TTC BLE SDK Description

Version：V1.8

Shenzhen Shengrun Technology Co.,Ltd

21th Feb,2017

Version	Revised Date	RevisIOnist	Reviewer	Modified Content
1.0	2016-06-13	廖健焜	张眼	First Release
1.1	2016-12-05	徐凯翔	张眼/廖健焜	1.Add SDKDriver instructions 2.Modify some of the original SDK API instructions
1.2	2016-12-09	徐凯翔	张眼/廖健焜	1.Modify the use of GPIO examples 2.Modified the name of some entries in the directory
1.3	2016-12-13	徐凯翔	张眼/廖健焜	1. Increase production test description part
1.4	2017-01-06	郭高亮/廖健焜	廖健焜	1. Introduction to TI-RTOS 2. Increase the OAD operating instructions 3. Modify the program space calculation 4. Add the Host API
1.5	2017-01-12	郭高亮	廖健焜	1. Modify the OAD bin file generation method
1.6	2017-01-16	郭高亮	张眼	1. Remove TI-RTOS profile 2. Increase the TTC_BLE QR code
1.7	2017-01-18	郭高亮	张眼	1. Update the UART, UTC, SPI API and demo programs 2. Increase watchdog related API instructions 3. Increase the SDK in the head files diagram
1.8	2017-02-21	郭高亮	张眼	1. Complete the Bluetooth specification 2. Increase the Beacon description 3. Improve the serial port related instructions 4. Update the ADC demo program

Content

1.TTC SDK Description

The TTC SDK is a CC2640 Rapid Development Library that focuses on optimizing the Bluetooth protocol stack and RTOS and provides common hardware-driven APIs.

Designed to allow developers to save a lot of time in Bluetooth debugging. The TTC SDK provides APIs such as Bluetooth parameter settings, Bluetooth data transceiver, and Bluetooth status processing, as well as the test program required for productIOn. Developers no longer need to design test programs.

Use the TTC SDK to adapt the TTC-BLE software provided by our company to facilitate debugging data transmissIOn and receptIOn, and support data encryptIOn and decryptIOn function. This can greatly shorten the CC2640 development cycle.

1.1. CC2640 SDK Peripheral Open Source Features

SimpleBLEPeripheral project contains three configuratIOns, configuratIOn differences and space calculatIOn as follows.

Project	OAD	Test Program	ROM(byte)			RAM(byte)		
			Total Amount	Dosage	Remaining	Total Amount	Dosage	Remaining
0 (SNV=1 BOND)	no	no	65536	26602	38934	17388	10612	6776
		Yes		35439	30097		12189	5199
	On-chip	no	45056	32453	12603	17408	10876	6532
		Yes		41495	3561		12457	4951
	Off-chip	no	61440	39222	22218	17388	12507	4881
		Yes		47626	13814		14080	3308
1 (SNV=1 NO BOND)	no	no	69632	26602	43030	17592	10612	6980
		Yes		35439	34193		12189	5403
	On-chip	no	49152	32453	16699	17488	10876	6612
		Yes		41495	7657		12457	5031
	Off-chip	no	65536	39222	26314	17592	12507	5085
		Yes		47626	17910		14080	3512
2 （ SNV=0 NO BOND ）	no	no	73728	26602	47126	17596	10612	6984
		Yes		35439	38289		12189	5407
	On-chip	no	53248	32453	20795	17488	10876	6612
		Yes		41495	11753		12457	5031
	Off-chip	no	69632	39222	30410	17596	12507	5089
		Yes		47626	22006		14080	3516

1.2. Public characteristics

- Bluetooth Service UUID：1000

Channel	UUID	Features	Description
UUID1	1001	Write_NoRsp/Read/Notify	Bluetooth data receptIOn
UUID2	1002	Read/Notify	Bluetooth data transmissIOn
UUID3	1003	Write_NoRsp	Register data write
UUID4	1004	Read	Register data read
UUID5	1005	Write_NoRsp/Read	Select the register

1.3.TTC SDK Advantage

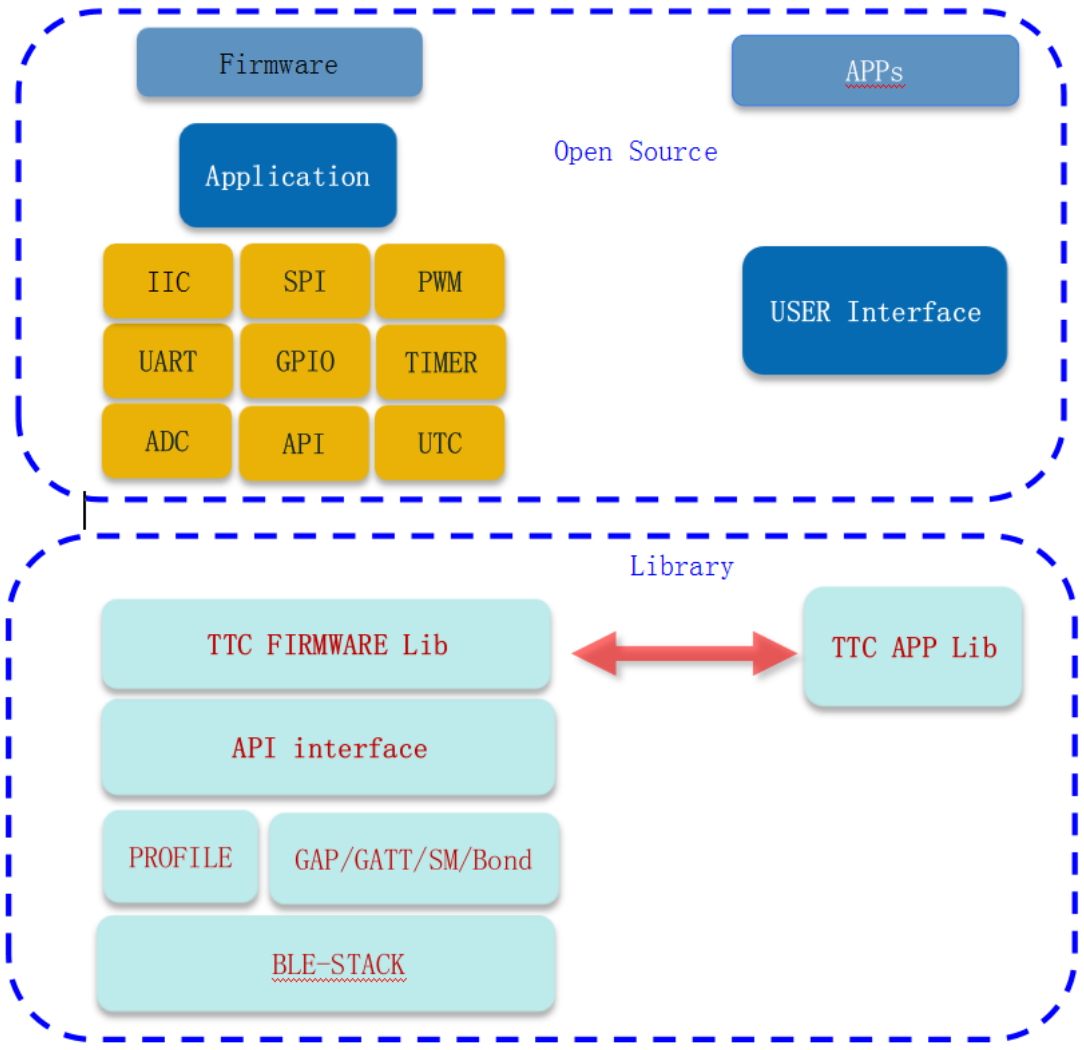
- Complete Bluetooth solutIOn (IC + firmware + APP + cloud)
- Simple Bluetooth settings and easy updating of firmware
- Similar to the serial port (UART) data transceiver Bluetooth interactive mode
- Complete SDK and tools available
- Fast start time (RTOS <500ms, OSAL <500ms)
- Ultra low power consumptIOn, CC2640 down to 1.5uA sleep current, available battery powered
- Data supports AES encryptIOn and decryptIOn
- Complete Bluetooth parameter verificatIOn.Adapt to Android and IOS, users no need to worry about Bluetooth adapter
- Provide complete test program, SDK has included Test Program, user no need to design Bluetooth Test Program
- Supporting the professIOnal test rack, SDK package, the development of debugging tools, and DEMO board
- Reduce the situatIOn due to Bluetooth equipment caused by abnormal work

1.4. The TTC SDK solves the problem

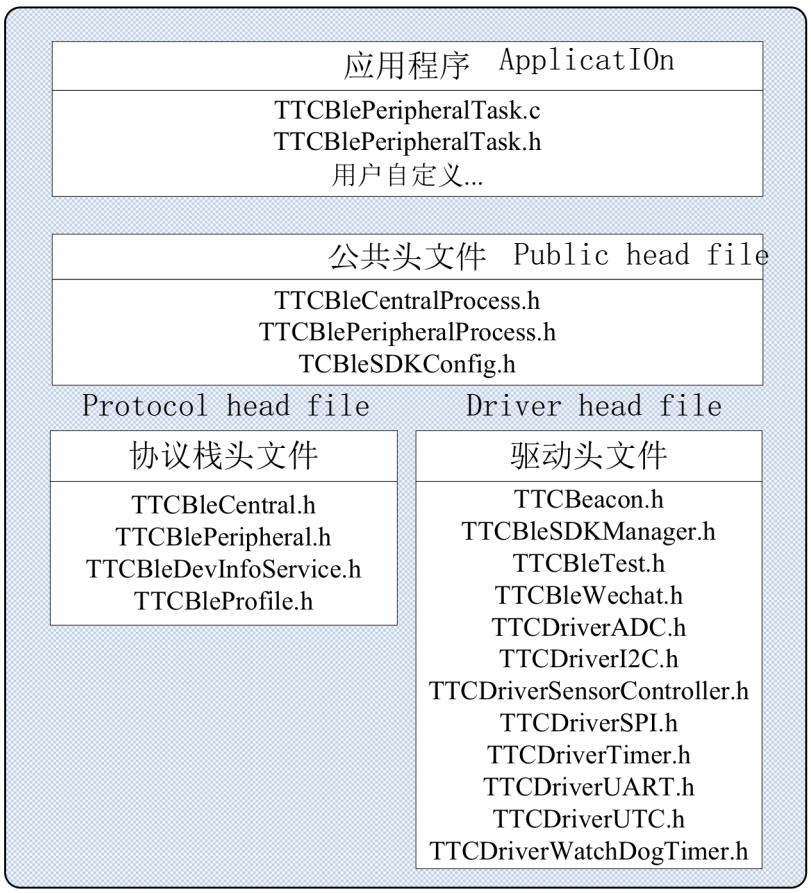
- Bluetooth firmware
- Dual Platform Bluetooth Development SDK
- CC2640 driver, which has been driven as follows：

  GPIO
  UART
  Timer（Contains PWM / Timing / Input Capture）
  ADC
  UTC
  IIC
  SPI
  Wechat
  WatchDog

2. SDK structure diagram



The relatIOnship between the varIOus header files in the SDK is shown below：



C: \ TTC_BLE_CC2640_SDK \ 1.0.7 \ TTC_CC2640_SDK \ Projects \ ble \ TTCBleSDK path under the file Description:

1.TTCDriverSPIFlash and TTCDriverSPISRAM folder to provide the open source code of external Flash and RAM operatIOno facilitate the user applicatIOn.

2.CC2640SDK_V1.0.6.a is the TTC SDK Lib library file.

3. The following is the library-related header files, the role of each header file is described as follows:

(1) TTCBeacon.h

The SDK supports the Beacon feature, which contains related parameters Initial value definitIOn (macro definitIOn),parameter settings， read and other related function declaratIOns。

(2) TTCBleCentral.h

Host role thread declaratIOn

(3) TTCBleCentralProcess.h

Host related operatIOn handling statement

(4) TTCBleDevInfoService.h

This header file has nothing to do with the Bluetooth device informatIOn, you can set to read the software Version, hardware Version, manufacturer, etc.

(5) TTCBlePeripheral.h

The peripheral role thread declaratIOn

(6) TTCBlePeripheralProcess.h

The peripheral related operatIOn handling statement

(7) TTCBleProfile.h

Related to the Bluetooth service, there is a statement to set the Bluetooth channel data can be sent through this function data to APP.

(8) TTCBleSDKConfig.h

Associated with GPIO, contains a statement of the function of the GPIO, and a Description of the relevant parameters.

(9) TTCBleSDKManager.h

(10) TTCBleTest.h

Related to the productIOn Test Program, including registered user custom tests, and Test Program Version callback function.

(11) TTCBleWechat.h

The SDK supports the WeChat function, which contains set the microphone broadcast data,WeChat Bluetooth initializatIOn, Add micro letter Bluetooth processing, WeChat data transmissIOn and other functions of the statement and related instructIOns.

(12) TTCDriverADC.h

Related to the ADC, including ADC sampling and other functions of the statement, and the relevant parameters

(13) TTCDriverI2C.h

Related to I2C,which contains I2C Read/White functions statement, and related parameter and instructIOns.

(14) TTCDriverSensorController.h

Only contains the SensorController initializatIOn function, if the project development need to use SensorController, more informatIOn please contact us.

(15) TTCDriverSPI.h

Related to SPI, including SPI read and write function declaratIOns, and related parameters and instructIOns.

(16) TTCDriverTimer.h

Related to the hardware timer, including PWM output, timing, and other functions related to the function statement, and the relevant parameters and instructIOns.

(17) TTCDriverUART.h

Related to UART, including UART data function declaratIOn, and related parameters and instructIOns

(18) TTCDriverUTC.h

Related to UTC, including UTC set time, read time and other functions of the statement, and the relevant parameters and instructIOns.

(19) TTCDriverWatchDogTimer.h

Related to Watchdog, including statements for watchdog initializatIOn and "feed dog" functions, as well as related parameters and instructIOns.

3.TTC SDK API Description

3.1. TTC SDK Bluetooth Files Description

Bluetooth part of a total of seven head files

- TTCBleSDKConfig.h :TTC SDK Project configuratIOn head file
- TTCBleDevInfoService.h:TTC SDK device informatIOn service head files
- TTCBlePeripheral.h:TTC SDK peripheral thread head files
- TTCBlePeripheralProcess.h:TTC SDK peripheral handing head files
- TTCBleProfile.h:TTC SDK handing Bluetooth data head files
- TTCBleCentral.h:TTC SDK host thread head files
- TTCBleCentralProcess.h:TTC SDK host handing head files

note:Function described "Do not call the function, the function for the SDK function" Please do not call the user or delete the function.

3.2. Bluetooth initializatIOn API (TTCBlePeripheralProcess.h)

The parameters of the Bluetooth can be set directly by the following structure and passed to the function

InitializatIOn structure of the peripheral in TTCBlePeripheralInit(TTCBlePeripheralInitConf_t * config)

```
typedef struct{
    TTCData_t               advData;            //Broadcast data
    TTCData_t               scanRspData;        //Scan response data
    TTCData_t               attDevName;         //Set the generic device feature name
    u16                     maxConnInterval; //Maximum connectIOn interval
    u16                     minConnInterval; //Minimum connectIOn interval
    u16                     advOffTime;         // When set to 0, the device continues to broadcast.
                                                   When the nonzero value n is set, the device
                                                   broadcasts n ms and then enters the wait state
                                                   until it is set to a non-zero value again
    u8                      advEnable;          // Bluetooth broadcast enable
    u8                      updateParEnable;    // Parameter update enable
    u16                     updateParDelay;     // Parameter update delay (default 6)
    u16                     slaveLatency;       // skiped responses packages number of the Peripheral
    u16                     connTimeout;        //ConnectIOn timedout
    u8                      txPower;            //TX
    u16                     advInterval;        //Broadcast interval
    u16                     rssiReadPerIOd;     //RSSI refresh cycle
    u8                      encryptEnable;      //EncryptIOn enabled
    u8                      advNoticeEnable;    //Broadcast callback enabled
    TTCPeripheralCBFxn_t    CB;                 // callback function
    TTCSdkClass_t           *appCB;             //SDK UNIVERSAL CALLBACK CLASS
    ICall_Semaphore         *sem;               //Thread semaphore
    Queue_Handle            *queueHandle;       //Message handle
    ICall_EntityID          *entity;            //Task ID
}TTCBlePeripheralInitConf_t;
```

```
/*******************************************************************************
【Function】  TTCBlePeripheralInit(TTCBlePeripheralInitConf_t * config)
【Overview】  The peripheral initializatIOn
【Parameter】  config:Bluetooth parameter configuratIOn initializatIOn structure
【Return】  no
【Description】  no
*******************************************************************************/
void TTCBlePeripheralInit(TTCBlePeripheralInitConf_t * config);

The host parameter initializatIOn structure
    typedef struct{
    u16                 scanDuratIOn;              // Scan duratIOn (unit：ms)
    u16                 scanInterval;             //Scan interval
    u16                 scanWindows;               //Scan window
    u16                 rssiReadPerIOd;            // RSSI acquisitIOn cycle (not open)
    u8                  encryptEnable;            //EncryptIOn enabled
    TTCBleCentralScanType_t  scanResultType;   //Scanned results type(You can set the required return data as needed
                                                  //    Content    Settings    Type    Reference TTCBleCentralScanType_t )
    u8                  filterType;               // Filter type (You can set the type of broadcast you want to filter
                                                  //setting type reference @defgroup GAP_ADTYPE_DEFINES )
    u8                  maxScanResult;             //Set the maximum number of scans
    const TTCData_t   filterData;               //Set to filter the broadcast data content
    const TTCData_t   attDevName;               //Set the generic device feature name
    TTCSdkClass_t      *appCB;                   //SDK UNIVERSAL CALLBACK CLASS
    ICall_Semaphore   *sem;                     //Thread semaphore
    Queue_Handle       *queueHandle;             //Message handle
    ICall_EntityID     *entity;                 //Task ID
}TTCBleCentralInitConf_t;
/*******************************************************************************
【Function】  TTCBleCentralInit(TTCBleCentralInitConf_t * config)
【Overview】  The host initializatIOn
【Parameter】  config : Bluetooth parameter configuratIOn initializatIOn structure
【Return】  no
【Description】  no
*******************************************************************************/
extern TTCDriverInfo_t TTCBleCentralInit(TTCBleCentralInitConf_t * config);
```

3.3.Bluetooth parameters and related operatIOns API(TTCBlePeripheral.h)

```
/*******************************************************************************
【Function】  TTCBlePeripheralSetParameter(u16 param, uint8_t len, void *pValue)
【Overview】  The peripheral parameter setting
【Parameter】  param: Refer to GAPROLE_PROFILE_PARAMETERS for parameter content
               len:Parameter length
               pValue:Parameter content
【Return】  no
【Description】  no
*******************************************************************************/
bStatus_t TTCBlePeripheralSetParameter(u16 param, uint8_t len, void *pValue);

/*******************************************************************************
【Function】  TTCBlePeripheralGetParameter(u16 param, void *pValue)
【Overview】  the peripheral get the parameters
【Parameter】  param:parameter,parameter content refer to GAPROLE_PROFILE_PARAMETERS
               *pValue:Parameter content
【Return】  no
【Description】  no
*******************************************************************************/
bStatus_t TTCBlePeripheralGetParameter(u16 param, void *pValue);

/*******************************************************************************
【Function】  TTCBlePeripheralGAPRoleTerminateConnectIOn(void)
【Overview】  Bluetooth disconnect the connectIOn
【Parameter】  no
【Return】  no
【Description】  no
*******************************************************************************/
bStatus_t TTCBlePeripheralGAPRoleTerminateConnectIOn(void);

/*******************************************************************************
【Function】  TTCBlePeripheralGAPRoleTerminateConnectIOn(void)
【Overview】  Bluetooth disconnect the connectIOn
【Parameter】  no
【Return】  no
【Description】  no
*******************************************************************************/
bStatus_t TTCBlePeripheralGAPRoleTerminateConnectIOn(void);
```

```
/*******************************************************************************
【Function】  TTCBlePeripheralGAPRoleSendUpdateParam(u16 minConnInterval,
                                                     u16 maxConnInterval,
                                                     u16 latency,
                                                     u16 connTimeout,
                                                     u8 handleFailure);
【Overview】  send update Bluetooth parameter request
【Parameter】  minConnInterval:Minimum connectIOn interval
               maxConnInterval:Maximum connectIOn interval
               latency:the peripheral timeout(Allows to skip the number of responses packets)
               connTimeout:ConnectIOn timedout
               handleFailure:update the failure operatIO,see GAPROLE_FAILED_UPDATE
【Return】  no
【Description】  no
*******************************************************************************/
bStatus_t TTCBlePeripheralGAPRoleSendUpdateParam(u16 minConnInterval,
                                                 u16 maxConnInterval,
                                                 u16 latency,
                                                 u16 connTimeout,
                                                 u8 handleFailure);
```

3.4.Bluetooth data transmissIOn related API（TTCBleProfile.h）

```
/*******************************************************************************
【Function】  TTCBleProfileSetParameter( u8 param,u16 len, void *value )
【Overview】  Set the Bluetooth channel data
【Parameter】  param: set the device informatIOn service parameter, fill in the following parameter:
                                        TTCBLE_PROFILE_CHAR1(Not open)
                                        TTCBLE_PROFILE_CHAR2
                                        TTCBLE_PROFILE_CHAR3(Not open)
                                        TTCBLE_PROFILE_CHAR4(Not open)
                                        TTCBLE_PROFILE_CHAR5(Not open)
               len:Send data length
               value:Data
【Return】  no
【Description】  no
*******************************************************************************/
extern bStatus_t TTCBleProfileSetParameter( u8 param,u16 len, u8 * value );
```

3.5.Bluetooth host related operatIOn API （TTCBleCentralPorcess.h）

```
/*****************************************************************************
【Function】  TTCBleCentralProcEstablishLink(u8 scanResIndex)
【Overview】  Simple mode call connectIOn function
【Parameter】  scanResIndex: Returns the index number of the scan result
【Return】  TRUE: Call successful,currently in a connected state
              FALSE:call failure
【Description】  reason for call failure：1. The scan results in the SDK are empty
                                          2. The scan index exceeds the total number of scan results
*****************************************************************************/
extern bool TTCBleCentralProcEstablishLink(u8 scanResIndex);
/*****************************************************************************
【Function】  TTCBleCentralProcTerminateLink(u8 connHandle)
【Overview】  Function for simple mode call disconnect
【Parameter】  connHandle: The current connectIOn handle
【Return】  TRUE:Call successful, currently in a disconnected state
              FALSE:Call failed
【Description】  Call failed reason：1.ConnectIOn handle error
                                     2.current in a disconnected state
*****************************************************************************/
extern bool TTCBleCentralProcTerminateLink(uint16_t connHandle);
/*****************************************************************************
【Function】  TTCBleCentralProcStartDiscovery(void)
【Overview】  Simple mode call scan the broadcast function
【Parameter】  no
【Return】  TRUE:Call successful,current in a scan state
              FALSE:Call failed
【Description】  Call failed reason：1.The current device is not ready
                                     2.current in a connectIOn or being disconnected
              If the simple mode cannot meet the need，
              please call TTCBleCentralStartDiscovery in TTCBleCentral.h
*****************************************************************************/
extern bool TTCBleCentralProcStartDiscovery(void);
/*****************************************************************************
【Function】  TTCBleCentralProcWriteData(u16 connHandle,u8 *wBuf,u8 len)
【Overview】  Simple mode call data send function
【Parameter】  connHandle : connect handle
              wBuf        : send data
              len         : Data length
【Return】  TRUE:Call successful,current in data transmissIOn state
              FALSE:Call failed
```

【Description】 Call failed reason：1.current device disconnected
2.ConnectIOn handle error
3. Data is too long
4.The host not ready
5.The send buffer is empty
**********/

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

3.6.SDK THREAD MESSAGE DESCRIPTION

The following message event is defined in TTCBleSDKConfig.h head files (do not modify it）

```
#define TTCSDK_MSG_GET_BLE_STATE_EVENT 0x0001 //bluetooth state event
#define TTCSDK_MSG_GET_BLE_DATA_EVENT 0x0002 //bluetooth data event
#define TTCSDK_MSG_REFRESH_RSSI_EVENT 0x0003 // Refresh RSSI values
#define TTCSDK_MSG_GET_BLE_PARAM_EVENT 0x0004 //获取协商后的蓝牙 parameter
#ifdef FEATURE_OAD
#define TTCSDK_MSG_OAD_EVENT 0x0005 //Off-chipOAD update event
#endif //FEATURE_OAD
#define TTCSDK_MSG_DRIVER_UART_EVENT 0x0006 //TTCSDK driver event for UART
#define TTCSDK_MSG_DRIVER_SPI_EVENT 0x0007 //TTCSDK driver event for SPI
#define TTCSDK_MSG_DRIVER_I2C_EVENT 0x0008 //TTCSDK driver event for IIC
#define TTCSDK_MSG_DRIVER_UTC_EVENT 0x0009 //TTCSDK driver event for UTC
#define TTCSDK_MSG_DRIVER_PWM_EVENT 0x000A //TTCSDK driver event for PWM
#define TTCSDK_MSG_BLE_WECHAT_EVENT 0x000B //TTCSDK BLE wechat event
#define TTCSDK_MSG_GET_BLE_SCAN_RES_EVENT 0x000C //get ble scan result
#define TTCSDK_MSG_GET_CONINFO_EVENT 0x000D // Get ConnectIOn InformatIOn Event
```

Messages receive processing in TTCBlePeripheralTask.c

****

【Function】 TTCBlePeripheralTaskProcessAppMsg(TTCMsg_t *pMsg)

【Overview】 Thread message handling function

【Parameter】 pMsg:message data

【Return】 no

【Description】 Please note the release of memory

**********/

static void TTCBlePeripheralTaskProcessAppMsg(TTCMsg_t *pMsg);

Receive message processing in TTCBleCentralTask.c.

/**********

【Function】 TTCBleCentralTaskProcessAppMsg(TTCMsg_t *pMsg)

【Overview】 Thread message handling function

【Parameter】 pMsg : message data

【Return】 no

【Description】 Please note the release of memory

**********/

static void TTCBleCentralTaskProcessAppMsg(TTCMsg_t *pMsg)

3.6.1.TTCSDK_MSG_GET_BLE_STATE_EVENT Bluetooth state processing

The event will handle the state of the Bluetooth, the developer needs to be applied according to the Bluetooth state, which can be handled in this TTCSDK_MSG_GET_BLE_STATE_EVENT event.

Note: Do not remove TTCBlePeripheralProcessStateChangeEvt ((gaprole_States_t) pMsg-> hdr.state) (Peripheral part)

TTCBleCentralStateChange ((gaprole_States_t) pMsg-> hdr.state); (host sectIOn)

3.6.2.TTCSDK_MSG_GET_BLE_DATA_EVENT Bluetooth receives data processing

Developers can receive and process Bluetooth data in the peripheral and host modes, respectively, through the following functions

TTCBlePeripheralTaskGetBleData(TTCMsg_t * TTCMsg)

TTCBleCentralStateChange(TTCMsg_t * TTCMsg)

TTCMsg-> hdr.state carries the contents of the Bluetooth channel.TTCData-> param Indicates whether the Bluetooth data is normal.In the case of encryptIOn, if the encrypted data is incorrect, the data will not be decrypted, and TTCData-> param carries the parameter TTCSDK_ERR_ENCRYPT_DATA, indicating a data error. In other cases TTCData-> param carries the parameter TTCSDK_NOERR_DATA. The message passes the Bluetooth data structure as:

```
typedef struct{
    u16         len         ;//length
    u16         param       ;//parameter
    u8 *        pValue      ;//data content
}TTCData_t;
```

3.6.3.TTCSDK_MSG_REFRESH_RSSI_EVENT RSSI value read in connectIOn state

This event is used to process the acquired RSSI value. Developers are available at TTCBl
-ePeripheralSetParameter function to set the refresh cycle for RSSI, set RSSI refresh cycleparameter to GAPROLE_RSSI_READ_RATE. When the set value is 0, the RSSI value will not be refreshed. When the set value is non 0, the RSSI value will be updated perIOdically with the set value as a perIOd. The contents of the message are s8 (signed char) type data.

Note: The RSSI value will only be refreshed in the connected state.

3.6.4.TTCSDK_MSG_GET_BLE_PARAM_EVENT

This event is used to inform the developer of the host and peripheral devices the Bluetooth parameterof after negotiatIOn with . The developer can get the final negotiated Bluetooth parameter in the event. Message The structure of the Bluetooth parameter is

```
typedef struct{
u16    connInterval          ;//connect interval
u16    connSlaveLatency ;// Skip the number of responses packets that skiped by the peripheral
u16    connTimeout           ;//ConnectIOn timedout
}TTCBleParamUpdate_t;
```

3.6.5.TTCSDK_MSG_GET_BLE_SCAN_RES_EVENT Get scan result event

This event is used to get the results of the host scan. The scan result structure is

```
typedef struct{
    u8    MAC[6];              //Scanned device MAC adress
    u8    addrType;            // Match the address type
    s8    rssi;                //RSSI value
    u8    advDataLen;          //Broadcast datalength
    u8    scanDataLen;         //Scan response datalength
    u8    *advData;            //Broadcast data content
    u8    *scanData;           //Scan response data content
}TTCBleCentralScanResult_t;
```

3.6.6.TTCSDK_MSG_GET_CONINFO_EVENT get connectIOn informatIOn event

This event is used to get host connectIOn informatIOn. The connectIOn informatIOn structure is

```
typedef struct{
    u16   connHandle;          //connect handle
    u8    connDevAddr[6];      //Current connected device MAC adress
}TTCBleCentralConnInfo_t;
```

3.7.Device informatIOn service parameter setting （TTCBleDevInfoService.h）

/***

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

【Overview】 Set the device informatIOn service parameter

【Parameter】 param: Set the device informatIOn service parameter，fill in parameter as below:

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: Data length，corresponding parameter Data length as below

DEVINFO_SYSTEM_ID_LEN 8（Fixed value, length must be equal to 8 otherwise invalid）

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 customized

DEVINFO_PNP_ID_LEN 7（Fixed value, length must be equal to 7 otherwise invalid）

value:set data

【Return】 no

【Description】 no

***/

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

/***

【Function】 TTCBleDevInfoGetParameter(u8 param, void *value)

【Overview】 Get the device informatIOn service parameter

【Parameter】 param:Set the parameter, fill in the parameter as below

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:Read the data

【Return】 no

【Description】 no

**********/

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

3.8.ApplicatIOn thread public callback function Description

The public callback function was declared in TTCBleSDKConfig.h（Do not modify）

/**********

【Function】 (*TTCSdkSetEvent_t)(ICall_Semaphore sem,
u16 * events,
UArg arg)

【Overview】 Thread set event

【Parameter】 sem : Thread semaphore
events : Event source
arg : Marks the Event source event

【Return】 no

【Description】 the function is encapsulated into TTCBlePeripheralTaskClass_t as a utility function

**********/

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

/**********

【Function】 (*TTCSdkTaskEnqueueMsg_t)(ICall_Semaphore sem,
Queue_Handle queueHandle,
u16 event,
u16 state,
void * pValue)

【Overview】 Thread set event

【Parameter】 sem : Semaphore
queueHandle : Message handle
event : Message carrying event
state : Message carrying state

pValue : Message carrying data

【Return】 no

【Description】 the function is encapsulated into TTCBlePeripheralTaskClass_t as a utility function（do not modify）

**********/

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

/**********

【Function】 (*TTCSdkDriverCB_t)(TTCDriverType_t driverType,

u32 driverState,

u8 * buffer,

u16 len)

【Overview】 SDK DRIVER CALLBACK

【Parameter】 driverType : Driver type

driverState : driver state

buffer : data cache

len : Data length

【Return】 no

【Description】 The function is encapsulated into TTCBlePeripheralTaskClass_t, and the user can add the processing according to the usage situatIOn.

Note: It is strictly forbidden that this function calls an interrupt function.

**********/

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

/**********

【Function】 (*TTCSdkBleCB_t)(TTCBleType_t bleType,

u8 param,

u16 *len,

u8 *pValue)

【Overview】 BLE RELATED CALLBACK

【Parameter】 bleType : BLE RELATED OPERATIONS TYPE

param : parameter

len : Data length

pValue : data

【Return】 no

【Description】 The function is encapsulated into TTCBlePeripheralTaskClass_t, and the user can add

```
the processing according to the usage situatIOn.
            Note: It is strictly forbidden that this function calls an interrupt function.
*****************************************************************************/
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;
```

Above callback function Prototype In TTCBlePeripheralTask.c, the developer can use the callback to set up a thread event and send a message.

Note: TTCSdkDriverCB_t is not open

4. TTC SDK driver API instructIOns

4.1. GPIO Description

CC2640 has a wealth of GPIO resources to meet a variety of development needs, which up to 31 GPIOs can be provided for developers to support a variety of configuratIOns such as up and down, open drain, push-pull output. Each GPIO can be configured interrupt function, interrupt ways can also be flexible configuratIOn, such as rising edge interrupt, falling edge interrupt, rising and falling are interrupted and so on. In additIOn, each GPIO can be any mapping On-chip peripheral resources, such as PWM output, ADC input, etc.

4.1.1.GPIO API Description

4.1.1.1 TTCDriverIOOpen()

```
/*****************************************************************************
【Function】  TTCDriverIOOpen(PIN_Handle * pinHandle,
                              PIN_State * pinState,
                              const PIN_Config pinList[])
【Overview】  Open the IO port function
【Parameter】  pinHandle : Control the IO handle
               pinState  : Control IO status
               pinList   : IO PORT CONFIGURATION
【Return】  MANAGER_INFO_REQUEST_IO_SUCCESS：Open IO port successfully
            MANAGER_INFO_REQUEST_IO_FAILED：Open IO port failed
【Description】  Open IO port failure: If the IO port fails to open, please check whether there are
```

peripherals using the configuratIOn of the IO port.
**********/

```
extern TTCBleSDKManagerInfo_t TTCDriverIOOpen(PIN_Handle * pinHandle,
                                               PIN_State * pinState,
                                               const PIN_Config pinList[]);
```

Note: If you want to use GPIO, the first call must be this function, the function is to allocate resources for one or more GPIO, while returning a handle, GPIO all other operatIOns are based on this handle, A valid handle can not be manipulated before GPIO.

4.1.1.2 TTCDriverIOAdd()

```
/**********************************************************************
【Function】  TTCDriverIOAdd(PIN_Handle * pinHandle,
                             PIN_Config addIO)
【Overview】  Add IO port to handle
【Parameter】  pinHandle : Control the IO handle
               addIO     : Added IO port and configuratIOn
【Return】  MANAGER_INFO_REQUEST_IO_SUCCESS：Add IO port success
            MANAGER_INFO_REQUEST_IO_FAILED：Add IO port failed
【Description】If the IO port fails to open, please check whether the peripherals use the IO port.
**********/
extern TTCBleSDKManagerInfo_t TTCDriverIOAdd(PIN_Handle * pinHandle,
                                             PIN_Config addIO);
```

Description: This API is used to add a new GPIO handle to a GPIO, use only need to provide the specified and effective GPIO handle and the new GPIO configuratIOn informatIOn can be added after the completIOn of the new Increased GPIO operatIOn。

4.1.1.3 TTCDriverIORemove()

```
/**********************************************************************
【Function】  TTCDriverIORemove(PIN_Handle * pinHandle,
                                PIN_Id removeIO)
【Overview】  Remove the IO port from the handle
【Parameter】  pinHandle : Control the IO handle
               removeIO  : Removed IO port
【Return】  MANAGER_INFO_RELEASE_IO_SUCCESS：Add IO port success
            MANAGER_INFO_RELEASE_IO_FAILED：Add IO port failed
【Description】  Open IO port failure: If the IO port fails to open, please check whether there are peripherals using the configuratIOn of the IO port.
**********/
extern TTCBleSDKManagerInfo_t TTCDriverIORemove(PIN_Handle * pinHandle,
```

PIN_Id removeIO);

Description: This API user removes a GPIO from an Opened GPIO handle, and the GPIO can be used for other purposes after the removal is complete。

4.1.1.4 TTCDriverIOGetInputValue()

```
/*******************************************************************************
【Function】 TTCDriverIOGetInputValue(PIN_Id pinId)
【Overview】 Read the input value of the IO port
【Parameter】 pinId : IO port which need to get
【Return】 1：High level
           0：Low level
【Description】 no
*******************************************************************************/
extern u8 TTCDriverIOGetInputValue(PIN_Id pinId);
```

ExplanatIOn: We can use this API to get the power level of the GPIO port with the input function.

4.1.1.5 TTCDriverIOGetOutputValue()

```
/*******************************************************************************
【Function】 TTCDriverIOGetOutputValue(PIN_Id pinId)
【Overview】 Read the output value of the IO port
【Parameter】 pinId：IO port which need to get
【Return】 1：High level
           0：Low level
【Description】 no
*******************************************************************************/
extern u8 TTCDriverIOGetOutputValue(PIN_Id pinId);
```

ExplanatIOn: We can use this API to get the output power level of the GPIO port with output function. A typical example is the level flip output experiment for the GPIO port.

4.1.1.6 TTCDriverIOSetOutputVaule()

```
/*******************************************************************************
【Function】 TTCDriverIOSetOutputVaule(PIN_Handle * pinHandle,
                                       PIN_Id pinId,
                                       u8 val)
【Overview】 Set the IO port value
【Parameter】 pinHandle : Control the IO handle
              pinId     : IO port which need to control
              val       : value which want to output
【Return】 MANAGER_INFO_CONTROL_IO_FAILED： IO PORT SETTING FAILED
           MANAGER_INFO_CONTROL_IO_SUCCESS：IO PORT SET UP SUCCESSFULLY
           MANAGER_INFO_IO_NOT_ALLOCATED：The IO port is not included in the handle
           MANAGER_INFO_IO_HANDLE_ERROR：handle error
```

【Description】 no

*****/

extern TTCBleSDKManagerInfo_t TTCDriverIOSetOutputVaule(PIN_Handle * pinHandle,
PIN_Id pinId,
u8 val);

Description: This API can set the output level for a GPIO with output function.

4.1.1.7 TTCDriverIOSetConfig()

/*****

【Function】 TTCDriverIOSetConfig(PIN_Handle * pinHandle,
PIN_Config bmMask,
PIN_Config pinCfg)

【Overview】 Set the IO port function

【Parameter】 pinHandle：Control the IO handle
bmMask：Mark IO port operatIOn
pinCfg：IO PORT CONFIGURATION

【Return】 MANAGER_INFO_CONTROL_IO_FAILED： IO PORT SETTING FAILED
MANAGER_INFO_CONTROL_IO_SUCCESS：IO PORT SET UP SUCCESSFULLY
MANAGER_INFO_IO_NOT_ALLOCATED：The IO port is not included in the handle
MANAGER_INFO_IO_HANDLE_ERROR：handle error

【Description】 no

*****/

TTCBleSDKManagerInfo_t TTCDriverIOSetConfig(PIN_Handle * pinHandle,
PIN_Config bmMask,
PIN_Config pinCfg);

Note: This API is used to set some special features for the GPIO configuratIOn, such as interrupt function configuratIOn, the specific use of the reference to the next sectIOn of the use of examples.

4.1.1.8 TTCDriverIOGetConfig()

/*****

【Function】 TTCDriverIOGetConfig(PIN_Id pinId)

【Overview】 Read IO PORT CONFIGURATION

【Parameter】 pinId : Need to get the configuratIOn of the IO port

【Return】 IO port configuratIOn value

【Description】 no

*****/

extern PIN_Config TTCDriverIOGetConfig(PIN_Id pinId);

Description: This API is used to get the configuratIOn informatIOn for the specified GPIO. Based on this

informatIOn, we can know what the GPIO has.

4.1.1.9 TTCDriverIORegisterIntCallBack()

```
/*****************************************************************************
【Function】 TTCDriverIORegisterIntCallBack(PIN_Handle *pinHandle,
                                            PIN_IntCb pCb)
【Overview】 Set the IO port callback function
【Parameter】 pinHandle : Control the IO handle
              pCb       : Interrupt callback function
【Return】 MANAGER_INFO_CONTROL_IO_FAILED： IO PORT SETTING FAILED
           MANAGER_INFO_CONTROL_IO_SUCCESS：IO PORT SET UP SUCCESSFULLY
           MANAGER_INFO_IO_NOT_ALLOCATED：The IO port is not included in the handle
           MANAGER_INFO_IO_HANDLE_ERROR：handle error
【Description】 no
*****************************************************************************/
extern TTCBleSDKManagerInfo_t TTCDriverIORegisterIntCallBack(PIN_Handle *pinHandle,
                                                             PIN_IntCb pCb);
```

Description: This API is used to set the Interrupt callback function for a GPIO. For details, please refer to the example of the next sectIOn.

4.1.2. Examples of GPIO use

```
/*****************************************************************************
【File】 TTCDriverGPIODemo.c
【Overview】 TTC SDK GPIO demo code
【Edit】 SDK WORKING GROUP
【Revise】 SDK WORKING GROUP
【Revised date】 2016/12/01
【Version】 V1.0.0
【Description】

Function Description:
    ConfiguratIOn IOID_9 is the output port, configure IOID_1 is the input port and have the interrupt function, IOID_1 each pull down once, IOID_9 state flips once.
  Adding steps:
    1, add the corresponding head files TTCDriverGPIODemo.h
    2, in the project OpIOns / C / C + + Compiler / Defined symbols defined TTCDRIVER_GPIO
    3, the static void  TTCSDKDriverInit (void) function in the main thread to add code as below.
          TTCDriverDemoIOInit (KeyPressHandler);
      Note: If you use the Transparent Development Kit to debug the timer, unplug all the jumpers on the kit except 3.3V.
*****************************************************************************/
#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"

/*********************************************************************
 * Local variable
 */
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;

/*********************************************************************
 * Local function declaratIOn
 */
static void TTCDrierDemoIOIsrCallback(PIN_Handle handle, PIN_Id pinId);
static void TTCDriverDemoIOChangeClockHandler(UArg a0);


/*********************************************************************************
【Function】    TTCDriverDemoIOInit(keysPressedCB_t appKeyCB)
【Overview】    TTCDriver driver initializatIOn
【Parameter】    appKeyCB
【Return】    no
【Description】    This function configures IOID_9 as the output port, configures IOID_1 as the input port
```

```
and has the interrupt function. When IOID_1 is pulled down once, the status of IOID_9 is flipped once.
*****************************************************************************/
void TTCDriverDemoIOInit(keysPressedCB_t appKeyCB){
   TTCBleSDKManagerInfo_t err;
   err = TTCDriverIOOpen(&IOTestHandle,&IOTestState,(const PIN_Config *)IOTestConfig);
   // An IO group is opened successfully
   if(err != MANAGER_INFO_REQUEST_IO_SUCCESS){
      return;
   }

   // Add an IO to a group, add it after the common handle can be used to operate the new IO
   err = TTCDriverIOAdd(&IOTestHandle,IOID_9 |
                                          PIN_GPIO_OUTPUT_EN |
                                          PIN_INPUT_DIS |
                                          PIN_GPIO_HIGH);

   if(err != MANAGER_INFO_REQUEST_IO_SUCCESS){
      return;
   }

// Control this newly added IO output Low level
   err = TTCDriverIOSetOutputVaule(&IOTestHandle,IOID_9,0);

   // Indicates that the operatIOn was unsuccessful
   if(err != MANAGER_INFO_CONTROL_IO_SUCCESS){
      return;
   }


   // Add an IO, configured as a pull-up input
   err = TTCDriverIOAdd(&IOTestHandle,IOID_1 | PIN_GPIO_OUTPUT_DIS | PIN_INPUT_EN |
PIN_PULLUP);
   if(err != MANAGER_INFO_REQUEST_IO_SUCCESS){
      return;
   }
   //Register an IO Interrupt callback function
   err = TTCDriverIORegisterIntCallBack(&IOTestHandle,TTCDrierDemoIOIsrCallback);
   if(err != MANAGER_INFO_CONTROL_IO_SUCCESS){
      return;
   }

   // Configure interrupt function for IOID_1
```

```
  err = TTCDriverIOSetConfig(&IOTestHandle,PIN_BM_IRQ,IOID_1 | PIN_IRQ_NEGEDGE);
  if(err != MANAGER_INFO_CONTROL_IO_SUCCESS){
     return;
  }
  // If the system is opened the power saving function, you need to configure the wake-up function for this IO
  #ifdef POWER_SAVING

     // Configure interrupt wakeup function for IOID_1
     err=TTCDriverIOSetConfig(&IOTestHandle,

PINCC26XX_BM_WAKEUP,IOID_1|PINCC26XX_WAKEUP_NEGEDGE);

     if(err != MANAGER_INFO_CONTROL_IO_SUCCESS){
        return;
     }
  #endif

  // et the key processing function (debapping)
  Util_constructClock(&keyChangeClock,                    //
                      TTCDriverDemoIOChangeClockHandler,
                      200,                                  //200mS
                      0,                                 // The cycle period is 0 (no loop timing)
                      false,                             // Do not start this timer first
                      0);                                // Do not need to transfer parameter
  // Register the user's key handling function
  appKeyChangeHandler = appKeyCB;
}

/*********************************************************************************
【Function】   TTCDiver_Inter_Callback(PIN_Handle handle, PIN_Id pinId)
【Overview】   IOInterrupt callback function
【Parameter】   handle：handle
                pinId：IO Port
【Return】   no
【Description】   Interrupt callback processing function
*********************************************************************************/
static void TTCDrierDemoIOIsrCallback(PIN_Handle handle, PIN_Id pinId){
  Util_startClock(&keyChangeClock);    // Turn on this software timer (equivalent to delay)
}
/*********************************************************************************
【Function】   TTCDriverDemoIOChangeClockHandler(UArg a0)
```

```
【Overview】
【Parameter】    a0：event
【Return】    no
【Description】    no
*****************************************************************************/
static void TTCDriverDemoIOChangeClockHandler(UArg a0){
  (void)a0;
  if(TTCDriverIOGetInputValue(IOID_1) == 0){     // Determine the power level of IO
    keysPressed |= 0x01;
  }else{
    keysPressed &= ~0x01;
  }
  if (appKeyChangeHandler != NULL){
    (*appKeyChangeHandler)(keysPressed);       // Call the user's handler function
  }
}
/*****************************************************************************
【Function】    KeyPressHandler(uint8_t keys)
【Overview】    The user's key handling function
【Parameter】    keys：Key code
【Return】    no
【Description】    no
*****************************************************************************/
void KeyPressHandler(u8 keys){
  if(keys & 0x01){
      TTCBleSDKManagerInfo_t err;
      // Invert the status of IOID_9 feet
      err= TTCDriverIOSetOutputVaule(&IOTestHandle,
                                     IOID_9,!TTCDriverIOGetOutputValue(IOID_9));
      // Indicates successful operation
      if(err == MANAGER_INFO_CONTROL_IO_SUCCESS){
      }
  }
}
```

4.2 UART Description

CC2640 UART has the following characteristics：

1. Programmable baud rate generator with a maximum rate of up to 3 Mbps.
2. With independent 32 × 8 transmit (TX) and 32 × 12 receive (RX) FIFO buffer, can reduce the CPU interrupt handling action.
3. Standard asynchronous access bits with start, stop and parity.
4. Supports CTS and RTS functions.
5. Use uDMA to transfer data.
6. With programmable hardware flow control.

4.2.1 UART API Description

4.2.1.1 TTCDriverUartEvent()

/***

【Function】 TTCDriverUartEvent(void)

【Overview】 UART EVENT HANDING

【Parameter】 no

【Return】 no

【Description】 no

***/

extern void TTCDriverUartEvent(void);

Note: This API is a function associated with the SDK that must be added by the user to the application's thread, otherwise the UARTdata can not be transferred.

4.2.1.2 TTCDriverUartInitDefaultParam()

/***

【Function】 void TTCDriverUartInitDefaultParam(Uart_Handle * uartHandle)

【Overview】 Set the UART default parameter

【Parameter】 uartHandle : UART HANDLE

【Return】 no

【Description】 no

***/

extern void TTCDriverUartInitDefaultParam(Uart_Handle * uartHandle);

Description: Initialize the UART HANDLE, which should be initialized by calling this API before proceeding with the UART initialization operation.

4.2.1.3 TTCDriverUartInit()

/***

【Function】 TTCDriverUartInit(TTCSdkClass_t *appCallbacks,

TTCSdkUartWriteCB_t writeCB,

TTCSdkUartReadCB_t readCB,
Uart_Handle * uartHandle,
TTCDriverUartParams_t param)

【Overview】 Initialize the UART

【Parameter】 appCallbacks : Registration callback
writeCB : Serial write callback registration (if not need to fill in NULL)
readCB : Serial readback callback registration
uartHandle : UART HANDLE
param : UARTparameter

【Return】 Please refer to the reason for initialization failure： TTCDriverInfo_t

【Description】 If the initialization fails, the UART will return to the cause of the initialization failure.

**********/

```
extern TTCDriverInfo_t TTCDriverUartInit(TTCSdkClass_t *appCallbacks,
                                         TTCSdkUartWriteCB_t writeCB,
                                         TTCSdkUartReadCB_t readCB,
                                         Uart_Handle * uartHandle,
                                         TTCDriverUartParams_t param);
```

Description: Initialize the UART, pay attention the callback function, the actual application do not send data in the callback function.

4.2.1.4 TTCDriverUartClose()

/**********

【Function】 TTCDriverUartClose(Uart_Handle * uartHandle)

【Overview】 Turn off the UART

【Parameter】 uartHandle : UART HANDLE

【Return】 Please refer to TTCDriverInfo_t

【Description】 If the initialization fails, the UART will return to the cause of the initialization failure.

**********/

```
extern TTCDriverInfo_t TTCDriverUartClose(Uart_Handle * uartHandle);
```

Description:Turn off the UART。

4.2.1.5 TTCDriverUartWrite()

/**********

【Function】 TTCDriverUartWrite(Uart_Handle * uartHandle,
u8 *buffer,
u16 len)

【Overview】 Write data to the UART TX cache and send it

【Parameter】 uartHandle : UART HANDLE

buffer : Data buffer

size : Data length

【Return】 Send the cause of the failure，Please refer to TTCDriverInfo_t

【Description】 If the current Wakeup pin is pulled high

**********/

```
extern TTCDriverInfo_t TTCDriverUartWrite(Uart_Handle * uartHandle,
                                          u8 *buffer,
                                          u16 len);
```

Description:send data.

4.2.2 UART use example

```
/*****************************************************************************
【File】 TTCDriverUARTDemo.c
【Overview】 TTC SDK UART demo code
【Edit】 SDK WORKING GROUP
【Revise】 SDK WORKING GROUP
【Revised date】 2016/12/02
【Version】 V1.0.0
【Description】
  Adding steps:
   (1) First add the corresponding head files TTCDriverUARTDemo.h
   (2) Define TTCDRIVER_UART in OpIOns / C / C ++ Compiler / Defined symbols in project
   (3) Added in the TTCBlePeripheralTaskFxn () function
          #ifdef TTCDRIVER_UART
            TTCDriverUartEvent();                          //SDK
            TTCSDKDriverUARTEvent();                       //users
          #endif //TTCDRIVER_UART
   (4)add below code to the static void TTCSDKDriverInit (void) function in the main thread.
          #ifdef TTCDRIVER_UART
             TTCDriverDemoUARTInit(&sem,&appMsgQueue);
          #endif
   (5)The pin is defined in TTCSDKBoard.h

*****************************************************************************/
#ifdef  TTCDRIVER_UART
/*****************************************************************
 * head files
 */
#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"

/*********************************************************************
 * Constants and macro definitions
 */
#define TTCBLE_SDK_UART_EVN                         0x0001

/*********************************************************************
 * Local variable
 */
static TTCDriverInfo_t UartErrCode;
Uart_Handle uartHandle;
static Clock_Struct uClock;
static PIN_Handle uLedHandle;
static PIN_State    uLedState;
PIN_Config uLedConfig[] = {
    IOID_0 | PIN_GPIO_OUTPUT_EN   | PIN_INPUT_DIS   |   PIN_GPIO_HIGH,
    PIN_TERMINATE
};

static  ICall_Semaphore  *  UartSem;                                    //Thread
semaphore，used to wake up threads
static  Queue_Handle  *  UartMsgQueue;                                  //Message
handle
static u16 events;                                                      //local event

/*********************************************************************
 * Drive hardware attribute table statement, please do not delete
 */
extern const UARTCC26XX_HWAttrs uartCC26XXHWAttrs[];


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

```
 * Local function declaratIOn
 */
static void TestUartCB(void * uartHandle,u8 * buffer,u16 len);
static void TTCSDKDriverUARTSetEvent(UArg arg);
static void TTCDriverDemoUARTClockHandler(UArg arg);

/*****************************************************************************
【Function】  TTCDriverDemoUARTClockHandler(UArg arg)
【Overview】  Thread set event
【Parameter】  arg     : Marks the Event source event
【Return】  no
【Description】
*****************************************************************************/
static void TTCDriverDemoUARTClockHandler(UArg arg){
    TTCSDKDriverUARTSetEvent(arg);
}

/*****************************************************************************
【Function】  TTCSDKDriverUARTSetEvent(UArg arg)
【Overview】  Tag events and wake up threads
【Parameter】  arg : Tag events
【Return】  no
【Description】  no
*****************************************************************************/
static void TTCSDKDriverUARTSetEvent(UArg arg){
    events |= arg;
    Semaphore_post(*UartSem);                                    //Wake up
thread
}

/*****************************************************************************
【Function】  TestUartCB(TTCDriverUartState_t uartState,u8 * buffer,u16 len)
【Overview】  UART receive callback function
【Parameter】
【Return】  no
【Description】  no
*****************************************************************************/
static void TestUartReadCB(void * uartHandle,u8 * buffer,u16 len){
   static u8 ubf[50] = {0};
    memcpy(ubf,buffer,len);
    if(uLedHandle != NULL){
        TTCDriverIOSetOutputVaule(&uLedHandle,IOID_0,~TTCDriverIOGetOutputValue(IOID_0));
```

```
    }
    TTCDriverUartWrite(uartHandle,buffer,len);
}
/*********************************************************************************
【Function】   TestUartCB(TTCDriverUartState_t uartState,u8 * buffer,u16 len)
【Overview】   UART send callback function
【Parameter】
【Return】   no
【Description】   no
*********************************************************************************/
static void TestUartWritetCB(void * uartHandle,u8 * buffer,u16 len){

}
/*********************************************************************************
【Function】   TTCSDKDriverADCEvent(void)
【Overview】   Demo UART event handling
【Parameter】   no
【Return】   no
【Description】   no
*********************************************************************************/
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);
    }
}

/*********************************************************************************
【Function】   TTCDriverDemoUARTInit(TTCSdkClass_t *appCallbacks,
                                    ICall_Semaphore * sem,
                                    Queue_Handle * appMsgQueue)
【Overview】   UART example initialization
【Parameter】   appCallbacks：Registration callback
                sem          ：Semaphore
                appMsgQueue  ：Message handle
【Return】   no
【Description】   no
*********************************************************************************/
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);                                    //not must
    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,
                                    TestUartWritetCB,                              //writeCB
                                    TestUartReadCB,                                 //readCB
                                    &uartHandle,
                                     uartParam);

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

     // When the test timer's input is captured, it needs to be set to false, ie, do not open the GPIO to
prevent GPIO conflicts
     #if 0
       TTCBleSDKManagerInfo_t Err;
       Err = TTCDriverIOOpen(&uLedHandle,&uLedState,(const PIN_Config *)uLedConfig);
       if(MANAGER_INFO_REQUEST_IO_SUCCESS == Err){
          asm("nop");
       }
     #endif
      #if 0                              // When testing other drivers here need to be set to false to
ensure the effect of printing
```

```
        Util_constructClock(&uClock,        // Periodic printing "TTCDriverUART Test\r\n"
                            TTCDriverDemoUARTClockHandler,
                            1000,
                            0,
                            true,
                            TTCBLE_SDK_UART_EVN);
    #endif
}

/*******************************************************************************
【Function】  u8 TTCDriverVal2Str(uint32 val,uint8 * buf)
【Overview】  Converts a number to a string
【Parameter】  val : number which need to convert
               buf ：Converted result storage area
【Return】  The length of the string
【Description】  no
*******************************************************************************/
u8 TTCDriverVal2Str(u32 val,u8 * buf){
    u8   len = 0,i=0;
    u32 nr = 0;
    if(buf != NULL){
        nr = val;
        do{                                          // Calculate the length of the data
            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 Timer Description

The CC2640 has eight 16-bit timers, each of which can be individually configured for different modes. Support programmable counting mode, and it also supports the simultaneous start of more than one timer for some special use of the occasion.

4.3.1 Timer API Description

4.3.1.1 TTCDriverTimerInit()

```
/*******************************************************************************
【Function】 TTCDriverTimerInit(TTCDriverTimerInit_t timerInitParam,
                                Timer_Handle * timerHandle)
【Overview】 Timer initialization
【Parameter】 appCallbacks : Registration callback
              timerInitParam : Timer initialization parameter
              timerHandle : Timer handle
【Return】 Please refer to Appendix A TTCDriverInfo_t
【Description】 If Timer_Handle is not initialized, it can not be used
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerInit(TTCDriverTimerInit_t timerInitParam,
                                          Timer_Handle * timerHandle);
Description:Timer initialization 。
```

4.3.1.2 TCDriverTimerRegisterIntCallBack()

```
/*******************************************************************************
【Function】 TCDriverTimerRegisterIntCallBack(Timer_Handle * timerHandle,
                                              TTCSdkTimerCB_t pCB)
【Overview】 The timer sets the interrupt function callback
【Parameter】 timerHandle : Timer handle
              pCB : Set the timer interrupt callback function
【Return】 Please refer to TTCDriverInfo_t
【Description】 If Timer_Handle is not initialized, it can not be used
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerRegisterIntCallBack(Timer_Handle * timerHandle,
                                                         TTCSdkTimerCB_t pCB);
```

Description: Register various interrupt callback function, such as timing interrupt callback function, input count Interrupt callback function, input capture Interrupt callback function and so on.

4.3.1.3 TTCDriverTimerCounterSetParam()

```
/*******************************************************************************
【Function】  TTCDriverTimerCounterSetParam(Timer_Handle * timerHandle,
                                            TTCDriverTimerCounterParam_t param,
                                            bool enable)
【Overview】  Timer count parameter setting
【Parameter】  timerHandle : Timer handle
               param       : Count parameter
               enable      : Whether it is enabled
【Return】  Please refer to TTCDriverInfo_t
【Description】  If Timer_Handle is not initialized, it can not be used
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerCounterSetParam(Timer_Handle * timerHandle,
                                                     TTCDriverTimerCounterParam_t param,
                                                     bool enable);
```

Description: Timer timing function configuration, including the timing of the cycle and so on.

4.3.1.4 TTCDriverTimerEdgCountSetParam()

```
/*******************************************************************************
【Function】  TTCDriverTimerEdgCountSetParam(Timer_Handle * timerHandle,
                                             TTCDriverTimerEdgeCountParam_t param,
                                             PIN_Id edgeCountPin,
                                             bool enable)
【Overview】  Timer edge count mode parameter setting
【Parameter】  timerHandle  : Timer handle
               param        : Count parameter
               edgeCountPin : Input capture pin
               enable       : Whether it is enabled
【Return】  Please refer to TTCDriverInfo_t
【Description】  If Timer_Handle is not initialized, it can not be used
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerEdgCountSetParam(Timer_Handle * timerHandle,
                                                      TTCDriverTimerEdgeCountParam_t
param,
                                                      PIN_Id edgeCountPin,
                                                      bool enable);
```

Description: The external input count function of the timer is configured, including the number of counts (ie, how many counts are generated), overflows, and so on.

4.3.1.5 TTCDriverTimerEdgTimingSetParam()

```
/*******************************************************************************
【Function】 TTCDriverTimerEdgTimingSetParam(Timer_Handle * timerHandle,
                                              TTCDriverTimerEdgeTimingParam_t param,
                                              PIN_Id edgeTimerPin,
                                              bool enable)
【Overview】 Timer edge timing mode parameter setting
【Parameter】 timerHandle : Timer handle
              param : Edge timing parameter
              edgeTimerPin : Input capture pin
              enable : Whether it is enabled
【Return】 Please refer to TTCDriverInfo_t
【Description】 If Timer_Handle is not initialized, it can not be used
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerEdgTimingSetParam(Timer_Handle * timerHandle,
                                                       TTCDriverTimerEdgeTimingParam_t
param,
                                                       PIN_Id edgeTimerPin,
                                                       bool enable);
```

Description: The timer input capture function configuration, including the captured edges, the direction of the count, and so on.

4.3.1.6 TTCDriverTimerPwmSetParam()

```
/*******************************************************************************
【Function】 TTCDriverTimerPwmSetParam(Timer_Handle * timerHandle,
                                       TTCDriverTimerPwmParams_t param,
                                       PIN_Id pwmPin,
                                       bool enable)
【Overview】 Timer PWM mode parameter setting
【Parameter】 timerHandle : Timer handle
              param : Edge timing parameter
              pwmPin : PWM OUTPUT PIN
              enable : Whether it is enabled
【Return】 Please refer to TTCDriverInfo_t
【Description】 If Timer_Handle is not initialized, it can not be used。
              pwmPin If not required, set to PIN_UNASSIGNED.
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerPwmSetParam(Timer_Handle * timerHandle,
                                                 TTCDriverTimerPwmParams_t param,
                                                 PIN_Id pwmPin,
```

bool enable);

Description: The PWM configuration of the timer includes the cycle and duty cycle, as well as the polarity of the active level.

4.3.1.7 TTCDriverTimerStop()

```
/*******************************************************************************
【Function】 TTCDriverTimerStop(Timer_Handle * timerHandle)
【Overview】 The timer stops
【Parameter】 timerHandle : Timer handle
【Return】 Please refer to TTCDriverInfo_t
【Description】 1.If Timer_Handle is not initialized, it can not be used
2. This function will not clear the relevant configuration, when you want to open directly call TTCDriverTimerStart run again
3. If the timer is in PWM mode, calling this function will cause the PWM OUTPUT PIN output to set the idle power level
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerStop(Timer_Handle * timerHandle);
```

Description: Temporarily shut down a timer.

4.3.1.8 TTCDriverTimerStart()

```
/*******************************************************************************
【Function】 TTCDriverTimerStart(Timer_Handle * timerHandle)
【Overview】 The timer starts
【Parameter】 timerHandle : Timer handle
【Return】 Please refer to TTCDriverInfo_t
【Description】 If Timer_Handle is not initialized, it can not be used
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverTimerStart(Timer_Handle * timerHandle);
```

Description: Turn on a timer.

4.3.1.9 TTCDriverTimerClose()

```
/*******************************************************************************
【Function】 TTCDriverTimerClose(Timer_Handle * timerHandle)
【Overview】 The timer is off
【Parameter】 timerHandle : Timer handle
【Return】 Please refer to TTCDriverInfo_t
【Description】 1.If Timer_Handle is not initialized, it can not be used
```

2. After calling this function, if you want to run the timer again, you need to call the setup function again. Call TTCDriverTimerStart no effect

**********/

extern TTCDriverInfo_t TTCDriverTimerClose(Timer_Handle * timerHandle);

Description: Directly shut down a timer, to re-open the need to re-configure.

4.3.1.10 TTCDriverTimerSync()

/**********

【Function】 TTCDriverTimerSync

【Overview】 Select the timer to start synchronously

【Parameter】 timerHandle : Timer handle

syncTimer : Select the timer that needs to be synchronized

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

【Return】 no

【Description】 Call the function after the associated timer starts.

**********/

extern void TTCDriverTimerSync(Timer_Handle * timerHandle,CC2650_TimerSync syncTimer);

Description: Used to synchronize the start of multiple timers, such as two timers are used for PWM output, let it open together to ensure that the output signal at the same time output.

4.3.2 Example of timer use

/**********

【File】 TTCDirverTimerDemo.c

【Overview】 TTC SDK Timer demo code

【Edit】 SDK WORKING GROUP

【Revise】 SDK WORKING GROUP

【Revised date】 2016/12/01

【Version】 V1.0.0

【Description】

Function Description:

1, configure any timer output PWM signal, and let the PWM output switch back and forth between the

Board_PWM0 and Board_PWM7 pin .

2, configuration timer interrupt function for any timer, flip Board_PWM1 pin output status in the Interrupt callback function

3, configure the timer CC2650_TIMER0_A from the Board_PWM0 pin output PWM signal;

Configure the specified timer as the input count mode. The signal is input from the board_PWM2 pin and the output status of the board_PWM3 pin is inverted in the Interrupt callback function.

4, configure the timer CC2650_TIMER0_A from the Board_PWM0 pin output PWM signal;

Configure the specified timer TimerName for the input capture mode, the signal input from the Board_PWM2 pin, in the capture Interrupt callback function flip Board_PWM7, and if the UART driver is opened, it will print through the UART twice the time difference (this difference is the input signal The time between the two edges.) The serial debugging assistant displays the number of waveforms captured between the two rising edges (48 MHz clock source) to 48001. (Note: 1 / 48MHz * 48001 = 1000.02us, in line with DIO_7 reversal time 1ms.)

Adding steps:

1． add the corresponding head files TTCDirverTimerDemo.h

2. Define TTCDRIVER_TIMER in OpIOns / C / C ++ Compiler / Defined symbols in project.

3． in the TTCBlePeripheralTaskFxn () function to add

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

4． add below code to the static void TTCSDKDriverInit (void) function in the main thread.

Note: Only one case can be tested at a time

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

Note：

((1) When you want to use the Transparent Development Kit to debug the timer, unplug all the jumpers on the kit except 3.3V,

(2) In addition, if you want to use the logic analyzer to catch the waveform, please set the appropriate frequency of crawl.

```
*********************************************************************************/
#ifdef TTCDRIVER_TIMER
/*********************************************************************
 * head files
 */
#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"
/*********************************************************************
 * Constants and macro definitions
 */
#define TTCBLE_SDK_TIME_EVN                         0x0001

/*********************************************************************
 * Local variable
 */
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;                                          //Thread
semaphore，used to wake up threads
static  Queue_Handle  *  TimMsgQueue;                                        //Message
handle
```

```
static u16 events;                                                    //local event

/*********************************************************************
 * Local function declaratIOn
 */
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 );

/*********************************************************************************
【Function】    TTCDriverDemoTimerClockHandler(UArg arg)
【Overview】    Thread set event
【Parameter】    arg      : Marks the Event source event
【Return】    no
【Description】
*********************************************************************************/
static void TTCDriverDemoTimerClockHandler(UArg arg){
    TTCSDKDriverTIMSetEvent(arg);
}
/*********************************************************************************
【Function】    TTCDriverDemoTimerSetEvent(UArg arg)
【Overview】
```

```
【Parameter】   arg：Event sign
【Return】   no
【Description】   no
*********************************************************************************/
static void TTCDriverDemoTimerSetEvent(UArg arg){
    events |= arg;
    Semaphore_post(*TimSem);
}
/*********************************************************************************
【Function】   TTCDriverDemoTimerInit(ICall_Semaphore * sem,
                                      Queue_Handle * appMsgQueue,
                                      CC2650_TimerName TimerName)
【Overview】   Specified Timer initialization n
【Parameter】   sem: Semaphore of the main thread
                appMsgQueue: Main thread queue
                TimerName: Timer name
【Return】   See Appendix A TTCDriverInfo_t
【Description】
*********************************************************************************/
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){          //Initialization was unsuccessful
            return TimErrCode;
        }
    }else{                                                 // The input parameter is incorrect
        return TTCDRIVER_INIT_SYSPARA_FAILED;
    }
    return TTCDRIVER_INIT_SUCCESS;
}
```

```
/*******************************************************************************
【Function】  TTCSDKDriver_PWMInit(CC2650_TimerName TimerName,uint8 brd)
【Overview】  The specified timer PWM output initialization
【Parameter】  nTmr ：  Timer number
               brd ：  The pin number of the output
【Return】  See Appendix A TTCDriverInfo_t
【Description】  Note：This function should be executed after TTCSDKDriver_TIM_Init ()
*******************************************************************************/
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                               //idle power level
    };
    i = TimerName/2;
    j = TimerName%2;

    TimErrCode = TTCDriverTimerPwmSetParam(&Handle[i][j],     //Timer handle
                                           param,             //PWM parameter
                                           brd,               //Specified PWM signal output pin
                                           false);            //Enable the timer immediately false
true

    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){                 //PWMInitialization was
unsuccessful
        return TimErrCode;
    }
    TTCDriverTimerStart(&Handle[i][j]);
    return TTCDRIVER_INIT_SUCCESS;
}

/*******************************************************************************
【Function】  TTCSDKDriver_Inter_Init(CC2650_TimerName TimerName)
【Overview】  配置指定定时器的中断功能
【Parameter】  TimerName：Timer number
【Return】  See Appendix A TTCDriverInfo_t
【Description】  Note：This function should be executed after TTCSDKDriver_TIM_Init ()
*******************************************************************************/
```

```
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;
}

/*****************************************************************************
【Function】   TTCSDKDriver_Cat_Init(CC2650_TimerName TimerName,uint8 brd)
【Overview】   指定定时器的输入捕获初始化
【Parameter】   nTmr ：  Timer number
                brd ：   The pin number of the output
【Return】   See Appendix A TTCDriverInfo_t
【Description】   Note：This function should be executed after TTCSDKDriver_TIM_Init ()
*****************************************************************************/
static TTCDriverInfo_t TTCSDKDriver_Cat_Init(CC2650_TimerName TimerName,uint8 brd){
    uint8 i,j;

    TTCDriverTimerEdgeTimingParam_t param = {
        .countValue                 = 65535,
        .upCountMode                = 1,                        // Count up
        .edgeMode                   = TIMER_POSITIVE_EDGE,      // Rising edge capture
    };

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

    TimErrCode = TTCDriverTimerEdgTimingSetParam(&Handle[i][j],
                                                 param,
                                                 brd,           // Set the input port
```

```
                                                    true);
    return TimErrCode;
}
/*******************************************************************************
【Function】  TTCSDKDriver_Count_Init(CC2650_TimerName TimerName,uint8 brd)
【Overview】  Specified timer input initialized
【Parameter】  nTmr ：  Timer number
               brd ：  The pin number of the output
【Return】  See Appendix A TTCDriverInfo_t
【Description】  Note：This function should be executed after TTCSDKDriver_TIM_Init ()
*******************************************************************************/
static TTCDriverInfo_t TTCSDKDriver_Count_Init(CC2650_TimerName TimerName,uint8 brd){
    uint8 i,j;
    TTCDriverTimerEdgeCountParam_t EdgeCountParam = {
        .countValue          = 1000,
        .upCountMode         = TRUE,                                    //count up
        .matchValue          = 10,                                      //10
        .edgeMode            = TIMER_POSITIVE_EDGE,                     //rising
edge
    };

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

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

/*******************************************************************************
【Function】  TTCSDKDriver_Callback_Register(CC2650_TimerName TimerName,TTCSdkTimerCB_t
func)
【Overview】  Register the callback function of the specified timer
【Parameter】  TimerName：Timer number
               Func：  callback function
【Return】  See Appendix A TTCDriverInfo_t
【Description】  Note：This function should be executed after TTCSDKDriver_TIM_Init ()
*******************************************************************************/
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){                    // registration failed
            return TimErrCode;
        }
    }else{
        return TTCDRIVER_INIT_PARA_ERROR;
    }
    return TTCDRIVER_INIT_SUCCESS;
}

/*****************************************************************************
 【Function】  TTCSDKDriver_Inst_IO_Init(PIN_Config * config)
 【Overview】  Interrupt indication IO INITIALIZATION
 【Parameter】  config：IO port configuration information
 【Return】  See Appendix A TTCBleSDKManagerInfo_t
 【Description】  If config is empty, press the default setting
*****************************************************************************/
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; // Returns the error code
}

/*****************************************************************************
 【Function】  TTCDriveTimerFuncIsrCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue)
 【Overview】  Timed interrupt call processing function
 【Parameter】  isrInfo :
                timerCurrectValue：
 【Return】  no
 【Description】  no
*****************************************************************************/
static void TTCDriveTimerFuncIsrCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue){
    TTCDriverIOSetOutputVaule(&EdGeHandle,
```

```
                                Board_PWM1,
                                !TTCDriverIOGetOutputValue(Board_PWM1));
}

/*****************************************************************************
【Function】  TTCDriveTimerFuncCountCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue)
【Overview】  Timer count callback processing function
【Parameter】  isrInfo :
               timerCurrectValue：
【Return】  no
【Description】  no
*****************************************************************************/
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));    // Print count value
#endif

}

/*****************************************************************************
【Function】  TTCDriveTimerFuncCaptureCB(TTCDriverTimerIsrInfo_t isrInfo,u32 timerCurrectValue)
【Overview】  The timer input captures the callback handler
【Parameter】  isrInfo :
               timerCurrectValue：
【Return】  no
【Description】  Where the print function does not support the case of too high input waveforms, 100K is
the upper limit of test at 256000bps
*****************************************************************************/
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){                    // To avoid some special time period test when it can be handled like
this
              #ifdef TTCDRIVER_UART
                uint8 buf[20] = {0};
                TTCDriverUartWrite(&uartHandle,buf,TTCDriverVal2Str((u32)deta,buf));
              #endif
            }
        }

        cnt++;
        if(cnt == 2){
           cnt = 0;
        }
}

/*****************************************************************************
【Function】  void TTCDriverSinglePWMSignalTestOther(CC2650_TimerName TimerName)
【Overview】  Output all the way after the PWM signal is closed and turned on, 20 times after the cycle
began to change the duty cycle
【Parameter】  TimerName：Timer number
【Return】  no
【Description】  Other functional tests on PWM, including the control of idle power level
*****************************************************************************/
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){                                                    //on/off test
            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,                                  //Change the duty cycle each time
                PWM_OUTPUT_NOT_INVERTED,
                false,
                0
            };
            TimErrCode = TTCDriverTimerPwmSetParam(&Handle[i][j],     //Timer handle
                                                   param,             //PWM parameter
                                                Board_PWM0,        //Specified PWM signal output
pin
                                                true);             //Enable the timer immediately

        }else{
            TTCDriverTimerPwmParams_t param = {
                48000,                          //48000000/48000 = 1000Hz
                36000,                          //Change the duty cycle each time
                PWM_OUTPUT_NOT_INVERTED,
                false,
                0
            };
            TimErrCode = TTCDriverTimerPwmSetParam(&Handle[i][j],   //Timer handle
                                                   param,           //PWM parameter
                                                   Board_PWM0,      // Specified output pin
                                                   true);       //Enable the timer immediately
        }
        if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
            return;
        }
    }else{
       Ct = 0;
    }
    Util_startClock(&TimerClock);
}

/*********************************************************************************
【Function】  void TTCDriverSinglePWMSignalTest(ICall_Semaphore * sem,
                                                Queue_Handle * appMsgQueue,
                                                CC2650_TimerName TimerName)
【Overview】  Output a PWM signal, and then switch back and forth between PWM0 and PWM7 outputs
【Parameter】  sem：the Main Thread semaphore
```

```
                appMsgQueue：Main thread message queue
                TimerName: timer
【Return】   no
【Description】   no
*********************************************************************************/
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);   // Initialize the timer
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    TimErrCode = TTCSDKDriver_PWMInit(TimerName,IOID_0);// Configure the PWM function, the signal output from the Board_PWM0
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    Util_constructClock(&TimerClock,                    //Configure a software timer
                        TTCSDKDriverTIMclockHandler,
                        100,                            //Timing 100mS
                        0,
                        true,                           // Start now
                        TTCBLE_SDK_TIME_EVN);
}

/*********************************************************************************
【Function】   TTCDriverSinglePWMSignalDispose(CC2650_TimerName TimerName)
【Overview】   Achieve a PWM timer output switch back and forth between the PWM0-PWM7
【Parameter】   TimerName：Timer number
【Return】   no
【Description】Test check IOID_0 - IOID_7 is available (not being applied), the other need to actually look at the corresponding pin has no external circuit (jump cap on"transparent transmission Kit" , as well as download line need to pull out)
*********************************************************************************/
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],        //Timer handle
                                           param,                 //PWM parameter
                                           Board_PWM0+cntt,     //Specified PWM signal output
pin
                                           true);                 //Enable the timer immediately

    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    Util_startClock(&TimerClock);
}


/*********************************************************************************
 【Function】   void TTCDriverSingleInterSignalTest(ICall_Semaphore * sem,
                                                    Queue_Handle * appMsgQueue,
                                                    CC2650_TimerName TimerName)
 【Overview】   Configure a timer interrupt
 【Parameter】   sem：the main thread semaphore
                 appMsgQueue：Main thread message queue
                TimerName: timer
 【Return】   no
 【Description】   Flip the board_PWM1 in the time Interrupt callback function
*********************************************************************************/
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);     //Initialize the timer
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    //Register the callback function
    TimErrCode = TTCSDKDriver_Callback_Register(TimerName,TTCDriveTimerFuncIsrCB);
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    TTCBleSDKManagerInfo_t ErrCode;
    ErrCode = TTCSDKDriver_Inst_IO_Init(NULL); // Initialize the GPIO which need to use , NULL means use the default GPIO

    if(ErrCode != MANAGER_INFO_REQUEST_IO_SUCCESS){
        return;
    }
    TimErrCode = TTCSDKDriver_Inter_Init(TimerName);     // Configure the timer interrupt

}

/*****************************************************************************
【Function】  void TTCDriverSingleCountSignalTest(ICall_Semaphore * sem,
                                                  Queue_Handle * appMsgQueue,
                                                  CC2650_TimerName TimerName)
【Overview】  Configure an external input count interrupt
【Parameter】  sem：the main thread semaphore
               appMsgQueue：Main thread message queue
               TimerName: timer
【Return】  no
【Description】  This function sets the timer CC2650_TIMER0_A to output the PWM signal from the board_PWM0 pin;Set the specified timer TimerName as input count mode, the signal input from the Board_PWM2 pin;flip Board_PWM3 in the count Interrupt callback function
            Note：The actual output of the Board_PWM3 waveform you can see in a pulse width contains 11 rising edges, because our output is not reasonable, the actual count value we can get in the count interrupt callback.
*****************************************************************************/
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);     //Initialize
the timer
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    //Configure the PWM function, the signal output from the Board_PWM0
    TimErrCode = TTCSDKDriver_PWMInit(CC2650_TIMER0_A,Board_PWM0);
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }

    TimErrCode    =    TTCSDKDriver_TIM_Init(sem,appMsgQueue,TimerName);//Initialize    the    timer
TimerName
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return ;
    }
    //注册计数 Interrupt callback function
    TimErrCode = TTCSDKDriver_Callback_Register(TimerName,TTCDriveTimerFuncCountCB);
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }

    TTCBleSDKManagerInfo_t ErrCode;
    ErrCode = TTCSDKDriver_Inst_IO_Init(NULL);                              //IO INITIALIZATION
    if(ErrCode != MANAGER_INFO_REQUEST_IO_SUCCESS){
        return;
    }
    // Configure the count function, the signal    input from the Board_PWM2
    TimErrCode = TTCSDKDriver_Count_Init(TimerName,Board_PWM2);
}

/*****************************************************************************
【Function】    void TTCDriverSingleCatSignalTest(ICall_Semaphore * sem,
                                                  Queue_Handle * appMsgQueue,
```

CC2650_TimerName TimerName)

【Overview】 Configure an input capture interrupt

【Parameter】 sem：the main thread semaphore

appMsgQueue：Main thread message queue

TimerName: timer

【Return】 no

【Description】 This function sets the timer CC2650_TIMER0_A output the PWM signal from the board_PWM0 pin;

Set the specified timer TimerName is the input capture mode, the signal input from the Board_PWM2 pin;

flip Board_PWM7 in the count Interrupt callback function, while the serial port to print the time difference between the twice captured (the difference is the input signal between the two edges)

Note: This is useful for UART driver, so please set up the UART-related content.

**********/

```
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);  //Initialize the timer
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    //Configure the PWM function, the signal output from the Board_PWM0
    TimErrCode = TTCSDKDriver_PWMInit(CC2650_TIMER0_A,Board_PWM0);
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }

    TimErrCode  =  TTCSDKDriver_TIM_Init(sem,appMsgQueue,TimerName);//Initialize the timer TimerName
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
     //Register the callback function
```

```
    TimErrCode = TTCSDKDriver_Callback_Register(TimerName,TTCDriveTimerFuncCaptureCB);
    if(TimErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    TTCBleSDKManagerInfo_t ErrCode;
    ErrCode = TTCSDKDriver_Inst_IO_Init(NULL);     //IO INITIALIZATION
    if(ErrCode != MANAGER_INFO_REQUEST_IO_SUCCESS){
        return;
    }
    // Configure the capture function, the signal input from the Board_PWM2
    TimErrCode = TTCSDKDriver_Cat_Init(TimerName,Board_PWM2);
}
/*******************************************************************************
 【Function】    TTCSDKDriverDemoTimerEvent(UArg arg)
 【Overview】    The local event handler
 【Parameter】    no
 【Return】    no
 【Description】    no
*******************************************************************************/
void TTCSDKDriverDemoTimerEvent(void){
    if(events & TTCBLE_SDK_TIME_EVN){
        events &= ~TTCBLE_SDK_TIME_EVN;
        TTCDriverSinglePWMSignalDispose(CC2650_TIMER0_A);
    }
}

/*******************************************************************************
 【Function】    TTCSDKDriverTIMInit(ICall_Semaphore * sem,
                                    Queue_Handle * appMsgQueue,
                                    CC2650_TimerName TimerName)
 【Overview】    Timer Example Initialization
 【Parameter】    sem          : Semaphore
                 appMsgQueue  : Message handle
                 TimerName    : timer
 【Return】    no
 【Description】    test one at a time, or there will be a conflict
*******************************************************************************/
void TTCSDKDriverTIMInit(ICall_Semaphore * sem,
                         Queue_Handle * appMsgQueue,
                         CC2650_TimerName TimerName){
    //PWM output
    TTCDriverSinglePWMSignalTest(sem,appMsgQueue,TimerName);
```

```
        //Timed interrupt
//      TTCDriverSingleInterSignalTest(sem,appMsgQueue,TimerName);
        // External input count (here can not be set to CC2650_TIMER0_A)
//      TTCDriverSingleCountSignalTest(sem,appMsgQueue,TimerName);
        // External input capture (here can not be set to CC2650_TIMER0_A)
//      TTCDriverSingleCatSignalTest(sem,appMsgQueue,TimerName);

}

#endif
```

4.4 ADC DESCRIPTION

The CC2640 has an 8-channel 12-bit ADC channel that supports a sampling rate of 200K samples, and its clock source is freely configurable, including timers, I / O pins, software, analog comparators and RTC. In addition it can be collected the current temperature value to the On-chip temperature sensor and through the internal circuit to collect the power supply voltage, easy to achieve battery management.

The CC2640 5 * 5 package supports 8-channel analog inputs, and DIO_7 to DIO_14 can be set as analog input pins for the ADC. The ADC reference voltage can be set to VDDS, or the internal reference voltage (eg 1.43V, 4.3V, etc.), but does not support external reference voltage.

4.4.1 ADC API INSTRUCTIONS

4.4.1.1 TTCDrvierAdcReadSync()

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

【Function】 TTCDrvierAdcReadSync

【Overview】 Synchronous ADC sampling

【Parameter】 adconfig :ADC configuration parameter

simpleTimes:Set the number of consecutive samples for the specified channel, the sampling frequency range 1-255

adcReadBuf :Save the AD value cache, the cache size should be greater than or equal to the number of sampling. Note that must not be less than the number of sampling.

【Return】 ADC SAMPLE VALUE

【Description】

adconfig.refsource Reference voltage source：

AUXADC_REF_FIXED (nominally 4.3 V)

AUXADC_REF_VDDA_REL (nominally VDDS)

adconfig.auxIO Internal, external input channel selection：

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);

Description:

4.4.1.2 TTCDrvierAdcReadAsync()

/**********

【Function】 TTCDrvierAdcReadAsync

【Overview】 Asynchronous ADC sampling

【Parameter】 adconfig :ADC configuration parameter

simpleTimes:Set the number of consecutive samples for the specified channel, the sampling frequency range 1-255

adcReadBuf :Save the AD value cache, the cache size should be greater than or equal to the number of sampling. Note that must not be less than the number of sampling.

【Return】 ADC SAMPLE VALUE

【Description】 adconfig.time setting invalid

adconfig.nominally Reference voltage source：

AUXADC_REF_FIXED (nominally 4.3 V)
AUXADC_REF_VDDA_REL (nominally VDDS)

adconfig.auxIO Internal, external input channel selection：

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);

Description:

4.4.1.3 TTCDriverBatVoltageGet()

/**********

【Function】 TTCDriverBatVoltageGet

【Overview】 Read the battery AD value

【Parameter】 no

【Return】 Battery voltage (AD value)

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

Return Value 32 [10: 8] represents INT. [7: 0] represents FRAC, for fractional parts,

A unit representing 0.00390625v, the fractional part of the resolution is only 50mV (TYP)

**********/

extern u32 TTCDriverBatVoltageGet(void);

Description: Call this API to get the AD value of the supply voltage directly.

4.4.1.4 TTCDriverBatTempClose()

/**********

【Function】 TTCDriverBatTempClose

【Overview】 Turn off the sampling voltage, temperature AD value

【Parameter】 no

【Return】 no

【Description】

**********/

extern void TTCDriverBatTempClose(void);

Description:

4.4.1.5 TTCDriverTempGet()

/**********

【Function】 TTCDriverTempGet

【Overview】 Read the temperature value

【Parameter】 no

【Return】 Temperature (Celsius)

【Description】 The return value is Celsius

***/

extern u32 TTCDriverTempGet(void);

Description: Read the AD value of the On-chip temperature sensor.

4.4.2 ADC USE EXAMPLES

```
/*****************************************************************************
【File】      TTCDriverADCDemo.c
【Overview】      TTC SDK ADC demo code
【Edit】    SDK WORKING GROUP
【Revise】      SDK WORKING GROUP
【Revised date】      2016/12/02
【Version】      V1.0.0
【Description】      Adding steps:
                (1) first add the corresponding head files TTCDriverADCDemo.h
(2) Define TTCDRIVER_ADC in OpIOns / C / C ++ Compiler / Defined symbols in project
(3) in the TTCBlePeripheralTaskFxn () function to add
                #ifdef TTCDRIVER_ADC
                   TTCSDKDriverDemoADCEvent ();
                #endif // TTCDRIVER_ADC
(4) static void TTCSDKDriverInit (void) function in the main thread to add
                #ifdef TTCDRIVER_ADC.
                   TTCDriverDemoADCInit (& sem, & appMsgQueue);
                #endif
*****************************************************************************/
#ifdef TTCDRIVER_ADC
/*****************************************************************
 * head files
 */
#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"
#include "TTCDriverADCDemo.h"
```

```
#include "bcomdef.h"
#include "TTCBleProfile.h"

/*********************************************************************
 * Constants and macro definitions
 */
#define TTCBLE_SDK_ADC_EVT                0x0001              //ADC EVENT

/*********************************************************************
 * Local variable
 */
static u16 adc[20]={0};                                    // ADC sampling result cache
static Clock_Struct adcClock;                              // ADC timing sampling clock
static TTCDriverAdcConfing adconfig;                        // ADC configuration structure

static PIN_Handle adPinHandle;                              //ADC pin
static PIN_State   adPinState;

const PIN_Config adPinConfig[] = {
    IOID_7 | PIN_GPIO_OUTPUT_DIS | PIN_INPUT_EN | PIN_NOPULL,   // Configured as a floating input
    PIN_TERMINATE
};
static ICall_Semaphore *AdcSem;                             //Thread semaphore，used to wake up threads
static Queue_Handle *AdcMsgQueue;                           //Message handle
static u16 adcDemoEvents;                                  //ADC EVENT

/*********************************************************************
 * Local function declaratIOn
 */
static void TTCSDKDriverADCRead(void);
static void TTCDriverDemoADCClockHandler(UArg arg);
static void TTCDriverDemoADCSetEvent(UArg arg);

/*********************************************************************************
【Function】   TTCDriverDemoADCClockHandler(UArg arg)
【Overview】   Timing task callback function
【Parameter】   arg : Tag events
【Return】   no
【Description】   no
```

```
*****************************************************************************/
static void TTCDriverDemoADCClockHandler(UArg arg){
    TTCDriverDemoADCSetEvent(arg);
}

/*****************************************************************************
【Function】  TTCDriverDemoADCInit(ICall_Semaphore * sem,Queue_Handle * appMsgQueue)
【Overview】  ADC SAMPLE INITIALIZATION
【Parameter】  sem          ：Semaphore
               appMsgQueue  ：Message handle
【Return】  no
【Description】  no
*****************************************************************************/
void TTCDriverDemoADCInit(ICall_Semaphore * sem,Queue_Handle * appMsgQueue){
      if(sem == NULL || appMsgQueue == NULL){
      return;
    }

    AdcSem      = sem;
    AdcMsgQueue = appMsgQueue;
    (void)AdcMsgQueue;
  // First set the corresponding GPIO to float input, we must first configure the corresponding GPIO for the floating input, otherwise the conVersion of the voltage value is wrong.
    TTCBleSDKManagerInfo_t ErrCode;
    ErrCode = TTCDriverIOOpen(&adPinHandle,&adPinState,(const PIN_Config *)adPinConfig);
    if(MANAGER_INFO_REQUEST_IO_SUCCESS != ErrCode){
      return;
    }

    adconfig.refsource = ADC_REF_FIXED;
    adconfig.ref       = ADC_REF_4P3V;
    adconfig.time      = ADC_TIME_2P7_US;
    adconfig.trigger   = ADC_TRIGGER_GPT0A;   // There is no effect on the use of the timer
    adconfig.auxIO=   ADC_COMPB_IN_AUXIO7;   // Correspondence Please check the TTCDriverADC.h file

    Util_constructClock(&adcClock,              // Configure a software timer, periodically calls the ADC acquisition function
                        TTCDriverDemoADCClockHandler,
                        100,                    //Timing 100mS
                        0,
                        true,
```

```
                                TTCBLE_SDK_ADC_EVT);
}


/*******************************************************************************
【Function】   TTCDriverDemoADCSetEvent(UArg arg)
【Overview】   Tag events and wake up threads
【Parameter】   arg : Tag events
【Return】   no
【Description】   no
*******************************************************************************/
static void TTCDriverDemoADCSetEvent(UArg arg){
    adcDemoEvents |= arg;
    Semaphore_post(*AdcSem);                        //Wake up thread
}

/*******************************************************************************
【Function】   TTCSDKDriverDemoADCEvent(void)
【Overview】   Demo ADC event handling
【Parameter】   no
【Return】   no
【Description】   no
*******************************************************************************/
void TTCSDKDriverDemoADCEvent(void){
    if(adcDemoEvents & TTCBLE_SDK_ADC_EVT){
        adcDemoEvents &= ~TTCBLE_SDK_ADC_EVT;
        TTCSDKDriverADCRead();
    }
}

/*******************************************************************************
【Function】   TTCSDKDriverADCRead(void)
【Overview】
【Parameter】   no
【Return】   no
【Description】   no
*******************************************************************************/
static void   TTCSDKDriverADCRead(void){
    TTCDrvierAdcReadSync(adconfig,20,(u16 *)adc);              // Continuous collection 20 times
    u8 i=0;
    u32 sum = 0;
    u8   adcValue[2] = {0};
```

```
        for(i=0;i<20;i++){
            sum += adc[i];
        }
        sum /= 20;
        // Decimal to hexadecimal for APP to view the results. Such as sum for 1234, APP shows 0x1234
        adcValue[0] = (sum / 100) / 10 * 16 + (sum / 100) % 10;
        adcValue[1] = (sum % 100) / 10 * 16 + (sum % 100) % 10;
        // After receiving the Bluetooth data, the data is sent back to the host via the UUID 1002 feature
        TTCBleProfileSetParameter(TTCBLE_PROFILE_CHAR2, 2, adcValue);
        Util_startClock(&adcClock);
}
#endif //TTCDRIVER_ADC
```

4.5 UTC INSTRUCTIONS

CC2640 RTC clock come from the 32Khz external crystal, in the case of continuous power will always be automatically counted. In addition it also has a 70bit programmable counter and three common channels with its compare register to use, can generate time-related notifications to inform the application layer to achieve some of the necessary functions.

4.5.1 UTC API DESCRIPTION

4.5.1.1 TTCDriverUTCInit()

```
/*******************************************************************************
【Function】  TTCDriverInfo_t TTCDriverUTCInit(TTCSdkClass_t *appCallbacks,
                                                TTCDriverUTCParams_t utcConfig)
【Overview】  Initialize UTC
【Parameter】  appCallbacks : Registration callback
               utcConfig    : UTC CONFIGURATION INFORMATION
【Return】  Please refer to TTCDriverUartInfo_t
【Description】
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverUTCInit(TTCSdkClass_t *appCallbacks,
                                        TTCDriverUTCParams_t utcConfig);
```

Description: Initialize UTC, register callback function. UTC callback function passed by appCallbacksparameter.

4.5.1.2 TTCDriverUTCReschedule

```
/*******************************************************************************
【Function】  TTCDriverInfo_t TTCDriverUTCReschedule(TTCDriverUTCParams_t utcConfig)
```

【Overview】 Set UTC
【Parameter】 utcConfig : UTC CONFIGURATION INFORMATION
【Return】 Please refer to TTCDriverUartInfo_t
【Description】
*****/

```
extern TTCDriverInfo_t TTCDriverUTCReschedule(TTCDriverUTCParams_t utcConfig);
```

Description:Set UTC,used to update UTC time.

4.5.1.3 TTCDriverUTCGetClock()

/*****

【Function】 TTCDriverUTCGetClock(TTCDriverUTCTime_t * time)
【Overview】 Read the current clock
【Parameter】 time : Time structure
【Return】 no
【Description】 no
*****/

```
extern void TTCDriverUTCGetClock(TTCDriverUTCTime_t * time);
```

Description: Get information about the current UTC.

4.5.2 UTC USE EXAMPLES

/**/*****

【File】 TTCDriverUTCDemo.c
【Overview】 TTC SDK UTC demo code
【Edit】 SDK WORKING GROUP
【Revise】 SDK WORKING GROUP
【Revised date】 2016/12/02
【Version】 V1.0.0
【Description】 Adding steps:
First add the corresponding head files TTCDriverUTCDemo.h
Create TTCDRIVER_UTC in OpIOns / C / C ++ Compiler / Defined symbols in project.
Added in the TTCBlePeripheralTaskFxn () function
#ifdef TTCDRIVER_UTC
TTCSDKDriverDemoUTCEvent ();
#endif
Added in the TTCSDKDriverInit (void) function
#ifdef TTCDRIVER_UTC.
TTCDriverDemoUTCInit (& 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"

/*********************************************************************
 * Constants and macro definitions
 */
#define TTCBLE_SDK_UTC_EVN                              0x0008
/*********************************************************************
 * Local variable
 */
static TTCDriverInfo_t UTCErrCode;
TTCDriverUTCTime_t    sysUTC;
Clock_Struct UtcClock;
static ICall_Semaphore * UTCSem;                          //Thread semaphore，used to wake up
threads
static Queue_Handle * UTCMsgQueue;                          //Message handle
static u16 events;                                        //local event

/*********************************************************************
 * Local function declaratIOn
 */
static void TTCDriverUtcPrintf(TTCDriverUTCTime_t time);
static void TTCDriverDemoUTCClockHandler(UArg arg);
static void TTCSDKDriverUTCSetEvent(UArg arg);
```

```
/*****************************************************************************
【Function】   TTCDriverDemoUTCClockHandler(UArg arg)
【Overview】   Thread set event
【Parameter】   arg      : Marks the Event source event
【Return】   no
【Description】
*****************************************************************************/
static void TTCDriverDemoUTCClockHandler(UArg arg){
    TTCSDKDriverUTCSetEvent(arg);
}

/*****************************************************************************
【Function】   TTCSDKDriverUARTSetEvent(UArg arg)
【Overview】   Tag events and wake up threads
【Parameter】   arg : Tag events
【Return】   no
【Description】   no
*****************************************************************************/
static void TTCSDKDriverUTCSetEvent(UArg arg){
    events |= arg;
    Semaphore_post(*UTCSem);                                    //Wake up
thread
}

/*****************************************************************************
【Function】   TTCSDKDriverDemoUTCEvent(void)
【Overview】   Demo UTC event handling
【Parameter】   no
【Return】   no
【Description】   no
*****************************************************************************/
void TTCSDKDriverDemoUTCEvent(void){
    if(events & TTCBLE_SDK_UTC_EVN){
        events &= ~TTCBLE_SDK_UTC_EVN;
        TTCSDKDriver_UTC_Display();
    }
}

/*****************************************************************************
【Function】   TTCDriverDemoUTCInit(TTCSdkClass_t * appCb,
                                    ICall_Semaphore * sem,
```

```
                                        Queue_Handle * appMsgQueue)
【Overview】  TTCDriver driver initializatIOn
【Parameter】  no
【Return】  no
【Description】  no
*****************************************************************************/
void TTCDriverDemoUTCInit(TTCSdkClass_t * appCb,
                          ICall_Semaphore * sem,
                          Queue_Handle * appMsgQueue){
    if(sem == NULL || appMsgQueue == NULL || appCb == 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 time refresh cycle
    utcConfig.updateMsgFlag = true;                      //true false

    UTCErrCode = TTCDriverUTCInit(appCb,utcConfig);

    if(UTCErrCode != TTCDRIVER_INIT_SUCCESS){
        return;
    }
    // When did't open the message notice, we can define a cycle of events to take the initiative to obtain
UTC time
    if(utcConfig.updateMsgFlag == false){
        Util_constructClock(&UtcClock,
                            TTCDriverDemoUTCClockHandler,
                            1000,
                            0,
                            true,
                            TTCBLE_SDK_UTC_EVN);
```

```
    }
}

/*****************************************************************************
【Function】  TTCSDKDriver_UTC_Display(void)
【Overview】  TTCDriver driver initializatIOn
【Parameter】  no
【Return】  no
【Description】  no
*****************************************************************************/
void TTCSDKDriver_UTC_Display(void){
    TTCDriverUTCGetClock(&sysUTC);
    TTCDriverUtcPrintf(sysUTC);
    Util_startClock(&UtcClock);
}

/*****************************************************************************
【Function】  TTCDriverUtcProcess( TTCDriverUTCTime_t * time )
【Overview】  Handing UTC
【Parameter】  time : Receive UTCdata message structure
【Return】  no
【Description】  Note：Here the pointer parametertime is prohibited from releasing the memory
*****************************************************************************/
void TTCDriverUtcProcess( TTCDriverUTCTime_t * time ){
    memcpy(&sysUTC,time,sizeof(TTCDriverUTCTime_t));
    TTCDriverUtcPrintf(sysUTC);
}
/*****************************************************************************
【Function】  TTCDriverUtcPrintf( TTCDriverUTCTime_t  time )
【Overview】  print data
【Parameter】  time
【Return】  no
【Description】  no
*****************************************************************************/
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 INSTRUCTIONS

The IIC interface can be used to communicate with other devices that support the IIC protocol, such as ROM, LCD, and multiple sensors. The normal mode rate is 100KHz and the fast mode rate is 400KHz.

4.6.1 IIC API DESCRIPTION

4.6.1.1 TTCDriverI2CInitDefaultParam()

```
/*****************************************************************************
【Function】  TTCDriverI2CInitDefaultParam(I2C_Handle * I2CHandle)
【Overview】  Set the default parameter for the I2C handle
【Parameter】  I2CHandle : I2C HANDLE
【Return】  no
【Description】  no
*****************************************************************************/
extern void TTCDriverI2CInitDefaultParam(I2C_Handle * I2CHandle);
```

4.6.1.2 TTCDriverI2CInit()

```
/*****************************************************************************
【Function】  TTCDriverInfo_t TTCDriverI2CInit(TTCSdkI2CCB_t *appCallbacks,
                                               I2C_Handle * I2CHandle,
                                               TTCDriverI2CParams_t param)
【Overview】  Initialize I2C
【Parameter】  appCallbacks : Registration callback
               I2CHandle    : I2C HANDLE
               param        : I2Cparameter
【Return】  Please refer to TTCDriverUartInfo_t
【Description】  If the initialization fails, I2C will return the cause of the initialization failure.
               Note：If I2C is callback mode, you must set callback function, otherwise the initialization fails
```

```
                    If I2C is blocking mode, set callback function
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverI2CInit(TTCSdkI2CCB_t appCallbacks,
                                        I2C_Handle * I2CHandle,
                                        TTCDriverI2CParams_t param);

 callback function format:
typedef void (*TTCSdkI2CCB_t)(TTCDriverI2CState_t I2CState,u8 * buffer,u16 len,void * arg);
```

4.6.1.3 TTCDriverI2CRead

```
/*****************************************************************************
【Function】  TTCDriverI2CRead(I2C_Handle * I2CHandle,
                               u8 slaveAddr,
                               u8 * wBuf,
                               u8 * rBuf,
                               u16 wLen,
                               u16 rLen,
                               void * arg)
【Overview】  I2C read data
【Parameter】  I2CHandle : I2C HANDLE
               slaveAddr : I2C the peripheral address
               wBuf      : Write cache
               rBuf      : Read cache
               wLen      : Write cache length
               rLen      : Read cache length
               arg       : The parameter before transmission in callback mode is passed to the
callback function
【Return】  Please refer to TTCDriverInfo_t
【Description】  Note：1. In callback mode, Read cache can be set to NULL
                        In the blocking mode, Read cache can not be NULL
                    2.slaveAddr is the true address of the I2C peripheral device
*****************************************************************************/
extern TTCDriverInfo_t TTCDriverI2CRead(I2C_Handle * I2CHandle,
                                        u8 slaveAddr,
                                        u8 * wBuf,
                                        u8 * rBuf,
                                        u16 wLen,
                                        u16 rLen,
                                        void * arg);
```

4.6.1.4 TTCDriverI2CWrite()

```
/*******************************************************************************
【Function】 TTCDriverInfo_t TTCDriverI2CWrite(I2C_Handle * I2CHandle,
                                               u8 slaveAddr,
                                               u8 * wBuf,
                                               u16 wLen,
                                               void * arg)
【Overview】 I2C write data
【Parameter】 I2CHandle : I2C HANDLE
              slaveAddr : I2C the peripheral address
              wBuf      : Write cache
              wLen      : Write cache length
              arg       : The parameter before transmission in callback mode is passed to the
callback function
【Return】 Please refer to TTCDriverInfo_t
【Description】 Note：SlaveAddr is the true address of the I2C peripheral device
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverI2CWrite(I2C_Handle * I2CHandle,
                                         u8 slaveAddr,
                                         u8 * wBuf,
                                         u16 wLen,
                                         void * arg);
```

4.6.1.5 TTCDriverI2CBusy()

```
/*******************************************************************************
【Function】 TTCDriverI2CBusy(void)
【Overview】 Determine whether I2C transmission is busy
【Parameter】 no
【Return】 true : Busy state
           false : idle
【Description】 You need to call this function when reading and writing large amounts of data.
*******************************************************************************/
extern bool TTCDriverI2CBusy(void);
```

4.6.2 IIC USE EXAMPLES

```
/*******************************************************************************
【File】 TTCDriverI2CDemo.c
```

【Overview】 TTC SDK I2C Demo code
【Edit】 SDK WORKING GROUP
【Revise】 SDK WORKING GROUP
【Revised date】 2016/11/29
【Version】 V1.0.0
【Description】 This demo code is based on LIS3DH's I2C mode debugging

Function Description:

This file is based on the IIC driver of the 2640 SDK, configures the Gsensor LIS3DH and collects the original data,
The original data and then through the serial port printing, serial data format has been done, you can use anonymous four-axis host computer to view real-time waveform.
Adding Steps

1. Open the corresponding macro definition TTCDRIVER_I2C, need to use the serial port and open the macro TTCDRIVER_UART

2. the driver example initialization TTCSDKDriverInit () in the main thread initialization; increase IIC initialization configuration TTCDriverDemoI2Cnit (& sem, & appMsgQueue);

3. In the main thread processing function to increase the IIC internal event handling TTCDriverI2CEvent (); and DEMO IICdata deal with TTCSDKDriverDemoI2CEvent ();4.The pin is defined in TTCSDKBoard.h

```
*********************************************************************************/
#ifdef TTCDRIVER_I2C
/*********************************************************************
 * head files
 */
#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"

/*********************************************************************
 * Constants and macro definitions
 */
//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],
};

/*********************************************************************
 * Local variable
 */
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

/*********************************************************************
 * Local function declaratIOn
 */
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);

/*********************************************************************
 *    callback function
 */
LIS3DHiicCB TTCiicArgcb = {
   .param = 1,
   .iiccb = &SendRegiicCB,
};

/*********************************************************************************
【Function】    TTCDriverDemoI2CInit(ICall_Semaphore *sem,Queue_Handle *queueHandle)
【Overview】    I2C initialization and parameter setting
【Parameter】    sem            ：Semaphore
                 appMsgQueue ：Message handle
【Return】    no
【Description】    no
*********************************************************************************/
```

```
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 initialization

    Util_constructClock(&iicClock,
                        TTCDriverDemoI2CClockHandler,
                        1000,
                        40,
                        false,
                        SBP_IIC_EVT);       // Construct IIC Timing to Collect Gsensordata Events

    if(I2CParam.transferMode == I2C_MODE_BLOCKING){
        LIS3DH_Init(&I2CHandle);                    //Blocking mode
    }else{
        LIS3DH_Init_cb(&I2CHandle);                 //Callback mode
    }


}


/*********************************************************************************
【Function】   TTCSDKDriverDemoI2CEvent(void)
【Overview】   Read the Gsensordata event
【Parameter】   no
【Return】   no
【Description】   Current every 40MS sampling frequency
*********************************************************************************/
void TTCSDKDriverDemoI2CEvent(void) {
    if(events & SBP_IIC_EVT){
        events &= ~SBP_IIC_EVT;
        if(I2CParam.transferMode == I2C_MODE_CALLBACK){
            LIS3DH_GetAccAxesRaw_cb(&I2CHandle);            //Callback ways sampling
            Util_stopClock(&iicClock);
        } else{
            LIS3DH_GetAccAxesRaw(&I2CHandle,&Acc_data);     //Block ways sampling
```

```
                IICDatatoUart(&IIcRxbuffer[1], 6);                  //Serial port to print sample data
            }
        }
}

/*******************************************************************************
【Function】    LIS3DH_Init(void)
【Overview】    LIS3DH INITIALIZATION
【Parameter】    I2CHandle :    I2C handle
【Return】    no
【Description】    Blocking mode
*******************************************************************************/
static void LIS3DH_Init(I2C_Handle * I2CHandle){
    uint8 id_temp = 0;
    LIS3DH_ReadReg(I2CHandle, LIS3DH_WHO_AM_I, &id_temp);     //Read 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);
}

/*******************************************************************************
【Function】    LIS3DH_Init_cb(I2C_Handle * I2CHandle)
【Overview】    LIS3DH INITIALIZATION
【Parameter】    I2CHandle：IIC HANDLE
【Return】    no
【Description】    Callback mode
*******************************************************************************/
static void LIS3DH_Init_cb(I2C_Handle * I2CHandle){
    // Callback mode initialization, write the first value, call back to the data and then write the second ...
    LIS3DH_ReadReg_cb(I2CHandle, LIS3DH_WHO_AM_I);
}

/*******************************************************************************
【Function】    LIS3DH_ReadReg(I2C_Handle * I2CHandle, uint8 Reg, uint8 *Data)
【Overview】    Read register
【Parameter】    I2CHandle : I2C handle
```

```
                Reg         : Register
                Data        : data
【Return】    no
【Description】    Blocking mode
*****************************************************************************/
static uint8 LIS3DH_ReadReg(I2C_Handle * I2CHandle, uint8 Reg, uint8 *Data) {
    uint8 state;
    state = TTCDriverI2CRead(I2CHandle, SLAVEADDR, &Reg, Data,1,1,NULL);
    return state;
}
/*****************************************************************************
【Function】    LIS3DH_ReadReg_cb(I2C_Handle * I2CHandle, uint8 Reg, uint8 *Data)
【Overview】    Read register
【Parameter】    I2CHandle : I2C handle
                Reg         : Register
                Data        : data
【Return】    no
【Description】    Callback mode
*****************************************************************************/
static uint8 LIS3DH_ReadReg_cb(I2C_Handle * I2CHandle, uint8 Reg) {
    uint8 state;
    state = TTCDriverI2CRead(I2CHandle, SLAVEADDR,&Reg, NULL,1,1,&TTCiicArgcb);
    return state;
}

/*****************************************************************************
【Function】    LIS3DH_WriteReg_cb(I2C_Handle * I2CHandle,uint8 Reg, uint8 Data)
【Overview】    Write register
【Parameter】    I2CHandle : I2C handle
                Reg         : Register
                Data        : data
【Return】    no
【Description】    Callback mode
*****************************************************************************/
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;
}
```

```
/*****************************************************************************
【Function】  LIS3DH_WriteReg(I2C_Handle * I2CHandle,uint8 Reg, uint8 Data)
【Overview】  Write register
【Parameter】  I2CHandle : I2C handle
               Reg       : Register
               Data      : data
【Return】  no
【Description】  Blocking mode
*****************************************************************************/
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;
}
/*****************************************************************************
【Function】  LIS3DH_GetAccAxesRaw(I2C_Handle * I2CHandle, AxesRaw_t* buff)
【Overview】  Read Gsensordata
【Parameter】  I2CHandle : I2C handle
               buff      : Three - axis data
【Return】  no
【Description】  Blocking mode
*****************************************************************************/
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;
}

/*******************************************************************************
【Function】  LIS3DH_GetAccAxesRaw_cb(I2C_Handle * I2CHandle)
【Overview】  Read Gsensordata
【Parameter】  I2CHandle : I2C handle
【Return】  no
【Description】  Callback mode
*******************************************************************************/
uint8 LIS3DH_GetAccAxesRaw_cb(I2C_Handle * I2CHandle) {
    // Callback read Gsensordata, first write the first address, callback return value, and then write the
second address ...
    if(                    TTCDRIVER_TRANSFER_DATA_SUCCESS                    !=
LIS3DH_ReadReg_cb(I2CHandle,LIS3DH_OUT_X_L) )
        return MEMS_ERROR;
    return MEMS_SUCCESS;
}
```

```
/*****************************************************************************
【Function】   LIS3DHI2CCB(TTCDriverI2CState_t I2CState,u8 * buffer,u16 len,void * arg)
【Overview】   IIC read data callback
【Parameter】   I2CState : I2C transmission status
                buffer    : I2C read data
                arg        : carried parameters   before I2C transfers
【Return】   no
【Description】   no
*****************************************************************************/
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{

    }
}
/*****************************************************************************
【Function】   SendRegiicCB(u32 param,u8 * buffer,u16 len)
【Overview】   Callback mode --- IICdata returns callback entity function
【Parameter】   no
【Return】   no
【Description】   no
*****************************************************************************/
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;
        }
}
/*********************************************************************
【Function】    TTCDriverDemoI2CClockHandler(UArg arg)
```

```
【Overview】    Tag events and wake up threads
【Parameter】    arg : Tag events
【Return】    no
【Description】    no
*****************************************************************************/
static void TTCDriverDemoI2CClockHandler(UArg arg){
  events |= SBP_IIC_EVT;
  Semaphore_post(*iicsem);
}
/*****************************************************************************
【Function】    void IICDatatoUart(uint8 *Data,u16 len)
【Overview】    The serial port prints the data read by the IIC
【Parameter】    no
【Return】    no
【Description】
*****************************************************************************/
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 INSTRUCTIONS

SPI is a synchronous serial interface for CPU and all kinds of peripheral devices for full duplex. Synchronous serial communication has four lines, the clock line ：SCLK, the host input slave output data line： MISO, the host output slave input data line ：MOSI, chip select line ：CSN, frequency programmable and write conflict protection and so on. 2640 SPI driver to be the host support the maximum of 24M, to be the peripherial support the maximum of 4M.

4.7.1 SPI API instructions

4.7.1.1 TTCDriverSpiInitDefaultParam()

```
/*******************************************************************************
【Function】 TTCDriverSpiInitDefaultParam(Spi_Handle * spiHandle)
【Overview】 Set the SPI handle default parameter
【Parameter】 spiHandle : SPI handle
【Return】 no
【Description】 no
*******************************************************************************/
extern void TTCDriverSpiInitDefaultParam(Spi_Handle * spiHandle);
```

4.7.1.2 TTCDriverSpiInit()

```
/*******************************************************************************
【Function】 TTCDriverInfo_t TTCDriverSpiInit(TTCSdkSpiCB_t appCallbacks,
                                              Spi_Handle * spiHandle,
                                              TTCDriverSpiParams_t param)
【Overview】 initialize SPI
【Parameter】 appCallbacks : Registration callback
              spiHandle    : SPI handle
              param        : SPIparameter
【Return】 Please refer to TTCDriverInfo_t
【Description】 If the initialization fails, the SPI returns the cause of the initialization failure.
Note：If the SPI is Callback mode, you must set the callback function, otherwise the initialization fails
      If the SPI is Blocking mode, you do not need to set the callback function
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverSpiInit(TTCSdkSpiCB_t appCallbacks,
                                        Spi_Handle * spiHandle,
                                        TTCDriverSpiParams_t param);
```

callback function format:

typedef void (*TTCSdkSpiCB_t)(SPI_Status spiState,u8 * buffer,u16 len,void * arg);

4.7.1.3 TTCDriverSpiRead()

```
/*******************************************************************************
【Function】 TTCDriverSpiRead(Spi_Handle * spiHandle,
                              u8* rBuf,
                              u16 len,
                              void * arg)
【Overview】 SPI read data
【Parameter】 spiHandle : SPI handle
              rBuf      : Read data cache
              len       : Data length
              arg       : The parameter before transmission in callback mode is passed to the callback function
【Return】 Please refer to TTCDriverInfo_t
【Description】 Applicable to SPI host
              Note：Receive data under the callback in Callback Mode
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverSpiRead(Spi_Handle * spiHandle,
                                        u8* rBuf,
                                        size_t len,
                                        void * arg);
```

4.7.1.4 TTCDriverSpiWrite()

```
/*******************************************************************************
【Function】 TTCDriverSpiWrite(Spi_Handle spiHandle,
                               u8 *wBuf,
                               size_t len,
                               void * arg)
【Overview】 Write data to SPI TX cache and send it
【Parameter】 spiHandle : SPI handle
              wBuf      :  write data cache
              len       : Data length
              arg       : The parameter before transmission in callback mode is passed to the callback function
【Return】 Please refer to TTCDriverInfo_t
【Description】 Applicable to SPI Master and Slave modes
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverSpiWrite(Spi_Handle * spiHandle,
```

```
                                        u8 *wBuf,
                                        size_t len,
                                        void * arg);
```

4.7.1.5 TTCDriverSpiWriteRead()

```
/*******************************************************************************
【Function】 TTCDriverSpiWriteRead(Spi_Handle * spiHandle,
                                   uint8_t *buf,
                                   size_t len,
                                   void * arg)
【Overview】 SPI 写 read data
【Parameter】 spiHandle : SPI handle
              buf       : Read / Write data cache
              len       : Read / write data length
              arg       : The parameter before transmission in callback mode is passed to the
callback function
【Return】 Please refer to TTCDriverInfo_t
【Description】 Applicable to SPI Master mode
          Note：1. Buf cache length should not be less than len
                2.Receive data under the callback in Callback Mode
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverSpiWriteRead(Spi_Handle * spiHandle,
                                             uint8_t *buf,
                                             size_t len,
                                             void * arg);
```

4.7.1.6 TTCDriverSPIBusy()

```
/*******************************************************************************
【Function】 TTCDriverSPIBusy(void)
【Overview】 Determines whether SPI transmission is busy
【Parameter】 no
【Return】 true : Busy state
           false : idle
【Description】You need to call this function when reading and writing large amounts of data。
*******************************************************************************/
extern bool TTCDriverSPIBusy(void);
```

4.7.2 SPI USE EXAMPLES

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

【File】 TTCDriverSPIDemo.c
【Overview】 TTC SDK SPI DEMO CODE
【Edit】 SDK WORKING GROUP
【Revise】 SDK WORKING GROUP
【Revised date】 2016/12/28
【Version】 V1.0.5
【Description】

1. Debugging of SPI Mode Based on LIS3DH

TTCDriverSPIDemo.cFunction Description:

This file uses the 2640 SDK's SPI driver to configure the Gsensor LIS3DH and collect the raw data. the original data and then through the serial print, serial data format has been processed, you can use anonymous four-axis host computer to view real-time waveform.

Adding steps：

(1) to open the corresponding macro definition TTCDRIVER_SPI, need to use the serial port also need to open the macro TTCDRIVER_UART

(2) the driver example initialization TTCSDKDriverInit () in the main thread initialization; increase the SPI initialization configuration TTCDriverDemoSPIInit (& sem, & appMsgQueue);

(3) Increase the SPI internal event handling TTCDriverSpiEvent () in the main thread processing function TTCBlePeripheralTaskFxn (UArg a0, UArg a1); and TTCSDKDriverDemoSPIEvent () at DEMO SPIdata;

(4)The pin is defined in TTCSDKBoard.h

2. Debugging of SPI Mode Based on W25X Series

Using the SPI driver of the 2640 SDK,W25 series FLASH to open, close, read, write, erase and so on.

Adding steps：

(1) Open the corresponding macro definition TTCDRIVER_SPI.

(2) Initialize TTCSDKDriverInit () in the main thread initialization driver demo; Increase the SPI initialization configuration TTCDriverDemoSPIInit_FLASH (& sem, & appMsgQueue);

(3) Increase the SPI internal event handling in the main thread processing function TTCBlePeripheralTaskFxn (UArg a0, UArg a1) TTCDriverSpiEvent (); and DEMO SPIdata handle TTCSDKDriverDemoSPIEvent_FLASH ();

```
**********************************************************************************/
#ifdef TTCDRIVER_SPI
/*********************************************************************
 * head files
 */
#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 "comdef.h"
#include "TTCSDKBoard.h"
#include "TTCBleSDKConfig.h"
#include "TTCBleDevInfoService.h"
#include "TTCBlePeripheral.h"
#include "TTCBlePeripheralProcess.h"
#include "TTCBleProfile.h"
#include "TTCBleSDKManager.h"
#include "TTCDriverSPIFlash.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"

/*********************************************************************
* Constants and macro definitions
*/
//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_1

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 spiParam2 = {
      .spiName = CC2640_OAD,
      .spiMode = SPI_MASTER,
      .bitRate = 2000000,
      .dataSize = 8,
      .frameFormat = SPI_POL0_PHA0,                    // The SPI ways should be consistent with the communication ways
      .transferMode =     SPI_MODE_BLOCKING,     // SPI_MODE_CALLBACK, //
      .slaveCSPin = PIN_UNASSIGNED,
      .spiTransferBufLen = 0,
      .spiHWAttr = &spiCC26XXDMAHWAttrs[CC2640_OAD],      // The CC2640_SPI0 index number corresponds to the configured IO
};

/*********************************************************************
 * Local variable
 */
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
volatile 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)

/*********************************************************************
 * Local function declaratIOn
 */
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);

/*********************************************************************
 *   callback function
 */
LIS3DHspiCB TTCSpiArgcb = {
   .param = 1,
   .spicb = &SendRegCB,
};

TTCSdkSpiCB_t TTCBlePeripheralSPICB = {
   TTCSdkDriverSpiCB
};

/*****************************************************************************
【Function】   TTCDriverDemoI2CInit(ICall_Semaphore *sem,Queue_Handle *queueHandle)
【Overview】   SPI initialization and parameter setting
【Parameter】   sem          ：Semaphore
                appMsgQueue ：Message handle
【Return】   no
【Description】   no
*****************************************************************************/
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 = CC2640_OAD,
        .spiMode = SPI_MASTER,
        .bitRate = 4000000,
        .dataSize = 8,
        .frameFormat = SPI_POL0_PHA0,
        .transferMode =     SPI_MODE_BLOCKING,//SPI_MODE_CALLBACK,//
        .slaveCSPin = PIN_UNASSIGNED,
        .spiTransferBufLen = 0,
        .spiHWAttr = &spiCC26XXDMAHWAttrs[CC2640_OAD],
    };
    // SPI initialization, pay attention to determine whether the initialization is successful
    SPI_state = TTCDriverSpiInit(TTCBlePeripheralSPICB,&spiHandle,spiParam);
    Util_constructClock(&spiClock,
                        TTCDriverDemoSPIClockHandler,
                        1000,
                        40,
                        false,
                        SBP_SPI_EVT);                       // Build SPI Timing to Collect
Gsensordata Events

    if(spiParam.transferMode == SPI_MODE_CALLBACK){
        LIS3DH_Init_SPI_cb(&spiHandle);                 //Callback mode
    }else{
        LIS3DH_Init_SPI(&spiHandle);                    //Blocking mode
    }
}

/*********************************************************************************
【Function】    TTCSDKDriverDemoSPIEvent(void)
【Overview】    Read the Gsensordata event
【Parameter】    no
【Return】    no
【Description】
*********************************************************************************/
void TTCSDKDriverDemoSPIEvent(void){
    if(events & SBP_SPI_EVT){
```

```
            events &= ~SBP_SPI_EVT;
            if(spiParam2.transferMode == SPI_MODE_CALLBACK){
                LIS3DH_GetAccAxesRaw_SPI_cb();                    //Callback ways sampling
                Util_stopClock(&spiClock);
            }else{
                LIS3DH_GetAccAxesRaw_SPI(&spiHandle,(AxesRaw_t *)&Acc_data);      //Block ways
sampling
#ifdef TTCDRIVER_I2C
                asm("nop");
                Display_aixData((AxesRaw_t2 *)&Acc_data);
#endif
            }
        }
}

/*******************************************************************************
【Function】    SPIDatatoUart(uint8 *Data,u16 len)
【Overview】    The serial port prints the data read by the SPI
【Parameter】    Data                : the data need to print
                 len                 ：Data length
【Return】    no
【Description】
*******************************************************************************/
static void SPIDatatoUart(uint8 *Data,u16 len){
    if(len == 7){
        uint8 GSensorData[20]={0};

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

/*****************************************************************************
【Function】  LIS3DH_Init_SPI(Spi_Handle * spiHandle)
【Overview】  LIS3DH INITIALIZATION
【Parameter】  spiHandle        ：SPI HANDLE
【Return】  no
【Description】  Blocking method
*****************************************************************************/
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();
    //200hz LOWPOWER_MODE X_EN_Y_EN_Z_EN
    LIS3DH_WriteReg_SPI(spiHandle, LIS3DH_CTRL_REG1, 0x67);
    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();


    SPILis3dhSelect();
    //200hz LOWPOWER_MODE X_EN_Y_EN_Z_EN
    LIS3DH_ReadReg_SPI(spiHandle, LIS3DH_CTRL_REG1, &id_temp);
    SPILis3dhClose();

    SPILis3dhSelect();
    //200hz LOWPOWER_MODE X_EN_Y_EN_Z_EN
    LIS3DH_ReadReg_SPI(spiHandle, LIS3DH_CTRL_REG2, &id_temp);
```

```
    SPILis3dhClose();

    SPILis3dhSelect();
    //200hz LOWPOWER_MODE X_EN_Y_EN_Z_EN
    LIS3DH_ReadReg_SPI(spiHandle, LIS3DH_CTRL_REG4, &id_temp);
    SPILis3dhClose();

    Util_startClock(&spiClock);

}

/*********************************************************************
 【Function】  LIS3DH_ReadReg(uint8 Reg, uint8* Data)
 【Overview】  Read register
 【Parameter】  spiHandle        ：SPI HANDLE
                Reg              : Register
                Data             ：data
 【Return】  no
 【Description】  Blocking method
*********************************************************************/
static uint8 LIS3DH_ReadReg_SPI(Spi_Handle * spiHandle, uint8 Reg, uint8 *Data) {
    uint8 state;
    uint8 txBuf[2];
    txBuf[0] = Reg|0X80;
    txBuf[1] = 0;
    state = TTCDriverSpiWriteRead(spiHandle,txBuf,2, NULL);    //Write address, return valid value
    // TxBuf [0] returns 0xFF, valid values are stored in txBuf [1]
    *Data = txBuf[1];
    return state;
}

/*********************************************************************
 【Function】  LIS3DH_WriteReg_SPI(Spi_Handle * spiHandle,uint8 Reg, uint8 Data)
 【Overview】  Write register
 【Parameter】  no
 【Return】  no
 【Description】  Blocking method
*********************************************************************/
static uint8 LIS3DH_WriteReg_SPI(Spi_Handle * spiHandle,uint8 Reg, uint8 Data){
    uint8 state;
    uint8 txBuf[2];
    txBuf[0] = Reg;
```

```
    txBuf[1] = Data;
    //manipulat the register address and the value into the cache and send out
    state = TTCDriverSpiWrite(spiHandle,txBuf,2, NULL);
    return state;
}

/*******************************************************************************
【Function】  LIS3DH_GetAccAxesRaw(AxesRaw_t* buff)
【Overview】  Read Gsensordata
【Parameter】  spiHandle        ：SPI HANDLE
               buff             ：Three - axis data
【Return】  no
【Description】  Blocking method
*******************************************************************************/
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, 7, 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();

    SPIDatatoUart(spi_rx_buf,7);
}

/*******************************************************************************
【Function】  TTCSdkDriverSpiCB(SPI_Status spiState,u8 * buffer,u16 len,void * arg)
【Overview】  SPI data returns a callback
【Parameter】  spiState         ：SPI state
               buffer           ：read the data
               len              ：Data length
               arg              ：user callback function
【Return】  no
【Description】  Callback ways
*******************************************************************************/
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);
    }
}

/*****************************************************************************
【Function】  LIS3DH_Init_SPI_cb(Spi_Handle * spiHandle)
【Overview】  LIS3DH INITIALIZATION
【Parameter】  spiHandle        ：SPI HANDLE
【Return】  no
【Description】  Callback ways
*****************************************************************************/
static void LIS3DH_Init_SPI_cb(Spi_Handle * spiHandle){
    SPILis3dhSelect();
    // First write the first value, call back and then write the second ...
    LIS3DH_ReadReg_SPI_cb(spiHandle, LIS3DH_WHO_AM_I);
}

/*****************************************************************************
【Function】  LIS3DH_ReadReg_SPI_cb(Spi_Handle * spiHandle, uint8 Reg)
【Overview】  Read register
【Parameter】  spiHandle        ：SPI HANDLE
              Reg               : Register
【Return】  no
【Description】  Callback ways
*****************************************************************************/
uint8 LIS3DH_ReadReg_SPI_cb(Spi_Handle * spiHandle, uint8 Reg) {
    uint8 state;
    uint8 txBuf[2];
    txBuf[0] = Reg|0X80;
    // Write address, return valid value in callback function
    state = TTCDriverSpiWriteRead(spiHandle,txBuf, 2,  &TTCSpiArgcb);
    asm("nop");
    return state;
}

/*****************************************************************************
【Function】  uint8 LIS3DH_WriteReg_SPI_cb(Spi_Handle * spiHandle,uint8 Reg，uint8 Data)
【Overview】  Write register
【Parameter】  spiHandle        ：SPI HANDLE
              Reg               : Register
              Data              ：data
```

```
【Return】    no
【Description】    Callback ways
*****************************************************************************/
uint8 LIS3DH_WriteReg_SPI_cb(Spi_Handle * spiHandle, uint8 Reg ,uint8 Data){
    uint8 state;
    uint8 txBuf[2];
    txBuf[0] = Reg;
    txBuf[1] = Data;
    // When write register does not require a return value
    state = TTCDriverSpiWrite(spiHandle,txBuf,2, &TTCSpiArgcb);
    //manipulat the register address and the value into the cache and send out

    return state;
}

/*****************************************************************************
【Function】    LIS3DH_GetAccAxesRaw_SPI_EVT(void)
【Overview】    Read Gsensordata
【Parameter】    no
【Return】    no
【Description】    Callback ways
*****************************************************************************/
void LIS3DH_GetAccAxesRaw_SPI_cb(void){
    uint8 spi_rx_buf[7] = {0};
    SPILis3dhSelect();
    spi_rx_buf[0] = 0xC0|LIS3DH_OUT_X_L;                    //0xC0 is a continuous read instruction
    TTCDriverSpiWriteRead(&spiHandle,spi_rx_buf, 7, &TTCSpiArgcb);
}

/*****************************************************************************
【Function】    SendRegCB(u32 param,u8 * buffer,u16 len)
【Overview】    SPI data Returns the callback entity function
【Parameter】    param          : Operation number
                 buffer         ：Read the data
                 len            ：Data length
【Return】    no
【Description】
*****************************************************************************/
static void SendRegCB(u32 param,u8 * buffer,u16 len){
    switch(param){
        case 1:                                                  //who am i
            if(buffer[1] == 0x33){
```

```
                TTCSpiArgcb.param++;
                SPILis3dhSelect();
                //200hz LOWPOWER_MODE X_EN_Y_EN_Z_EN
                LIS3DH_WriteReg_SPI_cb(&spiHandle, LIS3DH_CTRL_REG1, 0x67);
            }
        break;
        case 2:
            TTCSpiArgcb.param++;
            SPILis3dhSelect();
              //click fliter enable
            LIS3DH_WriteReg_SPI_cb(&spiHandle, LIS3DH_CTRL_REG2, 0x04);
        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;
    }
}

/*********************************************************************************
【Function】   TTCDriverDemoI2CClockHandler(UArg arg)
【Overview】   Tag events and wake up threads
【Parameter】   arg : Tag events
【Return】   no
【Description】   no
*********************************************************************************/
static void TTCDriverDemoSPIClockHandler(UArg arg){
    events |= SBP_SPI_EVT;
    Semaphore_post(*spisem);
}

//-------------------------------------------------------------------------------------------//SPI  BLOCKING  FOR  FALSH
TEST
//SPI Blocking modeFLASH test
//-------------------------------------------------------------------------------------------
```

```
//#ifdef TTCDRIVER_SPI_FLASH
/*****************************************************************************
【Function】   TTCDriverDemoSPIInit_Flash(ICall_Semaphore *sem,Queue_Handle *queueHandle)
【Overview】   SPI initialization and parameter setting
【Parameter】   sem            ：Semaphore
                appMsgQueue ：Message handle
【Return】   no
【Description】   no
*****************************************************************************/
typedef struct{
  uint8   test1_0;
  uint8   test1_1;
  uint8   test1_2;
  uint8   test1_3;
  uint8   test1_4;
  uint8   test1_5;
  uint8   test1_6;
  uint8   test1_7;
  uint8   test1_8;
  uint8   test1_9;
  uint32 test2_1;
  uint32 test2_2;
  uint32 test2_3;
  uint32 test2_4;
  uint32 test2_5;
  uint16 test3_0;
  uint16 test3_1;
  uint16 test3_2;
  uint16 test3_3;
  uint16 test3_4;
  uint16 test3_5;
  uint16 test3_6;
  uint16 test3_7;
  uint16 test3_8;
  uint16 test3_9;
}test_t;
test_t test_write;
uint8 *FlashtxBuf = NULL;
uint8 writeFlash_state;
uint8 openspiFlash_state;
uint8 erasespiFlash_state;
uint8 Test_Data;
```

```
#define tx_len 1024
void TTCDriverDemoSPIInit_FLASH(ICall_Semaphore *sem,Queue_Handle *queueHandle){
        spisem                                          = sem;

        FlashtxBuf = ICall_malloc(tx_len);
        if(FlashtxBuf != NULL)
        {
           asm("nop");
        }
        asm("nop");


        memset(FlashtxBuf,0x11,tx_len);
        openspiFlash_state = spiFlashOpen(sem,queueHandle );
        if(openspiFlash_state == TRUE)
        {
           asm("nop");
        }
           asm("nop");
        erasespiFlash_state = spiFlashErase(0x00000,tx_len);
        if(erasespiFlash_state == TRUE)
        {
           asm("nop");
        }
           asm("nop");
        writeFlash_state = spiFlashWrite(0 ,tx_len,(uint8_t *)FlashtxBuf);
        if(writeFlash_state == TRUE)
        {
           asm("nop");
        }
        asm("nop");
        writeFlash_state = spiFlashRead(0 ,200,(uint8_t *)FlashtxBuf);
        if(writeFlash_state == TRUE)
        {
           asm("nop");
        }
        spiFlashClose();
        asm("nop");
        openspiFlash_state = spiFlashOpen(sem,queueHandle );
        if(openspiFlash_state == TRUE)
        {
           asm("nop");
```

```
        }
          asm("nop");
        for(uint8 i=0;i<30;i++)
        {
          writeFlash_state = spiFlashRead(0 ,sizeof(test_write),(uint8_t *)FlashtxBuf);
          if(writeFlash_state == TRUE)
          {
            asm("nop");
          }
          asm("nop");
        }
        spiFlashClose();
        asm("nop");
    Util_constructClock(&spiClock,
                        TTCDriverDemoSPIClockHandler,
                        1000,
                        1000,
                        TRUE,
                        SBP_SPI_EVT);                    // Build SPI Timing Read FLASHdata event
}

/*********************************************************************************
【Function】   TTCSDKDriverDemoSPIEvent_FLASH(void)
【Overview】   Read the FLASHdata event
【Parameter】   no
【Return】   no
【Description】
*********************************************************************************/
void TTCSDKDriverDemoSPIEvent_FLASH(ICall_Semaphore *sem,Queue_Handle *queueHandle){
    if(events & SBP_SPI_EVT){
        events &= ~SBP_SPI_EVT;
        openspiFlash_state = spiFlashOpen(sem,queueHandle );
        if(openspiFlash_state == TRUE)
        {
          asm("nop");
        }
          asm("nop");
        for(uint8 i=0;i<3;i++)
        {
          memset(FlashtxBuf,i,tx_len);
          erasespiFlash_state = spiFlashErase(0x00000,tx_len);
          if(erasespiFlash_state == TRUE)
```

```
            {
               asm("nop");
            }
               asm("nop");

            writeFlash_state = spiFlashWrite(0 ,tx_len,(uint8_t *)FlashtxBuf);
            if(writeFlash_state == TRUE)
            {
               asm("nop");
            }
            asm("nop");
//            writeFlash_state = spiFlashRead(0 ,sizeof(test_write),(uint8_t *)FlashtxBuf);
            writeFlash_state = spiFlashRead(0 ,tx_len,(uint8_t *)FlashtxBuf);
            if(writeFlash_state == TRUE)
            {
               asm("nop");
            }
            asm("nop");
         }
         spiFlashClose();
         asm("nop");
      }
}

//#endif //TTCDRIVER_SPI_FLASH
#endif //TTCDRIVER_SPI
```

4.8.Wechat Description

WeChat drive contains WeChat AirSync Bluetooth communication protocol; automatic processing by WeChat found, and WeChat handshake connection, data packet, etc., the user only need to be related to initialization, and call the corresponding API to receive, send data.

1. Broadcast format requirements

1)the Wechat service UUID: 0xFEE7 under GAP_ADTYPE_16BIT_MORE does not allow to change.

2)The last 2 bytes of the vendor ID need to be reserved for the company ID and the 6-byte MAC address of the device. See DEMO.

2. WeChat Bluetooth initialization

```
 TTCBleWechatParam_t wechatParam= {
.ramSize=50,// Send cache size (no less than 30)
.authSelect=AUTH_MAC_NOENCRYPT, //Login mode selection
.dataDirSelect=DATA_DIR_BG,// Data direction
.deviceMac={0x24,0x71,0x89,0xff,0x00,0x02}, //Device Mac Adress
};

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

3. Add Wechat Bluetooth processing

In the static void TTCBlePeripheralTaskFxn (UArg a0, UArg a1) to add WeChat event processing, in the ICall_Errno errno = ICall_wait (ICALL_TIMEOUT_FOREVER) add

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

in static void TTCBlePeripheralTaskProcessAppMsg (TTCMsg_t * pMsg) add code

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

Add function：

Note：The following uses the UART function, please refer to the TTCDriverUART.h driver file to add the UART driver.

4.8.1 Wechat API 说明

4.8.1.1 TTCBleWechatEvent()

```
/*******************************************************************************
【Function】  TTCBleWechatEvent(void
【Overview】  Wechat event handling
【Parameter】  no
【Return】  no
【Description】  no
*******************************************************************************/
extern void TTCBleWechatEvent(void);
```

4.8.1.2 TTCBleWechatInit()

```
/*******************************************************************************
【Function】  TTCBleWechatInit(TTCSdkClass_t *appCallbacks,
                               ICall_Semaphore *sem,
                               Queue_Handle *queueHandle,
                               ICall_EntityID *selfEntity,
                               TTCBleWechatParam_t param)
【Overview】  Initialize Wechat
【Parameter】  appCallbacks :  callback function
               sem             : Semaphore
               queueHandle     : Message handle
               selfEntity      : ID
               param           : parameter
【Return】  no
【Description】  no
*******************************************************************************/
extern TTCDriverInfo_t TTCBleWechatInit(TTCSdkClass_t *appCallbacks,
                                        ICall_Semaphore *sem,
                                        Queue_Handle *queueHandle,
                                        ICall_EntityID *selfEntity,
                                        TTCBleWechatParam_t param);
```

4.8.1.3 TTCBleWechatSend()

```
/*******************************************************************************
【Function】  TTCBleWechatSend(u8 *sendData,u16 len)
【Overview】  Send data to Wechat
【Parameter】  sendData : Send the data
               len : Data length
```

【Return】 失败原因 Please refer to TTCDriverInfo_t

【Description】 no

**********/

extern TTCDriverInfo_t TTCBleWechatSend(u8 *sendData,u16 len);

4.8.2 Wechat uses examples

/**********

【File】 TTCDriverWechatDemo.c

【Overview】 TTC SDK Wechat demo code

【Edit】 SDK WORKING GROUP

【Revise】 SDK WORKING GROUP

【Revised date】 2016/12/02

【Version】 V1.0.0

【Description】

Function Description:

When initialization broadcast format in accordance with the WeChat protocol to load, to ensure that can be found by WeChat public account; the corresponding API interface to the WeChat public account to send data and receive data from the WeChat public account. When the user writes data the Firmware request format packet, sent data to the WeChat automatically unpacked as the original data, to achieve data transmission. WeChat write data to the device, the underlying equipment to unpack the data passed to the user layer API, the user receives data from the cache。

Adding Steps

1. Open the corresponding macro definition TTCBLE_WECHAT, you need to use the serial port and open the macro TTCDRIVER_UART

2. In the main thread initialization driver example initialization TTCSDKDriverInit (); increase the WeChat initialization configuration TTCDriverDemoWechatInit (& TTCBlePeripheralTaskCls, & sem, & appMsgQueue, & selfEntity);

3. In the main thread processing function to increase the WeChat internal event processing TTCBleWechatEvent (); and DEMO micro-data processing TTCSDKDriverDemoWechatEvent ();

4. Add the WeChat state handler TTCBleWechatStateClear (); and TTCDriverDemoWechatReadToSendStateSet (false) to the TTCSDK_MSG_GET_BLE_STATE_EVENT message in the Thread message handling functionTTCBlePeripheralTaskProcessAppMsg (TTCMsg_t * pMsg)

5. Add the processing micro-message function TTCBleWechatProcess (TTCMsg_t * TTCMsg) in the TTCSDK_MSG_BLE_WECHAT_EVENT message of the Thread message handling functionTTCBlePeripheralTaskProcessAppMsg (TTCMsg_t * pMsg)

**********/

#ifdef TTCBLE_WECHAT

/**********

* head files

*/

```
#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"

/*********************************************************************
 * Constants and macro definitions
 */
#define SBP_WECHAT_SEND_EVT                  0x0001
#define SBP_WECHAT_RECEIVED_EVT              0x0002
```

```
#define ADVERT_MAC_ADDR                         23
// The start of the MAC address in the broadcast packet,
The actual application should match the MAC address font in the broadcast packet
#define WECHAT_SERV_UUID                        0xFEE7              //Wechat UUID
#define DEFAULT_DISCOVERABLE_MODE_wechat                    GAP_ADTYPE_FLAGS_GENERAL

/*********************************************************************
 * Local variable
 */
#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[] = {        //Broadcast data，Maximum no more than 31 bytes
 //Note：IOS custom vendor identifier data (GAP_ADTYPE_MANUFACTURER_SPECIFIC) no more than
18 bytes
      0x02,                                                 //Data length
      GAP_ADTYPE_FLAGS,                                       // Broadcast
type identifier
      DEFAULT_DISCOVERABLE_MODE_wechat                                    |
GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED, // Broadcast type

      0x03,                                                 //Data length
      GAP_ADTYPE_16BIT_MORE,                                  //16Bit 服务
UUID
      LO_UINT16(WECHAT_SERV_UUID),                              // Service
type
      HI_UINT16(WECHAT_SERV_UUID),
```

```
    0x15,                                                   // Change according to
actual usage length
    GAP_ADTYPE_MANUFACTURER_SPECIFIC, // Use the WeChat function where the MAC
address should be added (can be changed according to actual needs)
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x0d,0x0a,// Here you can fill in the relevant data according to actual
needs
    0x44 ,0xA6,0xE5,0x07,0x99,0x7A      // Here must fill in Device Mac Address (the following test
using the MAC address)

};

/*********************************************************************
 * Local function declaratIOn
 */
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);

/*********************************************************************************
【Function】   TTCDriverDemoWechatInit(TTCSdkClass_t *appCallbacks,
                                       ICall_Semaphore *sem,
                                       Queue_Handle *queueHandle,
                                       ICall_EntityID *selfEntity)
【Overview】   WeChat initialization and parameter settings
【Parameter】   sem          ：Semaphore
                appMsgQueue ：Message handle
                selfEntity   ：Task ID
【Return】   no
【Description】   no
*********************************************************************************/
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,                                    // Send cache size (no less than 30)
      .authSelect = AUTH_MAC_NOENCRYPT,                       //Login mode selection
      .dataDirSelect = DATA_DIR_BG,                     //data direction
      .deviceMac = {                                    //Device Mac Adress
                        Wechat.ownAddress[5],
                        Wechat.ownAddress[4],
                        Wechat.ownAddress[3],
                        Wechat.ownAddress[2],
                        Wechat.ownAddress[1],
                        Wechat.ownAddress[0]
                    },
    };
    TTCBleWechatInit(appCallbacks,                      // WeChat initialization
                     sem,
                     queueHandle,
                     selfEntity,
                     wechatParam);
}

/*******************************************************************************
【Function】   TTCSDKDriverDemoWechatEvent(void)
【Overview】   Demo Wechat event handling
【Parameter】   no
【Return】   no
【Description】   no
*******************************************************************************/
void TTCSDKDriverDemoWechatEvent(void){
    if(events & SBP_WECHAT_SEND_EVT){
        events &= ~SBP_WECHAT_SEND_EVT;
        if(ReadToSendStateGet()){
            WechatDataSend("X",1);          // Send the data test regularly
        }
    }
```

```
    if(events & SBP_WECHAT_RECEIVED_EVT){
        events &= ~SBP_WECHAT_RECEIVED_EVT;
        if(ReceivedDataStateGet()){
            // RX cache has received data
            ReceivedDataStateSet(false);
            Wechat.WechatState.Data_received_flag = 0;
#ifdef TTCDRIVER_UART
            TTCDriverUartWrite(&uartHandle,Wechat.Wechat_RX,Wechat.Wechat_rx_len);      //print
#endif          //TTCDRIVER_UART
        }
    }
}


/*****************************************************************************
【Function】  TTCBleWechatProcess(TTCMsg_t * TTCMsg)
【Overview】  process Wechat data
【Parameter】  TTCMsg : Receive the Wechat data message structure
【Return】  no
【Description】  no
*****************************************************************************/
void TTCBleWechatProcess(TTCMsg_t * TTCMsg){
    TTCData_t * TTCData = NULL;
    TTCData = TTCMsg->pValue;

    WechatReportStatus wechatStatus = (WechatReportStatus)TTCData->param;
    switch(wechatStatus){
        case WECHAT_REPORT_STATUS_AUTH:{                        // Completed login
            #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:{          // Received vendor server
response data
#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:{            // Has been initialized, the initialization is
complete before sending data
            TTCDriverDemoWechatReadToSendStateSet(true);      // Set the data status ready to
send
            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:{            // Receive the data sent by the
vendor server
            TTCBleWechatSend(TTCData->pValue,TTCData->len);
            WechatDataReceived(TTCData->pValue,TTCData->len);    // Put the data into the receive
cache
            ReceivedDataStateSet(true);                          // Set the data receive status
            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:{                                    //
Interface switch
            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:{                    // Background switch
            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);
}


/*******************************************************************************
 【Function】   WechatDataSend(u8 *sendData,u16 len)
 【Overview】   send data to Wechat
 【Parameter】   sendData ：send the data
                 len       ：send the Data length
 【Return】   no
 【Description】   no
*******************************************************************************/
static TTCDriverInfo_t WechatDataSend(u8 *sendData,u16 len){
    TTCDriverInfo_t state;
```

```
    state = TTCBleWechatSend(sendData,len);
    return state;
}

/*******************************************************************************
【Function】   WechatDataReceived(u8 *sendData,u16 len)
【Overview】   receive Wechat data
【Parameter】   receivedData ： received Wechat data
                len : received Wechat Data length
【Return】   no
【Description】   no
*******************************************************************************/
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);
}


/*******************************************************************************
【Function】   UartRxToWechat(u8 *brfferData,u16 len)
【Overview】   Read data from received cache
【Parameter】   brfferData ： Serial cache data
                len          ： Cache Data length
【Return】   no
【Description】   no
*******************************************************************************/
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;
}

/*******************************************************************************
【Function】   TTCDriverDemoWechatClockHandler(UArg arg)
【Overview】   Timing task callback function
【Parameter】   arg : Tag events
【Return】   no
【Description】   no
*******************************************************************************/
static void TTCDriverDemoWechatClockHandler(UArg arg){
    events |= arg;
```

```
    Semaphore_post(*Wechatsem);
}

/*******************************************************************************
【Function】  TTCDriverDemoWechatReadToSendStateSet(bool flag)
【Overview】  Set the send Wechat enable
【Parameter】  flag：1:allow，0:not ready
【Return】  no
【Description】  no
*******************************************************************************/
void TTCDriverDemoWechatReadToSendStateSet(bool flag){
    Wechat.WechatState.init_ok_flag = flag;
}

/*******************************************************************************
【Function】  ReadToSendStateGet(bool flag)
【Overview】  Read Send Wechat data enable flag
【Parameter】  flag：1 allowed, 0 not ready
【Return】  Returns whether Wechat is ready to send data status
【Description】  no
*******************************************************************************/
static bool ReadToSendStateGet(void){
    return Wechat.WechatState.init_ok_flag;
}

/*******************************************************************************
【Function】  ReceivedDataStateSet(bool flag)
【Overview】  Receive Wechat data enable settings
【Parameter】  flag：1 has data, 0 no data
【Return】  no
【Description】  no
*******************************************************************************/
static void ReceivedDataStateSet(bool flag){
    Wechat.WechatState.Data_received_flag = flag;
}

/*******************************************************************************
【Function】  ReceivedDataStateGet(void)
【Overview】  Get Receive Wechat data enable status
【Parameter】  no
【Return】  Returns whether received the Wechat data status
【Description】  no
```

```
*****************************************************************************/
static bool ReceivedDataStateGet(void){
    return Wechat.WechatState.Data_received_flag;
}

/*****************************************************************************
【Function】  void WechatGetMacAddrUpdatetoAdvertData(void)
【Overview】  Update the MAC address to Broadcast data
【Parameter】  no
【Return】  no
【Description】  no
*****************************************************************************/
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
```

4.9 Watchdog instructions

The watchdog is used to prevent the program from entering "dead circulation", and the MCU is reset to a certain extent to ensure the normal operation of the device. Ultra-wide overflow time setting, ranging from 1ms to 2863311 ms, and can be re-adjusted at any time. Note that the watchdog's overflow timer is suspended when the MCU is in sleep mode.

4.9.1 Watchdog API Description

Watchdog related API has two, as follows：

4.9.1.1 TTCDriverWatchDogInit()

```
/*******************************************************************************
【Function】  TTCDriverWatchDogInit(u32 milliseconds)
【Overview】  WatchDog initialization
【Parameter】  milliseconds: Watchdog timeout time, unit：  ms
【Return】  no
【Description】  The maximum can not exceed 2863311 ms
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverWatchDogInit(u32 milliseconds);
```

4.9.1.2 TTCDriverWatchDogSetReload()

```
/*******************************************************************************
【Function】  TTCDriverWatchDogSetReload(u32 milliseconds)
【Overview】  WatchDog reloads
【Parameter】  milliseconds: Watchdog timeout time, unit：  ms
【Return】  no
【Description】  The maximum can not exceed 2863311 ms
*******************************************************************************/
extern TTCDriverInfo_t TTCDriverWatchDogSetReload(u32 milliseconds);
```

4.9.2 Watchdog instructions

WatchDog function instructions

1. When the program is initialized, the watchdog timer starts timing: TTCDriverWatchDogInit (1000); // Set the watchdog timeout to 1 second

2. Do not overload the watchdog, if the system is running task: watchdog overflow, program reset. If the system is sleep: the watchdog stops counting, the program will not reset until the system wakes up. watchdog overflow, the program will be reset.

3. Timing on the watchdog overload: TTCDriverWatchDogSetReload (1000); the system is running normally, the program will not reset, when the program from entering“dead circulation” and causes the event can not run, the program will be reset.

Recommendation: Watchdog reload can use timed events, or you can calculate the code run time into the program.

4.10 Beacon Application Description

Please refer to the documentation "TTC_Beacon specification V1.4.pdf". Note: The parameter settings in the Beacon application support power-down storage. Use the power-down storage function, need to select the appropriate project configuration.

5.TTC SDK OAD

5.1.OAD INTRODUCTION

OAD, Over-the-Air Download.

This chapter mainly introduces the OAD usage method in the TTC SDK. OAD is divided into two types: one is the external OAD (Off-Chip OAD) and the other is the internal OAD (On-Chip OAD). Off-Chip OAD and On-Chip OAD differences in whose hardware has an extra flash. Using an external flash, we call it Off-Chip OAD, and if we do not use external flash, we call it On-Chip OAD.

In software, Off-Chip OAD principle is the BLE first stored the data in the external flash, and then compared to the BLE issued checksum and flash the actual data calculation of the checksum, if the match is successful, set The corresponding start metadata, and then restart the system, load the external flash image to the internal flash, loading is completed, the system will set the corresponding metadata and confirm the firmware has been updated the latest Version. The system will then jump to the specified mirror address to start the program.

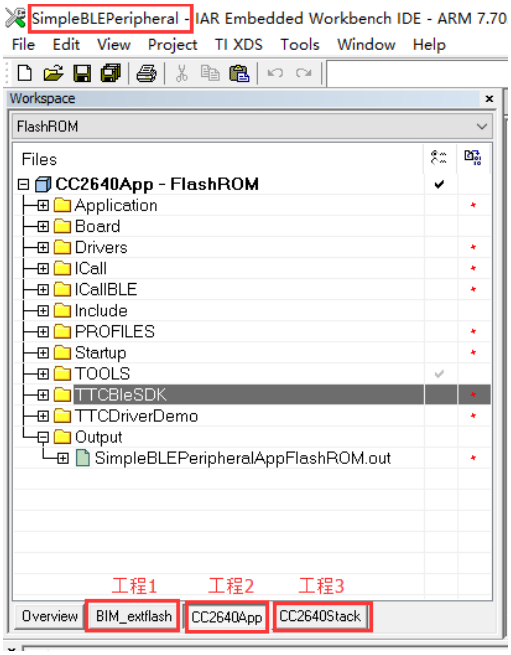
On-Chip OAD principle is BLE send OAD upgrade command to wakeup module OAD upgrade service program, the system receives the BLE sent OAD upgrade command, set the metadata, reset the system, and then jump to the OAD upgrade service program, upgrade the service program and BLE connection , BLE can begin to send the image file, OAD upgrade service program will be directly written to the mirror image of the original space, after writing, compared to the BLE issued by the checksum and write the internal flash checksum, compared success, update the metadata and jump to the new image file and start executing the new program, so that the On-Chip OAD is complete.

5.2 OADproject structure introduction

5.2.1 Three projects have different roles

Open SimpleBLEPeripheralproject, a total of three projects, as shown below. Project path C: \ TTC_BLE_CC2640_SDK \ 1.0.7 \TTC_CC2640_SDK\Projects\ble\SimpleBLEPeripheral \CC26xx\IAR

- project1 (BIM_extflash - Off-chipOAD startup code)
- project2 (CC2640App - application)
- project3 (CC2640Stack - protocol stack)



according to whether to use OAD, there have three cases when development, each case on the use of the above three different projects, as follows:

(1) do not use OAD function, need to use project2 and project3, without the use of project1;

(2) use On-chipOAD function, only need to use project2.

(3) use Off-chipOAD function, the above three projects need to modify the configuration;

5.2.2 Each project has different configuration

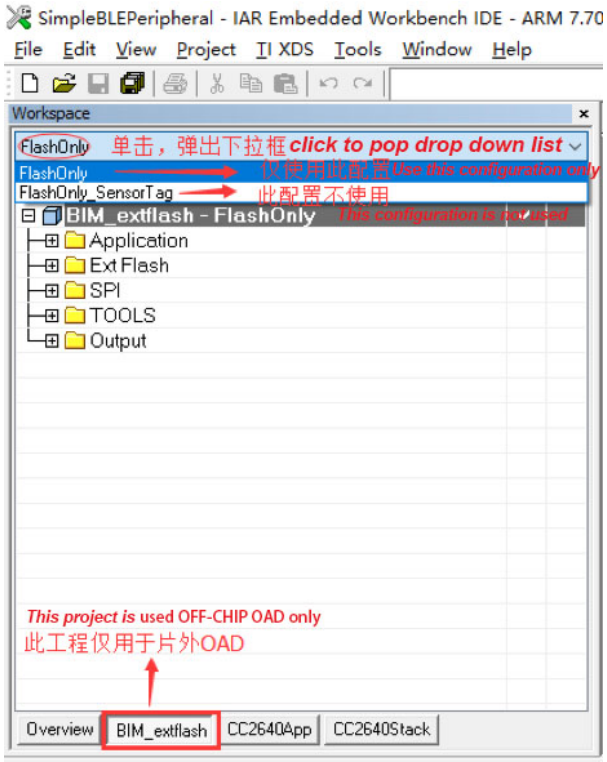
According to the development project needs OAD function, whether to need for power-down storage function, whether to need Bluetooth binding function, select a different project, select the configuration of each project. Specific configuration method as below table：

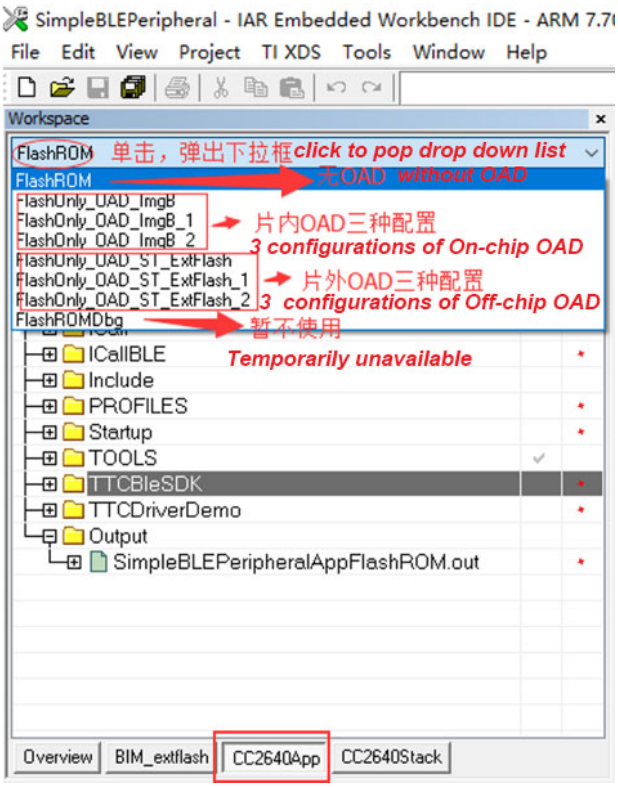
OAD	Function		project1 (BIM_extflash)	project2 (CC2640App)	project3 (CC2640Stack)
	Power down storage	Bluetooth binding			
noOAD	√	√	\	FlashROM	FlashROM
	√	×	\	FlashROM	FlashROM_1
	×	×	\	FlashROM	FlashROM_2
YesOn-chip OAD	√	√	\	FlashOnly_OAD_ImgB	\
	√	×	\	FlashOnly_OAD_ImgB_1	\
	×	×	\	FlashOnly_OAD_ImgB_2	\
YesOff-chipOAD	√	√	FlashOnly	FlashOnly_OAD_ST_ExtFlash	FlashROM
	√	×	FlashOnly	FlashOnly_OAD_ST_ExtFlash_1	FlashROM_1
	×	×	FlashOnly	FlashOnly_OAD_ST_ExtFlash	FlashROM_2

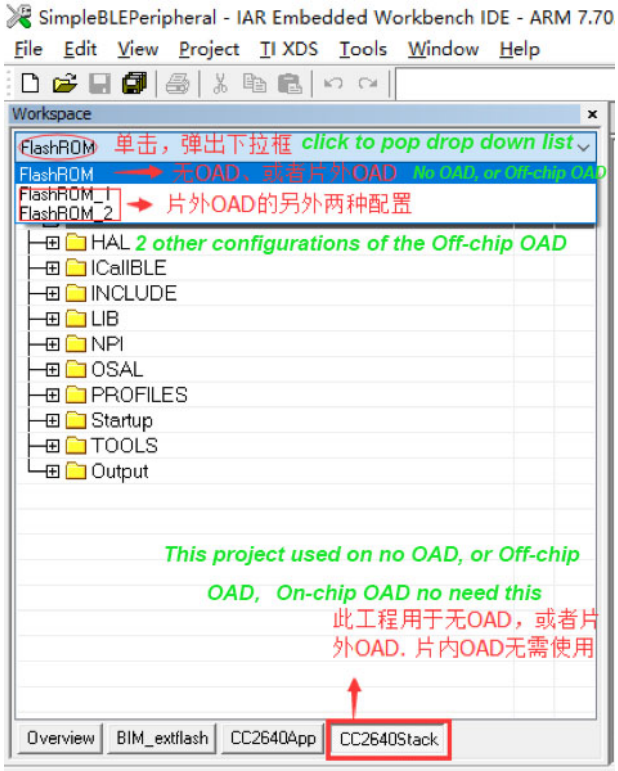
				_2	

注：：“√” required，“×”not required，“\”no need to configure this project.

Specific 3 different project configuration, can be set as shown below：

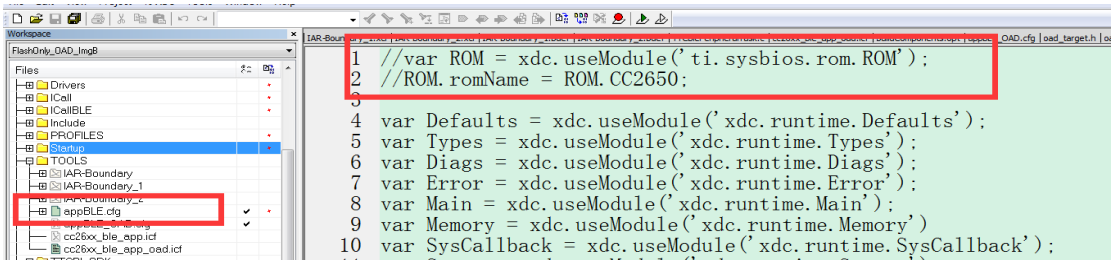






5.2.3 Particular Attention

Special Note: Off-chip OAD and On-chip OADproject Be sure to comment out the two lines of code in the following figure. The two lines of code in no OAD project have to be annotated.



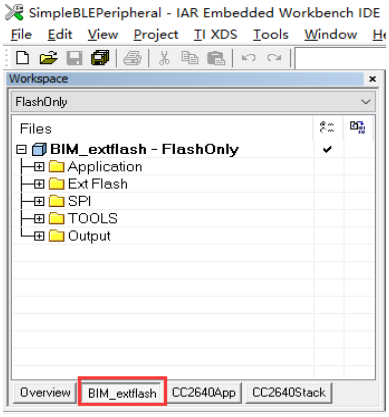
5.3.Off-chip OAD

Off-chip OAD is divided into two configurations: with and without power-down storage function. Take the with power-down storage function as an example to illustrate the Off-chip OAD process.

5.3.1 BIM_extflashproject(BIM file production)

BIM-Boot Image Manager, the software bootloader, startup the code. CC2640 power-on reset, check Off-chip flash whether has a new program that needs to be upgraded to CC2640 internal flash by BIM.

Open SimpleBLEPeripheralproject, switch to BIM_extflashproject (this project is only used in Off-chip OAD), set the external flash pin and ID. Path C: \ TTC_BLE_CC2640_SDK \ 1.0.7 \ TTC_CC2640_SDK \ Projects \ ble \ SimpleBLEPeripheral \ CC26xx \ IAR, such as Figure：



(1) view the schematic, set the flash SPI interface;

(2) According to the type of flash used, set the vendor ID and device ID in bsp.h;

C: \ TTC_BLE_CC2640_SDK \ 1.0.7 \ TTC_CC2640_SDK \ Projects \ ble \ util \ BIM_extflash \ CC26xx \ Source \ CC26XXST_0120

(3) recompile the project, generate BIM_ext.hex file. file path:

C: \ TTC_BLE_CC2640_SDK \ 1.0.7 \ TTC_CC2640_SDK \ Projects \ ble \ util \ BIM_extflash \ CC26xx \ IAR \ FlashOnly \ Exe

5.3.2 CC2640project Selection

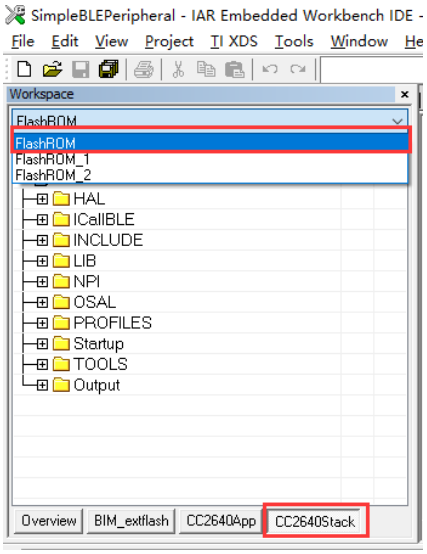
Refer to the document "SimpleBLEPeripheralproject configuration instructions .txt" to select the appropriate project according to the actual requirements (whether you need power-down storage and Bluetooth binding). Special Description: When using Off-chip OAD, the configurations of "CC2640App" and "CC2640Stack" must match the configuration of the two projects in the configuration instructions.

Document path: C: \ TTC_BLE_CC2640_SDK \ 1.0.7 \ TTC_CC2640_SDK \ Projects \ ble \ SimpleBLEPeripheral

The following to FlashOnly_OAD_ST_ExtFlash and FlashROM configuration as an example。

5.3.3 CC2640Stackproject Configuration

Switch to CC2640Stack project, select the Flash ROM configuration, this project no need other modifications, recompile the project.



5.3.4 CC2640Appproject Setting

(1)Open CC2640Appproject, select FlashOnly_OAD_ST_ExtFlash configuration, as shown in Figure 2-3-1.

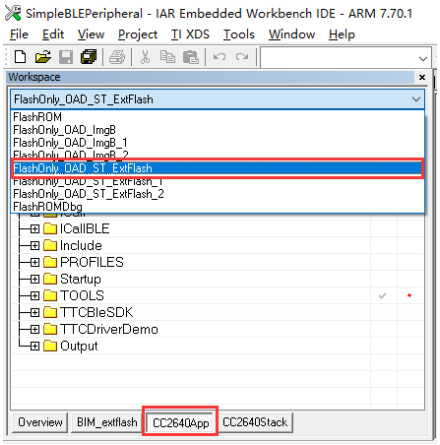


Figure 2-3-1 congfig Setting

（2）Re-compile the project, generate the protocol stack, the application's composite file OAD_FULL_IMAGE.hex.

Path: C: \ TTC_BLE_CC2640_SDK \ 1.0.7 \ TTC_CC2640_SDK \ Projects \ ble
\ SimpleBLEPeripheral \ CC26xx \ IAR \ ApplicatIOn \ CC2640 \ FlashOnly_OAD_ST_ExtFlash \ Exe

5.3.5 Make a Programmable File

（1）Open Flash Programmer 2, select Multiple mode, load two flash image files, and program. As shown in Figure 2-4-1.

- 3.1 steps (3) Generated BIM_ext.hex
- 3.4 steps(2) generated OAD_FULL_IMAGE.hex

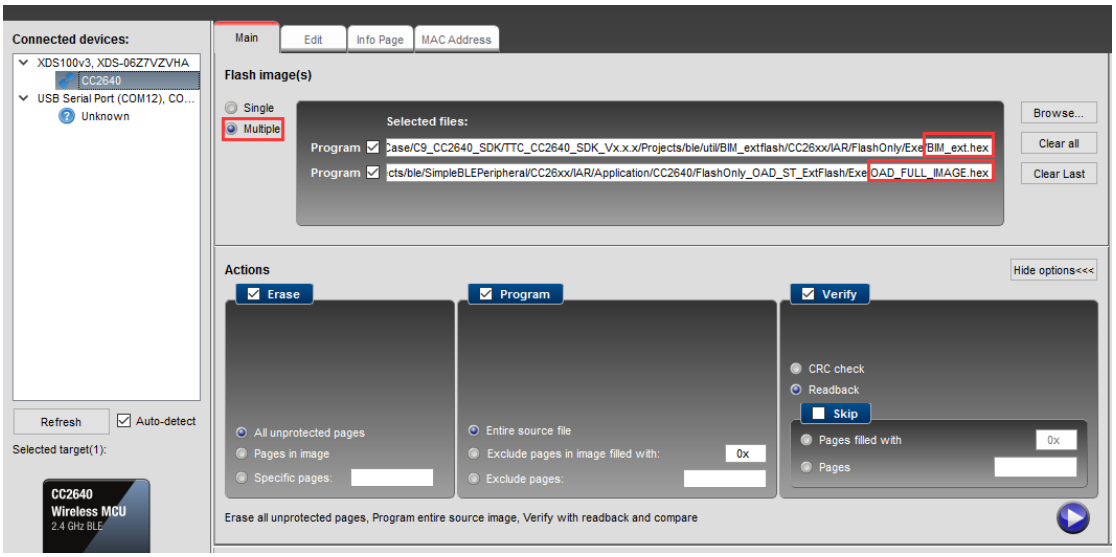


图 2-4-1 Program

（2）Generates BIM_ext.hex and OAD_FULL_IMAGE.hex merge files (for production program).

After completing step (1) above, read CC2640 internal flash content, that is, the final merger file. Select the file to save the path and customize the name. The implementation is shown in Figure 2-4-2.

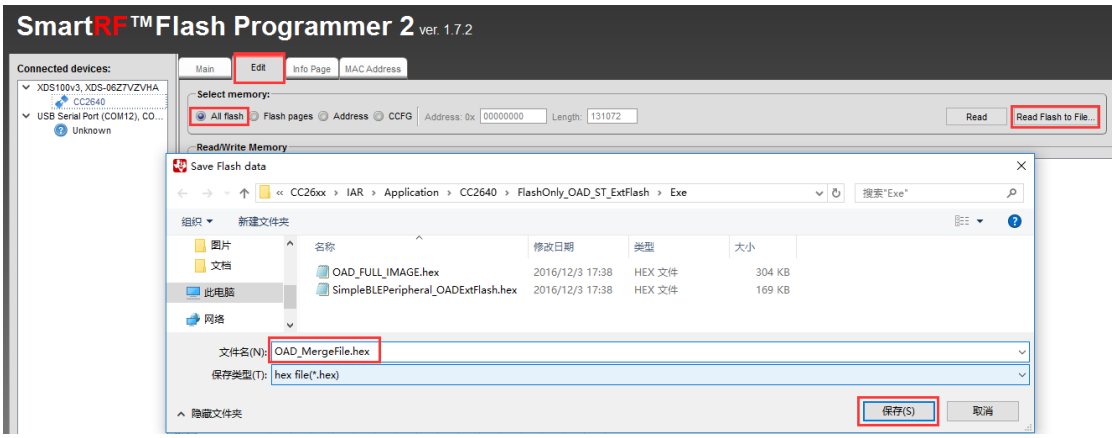
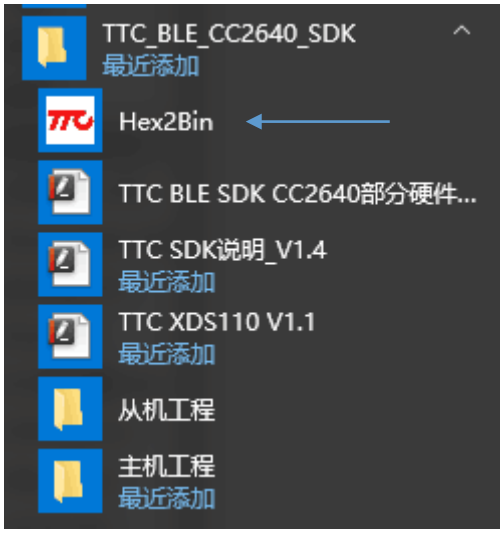


Figure 2-4-2 Reading the merge file

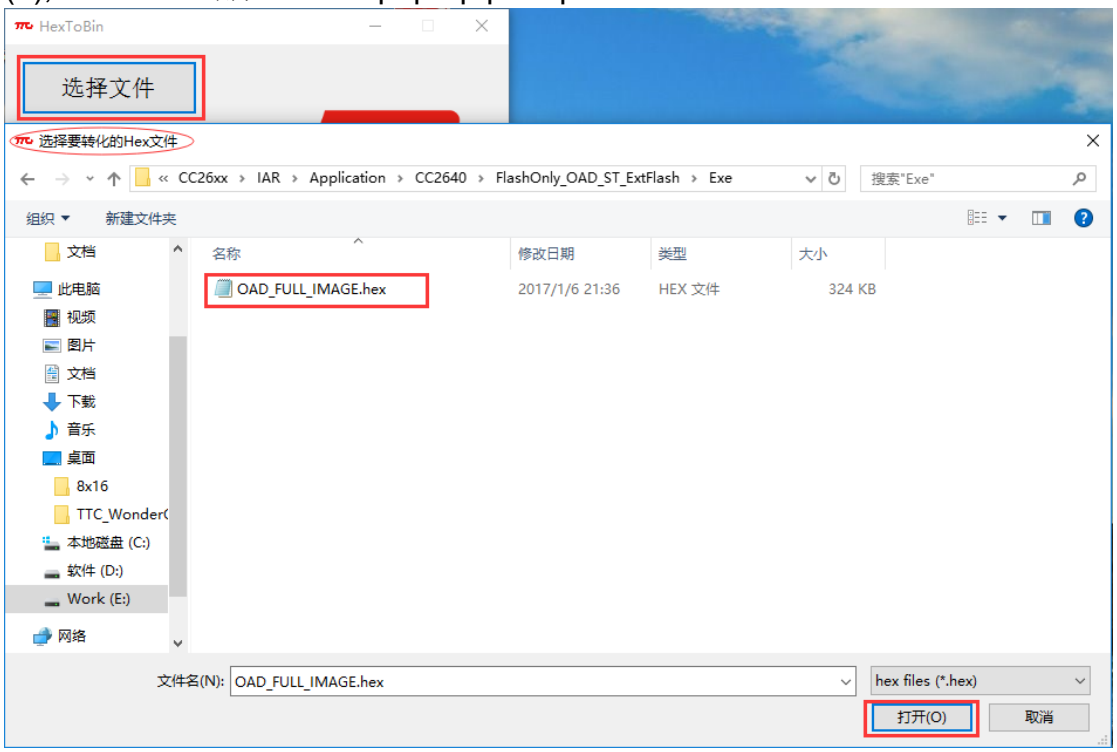
5.3.6 Phone APP OAD steps

5.3.6.1 Generates the bin file used by the Off-chip OAD

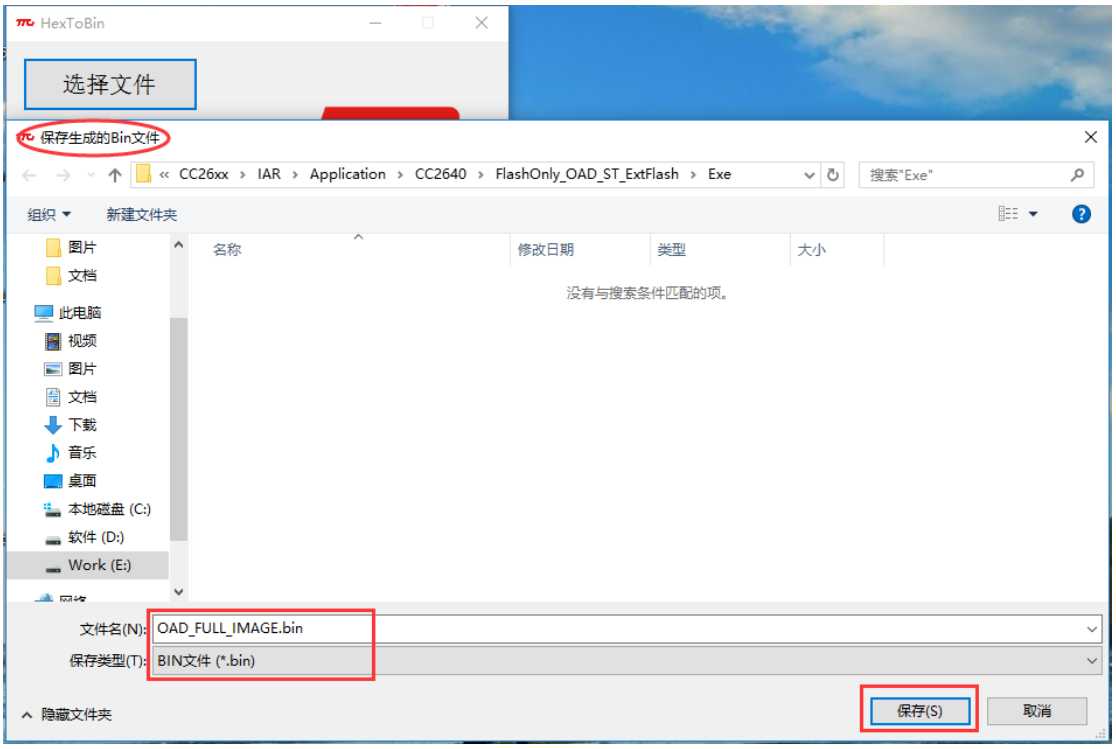
（1） In the Windows Start menu, open Hex2Bin.exe, as shown below



(2) Click "选择文件" to import OAD_FULL_IMAGE.hex generated in 3.4 steps (2), and click "确认" in the pop-up prompt window.



（3）Save the generated Bin file OAD_FULL_IMAGE.bin



5.3.6.2 Use TTC_BLE to have OAD process

（1) Import the bin file OAD_FULL_IMAGE.bin generated from 3.6.1 steps (3) to the mobile phone root directory Download directory;

(2) scan the QR code, download and install TTC_BLE, and turn on the phone Bluetooth；

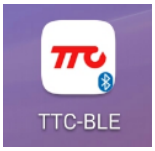


图 2-5-6 APP

(3) open the APP, click OAD, enter the OAD mode, as shown in Figure 2-5-7;

(4) click the device, and establish a connection with the device, as shown in Figure 2-5-8;

(5) Click "OAD", as shown in Figure 2-5-9;

(6) Select the OAD type "CC2640 Off-Chip OAD", as shown in Figure 2-5-10;

(7) After the success of the connection, the display Target Image Type is A; click "+", import bin file, file import, File Image Type B, set the transmission interval of 24ms, click to start, as shown in Figure 2-5-11;

(8) after transmission to 100%, do not need to operate APP. Wait about 10 seconds, the device reset, run the new program, and APP disconnect the link;

Note: If APP is connected to the device, Target Image Type is not A, you can try to restart the phone Bluetooth re-operation。

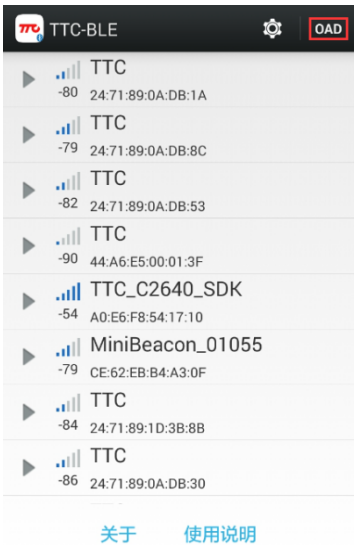


Figure 2-5-7 OAD mode settings

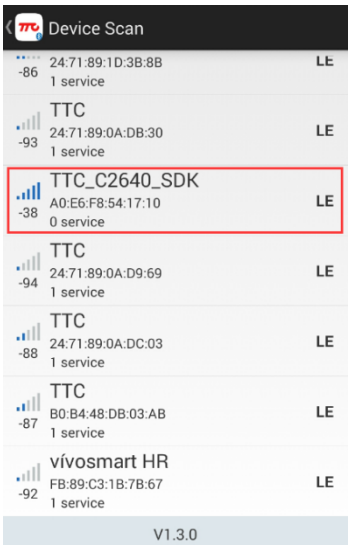


Figure 2-5-8 connect devices

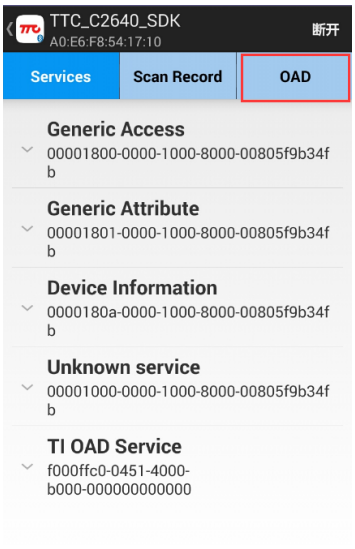


Figure 2-5-9 OAD

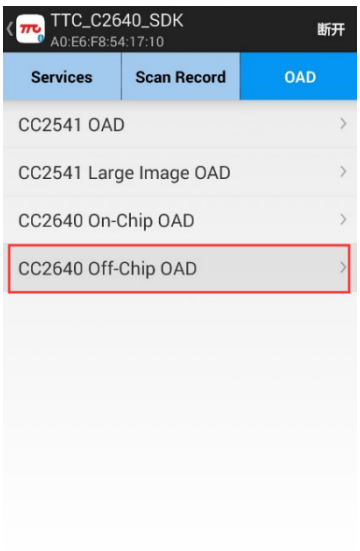


Figure 2-5-10 OAD type

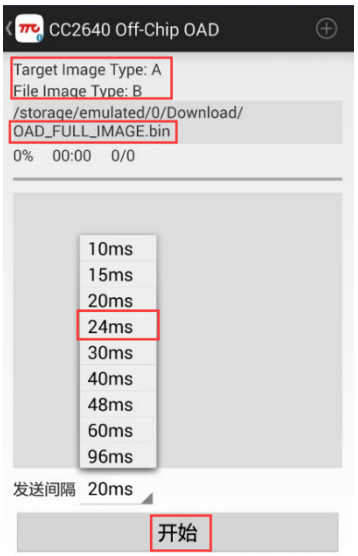


Figure 2-5-11 Import file

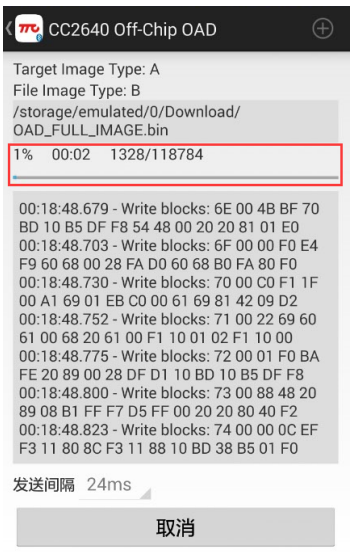


Figure 2-5-12 Start OAD

5.4.On-chip OAD

5.4.1 On-chip OAD INTRODUCTION

CC2640 When using On-chip OAD, the internal flash contains BIM, ImageA, ImageB, and Stack. To simplify the operation, the SDK has combined the BIM, ImageA, and Stack parts into a file OAD_Merge_x.hex (different packages, different configurations, merging files are not consistent), and only need to update ImageB on On-chip OAD.

Merge file path: C: \ TTC_BLE_CC2640_SDK \ 1.0.7 \ TTC_CC2640_SDK \ TTC_CC2640_SDK \ Projects \ ble \ SimpleBLEPeripheral \ CC26xx \ IAR \ ApplicatIOn \ CC2640 \ FlashOnly_OAD_ImgB_x \ OADMergeFile

On-chip OADYes three configurations: FlashOnly_OAD_ImgB, FlashOnly_OAD_ImgB_1,

FlashOnly_OAD_ImgB_2, specifically how to choose See "SimpleBLEPeripheralproject configuration instructions .txt", document path: C: \ TTC_BLE_CC2640_SDK \ 1.0.7 \ TTC_CC2640_SDK \ Projects \ ble \ SimpleBLEPeripheral

The following is a FlashOnly_OAD_ImgB configuration as an example.

5.4.2 On-chip OAD Operation Steps

5.4.2.1project setting

When using On-chip OAD, you do not need to configure, compile BIM_extflashproject and CC2640Stackproject. Change the project configuration to FlashOnly_OAD_ImgB, as shown in Figure 3-2-1.

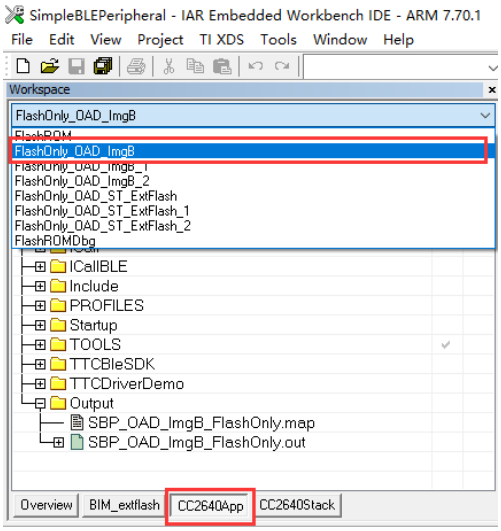


Figure 3-2-1 Select the project configuration

Compile and compile the application will generate the ImageB file: SBP_OAD_ImgB.hex, the path is C: \ TTC_BLE_CC2640_SDK \ 1.0.7 \ TTC_CC2640_SDK \ Projects \ ble \ SimpleBLEPeripheral \ CC26xx \ IAR \ ApplicatIOn \ CC2640 \ FlashOnly_OAD_ImgB \ Exe

5.4.2.2 Program burn

Open the Flash Programmer 2, select the Multiple mode, load two flash image files, and burn, as shown in Figure 3-2-2. Two image files are as follows:

- The merged file OAD_Merge.hex is mentioned in Section 4.1
- SBP_OAD_ImgB.hex generated in section4.2.1

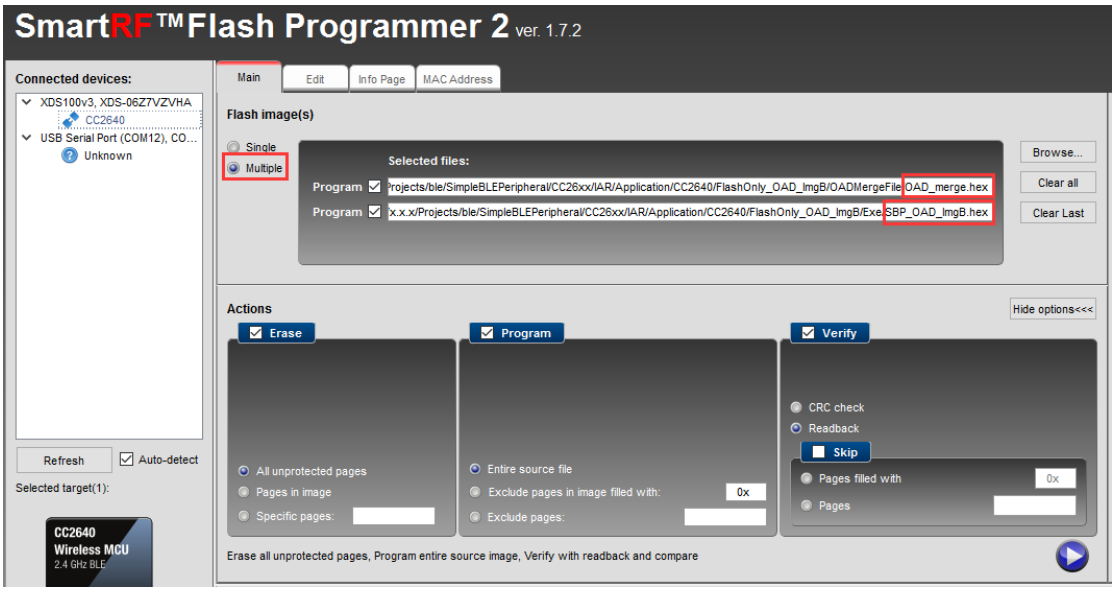
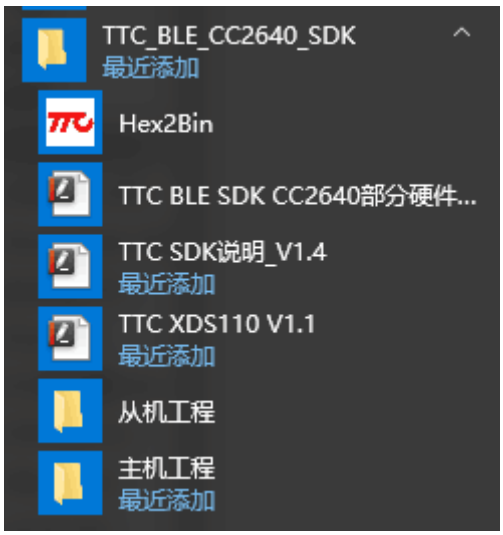


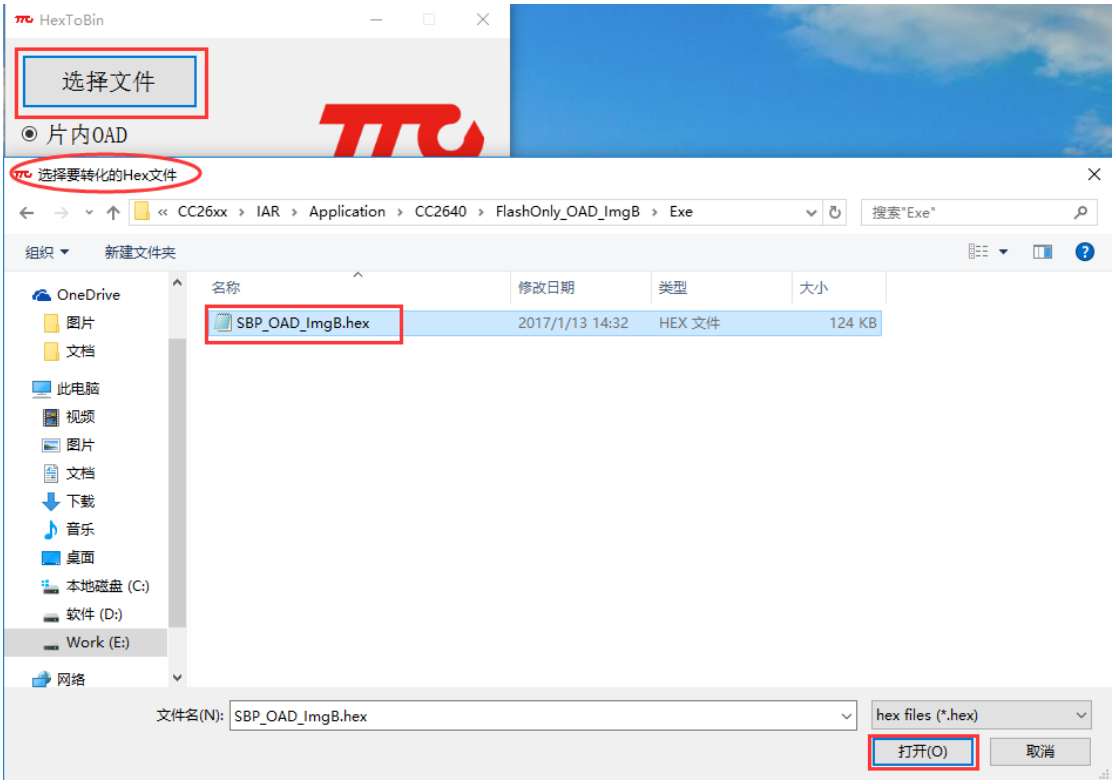
图 3-2-1 烧录

5.4.2.3 Generates the bin file used by the On-chip OAD

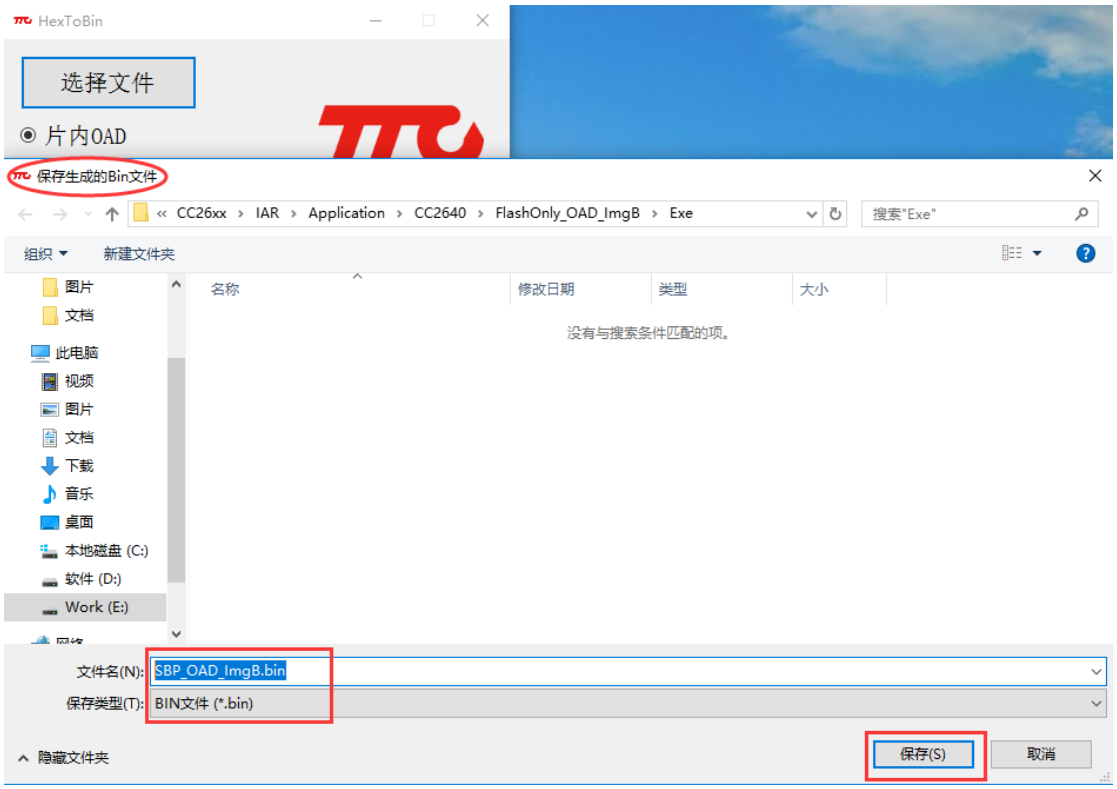
（1）In the Windows Start menu, open Hex2Bin.exe, as shown in the figure



（2）Click "选择文件", import 4.2.1 step generated SBP_OAD_ImgB.hex, in the pop-up prompt window, click "确认".



（3）Save the generated Bin file SBP_OAD_ImgB.bin



5.3.2.4 use TTC_BLE to OTC operation

（1) bin file SBP_OAD_ImgB.bin generated by Section 4.2.3 Steps (3) to import to mobile phone root directory Download directory;

(2) scan the QR code, download and install TTC_BLE, and turn on the phone Bluetooth

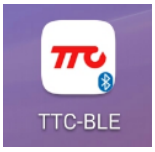


figure3-2-6

(3) open the APP, click OAD, enter the OAD mode, as shown in Figure 3-2-7;

(4) click on the corresponding equipment, and equipment to establish a link, as shown in Figure 3-2-8;

(5) find 0xf000ffd1 channel, write 01, click write, as shown in Figure 3-2-9 and 3-2-10;

(6) the device automatically disconnect, return to the previous interface, click on the connection, as shown in Figure 3-2-11.

(7) After the connection is successful, only TI OAD Service, click OAD, as shown in Figure 3-2-12.

(8) Select the OAD type "CC2640 On-Chip OAD", as shown in Figure 3-2-10;

(7) Show Target Image Type is A; Click "+" to import bin file, File Image Type B, set transmission interval of 24ms, click to start, as shown in Figure 3-2-14 and 3-2-15;

(8) after transmission to 100%, do not need to operate APP. Device reset, run new program, disconnect with APP, as shown in Figure 3-2-16 and 3-2-17;

Note: If APP is connected to the device, Target Image Type is not A, you can try to restart the phone Bluetooth re-operation.

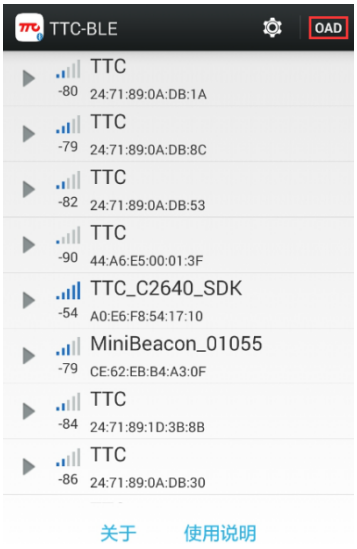


Figure 3-2-7 OAD mode instruction

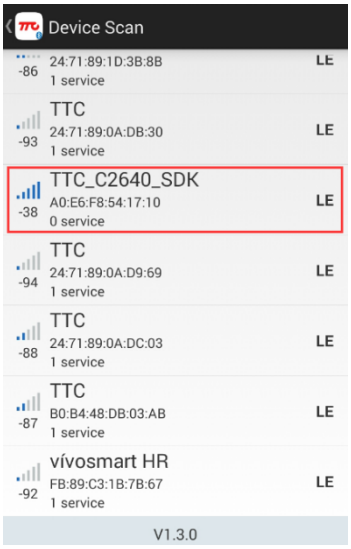


Figure 3-2-8 connect the device

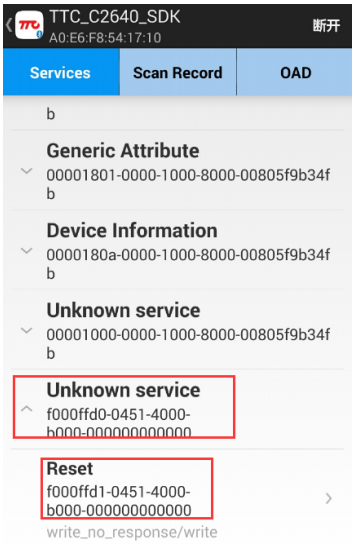
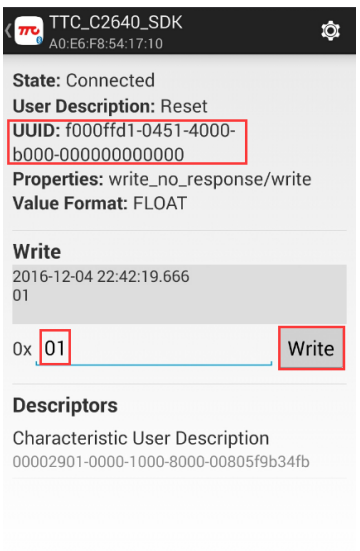
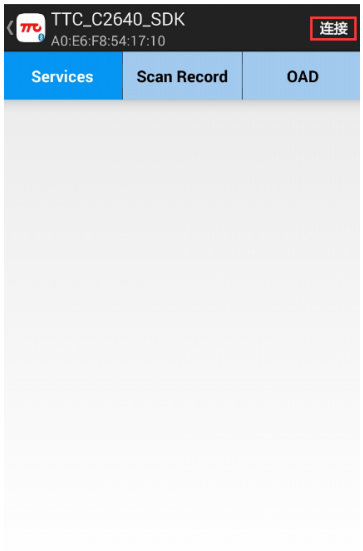


Figure 3-2-9 OAD channel





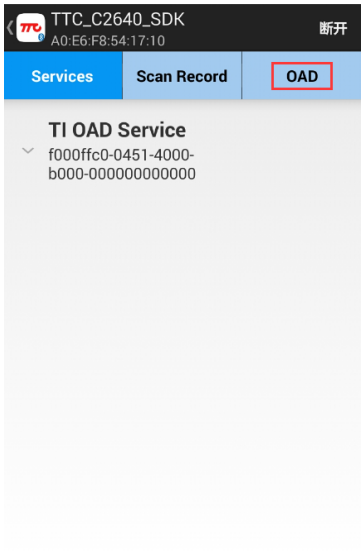
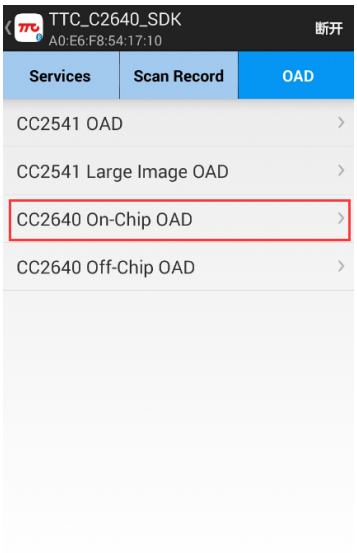
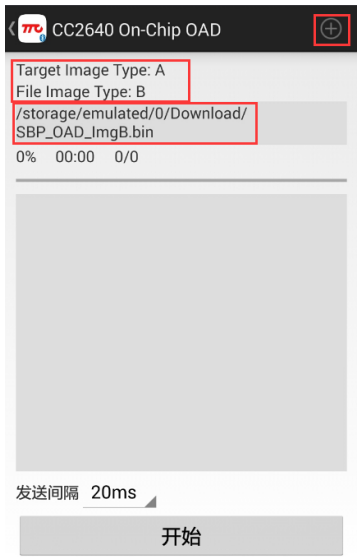


Figure 3-2-10 OAD instruction mode

Figure 3-2-11 Reconnection

Figure 3-2-12 OAD





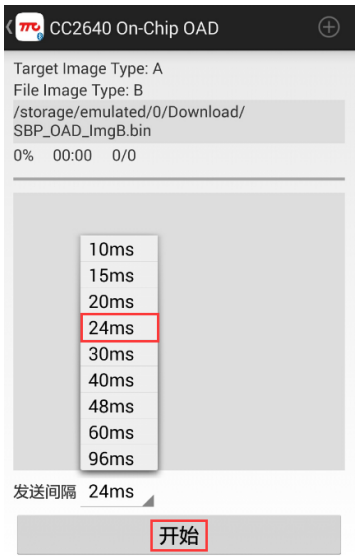


Figure 3-2-13 Select On-chip OAD

Figure 3-2-14 Load the file

Figure 3-2-15 Set the interval

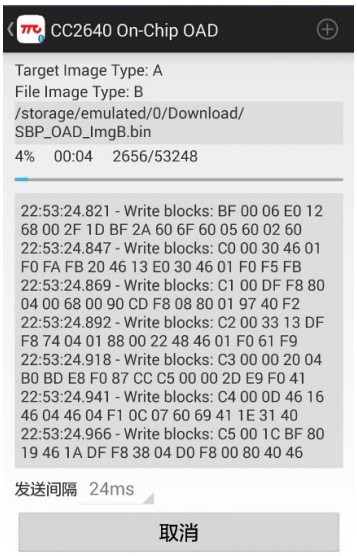


Figure 3-2-16 is in OAD

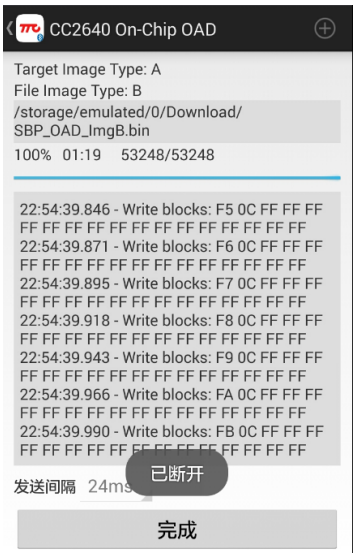


Figure 3-2-17 OAD completed

6. Production test related introduction

6.1 Test pin definition

#define TTCTEST_ENTER_TEST_IO IOID_0 // Pull low before power on, entering test mode
#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 //

These definitions can be found / modified on TTCSDKBoard.h.
Note: This is only allowed to modify the following IOID, prohibit the modification of the previous macro flag.
Description:
TTCTEST_ENTER_TEST_IO
pull low test mode trigger pin before powering on, causing the SDK to enter the test mode and execute the relevant test code. According to the test results we can know whether the module is working properly.

TTCTEST_PWM_PIN
48M test output port.

TTCTEST_UART_RX
Test the RX pin of the UART.

TTCTEST_UART_TX
Test the TX pin for the UART.

TTCTEST_UART_WAKEUP
Test the UART's WAKEUP pin

During the test process, the AT command can be used to control the test items.

6.2. Test Methods

After entering the test mode, we can send the AT command to tell the module we want to test the contents of the instructions, the format is as follows：
Command format : DATA0 | DATA1 | DATA2 | DATA 3 |....| DATAn
0X33 | Command head | length | data | | Checksum

Length: the total length of the data, does not contain DATA0, DATA1, DATA2, Checksum.
Checksum: the sum of DATA0 + DATA1 + ... DATA (n-1)。

Test command introduction

Description	Command	Response	Mark
Check the command	33 FF 01 00 33	33 FF 01 01 34	
32K test	33 C0 01 00 F4	33 C0 01 01 F5	pull high the WAKEUP pin after send test command,wait 500ms, if you can accept the 32K reply command is normal.
48M test	33 C1 01 00 F5	33 C1 01 01 F6	the TTCTEST_PWM_PIN pin is detected After sending the test command, and there will output a period of 20us PWM square wave. The duty cycle is 50% and the PWM is automatically turned off after 500ms.
IO test	33 C2 01 00 F6	33 C2 01 01 F7	All pins are detected immediately after sending the test command, except for the UART_TX / UART_RX / WAKEUP pin. The two adjacent feet are not the same. After 100 ms all pin levels are inverted and detected again.
TX test	33 C3 01 00 F7	33 C3 01 01 F8	test current after sending the test command, then pull high the WAKEUP pin, the current should be about 6.043mA. After the test is completed, you need to send the exit command.
RX test	33 C4 01 00 F8	33 C4 01 01 F9	test current after sending the test command, then pull high the WAKEUP pin, the current should be about 6.471mA. After the test is completed, you need to send the exit command.。
Exit test mode	33 C5 01 00 F9	33 C5 01 01 FA	
User-defined test	33 E0 01 00 14	33 E0 01 01 15(success) 33 E0 02 XX XX XX（failure）	the code in the module will call the user's code which prepared in advance after sending this command, if the success will return a successful response.

Read the MAC address	33 AD 01 00 E1	33 AD 06 XX XX XX XX XX XX XX	After sending the command will open the Bluetooth broadcast, and return to the MAC address. Only the command can turn on the Bluetooth broadcast
Version test	33 F0 01 00 24	33 F0 VH VL XX	Version number: 0xVHVL, check: XX

7.Contact us

深圳市昇润科技有限公司
ShenZhen ShengRun Technology Co.,Ltd.
Tel: 0755-86233846　Fax: 0755-82970906
Website：www.tuner168.com
Alibaba Shop：http://shop1439435278127.1688.com
E-mail:　marketing@tuner168.com
地址：广东省深圳市南山区西丽镇龙珠四路金谷创业园 B 栋 6 楼 601-602
ADD：　the 6F，BBlock of jingu Pioneer Park，　longzhu 4th Road，　Nanshan District，Shenzhen，China

Appendix A parameter description

```
typedef enum {                          //GAPROLE State
    GAPROLE_INIT = 0,                   // Wait Bluetooth startup state
    GAPROLE_STARTED,                    // Bluetooth has been started but has not yet been broadcast
    GAPROLE_ADVERTISING,                // Broadcast state
    GAPROLE_ADVERTISING_NONCONN,        //can not connect to broadcast status
    GAPROLE_WAITING,                    // The device is on but not broadcast, waiting for the broadcast startup
    GAPROLE_WAITING_AFTER_TIMEOUT,      // The device was just disconnected but not yet broadcast，
                                        // Waiting to start next broadcast
    GAPROLE_CONNECTED,                  //Connection state
    GAPROLE_CONNECTED_ADV,              //Connection+broadcast state
    GAPROLE_ERROR                       // Error (invalid state)
} gaprole_States_t;

typedef enum{                           //TTC SDKDriver type
    TTCDRIVER_TYPE_UART = 0,            //UARTDriver
    TTCDRIVER_TYPE_SPI ,                //SPIDriver
    TTCDRIVER_TYPE_I2C,                 //IICDriver
    TTCDRIVER_TYPE_WATCHDOG,            //WatchDogDriver
    TTCDRIVER_TYPE_TIMER ,              //TimerDriver
    TTCDRIVER_TYPE_SENSOR_ADC,          //SensorController-ADCDriver
    TTCDRIVER_TYPE_SIZE,                // invalid state
}TTCDriverType_t;

typedef enum{
    /* Driver Initialization state */
    TTCDRIVER_INIT_SUCCESS= 0x00,       // Initialized successfully
    TTCDRIVER_INIT_IO_FAILED = 0x01,    //Initialization failed: IO configuration failed
    TTCDRIVER_INIT_RAM_FAILED= 0x02,    //Initialization failed: TX or RX cache is not allocated or allocation failed
    TTCDRIVER_INIT_SYSPARA_FAILED= 0x03, //Initialization failed: Semaphore or Message handle error
    TTCDRIVER_INIT_FAILED_IS_OPENED= 0x04,//Initialization failed: Driver enabled
    TTCDRIVER_INIT_PARA_ERROR = 0x05,   //Initialization failed: parameter error
    TTCDRIVER_CLOSE_SUCCESS = 0x06,     //close success
    /* Driver state */
    TTCDRIVER_HANDLE_ERROR = 0x07,      //handle error
    TTCDRIVER_TRANSFER_DATA_SUCCESS = 0x08,  // Transfer data successfully or put the cache successful
```

```
    TTCDRIVER_TRANSFER_DATA_BUSY= 0x09,   // Transmission failed, wait for the TX cache to send before sending
    TTCDRIVER_TRANSFER_DATA_RAM_OVERFLOW= 0x0A, // Transfer data overflow
    TTCDRIVER_TRANSFER_DATA_ERROR= 0x0B,          // Failed to transfer data
    TTCDRIVER_SLEEPED = 0x0C,                     // Not awakened
    TTCDRIVER_WAKEUP = 0x0D,                      // Awakened
    TTCDRIVER_INFO_SIZE
}TTCDriverInfo_t;
typedef enum{
    TTCBLE_TYPE_READ_OPERATION = 0,          // The host has the read operation to the peripheral
    TTCBLE_TYPE_WRITE_OPERATION ,            // The host has the write operation to the peripheral (no need now)
    TTCBLE_TYPE_SIZE
}TTCBleType_t;

typedef enum{
    MANAGER_INFO_REQUEST_IO_SUCCESS= 0x00, //Apply for use IO success
    MANAGER_INFO_REQUEST_IO_FAILED= 0x01,   //Apply for use IO failure
    MANAGER_INFO_RELEASE_IO_SUCCESS = 0x02, //Apply to release IO success
    MANAGER_INFO_RELEASE_IO_FAILED = 0x03, //Apply to release IO failure
    MANAGER_INFO_CONTROL_IO_SUCCESS= 0x04, //OperatIOn IO Success
    MANAGER_INFO_CONTROL_IO_FAILED = 0x05, //OperatIOn IO failure
    MANAGER_INFO_IO_ID_ERROR = 0x06,//IOID error
    MANAGER_INFO_IO_HANDLE_ERROR= 0x07,//IO handle error
    MANAGER_INFO_IO_NOT_ALLOCATED= 0x08, // The IO is not allocated in the handle
    MANAGER_INFO_SIZE
}TTCBleSDKManagerInfo_t;
```