

扫地机开发指南

最新版本：1.0

# 目录

[前言 1](#_Toc12948)

[开发相关 2](#_Toc14777)

[1.1相关术语表 2](#_Toc4367)

[1.2量产工具 3](#_Toc8364)

[1.3测试模式 3](#_Toc23852)

[基本介绍 4](#_Toc8468)

[适配步骤 6](#_Toc30422)

[3.1 步骤一 电池和充电功能参数修改 6](#_Toc10261)

[3.1.1 与电池相关的代码 6](#_Toc30712)

[3.1.2 充电参数 6](#_Toc11229)

[3.1.3 充电配置 6](#_Toc31285)

[3.1.4 运行时的相关参数 8](#_Toc4868)

[3.1.5 适配测试 8](#_Toc14141)

[3.2 步骤二 电机驱动芯片、定时器配置 9](#_Toc20497)

[3.2.1 电机配置相关代码 9](#_Toc16584)

[3.2.2 电机部分配置 9](#_Toc6652)

[3.2.3 电机应用场景开关 10](#_Toc13704)

[3.2.4 异常检测 11](#_Toc12769)

[3.2.5 适配测试 11](#_Toc17038)

[3.3 步骤三 light touch通道修改和使能 12](#_Toc26442)

[3.3.1 与light touch有关的代码 12](#_Toc31993)

[3.3.2 light touch分布方位图 13](#_Toc22451)

[3.3.3 实现原理 13](#_Toc15842)

[3.3.4 adc通道修改 14](#_Toc2864)

[3.3.5 红外发射管使能管脚修改 14](#_Toc15074)

[3.3.6 物理通道共用 14](#_Toc24426)

[3.3.7 适配测试 14](#_Toc10157)

[3.3.8 阈值设定 15](#_Toc31555)

[3.4 步骤四 ciff通道修改和使能 16](#_Toc20881)

[3.4.1 与cliff相关的代码 16](#_Toc6375)

[3.4.2 cliff分布方位图 16](#_Toc27609)

[3.4.3 实现原理 17](#_Toc8466)

[3.4.4 adc通道修改 17](#_Toc15465)

[3.4.5 红外发射管使能管脚修改 17](#_Toc10703)

[3.4.6 物理通道共用 17](#_Toc4617)

[3.4.7 适配测试 17](#_Toc15368)

[3.4.8 阈值相关 18](#_Toc31288)

[3.5 步骤五 bump管脚、触发电平修改 18](#_Toc7823)

[3.5.1 bump相关的代码 18](#_Toc6791)

[3.5.2 bump分布方位图 18](#_Toc6804)

[3.5.3 bump原理 19](#_Toc29124)

[3.5.4 管脚及触发电平修改 19](#_Toc3230)

[3.5.5 适配测试 19](#_Toc10109)

[3.6 步骤六 轮子抬起检测管脚、触发电平修改 20](#_Toc28294)

[注意：由于该功能部分机型没有，所以要先确认适配机型是否有该功能。 20](#_Toc1191)

[3.6.1 轮子抬起检测相关的代码 20](#_Toc10754)

[3.6.2 轮子抬起检测原理 20](#_Toc18025)

[3.6.3 管脚及触发电平修改 20](#_Toc16573)

[3.6.4 适配测试 20](#_Toc17501)

[3.7 步骤七 其他的一些物理结构 21](#_Toc32278)

[3.7.1 相关代码 21](#_Toc28107)

[3.7.2 参数配置 21](#_Toc18431)

[3.8 步骤八 IR接收管脚修改和解码 22](#_Toc32156)

[3.8.1 与ir有关的代码 22](#_Toc3978)

[3.8.2 红外接收头方位分布图 22](#_Toc800)

[3.8.3 ir的作用 23](#_Toc10024)

[3.8.4 接收管脚修改 23](#_Toc3555)

[3.8.5 增加接收头 24](#_Toc13227)

[3.8.6 红外解码 24](#_Toc32020)

[3.8.7 适配测试 24](#_Toc22151)

# 

# 前言

在如今扫地机器人发展如此迅猛的时代，产品频频更新换代。为了降低扫地机器人的开发门槛，本公司推出了一套专门针对智能扫地机器人开发的SDK套件。开发者使用我们APP系统进行开发时，需要针对不同的方案做不同的修改。这其中必然涉及到对某些外设的重新配置。本文档主要讲解APP系统中已经开放的外设该如何去配置与如何利用SDK套件快速了解并上手app系统，对产品进行适配，减少产品开发过程中不必要的麻烦。

# 开发相关

## 1.1相关术语表

|  |  |
| --- | --- |
| **缩写和术语** | **解 释** |
| DOCK | 回座充电（简称：自座） |
| ir | 红外 |
| cliff | 悬崖检测（简称：地检） |
| light touch | 靠墙检测、障碍检测（简称：墙检） |
| lt | 即light touch |
| bump | 碰撞检测 |
| lt bump | 用lt实现类似bump的功能 |
| GPTM | 通用定时器 |
| MCTM | 高级定时器 |
| force\_field | 圆泡或圆泡信号 |
| buoy\_left | 左目，充电座发射的左信号 |
| buoy\_right | 右目，充电座发射的右信号 |
| remote | 遥控器 |
| bat | 即battery，电池。但在例程中有多次表示：电池电压 |
| Debouncer | 对信号捕捉和延时 |
| JACK | 充电适配器 |
| TEMP | 温度 |
| UI | APP系统UI |
| vaccum | 真空、吸尘 |
| GYRO | 陀螺仪 |
| WALLFLOW | 沿边清扫 |
| STASIS | 万向轮 |

## 1.2量产工具

量产工具要具备分别升级完整固件，升级APP程序，升级config区的功能。（具体参考《量产工具使用手册》）

量产工具

Config

Bootloader

SDK主程序

APP程序

使用步骤如下：

1. 使用iar编译项目文件；

## 1.3测试模式

测试模式是一个非常重要的功能，它可以帮助开发者在开发的过程中测试相应部分的功能是否已经正常工作。也可以在机器工作不正常时进入相关功能的测试，进行故障排除。在后面适配的步骤中会经常使用到测试模式。机器长按启动/暂停键大约5秒就可以进入测试模式，并开始第一项项目测试，每短按一下按键跳转下一个项目测试。到最后一个项目测试后再继续短按又回到第一个项目测试依此循环。测试模式的代码是test\_cmd\_handle.c中handle\_test\_cmd()函数，这个函数包括了所有项目的测试。测试项目是test\_item\_table[]数组中的内容，默认最大15项，数组内容的顺序就是测试项目的顺序。所有的测试项目定义在ui\_commands.h的枚举变量TEST\_CMD\_E中，把TEST\_CMD\_E中的成员放入test\_item\_table[]数组就可以进行相应的测试。如下面代码

|  |
| --- |
| TEST\_CMD\_E test\_item\_table[UI\_TEST\_ITEM\_MAX]={  CMD\_BAT,  CMD\_TEST\_CLIFF\_LIGHTTOUCH,  CMD\_TEST\_BUMP\_DROP,  CMD\_BAT,  CMD\_SIDE\_FORWARD,  CMD\_WHEEL\_L\_FORWARD,  CMD\_WHEEL\_R\_FORWARD,  CMD\_WHEEL\_L\_REV\_SPD,  CMD\_WHEEL\_R\_REV\_SPD,  CMD\_MAIN\_FORWARD,  CMD\_MAIN\_REV,  CMD\_SIDE\_FORWARD,  CMD\_SIDE\_REV,  CMD\_DOCK\_CURRENT,  CMD\_VACCUM}; |

# 基本介绍

扫地机基本功能是由电机、light touch、cliff、bump和ir这几部分组成。电机有主刷电机、边刷电机、真空吸尘电机和轮子电机；linght touch是墙检，主要是实现避开障碍物（具体原理可参考3.3.3）；cliff是地检，主要是实现躲避悬崖（具体原理课看3.4.3）；bump是碰撞检测，light touch无法检测到的障碍物由bump进行检测，也是实现躲避障碍物（具体原理可看3.5.3）；ir是实现红外信号接收。下图是某款不带主刷扫地机器人的外形图



启动/暂停按键

充电座接触金属片

万向轮



左轮

右轮

真空入风口

吸尘出风口

右边刷

左边刷

带主刷机型



真空入风口

主刷

## 

# 适配步骤

## 3.1 步骤一 电池和充电功能参数修改

### 3.1.1 与电池相关的代码

|  |  |  |
| --- | --- | --- |
| 文件名 | 所在文件夹 | 作用 |
| charge.c | charge | 充电流程控制 |
| robot-battery-monitor.c | ui-manager | 检测充电的一些状态，并执行相关动作。 |
| robot-battery.c | ui-manager | 充电初始化，检测充电的一些状态，并执行相关动作。 |
| charge.h | src/charge | 电池与充电相关系数 |
| adc\_chan.h | src/cfg | adc通道定义 |

### **3.1.2 充电参数**

要对充电器的一些参数进行修改打开charge.h找到如下代码。在如下代码把BATTERY\_LOW\_SLOWVECTORY 和CHARGING\_CUTOFF\_VOLTAGE这两个电压参数配置正确后，若是需要先把机器适配得能跑起来，下面参数适配部分可以先跳过，进入下一节的适配。若是需要使机器能正常充电，下面参数的适配要正确。

|  |
| --- |
| #define CHARGING\_MAXVOL (3628)//16.8V  #define CHARGING\_RECOVERYVOL 2591 //12v  #define CHARGING\_AFTERFULL\_MINVOL 3563 //16.5V  #define CHARGING\_LOW\_VOLTAGE 2851 //13.2V  #define CHARGING\_CUTOFF\_VOLTAGE 2634 //2591 //12.2v  #define BATTERY\_LOW\_SLOWVECTORY 2268 //13.5v |

这些数据都是adc值，转换成电压的公式是((18975\*v\_adc)/4096)mv，其中这几个参数意义如下：

|  |  |
| --- | --- |
| CHARGING\_MAXVOL | 电池充电达到的最大电压； |
| CHARGING\_RECOVERYVOL | 低于这个电压就涓流充电； |
| CHARGING\_AFTERFULL\_MINVO | 充电完成后，插着充电器，低于这个电压就继续充电； |
| CHARGING\_LOW\_VOLTAGE | 低于这个电压，停止清扫回座充电； |
| CHARGING\_CUTOFF\_VOLTAGE | 低于这个电压，关闭DC-DC； |
| BATTERY\_LOW\_SLOWVECTORY | 低于这个电压，降速行走。 |

这些是对应充电的一些参数需要针对机器进行修改，

在am\_config\_macro.h中

|  |
| --- |
| ADAPTER\_OUTPUT\_VOL = 22, //适配器电压（V）  ADAPTER\_OUTPUT\_CURRENT = 800, //适配器电流（mA） |

是适配器有关的参数。下面是电池最大电压

|  |
| --- |
| BAT\_VOL\_MAX = 3628, //电池在16.8V时对应的ADC |

### **3.1.3 充电配置**

cpu对于充电控制和检测的一些管脚要进行宏定义修改和管脚初始化。这些管脚如下表

|  |  |  |
| --- | --- | --- |
| 管脚宏定义 | 所在文件 | 作用 |
| IO\_VCC\_KILL | hal\_amicro\_gpio.h | 使能DC-DC |
| IO\_CHARGER\_HIGH | hal\_amicro\_gpio.h | 适配器插入检测 |
| IO\_CHARGER\_LOW | hal\_amicro\_gpio.h | 充电座检测 |
| AM\_IO\_SWITCHED\_PWR\_5V | hal\_amicro\_gpio.h | 使能5V DC |
| AM\_IO\_SWITCHED\_PWR\_3V | hal\_amicro\_gpio.h | 使能3.3V DC |
| AM\_IO\_WATCHDOG\_CLEAR | hal\_amicro\_gpio.h | 充电看门狗（防止程序跑飞还在充电） |

首先要对应机器修改宏定义对应的管脚，然后修改管脚的配置。对于GPIO管脚的修改和配置可参考《SDK-APP设计说明》中GPIO部分。

在adc\_chan.h中找到下面代码

|  |
| --- |
| m(ADC\_CHAN\_BATTERY\_THERMISTOR ,13 )\  m(ADC\_CHAN\_BATTERY\_CURRENT ,14 )\  m(ADC\_CHAN\_BATTERY\_VOLTAGE ,15 )\ |

这三个分别是电池热敏电阻电压adc获取通道，电池电流adc获取通道，电池电压adc获取通道，要修改好相应adc通道，进入adc\_chan.h找到如上这几个通道参数，对应机器通道修改即可，排列的顺序不能改，只能修改后面的数字，数字x即对应通道ADCx。

此外，还有充电电压的PWM控制，打开tm\_cfg.c找到如下代码

|  |
| --- |
| charge\_pwm\_id = open(DEV\_MCTM,MC\_TM2\_ID);//tm\_request(MCTM,MC\_TM2\_ID);  mctmInit.CounterReload = 720;  mctmInit.Prescaler = 0;  mctmInit.CounterMode = MCTM\_CNT\_MODE\_UP;  mctmInit.PSCReloadTime = MCTM\_PSC\_RLD\_IMMEDIATE;  ioctl(charge\_pwm\_id, TM\_MCTM\_BASE\_INIT, &mctmInit);  MUserOutInit.OutInit.OutputMode = MCTM\_OM\_PWM2;  MUserOutInit.OutInit.Control = MCTM\_CHCTL\_ENABLE;  MUserOutInit.OutInit.ControlN = MCTM\_CHCTL\_ENABLE;  MUserOutInit.OutInit.Polarity = MCTM\_CHP\_NONINVERTED;  MUserOutInit.OutInit.PolarityN = MCTM\_CHP\_NONINVERTED;  MUserOutInit.OutInit.IdleState = MCTM\_OIS\_LOW;  MUserOutInit.OutInit.IdleStateN = MCTM\_OIS\_HIGH;  MUserOutInit.OutInit.Compare = 0;  MUserOutInit.OutMode.irq = 0;  MUserOutInit.OutMode.dmaReqMode = 0;  MUserOutInit.OutInit.Channel = (MCTM\_CH\_Enum)1;  ioctl(charge\_pwm\_id, TM\_MCTM\_OUT\_MODE\_INIT, &MUserOutInit);  request\_irq(charge\_pwm\_id,(long)charger\_pwm\_handler,0); |

注意定时器与申请ID的对应关系

|  |  |  |
| --- | --- | --- |
| 定时器 | 申请时ID | 在MCU名称 |
| MCTM0 | MC\_TM1\_ID | MT0 |
| MCTM1 | MC\_TM2\_ID | MT1 |

定时器主频为72M。故这一路PWM输出频率为72M/((720+1)\*(0+1))=99.861 KHz。配置好定时器后，还需要对sys\_set\_pwm\_out\_duty()这个函数进行修改，在tm\_cfg.c中找到sys\_set\_pwm\_out\_duty()函数。对应定时器的配置将配置的参数填入函数中。例如下面代码表示充电控制PWM通道，将PWM通道按照格式加入，charge\_pwm\_id就是用open()函数打开设备时返回的设备码，GPTM\_CHAN1\_SET\_DUTY是定时器通道1 。

|  |
| --- |
| case CHARGER\_PWM\_CHANNEL:  id = charge\_pwm\_id;  ioctl(id,MCTM\_CHAN1\_SET\_DUTY,(void \*)&Dutyfactor);  break; |

其中我们只需要关心如下四个参数

|  |  |
| --- | --- |
| SIDE\_BRUSH\_PWM\_CHANNEL | 边刷电机PWM通道 |
| MAIN\_BRUSH\_PWM\_CHANNEL | 主刷电机PWM通道 |
| VACUUM\_PWM\_CHANNEL | 真空电机PWM通道 |
| CHARGER\_PWM\_CHANNEL | 充电控制PWM通道 |

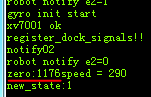
上面的代码配置中

|  |
| --- |
| request\_irq(charge\_pwm\_id,(long)charger\_pwm\_handler,0); |

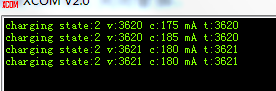
是申请了一个中断， 所有定时器都可以触发中断， 但是这个中断的产生有可能会对系统产生一定的影响，所以这个功能需要斟酌使用。

### **3.1.4 运行时的相关参数**

充电器和电池适配完成后，打开电源，系统启动过程中会打印一些信息，找到如下打印



zero是基准电压的值，应该在1200左右。插上适配器等待10秒左右，应该会有如下打印



charging state是充电状态，充电状态在charge.h中CHARGING\_STATE\_E结构体定义，v表示电池电压，c表示充电电流，t表示电池温度（目前由于大部分方案没有电池温度检测，所以当前值是电池电压值）。如果需要添加电池温度，打开current.c找到如下

|  |
| --- |
| tmp16 = get\_adc\_chan\_val(ADC\_CHAN\_BATTERY\_VOLTAGE);  battery\_temperature = tmp16/10 + last\_battery\_temperature\*9/10;  last\_battery\_temperature = battery\_temperature; |

将ADC\_CHAN\_BATTERY\_VOLTAGE改为电池热敏电阻电压ADC\_CHAN\_BATTERY\_THERMISTOR，并且要修改好相应adc通道，进入adc\_chan.h找到ADC\_CHAN\_BATTERY\_THERMISTOR，对应机器通道修改即可，排列的顺序不能改，只能修改后面的数字，数字x即对应通道ADCx。并将tmp16/10 + last\_battery\_temperature\*9/10这一句替换成相应温度转换公式

### **3.1.5 适配测试**

修改测试模式数组第一项为CMD\_BAT，代码如下

|  |
| --- |
| TEST\_CMD\_E test\_item\_table[UI\_TEST\_ITEM\_MAX]={  CMD\_BAT, //电池测试  CMD\_TEST\_CLIFF\_LIGHTTOUCH,  CMD\_TEST\_BUMP\_DROP,  CMD\_SIDE\_FORWARD,  CMD\_WHEEL\_L\_FORWARD, CMD\_WHEEL\_R\_FORWARD,  CMD\_WHEEL\_L\_REV\_SPD, CMD\_WHEEL\_R\_REV\_SPD,  CMD\_VACCUM,  CMD\_DOCK\_CURRENT, CMD\_MAIN\_REV,  CMD\_MAIN\_FORWARD,  CMD\_SIDE\_FORWARD, CMD\_SIDE\_REV,  CMD\_VACCUM}; |

下载后上电等待系统启动完成后，长按启动/暂停按键进入测试模式第一项，会有如下打印，这是电池电压的adc数据



## 3.2 步骤二 电机驱动芯片、定时器配置

### 3.2.1 电机配置相关代码

|  |  |  |
| --- | --- | --- |
| 文件名 | 所在文件夹 | 作用 |
| tm\_cfg.c | src/cfg | 定时器配置，PWM更新。 |
| ui\_config.c | src/ui-manager | ui配置 |
| exception.c | src/ui-manager | 异常检测 |

### 3.2.2 电机部分配置

对于电机部分APP中开放了四个类型的电机的配置，分别是主刷电机、边刷电机、真空吸尘器电机和轮子电机。轮子电机没有开放定时器配置，只需要把驱动芯片选择好。打开ui\_config.c，找到这一句代码

|  |
| --- |
| .clean\_fun.bits.motor\_driver = 0, |

这句代码是对轮子电机驱动芯片的选择，电机驱动芯片类型0:8843 1:8870。

|  |
| --- |
| .clean\_fun.bits.wheel\_left\_dir = 0, //左轮旋转方向  .clean\_fun.bits.wheel\_right\_dir= 0, //右轮旋转方向 |

这一句是轮子转动方向的修改，当轮子转动方向不对时，可以修改这里。

其他电机需要配置相应的定时器，定时器的配置可见APP系统中的tm\_cfg.c文件。

当前tm\_cfg.c文件定义的硬件设备与定时器通道对应的关系如下表

|  |  |  |
| --- | --- | --- |
| 硬件设备 | 定时器 | 通道 |
| 主刷电机 | GPTM1 | 3 |
| 边刷电机 | GPTM2 | 2 |
| 真空电机 | GPTM0 | 0 |
| 充电控制 | MCTM2 | 1 |

|  |  |  |
| --- | --- | --- |
| 定时器 | 申请时ID | 在MCU名称 |
| GPTM0 | GP\_TM0\_ID | T0 |
| GPTM1 | GP\_TM1\_ID | T1 |
| GPTM2 | GP\_TM2\_ID | T2 |

如下代码是某一款机型的主刷和边刷的PWM通道配置

|  |
| --- |
| gtm1\_id = open(DEV\_GPTM,GP\_TM1\_ID);  gftmInit.CounterReload = 1000;  gftmInit.Prescaler = 5; //lqw  gftmInit.CounterMode = GPTM\_CNT\_MODE\_UP;  gftmInit.PSCReloadTime = GPTM\_PSC\_RLD\_IMMEDIATE;ioctl(gtm1\_id, TM\_GPTM\_BASE\_INIT, &gftmInit);  UserOutInit.OutInit.OutputMode = GPTM\_OM\_PWM1;  UserOutInit.OutInit.Control = GPTM\_CHCTL\_ENABLE; UserOutInit.OutInit.Polarity = GPTM\_CHP\_NONINVERTED; UserOutInit.OutInit.Compare = 0; //占空比  UserOutInit.OutInit.AsymmetricCompare = 0;  UserOutInit.OutMode.irq = 0;  UserOutInit.OutMode.dmaReqMode = 0;//dma request  for(int i = 2; i<4; i++)  {  UserOutInit.OutInit.Channel = (GPTM\_CH\_Enum)i;  Ioctl(gtm1\_id,TM\_GPTM\_OUT\_MODE\_INIT, &UserOutInit);  } |

GP\_TM1\_ID表示这个定时器是GPTM1。(GPTM\_CH\_Enum)i，中的i取值2和3，表示这两个类型的电机用的是GPTM1的通道2和通道3 。然后需要对应修改sys\_set\_pwm\_out\_duty()函数，这里可以参考电池部分1.1.3节对sys\_set\_pwm\_out\_duty()函数的修改介绍。

### 3.2.3 电机应用场景开关

而对于电机在某些场合的控制可以打开ui\_config.c找到如下代码

|  |
| --- |
| .clean\_motor\_config =  {  .clean\_fun.bits.main\_brush\_clean = 0,  .clean\_fun.bits.main\_brush\_dock = 0,  .clean\_fun.bits.side\_brush\_clean = 1,  .clean\_fun.bits.side\_brush\_dock = 1,  .clean\_fun.bits.vaccum\_clean = 1,  .clean\_fun.bits.vaccum\_dock = 0,  .clean\_fun.bits.water\_box = 0,  .clean\_fun.bits.resume = 0,  }, |

这部分代码就是对于清扫和上座相应电机的打开和关闭，其中0是关闭，1是打开。这几个功能介绍如下：

|  |  |
| --- | --- |
| main\_brush\_clean | 清扫时主刷电机的开关 |
| main\_brush\_dock | 回座充电时主刷电机的开关 |
| side\_brush\_clean | 清扫时边刷电机的开关 |
| side\_brush\_dock | 回座充电时边刷电机的开关 |
| vaccum\_clean | 清扫时真空电机的开关 |
| vaccum\_dock | 回座充电时真空电机的开关 |
| water\_box | 水箱开关 |
| resume | 断点续扫功能 |

### 3.2.4 异常检测

在exception.c中对于电机会有如下异常检测

1 主边刷被缠住、真空被堵住

这三个检测原理都是一样的，检测到电机电流超过电流设定值就开始计数，连续计数超过计数设定值就返回相应电机的异常给ui，并通过语音播报相应电机的异常。

2 垃圾盒没有装入

如果真空电机已经设定了相应的电压adc值，但是真空电机电流小于设定值，开始计数，连续计数超过设定的计数值，就返回相应的异常给ui，并通过语音播报该异常。

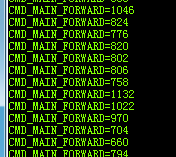
出现上述的异常，主要有两个原因，一个是电机电流采样电路相应的放大倍数不对。另外一个是电机相应的设定值不对。

### 3.2.5 适配测试

上述参数都配置好之后需要进测试模式测试一下。测试项目中主边刷测试放在第二、三项，左、右轮子的前进、后退，分别放在第四、五、六、七项，第八项放真空测试。

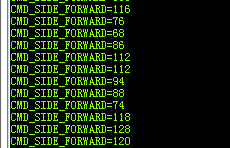
|  |
| --- |
| TEST\_CMD\_E test\_item\_table[UI\_TEST\_ITEM\_MAX]={  CMD\_BAT,  CMD\_MAIN\_FORWARD,  CMD\_SIDE\_FORWARD,  CMD\_WHEEL\_L\_FORWARD,  CMD\_WHEEL\_R\_FORWARD,  CMD\_WHEEL\_L\_REV\_SPD, CMD\_WHEEL\_R\_REV\_SPD,  CMD\_VACCUM,  CMD\_TEST\_CLIFF\_LIGHTTOUCH,  CMD\_TEST\_BUMP\_DROP,  CMD\_SIDE\_FORWARD,  CMD\_BAT,  CMD\_SIDE\_REV,  CMD\_DOCK\_CURRENT,  CMD\_VACCUM}; |

下载后上电等待系统启动完成后，长按启动/暂停按键进入测试模式第一项，短按一次进入第二项，进入主刷测试



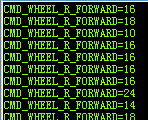
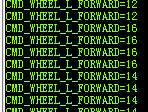
上图是主刷运行时的测试项目打印电流值。

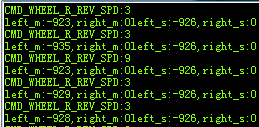
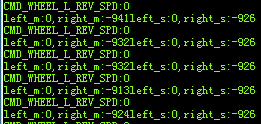
然后再按一次是边刷



上图是边刷运行时的测试项目打印电流值。

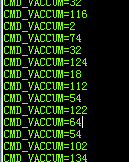
继续测试后面的轮子





以上是轮子两个测试项目，第一张和第二张图打印轮子的电流值，第三张和第四张图是反转的速度，其中左轮的left\_m是实际速度，left\_s是预设速度，CMD\_WHEEL\_L\_REV\_SPD是预设速度与实际速度的差值，右轮同样的道理。

最后测试真空



上图是真空电机运行时的测试项目打印电流值。

## 3.3 步骤三 light touch通道修改和使能

### 3.3.1 与light touch有关的代码

|  |  |  |
| --- | --- | --- |
| 文件名 | 所在文件夹 | 作用 |
| lt\_bump.c | src/sensor | lt bump的判断 |
| sensor.c | src/sensor | cliff、light touch adc数据采集 |
| sensor\_comm.c | src/sensor | 申请中断初始化并处理adc采样过程 |
| adc\_chan.h | src/cfg | adc通道定义 |
| hal\_amicro\_gpio.h | src/cfg | gpio管脚宏定义 |
| am\_config\_macro.h | includes | 机器工作参数 |

### 3.3.2 light touch分布方位图



FRONTRIGHT

FRONTLEFT

Light touch接收窗

圆泡



RIGHT

LEFT

CENTERLEFT

CENTERRIGHT

### 3.3.3 实现原理

light touch实现的原理是，红外发射管发射红外线，接收管串联在电阻上，cpu通过采集电阻上开红外发射管与关红外发射管之间的电压，取两个电压之间的差值即可感知机器前方距离障碍物的远近。Light touch的采集放在一个4K的定中断里执行，采集时间也就是1s/4K=0.25ms。

### **3.3.4 adc通道修改**

首先要确定好机器上一共有几路light touch，并根据APP系统中的逻辑分配好通道。打开adc\_chan.h找到light touch adc通道代码，如下

|  |
| --- |
| m(ADC\_CHAN\_LT\_RIGHT ,6 )  m(ADC\_CHAN\_LT\_CENTERLEFT ,7 )  m(ADC\_CHAN\_LT\_CENTERRIGHT ,8 )  m(ADC\_CHAN\_LT\_FRONTRIGHT ,9 )  m(ADC\_CHAN\_LT\_FRONTLEFT ,10 )  m(ADC\_CHAN\_LT\_LEFT ,11 ) |

对应机器light touch adc通道修改即可，排列的顺序不能改，只能修改后面的数字，数字x即对应通道ADCx。

### **3.3.5** 红外发射管使能管脚修改

修改好light touch adc通道之后，还要对红外发射管的电源使能。打开hal\_amicro\_gpio.h文件，找到AM\_IO\_LIGHT\_TOUCH\_LED宏定义

|  |
| --- |
| #define AM\_IO\_LIGHT\_TOUCH\_LED GPIOE(10) |

这个就是红外发射管电源使能管脚的宏定义，将它修改成机器对应的管脚。同时打开gpio\_init.h找到对应管脚的初始化状态，如当前light touch红外发射管的电源使能管脚为PE10

|  |
| --- |
| m( GPIO\_DIR\_OUT , 4 , 10 , 1 ,GPIO\_PR\_UP )， |

这句代码中的GPIO\_DIR\_OUT表示管脚输出方向，4表示GPIOE，10表示第十个管脚，1表示管脚宽度，GPIO\_PR\_UP表示上拉。并且要注意管脚宏定义中，不能有与AM\_IO\_LIGHT\_TOUCH\_LED相同管脚的宏，否则会由于管脚重复导致一些错误产生。

### 3.3.6 物理通道共用

如果机器light touch只有前左或者前右接收头，sensor.c中的逻辑通道和物理通道映射关系结构体表remap[]要修改成共用，如remap[]中代码

|  |
| --- |
| {LT\_CENTERRIGHT,ADC\_CHAN\_LT\_CENTERRIGHT},  {LT\_FRONTLEFT,ADC\_CHAN\_LT\_FRONTLEFT},  {LT\_RIGHT,ADC\_CHAN\_LT\_RIGHT},  {LT\_LEFT,ADC\_CHAN\_LT\_LEFT},  {LT\_FRONTRIGHT,ADC\_CHAN\_LT\_FRONTLEFT},  {LT\_CENTERLEFT,ADC\_CHAN\_LT\_CENTERLEFT}, |

代码左边为逻辑通道，右边为物理通道，其中逻辑通道LT\_FRONTLEFT和LT\_FRONTRIGHT共用了物理通道ADC\_CHAN\_LT\_FRONTLEFT，这是因为机器只有前左接收头。

### **3.3.7 适配测试**

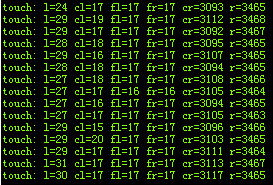
测试模式数组第九项修改为CMD\_TEST\_CLIFF\_LIGHTTOUCH，

|  |
| --- |
| TEST\_CMD\_E test\_item\_table[UI\_TEST\_ITEM\_MAX]={  CMD\_BAT,  CMD\_MAIN\_FORWARD,  CMD\_SIDE\_FORWARD,  CMD\_WHEEL\_L\_FORWARD,  CMD\_WHEEL\_R\_FORWARD,  CMD\_WHEEL\_L\_REV\_SPD, CMD\_WHEEL\_R\_REV\_SPD,  CMD\_VACCUM,  CMD\_TEST\_CLIFF\_LIGHTTOUCH,  CMD\_TEST\_BUMP\_DROP,  CMD\_SIDE\_FORWARD,  CMD\_BAT,  CMD\_SIDE\_REV,  CMD\_DOCK\_CURRENT,  CMD\_VACCUM}; |

但是原来的测试内容不大方便我们看数据，我们稍微修改一下CMD\_TEST\_CLIFF\_LIGHTTOUCH的内容。首先CMD\_TEST\_CLIFF\_LIGHTTOUCH内的所有内容删除或注释，在sensor.c中我们提供了一个打印light touch adc值的函数print\_touch()，把这个函数加入到CMD\_TEST\_CLIFF\_LIGHTTOUCH下，记得要先声明一下，代码如下

|  |
| --- |
| case CMD\_TEST\_CLIFF\_LIGHTTOUCH:  {  print\_touch();  } |

下载测试，效果如下



图中的r和cr的数值就是light touch被遮挡时的adc数据，其他的是未遮挡的数据。

### **3.3.8 阈值设定**

在am\_config\_macro.h中如下代码是关于light touch的阈值设定，当机器达到这个阈值时，机器会减速行走。元素中的ON与OFF原来是表示测量数据时动态变化的最高与最低，现在统一表示一样的值。

|  |
| --- |
| typedef enum  {  LT\_CENTERLEFT\_THRESHOLD\_ON = 1100, /\*中间左\*/  LT\_CENTERLEFT\_THRESHOLD\_OFF = 1100,  LT\_CENTERRIGHT\_THRESHOLD\_ON = 1100, /\*中间右\*/  LT\_CENTERRIGHT\_THRESHOLD\_OFF = 1100,  LT\_LEFT\_THRESHOLD\_ON = 600, /\*左\*/  LT\_LEFT\_THRESHOLD\_OFF = 600,  LT\_RIGHT\_THRESHOLD\_ON = 600, /\*右\*/  LT\_RIGHT\_THRESHOLD\_OFF = 600,  // LT\_FRONT\_LEFT\_THRESHOLD\_ON = 95, /\*前左\*/  // LT\_FRONT\_LEFT\_THRESHOLD\_OFF = 95,  LT\_FRONT\_LEFT\_THRESHOLD\_ON = 300, /\*前左\*/  LT\_FRONT\_LEFT\_THRESHOLD\_OFF = 300,  LT\_FRONT\_RIGHT\_THRESHOLD\_ON = 300, /\*前右\*/  LT\_FRONT\_RIGHT\_THRESHOLD\_OFF = 300  }LT\_E; |

在am\_config\_macro.h如下代码是关于LT\_bump的阈值设定，当机器达到该阈值UI\_SET\_LT\_MAX\_LIGHT\_VALUE 时会调头，详细处理过程可参考lt\_bump.c中touch\_bump\_run函数。

|  |
| --- |
| typedef enum  {  UI\_SET\_ROBOT\_BUMP\_USING\_LIGHT\_TOUCH = 1,  UI\_SET\_LT\_ACC\_MMPS = 800,  UI\_SET\_LT\_BUMP\_THROD = 60 ,  UI\_SET\_LT\_BUMP\_LEFT\_RIGHT\_THROD = 120,  UI\_SET\_LT\_MAX\_LIGHT\_VALUE = 200,  UI\_SET\_LT\_BUMP\_ACTIVE\_DIR = LT\_FRONT\_LEFT|LT\_FRONT\_RIGHT,  }LT\_BUMP\_E; |

**注意：**如果需要对相关阈值进行修改，就要先测量相关阈值的数据。例如要测量机器的light touch的阈值，假定是10cm。按照上面配置好之后，进入测试模式并连接打印。用一张白纸分别与每个接收头距离10cm并且正对接收头，此时的adc数据就可以设为该接收头light touch的阈值，这样同样可以对LT\_bump与cliff进行阈值测量。

## **3.4 步骤四 ciff通道修改和使能**

### **3.4.1 与cliff相关的代码**

|  |  |  |
| --- | --- | --- |
| 文件名 | 所在文件夹 | 作用 |
| sensor.c | src/sensor | cliff、light touch adc数据采集 |
| sensor\_comm.c | src/sensor | 申请中断初始化并处理adc采样过程 |
| adc\_chan.h | src/cfg | adc通道定义 |
| hal\_amicro\_gpio.h | src/cfg | gpio管脚宏定义 |
| am\_config\_macro.h | includes | 机器工作参数 |

### **3.4.2 cliff分布方位图**

FRONTLEFT

FRONTRIGHT



RIGHT

LEFT

### **3.4.3 实现原理**

cliff实现的原理与light touch 相同，cliff的作用是感应机器与地面的距离，防止机器从高处跌落。同样cliff采集也是放在4K中断里执行。

### **3.4.4 adc通道修改**

与light touch 一样，在adc\_chan.h找到cliff adc通道代码

|  |
| --- |
| m(ADC\_CHAN\_CLIFF\_LEFT ,2 )\  m(ADC\_CHAN\_CLIFF\_FRONTLEFT ,1 )\  m(ADC\_CHAN\_CLIFF\_RIGHT ,3 )\  m(ADC\_CHAN\_REAR\_CLIFF\_LEFT ,0 )\  m(ADC\_CHAN\_REAR\_CLIFF\_RIGHT ,4 )\  m(ADC\_CHAN\_CLIFF\_FRONTRIGHT ,5 )\ |

对应机器cliff adc通道修改，同样排列的顺序不能改，只能修改后面的数字，数字x即对应通道ADCx。

### **3.4.5** 红外发射管使能管脚修改

修改好之后在hal\_amicro\_gpio.h文件，找到AM\_IO\_CLIFF\_LED宏定义，

|  |
| --- |
| #define AM\_IO\_CLIFF\_LED GPIOF(1) |

参照light touch红外发射管使能管脚修改部分将它修改成机器对应的管脚。并且要注意管脚宏定义中，不能有与AM\_IO\_CLIFF\_LED相同管脚的宏。在gpio\_init.h中修改cliff红外发射管的电源使能管脚初始化状态。

### **3.4.6** 物理通道共用

当机器cliff只有前左或者前右时也需要修改物理通道共用，具体参考light touch的修改。

### **3.4.7 适配测试**

同样在测试模式中修改CMD\_TEST\_CLIFF\_LIGHTTOUCH内容成sensor.c里的print\_cliff()查看打印cliff的数据

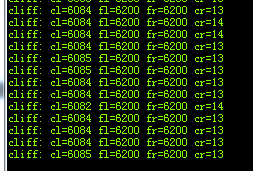
下载后上电等待系统启动完成后，进入测试模式，打印如下

case CMD\_TEST\_CLIFF\_LIGHTTOUCH:

{

print\_cliff();

}



这个是机器正常放置时cr为悬空的数据其他为未悬空数据。

### **3.4.8 阈值相关**

在am\_config\_macro.h中如下代码是cliff相关的阈值，可根据机器需要修改。当机器达到这个阈值时会躲开悬崖。元素中的ON与OFF原来是表示测量数据时动态变化的最高与最低，现在统一表示一样的值。

|  |
| --- |
| typedef enum  {  CLIFF\_REAR\_LEFT\_THRESHOLD\_ON = -1, /\*后左\*/  CLIFF\_REAR\_LEFT\_THRESHOLD\_OFF = -1,  CLIFF\_REAR\_RIGHT\_THRESHOLD\_ON = -1, /\*后右\*/  CLIFF\_REAR\_RIGHT\_THRESHOLD\_OFF = -1,  CLIFF\_LEFT\_THRESHOLD\_ON = 100, /\*左\*/  CLIFF\_LEFT\_THRESHOLD\_OFF = 100,  CLIFF\_RIGHT\_THRESHOLD\_ON = 100, /\*右\*/  CLIFF\_RIGHT\_THRESHOLD\_OFF = 100,  CLIFF\_FRONT\_LEFT\_THRESHOLD\_ON = 100, /\*前左\*/  CLIFF\_FRONT\_LEFT\_THRESHOLD\_OFF = 100,  CLIFF\_FRONT\_RIGHT\_THRESHOLD\_ON = 100, /\*前右\*/  CLIFF\_FRONT\_RIGHT\_THRESHOLD\_OFF = 100  }CLIFF\_E; |

如果需要对相关阈值进行修改，参考上一节。

## 3.5 步骤五 bump管脚、触发电平修改

### **3.5.1 bump相关的代码**

|  |  |  |
| --- | --- | --- |
| 文件名 | 所在文件夹 | 作用 |
| hal\_amicro\_gpio.h | src/cfg | gpio管脚宏定义。 |
| test\_cmd\_handle.c | src/ui-manager | 轮子抬起检测、bump检测、电池温度获取和测试相关。 |
| hal\_amicro\_gpio.h | src/cfg | gpio管脚宏定义 |

3.5.2 bump分布方位图

BUMP挡板



AM\_IO\_BUMP\_RIGHT

AM\_IO\_BUMP\_LEFT

### **3.5.3 bump原理**

bump的原理是机器前头有左右两个微动开关，与前面板相应的机械传动结构连结，当机器右边碰撞到障碍物时，右边开关触发信号，当机器左边碰撞到障碍物时，左边开关触发信号，当中间碰撞到障碍物时，两边同时触发信号。依此判断障碍物的方位。当有些障碍物不是很大，light touch有可能检测不到。bump是对light touch功能的一个补充。

### **3.5.4 管脚及触发电平修改**

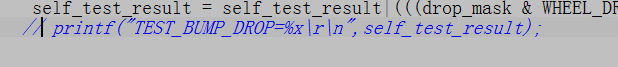
bump只需要对应机器进行管脚修改就可以了，打开hal\_amicro\_gpio.h找到AM\_IO\_BUMP\_LEFT和AM\_IO\_BUMP\_RIGHT对应bump的左右，参照light touch红外发射管使能管脚修改部分将它修改成机器对应的管脚，同样要注意不要有管脚相同的宏定义。有一个需要注意的地方是，bump的触发分为高电平和低电平，打开ui\_manager文件夹中的test\_cmd\_handle.c找到代码

|  |
| --- |
| BumpState robot\_bump\_mask(void)  {  BumpState mask = BUMP\_MASK\_NONE;  if (gpio\_get\_value(AM\_IO\_BUMP\_LEFT)==0) {  mask |= BUMP\_FRONT\_LEFT;  }  if (gpio\_get\_value(AM\_IO\_BUMP\_RIGHT)==0) {  mask |= BUMP\_FRONT\_RIGHT;  }  return mask;  } |

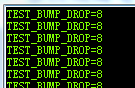
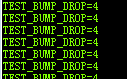
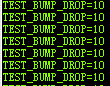
其中判断条件中的==0表示低电平触发，若是机器为高电平触发则需要将0改成1。

### **3.5.5 适配测试**

在测试项目里CMD\_TEST\_BUMP\_DROP包含bump的测试，我们把它设为第十项，并把下面这一句打印去掉注释。



下载后上电等待系统启动完成后，进入测试，打印如下



上面分别是左边、中间和右边触发bump时打印的状态。

## 3.6 步骤六 轮子抬起检测管脚、触发电平修改

### **注意**：由于该功能部分机型没有，所以要先确认适配机型是否有该功能。

### **3.6.1** 轮子抬起检测**相关的代码**

|  |  |  |
| --- | --- | --- |
| 文件名 | 所在文件夹 | 作用 |
| hal\_amicro\_gpio.h | src/cfg | gpio管脚宏定义。 |
| test\_cmd\_handle.c | src/ui-manager | 轮子抬起检测、bump检测、电池温度获取和测试相关。 |
| hal\_amicro\_gpio.h | src/cfg | gpio管脚宏定义 |

### **3.6.2** 轮子抬起检测**原理**

轮子抬起检测的原理是左右轮子机械结构上各连接着一个微动开关，当轮子抬起时会触发开关。

### **3.6.3 管脚及触发电平修改**

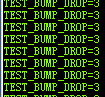
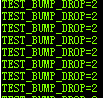
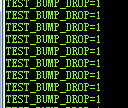
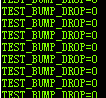
轮子抬起检测只需要对应机器进行管脚修改就可以了，打开hal\_amicro\_gpio.h找到AM\_IO\_WHEEL\_DROP\_ LEFT和AM\_IO\_WHEEL\_DROP\_RIGHT对应轮子抬起检测的左右，参照light touch红外发射管使能管脚修改部分将它修改成机器对应的管脚，同样要注意不要有管脚相同的宏定义。有一个需要注意的地方是，轮子抬起检测的触发分为高电平和低电平，打开ui\_manager文件夹中的test\_cmd\_handle.c找到代码

|  |
| --- |
| WheelDropState robot\_wheel\_drop\_mask(void)  {  WheelDropState mask = WHEEL\_DROP\_STATE\_NONE;  if (gpio\_get\_value(AM\_IO\_WHEEL\_DROP\_RIGHT)==0) {  mask |= WHEEL\_DROP\_STATE\_RIGHT;  }  if (gpio\_get\_value(AM\_IO\_WHEEL\_DROP\_LEFT)==0) {  mask |= WHEEL\_DROP\_STATE\_LEFT;  }  return mask;  } |

其中判断条件中的==0表示低电平触发，若是机器为高电平触发则需要将0改成1。

### **3.6.4 适配测试**

轮子抬起检测与bump检测同在一个case语句里，所以下载启动系统后，同样进入第十项测试。



上面四个图分别是两轮未抬起、右轮抬起、左轮抬起和两轮都抬起的打印。

## 3.7 步骤七 其他的一些物理结构

### **3.7.1 相关代码**

|  |  |  |
| --- | --- | --- |
| 文件名 | 所在文件夹 | 作用 |
| am\_config\_macro.h | includes | 机器工作参数 |
| syscall\_sys\_api.h | libs | 系统部分开放函数 |

### **3.7.2 参数配置**

除了上面介绍的这些部分之外，还需要一些物理结构上的适配。主要是针对不同机器之间配置的差异。

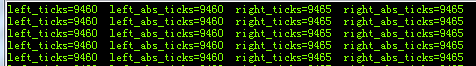
打开am\_config\_macro.h找到如下四个参数

|  |
| --- |
| WHEEL\_DIA 轮子直径  WHEEL\_LEFT\_RIGHT\_DISTANCE 左右轮子间距  WHEEL\_CODE\_RATIO 轮子码盘比  PAD\_WIDTH 机器机身最宽处 |

轮子码盘比是轮子转一圈机器从码盘采到的脉冲个数。可以通过syscall\_sys\_api.h的motor\_encode\_get函数测量，这个函数作用是获取轮子码盘转动的脉冲数。可以将如下代码放入主函数创建的main\_app\_task线程的while循环里。

|  |
| --- |
| printf("left\_ticks %d left\_abs\_ticks %d right\_ticks %d right\_abs\_ticks %d",\  motor\_encode\_get()->left\_ticks, motor\_encode\_get()->left\_abs\_ticks ,\  motor\_encode\_get()->right\_ticks, motor\_encode\_get()->right\_abs\_ticks); |

下图是某款机型轮子各转10圈的脉冲数，故这款机型码盘可以取946 。

 打开ui\_config.c，找到如下代码

|  |
| --- |
| .funtion\_info =  {  .clean\_fun.bits.motor\_driver = 0,//电机驱动芯片类型  .clean\_fun.bits.wheel\_left\_dir = 0, //左轮旋转方向  .clean\_fun.bits.wheel\_right\_dir= 0, //右轮旋转方向  .clean\_fun.bits.wheepdrop = 0,//是否有漏液检测  .clean\_fun.bits.stasis = 0,//是否有万向轮  .clean\_fun.bits.lt\_type = 0,//如果有远近两组lt的话填写 1  .clean\_fun.bits.rear\_cliff = 0,//如果模组后面有cliff，填 1  .clean\_fun.bits.ltBumpNoBack = 1,//lt bump后，机器是否先后退再转弯。 一般如果轮子不在模具的中间，则不适宜打开。  .clean\_fun.bits.gyro\_dir = 0,//陀螺仪方向  .clean\_fun.bits.wall\_follow\_dir = 0, //0: left and right, 1:left, 2:right  } |

这是硬件配置上一些差异的配置，按需求去修改。

机器适配到这里已经能够正常跑起来了。正常跑起来的现象是在空旷的地方没有任何障碍的情况下，机器启动清扫后一直循环向前行走4米左右然后回头行走4米左右的过程，这个过程不报错。

## 3.8 步骤八 IR接收管脚修改和解码

### **3.8.1 与ir有关的代码**

|  |  |  |
| --- | --- | --- |
| 文件名 | 所在文件夹 | 作用 |
| ir\_decode.c | src/ui-manager | 红外解码相关 |
| remote\_handle.c | src/ui-manager | 红外解码相关 |
| exti\_cfg.c | src/cfg | 中断唤醒配置 |

### **3.8.2 红外接收头方位分布图**



圆泡

红外接收窗

MIDDLE\_RIGHT

MIDDLE\_LEFT



LEFT

BACK\_LEFT

BACK\_RIGHT

LEFT

圆泡

|  |  |
| --- | --- |
| 宏定义 | 作用 |
| AM\_IO\_IR\_REC\_BACK\_LEFT | 后左红外接收头 |
| AM\_IO\_IR\_REC\_BACK\_RIGHT | 后右红外接收头 |
| AM\_IO\_IR\_REC\_MIDDLE\_RIGHT | 中右红外接收头 |
| AM\_IO\_IR\_REC\_MIDDLE\_LEFT | 中左红外接收头 |
| AM\_IO\_IR\_REC\_RIGHT | 右红外接收头 |
| AM\_IO\_IR\_REC\_LEFT | 左红外接收头 |

### **3.8.3 ir的作用**

ir是红外接收，作用有两个，一个是接收充电座和遥控器的信号，并进行分类解码。另一个是接收到信号就唤醒机器。

### **3.8.4 接收管脚修改**

接收头装在前面板上，圆泡在机器上方。打开hal\_amicro\_gpio.h，分别找到

|  |
| --- |
| #define AM\_IO\_IR\_REC\_BACK\_LEFT GPIOB(6)  #define AM\_IO\_IR\_REC\_BACK\_RIGHT GPIOB(7)  #define AM\_IO\_IR\_REC\_MIDDLE\_RIGHT GPIOE(12)  #define AM\_IO\_IR\_REC\_MIDDLE\_LEFT GPIOF(1)  #define AM\_IO\_IR\_REC\_RIGHT GPIOF(2)  #define AM\_IO\_IR\_REC\_LEFT GPIOF(3) |

根据机器修改宏定义对应管脚，依然要注意其他宏定义不要有相同的管脚。

同时，红外是可以使用为中断触发唤醒的，也就是说有信号接收就会触中断唤醒，如需要配置成中断唤醒可在exti\_cfg.c中修改或添加。参考其中一个接收头中断唤醒，代码如下

|  |
| --- |
| exti\_user\_init.Wakeup\_type = EXTI\_WAKEUP\_LOW\_LEVEL;  exti\_user\_init.Int\_type = EXTI\_POSITIVE\_EDGE;  id = open(DEV\_EXTI,AM\_IO\_IR\_REC\_LEFT);  ioctl(id,EXTI\_INIT,&exti\_user\_init);  request\_irq(id,(long)wake\_handler,0);  wakeup\_id[i++] = id; |

代码中AM\_IO\_IR\_REC\_LEFT是我们这个接收头管脚的宏定义，可以定义成任何名称，只要对应的管脚没错就可以了。

### **3.8.5 增加接收头**

如果机器上的接收头超过了6个就需要修改代码来增加了，增加接收头步骤如下

1. 在hal\_amicro\_gpio.h中增加宏定义和对应管脚。
2. 在remote\_handle.c找到ir\_gpio[]数组，在后面添加步骤①管脚的宏定义。
3. 在ui-ir.h中找到枚举变量IR\_REMOT\_POSITION\_E在后面添加接收头名字。
4. 在ui-ir.h中，将宏定义REMOTE\_DECODER\_MAX修改成相应的接收头数值。
5. 在ui-ir.h中，将IR\_DECODER\_MAX也修改成接收头数值。
6. 在docking\_new.h中找到枚举变量IR\_local\_Index在后面添加接收头名字。
7. 如果需要配置成中断唤醒可参考上面的内容。

### **3.8.6 红外解码**

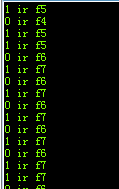
红外信号编码方式是"1"为高电平持续3ms,低电平持续1ms，"0"为高电平持续1ms,低电平持续3ms。信号每次发送一个字节信号，发送间隔90ms左右，解码函数放在一个主函数创建的线程里运行。如果解码出来的码值不对，需要对解码过程中的延时进行修改。具体的解码过程可以参考ir\_decode.c中的remote\_decode\_ir函数。

### **3.8.7 适配测试**

红外接收的测试我们需要自己加打印，打开ui\_manager文件夹，找到remote\_handle.c，在下面代码我们添上一句打印，修改如下

|  |
| --- |
| if ((result == 0)&&(remote\_decode[index].rxByte != 0))  {  current\_ir\_code = remote\_decode[index].rxByte;//存放遥控器的数据  remote\_timer = REMOTE\_DATA\_KEEP\_COUNT;  ir\_val[index] = remote\_decode[index].rxByte; //存放充电座的信号  ir\_val\_test[index] = ir\_val[index];  printf("%d ir %x \r\n",index,remote\_decode[index].rxByte);//添上打印    } |

这样哪个接收头收到信号，就会把信号打印出来。下载后上电等待系统启动完成后，收到信号就会打印出来，上图。



0就是前左，1就是前右，后面是信号的码值，图上的信号是充电座信号。对比一下信号码值是否正确，如果不正确，说明解码不对。打开ir\_decode.c找到remote\_decode\_ir函数，对其中解码信号的时延，对比座子信号发送的时序进行修改。