BK2461

应用开发指南

FLIP51 MCU+RF

深圳市集贤科技有限公司

电话：0755-82571152 传真：0755-88373753

网址：http://www.uascent.com

地址：深圳市南山区科技园高新南一道创维大厦 C 座2 层

**版本记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author（s）** | **Description** |
| **0.1** | **Apr.31,2016** | **beiley** | **Initial version** |
| **0.2** | **May.31,2016** | **beiley** | **Add code Led Example** |
| **0.3** | **June.23,2016** | **beiley** | **Increase the driver package** |
| **04** | **8.23,2016** | **beiley** | **增加烧录脚如何做普通io口使用方式** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

1. **Bk2461芯片简介....................................................................................................................4**

1.1功能框图**..................................................................................................................5**

1.2封装尺寸**..................................................................................................................5**

* 1. GPIO MAPPING**........................................................................................................6**
  2. 电气参数**................................................................................................................7**

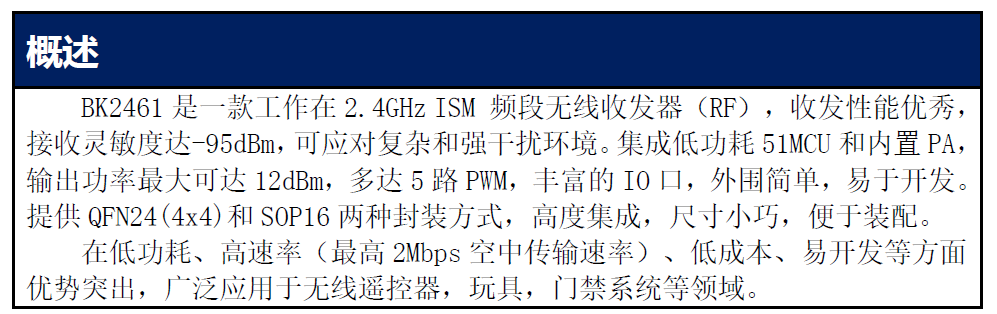
1. **开发板应用............................................................................................................................8**
   1. 板载资源**................................................................................................................8**
   2. 驱动安装**................................................................................................................8**
   3. 下载程序到flash**.................................................................................................10**
2. **bK2461 LAYOUT指南.............................................................................................................11**
   1. 芯片**.......................................................................................................................12**
   2. 3.3V电源**...............................................................................................................12**
   3. 射频**.......................................................................................................................13**
   4. 天线**.......................................................................................................................13**
   5. 晶体**.......................................................................................................................14**
   6. 原理图**...................................................................................................................15**
3. **BK2461驱动API...................................................................................................................28**
4. **BK2461收发包API...............................................................................................................34**

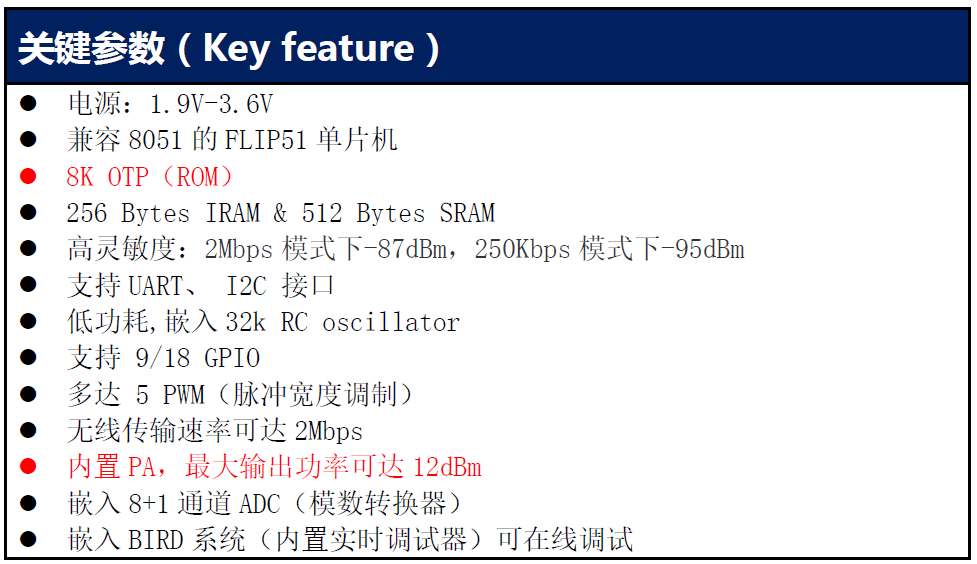
*附录1：*生产烧录步骤**...............................................................................................42**

*附录2：*不同功率spi配置**....................................................................................43**

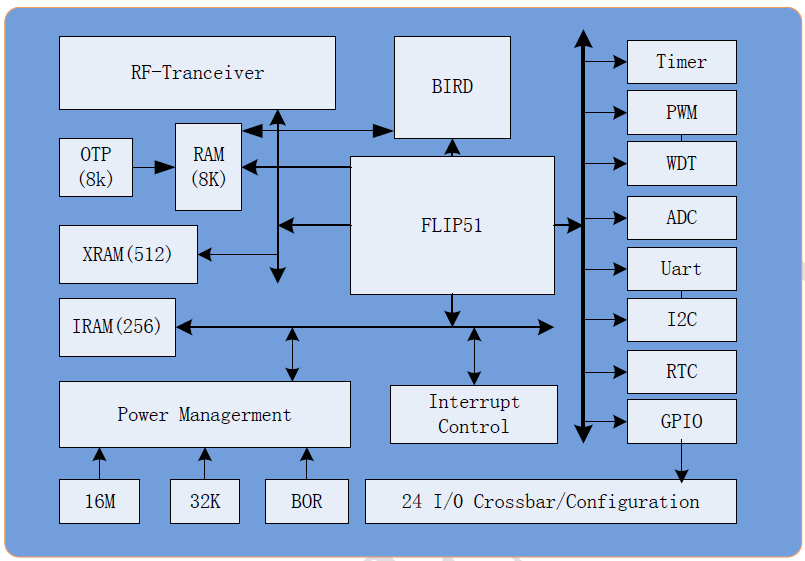
*附录3：*问题解答 **....................................................................................................44**

**1：****Bk2461 芯片简介**

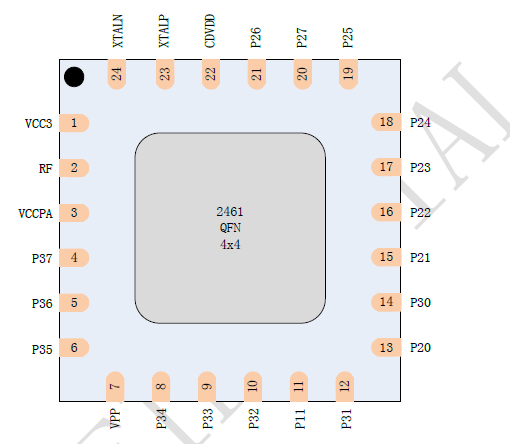
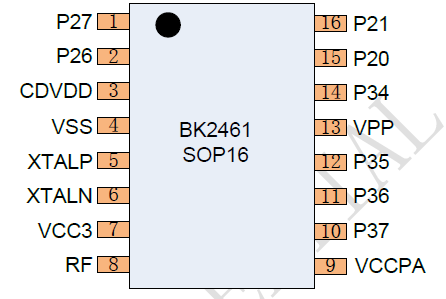




**1.1 功能框图**



**1.2 封装尺寸**

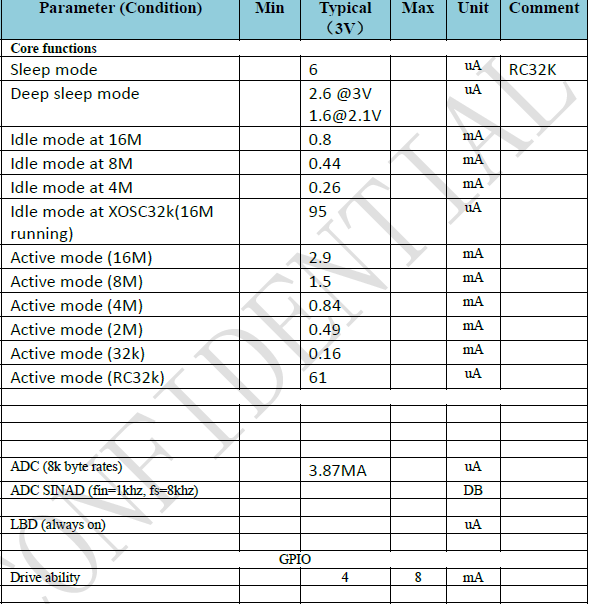


**1.3 GPIO Mapping**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **GPIO Number** | **Second Function** | **IO Direction** | **Download Mode** | **flash\_boot Mode** | **Debug Mode** | **Note** |
| P1.1 | int1\_n | I |  |  |  |  |
| P1.2 | PWM0 | O |  |  |  | VPP复用 |
| P2.0 | uart0\_rxd | I |  |  |  |  |
| P2.1 | uart0\_txd | O |  |  |  |  |
| P2.2 | SCL | O |  |  |  |  |
| P2.3 | SDA | O |  |  |  |  |
| P2.4 | PWM1 | O |  |  | tdo |  |
| P2.5 | PWM2 | O |  |  | tdi |  |
| P2.6 | PWM3 | O |  |  | tms |  |
| P2.7 | PWM4 | O |  |  | tck |  |
| P3.0 | ADC\_0 | I |  |  |  |  |
| P3.1 | ADC\_1 | I |  |  |  |  |
| P3.2 | ADC\_2 | I |  |  |  |  |
| P3.3 | ADC\_3 | I |  |  |  |  |
| P3.4 | ADC\_4 | I | spi\_mosi | boot\_si |  |  |
| P3.5 | ADC\_5 | I | spi\_miso | boot\_so |  |  |
| P3.6 | ADC\_6 | I | spi\_clk | boot\_sck |  |  |
| P3.7 | ADC\_7 | I | spi\_cs | boot\_csn |  |  |

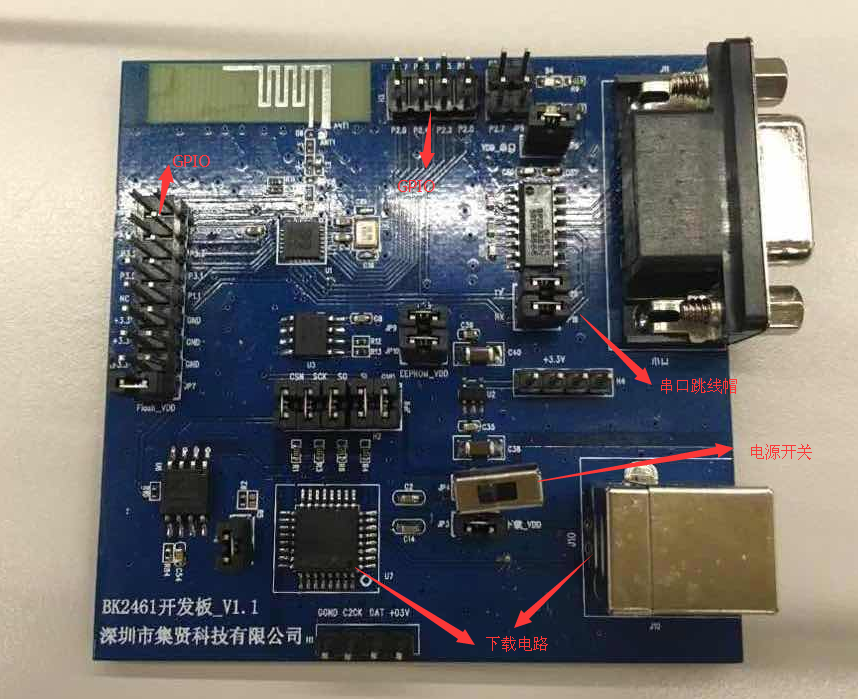
**图表 1 GPIO Mapping**

**1.4 电器参数**



**2：开发板应用**

**2.1 板载资源**



如上图所示，该开发板包括，下载器，串口，主控芯片 3大部分，主控芯片的io口已全部引出，开发板上的各模块可通过跳线帽来连接和断开，极大地方便了开发的使用。具体参照上图的标注！

**2.2 驱动安装**

PC第一次与开发板连接时，有可能会提示安装驱动，正确安装pc端驱动即可。（可参考如下安装方式安装，如未提示或已安装过，可直接跳过下述内容进入下一步）

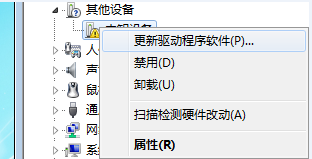
首先将开发板供烧录的 usb 端口连接在电脑上，同时 安装驱动文件（附件里有）：

安装方式为：

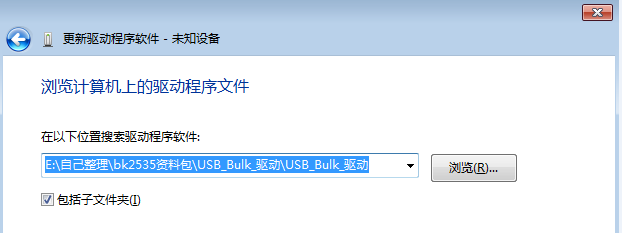
打开设备管理器，找到 有黄叹号的设备，如下图：

C:\Users\Administrator\Desktop\1.png

鼠标右击选择 “更新驱动程序软件”，如下图：



手动选择路径到我们的驱动文件，点击安装即可。如下图（注意：驱动仅支持win32操作系统，不支持win64）



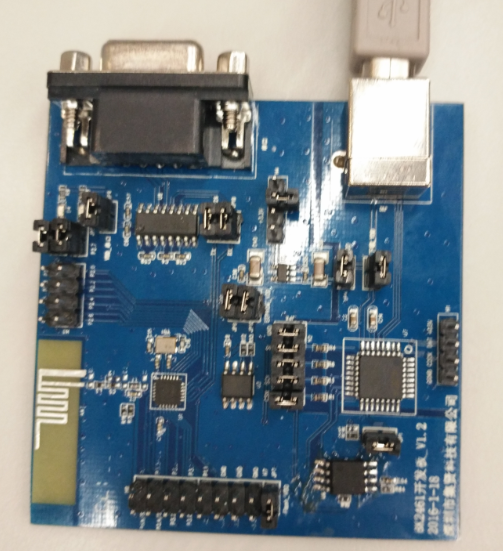
安装成功后 在设备管理器中可以看到成功的驱动，如下图：



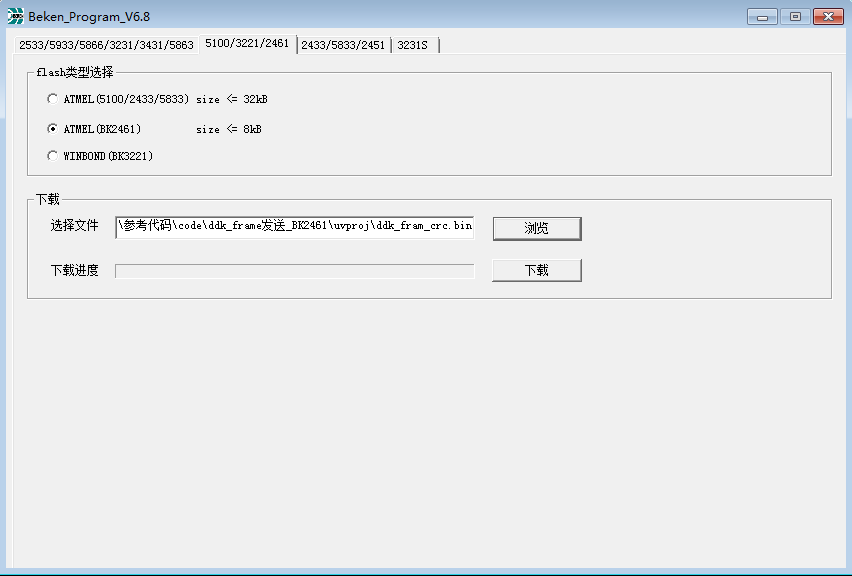
**2.3 下载程序到Flash**

接下来我们通过烧录按键点灯的例程来实现开发板的烧录。

首先，将开发板的跳线帽按照如下图片插上，并插上usb到开发板与电脑连接，如下图：



当开发板与电脑连接成功后，执行以下操作：



**使用步骤：**

1，打开烧录软件：Beken\_Program\_V6.8.exe。

2，选择正确的IC 型号，本次我们使用的是bk2461，如上图 “Flash类型选择”。

3，点击“浏览”按钮，进入目录：

bk2461资料包\参考代码\按键点灯代码\tx\uvproj\ddk\_fram\_crc.bin

选择正确下载bin文件(注意 ：烧录文件为带crc的文件“ddk\_fram\_crc.bin”)。

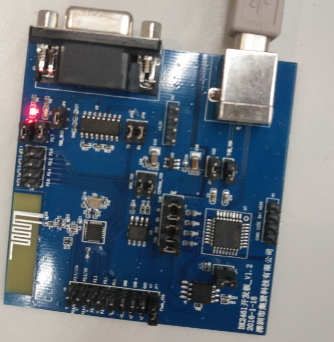
4，点击“下载” 按钮，等待烧录完成，如上图。

5, 下载完成后，在拿一块开发板按照以上流程下载 接收板程序。

下载路径为：

bk2461资料包\参考代码\按键点灯代码\rx\uvproj\ddk\_fram\_crc.bin

6，下载完成后 将两块板子上电，通过按下或者松开按键会控制led灯的亮灭。



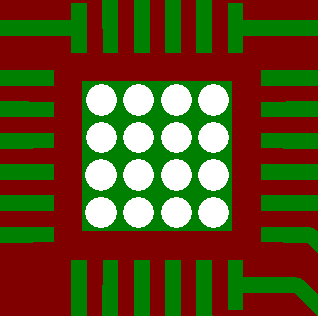
注意1：该烧录过程是Flash下载过程，将程序下载到了外挂的Flash中，试用于我们开发版，可以重复烧录，方便于开发。因为我们的bk2461 是otp的，在生产过程中，需要试用otp烧录方法，将程序烧录otp中，具体烧录方法见附录1

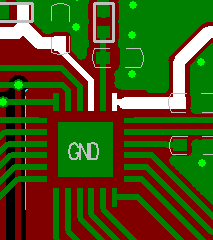
注意2： 软件编译使用的是 keil C51， 具体编写可参考我们给出的例程和相应的数据手册！

**3：BK2461 LAYOUT指南**

BK2461 的板子为两层板（top 层和bottom层），板材FR-4,表面处理工艺是喷锡板。元器件尽量放在top 面，bottom 面尽量有完整的地平面，元器件的接地PIN 应就近直接打过孔到bottom面。

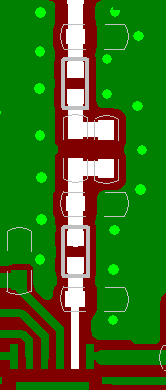
**3.1 芯片**

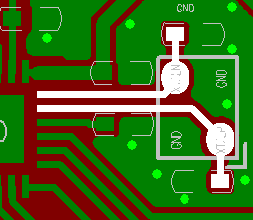
* 1. 芯片中心需加接地的大PAD，为保证与PCB 底层bottom的地平面完整接地，至少需要9-16个地过孔（Vias），过孔背面需要盖白油和绿油塞孔。
  2. 芯片正下方要求有完整的地平，尽量不要有走线和摆放元器件。
  3. 芯片PIN脚范围区域内使用keepout禁止铺铜，其目的是为了保证焊接的可靠性。

**3.2 3.3V电源**

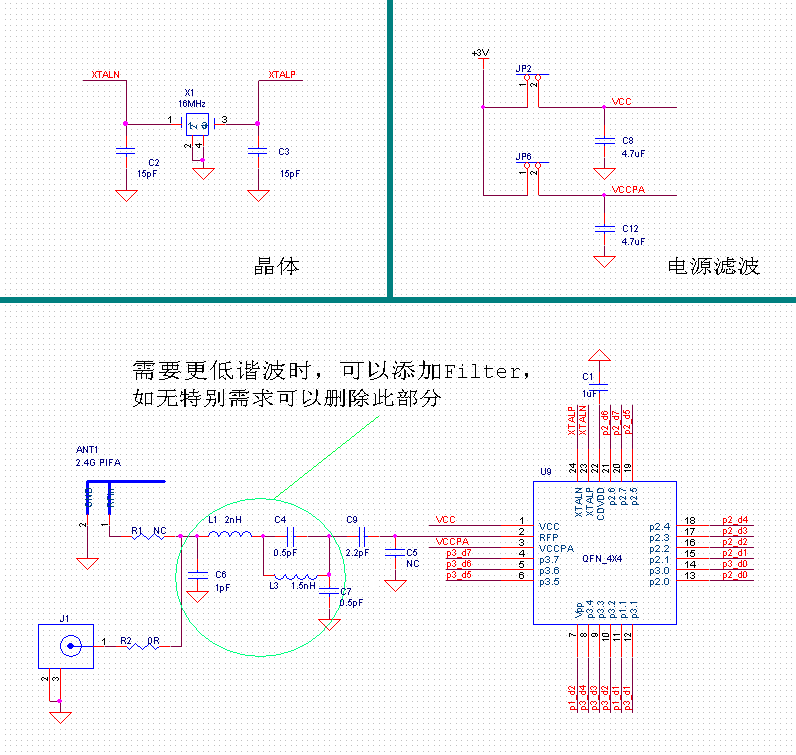
1. 芯片电源的滤波电容尽可能靠近芯片的电源PIN脚放置,小电容需要更靠近芯片PIN脚。
2. 电源走线需要先进入滤波电容，再从电容脚引线出来。

**3.3 射频**

由于射频匹配电路的**PCB Layout** 会严重影响到射频性能，实际layout 中必须考虑器件布局和走线，减少对射频匹配器件值的依赖，而减少频繁调试优化匹配。

1. 射频匹配电路对应的正下面要求有完整的地平面，不要有任何的走线和摆放元器件。
2. 射频线路左右包地，元器件与铺铜之间的间隔大于0.3MM，周边地过孔尽可能的多。
3. 天线尽可能远离螺丝孔。
   1. **天线**
4. 天线所在的所有层都需要净空，面积尽量大,且净空区周围需要打孔屏蔽。
5. PIFA 天线短路臂连到地以后，至少有两个对地过孔直接到 bottom 层。
6. 天线与 PCB 铺铜地之间的距离应大于0.5mm，靠近天线沿着地平面应该打一排对地过孔。
7. 天线周围最好不要有金属结构或元器件、铺地平面等，最多在其中一面距离一定间隔（至少5mm）可以放一些元器件,另外，天线位置要放在人使用时极少覆盖的地方.
   1. **晶体（16M）**
8. 晶体要求尽量靠近芯片，远离RF电源。
9. 晶体的两个地PAD要求接地过孔，确保晶体接地良好。
10. 晶体的正下方应有完整的地平面，晶体本体周围要求包地，隔离其他走线和元器件。
    1. **原理图**

## QFN4X4\_24Pin

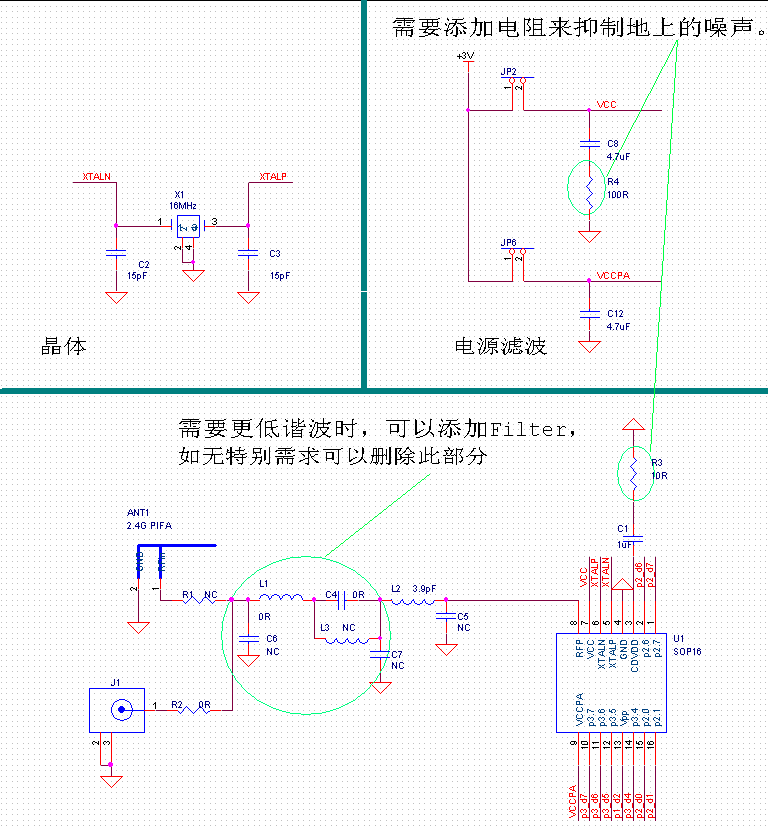


**图表 2 QFN4X4\_24Pin原理图**

图表1为DEMO板QFN4X4原理图，天线及测试口J1可以通过电阻R1，R2来选择。

如图表1所示，BK2461RF输出无需匹配，在需要滤波时，可以加入滤波器。天线需要匹配时也需要加入天线的匹配电路。

## SOP16



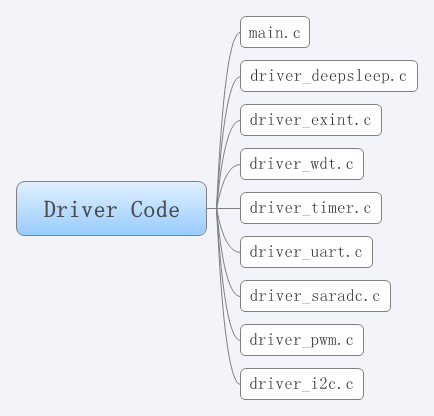
**图表 3 QFN4X4\_24Pin原理图**

图表2为DEMO板SOP16原理图，天线及测试口J1可以通过电阻R1，R2来选择。

如图表2所示，BK2461RF输出无需匹配，在需要滤波时，可以加入滤波器。天线需要匹配时也需要加入天线的匹配电路。由于SOP16封装Pin脚较长，且Gnd脚接触面积小，为了降低Noise需要在Pin7，Pin3上加入如图电阻来抑制噪声。

**3：BK2461 驱动API**

# （1）： 概述

本文用于描述BK2461各硬件模块驱动代码的API说明，程序的文件结构如下

各文件功能说明列述如下

# （2）： main.c

### void main(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 各模块的功能以test\_xxx()的形式在这里调用。 |

# （3）： driver\_deepsleep.c

Deep Sleep工作原理：

1. 设置唤醒 Deep Sleep的gpio。
2. 设置唤醒模式: 边沿或低电平触发
3. PCON2<3>先设置为1，再设置为0就进入了Deep Sleep模式，芯片停止工作
4. 将GPIO接地，芯片复位，重新工作

### void test\_deepsleep(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 设置wakeup pin及进入Deep Sleep模式 |

# （4）： driver\_extint.c

外部中断工作原理：

1. 只有标准外部中断1，对应P3.3；外部中断0被砍掉了
2. 只需要额外打开IO口第二功能：PALT0 |=0x02，但此处硬件有个bug需要设置成0x03
3. 中断的触发方式：低电平或下降沿触发
4. 将P3.3接地即触发中断

### void test\_ex1\_int(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 打开外部中断EX1 |

### void ex1\_int(void) interrupt 2

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 中断程序 |

# （5）： driver\_wdt.c

WDT分为两种情况：

1. 硬件WDT 往WDT（0xA6）寄存器写0xFF，开启硬件WDT，硬件WDT是无法关闭的。它在IDLE下是运行的，必须进行不间断喂狗（往WDT写0xA5）。（推荐使用SOFT WDT）
2. 软件WDT: 往WDT（0xA6）寄存器写0xA5，开启软件WDT，开后也可通过连续往WDT写 0xDE，0xAD值关闭软件WDT，否则需要喂狗（往WDT写0xA5）。关闭后也可以通过写0xA5，重新打开WDT。该WDT在IDLE默认运行，也可以往WDT连续写0xDE，0xDA，使其在IDLE下是不工作的（注意不在IDLE时候是工作的）。如果又重新想让其能在IDLE下工作，写0xD1即可。
3. WDT时间长度参考

默认X=7, T=512ms

说明：在配置好WDT之后，如需改变T的值，只要WDT<2:0>=X。在写WDT时候，保证WDT<7>==0且WDT<2:0>=X。之后再去配置WDT的各种工作状态。

### void wdt\_reset\_time(uint8 n)

|  |  |
| --- | --- |
| **参数** | n: 时间参数 |
| **返回值** | void |
| **注解** | 设置WDT的重启时间 |

### void hard\_wdt\_start(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 开启硬件WDT |

### void enter\_idle\_mode(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 进入idle模式 |

### void wdt\_feeddog(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 喂狗 |

### void soft\_wdt\_start(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 开启软件WDT |

### void soft\_wdt\_stop(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 停止软件WDT |

### void soft\_wdt\_idle\_stop(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 在idle模式下停止软件WDT |

### void test\_wdt(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 测试WDT |

# （6）： driver\_timer.c

工作原理：

1. 标准51 Timer 0、1、2
2. 只需要额外开启时钟：CLK\_EN\_CFG |= 0x40;
3. Timer仅剩Timer功能，没有外部输入PIN，故而不需要设置第二功能IO

### void timer0\_mode0(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 13-bit timer: TH0<7:0>+TL0<4:0> TL0低五位有效 |

### void timer0\_mode1(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 16-bit timer: TH0<7:0>+TL0<7:0> |

### void timer0\_mode2(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | Auto reload 8-bit timer: TL0用于计时， TH0用于保存TL0的重载值 |

### void timer0\_mode3\_timer0(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 8-bit timer: TL0用于Timer 0中断 |

### void timer0\_mode3\_timer1(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 8-bit timer: TH0用于Timer 1中断 |

### void timer1\_mode0(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 13-bit timer: TH1<7:0>+TL1<4:0> TL1低五位有效 |

### void timer1\_mode1(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 16-bit timer: TH1<7:0>+TL1<7:0> |

### void timer1\_mode2(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | Auto reload 8-bit timer: TL1用于计时，TH1用于保存TL1的重载值 |

### void timer2\_mode0(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 16-bit timer: TH2<7:0>+TL2<7:0> |

### void timer0\_int(void) interrupt 1

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | Timer 0中断程序 |

### void timer1\_int(void) interrupt 3

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | Timer 1中断程序 |

### void timer2\_int(void) interrupt 5

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | Timer 2中断程序 |

### void test\_timer(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 测试各种Timer模式 |

# （7）： driver\_uart.c

Uart有四种模式，mode 0 与 mode 2波特率是固定的，mode 1与mode 3波特率可设置，程序中使用的是mode 1

1. 打开Uart时钟 CLK\_EN\_CFG |= 0x20, 同时若Timer作为波特率发生器也要打开Timer的时钟CLK\_EN\_CFG |= 0x40
2. 设置第二功能IO PALT1 |=0x04， P2.0: Rx; P2.1: Tx
3. 若用中断方式开启中断ES0及EA，也可不开启而用轮询方式
4. 配置SCON0（0x98）寄存器

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [7:6] | 5 | REN | TB8 | RB8 | TI | RI |
| Mode选择 | 多机通信EN | Receive enable |  |  | Tx中断flag | Rx中断flag |
| 01(或11) | 0 | 1 | 0 | 0 | 0 | 0 |

1. 设置波特率：一般选用timer2，因为计数时间更长，波特率设置范围更加广阔。

T2CON设置成0x30（Tx/Rx时钟全部由Timer2产生）根据下面公式：

1. 对SBUF0寄存器读写，就可以进行收发操作。

### void uart\_config(uint16 baud)

|  |  |
| --- | --- |
| **参数** | baud: 波特率 |
| **返回值** | void |
| **注解** | 配置Uart的工作模式及波特率生成器 |

### void UartTx (uint8 val)

|  |  |
| --- | --- |
| **参数** | val：要发送的数据 |
| **返回值** | void |
| **注解** | 发送一个字节 |

### void uart\_int(void) interrupt 4

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | Uart中断程序 |

### void test\_uart(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 测试Uart的中断功能 |

# （8）： driver\_pwm.c

工作原理：

1. 打开PWM时钟

CLK\_EN\_CFG |= 0x10;

1. 使能PWM IO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| palt1[7] | palt1[6] | palt1[5] | palt1[4] | palt1[3] |
| PWM4\_IO | PWM3\_IO | PWM2\_IO | PWM1\_IO | PWM0\_IO |
| P2.7 | P2.6 | P2.5 | P2.4 | P1.2 |

1. 关闭模式选择电路

因为gpio1.2 是模式判断的IO，所以要想使用正常功能， 模式判断电路要关闭。

latch\_MODSEL = 1;

ana\_pwdMODSEL = 1

1. 配置PWM的周期，占空比

**freq = f\_sys/(2\*(PWMP+1)\*pwm\_resol) dc\_resol = 0**

**freq = f\_sys/(2\*(PWMP+1)\*(pwm\_resol + 768)) dc\_resol = 1**

**duty\_cycle = pwm\_dclsb/255 dc\_resol = 0**

**duty\_cycle = { pwm\_dcmsb[1:0] pwm\_dclsb[7:0]}/1023 dc\_resol = 1**

1. 使能PWM

pwm\_en = 1

### void pwm\_config(uint8 pwm\_n, uint8 standard, uint8 prescaler, uint8 dclsb, uint8 dcmsb)

|  |  |
| --- | --- |
| **参数** | pwm\_n: 产生PWM的IO通道号  standard: 0 标准精度； 1 增强精度  prescaler: PWM prescaler  dclsb: PWM Duty Cycle LSB  dcmsb: PWM Duty Cycle MSB |
| **返回值** | void |
| **注解** | 配置PWM参数以产生所需频率和占空比的波形 |

### void test\_pwm(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 测试各个通道的PWM功能 |

# （9）： driver\_saradc.c

工作原理：

1. 打开SARADC时钟

CLK\_EN\_CFG |= 0x80

1. 设置采样channel

adc\_chnn\_en = 1

vdd\_chnn = 0, adc\_chnn =0, gpio3.0作为adc输入的IO

vdd\_chnn = 0, adc\_chnn =1, gpio3.1作为adc输入的IO

vdd\_chnn = 0, adc\_chnn =2, gpio3.2作为adc输入的IO

vdd\_chnn = 0, adc\_chnn =3, gpio3.3作为adc输入的IO

vdd\_chnn = 0, adc\_chnn =4, gpio3.4作为adc输入的IO

vdd\_chnn = 0, adc\_chnn =5, gpio3.5作为adc输入的IO

vdd\_chnn = 0, adc\_chnn =6, gpio3.6作为adc输入的IO

vdd\_chnn = 0, adc\_chnn =7, gpio3.7作为adc输入的IO

1. 设置采样率（连续模式和滤波模式）

(采样率公式)

注：如果是滤波模式，ADC数据速率=Fs/4.

Fadc\_clk= 16M/{adc\_datah[6:4]+1}

4 （送给analog\_adc工作的clk）

(保证adc\_datah[6:4]>=0x01)

1. 打开中断使能
2. 选择需要的模式工作

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 基地址 | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 |
| 0x0922 | adc\_mode | | adc\_chnn | | | intr\_en | adc\_ch\_en | ready |
| adc\_mode:=00, 睡眠模式  adc\_mode:=01, 单步模式  adc\_mode:=10, 软件模式  adc\_mode:=10, 连续模式，如果filt\_mode = 1则是滤波连续模式。  adc\_chnn：adc的八个通道，对应gpio3.0~gpio3.7  intr\_en: SARADC中断使能。（此模块的使能，不是mcu的中断使能）  adc\_ch\_en: gpio作为adc通道使能，如：adc\_ch\_en = 1 且adc\_chnn = 2,则gpio3.2作为adc通道。  ready： 在软件模式和连续模式下，采样一个点后此位被置高，mcu读取数据后（数据的高比特位），此位被拉低。注意：在软件模式下，filt\_mode 应该设置为0 | | | | | | | | |
| 0x0920 | adc\_dataL[7:0] | | | | | | | |
| adc\_data的低8比特 | | | | | | | | |
| 0x0921 | adc\_setting | pre\_divid[2:0] | | | adc\_dataH | | | |
| adc\_setting: saradc设置工作模式后的等待时间，1：40us。 0：20us  pre\_divid[2:0]: 时钟分频数(需要设置成大于等于0x01)  adc\_dataH：adc\_data的高4比特， 对于滤波连续模式，数据是12bit，{adc\_dataH[3:0],adc\_dataL[7:0]},其他模式，数据是10bit, {adc\_dataH[1:0],adc\_dataL[7:0]}. | | | | | | | | |
| 0x0923 | adc\_rate[7:0] | | | | | | | |
| adc\_rate的低8比特 | | | | | | | | |
| 0x0924 | filt\_mode | adc\_dly[1：0] | | adc\_rate[12:8] | | | | |
| filt\_mode=1：连续滤波模式  adc\_dly:saradc setting 时间，可以设置为2或者3  vdd\_chnn: vdd channel 选择  dac\_rate[12:8]: adc\_rate的高5比特。 | | | | | | | | |

### void sleep\_mode(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 模式0：睡眠模式 |

### void single\_mode(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 模式1：单步模式 |

### void single\_mode\_vdd\_channel(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 模式1：睡眠模式，测量vdd channel |

### void soft\_mode(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 模式 2：软件模式 |

### void serial\_mode\_normal(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 模式 3：普通滤波模式 |

### void serial\_mode\_filt(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 模式 3：连续滤波模式 |

### void adc\_int(void) interrupt 14

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 模式 2：软件模式 |

### void test\_saradc(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 测试saradc的各种模式 |

**4：BK2461 收发包API**

# （1）：概述

本文用于描述BK2461的软件开发及相关的API函数，着重讲解收发包部分。有关RF Transceiver的工作原理可参看**错误!未找到引用源。**一节列出的文档，与软件开发相关的其它资料也一并列在其中。

* 1. **程序文件结构**

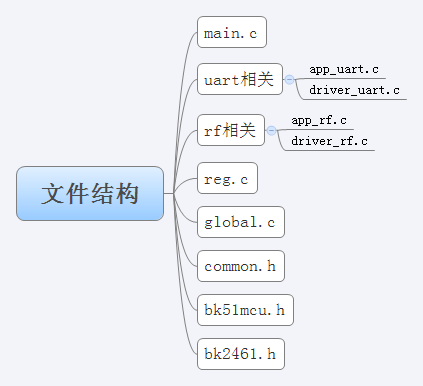


图 4 程序文件结构

程序的文件结构，如上图所示，其作用如下：

|  |  |  |
| --- | --- | --- |
| **文件名** | |  |
| **main.c** | | 轮询接收来自PC端的Uart命令，并执行相应动作 |
| **Uart相关** | **app\_uart.c** | 接收、解析、响应Uart命令 |
| **driver\_uart.c** | 设置Uart参数 |
| **rf 相关** | **app\_rf.c** | 收发包测试函数 |
| **app\_rf.h** | 收发包命令及BK2461寄存器设置 |
| **reg.c** | | 读写寄存器函数 |
| **global.c** | | 定义全局变量 |
| **common.h** | | 定义数据类型 |
| **bk51mcu.h** | | 定义Beken 51 MCU寄存器 |
| **bk2461.h** | | 定义Bk2461寄存器 |

## 1.2 Uart使用概述

Bk2461上电后执行firmware中的函数***UartConfig()***来设置Uart参数，其中为了使用C Lib中的函数***printf()***和***putchar()***，调用了***StdioFuncConfig()***进行设置。接下来在主循环中调用***UartRxCmd()***接收来自PC端的命令，并调用***UartCmdAction()***来解析、响应命令。程序支持的命令有：

|  |  |  |  |
| --- | --- | --- | --- |
| **命令名** | **命令码** | **功能描述** | **响应函数** |
| UART\_WRITE\_SFR | 0x80 | 写SFR寄存器命令 | ***Write\_Sfr()*** |
| UART\_READ\_SFR | 0x81 | 读SFR寄存器命令 | ***Read\_Sfr()*** |
| UART\_WRITE\_DIG\_RF | 0x82 | 写数字寄存器命令 | NA |
| UART\_READ\_DIG\_RF | 0x83 | 读数字寄存器命令 | NA |
| UART\_WRITE\_ANA | 0x84 | 写模拟寄存器命令 | ***Write\_Reg\_Ana()*** |
| UART\_READ\_ANA | 0x85 | 读模拟寄存器命令（实际上不能读模拟寄存器） | NA |
| UART\_WRITE\_DIG\_XRAM | 0x86 | 写数字XRAM中的寄存器 | NA |
| UART\_READ\_DIG\_XRAM | 0x87 | 读数字XRAM中的寄存器 | NA |
| UART\_SLEEP\_INIT | 0x90 | 测试sleep功能 | NA |
| UART\_SLEEP\_ANA | 0x91 |
| UART\_SLEEP | 0x92 |
| UART\_TEST\_TRANS | 0xF1 | 发送数据包 | ***TransTest()*** |
| UART\_TEST\_RECV | 0xF2 | 接收数据包 | ***RecvTest()*** |

\*NA: 无响应函数

## 1.3 收发包功能概述

BK2461上电后执行函数***Rf\_Init()***，设置寄存器***FEATURE***=0x07，启用3个功能：Dynamic Payload Length；Ack with Payload；及命令***W\_TX\_PAYLOAD\_NOACK***。同时设置***EN\_AA***=0x3F，启用6个Pipe的Auto Ack功能，本文中提及的ack即为硬件的auto ack功能。

启用了这些功能，并不一定要去使用。如启用了auto ack功能，收方也可以不回ack，这是由发方在发送payload数据时所使用的命令决定：若用命令***W\_TX\_PAYLOAD***发送，收方收到数据后需回ack；若用命令***W\_TX\_PAYLOAD\_NOACK***发送，收方则不需回ack。至于选用哪一个命令发送，可在PC端软件中指定。两个命令的区别是：用后者发完数据后，发方的中断状态位***TX\_DS***置1；而用前者发完数据后，发方需要收到收方回的ack后才会把***TX\_DS***位置1。

同样启用了Dynamic Payload Length功能，发方也可以发送固定长度的payload，要想使用变长功能发送，还需在PC端软件中进一步指定；启用了ack with payload功能，收方也可以只回ack而不带payload，要想使用ack with payload也需在PC软件中进一步指定。

初始化过后，BK2461要想进行收发工作，还需在PC端软件中设置一下收发参数，包括传输速率、频点、频偏N值及Kmod校准，之后才可以进行收发工作。若让其发送则调用函数***TransTest()***，若接收则调用***RecvTest()***, 若停止收发则调用***StopTest()***。程序中收发包函数的结构如下两图所示：

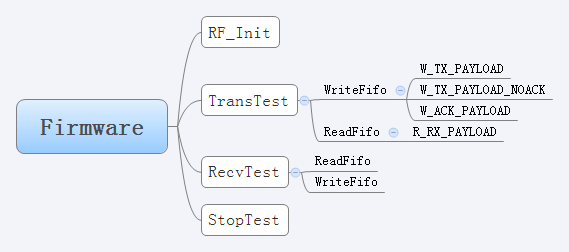


图 5 Firmware中收发包函数的结构

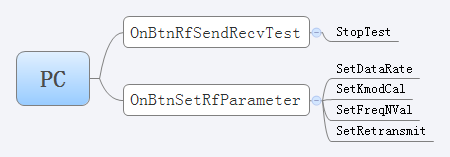


图 6 PC端收发包函数的结构

PC端需要传给BK2461的信息，及PC与firmware中的相关函数如下所述。

# （2）： Firmware中收发包相关全局变量说明

## 2.1 发方需从PC界面中传进的参数

|  |  |  |
| --- | --- | --- |
| **变量名称** | **类型** | **作用** |
| **ptx** | bit | 1：发方  0：收方 |
| **send\_need\_ack** | bit | 1：收方需回ack，相应发送命令为***W\_TX\_PAYLOAD***  0：收方不用回ack，相应命令为***W\_TX\_PAYLOAD\_NOACK*** |
| **uart\_command[5]** | uint8 | 发送两包的时间间隔，单位毫秒 |
| **send\_times** | uint8 | 发送次数 |
| **loop\_send** | bit | 1：发送完指定次数后，再次循环发送  0：发送完指定次数后，跳出***TransTest()***，结束发送 |
| **send\_to\_pipe** | uint8 | 指定收方的接收pipe，范围0~5 |
| **loop\_pipes** | uint8 | 每发送完20次后，切换到下一个pipe |
| **payload\_len** | uint8 | 发送包中payload的长度 |
| **dynamic\_payload** | bit | 1：每次发包的payload长度比上一次多1  0：按指定的payload长度发包 |

## 2.2 收方需从PC界面中传进的参数

|  |  |  |
| --- | --- | --- |
| **变量名称** | **类型** | **作用** |
| **ptx** | bit | 1：发方  0：收方 |
| **ack\_payload** | bit | 1：收方回带payload的ack  0：收方只回ack  收方在收到数据后，会把中断状态位***TX\_DR***置1，同时回ack给发方。若在回ack时收方的TX FIFO不空，则会把其中的数据一道ack回去，实现ack with payload；若TX FIFO为空，则只回ack。收方向TX FIFO中写数据是用命令***W\_ACK\_PAYLOAD***实现的。  程序中是把收到数据作为payload，ack回去的，当然也可ack回任意数据。在第一次收到数据时，TX FIFO是空的，只会回ack，接着用***ReadFifo()***读出收到的数据，再用***WriteFifo()***把数据写到TX FIFO中；第二次收到数据时，就会把上一次收到的数据ack回去；以后情况类似，这样最后一次写入TX FIFO中的数据就没机会被ack出去，若是进行loop\_pipes测试，切换pipe三次后TX FIFO就满了，需进行***FLUSH\_TX***清空操作。 |

## 2.3 收发包中用到的其它变量

|  |  |  |
| --- | --- | --- |
| **变量名称** | **类型** | **作用** |
| **SendTimes[6]** | int16 | 发往各个pipe的次数 |
| **RecvTimes[6]** | int16 | 各个pipe接收到数据的次数 |
| **TxDataCounter[6]** | uint16 | 发往各个pipe的数据，每次加1，等同于发送次数 |
| **RxDataCounter[6]** | uint16 | 各个pipe应该收到的数据值，每次加1，等同于接收次数。一发一收的次数应该是匹配的，可用此特点判断接收到的数据是否正确 |
| **Fifo\_Data[32]** | uint8 | 用于存放发送或接收的payload数据。硬件中的FIFO分两个，TX FIFO和RX FIFO，但地址相同。 |
| **err\_count** | int16 | 用于统计接收到数据的出错次数。错误包括两类：接收到的payload数据值不对；接收到数据中的pipe信息与实际的接收pipe不一致 |

# （3）： Firmware 中API说明

## 3.1 app\_rf.c中的函数

### bit StopTest(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | 收到停止命令则返回TRUE；否则返回FALSE |
| **注解** | 每一次收发结束后，BK2461都要执行此函数，检测PC端有无停止指令发送过来。PC端有同名函数与此对应。 |

### void TransTest(int interval\_time)

|  |  |
| --- | --- |
| **参数** | interval\_time [in] 发送两包的时间间隔，单位毫秒。 |
| **返回值** | void |
| **注解** | BK2461作为发方，调用此函数发送数据。此函数会一直运行，直到发送完毕，或者收到PC端的结束指令。 |

### void RecvTest(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | BK2461作为收方，调用此函数接收数据。此函数会一直运行，直到收到PC端的结束指令。 |

### void WriteFifo(uint8 bytes, uint8 pipe)

|  |  |
| --- | --- |
| **参数** | bytes [in] 要发送的payload数据长度  pipe [in] 要发向的收方pipe |
| **返回值** | void |
| **注解** | 把要发送的payload数据写入TX FIFO中。有三个命令会把数据写入TX FIFO中：发方的W\_TX\_PAYLOAD()和W\_TX\_PAYLOAD\_NOACK；收方的W\_ACK\_PAYLOAD。发方发送数据和收方回ack with payload时都会调用此函数。 |

### bit ReadFifo(uint8 bytes, uint8 pipe)

|  |  |
| --- | --- |
| **参数** | bytes [in] 接收到的payload数据长度  pipe [in] 接收到数据的pipe |
| **返回值** | 若接收到的数据正确，则返回TRUE；否则返回FALSE |
| **注解** | 把接收到的payload数据从RX FIFO中读出。读数据用的命令是R\_RX\_PAYLOAD。  收方收到数据和发方接收到ack with payload时都会调用此函数。 |

## 3.2 driver\_rf.c中的函数

### void RF\_Init(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | BK2461上电后要执行的初始化设置 |

### void PowerUp\_Rf(void)

### void PowerDown\_Rf(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 设定BK2461为Power Up或Power Down状态 |

### void SwitchToTxMode(void)

### void SwitchToRxMode(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 设定BK2461为PTX或PRX模式 |

### void ChangeTxPipe(uint8 pipe)

|  |  |
| --- | --- |
| **参数** | pipe [in] 要发向的收方pipe |
| **返回值** | void |
| **注解** | BK2461作为收方6个pipe都可接收，其地址各不相同，发方的发送地址要按照收方接收pipe的地址设置。发方只有pipe 0用于发送，但也要确保6个pipe地址各不相同；pipe 0还用于接收收方回的ack，故其地址要与发送地址相同。 |

### 4个收发命令函数

#### void R\_RX\_PAYLOAD(uint8 \*pBuf, uint8 bytes)

|  |  |
| --- | --- |
| **参数** | pBuf [out] 存放从RX FIFO中读出的数据  byte [in] 读取数据的长度 |
| **返回值** | void |
| **注解** | 收方或发方都可利用此函读取RX FIFO中接收到的payload数据。 |

#### void W\_TX\_PAYLOAD(uint8 \*pBuf, uint8 bytes)

|  |  |
| --- | --- |
| **参数** | pBuf [in] 要写入TX FIFO中的数据  byte [in] 写入数据的长度 |
| **返回值** | void |
| **注解** | 发方利用此函把payload数据写入TX FIFO中，需要收方回ack |

#### void W\_TX\_PAYLOAD\_NOACK(uint8 \*pBuf, uint8 bytes)

|  |  |
| --- | --- |
| **参数** | pBuf [in] 要写入TX FIFO中的数据  byte [in] 写入数据的长度 |
| **返回值** | void |
| **注解** | 发方发方利用此函把payload数据写入TX FIFO中，不需要收方回ack |

#### void W\_ACK\_PAYLOAD(uint8 \*pBuf, uint8 bytes, uint8 pipe)

|  |  |
| --- | --- |
| **参数** | pBuf [in] 要写入TX FIFO中的数据  byte [in] 写入数据的长度 |
| **返回值** | void |
| **注解** | 收方利用此函把payload数据写入TX FIFO中。收方在需回ack时，会把此中数据也ack出去，实现ack with payload功能。 |

## 3.3 app\_uart.c中的函数

### void UartRxCmd(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 接收来自PC端的Uart命令 |

### void UartCmdAction(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 解析收到的Uart命令，并调用响应函数。 |

### void Delay\_ms(int ms)

|  |  |
| --- | --- |
| **参数** | ms [in] 延时的毫秒数 |
| **返回值** | void |
| **注解** | 不精确的毫秒级延时。 |

## 3.4 driver\_uart.c中的函数

### void UartConfig(uint16 baud)

|  |  |
| --- | --- |
| **参数** | baud [in] 波特率 |
| **返回值** | void |
| **注解** | 设置Uart参数。 |

### void StdioFuncConfig(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 配置Uart参数，使程序能够使用C语言库中的printf(), putchar()函数。 |

## 3.5 reg.c中的函数

### uint8 Read\_SFR(uint8 addr)

|  |  |
| --- | --- |
| **参数** | addr [in] SFR寄存器地址 |
| **返回值** | 返回寄存器的值 |
| **注解** | 读SFR寄存器的值。 |

### void Write\_SFR(uint8 addr, uint8 val)

|  |  |
| --- | --- |
| **参数** | addr [in] SFR寄存器地址  val [in] 要写入寄存器的值 |
| **返回值** | void |
| **注解** | 写SFR寄存器的值。 |

### void Write\_Reg\_Ana(uint8 addr, uint32 val)

|  |  |
| --- | --- |
| **参数** | addr [in] 模拟寄存器地址  val [in] 要写入寄存器的值 |
| **返回值** | void |
| **注解** | 写模拟寄存器的值。 |

# （4）： PC端API说明

PC端与收发包相关的函数在MyPropertyPageTrx.CPP中

## UART指令格式

|  |  |  |  |
| --- | --- | --- | --- |
| **字节** | **发送** | **接收** | **停止** |
| 0 | 0x55 | 0x55 | 0x55 |
| 1 | 0xF1 | 0xF2 | 0xFF |
| 2 | 1: 需要ack  0: 不需ack | NA | NA |
| 3 | bit 1：loop pipes  bit 2：dynamic len  bit 4：loop send  bit 5~7：send to pipe  其它bit：NA | bit 0：ack payload  其它bit：NA | NA |
| 4 | payload length | NA | NA |
| 5 | 发送两包的间隔时间 | NA | NA |
| 6 | 发送次数 | NA | NA |
| 7 | NA | NA | NA |

\*NA：无效字节或bit，此数据不会被用到

## void OnBtnRfSendRecvTest(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 用于给BK2461下达指令：发送、接收或停止。 |

## void StopTest(void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 停止收发指令。Firmware中有相应的同名函数 |

## void SetDataRate(int iRate)

|  |  |
| --- | --- |
| **参数** | iRate [in] 0: 250K; 1: 1M; 2: 2M |
| **返回值** | void |
| **注解** | 设置传输速率 |

## void SetKmodCal(int iRate)

|  |  |
| --- | --- |
| **参数** | iRate [in] 0: 250K; 1: 1M; 2: 2M |
| **返回值** | void |
| **注解** | 根据不同的传输速率，校正调制深度。 |

## void SetFreqNVal(BYTE byFreq)

|  |  |
| --- | --- |
| **参数** | byFreq [in] 频点，单位MHz |
| **返回值** | void |
| **注解** | 设置频点和频偏N值 |

## void SetRetransmit (BYTE byARD, BYTE byARC)

|  |  |
| --- | --- |
| **参数** | byARD [in] 重发等待时间  byARC [in] 重发次数 |
| **返回值** | void |
| **注解** | 设置重发参数。在需要回ack的情况下，若发方在ARD单位时间内没有收到ack，则会重发此包；重发ARC次后仍没有收到ack，就丢失此包。 |

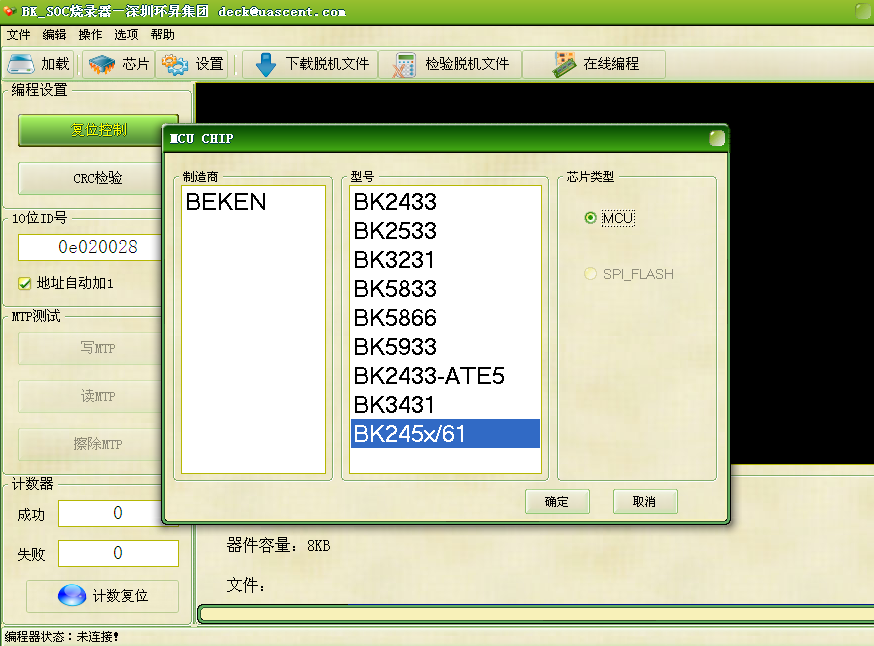
## void OnBtnSetRfParameter (void)

|  |  |
| --- | --- |
| **参数** | void |
| **返回值** | void |
| **注解** | 调用以上四个函数来设置BK2461的收发参数。 |

**附录1：带滚码的脱机烧录**

将BIN文件DownLoad到烧录器中，可以脱机带滚码烧录IC。

1. 打开sdiapp.exe软件，选择BK245X/2461选项
2. 加载不带CRC的bin文件，填写10位ID码，地址自动加1选项 打钩
3. 下载脱机文件到烧录器。



**附录2：不同功率SPI配置**





**附录3：精彩问答**

1：怎样在开发板上需要使用烧录脚作为正常的io口应用？

解答： 按照正常方式将开发板烧录程序（目前是将程序烧写到了flash中），然后让开发板正常工作，将VPP管脚接到VDD上边，此时 烧录脚即可作为正常io口使用。