cc2540速度测试

建立周期事件：

```
if ( events & SBP_PERIODIC_EVT )
{
  // Restart timer
  if ( SBP_PERIODIC_EVT_PERIOD )
  {
    osal_start_timerEx( simpleBLEPeripheral_TaskID, SBP_PERIODIC_EVT, SB
  }
  sendData();
  sendData();
  sendData();
  sendData();

  //HCI_TxDataBufferInit();
  return (events ^ SBP_PERIODIC_EVT);
}
```

在第一次产生SBP_PERIODIC_EVT事件后进入，在其中打开一个定时器，令其每次进入时都会再次产生SBP_PERIODIC_EVT，达到周期事件的目的。

我们会设置一个connection interval，这个interval表示的是两个event事件之间的间隔时间，也就是说，你不断的产生SBP_PERIODIC_EVT事件，在事件内进行一些动作，而每次这个事件结束后到下个事件发生时会有这么一个时间间隔。BLE这么处理是为了能够降低功耗，这也是蓝牙4.0设计的初衷。

事件内我们4次调用senddata（），每次调用会发送一个包，每个包20字节，这里注意下，每两次发包之间也会有150微秒的间隔。

周期事件的产生：

```
case GAPROLE_CONNECTED:
  {
    #if (defined HAL_LCD) && (HAL_LCD == TRUE)
      HalLcdWriteString( "Connected",  HAL_LCD_LINE_3 );
    #endif // (defined HAL_LCD) && (HAL_LCD == TRUE)
    simpleBLEState = BLE_STATE_CONNECTED;
     osal_start_timerEx( simpleBLEPeripheral_TaskID, SBP_PERIODIC_EVT, SI
    osal_setClock(0);
//  osal_start_timerEx( simpleBLEPeripheral_TaskID, SBP_PERIODIC_EVT, SBP_

  }
  break;
```

发送这个动作一定是在两个设备建立连接或者配对绑定后才会成功。在两个设备建立连接后，我们调用定时器去产生一个事件SBP_PERIODIC_EVT，这也就是我们第一次事件的产生，那么之后就是周期事件的不断循环。这里调用了osal_setClock（）是为了到最后计算时间。

发送函数：

```
static uint8 sendData(void )
{
  static float rate = 0;
    static uint16 counter=0;
    uint8 burstData[20] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

    burstData[0] = (counter & 0xFF00)>>8;
    burstData[1] = (counter & 0xFF);

    attHandleValueNoti_t nData;
    nData.len = 20;
    nData.handle = 20;

    osal_memcpy( &nData.value, &burstData, 20 );

  // Send the Notification
  if (GATT_Notification( 0, &nData, FALSE )==SUCCESS)
  {
    counter++;
  }
  //HalLcdWriteStringValue("count:",counter,10,HAL_LCD_LINE_5);

  if(counter >1000)
  {
    osal_stop_timerEx(simpleBLEPeripheral_TaskID,SBP_BURST_EVT);
    time2 = osal_getClock();
     rate = ((counter*20)/time2);
     HalLcdWriteStringValue("count:",counter,10,HAL_LCD_LINE_5);
     HalLcdWriteStringValue("time2",time2,10,HAL_LCD_LINE_8);
    HalLcdWriteStringValue("rate",rate,10,HAL_LCD_LINE_7);
    counter = 0;
  }

  return 0;
```

调用notification发包，通过对返回值的判断来确定发包是否成功。这里注意，notification的返回值有6个，你的每个包不一会都发送成功，可以自己去把成功1000包后其他5种包的数量打出来，你会吃惊的。

这里当counter到1000时我们再次计算时间，通过字节数和事件就可以计算出速度了。

还有一点，notification（ ）的第一个参数是handle，也就是连接句柄，也就是一条虚链路，表明当前发送的通信通道，这里填的是0，也是说明了从机只会和一个设备建立连接。

发送的速率可以通过三个值来改变：

```
// Minimum connection interval (units of 1.25ms, 80=100ms)
#define DEFAULT_DESIRED_MIN_CONN_INTERVAL     8

// Maximum connection interval (units of 1.25ms, 800=1000ms
#define DEFAULT_DESIRED_MAX_CONN_INTERVAL     8

// Slave latency to use if automatic parameter update reque
#define DEFAULT_DESIRED_SLAVE_LATENCY         0

// Supervision timeout value (units of 10ms, 1000=10s) if a
#define DEFAULT_DESIRED_CONN_TIMEOUT          1000
```

interval上面讲过了，这里强调一点，他有一个最大和最小，这是给设备去选择的空间，在ios上要求max要比min大20，而安卓上则不做要求。

测量方法和结果：

1、用两个2540。
一个主机一个从机，速度因为发包中HCI buffer的问题，会有大量的包发送不成功，发送1000个包SUCCESS时，底层会丢弃几万个包，导致了速度的降低。
2、一个2540，一个usbdongle。
速度可以达到6K。因为我的时间用的是秒级的接口，不是特别精确，没有小数部分，在意的朋友可以换成ms的时间接口。（ti官网上给的是5.9k，不过是好几年前的）。
这里我把包发给usbdongle，然后用BTOOL打出来。

结果1的解决方法：
使用1.40协议栈，在主机中：

```
  // Setup a delayed profile startup
  osal_set_event( simpleBLETaskId, START_DEVICE_EVT );

  //turn on overlapped processing
  HCI_EXT_HaltDuringRfCmd(HCI_EXT_HALT_DURING_RF_DISABLE);
  HCI_EXT_OverlappedProcessingCmd(HCI_EXT_ENABLE_OVERLAPPED_PROCESSING);
}
```

从机中：

```
  //turn on overlapped processing
  HCI_EXT_HaltDuringRfCmd(HCI_EXT_HALT_DURING_RF_DISABLE);
  HCI_EXT_OverlappedProcessingCmd(HCI_EXT_ENABLE_OVERLAPPED_PROCESSING);

  // Setup a delayed profile startup
  osal_set_event( simpleBLEPeripheral_TaskID, SBP_START_DEVICE_EVT );
}
```

采用OverlappedProcessing后会解决HCI buffer问题，速度可以达到5k以上。
下附OverlappedProcessing连接：
http://processors.wiki.ti.com/index.php/OverlappedProcessing