Kw36 多连接（服务端）

两块以上 KW36

1　首先导出 temp_sense 的 freertos 例程

2 修改 app_preinclude.h，作如下修改

```
#define gAppMaxConnections_c 8
#define gTmrStackTimers_c (6 + gAppMaxConnections_c)
```

不使用低功耗，则

```
#define cPWR_UsePowerDownMode 0
```

3　修改 temperature_interface.h

修改结构体为

```
typedef struct tmsConfig_tag
{
  uint16_t serviceHandle;
  int16_t temperature;
  bool_t* aValidSubscriberList;    //表示哪些注册设备是可使用的
  uint8_t validSubscriberListSize;  //可以连接设备的数量
} tmsConfig_t;
```

将以下声明函数修改，添加参数

```
bleResult_t Tms_Subscribe(tmsConfig_t *pServiceConfig, deviceId_t deviceId);

bleResult_t Tms_Unsubscribe(tmsConfig_t *pServiceConfig, deviceId_t deviceId);

bleResult_t Tms_RecordTemperatureMeasurement (tmsConfig_t *pServiceConfig);
```

4　修改temperature_service.c

将static deviceId_t mTms_SubscribedClientId; 注释

修改下面函数参数定义

```
static void Hts_SendTemperatureMeasurementNotification(tmsConfig_t *pServiceConfig,
uint16_t handle);
```

4.1 修改函数，**Tms_Start**

```
bleResult_t Tms_Start (tmsConfig_t *pServiceConfig)
{
      uint8_t mClientId = 0;

    /* reset all slots for valid subscribers */
    for(mClientId = 0; mClientId < pServiceConfig->validSubscriberListSize;
mClientId++)
    {
        pServiceConfig->aValidSubscriberList[mClientId] = FALSE;
    }

    return Tms_RecordTemperatureMeasurement(pServiceConfig);
}
```

4.2 修改函数**Tms_Stop**

```
bleResult_t Tms_Stop (tmsConfig_t *pServiceConfig)
{
    uint8_t mClientId = 0;

    /* reset all slots for valid subscribers */
    for(mClientId = 0; mClientId < pServiceConfig->validSubscriberListSize;
mClientId++)
    {
        pServiceConfig->aValidSubscriberList[mClientId] = FALSE;
    }

    return gBleSuccess_c;

}
```

4.3 修改函数**Tms_Subscribe**

```
bleResult_t Tms_Subscribe(tmsConfig_t *pServiceConfig, deviceId_t deviceId)
{
    if(deviceId >= pServiceConfig->validSubscriberListSize)
    {
        return gBleInvalidParameter_c;
    }

    pServiceConfig->aValidSubscriberList[deviceId] = TRUE;

    return gBleSuccess_c;
}
```

4.4 修改函数**Tms_Unsubscribe**

```
bleResult_t Tms_Unsubscribe(tmsConfig_t *pServiceConfig, deviceId_t deviceId)
{
    if(deviceId >= pServiceConfig->validSubscriberListSize)
    {
        return gBleInvalidParameter_c;
    }

    pServiceConfig->aValidSubscriberList[deviceId] = FALSE;
    return gBleSuccess_c;
}
```

4.5 修改函数**Tms_RecordTemperatureMeasurement**

```
bleResult_t Tms_RecordTemperatureMeasurement (tmsConfig_t *pServiceConfig)
{
    uint16_t  handle;
    bleResult_t result;
    bleUuid_t uuid = Uuid16(gBleSig_Temperature_d);

    /* Get handle of Temperature characteristic */
    result = GattDb_FindCharValueHandleInService(pServiceConfig->serviceHandle,
gBleUuidType16_c, &uuid, &handle);

    if (result != gBleSuccess_c)
        return result;

    /* Update characteristic value */
    result = GattDb_WriteAttribute(handle, sizeof(uint16_t), (uint8_t*)&pServiceConfig-
>temperature);

    if (result != gBleSuccess_c)
        return result;
```

```
        Hts_SendTemperatureMeasurementNotification(pServiceConfig, handle);

        return gBleSuccess_c;

    }
```

4.6 修改**Hts_SendTemperatureMeasurementNotification**

```
static void Hts_SendTemperatureMeasurementNotification(tmsConfig_t *pServiceConfig,
uint16_t handle)
{
    uint16_t  hCccd;
    bool_t isNotificationActive;
    uint8_t   mClientId = 0;

    /* Get handle of CCCD */
    if (GattDb_FindCccdHandleForCharValueHandle(handle, &hCccd) != gBleSuccess_c)
        return;
    for(mClientId = 0; mClientId < pServiceConfig->validSubscriberListSize;
mClientId++)
    {
      if(pServiceConfig->aValidSubscriberList[mClientId])
      {
            if (gBleSuccess_c == Gap_CheckNotificationStatus
                        (mClientId, hCccd, &isNotificationActive) &&
                              TRUE == isNotificationActive)
            {
                  GattServer_SendNotification(mClientId, handle);
            }
      }
    }
}
```

5 修改temperature_sensor.c

5.1 找到mPeerDeviceId 修改为

```
static deviceId_t mPeerDeviceId[gAppMaxConnections_c] ={gInvalidDeviceId_c};
```

5.2 定义uint8_t mActiveConnections = 0;

```
static bool_t tmsValidClientList[gAppMaxConnections_c] = {FALSE};
```

5.3 修改tmsServiceConfig ，为

```
static tmsConfig_t tmsServiceConfig = {service_temperature, 0, tmsValidClientList,
gAppMaxConnections_c};
```

5.4 找到DisconnectTimerCallback 函数注释掉

```
static void DisconnectTimerCallback(void* );
```

5.5 找到BleApp_Config，修改

```
tmsServiceConfig.temperature = 100 * BOARD_GetTemperature();
```

5.6 找到BleApp_ConnectionCallback，

5.6.1 在case gConnEvtConnected_c，修改

1) mPeerDeviceId[mActiveConnections] = peerDeviceId;

2) Tms_Subscribe(&tmsServiceConfig, peerDeviceId);

在 Tms_Subscribe 函数后mActiveConnections++;

5.6.2 在case gConnEvtDisconnected_c,

1) mPeerDeviceId[peerDeviceId] = gInvalidDeviceId_c;

2) Tms_Unsubscribe(&tmsServiceConfig, peerDeviceId);

在 Tms_Unsubscribe函数后mActiveConnections--;

找到cPWR_UsePowerDownMode，修改为

```
#if (cPWR_UsePowerDownMode)
            /* Go to sleep */
            #ifdef MULTICORE_HOST
                #if gErpcLowPowerApiServiceIncluded_c
                    PWR_ChangeBlackBoxDeepSleepMode(3);
                #endif
            #else
                if(mActiveConnections > 0)
                {
                        PWR_ChangeDeepSleepMode(1);
                }



                else
                {
                        PWR_ChangeDeepSleepMode(3);
                }
            #endif
            Led1Off();
#else
            LED_TurnOffAllLeds();
            LED_StartFlash(LED_ALL);
#endif
```

5.7 将DisconnectTimerCallback 函数注释掉

5.8 修改BleApp_SendTemperature

```
static void BleApp_SendTemperature(void)
{
    TMR_StopTimer(appTimerId);

    /* Update with initial temperature */
    tmsServiceConfig.temperature = BOARD_GetTemperature() * 100;
    Tms_RecordTemperatureMeasurement(&tmsServiceConfig);

/*
#if (cPWR_UsePowerDownMode)
     Start Sleep After Data timer
    TMR_StartLowPowerTimer(appTimerId,
                           gTmrLowPowerSecondTimer_c,
                           TmrSeconds(gGoToSleepAfterDataTime_c),
                           DisconnectTimerCallback, NULL);
#endif
*/
}
```

5.9 导出temp_coll工程

5.9.1 可以将低功耗关闭，如上个工程一样

5.9.2 找到BleApp_GattNotificationCallback

注释掉

```
/*
#if (cPWR_UsePowerDownMode)
         Restart Wait For Data timer
        TMR_StartLowPowerTimer(mAppTimerId,
                       gTmrLowPowerSecondTimer_c,
                       TmrSeconds(gWaitForDataTime_c),
                       DisconnectTimerCallback, &serverDeviceId);
#endif
*/|
```

6 分别将程序烧入两块板子，打开两个串口，一个板子按sw2开启广播，另一个按下sw2开启扫描，一会儿两个板子就可以连接成功，串口打印出消息。大约等5s之后获得第一条数据。随后就不再有输出

了，客户端也不再获得服务端的数据。服务端可以再按下sw2，可以再次开启广播，来连接另外一个开启扫描的板子。

若开启低功耗，则按下llwu后唤醒两个设备，两个设备自动一个开启广播，一个开启扫描，并自动连接，客户端约5s获得一次消息。然后两个设备进入休眠，可以通过按下llwu再次唤醒两个设备，这样服务端会再发消息，并且不再需要等待5s。

7　在不进入低功耗情况下，服务端只能发送一次数据，可以做以下修改使得客户端来不断获得服务器数据。因为这两个例程，简单来说就是，客户端发送notification，服务端接收到这个请求，发送数据给客户端，那么只要我们定时发送这个notification，就可以不断获得数据。

7.1 在temp_coll工程里，找到temperature_collector.c文件，定义定时器ID

```
static tmrTimerID_t mNotificationId;
```

7.2 定义该ID的回调函数

```
static void TimerNotificationCallback(void *pParam)
{
    uint16_t value = (uint16_t)gCccdNotification_c;

    /* Allocate buffer for the write operation */
    if( mpCharProcBuffer == NULL )
    {
        mpCharProcBuffer = MEM_BufferAlloc(sizeof(gattAttribute_t) + gAttDefaultMtu_c);
    }

    if( mpCharProcBuffer != NULL )
    {
        /* Populate the write request */
        mpCharProcBuffer->handle = mPeerInformation.customInfo.tempClientConfig.hTempCccd;
        mpCharProcBuffer->uuid.uuid16 = gBleSig_CCCD_d;
        mpCharProcBuffer->valueLength = 0;
        (void)GattClient_WriteCharacteristicDescriptor(mPeerInformation.deviceId,
                                                       mpCharProcBuffer,
                                                       (uint16_t)sizeof(value), (void*)&value);
    }
}
```

7.3 找到BleApp_StateMachineHandler，找到case为*mAppRunning_c*

在mpCharProcBuffer = NULL;后面开启定时器

```
TMR_StartLowPowerTimer(mNotificationId,
gTmrLowPowerIntervalMillisTimer_c,TmrSeconds(1),TimerNotificationCallback,NULL);
```

效果如图