeCountParam = {
        .countValue         = 1000,
        .upCountMode        = TRUE,                                    //向上计数
        .matchValue         = 10,                                      //10 个
        .edgeMode           = TIMER_POSITIVE_EDGE,                     //上升沿
    };

    i = TimerName/2;
    j = TimerName%2;

    TimErrCode = TTCDriverTimerEdgCountSetParam(&Handle[i][j],
                                                EdgeCountParam,
                                                brd,                   //
                                                true);
    return TimErrCode;
}

/*****************************************************************************
【函    数】  TTCSDKDriver_Callback_Register(CC2650_TimerName TimerName,TTCSdkTimerCB_t func)
【概    述】  注册指定定时器的回调函数
【入口参数】  TimerName：定时器编号
              Func：回调函数
【返回参数】  参见附录 A 里的 TTCDriverInfo_t
【说    明】  注意：本函数应该在 TTCSDKDriver_TIM_Init() 之后执行
*****************************************************************************/
static TTCDriverInfo_t TTCSDKDriver_Callback_Register(CC2650_TimerName TimerName,
                                                      TTCSdkTimerCB_t func){

    uint8 i,j;
    if(func != NULL){
        i = TimerName/2;
        j = TimerName%2;
        TimErrCode = TTCDriverTimerRegisterIntCallBack(&Handle[i][j],func);
        if(TimErrCode != TTCDRIVER_INIT_SUCCESS){                     //注册失败
            return TimErrCode;
        }
    }else{
        return TTCDRIVER_INIT_PARA_ERROR;
    }
```

```
    return TTCDRIVER_INIT_SUCCESS;
}

/*****************************************************************************
【函    数】  TTCSDKDriver_Inst_IO_Init(PIN_Config * config)
【概    述】  中断指示 IO 初始化
【入口参数】  config：IO 口的配置信息
【返回参数】  参见附录 A 里的 TTCBleSDKManagerInfo_t
【说    明】  若 config 为空，则按默认的设置
*****************************************************************************/
static TTCBleSDKManagerInfo_t TTCSDKDriver_Inst_IO_Init(PIN_Config * config){
    TTCBleSDKManagerInfo_t iErr;
    if(config != NULL){
        iErr = TTCDriverIOOpen(&EdGeHandle,&EdGeState,(const PIN_Config *)config);
    }else{
        iErr = TTCDriverIOOpen(&EdGeHandle,&EdGeState,(const PIN_Config *)EdGeConfig);
    }
    return iErr; //返回错误代码
}

/*****************************************************************************
【函    数】  TTCDriveTimerFuncIsrCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue)
【概    述】  定时器定时中断回调处理函数
【入口参数】  isrInfo :
              timerCurrectValue:
【返回参数】  无
【说    明】  无
*****************************************************************************/
static void TTCDriveTimerFuncIsrCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue){
    TTCDriverIOSetOutputVaule(&EdGeHandle,
                              Board_PWM1,
                              !TTCDriverIOGetOutputValue(Board_PWM1));
}

/*****************************************************************************
【函    数】  TTCDriveTimerFuncCountCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue)
【概    述】  定时器计数回调处理函数
【入口参数】  isrInfo :
              timerCurrectValue:
【返回参数】  无
【说    明】  无
*****************************************************************************/
```

```
static void TTCDriveTimerFuncCountCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue){

TTCDriverIOSetOutputVaule(&EdGeHandle,Board_PWM3,~TTCDriverIOGetOutputValue(Board_PWM3));
#ifdef TTCDRIVER_UART
    u32 se;
    se = timerCurrectValue;
    uint8 buf[20] = {0};
    TTCDriverUartWrite(&uartHandle,buf,TTCDriverVal2Str((u32)se,buf));   //打印计数值
#endif

}

/*****************************************************************************
【函    数】TTCDriveTimerFuncCaptureCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue)
【概    述】  定时器输入捕获回调处理函数
【入口参数】  isrInfo :
              timerCurrectValue:
【返回参数】  无
【说    明】  这里的打印功能不支持太高的输入波形的情况，256000bps 情况下 100K 是测试的上限
*****************************************************************************/
static void TTCDriveTimerFuncCaptureCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue){
    static u8 cnt = 0;
    u32 ern = timerCurrectValue;
    TTCDriverIOSetOutputVaule(&EdGeHandle,
                              Board_PWM7,
                              !TTCDriverIOGetOutputValue(Board_PWM7));
    if(cnt == 0){
        crt[0] = ern;
        crt[1] = 0;
    }else if(cnt == 1){
        crt[1] = ern;
        s32 deta = crt[1] - crt[0];
        if(deta > 0){             //规避掉一些特殊的时间段    测试的时候 就可以这样处理一下
          #ifdef TTCDRIVER_UART
            uint8 buf[20] = {0};
            TTCDriverUartWrite(&uartHandle,buf,TTCDriverVal2Str((u32)deta,buf));
          #endif
        }
    }

    cnt++;
    if(cnt == 2){
```

```
        cnt = 0;
    }
}

/*********************************************************************************
【函    数】  void TTCDriverSinglePWMSignalTestOther(CC2650_TimerName TimerName)
【概    述】  输出一路 PWM 信号，之后定时关闭和开启，20 次之后开始周期性的改变占空比
【入口参数】  TimerName：定时器编号
【返回参数】  无
【说    明】   关于 PWM 的其他功能测试，包括空闲电平的控制等
*********************************************************************************/
static void TTCDriverSinglePWMSignalTestOther(CC2650_TimerName TimerName){
    static uint8 Ct = 0,eg = 0;
    uint8 i,j;
    i = TimerName/2;
    j = TimerName%2;
    Ct++;
    if(Ct<20){                                          //开关测试
        if(eg == 0){
            eg = 1;
            TTCDriverTimerStop(&Handle[i][j]);
        }else{
            eg = 0;
            TTCDriverTimerStart(&Handle[i][j]);
        }
    }else if(Ct < 100){
        if(Ct < 60){
            TTCDriverTimerPwmParams_t param = {
                48000,                                  //48000000/48000 = 1000Hz
                12000,                                  //每次更改一下占空比
                PWM_OUTPUT_NOT_INVERTED,
                false,
                0
            };
            TimErrCode = TTCDriverTimerPwmSetParam(&Handle[i][j],   //定时器的句柄
                                                   param,           //PWM 的参数
                                                   Board_PWM0,      //指定 PWM 信号的输出脚
                                                   true);           //立即使能定时器

        }else{
            TTCDriverTimerPwmParams_t param = {
                48000,                      //48000000/48000 = 1000Hz
```

```
                36000,                          //每次更改一下占空比
                PWM_OUTPUT_NOT_INVERTED,
                false,
                0
            };
            TimErrCode = TTCDriverTimerPwmSetParam(&Handle[i][j],  //定时器的句柄
                                                   param,          //PWM 的参数
                                                   Board_PWM0,     //指定输出脚
                                                   true);          //立即使能定时器
        }
        if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
            return;
        }
    }else{
      Ct = 0;
    }
    Util_startClock(&TimerClock);
}

/*********************************************************************************
【函    数】  void TTCDriverSinglePWMSignalTest(ICall_Semaphore * sem,
                                                 Queue_Handle * appMsgQueue,
                                                 CC2650_TimerName TimerName)
【概    述】  输出一路 PWM 信号，之后在 PWM0-PWM7 之间来回切换输出
【入口参数】  sem：主线程信号量
              appMsgQueue：主线程消息队列
             TimerName：定时器
【返回参数】  无
【说    明】  无
*********************************************************************************/
static void TTCDriverSinglePWMSignalTest(ICall_Semaphore * sem,
                                         Queue_Handle * appMsgQueue,
                                         CC2650_TimerName TimerName){
    if(sem == NULL || appMsgQueue == NULL){
        return;
    }
    TimSem      = sem;
    TimMsgQueue = appMsgQueue;
    TimErrCode = TTCSDKDriver_TIM_Init(TimSem,TimMsgQueue,TimerName);  //初始化 定时器
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
```

```
    TimErrCode = TTCSDKDriver_PWMInit(TimerName,IOID_0);//配置 PWM 功能  信号从 Board_PWM0 输出
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    Util_constructClock(&TimerClock,                    //配置一个软件定时器
                        TTCSDKDriverTIMclockHandler,
                        100,                            //定时 100mS
                        0,
                        true,                           //立即启动
                        TTCBLE_SDK_TIME_EVN);
}

/*******************************************************************************
【函    数】  TTCDriverSinglePWMSignalDispose(CC2650_TimerName TimerName)
【概    述】  实现将一路 PWM 定时在 PWM0-PWM7 之间来回切换输出
【入口参数】  TimerName：定时器编号
【返回参数】  无
【说    明】  测试时请检查 IOID_0 - IOID_7 处于可用状态(没被申请)，另外需要实际查看
              一下对应的脚有没有接什么外部的电路("透传套件"上的跳帽,还有下载线需要拔掉).
*******************************************************************************/
static void TTCDriverSinglePWMSignalDispose(CC2650_TimerName TimerName){
    static uint8 cntt = 0;
    uint8 i,j;
    cntt++;
    if(cntt == 8){
        cntt = 0;
    }
    TTCDriverTimerPwmParams_t param = {
        48000,  //48000000/48000 = 1000Hz
        24000,
        PWM_OUTPUT_NOT_INVERTED,
        false,
        0
    };
    i = TimerName/2;
    j = TimerName%2;
    TimErrCode = TTCDriverTimerPwmSetParam(&Handle[i][j],      //定时器的句柄
                                           param,              //PWM 的参数
                                           Board_PWM0+cntt,    //指定 PWM 信号的输出脚
                                           true);              //立即使能定时器

    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
```

```
        return;
    }
    Util_startClock(&TimerClock);
}


/*********************************************************************************
【函    数】  void TTCDriverSingleInterSignalTest(ICall_Semaphore * sem,
                                                  Queue_Handle * appMsgQueue,
                                                  CC2650_TimerName TimerName)
【概    述】  配置一个定时中断
【入口参数】  sem：主线程信号量
              appMsgQueue：主线程消息队列
             TimerName：定时器
【返回参数】  无
【说    明】  在定时中断回调函数里翻转 Board_PWM1
*********************************************************************************/
static void TTCDriverSingleInterSignalTest(ICall_Semaphore * sem,
                                           Queue_Handle * appMsgQueue,
                                           CC2650_TimerName TimerName){
    if(sem == NULL || appMsgQueue == NULL){
        return;
    }
    TimSem      = sem;
    TimMsgQueue = appMsgQueue;
    TimErrCode = TTCSDKDriver_TIM_Init(TimSem,TimMsgQueue,TimerName);   //初始化 定时器
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    //注册回掉函数
    TimErrCode = TTCSDKDriver_Callback_Register(TimerName,TTCDriveTimerFuncIsrCB);
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    TTCBleSDKManagerInfo_t ErrCode;
    ErrCode = TTCSDKDriver_Inst_IO_Init(NULL); //初始化 需要用到的 GPIO  NULL 表示用默认的 GPIO

    if(ErrCode != MANAGER_INFO_REQUEST_IO_SUCCESS){
        return;
    }
    TimErrCode = TTCSDKDriver_Inter_Init(TimerName);   //配置定时中断
```

```
}

/*********************************************************************************
【函    数】  void TTCDriverSingleCountSignalTest(ICall_Semaphore * sem,
                                                  Queue_Handle * appMsgQueue,
                                                  CC2650_TimerName TimerName)
【概    述】  配置一个外部输入计数中断
【入口参数】  sem：主线程信号量
              appMsgQueue：主线程消息队列
              TimerName：定时器
【返回参数】  无
【说    明】  本函数设置定时器 CC2650_TIMER0_A 从 Board_PWM0 脚输出 PWM 信号；
              设置指定的 定时器 TimerName 为输入计数模式，信号从 Board_PWM2 脚输入；
              在计数中断回调函数里翻转 Board_PWM3

          注意：实际看 Board_PWM3 的输出波形会发现，它的一个脉宽里包含 11 个上升沿
                这个不是说计数不准，只是因为我们的输出方式不太合理，实际的计数值
                我们可以在计数中断回调里获取到。
*********************************************************************************/
static void TTCDriverSingleCountSignalTest(ICall_Semaphore * sem,
                                           Queue_Handle * appMsgQueue,
                                           CC2650_TimerName TimerName){
    if(sem == NULL || appMsgQueue == NULL){
        return;
    }
    if(TimerName == CC2650_TIMER0_A){
        return;
    }
    TimSem      = sem;
    TimMsgQueue = appMsgQueue;
    TimErrCode = TTCSDKDriver_TIM_Init(TimSem,TimMsgQueue,CC2650_TIMER0_A);    //初始化 定时器
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    //配置 PWM 功能 信号从 Board_PWM0 输出
    TimErrCode = TTCSDKDriver_PWMInit(CC2650_TIMER0_A,Board_PWM0);
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }

    TimErrCode = TTCSDKDriver_TIM_Init(sem,appMsgQueue,TimerName);//初始化 定时器 TimerName
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
```

```
        return ;
    }
    //注册计数中断回调函数
    TimErrCode = TTCSDKDriver_Callback_Register(TimerName,TTCDriveTimerFuncCountCB);
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }

    TTCBleSDKManagerInfo_t ErrCode;
    ErrCode = TTCSDKDriver_Inst_IO_Init(NULL);                          //IO 初始化
    if(ErrCode != MANAGER_INFO_REQUEST_IO_SUCCESS){
        return;
    }
    //配置计数功能  信号从 Board_PWM2 输入
    TimErrCode = TTCSDKDriver_Count_Init(TimerName,Board_PWM2);
}

/*****************************************************************************
【函    数】  void TTCDriverSingleCatSignalTest(ICall_Semaphore * sem,
                                                 Queue_Handle * appMsgQueue,
                                                 CC2650_TimerName TimerName)
【概    述】  配置一个输入捕获中断
【入口参数】  sem：主线程信号量
              appMsgQueue：主线程消息队列
             TimerName：定时器
【返回参数】  无
【说    明】  本函数设置定时器 CC2650_TIMER0_A 从 Board_PWM0 脚输出 PWM 信号；
              设置指定的 定时器 TimerName 为输入捕获模式，信号从 Board_PWM2 脚输入；
              在计数中断回调函数里翻转 Board_PWM7,，同时通过串口打印两次捕获的时间
              差（此差值就是输入信号两个边沿之间时间）

              注意：这里有用到 UART 驱动，所以请设置好与 UART 相关的内容。
*****************************************************************************/
static void TTCDriverSingleCatSignalTest(ICall_Semaphore * sem,
                                         Queue_Handle * appMsgQueue,
                                         CC2650_TimerName TimerName){
    if(sem == NULL || appMsgQueue == NULL){
        return;
    }
    if(TimerName == CC2650_TIMER0_A){
        return;
    }
```

```
    TimSem      = sem;
    TimMsgQueue = appMsgQueue;
    TimErrCode = TTCSDKDriver_TIM_Init(TimSem,TimMsgQueue,CC2650_TIMER0_A); //初始化 定时器
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    //配置 PWM 功能 信号从 Board_PWM0 输出
    TimErrCode = TTCSDKDriver_PWMInit(CC2650_TIMER0_A,Board_PWM0);
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }

    TimErrCode = TTCSDKDriver_TIM_Init(sem,appMsgQueue,TimerName);//初始化 定时器 TimerName
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
     //注册回掉函数
    TimErrCode = TTCSDKDriver_Callback_Register(TimerName,TTCDriveTimerFuncCaptureCB);
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    TTCBleSDKManagerInfo_t ErrCode;
    ErrCode = TTCSDKDriver_Inst_IO_Init(NULL);   //IO 初始化
    if(ErrCode != MANAGER_INFO_REQUEST_IO_SUCCESS){
        return;
    }
    //配置捕获功能信号从 Board_PWM2 输入
    TimErrCode = TTCSDKDriver_Cat_Init(TimerName,Board_PWM2);
}



/*****************************************************************************
【函    数】  TTCSDKDriverDemoTimerEvent(UArg arg)
【概    述】  本地事件处理函数
【入口参数】  无
【返回参数】  无
【说    明】  无
*****************************************************************************/
void TTCSDKDriverDemoTimerEvent(void){
    if(events & TTCBLE_SDK_TIME_EVN){
        events &= ~TTCBLE_SDK_TIME_EVN;
```

```
        TTCDriverSinglePWMSignalDispose(CC2650_TIMER0_A);
    }
}

/*******************************************************************************
【函    数】  TTCSDKDriverTIMInit(ICall_Semaphore * sem,
                                  Queue_Handle * appMsgQueue,
                                  CC2650_TimerName TimerName)
【概    述】  定时器示例初始化化
【入口参数】  sem         : 信号量
              appMsgQueue : 消息句柄
              TimerName   : 定时器
【返回参数】  无
【说    明】  每次只能测试一个，不然会有冲突
*******************************************************************************/
void TTCSDKDriverTIMInit(ICall_Semaphore * sem,
                         Queue_Handle * appMsgQueue,
                         CC2650_TimerName TimerName){
       //PWM 输出
     TTCDriverSinglePWMSignalTest(sem,appMsgQueue,TimerName);
       //定时中断
//     TTCDriverSingleInterSignalTest(sem,appMsgQueue,TimerName);
       //外部输入计数 （此处不能设置为 CC2650_TIMER0_A）
//     TTCDriverSingleCountSignalTest(sem,appMsgQueue,TimerName);
       //外部输入捕获 （此处不能设置为 CC2650_TIMER0_A）
//     TTCDriverSingleCatSignalTest(sem,appMsgQueue,TimerName);

}

#endif
```

4.4 ADC 说明

CC2640 拥有 8 路 12bit 的 ADC 通道，支持 200Ksamples 的采样率，它的时钟源可以自由设置，包括定时器，I/O 引脚，软件，模拟比较器和 RTC。另外它可以采集到片内温度传感器的当前温度值以及通过内部电路采集到电源电压，方便实现电池的管理。

4.4.1 ADC API 说明

4.4.1.1 TTCDrvierAdcReadSync()

```
/*****************************************************************************
【函    数】 TTCDrvierAdcReadSync
【概    述】 同步 ADC 采样
【入口参数】 adconfig   :ADC 配置参数
             simpleTimes:对指定的通道设置连续采样次数，采样次数范围 1-255
             adcReadBuf :保存 AD 值缓存，缓存大小应大于等于采样次数。注意一定不能小于采样次数。
【返回参数】 ADC 采样值
【说    明】
             adconfig.refsource 参考电压源：
                                  AUXADC_REF_FIXED (nominally 4.3 V)
                                  AUXADC_REF_VDDA_REL (nominally VDDS)
                                  AUXADC_REF_EXT (unscaled voltage applied to AUX I/O 7)
             adconfig.auxIo 内部、外部输入通道选择：
                                      ADC_COMPB_IN_VDD1P2V
                                      ADC_COMPB_IN_VSSA
                                      ADC_COMPB_IN_VDDA3P3V
                                      ADC_COMPB_IN_AUXIO7
                                      ADC_COMPB_IN_AUXIO6
                                      ADC_COMPB_IN_AUXIO5
                                      ADC_COMPB_IN_AUXIO4
                                      ADC_COMPB_IN_AUXIO3
                                      ADC_COMPB_IN_AUXIO2
                                      ADC_COMPB_IN_AUXIO1
                                      ADC_COMPB_IN_AUXIO0
*****************************************************************************/
void TTCDrvierAdcReadSync(TTCDriverAdcConfing adconfig,u8 simpleTimes,u32 * adcReadBuf);
```

说明：

4.4.1.2 TTCDrvierAdcReadAsync()

```
/*****************************************************************************
```

【函　　数】 TTCDrvierAdcReadAsync

【概　　述】 异步 ADC 采样

【入口参数】 adconfig　:ADC 配置参数

simpleTimes:对指定的通道设置连续采样次数，采样次数范围 1-255

adcReadBuf :保存 AD 值缓存，缓存大小应大于等于采样次数。注意一定不能小于采样次数。

【返回参数】 ADC 采样值

【说　　明】 adconfig.time 设置无效

adconfig.nominally 参考电压源：

AUXADC_REF_FIXED (nominally 4.3 V)

AUXADC_REF_VDDA_REL (nominally VDDS)

AUXADC_REF_EXT (unscaled voltage applied to AUX I/O 7)

adconfig.auxIo 内部、外部输入通道选择：

ADC_COMPB_IN_VDD1P2V

ADC_COMPB_IN_VSSA

ADC_COMPB_IN_VDDA3P3V

ADC_COMPB_IN_AUXIO7

ADC_COMPB_IN_AUXIO6

ADC_COMPB_IN_AUXIO5

ADC_COMPB_IN_AUXIO4

ADC_COMPB_IN_AUXIO3

ADC_COMPB_IN_AUXIO2

ADC_COMPB_IN_AUXIO1

ADC_COMPB_IN_AUXIO0

```
*****************************************************************************/
void TTCDrvierAdcReadAsync(TTCDriverAdcConfing adconfig,u8 simpleTimes,u32 * adcReadBuf);
```

说明：

4.4.1.3 TTCDriverBatVoltageGet()

```
/*****************************************************************************
```

【函　　数】 TTCDriverBatVoltageGet

【概　　述】 读取电池 AD 值

【入口参数】 无

【返回参数】 电池电压（AD 值）

【说　　明】 BatVoltage = ((batval&0xff)>>5)*0.125 + (((batval>>8)&0xff));

返回值 32 位中[10:8]代表 INT 。[7:0]代表 FRAC ,对于小数部分，

一个单位代表 0.00390625v，小数部分的分辨率只有 50mV(TYP)

```
*******************************************************************************/
extern u32 TTCDriverBatVoltageGet(void);
```

说明：调用此 API 能直接获取电源电压的 AD 值。

4.4.1.4 TTCDriverBatTempClose()

```
/*******************************************************************************
【函    数】  TTCDriverBatTempClose
【概    述】  关闭采样电压、温度 AD 值
【入口参数】  无
【返回参数】  无
【说    明】
*******************************************************************************/
extern void TTCDriverBatTempClose(void);
```

说明：

4.4.1.5 TTCDriverTempGet()

```
/*******************************************************************************
【函    数】  TTCDriverTempGet
【概    述】  读取温度值
【入口参数】  无
【返回参数】  温度值（摄氏温度）
【说    明】  返回值为摄氏温度
*******************************************************************************/
extern u32 TTCDriverTempGet(void);
```

说明：读取片内温度传感器的 AD 值。

4.4.2 ADC 使用示例

```
/*******************************************************************************
【文    件】  TTCDriverADCDemo.c
【概    述】  TTC SDK ADC 示例代码
【编    写】  SDK 工作小组
【修    订】  SDK 工作小组
【修订日期】  2016/12/02
【版    本】  V1.0.0
【说    明】
```

功能:从 IOID_7 脚采集电压，若开启了 UART 驱动，则会利用 UART 将采集到的电压对应的 AD 值打印到串口助手上。

添加的步骤：

1、添加对应的头文件 TTCDriverADCDemo.h

2、在工程中的 Opions/C/C++ Compiler/Defined symbols 中定义 TTCDRIVER_ADC

3、在 TTCBlePeripheralTaskFxn()函数里添加

```
#ifdef TTCDRIVER_ADC
    TTCSDKDriverDemoADCEvent();
#endif //TTCDRIVER_ADC
```

4、在主线程里的 static void TTCSDKDriverInit(void)函数中添加

```
#ifdef TTCDRIVER_ADC。
    TTCDriverDemoADCInit(&sem,&appMsgQueue);
#endif
```

```
*********************************************************************************/
#ifdef TTCDRIVER_ADC
/*********************************************************************
 * 头文件
 */
#include <stdio.h>
#include <string.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Queue.h>
#include <ti/drivers/PIN.h>
#include <ti/drivers/pin/PINCC26XX.h>
#include <driverlib/aux_adc.h>
#include "TTCSDKBoard.h"
#include "TTCBleSDKConfig.h"
#include "TTCBleSDKManager.h"
#include "TTCDriverADC.h"
#ifdef TTCDRIVER_UART
#include <ti/drivers/UART.h>
#include <ti/drivers/uart/UARTCC26XX.h>
#include "TTCDriverUART.h"
#include "TTCDriverUARTDemo.h"
#endif //TTCDRIVER_UART
#include "TTCDriverADCDemo.h"
#define TTCBLE_SDK_ADC_EVN                    0x0001                          //

/*********************************************************************
 * 本地变量
```

```
 */
u32 adc[20]={0};
Clock_Struct adcClock;
TTCDriverAdcConfing adconfig;
PIN_Handle adPinHandle;
PIN_State  adPinState;
PIN_Config adPinConfig[] = {

  IOID_7 | PIN_GPIO_OUTPUT_DIS | PIN_INPUT_EN | PIN_NOPULL,    //配置为浮空输入
  PIN_TERMINATE
};

static ICall_Semaphore *AdcSem;                                          //线程信号量,
用于唤醒线程
static Queue_Handle *AdcMsgQueue;                                         //消息句柄
static u16 events;                                                        //本地事件

/*********************************************************************
 * 本地函数声明
 */
static void TTCSDKDriverADCRead(void);
static void TTCDriverDemoADCClockHandler(UArg arg);
static void TTCDriverDemoADCSetEvent(UArg arg);

/*********************************************************************************
【函    数】  TTCDriverDemoADCClockHandler(UArg arg)
【概    述】  线程置事件
【入口参数】  arg    : 标记事件源事件
【返回参数】
【说    明】
*********************************************************************************/
static void TTCDriverDemoADCClockHandler(UArg arg){

  TTCDriverDemoADCSetEvent(arg);
}
/*********************************************************************************
【函    数】  TTCDriverDemoADCInit(ICall_Semaphore * sem,Queue_Handle * appMsgQueue)
【概    述】  TTCDriver 驱动初始化
【入口参数】  sem:主线程的信号量
              appMsgQueue:主线程的消息队列
【返回参数】  无
【说    明】  无
```

```
*****************************************************************************/
void TTCDriverDemoADCInit(ICall_Semaphore * sem,Queue_Handle * appMsgQueue){

    if(sem == NULL || appMsgQueue == NULL){
      return;
    }

    AdcSem      = sem;
    AdcMsgQueue = appMsgQueue;

  //先将对应的GPIO设置为浮空输入,一定要先配置对应的GPIO为浮空输入,否则转换出来的电压值不对。

    TTCBleSDKManagerInfo_t ErrCode;

    ErrCode = TTCDriverIOOpen(&adPinHandle,&adPinState,(const PIN_Config *)adPinConfig);

    if(MANAGER_INFO_REQUEST_IO_SUCCESS != ErrCode){
      return;
    }

    adconfig.refsource = ADC_REF_VDDA_REL;
    adconfig.ref       = ADC_REF_1P43V;        //
    adconfig.time      = ADC_TIME_2P7_US;
    adconfig.trigger   = ADC_TRIGGER_GPTOA;    //对定时器的使用没有影响
    adconfig.auxIo     = ADC_COMPB_IN_AUXIO7;  //IOID_7  对应关系请查看TTCDriverADC.h文件

    Util_constructClock(&adcClock,             //配置一个软件定时器 周期性的调用ADC采集函数
                        TTCDriverDemoADCClockHandler,
                        100,                   //定时100mS
                        0,
                        true,
                        TTCBLE_SDK_ADC_EVN);
}


/*****************************************************************************
【函    数】  void TTCDriverDemoADCSetEvent(UArg arg)
【概    述】
【入口参数】  arg：事件
【返回参数】  无
【说    明】  无
*****************************************************************************/
```

```
static void TTCDriverDemoADCSetEvent(UArg arg){
  events |= arg;
  Semaphore_post(*AdcSem);
}
/*****************************************************************************
【函    数】  TTCSDKDriverADCEvent(void)
【概    述】  处理本地事件
【入口参数】  无
【返回参数】  无
【说    明】  无
*****************************************************************************/
void TTCSDKDriverADCEvent(void){

  if(events & TTCBLE_SDK_ADC_EVN){

    events &= ~TTCBLE_SDK_ADC_EVN;

    TTCSDKDriverADCRead();
  }
}
/*****************************************************************************
【函    数】  TTCSDKDriverADCRead(void)
【概    述】
【入口参数】  无
【返回参数】  无
【说    明】  无
*****************************************************************************/
static void  TTCSDKDriverADCRead(void){

    TTCDrvierAdcReadSync(adconfig,20,(u32 *)adc);       //连续采集 20 次

    #ifdef TTCDRIVER_UART

        u8 printfBuf[20] = {0};
        u8 i=0;
        u32 sum = 0;

        for(i=0;i<20;i++){
            sum += adc[i];
        }
        sum /= 20;       //取 20 个 ad 值得平均值
        sprintf((char *)printfBuf,"adcValue:%d\r\n",sum);
```

```
        TTCDriverUartWrite(&uartHandle,(u8 *)printfBuf,strlen((const char *)printfBuf));

//      u32 BatVoltage = 0;
//      float bat = 0.0;
//
//      BatVoltage = TTCDriverBatVoltageGet();       //读取电源电压
//
//     返回值 32 位中[10:8]代表 INT ,[7:0]代表 FRAC.
//      bat = ((BatVoltage&0xff)>>5)*0.125 + (((BatVoltage>>8)&0xff));  //
//      sprintf((char *)printfBuf,"BatVoltage:%d\r\n",(u32)(bat*100));
//      TTCDriverUartWrite(&uartHandle,(u8 *)printfBuf,strlen((const char *)printfBuf));

    #endif

    Util_startClock(&adcClock);
}

#endif
```

4.5 UTC 说明

CC2640 的 RTC 时钟来源于 32Khz 的外部晶振，在不断电的情况下会一直自动计数。另外它还拥有一个 70bit 的可编程的计数器以及三个通用的通道，配合它的比较寄存器使用，可以产生与时间相关的通知来告知应用层，以实现一些必要的功能。

4.5.1 UTC API 说明

4.5.1.1 TTCDriverUTCInit()

```
/*****************************************************************************
【函    数】  TTCDriverInfo_t TTCDriverUTCInit(TTCSdkClass_t *appCallbacks,
                                                TTCDriverUTCParams_t utcConfig)
【概    述】  初始化 UTC
【入口参数】  appCallbacks : 注册回调
              utcConfig    : UTC 配置信息
【返回参数】  请参考 TTCDriverUartInfo_t
【说    明】
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverUTCInit(TTCSdkClass_t *appCallbacks,
                                        TTCDriverUTCParams_t utcConfig);
```

说明：初始化 UTC，注册回调函数。UTC 的回调函数由 appCallbacks 参数传入。

4.5.1.2 TTCDriverUTCReschedule

```
/*****************************************************************************
【函    数】  TTCDriverInfo_t TTCDriverUTCReschedule(TTCDriverUTCParams_t utcConfig)
【概    述】  设置 UTC
【入口参数】  utcConfig : UTC 配置信息
【返回参数】  请参考 TTCDriverUartInfo_t
【说    明】
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverUTCReschedule(TTCDriverUTCParams_t utcConfig);
```

说明：设置 UTC，用于更新 UTC 的时间。

4.5.1.3 TTCDriverUTCGetClock()

```
/*****************************************************************************
```

```
【函    数】  TTCDriverUTCGetClock(TTCDriverUTCTime_t * time)
【概    述】  读取当前时钟
【入口参数】  time : 时间结构体
【返回参数】  无
【说    明】  无
*****************************************************************************/
extern void TTCDriverUTCGetClock(TTCDriverUTCTime_t * time);
```

说明：获取当前 UTC 的信息。

4.5.2 UTC 使用示例

```
/*****************************************************************************
【文    件】  TTCDriverUTCDemo.c
【概    述】  TTC SDK UTC 示例代码
【编    写】  SDK 工作小组
【修    订】  SDK 工作小组
【修订日期】  2016/12/02
【版    本】  V1.0.0
【说    明】
```

功能说明：设置了一个 UTC 时间，同时开启了一个周期事件来定时读取 UTC 的时间，若开启了 UART 的驱动，则会在读取时间之后将这个时间值打印到串口助手上。

添加的步骤：

1、添加对应的头文件 TTCDriverUTCDemo.h

2、在工程中的 Opions/C/C++ Compiler/Defined symbols 中定义 TTCDRIVER_UTC

3、在 TTCBlePeripheralTaskFxn()函数里添加

```
#ifdef TTCDRIVER_UTC
    TTCSDKDriverDemoUTCEvent();
#endif
```

4、在 TTCSDKDriverInit(void)函数中添加

```
#ifdef TTCDRIVER_UTC。
    TCDriverDemoUTCInit(&sem,&appMsgQueue);
#endif
```

```
*****************************************************************************/
#ifdef TTCDRIVER_UTC
#include <stdio.h>
#include <string.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Queue.h>
#include <ti/drivers/PIN.h>
```

```
#include <ti/drivers/pin/PINCC26XX.h>
#include "TTCSDKBoard.h"
#include "TTCBleSDKConfig.h"
#include "TTCBleSDKManager.h"
#ifdef TTCDRIVER_UART
#include <ti/drivers/UART.h>
#include <ti/drivers/uart/UARTCC26XX.h>
#include "TTCDriverUART.h"
#include "TTCDriverUARTDemo.h"
#endif //TTCDRIVER_UART
#include "TTCDriverUTC.h"
#include "TTCDriverUTCDemo.h"

/*********************************************************************
 * 常量及宏定义
 */
#define TTCBLE_SDK_UTC_EVN                    0x0008

/*********************************************************************
 * 本地变量
 */
static TTCDriverInfo_t UTCErrCode;

TTCDriverUTCTime_t  sysUTC;
Clock_Struct UtcClock;

static ICall_Semaphore * UTCSem;                    //线程信号量，用于唤醒线程
static Queue_Handle * UTCMsgQueue;                  //消息句柄
static u16 events;                                  //本地事件

extern TTCSdkClass_t TTCBlePeripheralTaskCls;

/*********************************************************************
 * 本地函数声明
 */
static void TTCDriverUtcPrintf(TTCDriverUTCTime_t time);
static void TTCDriverDemoUTCClockHandler(UArg arg);
static void TTCSDKDriverUTCSetEvent(UArg arg);

/*********************************************************************************
【函    数】  TTCDriverDemoUTCClockHandler(UArg arg)
【概    述】  线程置事件
```

```
【入口参数】  arg    : 标记事件源事件
【返回参数】  无
【说    明】
*******************************************************************************/
static void TTCDriverDemoUTCClockHandler(UArg arg){
    TTCSDKDriverUTCSetEvent(arg);
}

/*******************************************************************************
【函    数】  TTCSDKDriverUTCSetEvent(UArg arg)
【概    述】  标记事件并唤醒线程
【入口参数】  arg : 标记事件
【返回参数】  无
【说    明】  无
*******************************************************************************/
static void TTCSDKDriverUTCSetEvent(UArg arg){
    events |= arg;
    Semaphore_post(*UTCSem);
}

/*******************************************************************************
【函    数】  TTCSDKDriverDemoUTCEvent(void)
【概    述】  Demo UTC 事件处理
【入口参数】  无
【返回参数】  无
【说    明】  无
*******************************************************************************/
void TTCSDKDriverDemoUTCEvent(void){
    if(events & TTCBLE_SDK_UTC_EVN){
        events &= ~TTCBLE_SDK_UTC_EVN;
        TTCSDKDriver_UTC_Display();
    }
}

/*******************************************************************************
【函    数】  TTCDriverDemoUTCInit(ICall_Semaphore * sem,Queue_Handle * appMsgQueue)
【概    述】  TTCDriver 驱动初始化
【入口参数】  sem：主线程的信号量
              appMsgQueue：主线程的消息队列
【返回参数】  无
【说    明】  无
*******************************************************************************/
```

```
void TTCDriverDemoUTCInit(ICall_Semaphore * sem,Queue_Handle * appMsgQueue){
    if(sem == NULL || appMsgQueue == NULL){
        return;
    }
    UTCSem      = sem;
    UTCMsgQueue = appMsgQueue;

    TTCDriverUTCParams_t utcConfig;
    utcConfig.sem           = UTCSem;
    utcConfig.queueHandle   = UTCMsgQueue;
    utcConfig.time.year     = 2016;
    utcConfig.time.month    = 12;
    utcConfig.time.day      = 3;
    utcConfig.time.hour     = 21;
    utcConfig.time.minutes  = 45;
    utcConfig.time.seconds  = 0;
    utcConfig.time.week     = 0;
    utcConfig.upDatePeriod  = 1000;             //UTC 时间的刷新周期
    utcConfig.updateMsgFlag = false;            //true false

    UTCErrCode = TTCDriverUTCInit(&TTCBlePeripheralTaskCls,utcConfig);

    if(UTCErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    //没有开启消息通知时我们可以自己定义一个周期事件来主动获取 UTC 时间
    if(utcConfig.updateMsgFlag == false){
        Util_constructClock(&UtcClock,
                            TTCDriverDemoUTCClockHandler,
                            1000,
                            0,
                            true,
                            TTCBLE_SDK_UTC_EVN);
    }
}

/*********************************************************************************
【函    数】  TTCSDKDriver_UTC_Display(void)
【概    述】  打印 UTC 信息
【入口参数】  无
【返回参数】  无
【说    明】  无
```

```
*****************************************************************************/
void TTCSDKDriver_UTC_Display(void){
    TTCDriverUTCGetClock(&sysUTC);
    TTCDriverUtcPrintf(sysUTC);
    Util_startClock(&UtcClock);
}

/*****************************************************************************
【函    数】  TTCDriverUtcProcess(TTCDriverUTCTime_t * TTCMsg)
【概    述】  处理 UTC
【入口参数】  time : 接收 UTC 数据消息结构体
【返回参数】  无
【说    明】  无
*****************************************************************************/
void TTCDriverUtcProcess( TTCDriverUTCTime_t * time ){
    memcpy(&sysUTC,time,sizeof(TTCDriverUTCTime_t));
    ICall_free(time);
    TTCDriverUtcPrintf(sysUTC);
}
/*****************************************************************************
【函    数】  TTCDriverUtcPrintf( TTCDriverUTCTime_t  time )
【概    述】  打印时间数据
【入口参数】  time：
【返回参数】  无
【说    明】  无
*****************************************************************************/
static void TTCDriverUtcPrintf(TTCDriverUTCTime_t time){
  #ifdef TTCDRIVER_UART
    u8 printfBuf[20] = {0};
    sprintf((char *)printfBuf,"UTC %d %d %d %d %d %d\r\n",
            time.year,
            time.month,
            time.day,
            time.hour,
            time.minutes,
            time.seconds);
    TTCDriverUartWrite(&uartHandle,printfBuf,strlen((const char *)printfBuf));
  #endif
}
#endif
```

4.6. IIC 说明

IIC 接口可用与其他支持 IIC 协议的器件通信，如 ROM，LCD 及多种传感器等。普通模式速率为 100KHz，快速模式速率为 400KHz。

4.6.1 IIC API 说明

4.6.1.1 TTCDriverI2cInitDefaultParam()

```
/*****************************************************************************
【函    数】  TTCDriverI2cInitDefaultParam(I2c_Handle * i2cHandle)
【概    述】  设置 I2C 句柄默认参数
【入口参数】  i2cHandle : I2C 句柄
【返回参数】  无
【说    明】  无
*****************************************************************************/
extern void TTCDriverI2cInitDefaultParam(I2c_Handle * i2cHandle);
```

4.6.1.2 TTCDriverI2cInit()

```
/*****************************************************************************
【函    数】  TTCDriverInfo_t TTCDriverI2cInit(TTCSdkI2cCB_t *appCallbacks,
                                               I2C_Handle * i2cHandle,
                                               TTCDriverI2cParams_t param)
【概    述】  初始化 I2C
【入口参数】  appCallbacks : 注册回调
              i2cHandle    : I2C 句柄
              param        : I2C 参数
【返回参数】  请参考 TTCDriverUartInfo_t
【说    明】  若初始化失败 I2C 会返回初始化失败原因。
              注意：若 I2C 为回调模式则必须设置回调函数，否则初始化失败
                    若 I2C 为阻塞模式则无需设置回调函数
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverI2cInit(TTCSdkI2cCB_t appCallbacks,
                                        I2c_Handle * i2cHandle,
                                        TTCDriverI2cParams_t param);
```

回调函数格式：

```
typedef void (*TTCSdkI2cCB_t)(TTCDriverI2cState_t i2cState,u8 * buffer,u16 len,void * arg);
```

4.6.1.3 TTCDriverI2cRead

```
/*****************************************************************************
【函    数】  TTCDriverI2cRead(I2c_Handle * i2cHandle,
                               u8 slaveAddr,
                               u8 * wBuf,
                               u8 * rBuf,
                               u16 wLen,
                               u16 rLen,
                               void * arg)
【概    述】  I2C 读取数据
【入口参数】  i2cHandle : I2C 句柄
              slaveAddr : I2C 从设备地址
              wBuf      : 写缓存
              rBuf      : 读缓存
              wLen      : 写缓存长度
              rLen      : 读缓存长度
              arg       : 回调模式下传输前传入的参数，将会传递到回调函数中
【返回参数】  请参考 TTCDriverInfo_t
【说    明】  注意：1.在回调模式下读缓存可直接设置为 NULL
                    在阻塞模式下读缓存不能为 NULL
                  2.slaveAddr 为 I2C 从设备的真实地址
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverI2cRead(I2c_Handle * i2cHandle,
                                        u8 slaveAddr,
                                        u8 * wBuf,
                                        u8 * rBuf,
                                        u16 wLen,
                                        u16 rLen,
                                        void * arg);
```

4.6.1.4 TTCDriverI2cWrite()

```
/*****************************************************************************
【函    数】  TTCDriverInfo_t TTCDriverI2cWrite(I2c_Handle * i2cHandle,
                                                u8 slaveAddr,
                                                u8 * wBuf,
```

```
                                        u16 wLen,
                                        void * arg)
【概    述】  I2C 写数据
【入口参数】  i2cHandle : I2C 句柄
              slaveAddr : I2C 从设备地址
              wBuf      : 写缓存
              wLen      : 写缓存长度
              arg       : 回调模式下传输前传入的参数，将会传递到回调函数中
【返回参数】  请参考 TTCDriverInfo_t
【说    明】  注意：slaveAddr 为 I2C 从设备的真实地址
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverI2cWrite(I2c_Handle * i2cHandle,
                                         u8 slaveAddr,
                                         u8 * wBuf,
                                         u16 wLen,
                                         void * arg);
```

4.6.1.5 TTCDriverI2CBusy()

```
/*******************************************************************************
【函    数】  TTCDriverI2CBusy(void)
【概    述】  判断 I2C 传输是否忙
【入口参数】  无
【返回参数】  true : 忙状态
              false : 空闲
【说    明】  读写大量数据时需要调用此函数。
*******************************************************************************/
extern bool TTCDriverI2CBusy(void);
```

4.6.2 IIC 使用示例

```
/*******************************************************************************
【文    件】  TTCDriverI2CDemo.c
【概    述】  TTC SDK I2C 示例代码
【编    写】  SDK 工作小组
【修    订】  SDK 工作小组
【修订日期】  2016/11/29
【版    本】  V1.0.0
【说    明】  本示例代码基于 LIS3DH 的 I2C 模式调试
```

功能说明：

此文件使用 2640 的 SDK 的 IIC 驱动为基础，对 Gsensor LIS3DH 进行相关配置并采集原始数据，原始数据再通过串口打印，串口的数据格式已经做了处理，可以用匿名四轴上位机进行查看实时波形。

添加步骤：

1.打开相应的宏定义 TTCDRIVER_I2C，需要用到串口时还需要打开宏 TTCDRIVER_UART

2.在主线程初始化中的驱动示例初始化 TTCSDKDriverInit();中增加 IIC 初始化配置 TTCDriverDemoI2CInit(&sem,&appMsgQueue);

3.在主线程处理函数中增加 IIC 内部事件处理 TTCDriverI2cEvent();和 DEMO IIC 数据处理 TTCSDKDriverDemoI2CEvent();

4.脚位的定义在 TTCSDKBoard.h 中

```
*********************************************************************************/
#ifdef TTCDRIVER_I2C
/*********************************************************************
 * 头文件
 */
#include <string.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Queue.h>
#include <ti/drivers/PIN.h>
#include <ti/drivers/pin/PINCC26XX.h>
#include <ti/drivers/I2C.h>
#include <ti/drivers/i2c/I2CCC26XX.h>
#include <driverlib/aux_wuc.h>
#include "hci_tl.h"
#include "gatt.h"
#include "gapgattserver.h"
#include "gattservapp.h"
#include "gapbondmgr.h"
#include "osal_snv.h"
#include "ICallBleAPIMSG.h"
#include "util.h"
#include "TTCSDKBoard.h"
#include "TTCBleSDKConfig.h"
#include "TTCBleSDKManager.h"
#include "TTCBleDevInfoService.h"
#include "TTCBlePeripheral.h"
#include "TTCBlePeripheralProcess.h"
#include "TTCBleProfile.h"
```

```
#ifdef TTCDRIVER_UART
#include <ti/drivers/UART.h>
#include <ti/drivers/uart/UARTCC26XX.h>
#include "TTCDriverUART.h"
#include "TTCDriverUARTDemo.h"
#endif //TTCDRIVER_UART

#include "TTCDriverI2C.h"
#include "TTCDriverI2CDemo.h"

/*********************************************************************
 * 常量及宏定义
 */
//Register Definition
#define LIS3DH_WHO_AM_I             0x0F  // device identification register

// CONTROL REGISTER 1
#define LIS3DH_CTRL_REG1            0x20
#define LIS3DH_ODR_BIT              BIT(4)
#define LIS3DH_LPEN                 BIT(3)
#define LIS3DH_ZEN                   BIT(2)
#define LIS3DH_YEN                   BIT(1)
#define LIS3DH_XEN                   BIT(0)

//CONTROL REGISTER 2
#define LIS3DH_CTRL_REG2                0x21
#define LIS3DH_HPM                       BIT(6)
#define LIS3DH_HPCF                     BIT(4)
#define LIS3DH_FDS                       BIT(3)
#define LIS3DH_HPCLICK                  BIT(2)
#define LIS3DH_HPIS2                    BIT(1)
#define LIS3DH_HPIS1                    BIT(0)

//CONTROL REGISTER 3
#define LIS3DH_CTRL_REG3                0x22
#define LIS3DH_I1_CLICK                  BIT(7)
#define LIS3DH_I1_AOI1                  BIT(6)
#define LIS3DH_I1_AOI2                   BIT(5)
#define LIS3DH_I1_DRDY1                  BIT(4)
#define LIS3DH_I1_DRDY2                  BIT(3)
#define LIS3DH_I1_WTM                   BIT(2)
#define LIS3DH_I1_ORUN                  BIT(1)
```

```
//CONTROL REGISTER 6
#define LIS3DH_CTRL_REG6                0x25
#define LIS3DH_I2_CLICK                 BIT(7)
#define LIS3DH_I2_INT1                  BIT(6)
#define LIS3DH_I2_BOOT                  BIT(4)
#define LIS3DH_H_LACTIVE                BIT(1)

//TEMPERATURE CONFIG REGISTER
#define LIS3DH_TEMP_CFG_REG             0x1F
#define LIS3DH_ADC_PD                   BIT(7)
#define LIS3DH_TEMP_EN                  BIT(6)

//CONTROL REGISTER 4
#define LIS3DH_CTRL_REG4            0x23
#define LIS3DH_BDU                  BIT(7)
#define LIS3DH_BLE                  BIT(6)
#define LIS3DH_FS                   BIT(4)
#define LIS3DH_HR                   BIT(3)
#define LIS3DH_ST                   BIT(1)
#define LIS3DH_SIM                  BIT(0)

//CONTROL REGISTER 5
#define LIS3DH_CTRL_REG5            0x24
#define LIS3DH_BOOT                             BIT(7)
#define LIS3DH_FIFO_EN                          BIT(6)
#define LIS3DH_LIR_INT1                         BIT(3)
#define LIS3DH_D4D_INT1                         BIT(2)

//REFERENCE/DATA_CAPTURE
#define LIS3DH_REFERENCE_REG            0x26
#define LIS3DH_REF                      BIT(0)

//STATUS_REG_AXIES
#define LIS3DH_STATUS_REG               0x27
#define LIS3DH_ZYXOR                            BIT(7)
#define LIS3DH_ZOR                              BIT(6)
#define LIS3DH_YOR                              BIT(5)
#define LIS3DH_XOR                              BIT(4)
#define LIS3DH_ZYXDA                            BIT(3)
#define LIS3DH_ZDA                              BIT(2)
#define LIS3DH_YDA                              BIT(1)
```

```
#define LIS3DH_XDA                              BIT(0)

//STATUS_REG_AUX
#define LIS3DH_STATUS_AUX               0x07

//INTERRUPT 1 CONFIGURATION
#define LIS3DH_INT1_CFG                 0x30
#define LIS3DH_ANDOR                            BIT(7)
#define LIS3DH_INT_6D                           BIT(6)
#define LIS3DH_ZHIE                             BIT(5)
#define LIS3DH_ZLIE                             BIT(4)
#define LIS3DH_YHIE                             BIT(3)
#define LIS3DH_YLIE                             BIT(2)
#define LIS3DH_XHIE                             BIT(1)
#define LIS3DH_XLIE                             BIT(0)

//FIFO CONTROL REGISTER
#define LIS3DH_FIFO_CTRL_REG                    0x2E
#define LIS3DH_FM                               BIT(6)
#define LIS3DH_TR                               BIT(5)
#define LIS3DH_FTH                              BIT(0)

//OUTPUT REGISTER
#define LIS3DH_OUT_X_L                  0x28
#define LIS3DH_OUT_X_H                  0x29
#define LIS3DH_OUT_Y_L                  0x2A
#define LIS3DH_OUT_Y_H                  0x2B
#define LIS3DH_OUT_Z_L                  0x2C
#define LIS3DH_OUT_Z_H                  0x2D

//INT1 REGISTERS
#define LIS3DH_INT1_THS                         0x32
#define LIS3DH_INT1_DURATION                    0x33

//INTERRUPT 1 SOURCE REGISTER
#define LIS3DH_INT1_SRC                   0x31



#define TXBUF_LEN               20
#define RXBUF_LEN               20
#define SBP_IIC_EVT             1
```

```
#define SLAVEADDR               0x19

typedef enum {
    MEMS_SUCCESS =  0x01,
    MEMS_ERROR   =  0x00
} status_t;

typedef enum {
    MEMS_ENABLE   = 0x01,
    MEMS_DISABLE  = 0x00
} State_t;

typedef struct {
    int16 AXIS_X;
    int16 AXIS_Y;
    int16 AXIS_Z;
} AxesRaw_t;


extern const I2CCC26XX_HWAttrs  i2cCC26XXHWAttrs[];

const TTCDriverI2cParams_t i2cParam = {
    .i2cName = CC2650_I2C0,
    .bitRate = I2C_400kHz,
    .transferMode = I2C_MODE_BLOCKING,  // I2C_MODE_CALLBACK
    .i2cHWAttr = &i2cCC26XXHWAttrs[CC2650_I2C0],
};

/*********************************************************************
 * 本地变量
 */
uint8 IIcRxbuffer[RXBUF_LEN]={0};
I2c_Handle i2cHandle;
Clock_Struct IICClock;
TTCDriverInfo_t IIC_state = TTCDRIVER_INFO_SIZE;
static ICall_Semaphore *iicsem;

static Clock_Struct iicClock;
static u16 events;
static AxesRaw_t Acc_data;
#ifdef TTCDRIVER_UART
extern Uart_Handle uartHandle;
```

```
#endif //TTCDRIVER_UART

/*********************************************************************
 * 本地函数声明
 */
static void LIS3DHI2cCB(TTCDriverI2cState_t i2cState,u8 * buffer,u16 len,void * arg);
static void SendRegiicCB(u32 param,u8 * buffer,u16 len);
static uint8 LIS3DH_GetAccAxesRaw(I2c_Handle * i2cHandle, AxesRaw_t* buff);
static uint8 LIS3DH_GetAccAxesRaw_cb(I2c_Handle * i2cHandle);
static uint8 LIS3DH_WriteReg(I2c_Handle * i2cHandle,uint8 Reg, uint8 Data);
static uint8 LIS3DH_WriteReg_cb(I2c_Handle * i2cHandle,uint8 Reg, uint8 Data);
static uint8 LIS3DH_ReadReg_cb(I2c_Handle * i2cHandle, uint8 Reg);
static uint8 LIS3DH_ReadReg(I2c_Handle * i2cHandle, uint8 Reg, uint8 *Data);
static void LIS3DH_Init_cb(I2c_Handle * i2cHandle);
static void LIS3DH_Init(I2c_Handle * i2cHandle);
static void IICDatatoUart(uint8 *Data,u16 len);
static void TTCDriverDemoI2CClockHandler(UArg arg);

/*********************************************************************
 * 回调函数
 */
LIS3DHiicCB TTCiicArgcb = {
  .param = 1,
  .iiccb = &SendRegiicCB,
};

/*****************************************************************************
【函    数】  TTCDriverDemoI2CInit(ICall_Semaphore *sem,Queue_Handle *queueHandle)
【概    述】  I2C 初始化及参数设置
【入口参数】  sem          ：信号量
              appMsgQueue ：消息句柄
【返回参数】  无
【说    明】  无
*****************************************************************************/
void TTCDriverDemoI2CInit(ICall_Semaphore *sem,Queue_Handle *queueHandle){

    TTCDriverI2cInitDefaultParam(&i2cHandle);
    i2cHandle.sem                           = sem;
    iicsem                                  = sem;
    i2cHandle.queueHandle                   = queueHandle;
    IIC_state = TTCDriverI2cInit( &LIS3DHI2cCB,
                                  &i2cHandle,
```

```
                                    i2cParam);        //IIC 初始化

    Util_constructClock(&iicClock,
                        TTCDriverDemoI2CClockHandler,
                        1000,
                        40,
                        false,
                        SBP_IIC_EVT);      //构建 IIC 定时采 Gsensor 数据事件

    if(i2cParam.transferMode == I2C_MODE_BLOCKING){
        LIS3DH_Init(&i2cHandle);                    //阻塞模式
    }else{
        LIS3DH_Init_cb(&i2cHandle);                 //回调模式
    }


}


/*********************************************************************************
【函    数】  TTCSDKDriverDemoI2CEvent(void)
【概    述】  读取 Gsensor 数据事件
【入口参数】  无
【返回参数】  无
【说    明】  目前为 40MS 一次的采样频率
*********************************************************************************/
void TTCSDKDriverDemoI2CEvent(void) {
    if(events & SBP_IIC_EVT){
        events &= ~SBP_IIC_EVT;
        if(i2cParam.transferMode == I2C_MODE_CALLBACK){
            LIS3DH_GetAccAxesRaw_cb(&i2cHandle);            //回调方式采样
            Util_stopClock(&iicClock);
        } else{
            LIS3DH_GetAccAxesRaw(&i2cHandle,&Acc_data);    //阻塞方式采样
            IICDatatoUart(&IIcRxbuffer[1], 6);             //串口打印采样数据
        }
    }
}

/*********************************************************************************
【函    数】  LIS3DH_Init(void)
【概    述】  LIS3DH 初始化
```

```
【入口参数】  i2cHandle :  i2c 句柄
【返回参数】  无
【说    明】  阻塞模式
*****************************************************************************/
static void LIS3DH_Init(I2c_Handle * i2cHandle){
    uint8 id_temp = 0;
    LIS3DH_ReadReg(i2cHandle, LIS3DH_WHO_AM_I, &id_temp);   //读取 ID
    if(id_temp == 0x33){
        asm("nop");
    }
    LIS3DH_WriteReg(i2cHandle, LIS3DH_CTRL_REG1, 0x67);    //200hz
    LIS3DH_WriteReg(i2cHandle, LIS3DH_CTRL_REG2, 0x04);    //click fliter enable
    LIS3DH_WriteReg(i2cHandle, LIS3DH_CTRL_REG4, 0x10);    //4G
    LIS3DH_WriteReg(i2cHandle, LIS3DH_CTRL_REG4, 0x10);    //4G

    Util_startClock(&iicClock);
}

/*****************************************************************************
【函    数】  LIS3DH_Init_cb(I2c_Handle * i2cHandle)
【概    述】  LIS3DH 初始化
【入口参数】  i2cHandle：IIC 句柄
【返回参数】  无
【说    明】  回调模式
*****************************************************************************/
static void LIS3DH_Init_cb(I2c_Handle * i2cHandle){
    //回调方式初始化，写入第一个值，回调返回数据后再写入第二个……
    LIS3DH_ReadReg_cb(i2cHandle, LIS3DH_WHO_AM_I);
}

/*****************************************************************************
【函    数】  LIS3DH_ReadReg(I2c_Handle * i2cHandle, uint8 Reg, uint8 *Data)
【概    述】  读寄存器
【入口参数】  i2cHandle : i2c 句柄
              Reg       : 寄存器
              Data      : 数据
【返回参数】  无
【说    明】  阻塞模式
*****************************************************************************/
static uint8 LIS3DH_ReadReg(I2c_Handle * i2cHandle, uint8 Reg, uint8 *Data) {
    uint8 state;
    state = TTCDriverI2cRead(i2cHandle, SLAVEADDR, &Reg, Data,1,1,NULL);
```

```
    return state;
}

/*****************************************************************************
【函    数】  LIS3DH_ReadReg_cb(I2c_Handle * i2cHandle, uint8 Reg, uint8 *Data)
【概    述】  读寄存器
【入口参数】  i2cHandle : i2c 句柄
              Reg       : 寄存器
              Data      : 数据
【返回参数】  无
【说    明】  回调模式
*****************************************************************************/
static uint8 LIS3DH_ReadReg_cb(I2c_Handle * i2cHandle, uint8 Reg) {
    uint8 state;
    state = TTCDriverI2cRead(i2cHandle, SLAVEADDR,&Reg, NULL,1,1,&TTCiicArgcb);
    return state;
}

/*****************************************************************************
【函    数】  LIS3DH_WriteReg_cb(I2c_Handle * i2cHandle,uint8 Reg, uint8 Data)
【概    述】  写寄存器
【入口参数】  i2cHandle : i2c 句柄
              Reg       : 寄存器
              Data      : 数据
【返回参数】  无
【说    明】  回调模式
*****************************************************************************/
static uint8 LIS3DH_WriteReg_cb(I2c_Handle * i2cHandle,uint8 Reg, uint8 Data){
    uint8 state;
    uint8 txBuf[2];
    txBuf[0] = Reg;
    txBuf[1] = Data;
    state = TTCDriverI2cWrite(i2cHandle,SLAVEADDR,txBuf,2,&TTCiicArgcb);
    return state;
}


/*****************************************************************************
【函    数】  LIS3DH_WriteReg(I2c_Handle * i2cHandle,uint8 Reg, uint8 Data)
【概    述】  写寄存器
【入口参数】  i2cHandle : i2c 句柄
              Reg       : 寄存器
```

```
                Data        : 数据
【返回参数】    无
【说    明】    阻塞模式
*****************************************************************************/
static uint8 LIS3DH_WriteReg(I2c_Handle * i2cHandle,uint8 Reg, uint8 Data){
    uint8 state;
    uint8 txBuf[2];
    txBuf[0] = Reg;
    txBuf[1] = Data;
    state = TTCDriverI2cWrite(i2cHandle,SLAVEADDR,txBuf,2,NULL);
    return state;
}

/*****************************************************************************
【函    数】    LIS3DH_GetAccAxesRaw(I2c_Handle * i2cHandle, AxesRaw_t* buff)
【概    述】    读取 Gsensor 数据
【入口参数】    i2cHandle : i2c 句柄
                buff      : 三轴数据
【返回参数】    无
【说    明】    阻塞模式
*****************************************************************************/
uint8 LIS3DH_GetAccAxesRaw(I2c_Handle * i2cHandle, AxesRaw_t* buff){
    int16 value;
    uint8 *valueL = (uint8 *)(&value);
    uint8 *valueH = ((uint8 *)(&value)+1);

    if( TTCDRIVER_TRANSFER_DATA_SUCCESS != LIS3DH_ReadReg(i2cHandle, LIS3DH_OUT_X_L, valueL) )
        return MEMS_ERROR;

    if( TTCDRIVER_TRANSFER_DATA_SUCCESS != LIS3DH_ReadReg(i2cHandle, LIS3DH_OUT_X_H, valueH) )
        return MEMS_ERROR;

    buff->AXIS_X = value;

    if( TTCDRIVER_TRANSFER_DATA_SUCCESS != LIS3DH_ReadReg(i2cHandle, LIS3DH_OUT_Y_L, valueL) )
        return MEMS_ERROR;

    if( TTCDRIVER_TRANSFER_DATA_SUCCESS != LIS3DH_ReadReg(i2cHandle, LIS3DH_OUT_Y_H, valueH) )
        return MEMS_ERROR;

    buff->AXIS_Y = value;
```

```
    if( TTCDRIVER_TRANSFER_DATA_SUCCESS != LIS3DH_ReadReg(i2cHandle, LIS3DH_OUT_Z_L, valueL) )
        return MEMS_ERROR;

    if( TTCDRIVER_TRANSFER_DATA_SUCCESS != LIS3DH_ReadReg(i2cHandle, LIS3DH_OUT_Z_H, valueH) )
        return MEMS_ERROR;

    buff->AXIS_Z = value;

    IIcRxbuffer[1] = buff->AXIS_X>>8;
    IIcRxbuffer[2] = buff->AXIS_X&0xFF;
    IIcRxbuffer[3] = buff->AXIS_X>>8;
    IIcRxbuffer[4] = buff->AXIS_X&0xFF;
    IIcRxbuffer[5] = buff->AXIS_X>>8;
    IIcRxbuffer[6] = buff->AXIS_X&0xFF;
    return MEMS_SUCCESS;
}

/*****************************************************************************
【函    数】  LIS3DH_GetAccAxesRaw_cb(I2c_Handle * i2cHandle)
【概    述】  读取 Gsensor 数据
【入口参数】  i2cHandle : i2c 句柄
【返回参数】  无
【说    明】  回调模式
*****************************************************************************/
uint8 LIS3DH_GetAccAxesRaw_cb(I2c_Handle * i2cHandle) {
    //回调方式读 Gsensor 数据，先写入第一个地址，回调返回值后，再写入第二个地址……
    if( TTCDRIVER_TRANSFER_DATA_SUCCESS != LIS3DH_ReadReg_cb(i2cHandle,LIS3DH_OUT_X_L) )
        return MEMS_ERROR;
    return MEMS_SUCCESS;
}

/*****************************************************************************
【函    数】  LIS3DHI2cCB(TTCDriverI2cState_t i2cState,u8 * buffer,u16 len,void * arg)
【概    述】  IIC 读数据回调
【入口参数】  i2cState : i2c 传输状态
              buffer   : i2c 读取的数据
              arg      : i2c 传输前所携带的参数
【返回参数】  无
【说    明】  无
*****************************************************************************/
static void LIS3DHI2cCB(TTCDriverI2cState_t i2cState,u8 * buffer,u16 len,void * arg){
    if(i2cState == TTCDRIVER_I2C_TRANSFER_SUCCESS){
```

```
    LIS3DHiicCB *cb = (LIS3DHiicCB *)arg;
    if(cb&&cb->iiccb)
      cb->iiccb(cb->param,buffer,len);
  }else{

  }
}

/*****************************************************************************
【函    数】  SendRegiicCB(u32 param,u8 * buffer,u16 len)
【概    述】  回调方式---IIC 数据返回回调实体函数
【入口参数】  无
【返回参数】  无
【说    明】  无
*****************************************************************************/
void SendRegiicCB(u32 param,u8 * buffer,u16 len){
  switch(param){
    case 1:                                  //who am i
      if(buffer[0] == 0x33){
        TTCiicArgcb.param++;
        LIS3DH_WriteReg_cb(&i2cHandle, LIS3DH_CTRL_REG1, 0x67);     //200hz
      }
    break;
    case 2:
      TTCiicArgcb.param++;
      LIS3DH_WriteReg_cb(&i2cHandle, LIS3DH_CTRL_REG2, 0x04); //click fliter enable

    break;
    case 3:
      TTCiicArgcb.param++;
      LIS3DH_WriteReg_cb(&i2cHandle, LIS3DH_CTRL_REG4, 0x10);       //4G
    break;
    case 4:
      TTCiicArgcb.param++;
      LIS3DH_ReadReg_cb(&i2cHandle, LIS3DH_OUT_X_L);
    break;
    case 5:
      TTCiicArgcb.param++;
      IIcRxbuffer[1] = buffer[0];
      LIS3DH_ReadReg_cb(&i2cHandle, LIS3DH_OUT_X_H);
    break;
    case 6:
```

```
            TTCiicArgcb.param++;
            IIcRxbuffer[2] = buffer[0];
            LIS3DH_ReadReg_cb(&i2cHandle, LIS3DH_OUT_Y_L);
        break;
        case 7:
            TTCiicArgcb.param++;
            IIcRxbuffer[3] = buffer[0];
            LIS3DH_ReadReg_cb(&i2cHandle, LIS3DH_OUT_Y_H);
        break;
        case 8:
            TTCiicArgcb.param++;
            IIcRxbuffer[4] = buffer[0];
            LIS3DH_ReadReg_cb(&i2cHandle, LIS3DH_OUT_Z_L);
        break;
        case 9:
            TTCiicArgcb.param++;
            IIcRxbuffer[5] = buffer[0];
            LIS3DH_ReadReg_cb(&i2cHandle, LIS3DH_OUT_Z_H);
        break;
        case 10:
            TTCiicArgcb.param = 5;
            IIcRxbuffer[6] = buffer[0];
            IICDatatoUart(&IIcRxbuffer[1], 6);
            Util_restartClock(&iicClock,40);
        break;

        default:break;
    }
}

/*********************************************************************************
【函    数】  TTCDriverDemoI2CClockHandler(UArg arg)
【概    述】  标记事件并唤醒线程
【入口参数】  arg : 标记事件
【返回参数】  无
【说    明】  无
*********************************************************************************/
static void TTCDriverDemoI2CClockHandler(UArg arg){
  events |= SBP_IIC_EVT;
  Semaphore_post(*iicsem);
}
```

```
/*****************************************************************************
【函    数】  void IICDatatoUart(uint8 *Data,u16 len)
【概    述】  串口打印 IIC 读取到的数据
【入口参数】  无
【返回参数】  无
【说    明】
*****************************************************************************/
void IICDatatoUart(uint8 *Data,u16 len){
    if(len == 6){
        uint8 GSensorData[20]={0};

        GSensorData[4] = Data[1];
        GSensorData[3] = Data[0];
        GSensorData[6] = Data[3];
        GSensorData[5] = Data[2];
        GSensorData[8] = Data[5];
        GSensorData[7] = Data[4];

        GSensorData[0] = 0x88;
        GSensorData[1] = 0xA1;
        GSensorData[2] = 0x0E;
        GSensorData[17] = 0;

        for(uint8 i=0;i<17;i++){
            GSensorData[17] +=GSensorData[i];
        }
      #ifdef TTCDRIVER_UART
        TTCDriverUartWrite(&uartHandle,GSensorData,18);
      #endif //TTCDRIVER_UART
    }
}

#endif //TTCDRIVER_I2C
```

4.7.SPI 说明

SPI，是一种高速的，全双工，同步的通信总线，并且在芯片的管脚上只占用四根线，节约了芯片的管脚，同时为 PCB 的布局上节省空间，提供方便，正是出于这种简单易用的特性，如今越来越多的芯片集成了这种通信协议。SDK 中 SPI 最高速率可达 4Mbps.

4.7.1 SPI API 说明

4.7.1.1 TTCDriverSpiInitDefaultParam()

```
/*******************************************************************************
【函    数】  TTCDriverSpiInitDefaultParam(Spi_Handle * spiHandle)
【概    述】  设置 SPI 句柄默认参数
【入口参数】  spiHandle : SPI 句柄
【返回参数】  无
【说    明】  无
*******************************************************************************/
extern void TTCDriverSpiInitDefaultParam(Spi_Handle * spiHandle);
```

4.7.1.2 TTCDriverSpiInit()

```
/*******************************************************************************
【函    数】  TTCDriverInfo_t TTCDriverSpiInit(TTCSdkSpiCB_t appCallbacks,
                                               Spi_Handle * spiHandle,
                                               TTCDriverSpiParams_t param)
【概    述】  初始化 SPI
【入口参数】  appCallbacks : 注册回调
              spiHandle    : SPI 句柄
              param        : SPI 参数
【返回参数】  请参考 TTCDriverInfo_t
【说    明】  若初始化失败 SPI 会返回初始化失败原因。
              注意：若 SPI 为回调模式则必须设置回调函数，否则初始化失败
                    若 SPI 为阻塞模式则无需设置回调函数
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverSpiInit(TTCSdkSpiCB_t appCallbacks,
                                        Spi_Handle * spiHandle,
                                        TTCDriverSpiParams_t param);
回调函数格式：
typedef void (*TTCSdkSpiCB_t)(SPI_Status spiState,u8 * buffer,u16 len,void * arg);
```

4.7.1.3 TTCDriverSpiRead()

```
/*****************************************************************************
【函    数】  TTCDriverSpiRead(Spi_Handle * spiHandle,
                               u8* rBuf,
                               u16 len,
                               void * arg)
【概    述】  SPI 读取数据
【入口参数】  spiHandle : SPI 句柄
              rBuf      : 读数据缓存
              len       : 数据长度
              arg       : 回调模式下传输前传入的参数，将会传递到回调函数中
【返回参数】  请参考 TTCDriverInfo_t
【说    明】  适用于 SPI Master
              注意：回调模式下接收到的数据在回调中获取
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverSpiRead(Spi_Handle * spiHandle,
                                        u8* rBuf,
                                        size_t len,
                                        void * arg);
```

4.7.1.4 TTCDriverSpiWrite()

```
/*****************************************************************************
【函    数】  TTCDriverSpiWrite(Spi_Handle spiHandle,
                                u8 *wBuf,
                                size_t len,
                                void * arg)
【概    述】  向 SPI TX 缓存写入数据并发送
【入口参数】  spiHandle : SPI 句柄
              wBuf      : 写数据缓存
              len       : 数据长度
              arg       : 回调模式下传输前传入的参数，将会传递到回调函数中
【返回参数】  请参考 TTCDriverInfo_t
【说    明】  适用于 SPI Master 和 Slave 模式
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverSpiWrite(Spi_Handle * spiHandle,
                                         u8 *wBuf,
                                         size_t len,
                                         void * arg);
```

4.7.1.5 TTCDriverSpiWriteRead()

```
/*****************************************************************************
【函    数】 TTCDriverSpiWriteRead(Spi_Handle * spiHandle,
                                   uint8_t *buf,
                                   size_t wLen,
                                   size_t rLen,
                                   void * arg)
【概    述】 SPI 写读取数据
【入口参数】 spiHandle : SPI 句柄
             buf       : 读写数据缓存
             wLen      : 写数据长度
             rLen      : 读数据长度
             arg       : 回调模式下传输前传入的参数，将会传递到回调函数中
【返回参数】 请参考 TTCDriverInfo_t
【说    明】 适用于 SPI Master 模式
             注意：1.buf 缓存长度不应小于 wLen + rLen
                   2.回调模式下接收到的数据在回调中获取
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverSpiWriteRead(Spi_Handle * spiHandle,
                                             uint8_t *buf,
                                             size_t wLen,
                                             size_t rLen,
                                             void * arg);
```

4.7.1.6 TTCDriverSPIBusy()

```
/*****************************************************************************
【函    数】 TTCDriverSPIBusy(void)
【概    述】 判断 SPI 传输是否忙
【入口参数】 无
【返回参数】 true : 忙状态
             false : 空闲
【说    明】 读写大量数据时需要调用此函数。
*****************************************************************************/
extern bool TTCDriverSPIBusy(void);
```

4.7.2 SPI 使用示例

```
/*****************************************************************************
【文    件】 TTCDriverSPIDemo.c
```

【概　　述】　TTC SDK SPI 示例代码
【编　　写】　SDK 工作小组
【修　　订】　SDK 工作小组
【修订日期】　2016/11/29
【版　　本】　V1.0.0
【说　　明】　本示例代码基于 LIS3DH 的 SPI 模式调试

功能说明：

此文件使用 2640 的 SDK 的 SPI 驱动为基础，对 Gsensor LIS3DH 进行相关配置并采集原始数据，原始数据再通过串口打印，串口的数据格式已经做了处理，可以用匿名四轴上位机进行查看实时波形。

增加步骤：

1.打开相应的宏定义 TTCDRIVER_SPI，需要用到串口时还需要打开宏 TTCDRIVER_UART

2.在主线程初始化中的驱动示例初始化 TTCSDKDriverInit();中增加 SPI 初始化配置 TTCDriverDemoSPIInit(&sem,&appMsgQueue);

3.在主线程处理函数中增加 SPI 内部事件处理 TTCDriverSpiEvent();和 DEMO SPI 数据处理 TTCSDKDriverDemoSPIEvent();

4.脚位的定义在 TTCSDKBoard.h 中

```
*********************************************************************************/
#ifdef TTCDRIVER_SPI
/*********************************************************************
 * 头文件
 */
#include <string.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Queue.h>
#include <ti/drivers/PIN.h>
#include <ti/drivers/pin/PINCC26XX.h>
#include <ti/drivers/SPI.h>
#include <ti/drivers/spi/SPICC26XXDMA.h>
#include <driverlib/aux_wuc.h>
#include "hci_tl.h"
#include "gatt.h"
#include "gapgattserver.h"
#include "gattservapp.h"
#include "gapbondmgr.h"
#include "osal_snv.h"
#include "ICallBleAPIMSG.h"
#include "util.h"
```

```
#include "TTCSDKBoard.h"
#include "TTCBleSDKConfig.h"
#include "TTCBleDevInfoService.h"
#include "TTCBlePeripheral.h"
#include "TTCBlePeripheralProcess.h"
#include "TTCBleProfile.h"
#include "TTCBleSDKManager.h"
#ifdef TTCDRIVER_UART
#include <ti/drivers/UART.h>
#include <ti/drivers/uart/UARTCC26XX.h>
#include "TTCDriverUART.h"
#include "TTCDriverUARTDemo.h"
#endif //TTCDRIVER_UART
#include "TTCDriverSPI.h"
#include "TTCDriverSPIDemo.h"


/*********************************************************************
* 常量及宏定义
*/
//Register Definition
#define LIS3DH_WHO_AM_I                     0x0F  // device identification register

// CONTROL REGISTER 1
#define LIS3DH_CTRL_REG1                0x20
#define LIS3DH_ODR_BIT                      BIT(4)
#define LIS3DH_LPEN                     BIT(3)
#define LIS3DH_ZEN                  BIT(2)
#define LIS3DH_YEN                  BIT(1)
#define LIS3DH_XEN                  BIT(0)

//CONTROL REGISTER 2
#define LIS3DH_CTRL_REG2                0x21
#define LIS3DH_HPM                      BIT(6)
#define LIS3DH_HPCF                     BIT(4)
#define LIS3DH_FDS                  BIT(3)
#define LIS3DH_HPCLICK                  BIT(2)
#define LIS3DH_HPIS2                    BIT(1)
#define LIS3DH_HPIS1                    BIT(0)

//CONTROL REGISTER 3
#define LIS3DH_CTRL_REG3                0x22
```

```
#define LIS3DH_I1_CLICK                 BIT(7)
#define LIS3DH_I1_AOI1                  BIT(6)
#define LIS3DH_I1_AOI2                  BIT(5)
#define LIS3DH_I1_DRDY1                 BIT(4)
#define LIS3DH_I1_DRDY2                 BIT(3)
#define LIS3DH_I1_WTM                   BIT(2)
#define LIS3DH_I1_ORUN                  BIT(1)

//CONTROL REGISTER 6
#define LIS3DH_CTRL_REG6                0x25
#define LIS3DH_I2_CLICK                 BIT(7)
#define LIS3DH_I2_INT1                  BIT(6)
#define LIS3DH_I2_BOOT                      BIT(4)
#define LIS3DH_H_LACTIVE                BIT(1)

//TEMPERATURE CONFIG REGISTER
#define LIS3DH_TEMP_CFG_REG             0x1F
#define LIS3DH_ADC_PD                   BIT(7)
#define LIS3DH_TEMP_EN                  BIT(6)

//CONTROL REGISTER 4
#define LIS3DH_CTRL_REG4            0x23
#define LIS3DH_BDU                  BIT(7)
#define LIS3DH_BLE                  BIT(6)
#define LIS3DH_FS                   BIT(4)
#define LIS3DH_HR                   BIT(3)
#define LIS3DH_ST                   BIT(1)
#define LIS3DH_SIM                  BIT(0)

//CONTROL REGISTER 5
#define LIS3DH_CTRL_REG5            0x24
#define LIS3DH_BOOT                         BIT(7)
#define LIS3DH_FIFO_EN                      BIT(6)
#define LIS3DH_LIR_INT1                     BIT(3)
#define LIS3DH_D4D_INT1                     BIT(2)

//REFERENCE/DATA_CAPTURE
#define LIS3DH_REFERENCE_REG                    0x26
#define LIS3DH_REF                  BIT(0)

//STATUS_REG_AXIES
#define LIS3DH_STATUS_REG           0x27
```

```
#define LIS3DH_ZYXOR                    BIT(7)
#define LIS3DH_ZOR                      BIT(6)
#define LIS3DH_YOR                      BIT(5)
#define LIS3DH_XOR                      BIT(4)
#define LIS3DH_ZYXDA                    BIT(3)
#define LIS3DH_ZDA                      BIT(2)
#define LIS3DH_YDA                      BIT(1)
#define LIS3DH_XDA                      BIT(0)

//STATUS_REG_AUX
#define LIS3DH_STATUS_AUX               0x07

//INTERRUPT 1 CONFIGURATION
#define LIS3DH_INT1_CFG                 0x30
#define LIS3DH_ANDOR                    BIT(7)
#define LIS3DH_INT_6D                   BIT(6)
#define LIS3DH_ZHIE                     BIT(5)
#define LIS3DH_ZLIE                     BIT(4)
#define LIS3DH_YHIE                     BIT(3)
#define LIS3DH_YLIE                     BIT(2)
#define LIS3DH_XHIE                     BIT(1)
#define LIS3DH_XLIE                     BIT(0)

//FIFO CONTROL REGISTER
#define LIS3DH_FIFO_CTRL_REG            0x2E
#define LIS3DH_FM                       BIT(6)
#define LIS3DH_TR                       BIT(5)
#define LIS3DH_FTH                      BIT(0)

//OUTPUT REGISTER
#define LIS3DH_OUT_X_L                  0x28
#define LIS3DH_OUT_X_H                  0x29
#define LIS3DH_OUT_Y_L                  0x2A
#define LIS3DH_OUT_Y_H                  0x2B
#define LIS3DH_OUT_Z_L                  0x2C
#define LIS3DH_OUT_Z_H                  0x2D

//INT1 REGISTERS
#define LIS3DH_INT1_THS                 0x32
#define LIS3DH_INT1_DURATION            0x33

//INTERRUPT 1 SOURCE REGISTER
```

```
#define LIS3DH_INT1_SRC                 0x31

#define TXBUF_LEN                                   20
#define RXBUF_LEN                                   20

#define SBP_SPI_EVT                                 0x0001

#define Board_SPI0_LISEDH_CSN_ON                    0
#define Board_SPI0_LISEDH_CSN_OFF                   1

#define Board_SPI0_LISEDH_CSN                       IOID_0

typedef enum {
    MEMS_SUCCESS                            =       0x01,
    MEMS_ERROR                              =       0x00
} status_t;

typedef enum {
    MEMS_ENABLE                             =       0x01,
    MEMS_DISABLE                =       0x00
} State_t;

typedef struct {
    int16 AXIS_X;
    int16 AXIS_Y;
    int16 AXIS_Z;
} AxesRaw_t;


extern const SPICC26XX_HWAttrs  spiCC26XXDMAHWAttrs[];

const TTCDriverSpiParams_t param = {
    .spiName = CC2650_SPI0,
    .spiMode = SPI_MASTER,
    .bitRate = 400000,
    .dataSize = 8,
    .frameFormat = SPI_POL0_PHA0,                                   //SPI 的方式
要与通讯对象的方式一致
    .transferMode                       =                           SPI_MODE_BLOCKING,
//SPI_MODE_CALLBACK, //
    .slaveCSPin = PIN_UNASSIGNED,
    .spiTransferBufLen = 0,
```

```
    .spiHWAttr = &spiCC26XXDMAHWAttrs[CC2650_SPI0],                    //CC2650_SPI0
索引号要与所配置的 IO 对应
};

/*********************************************************************
 * 本地变量
 */
PIN_Config SPIcsPinConfig[] = {
    Board_SPI0_LISEDH_CSN | PIN_GPIO_OUTPUT_EN | PIN_GPIO_HIGH | PIN_PUSHPULL | PIN_DRVSTR_MIN,
/* Ext. flash chip select */
    PIN_TERMINATE
};
#ifdef TTCDRIVER_UART
extern Uart_Handle uartHandle;
#endif //TTCDRIVER_UART


static TTCBleSDKManagerInfo_t CSNstate;
static PIN_Handle hspiPin = NULL;
static PIN_State pinState;
static Spi_Handle spiHandle;
static ICall_Semaphore *spisem;
static TTCDriverInfo_t SPI_state = TTCDRIVER_INFO_SIZE;
static AxesRaw_t Acc_data;

static Clock_Struct spiClock;
static u16 events;

#define                                                    SPILis3dhSelect()
TTCDriverIOSetOutputVaule(&hspiPin,Board_SPI0_LISEDH_CSN,Board_SPI0_LISEDH_CSN_ON)
#define                                                     SPILis3dhClose()
TTCDriverIOSetOutputVaule(&hspiPin,Board_SPI0_LISEDH_CSN,Board_SPI0_LISEDH_CSN_OFF)


/*********************************************************************
 * 本地函数声明
 */
static void TTCSdkDriverSpiCB(SPI_Status spiState,u8 * buffer,u16 len,void * arg);
static void SPIDatatoUart(uint8 *Data,u16 len);
static void LIS3DH_Init_SPI(Spi_Handle * spiHandle);
static void LIS3DH_Init_SPI_cb(Spi_Handle * spiHandle);
static uint8 LIS3DH_ReadReg_SPI(Spi_Handle * spiHandle, uint8 Reg, uint8 *Data);
```

```
static uint8 LIS3DH_ReadReg_SPI_cb(Spi_Handle * spiHandle, uint8 Reg);
static uint8 LIS3DH_WriteReg_SPI(Spi_Handle * spiHandle,uint8 Reg, uint8 Data);
static uint8 LIS3DH_WriteReg_SPI_cb(Spi_Handle * spiHandle, uint8 Reg ,uint8 Data);
static void LIS3DH_GetAccAxesRaw_SPI(Spi_Handle * spiHandle, AxesRaw_t* buff);
static void TTCDriverDemoSPIClockHandler(UArg arg);
static void LIS3DH_GetAccAxesRaw_SPI_cb(void);
static void SendRegCB(u32 param,u8 * buffer,u16 len);

/*********************************************************************
 * 回调函数
 */
LIS3DHspiCB TTCSpiArgcb = {
  .param = 1,
  .spicb = &SendRegCB,
};

TTCSdkSpiCB_t TTCBlePeripheralSPICB = {
  TTCSdkDriverSpiCB
};

/*****************************************************************************
【函    数】  TTCDriverDemoI2CInit(ICall_Semaphore *sem,Queue_Handle *queueHandle)
【概    述】  SPI 初始化及参数设置
【入口参数】  sem          ：信号量
              appMsgQueue ：消息句柄
【返回参数】  无
【说    明】  无
*****************************************************************************/
void TTCDriverDemoSPIInit(ICall_Semaphore *sem,Queue_Handle *queueHandle){
    CSNstate = TTCDriverIOOpen(&hspiPin, &pinState,SPIcsPinConfig);
    TTCDriverSpiInitDefaultParam(&spiHandle);
    spiHandle.sem                              = sem;
    spisem                                    = sem;
    spiHandle.queueHandle                      = queueHandle;

    const TTCDriverSpiParams_t spiParam = {
        .spiName = CC2650_SPI0,
        .spiMode = SPI_MASTER,
        .bitRate = 400000,
        .dataSize = 8,
        .frameFormat = SPI_POL0_PHA0,
        .transferMode = SPI_MODE_CALLBACK,//SPI_MODE_BLOCKING,
```

```
        .slaveCSPin = PIN_UNASSIGNED,
        .spiTransferBufLen = 0,
        .spiHWAttr = &spiCC26XXDMAHWAttrs[CC2650_SPI0],
    };
    SPI_state = TTCDriverSpiInit(TTCBlePeripheralSPICB,&spiHandle,spiParam);     //SPI 初始化,
注意判断初始化是否成功

    Util_constructClock(&spiClock,
                        TTCDriverDemoSPIClockHandler,
                        1000,
                        40,
                        false,
                        SBP_SPI_EVT);                                            //构建 SPI 定
时采 Gsensor 数据事件

    if(param.transferMode == SPI_MODE_CALLBACK){
        LIS3DH_Init_SPI_cb(&spiHandle);                                          //阻塞模式
    }else{
        LIS3DH_Init_SPI(&spiHandle);                                             //回调模式
    }
}

/*********************************************************************************
【函    数】  TTCSDKDriverDemoSPIEvent(void)
【概    述】  读取 Gsensor 数据事件
【入口参数】  无
【返回参数】  无
【说    明】
*********************************************************************************/
void TTCSDKDriverDemoSPIEvent(void){
    if(events & SBP_SPI_EVT){
        events &= ~SBP_SPI_EVT;
        if(param.transferMode == SPI_MODE_CALLBACK){
            LIS3DH_GetAccAxesRaw_SPI_cb();                                       //回调方
式采样
            Util_stopClock(&spiClock);
        }else{
            LIS3DH_GetAccAxesRaw_SPI(&spiHandle,&Acc_data);                      //阻塞方
式采样
        }
    }
}
```

```
/*****************************************************************************
【函    数】  SPIDatatoUart(uint8 *Data,u16 len)
【概    述】  串口打印 SPI 读取到的数据
【入口参数】  无
【返回参数】  无
【说    明】
*****************************************************************************/
static void SPIDatatoUart(uint8 *Data,u16 len){
    if(len == 7){
        uint8 GSensorData[20]={0};

        GSensorData[4] = Data[2];
        GSensorData[3] = Data[1];
        GSensorData[6] = Data[4];
        GSensorData[5] = Data[3];
        GSensorData[8] = Data[6];
        GSensorData[7] = Data[5];

        GSensorData[0] = 0x88;
        GSensorData[1] = 0xA1;
        GSensorData[2] = 0x0E;
        GSensorData[17] = 0;

        for(uint8 i=0;i<17;i++){
            GSensorData[17] +=GSensorData[i];
        }
#ifdef TTCDRIVER_UART
        TTCDriverUartWrite(&uartHandle,GSensorData,18);
#endif //TTCDRIVER_UART
    }
}


/*****************************************************************************
【函    数】  LIS3DH_Init_SPI(Spi_Handle * spiHandle)
【概    述】  LIS3DH 初始化
【入口参数】  无
【返回参数】  无
【说    明】  阻塞方式
*****************************************************************************/
static void LIS3DH_Init_SPI(Spi_Handle * spiHandle){
```

```
    uint8 id_temp = 0;
    SPILis3dhSelect();
    LIS3DH_ReadReg_SPI(spiHandle, LIS3DH_WHO_AM_I, &id_temp);
    SPILis3dhClose();
    if(id_temp == 0x33){
        asm("nop");
    }
    SPILis3dhSelect();
    LIS3DH_WriteReg_SPI(spiHandle, LIS3DH_CTRL_REG1, 0x67);                    //200hz
LOWPOWER_MODE X_EN_Y_EN_Z_EN
    SPILis3dhClose();

    SPILis3dhSelect();
    LIS3DH_WriteReg_SPI(spiHandle, LIS3DH_CTRL_REG2, 0x04);                    //click
fliter enable
    SPILis3dhClose();
    SPILis3dhSelect();
    LIS3DH_WriteReg_SPI(spiHandle, LIS3DH_CTRL_REG4, 0x10);                  //4G
    SPILis3dhClose();

    Util_startClock(&spiClock);

}

/*********************************************************************************
【函    数】  LIS3DH_ReadReg(uint8 Reg, uint8* Data)
【概    述】  读寄存器
【入口参数】  无
【返回参数】  无
【说    明】  阻塞方式
*********************************************************************************/
static uint8 LIS3DH_ReadReg_SPI(Spi_Handle * spiHandle, uint8 Reg, uint8 *Data) {
    uint8 state;
    uint8 txBuf[2];
    txBuf[0] = Reg|0X80;
    state = TTCDriverSpiWriteRead(spiHandle,txBuf, 1, 1, NULL);                //写入地址,
返回有效值
    *Data = txBuf[1];                                                          //txBuf[0]返
回为 0xFF,有效值存放在 txBuf[1]
    return state;
}
```

```
/*****************************************************************************
【函    数】  LIS3DH_WriteReg_SPI(Spi_Handle * spiHandle,uint8 Reg, uint8 Data)
【概    述】  写寄存器
【入口参数】  无
【返回参数】  无
【说    明】  阻塞方式
*****************************************************************************/
static uint8 LIS3DH_WriteReg_SPI(Spi_Handle * spiHandle,uint8 Reg, uint8 Data){
    uint8 state;
    uint8 txBuf[2];
    txBuf[0] = Reg|0X80;
    txBuf[1] = Data;
    state = TTCDriverSpiWrite(spiHandle,txBuf,2, NULL);                    //把寄存器的
地址和要操作的值放入缓存发送出去
    return state;
}

/*****************************************************************************
【函    数】  LIS3DH_GetAccAxesRaw(AxesRaw_t* buff)
【概    述】  读取 Gsensor 数据
【入口参数】  无
【返回参数】  无
【说    明】  阻塞方式
*****************************************************************************/
static void LIS3DH_GetAccAxesRaw_SPI(Spi_Handle * spiHandle, AxesRaw_t* buff){
    uint8 spi_rx_buf[7] = {0};

    SPILis3dhSelect();
    spi_rx_buf[0] = 0xC0|LIS3DH_OUT_X_L;
    TTCDriverSpiWriteRead(spiHandle,spi_rx_buf, 1, 6, NULL);

    buff->AXIS_X = spi_rx_buf[2]<<8|spi_rx_buf[1];
    buff->AXIS_Y = spi_rx_buf[4]<<8|spi_rx_buf[3];
    buff->AXIS_Z = spi_rx_buf[6]<<8|spi_rx_buf[5];
    SPILis3dhClose();

    buff->AXIS_X = spi_rx_buf[2]<<8|spi_rx_buf[1];
    buff->AXIS_Y = spi_rx_buf[4]<<8|spi_rx_buf[3];
    buff->AXIS_Z = spi_rx_buf[6]<<8|spi_rx_buf[5];

    SPIDatatoUart(spi_rx_buf,7);
}
```

```
/*****************************************************************************
【函    数】  TTCSdkDriverSpiCB(SPI_Status spiState,u8 * buffer,u16 len,void * arg)
【概    述】  SPI 数据返回回调
【入口参数】  无
【返回参数】  无
【说    明】  回调方式
*****************************************************************************/
static void TTCSdkDriverSpiCB(SPI_Status spiState,u8 * buffer,u16 len,void * arg){
    SPILis3dhClose();
    LIS3DHspiCB *cb = (LIS3DHspiCB *)arg;
    if(cb&&cb->spicb){
      cb->spicb(cb->param,buffer,len);
    }
}


/*****************************************************************************
【函    数】  LIS3DH_Init_SPI_cb(Spi_Handle * spiHandle)
【概    述】  LIS3DH 初始化
【入口参数】  无
【返回参数】  无
【说    明】  回调方式
*****************************************************************************/
static void LIS3DH_Init_SPI_cb(Spi_Handle * spiHandle){
    SPILis3dhSelect();
    LIS3DH_ReadReg_SPI_cb(spiHandle, LIS3DH_WHO_AM_I);                    //先写入第一
个值，回调后再写第二个……
}

/*****************************************************************************
【函    数】  LIS3DH_ReadReg_SPI_cb(Spi_Handle * spiHandle, uint8 Reg)
【概    述】  读寄存器
【入口参数】  无
【返回参数】  无
【说    明】  回调方式
*****************************************************************************/
uint8 LIS3DH_ReadReg_SPI_cb(Spi_Handle * spiHandle, uint8 Reg) {
    uint8 state;
```

```
    uint8 txBuf[2];
    txBuf[0] = Reg|0X80;
    state = TTCDriverSpiWriteRead(spiHandle,txBuf, 1, 1,  &TTCSpiArgcb);        //写入地址，
在回调函数返回有效值

    return state;
}

/*****************************************************************************
【函    数】  uint8 LIS3DH_WriteReg_SPI_cb(Spi_Handle * spiHandle,uint8 Reg, uint8 Data)
【概    述】  写寄存器
【入口参数】  无
【返回参数】  无
【说    明】  回调方式
*****************************************************************************/
uint8 LIS3DH_WriteReg_SPI_cb(Spi_Handle * spiHandle, uint8 Reg ,uint8 Data){
    uint8 state;
    uint8 txBuf[2];
    txBuf[0] = Reg|0X80;
    txBuf[1] = Data;
    state = TTCDriverSpiWrite(spiHandle,txBuf,2, &TTCSpiArgcb);                    //写寄存
器时不需要返回值
    //把寄存器的地址和要操作的值放入缓存发送出去

    return state;
}

/*****************************************************************************
【函    数】  LIS3DH_GetAccAxesRaw_SPI_EVT(void)
【概    述】  读取 Gsensor 数据
【入口参数】  无
【返回参数】  无
【说    明】  回调方式
*****************************************************************************/
void LIS3DH_GetAccAxesRaw_SPI_cb(void){
    uint8 spi_rx_buf[7] = {0};
    SPILis3dhSelect();
    spi_rx_buf[0] = 0xC0|LIS3DH_OUT_X_L;                    //0xC0 为连续读指令
    TTCDriverSpiWriteRead(&spiHandle,spi_rx_buf, 1, 6, &TTCSpiArgcb);
}

/*****************************************************************************
```

【函　　数】　SendRegCB(u32 param,u8 * buffer,u16 len)
【概　　述】　SPI 数据返回回调实体函数
【入口参数】　无
【返回参数】　无
【说　　明】
***/
```
static void SendRegCB(u32 param,u8 * buffer,u16 len){
    switch(param){
        case 1:                                                         //who am i
            if(buffer[1] == 0x33){
                TTCSpiArgcb.param++;
                SPILis3dhSelect();
                LIS3DH_WriteReg_SPI_cb(&spiHandle,  LIS3DH_CTRL_REG1,  0x67);         //200hz
LOWPOWER_MODE X_EN_Y_EN_Z_EN
            }
        break;
        case 2:
            TTCSpiArgcb.param++;
            SPILis3dhSelect();
            LIS3DH_WriteReg_SPI_cb(&spiHandle,  LIS3DH_CTRL_REG2,  0x04);             //click
fliter enable
        break;
        case 3:
            TTCSpiArgcb.param++;
            SPILis3dhSelect();
            LIS3DH_WriteReg_SPI_cb(&spiHandle, LIS3DH_CTRL_REG4, 0x10);        //4G
        break;
        case 4:
            TTCSpiArgcb.param = 4;
            SPIDatatoUart(buffer, len);
            Util_restartClock(&spiClock,40);
        break;
        default:break;
    }
}

/*****************************************************************************
```
【函　　数】　TTCDriverDemoI2CClockHandler(UArg arg)
【概　　述】　标记事件并唤醒线程
【入口参数】　arg ： 标记事件
【返回参数】　无
【说　　明】　无

```
*****************************************************************************/
static void TTCDriverDemoSPIClockHandler(UArg arg){
    events |= SBP_SPI_EVT;
    Semaphore_post(*spisem);
}

#endif //TTCDRIVER_SPI
```

4.8. Wechat 说明

1.广播格式要求

1) 1.GAP_ADTYPE_16BIT_MORE 类型下的微信服务 UUID: 0xFEE7 不允许更改.

2)厂商标识符的最后需要预留 2 个字节用于填写公司 ID 以及 6 个字节的设备 MAC 地址详情见 DEMO.

2.微信蓝牙初始化

```
TTCBleWechatParam_t wechatParam= {
.ramSize=50,//发送缓存大小(不得小于 30)
.authSelect=AUTH_MAC_NOENCRYPT, //登录方式选择
.dataDirSelect=DATA_DIR_BG,//数据方向
.deviceMac={0x24,0x71,0x89,0xff,0x00,0x02}, //设备 MAC 地址
};

TTCBleWechatInit(&TTCBlePeripheralTaskCls,
                 &sem,
                 &appMsgQueue,
                 &selfEntity,
                  wechatParam);
```

3.添加微信蓝牙处理

在 static void TTCBlePeripheralTaskFxn(UArg a0, UArg a1)中添加微信事件处理，在 ICall_Errno errno = ICall_wait(ICALL_TIMEOUT_FOREVER);后添加

```
#ifdef TTCBLE_WECHAT
   TTCBleWechatEvent();
#endif //TTCBLE_WECHAT
```

在 static void TTCBlePeripheralTaskProcessAppMsg(TTCMsg_t *pMsg)中添加代码

```
case TTCSDK_MSG_BLE_WECHAT_EVENT:{
     TTCBleWechatProcess((TTCMsg_t *)pMsg);
  }break;
```

添加函数：

注意：以下使用了 UART 功能，如需使用请参照 TTCDriverUART.h 驱动文件添加 UART 驱动.

4.8.1 Wechat API 说明

4.8.1.1 TTCBleWechatEvent()

```
/*****************************************************************************
【函    数】 TTCBleWechatEvent(void
【概    述】 微信事件处理
【入口参数】 无
```

【返回参数】 无
【说　　明】 无
*****/
extern void TTCBleWechatEvent(void);

4.8.1.2 TTCBleWechatInit()

```
/*****************************************************************************
【函　　数】 TTCBleWechatInit(TTCSdkClass_t *appCallbacks,
                              ICall_Semaphore *sem,
                              Queue_Handle *queueHandle,
                              ICall_EntityID *selfEntity,
                              TTCBleWechatParam_t param)
【概　　述】 初始化微信
【入口参数】 appCallbacks : 回调函数
             sem          : 信号量
             queueHandle  : 消息句柄
             selfEntity   : ID
             param        : 参数
【返回参数】 无
【说　　明】 无
*****************************************************************************/
extern TTCDriverInfo_t TTCBleWechatInit(TTCSdkClass_t *appCallbacks,
                                        ICall_Semaphore *sem,
                                        Queue_Handle *queueHandle,
                                        ICall_EntityID *selfEntity,
                                        TTCBleWechatParam_t param);
```

4.8.1.3 TTCBleWechatSend()

```
/*****************************************************************************
【函　　数】 TTCBleWechatSend(u8 *sendData,u16 len)
【概　　述】 发送数据到微信
【入口参数】 sendData : 发送数据
             len : 数据长度
【返回参数】 失败原因请参考 TTCDriverInfo_t
【说　　明】 无
*****************************************************************************/
extern TTCDriverInfo_t TTCBleWechatSend(u8 *sendData,u16 len);
```

4.8.2 Wechat 使用示例

```
/*****************************************************************************
【文　　件】　TTCDriverWechatDemo.c
【概　　述】　TTC SDK Wechat 示例代码
【编　　写】　SDK 工作小组
【修　　订】　SDK 工作小组
【修订日期】　2016/12/02
【版　　本】　V1.0.0
【说　　明】

功能说明：
　　初始化时对广播格式按照微信外设协议进行装载，确保可以被微信公众号发现；相应的 API 接口进行向微信公从号发送数据和从微信公众号接收数据。用户写入数据时对底层对数据进行微信要求格式封包，发送到微信后自动解包为原始数据，实现数据的透传。微信向设备写数据，设备底层对数据进行解包后传到用户层 API，用户从缓存接收数据即可。

 添加步骤：
　　1.打开相应的宏定义 TTCBLE_WECHAT，需要用到串口时还需要打开宏 TTCDRIVER_UART
　　2.在主线程初始化中的驱动示例初始化 TTCSDKDriverInit();中增加微信初始化配置 TTCDriverDemoWechatInit(&TTCBlePeripheralTaskCls,&sem,&appMsgQueue,&selfEntity);
　　3.在主线程处理函数中增加 微信内部事件处理 TTCBleWechatEvent();和 DEMO 微信数据处理 TTCSDKDriverDemoWechatEvent();
　　4.在线程消息处理函数 TTCBlePeripheralTaskProcessAppMsg(TTCMsg_t *pMsg)的 TTCSDK_MSG_GET_BLE_STATE_EVENT 消息中增加微信状态处理函数 TTCBleWechatStateClear();和 TTCDriverDemoWechatReadToSendStateSet(false);
　　5.在线程消息处理函数 TTCBlePeripheralTaskProcessAppMsg(TTCMsg_t *pMsg)的 TTCSDK_MSG_BLE_WECHAT_EVENT 消息中添加处理微信消息函数 TTCBleWechatProcess(TTCMsg_t * TTCMsg)

*****************************************************************************/
#ifdef TTCBLE_WECHAT
/*****************************************************************************
 * 头文件
 */
#include <string.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/knl/Queue.h>
#include <ti/drivers/PIN.h>
#include <ti/drivers/UART.h>
#include <ti/drivers/uart/UARTCC26XX.h>
#include <ti/drivers/pin/PINCC26XX.h>
```

```
#include <driverlib/aux_wuc.h>
#include "hci_tl.h"
#include "gatt.h"
#include "gapgattserver.h"
#include "gattservapp.h"
#include "gapbondmgr.h"
#include "osal_snv.h"
#include "ICallBleAPIMSG.h"
#include "util.h"
#include "TTCSDKBoard.h"
#include "TTCBleSDKConfig.h"
#include "TTCBleDevInfoService.h"
#include "TTCBlePeripheral.h"
#include "TTCBlePeripheralProcess.h"
#include "TTCBleProfile.h"
#include "TTCBleSDKManager.h"

#ifdef TTCDRIVER_UART
#include <ti/drivers/UART.h>
#include <ti/drivers/uart/UARTCC26XX.h>
#include "TTCDriverUART.h"
#include "TTCDriverUARTDemo.h"
#endif //TTCDRIVER_UART

#include "TTCBleWechat.h"
#include "TTCDriverWechatDemo.h"

/*********************************************************************
 * 常量及宏定义
 */
#define SBP_WECHAT_SEND_EVT             0x0001
#define SBP_WECHAT_RECEIVED_EVT         0x0002


#define ADVERT_MAC_ADDR                 23
//广播包中 MAC 地址的起始位置，实际应用时要与广播包中的 MAC 地址字体相匹配
#define WECHAT_SERV_UUID                0xFEE7          //微信 UUID
#define DEFAULT_DISCOVERABLE_MODE_wechat            GAP_ADTYPE_FLAGS_GENERAL

/*********************************************************************
 * 本地变量
 */
```

```
#ifdef TTCDRIVER_UART
extern Uart_Handle uartHandle;
#endif //TTCDRIVER_UART
Clock_Struct WechatClock;
static ICall_Semaphore *Wechatsem;
static u16 events;

Wechat_t Wechat={
  {0},
  0,
  {0},
  0,
  {0x7A,0x99,0x07,0xE5,0xA6,0x44},
  {0,0,0,0,0}
};

static u8 WechatadvertData[] = {      //广播数据，最大不超过 31 个字节
 //注意：IOS 自定义厂商标识符数据（GAP_ADTYPE_MANUFACTURER_SPECIFIC）不超过 18 个字节
      0x02,                                                          //数据长度
      GAP_ADTYPE_FLAGS,                                              //广播类型标识符
      DEFAULT_DISCOVERABLE_MODE_wechat | GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED,    //广播类型

      0x03,                                                          //数据长度
      GAP_ADTYPE_16BIT_MORE,                                         //16Bit 服务 UUID
      LO_UINT16(WECHAT_SERV_UUID),                                   //服务类型
      HI_UINT16(WECHAT_SERV_UUID),

      0x15,                                                          //根据实际使用长度更改
      GAP_ADTYPE_MANUFACTURER_SPECIFIC, //使用微信功能该处应加上 MAC 地址(可根据实际需要更改)
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x0d,0x0a,//此处可根据实际需要填写相关数据
      0x44 ,0xA6,0xE5,0x07,0x99,0x7A      //此处必须填写设备 MAC 地址(以下为测试使用 MAC 地址)

};

/*********************************************************************
 * 本地函数声明
 */
static void WechatGetMacAddrUpdatetoAdvertData(void);
static bool ReceivedDataStateGet(void);
static void ReceivedDataStateSet(bool flag);
static bool ReadToSendStateGet(void);
```

```
static void TTCDriverDemoWechatClockHandler(UArg arg);
static void UartRxToWechat(u8 *brfferData,u16 len);
static void WechatDataReceived(u8 *receivedData,u16 len);
static TTCDriverInfo_t WechatDataSend(u8 *sendData,u16 len);

/*****************************************************************************
【函    数】  TTCDriverDemoWechatInit(TTCSdkClass_t *appCallbacks,
                                      ICall_Semaphore *sem,
                                      Queue_Handle *queueHandle,
                                      ICall_EntityID *selfEntity)
【概    述】  微信初始化及参数设置
【入口参数】  sem         : 信号量
              appMsgQueue : 消息句柄
              selfEntity  : 任务 ID
【返回参数】  无
【说    明】  无
*****************************************************************************/
void TTCDriverDemoWechatInit(TTCSdkClass_t *appCallbacks,
                             ICall_Semaphore *sem,
                             Queue_Handle *queueHandle,
                             ICall_EntityID *selfEntity){

    Wechatsem = sem;
    Util_constructClock(&WechatClock,
                        TTCDriverDemoWechatClockHandler,
                        1000,
                        500,
                        true,
                        SBP_WECHAT_SEND_EVT);


    TTCBleWechatParam_t wechatParam= {
      .ramSize = 50,                                    //发送缓存大小(不得小于 30)
      .authSelect = AUTH_MAC_NOENCRYPT,                 //登录方式选择
      .dataDirSelect = DATA_DIR_BG,                     //数据方向
      .deviceMac = {                                    //设备 MAC 地址
                        Wechat.ownAddress[5],
                        Wechat.ownAddress[4],
                        Wechat.ownAddress[3],
                        Wechat.ownAddress[2],
                        Wechat.ownAddress[1],
                        Wechat.ownAddress[0]
```

```
                    },
    };
    TTCBleWechatInit(appCallbacks,                          //微信初始化
                     sem,
                     queueHandle,
                     selfEntity,
                     wechatParam);
}

/*****************************************************************************
【函    数】  TTCSDKDriverDemoWechatEvent(void)
【概    述】  Demo Wechat 事件处理
【入口参数】  无
【返回参数】  无
【说    明】  无
*****************************************************************************/
void TTCSDKDriverDemoWechatEvent(void){
    if(events & SBP_WECHAT_SEND_EVT){
        events &= ~SBP_WECHAT_SEND_EVT;
        if(ReadToSendStateGet()){
            WechatDataSend("X",1);      //定时发送数据测试
        }
    }
    if(events & SBP_WECHAT_RECEIVED_EVT){
        events &= ~SBP_WECHAT_RECEIVED_EVT;
        if(ReceivedDataStateGet()){
            //RX 缓存已经收到有数据
            ReceivedDataStateSet(false);
            Wechat.WechatState.Data_received_flag = 0;
#ifdef TTCDRIVER_UART
            TTCDriverUartWrite(&uartHandle,Wechat.Wechat_RX,Wechat.Wechat_rx_len);    //打印
#endif      //TTCDRIVER_UART
        }
    }
}


/*****************************************************************************
【函    数】  TTCBleWechatProcess(TTCMsg_t * TTCMsg)
【概    述】  处理微信数据
【入口参数】  TTCMsg : 接收微信数据数据消息结构体
【返回参数】  无
```

【说　　明】 无

**********/

```
void TTCBleWechatProcess(TTCMsg_t * TTCMsg){
    TTCData_t * TTCData = NULL;
    TTCData = TTCMsg->pValue;

    WechatReportStatus wechatStatus = (WechatReportStatus)TTCData->param;
    switch(wechatStatus){
        case WECHAT_REPORT_STATUS_AUTH:{                        //已完成登陆
           #ifdef TTCDRIVER_UART
            TTCDriverUartWrite(&uartHandle,
                               "Wechat Auth Finish\r\n",
                               strlen("Wechat Auth Finish\r\n"));
          #endif //TTCDRIVER_UART
        }break;

        case WECHAT_REPORT_STATUS_SEND_TO_MFRSVR:{        //收到厂商服务器回应数据
#ifdef TTCDRIVER_UART
            TTCDriverUartWrite(&uartHandle,"Wechat Recv Svr Response\r\n",strlen("Wechat Recv
Svr Response\r\n"));
#endif //TTCDRIVER_UART
        }break;

        case WECHAT_REPORT_STATUS_INIT:{          //已初始化完成，初始化完成后才能发送数据
            TTCDriverDemoWechatReadToSendStateSet(true);    //设置准备好发数据状态
            TTCBleWechatSend("CC2640 SDK TEST V1.0 123456789\r\n",
                             strlen("CC2640 SDK TEST V1.0\r\n"));
          #ifdef TTCDRIVER_UART
            TTCDriverUartWrite(&uartHandle,
                               "Wechat Init Finish\r\n",
                               strlen("Wechat Init Finish\r\n"));
          #endif //TTCDRIVER_UART
        }break;

        case WECHAT_REPORT_STATUS_MFRSVR_SEND:{        //收到厂商服务器发送的数据
            TTCBleWechatSend(TTCData->pValue,TTCData->len);
            WechatDataReceived(TTCData->pValue,TTCData->len);   //把数据收入接收缓存
            ReceivedDataStateSet(true);                         //设置数据接收状态
            TTCDriverDemoWechatClockHandler(SBP_WECHAT_RECEIVED_EVT);
           #ifdef TTCDRIVER_UART
            TTCDriverUartWrite(&uartHandle,
                               "Wechat Recv Svr Data\r\n",
```

```
                            strlen("Wechat Recv Svr Data\r\n"));
        #endif //TTCDRIVER_UART
    }break;

    case WECHAT_REPORT_STATUS_SWITCH_VIEW:{                              //界面切换
        WechatSwitchView viewStatus = (WechatSwitchView)*(TTCData->pValue);
       #ifdef TTCDRIVER_UART
        if(viewStatus == WECHAT_SWITCH_VIEW_ENTER){
            TTCDriverUartWrite(&uartHandle,
                               "Wechat Switch View: Enter View\r\n",
                               strlen("Wechat Switch View: Enter View\r\n"));
        }else if(viewStatus == WECHAT_SWITCH_VIEW_EXIT){
            TTCDriverUartWrite(&uartHandle,
                               "Wechat Switch View: Exit View\r\n",
                               strlen("Wechat Switch View: Exit View\r\n"));
        }
        #endif //TTCDRIVER_UART
    }break;

    case WECHAT_REPORT_STATUS_SWITCH_BACKGROUND:{                        //后台切换
        WechatSwitchBackGround bgStatus = (WechatSwitchBackGround)*(TTCData->pValue);
        #ifdef TTCDRIVER_UART
        if(bgStatus == WECHAT_SWITCH_BG_ENTER_BG){
            TTCDriverUartWrite(&uartHandle,
                               "Wechat Switch BackGround: Enter BackGround\r\n",
                               strlen("Wechat Switch BackGround: Enter BackGround\r\n"));
        }else if(bgStatus == WECHAT_SWITCH_BG_ENTER_FG){
            TTCDriverUartWrite(&uartHandle,
                               "Wechat Switch BackGround: Enter ForGround\r\n",
                               strlen("Wechat Switch BackGround: Enter ForGround\r\n"));
        }else{
            TTCDriverUartWrite(&uartHandle,
                               "Wechat Switch BackGround: Enter Sleep\r\n",
                               strlen("Wechat Switch BackGround: Enter Sleep\r\n"));
        }
        #endif //TTCDRIVER_UART
    }break;
    default:break;
}

if(TTCData->pValue)
    ICall_free(TTCData->pValue);
```

```
    if(TTCData)
        ICall_free(TTCData);
}


/*******************************************************************************
【函    数】  WechatDataSend(u8 *sendData,u16 len)
【概    述】  向微信发送数据
【入口参数】  sendData ：需要发送的数据
              len      ：需要发送的数据长度
【返回参数】  无
【说    明】  无
*******************************************************************************/
static TTCDriverInfo_t WechatDataSend(u8 *sendData,u16 len){
    TTCDriverInfo_t state;
    state = TTCBleWechatSend(sendData,len);
    return state;
}

/*******************************************************************************
【函    数】  WechatDataReceived(u8 *sendData,u16 len)
【概    述】  接收微信数据
【入口参数】  receivedData ：接收到的微信数据
              len ：接收到的微信数据长度
【返回参数】  无
【说    明】  无
*******************************************************************************/
static void WechatDataReceived(u8 *receivedData,u16 len){
    Wechat.WechatState.Data_received_flag = 1;
    Wechat.Wechat_rx_len = len;
    memcpy(Wechat.Wechat_RX, receivedData, Wechat.Wechat_rx_len);
}


/*******************************************************************************
【函    数】  UartRxToWechat(u8 *brfferData,u16 len)
【概    述】  从接收缓存读取数据
【入口参数】  brfferData ：串口缓存数据
              len        ：缓存数据长度
【返回参数】  无
【说    明】  无
*******************************************************************************/
```

```
static void UartRxToWechat(u8 *brfferData,u16 len){
    Wechat.Wechat_tx_len = len;
    memcpy(Wechat.Wechat_TX, brfferData, Wechat.Wechat_tx_len);
    Wechat.WechatState.uart_input_flag = 1;
}

/*****************************************************************************
【函    数】  TTCDriverDemoWechatClockHandler(UArg arg)
【概    述】  定时任务回调函数
【入口参数】  arg : 标记事件
【返回参数】  无
【说    明】  无
*****************************************************************************/
static void TTCDriverDemoWechatClockHandler(UArg arg){
    events |= arg;
    Semaphore_post(*Wechatsem);
}

/*****************************************************************************
【函    数】  TTCDriverDemoWechatReadToSendStateSet(bool flag)
【概    述】  设置发送微信数据使能
【入口参数】  flag：1 允许，0 未准备好
【返回参数】  无
【说    明】  无
*****************************************************************************/
void TTCDriverDemoWechatReadToSendStateSet(bool flag){
    Wechat.WechatState.init_ok_flag = flag;
}

/*****************************************************************************
【函    数】  ReadToSendStateGet(bool flag)
【概    述】  读取发送微信数据使能标志
【入口参数】  flag：1 允许，0 未准备好
【返回参数】  返回微信是否准备好发送数据状态
【说    明】  无
*****************************************************************************/
static bool ReadToSendStateGet(void){
    return Wechat.WechatState.init_ok_flag;
}

/*****************************************************************************
【函    数】  ReceivedDataStateSet(bool flag)
```

```
【概    述】  接收微信数据使能设置
【入口参数】  flag：1 有数据，0 没有数据
【返回参数】  无
【说    明】  无
*******************************************************************************/
static void ReceivedDataStateSet(bool flag){
    Wechat.WechatState.Data_received_flag = flag;
}

/*******************************************************************************
【函    数】  ReceivedDataStateGet(void)
【概    述】  获取接收微信数据使能状态
【入口参数】  无
【返回参数】  返回是否接收到微信数据状态
【说    明】  无
*******************************************************************************/
static bool ReceivedDataStateGet(void){
    return Wechat.WechatState.Data_received_flag;
}

/*******************************************************************************
【函    数】  void WechatGetMacAddrUpdatetoAdvertData(void)
【概    述】  更新 MAC 地址到广播数据中
【入口参数】  无
【返回参数】  无
【说    明】  无
*******************************************************************************/
static void WechatGetMacAddrUpdatetoAdvertData(void){
    TTCBlePeripheralGetParameter(GAPROLE_BD_ADDR, Wechat.ownAddress);
    WechatadvertData[ADVERT_MAC_ADDR+0] = Wechat.ownAddress[5];
    WechatadvertData[ADVERT_MAC_ADDR+1] = Wechat.ownAddress[4];
    WechatadvertData[ADVERT_MAC_ADDR+2] = Wechat.ownAddress[3];
    WechatadvertData[ADVERT_MAC_ADDR+3] = Wechat.ownAddress[2];
    WechatadvertData[ADVERT_MAC_ADDR+4] = Wechat.ownAddress[1];
    WechatadvertData[ADVERT_MAC_ADDR+5] = Wechat.ownAddress[0];
    TTCBlePeripheralSetParameter( GAPROLE_ADVERT_DATA,
                                  sizeof( WechatadvertData ),
                                  WechatadvertData);
}

#endif //TTCBLE_WECHAT
```

5. TTC SDK OAD

5.1. OAD 简介

OAD, Over-the-Air Download, 即空中升级。

本章主要是介绍 TTC SDK 里面的 OAD 使用方法，OAD 分为两种:一种是外部 OAD，另外一种是内部 OAD，什么是外部？什么是内部？其实外部和内部在硬件上的差别仅仅是多了一个外部 flash，使用带外部 flash 的方式，我们成为外部 OAD 升级，如果没有使用外部 flash，我们称之为内部 OAD 升级。

在软件方面，外部 OAD 升级原理是，先将 BLE 透传过来的数据，存贮在外部 flash 中，然后比对 BLE 下发校验和与 flash 实际数据的计算的校验和，如果比对成功，设置相应的启动元数据，然后重启系统，加载外部 flash 镜像到内部 flash，加载完毕，系统会设置对应的元数据，并确认固件已经更新最新版本。然后系统将会跳转到指定镜像地址开始执行程序。

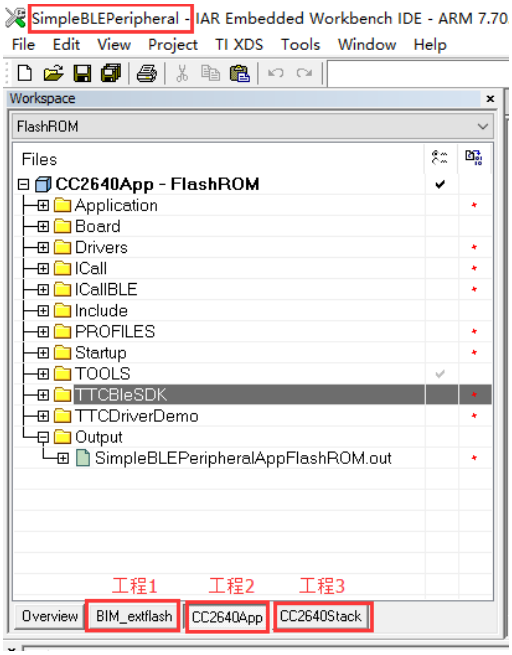
内部 oad 升级原理是，BLE 发送 OAD 升级命令，唤醒模块的 OAD 升级服务程序，系统接收到 BLE 发送的 OAD 升级命令，设置元数据，复位系统，然后跳转到 OAD 升级服务程序，升级服务程序与 BLE 连接后，BLE 就可以开始下发镜像文件，OAD 升级服务程序将镜像直接写入到原来的镜像空间，写完后，比对 BLE 下发的校验和与写入内部 flash 的校验和，比对成功，更新元数据，并跳转到新的镜像文件，开始执行新的程序，至此，内部 OAD 升级完成。

5.2 OAD 工程结构介绍

5.2.1 三个工程不同的作用

打开 SimpleBLEPeripheral 工程，共集合了三个工程，如下图。工程路径 TTC_CC2640_SDK_Vx.x.x \Projects\ble\SimpleBLEPeripheral\CC26xx\IAR

- 工程 1 (BIM_extflash - 片外 OAD 启动代码)
- 工程 2 (CC2640App - 应用程序)
- 工程 3 (CC2640Stack - 协议栈)



开发时，根据是否使用 OAD，可分为三种情况，每种情况对以上三个工程的使用的情况不同，具体如下：

(1) 不使用 OAD 功能，需使用工程 2 及工程 3，无需使用工程 1；

(2) 使用片内 OAD 功能，仅需使用工程 2。

(3) 使用片外 OAD 功能，以上三个工程均需要进行修改配置；

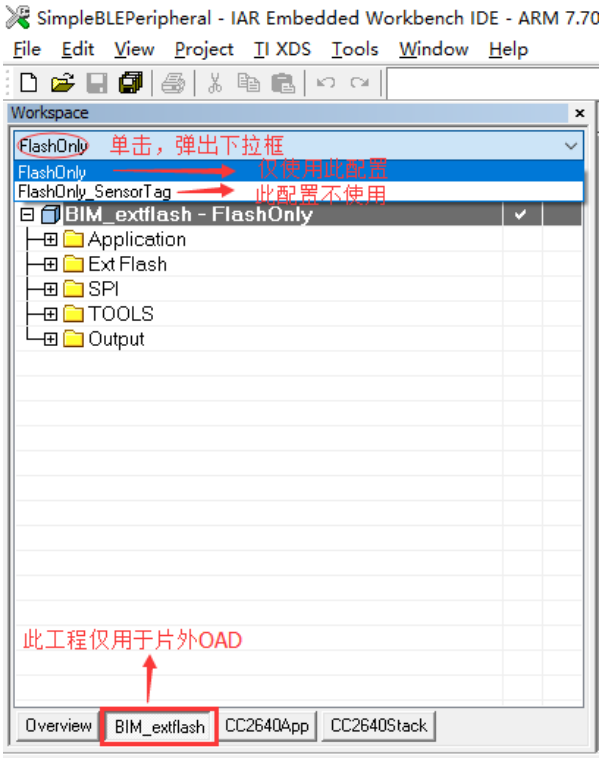
5.2.2 每个工程不同的配置

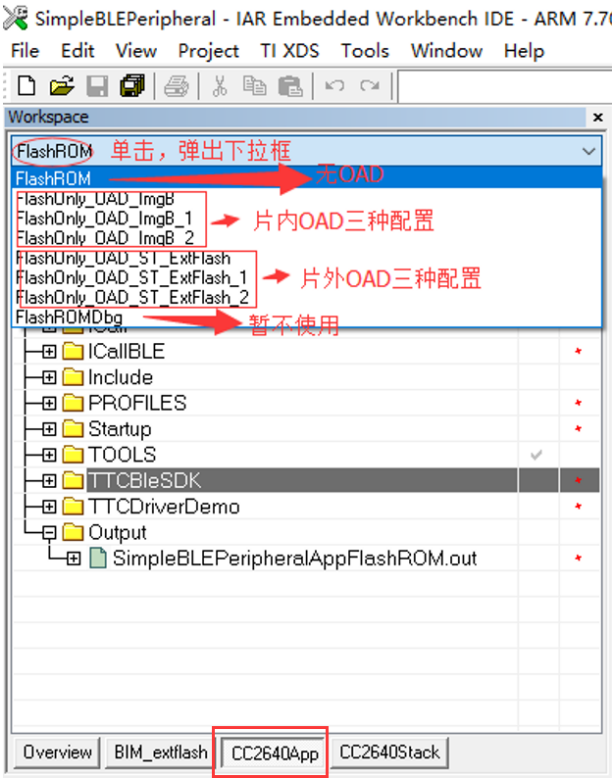
根据开发的项目是否需要 OAD 功能(分为片外 OAD、片内 OAD)，是否需要掉电存储功能，是否需要蓝牙绑定功能，选择不同的工程、选择每个工程的配置。具体配置方法如下表：

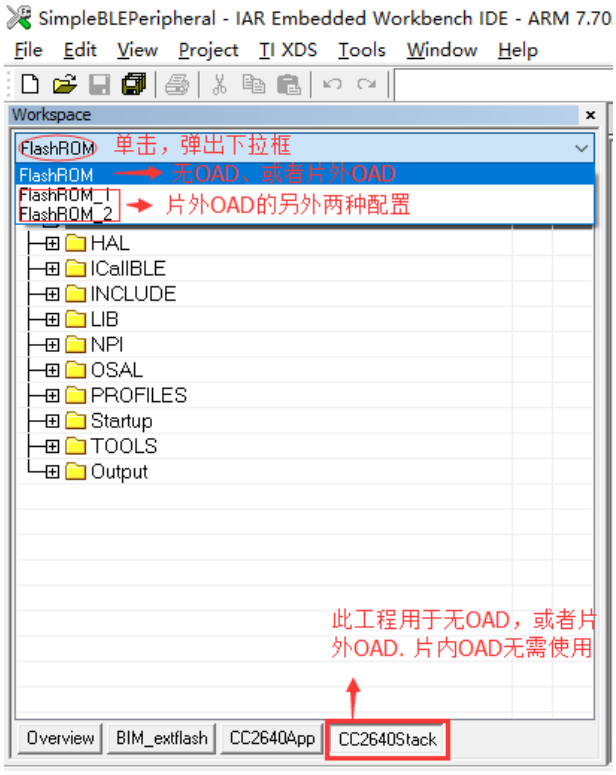
OAD 功能	功能		工程 1 配置	工程 2 配置	工程 3 配置
	掉电存储	蓝牙绑定	(BIM_extflash)	(CC2640App)	(CC2640Stack)
无 OAD 功能	√	√	\	FlashROM	FlashROM
	√	×	\	FlashROM	FlashROM_1
	×	×	\	FlashROM	FlashROM_2
有片内 OAD	√	√	\	FlashOnly_OAD_ImgB	\
	√	×	\	FlashOnly_OAD_ImgB_1	\
	×	×	\	FlashOnly_OAD_ImgB_2	\
有片外 OAD	√	√	FlashOnly	FlashOnly_OAD_ST_ExtFlash	FlashROM
	√	×	FlashOnly	FlashOnly_OAD_ST_ExtFlash_1	FlashROM_1
	×	×	FlashOnly	FlashOnly_OAD_ST_ExtFlash_2	FlashROM_2

注：“√”表示需要此功能，“×”表示不需要此功能，“\”表示不需要配置此工程。

具体 3 个工程各自不同配置，可按如下图方法设置：

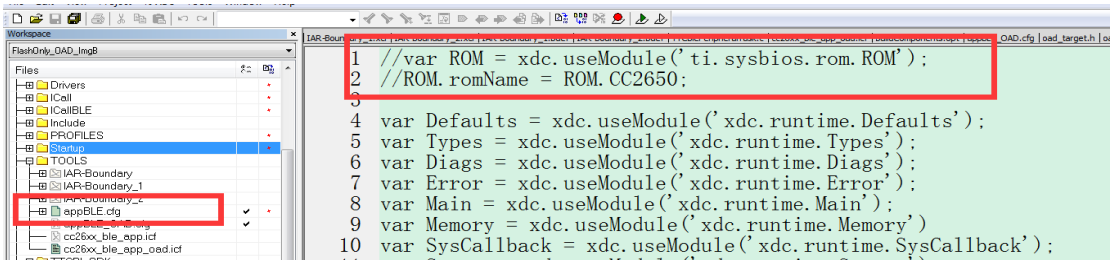






5.2.3 特别注意

特别注意：外部 OAD 和内部 OAD 工程务必要将下图中的两行代码注释掉。在无 OAD 的工程中这两行代码要取消注释。



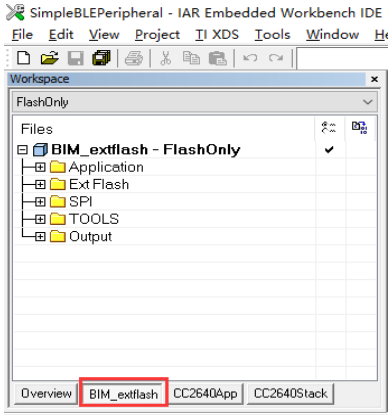
5.3. 外部 OAD

片外 OAD 分为两种配置：带掉电存储功能、不带掉电存储功能。以“带掉电存储功能”的工程为例，详细说明片外 OAD 过程。

5.3.1 BIM_extflash 工程(BIM 文件制作)

BIM-Boot Image Manager, the software bootloader, 启动代码。CC2640 上电复位后，由 BIM 检查片外 flash 是否存在新的程序需要升级至 CC2640 内部 flash。

打开 SimpleBLEPeripheral 工程，切换至 BIM_extflash 工程（此工程仅在外部 OAD 时使用），设置外部 flash 引脚及 ID. 路径 TTC_CC2640_SDK_Vx.x.x\Projects\ble\SimpleBLEPeripheral\CC26xx\IAR,如图：



(1) 查看原理图，设定 flash SPI 接口；

(2) 根据所使用的 flash 的类型，在 bsp.h 中设定厂商 ID 及器件 ID;
TTC_CC2640_SDK_Vx.x.x\Projects\ble\util\BIM_extflash\CC26xx\Source\CC26XXST_0120

(3) 重新编译工程，生成 BIM_ext.hex 文件。文件路径：
TTC_CC2640_SDK_Vx.x.x\Projects\ble\util\BIM_extflash\CC26xx\IAR\FlashOnly\Exe

5.3.2 CC2640 工程选择

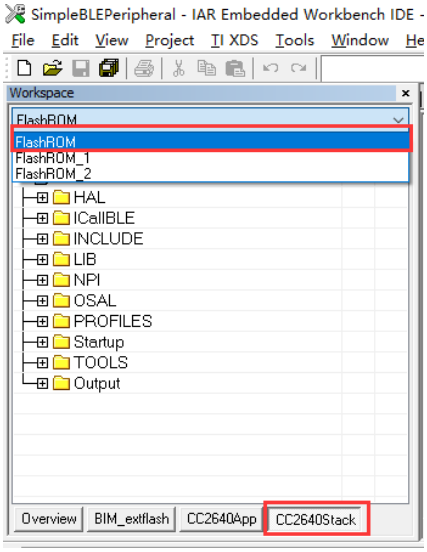
参照文档《SimpleBLEPeripheral 工程配置说明.txt》，根据实际需求（是否需要掉电存储及蓝牙绑定功能），选择合适的工程。特别说明：在使用外部 OAD 时，“CC2640App”及“CC2640Stack”两个工程的配置必须匹配，配置说明中有详细说明。

文档路径：TTC_CC2640_SDK_Vx.x.x\Projects\ble\SimpleBLEPeripheral

以下以 FlashOnly_OAD_ST_ExtFlash 与 FlashROM 配置为例。

5.3.3 CC2640Stack 工程配置

切换至 CC2640Stack 工程，选择 FlashROM 配置，此工程无需其他修改，重新编译工程。



5.3.4 CC2640App 工程设置

(1)打开 CC2640App 工程，选择 FlashOnly_OAD_ST_ExtFlash 配置，如图 2-3-1.

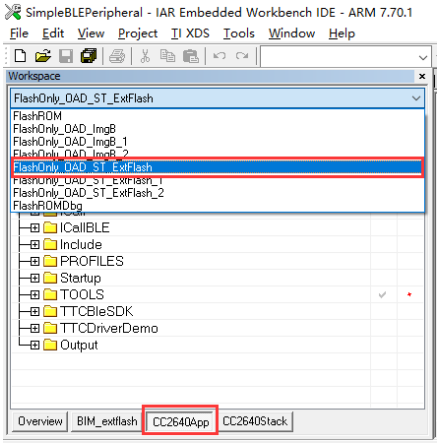


图 2-3-1 congfig 设置

（2）重新编译工程，生成协议栈、应用程序的合成文件 OAD_FULL_IMAGE.hex.

路径：TTC_CC2640_SDK_Vx.x.x\Projects\ble\SimpleBLEPeripheral\CC26xx\IAR\Application\CC2640\FlashOnly_OAD_ST_ExtFlash\Exe

5.3.5 烧录文件

（1）开启 Flash Programmer 2，选择 Multiple 模式，加载两个 flash image 文件，并烧录。如图 2-4-1.

- 3.1 小节(3)步骤生成的 BIM_ext.hex
- 3.4 小节(2)步骤生成的 OAD_FULL_IMAGE.hex

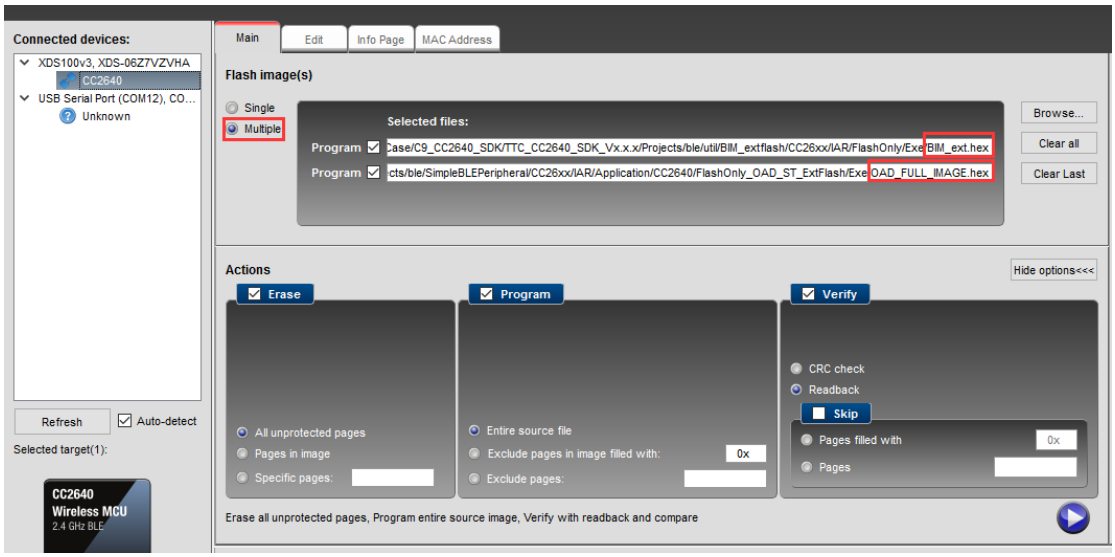


图 2-4-1 烧录

（2）生成 BIM_ext.hex 与 OAD_FULL_IMAGE.hex 的合并文件（便于生产烧录）。

在完成上述步骤（1）后，读取 CC2640 内部 flash 内容，即为最终合并文件。选择文件保存路径，并自定义命名。实现方式如图 2-4-2.

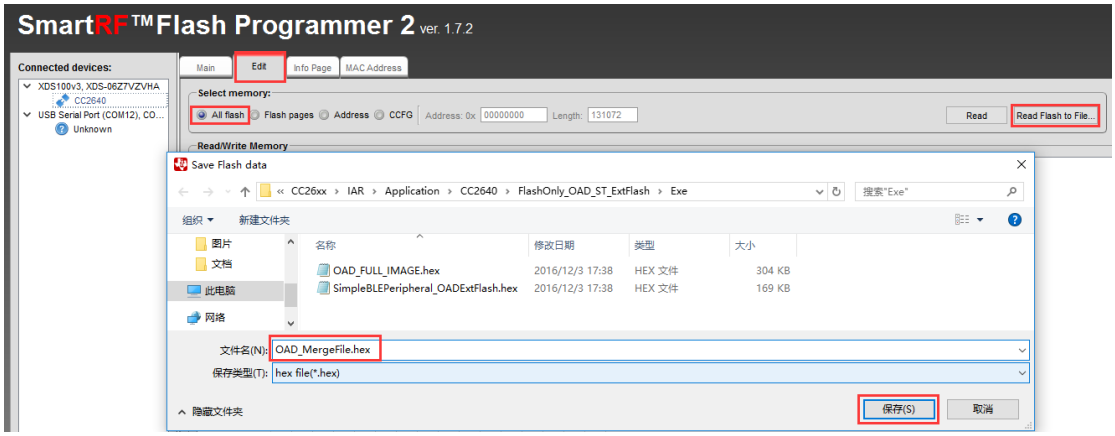
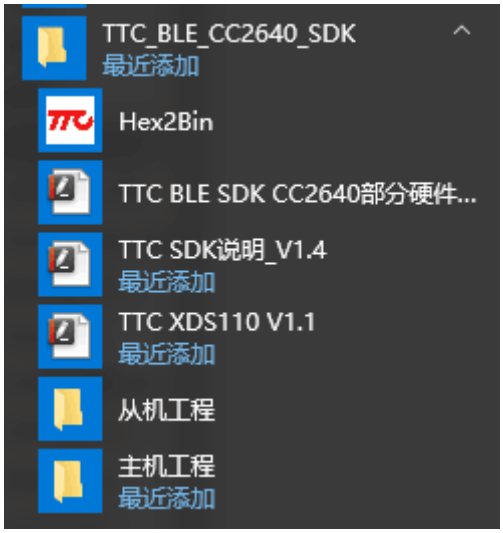


图 2-4-2 读取合并文件

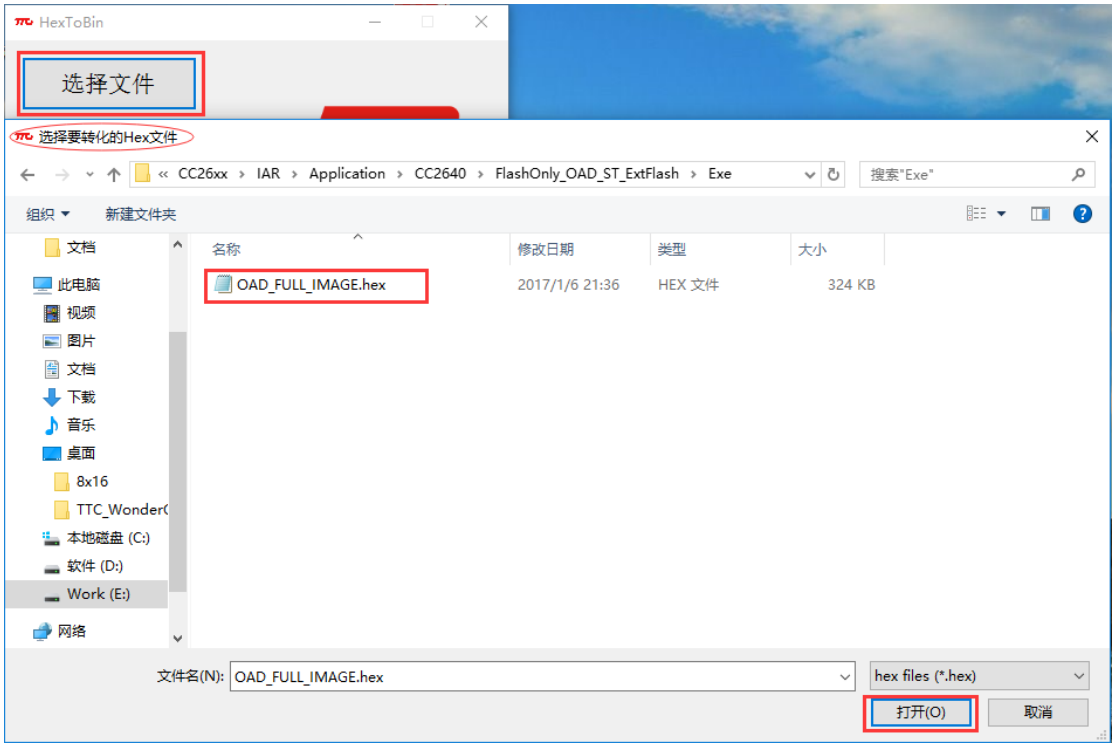
5.3.6 手机 APP OAD 操作步骤

5.3.6.1 生成片外 OAD 所使用 bin 文件

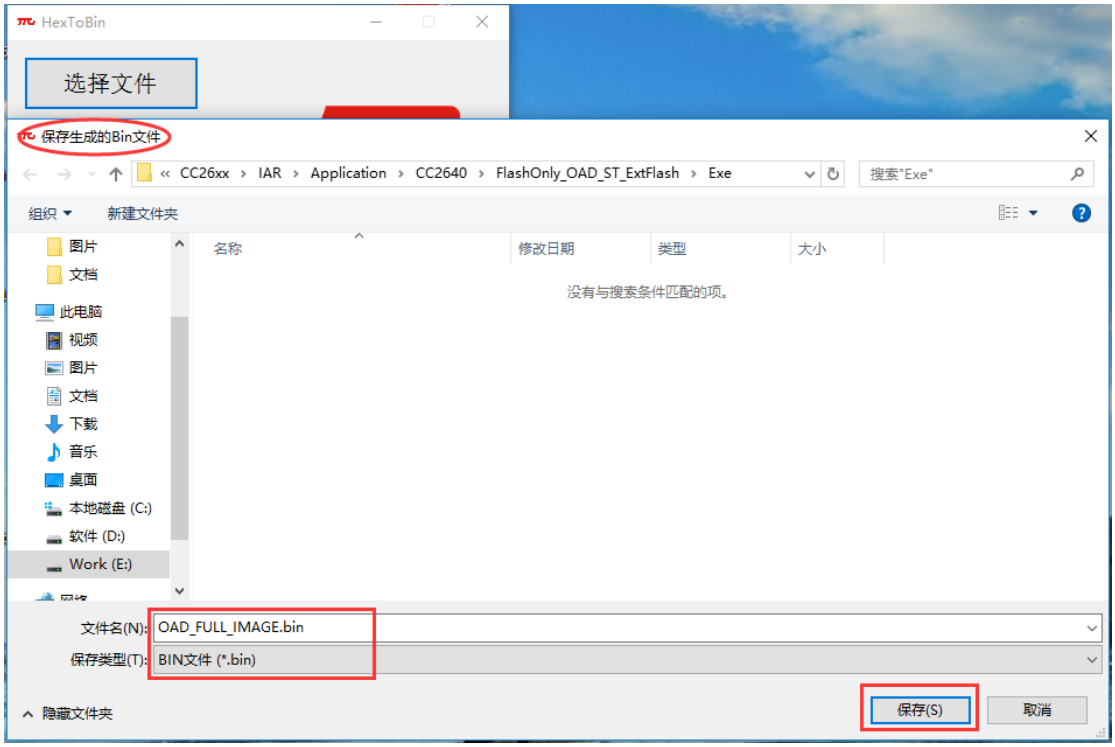
（1） 在 Windows 开始菜单中，打开 Hex2Bin.exe，如下图



（2）点击“选择文件”，导入 3.4 小节(2)步骤生成的 OAD_FULL_IMAGE.hex，在弹出的提示窗口中点击“确认”。



（3）保存生成的 Bin 文件 OAD_FULL_IMAGE.bin



5.3.6.2 使用 TTC_BLE 进行 OAD

（1）将 3.6.1 小结步骤(3)制作的 bin 文件 OAD_FULL_IMAGE.bin 导入手机根目录 Download 目录下；

（2）扫描二维码，下载安装 TTC_BLE，并开启手机蓝牙；

图 2-5-6 APP

（3）开启 APP，点击 OAD，进入 OAD 模式，如图 2-5-7；

（4）点击设备，与设备建立链接，如图 2-5-8；

（5）点击“OAD”，如图 2-5-9；

（6）选择 OAD 类型“CC2640 Off-Chip OAD”，如图 2-5-10；

（7）链接成功后，显示 Target Image Type 为 A；点击“+”，导入 bin 文件，文件导入后，File Image Type 为 B，设置传输间隔为 24ms，点击开始，如图 2-5-11；

（8）传输至 100%后，不需要操作 APP。等待 10 秒左右，设备复位，运行新程序，与 APP 断开链接；

注意：若 APP 与设备链接后，Target Image Type 不是 A，可以尝试重启手机蓝牙重新操作。

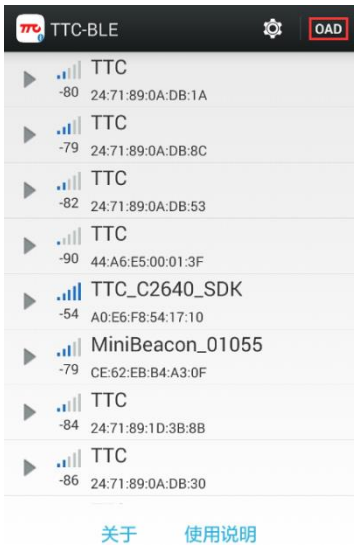


图 2-5-7 OAD 模式

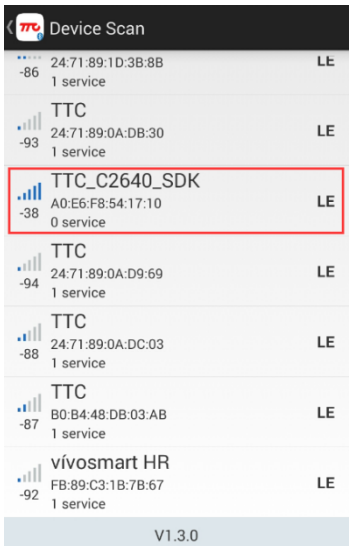


图 2-5-8 链接设备

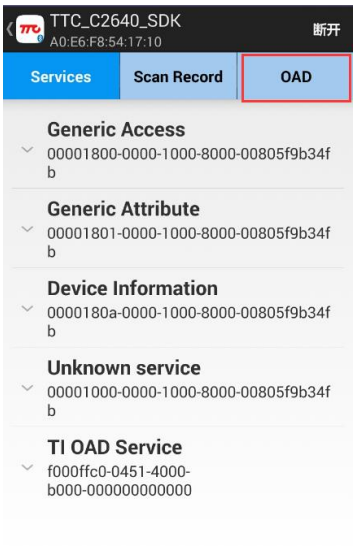


图 2-5-9 OAD 设置

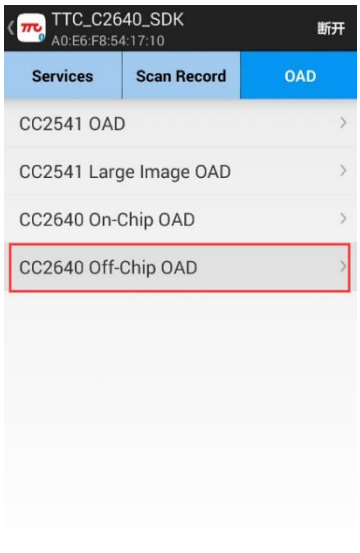


图 2-5-10 OAD 类型

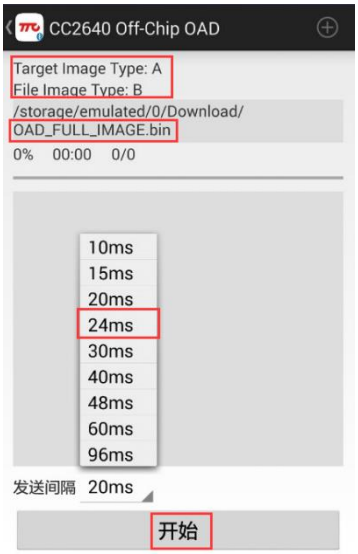


图 2-5-11 导入文件

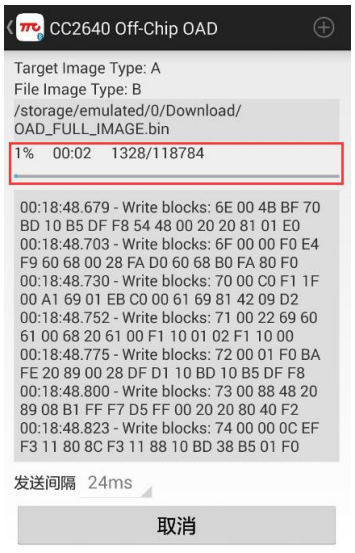


图 2-5-12 开始 OAD

5.4. 内部 OAD

5.4.1 内部 OAD 简介

CC2640 使用内部 OAD 时，内部 flash 包含有 BIM、ImageA、ImageB 以及 Stack。为简化操作，SDK 已经将 BIM、ImageA 及 Stack 部分合成为一个文件 OAD_Merge_x.hex（不同封装 IC，不同配置，合并文件并不一致），内部 OAD 时仅需更新 ImageB。

合并文件路径：TTC_CC2640_SDK_Vx.x.x\Projects\ble\SimpleBLEPeripheral \CC26xx \IAR\Application\CC2640\FlashOnly_OAD_ImgB_x \OADMergeFile

内部 OAD 有三种配置：FlashOnly_OAD_ImgB、FlashOnly_OAD_ImgB_1、FlashOnly_OAD_ImgB_2，具体如何选择见《SimpleBLEPeripheral 工程配置说明.txt》，文档路径：TTC_CC2640_SDK_Vx.x.x\Projects\ble\SimpleBLEPeripheral

以下以 FlashOnly_OAD_ImgB 配置为例。

5.4.2 内部 OAD 操作步骤

5.4.2.1 工程设置

使用内部 OAD 时，无需配置、编译 BIM_extflash 工程及 CC2640Stack 工程。将工程配置切换至 FlashOnly_OAD_ImgB，如图 3-2-1。

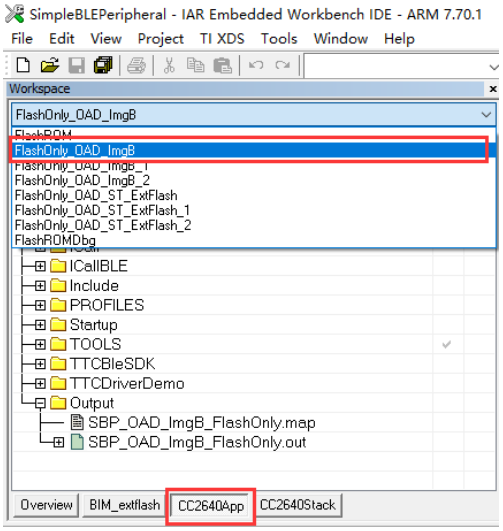


图 3-2-1 选择工程配置

编写并编译应用程序，将生成 ImageB 文件：SBP_OAD_ImgB.hex，路径为 TTC_CC2640_SDK_Vx.x.x\Projects\ble\SimpleBLEPeripheral\CC26xx\IAR\Application\CC2640\FlashOnly_OAD_ImgB\Exe

5.4.2.2 程序烧录

开启 Flash Programmer 2，选择 Multiple 模式，加载两个 flash image 文件，并烧录，如图 3-2-2. 两个 image 文件如下：

- 4.1 小节提及的合并文件 OAD_Merge.hex

➢ 4.2.1 小节生成的 SBP_OAD_ImgB.hex

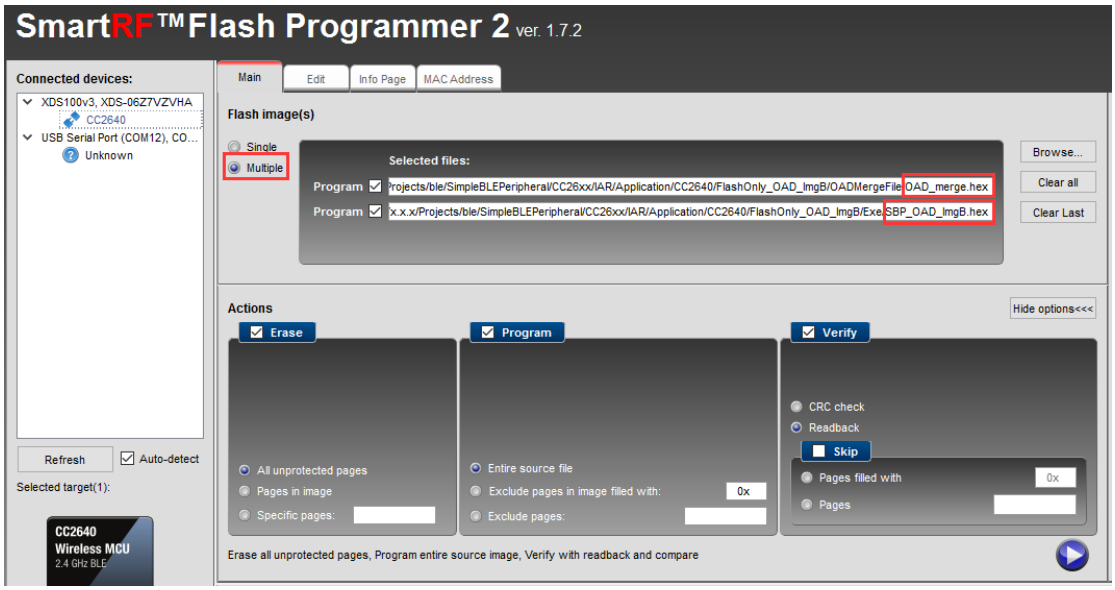
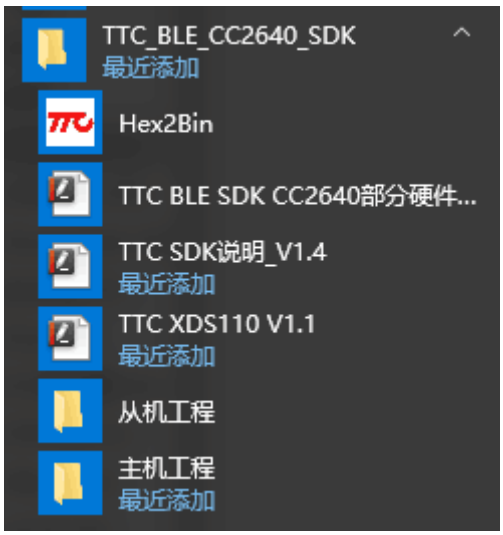


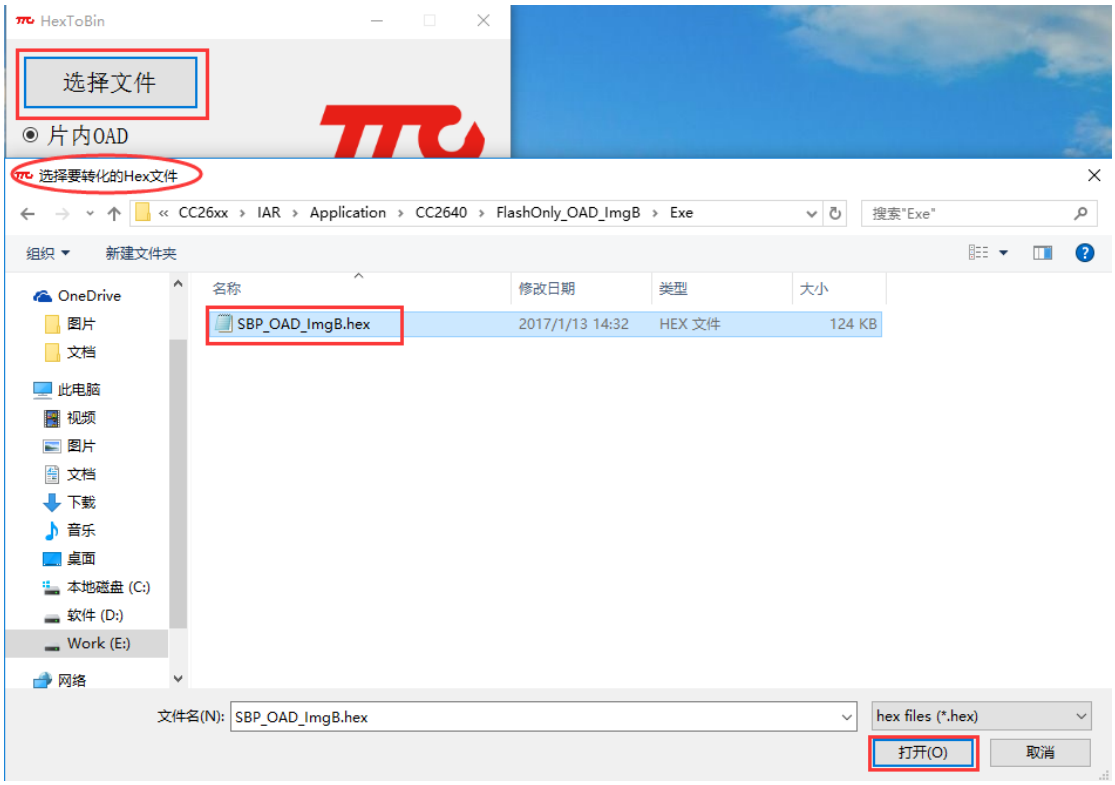
图 3-2-1 烧录

5.4.2.3 生成内部 OAD 所使用 bin 文件

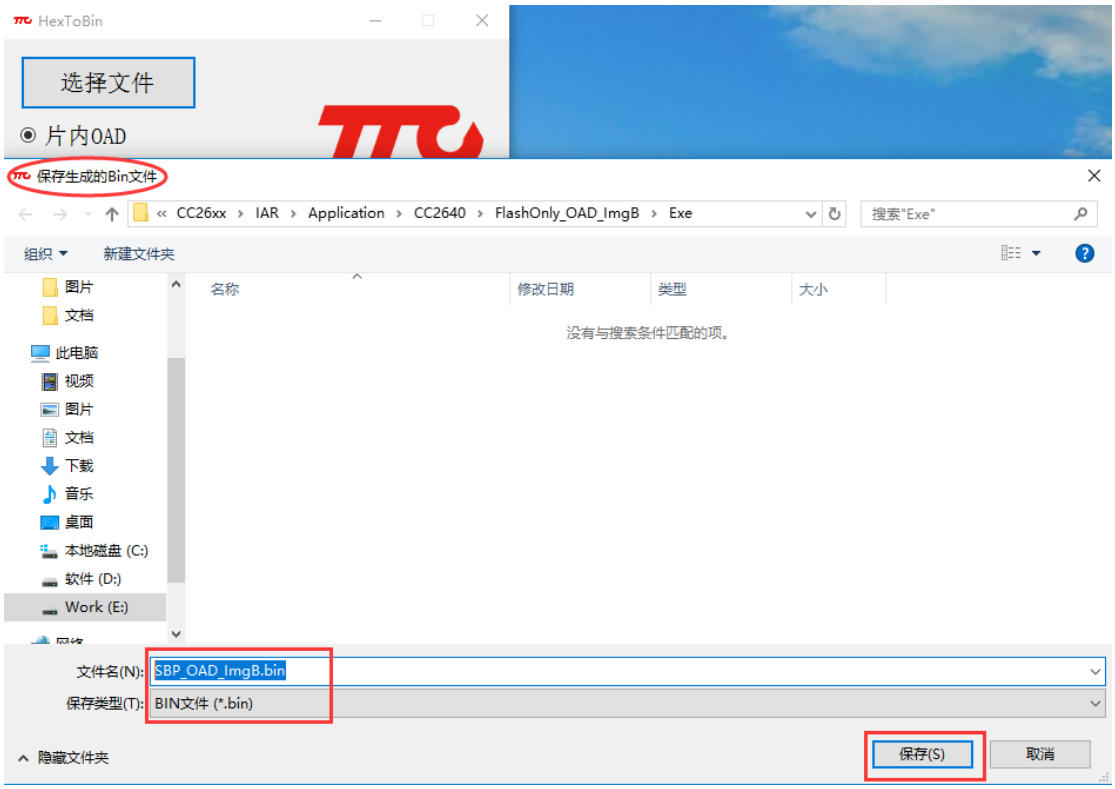
（1）在 Windows 开始菜单中，打开 Hex2Bin.exe,如下图



（2）点击“选择文件”，导入 4.2.1 步骤生成的 SBP_OAD_ImgB.hex，在弹出的提示窗口中点击“确认”。



（3）保存生成的 Bin 文件 SBP_OAD_ImgB.bin



5.3.2.4 使用 TTC_BLE 进行 OAD

（1）将 4.2.3 小结步骤(3)制作的 bin 文件 SBP_OAD_ImgB.bin 导入手机根目录 Download 目录下；

（2）扫描二维码，下载安装 TTC_BLE，并开启手机蓝牙；

图 3-2-6

（3）开启 APP，点击 OAD，进入 OAD 模式，如图 3-2-7；

（4）点击对应设备，与设备建立链接，如图 3-2-8；

（5）找到 0xf000ffd1 通道，写入 01，点击 write，如图 3-2-9 及 3-2-10；

（6）设备自动断开，返回上一个界面，点击连接，如图 3-2-11.

（7）连接成功后，仅有 TI OAD Service，点击 OAD，如图 3-2-12.

（8）选择 OAD 类型“CC2640 On-Chip OAD”，如图 3-2-10；

（7）显示 Target Image Type 为 A；点击“+”，导入 bin 文件，File Image Type 为 B，设置传输间隔为 24ms，点击开始，如图 3-2-14 及 3-2-15；

（8）传输至 100%后，不需要操作 APP。设备复位，运行新程序，与 APP 断开链接，如图 3-2-16 及 3-2-17；

注意：若 APP 与设备链接后，Target Image Type 不是 A，可以尝试重启手机蓝牙重新操作。

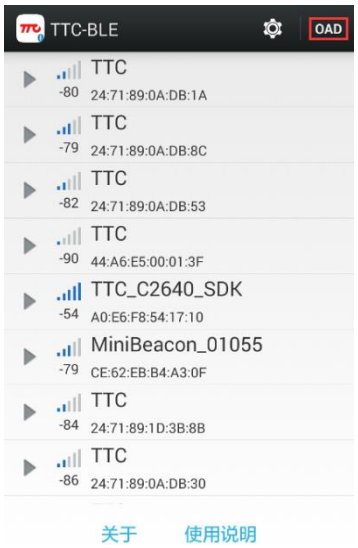


图 3-2-7 OAD 模式

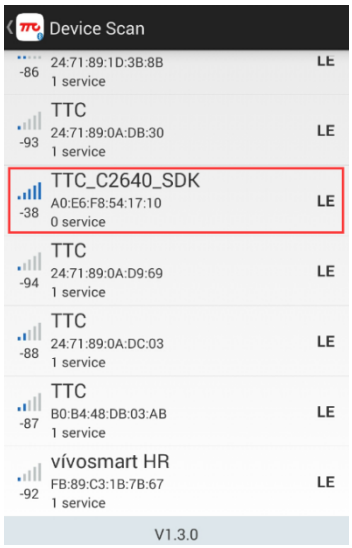


图 3-2-8 链接设备

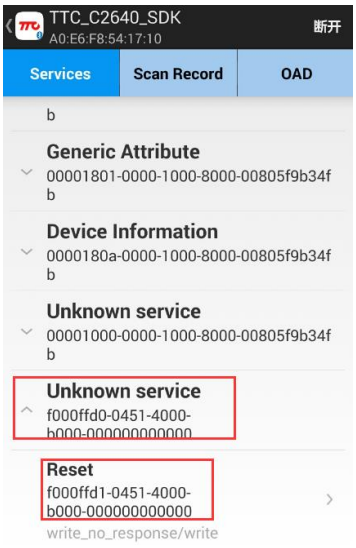


图 3-2-9 OAD 指令通道

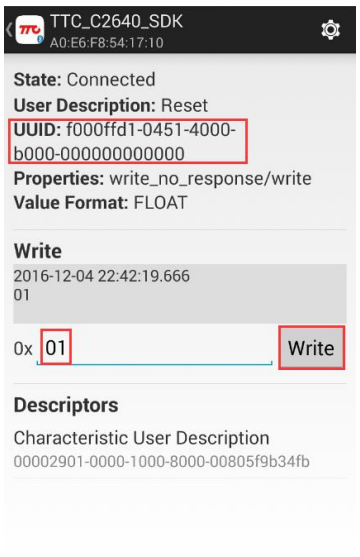


图 3-2-10 OAD 指令

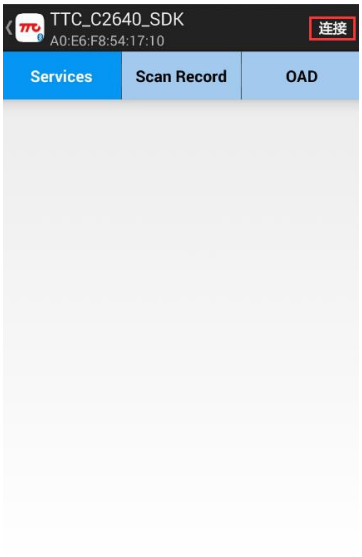


图 3-2-11 重连

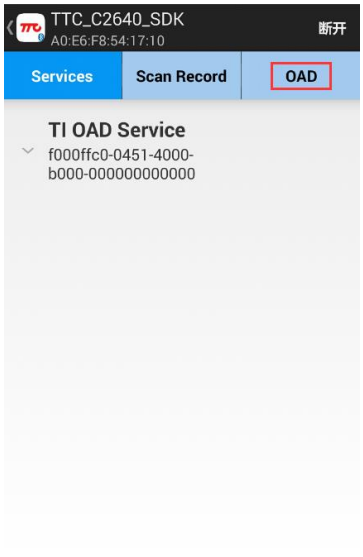


图 3-2-12 OAD 模式

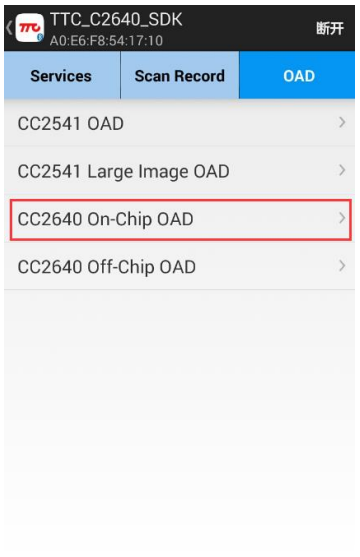


图 3-2-13 选择片内 OAD

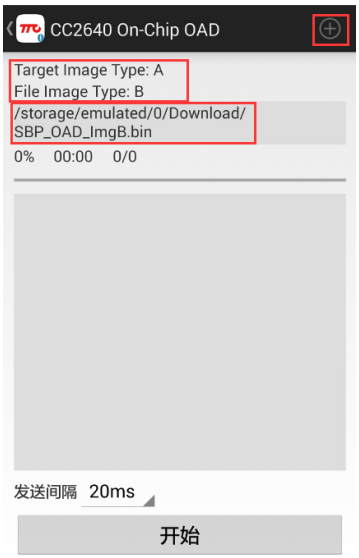


图 3-2-14 载入文件

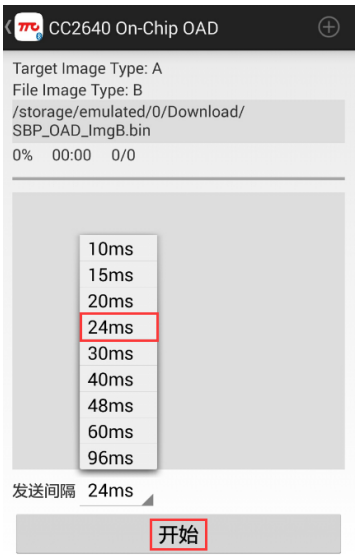


图 3-2-15 设置间隔

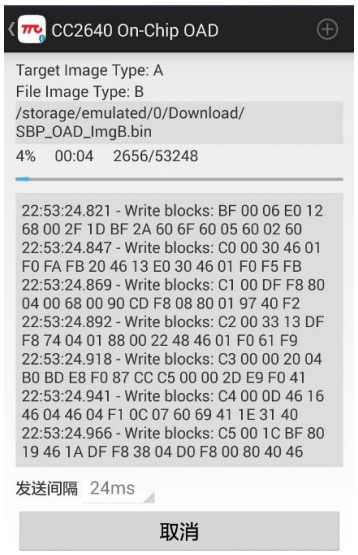


图 3-2-16 正在 OAD

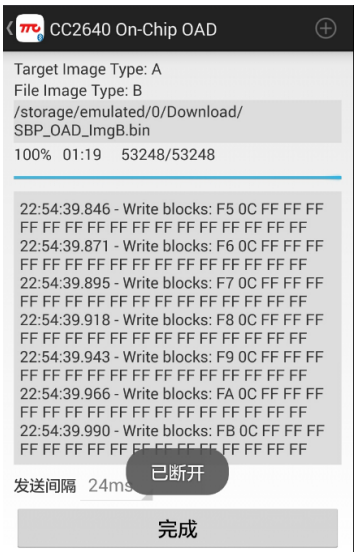


图 3-2-17 OAD 完成

6. 生产测试相关内容介绍

6.1 测试引脚的定义

```
#define TTCTEST_ENTER_TEST_IO          IOID_0      //上电前拉低 进入测试模式
#define TTCTEST_PWM_PIN                IOID_10     //48M
#define TTCTEST_UART_RX                IOID_14     //
#define TTCTEST_UART_TX                IOID_13     //
#define TTCTEST_UART_WAKEUP            IOID_12     //
```

以上这些定义可在 TTCSDKBoard.h 查找/修改。

注意：这里只允许修改后面的 IOID，禁止修改前面的宏标志。

说明：

TTCTEST_ENTER_TEST_IO

测试模式触发引脚，在上电之前先将此 IO 拉低，会使 SDK 进入测试模式，执行相关的测试代码。根据测试的结果我们就可以知道该模块是否工作正常。

TTCTEST_PWM_PIN

48M 测试输出口。

TTCTEST_UART_RX

测试用的 UART 的 RX 脚。

TTCTEST_UART_TX

测试用的 UART 的 TX 脚。

TTCTEST_UART_WAKEUP

测试用的 UART 的 WAKEUP 脚。

测试过程中会通过 AT 指令来进行测试项目的控制。

6.2. 测试方法

进入测试模式后我们可以通过发送 AT 指令来告诉模块我们要测试说明内容，指令的格式如下：

命令格式：	DATA0	DATA1	DATA2	DATA 3	....	DATAn
	0X33	命令头	长度	数据		校验值

长度：为数据的总长，不包含 DATA0，DATA1，DATA2，校验值。

校验值：为 DATA0+DATA1+...DATA(n-1)的总和。

测试命令介绍

描述	命令	应答	说明
检查命令	33 FF 01 00 33	33 FF 01 01 34	
32K 测试	33 C0 01 00 F4	33 C0 01 01 F5	发送测试指令后，应拉高 WAKEUP 引脚，等待 500ms 后，若能接受到 32K 回复指令则说明正常。
48M 测试	33 C1 01 00 F5	33 C1 01 01 F6	发送测试指令后，检测 TTCTEST_PWM_PIN 引脚，会输出周期为 20us 的 PWM 方波，占空比为 50%,持续 500ms 后会自动关闭 PWM。
IO 测试	33 C2 01 00 F6	33 C2 01 01 F7	发送测试指令后，立即开始检测所有引脚，除了 UART_TX/UART_RX/WAKEUP 引脚。相邻的两个脚电平不一样。100ms 后所有引脚电 平取反，再次检测。
TX 测试	33 C3 01 00 F7	33 C3 01 01 F8	发送指令后，测试电流，此时拉高 WAKEUP 引脚后，电流应为 6.043mA 左右。测试完成后需要发送退出测试指令。
RX 测试	33 C4 01 00 F8	33 C4 01 01 F9	发送指令后，测试电流，此时拉高 WAKEUP 引脚后，电流应为 6.471mA 左右。测试完成后需要发送退出测试指令。
退出测试模式	33 C5 01 00 F9	33 C5 01 01 FA	
用户自定义测试	33 E0 01 00 14	33 E0 01 01 15（成功） 33 E0 02 XX XX XX（不成功）	发送此命令后模块里的代码会调用用户事先编写好的代码，若执行成功则会返回成功的回应。
读取 MAC 地址	33 AD 01 00 E1	33 AD 06 XX XX XX XX XX XX XX	发送指令后会打开蓝牙广播，并返回 MAC 地址。只有该指令才能打开蓝牙广播。
版本号测试	33 F0 01 00 24	33 F0 VH VL XX	版本号：0xVHVL，校验：XX

7.联系我们

深圳市昇润科技有限公司
ShenZhen ShengRun Technology Co.,Ltd.
Tel: 0755-86233846 Fax: 0755-82970906
官网地址：www.tuner168.com
阿里巴巴网址：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

附录 A 参数说明

```
typedef enum {                    //GAPROLE 状态
    GAPROLE_INIT = 0,                 //等待蓝牙启动状态
    GAPROLE_STARTED,                  //已启动蓝牙但还未广播状态
    GAPROLE_ADVERTISING,              //广播状态
    GAPROLE_ADVERTISING_NONCONN,      //不可连接广播状态
    GAPROLE_WAITING,                  //设备已启动但未广播，等待启动广播状态
    GAPROLE_WAITING_AFTER_TIMEOUT,    //设备刚断开连接，但还未广播，
                                      //正在等待启动下一次广播启动状态
    GAPROLE_CONNECTED,                //连接状态
    GAPROLE_CONNECTED_ADV,            //连接+广播状态
    GAPROLE_ERROR                     //错误（无效状态）
} gaprole_States_t;

typedef enum{                  //TTC SDK 驱动类型
    TTCDRIVER_TYPE_UART = 0,      //UART 驱动
    TTCDRIVER_TYPE_SPI ,          //SPI 驱动
    TTCDRIVER_TYPE_I2C,           //IIC 驱动
    TTCDRIVER_TYPE_WATCHDOG,      //WatchDog 驱动
    TTCDRIVER_TYPE_TIMER ,        //Timer 驱动
    TTCDRIVER_TYPE_SENSOR_ADC,    //SensorController-ADC 驱动
    TTCDRIVER_TYPE_SIZE,          //无效状态
}TTCDriverType_t;

typedef enum{
    /* 驱动初始化状态 */
    TTCDRIVER_INIT_SUCCESS= 0x00,         //初始化成功
    TTCDRIVER_INIT_IO_FAILED = 0x01,      //初始化失败：IO 配置失败
    TTCDRIVER_INIT_RAM_FAILED= 0x02,      //初始化失败：TX 或 RX 缓存未分配或分配失败
    TTCDRIVER_INIT_SYSPARA_FAILED= 0x03, //初始化失败：信号量或消息句柄错误
    TTCDRIVER_INIT_FAILED_IS_OPENED= 0x04,//初始化失败：驱动已使能
    TTCDRIVER_INIT_PARA_ERROR = 0x05,      //初始化失败：参数错误
    TTCDRIVER_CLOSE_SUCCESS = 0x06,        //关闭成功
    /* 驱动状态 */
    TTCDRIVER_HANDLE_ERROR = 0x07,             //句柄错误
    TTCDRIVER_TRANSFER_DATA_SUCCESS = 0x08,    //传输数据成功或放入缓存成功
    TTCDRIVER_TRANSFER_DATA_BUSY= 0x09,   //传输失败，需等待 TX 缓存发送完毕后才能继续发送
    TTCDRIVER_TRANSFER_DATA_RAM_OVERFLOW= 0x0A, //传输数据溢出
    TTCDRIVER_TRANSFER_DATA_ERROR= 0x0B,         //传输数据失败
    TTCDRIVER_SLEEPED = 0x0C,                    //未唤醒
    TTCDRIVER_WAKEUP = 0x0D,                     //已唤醒
    TTCDRIVER_INFO_SIZE
```

```
}TTCDriverInfo_t;
typedef enum{
    TTCBLE_TYPE_READ_OPERATION = 0,        //主机对从机进行读操作
    TTCBLE_TYPE_WRITE_OPERATION ,          //主机对从机进行写操作(暂时不需要)
    TTCBLE_TYPE_SIZE
}TTCBleType_t;

typedef enum{
    MANAGER_INFO_REQUEST_IO_SUCCESS= 0x00, //申请使用 IO 成功
    MANAGER_INFO_REQUEST_IO_FAILED= 0x01,  //申请使用 IO 失败
    MANAGER_INFO_RELEASE_IO_SUCCESS = 0x02, //申请释放 IO 成功
    MANAGER_INFO_RELEASE_IO_FAILED = 0x03, //申请释放 IO 失败
    MANAGER_INFO_CONTROL_IO_SUCCESS= 0x04, //操作 IO 成功
    MANAGER_INFO_CONTROL_IO_FAILED = 0x05, //操作 IO 失败
    MANAGER_INFO_IO_ID_ERROR = 0x06,//IOID 错误
    MANAGER_INFO_IO_HANDLE_ERROR= 0x07,//IO 句柄错误
    MANAGER_INFO_IO_NOT_ALLOCATED= 0x08, //该 IO 未在句柄中分配
    MANAGER_INFO_SIZE
}TTCBleSDKManagerInfo_t;
```