目录

[1. NO-OS介绍 1](#_Toc24050)

[1.1. 配置要求 1](#_Toc30497)

[1.2. 源码介绍 2](#_Toc23505)

[1.3. 初始化流程 3](#_Toc29436)

[2. 芯片配置 4](#_Toc7165)

[2.1. 芯片模式配置 7](#_Toc23266)

[2.2. 射频端口配置 8](#_Toc8235)

[2.3. 数据接口配置 9](#_Toc31577)

[2.4. 带宽配置 11](#_Toc5602)

[2.4.1. 常用带宽 11](#_Toc31355)

[2.4.2. 自定义带宽 11](#_Toc18112)

[2.5. 系统参考时钟 12](#_Toc28598)

[2.6. 本振设置 12](#_Toc6069)

[2.7. 校准功能配置 12](#_Toc4743)

[2.8. 增益控制配置 12](#_Toc30546)

[2.9. TDD模式下线控配置 13](#_Toc30)

[2.10. 锁定模式配置 13](#_Toc23394)

[3. 芯片初始化 14](#_Toc22283)

[3.1. 检测芯片是否正常进入状态机 14](#_Toc12384)

[3.2. 检测TX和RX通路是否正常工作 14](#_Toc8546)

[4. 芯片API使用 15](#_Toc29065)

[4.1. 芯片初始化 15](#_Toc2300)

[4.2. 获取状态机状态 15](#_Toc5789)

[4.3. TRX频点切换 15](#_Toc28348)

[4.4. TX衰减设置 15](#_Toc11404)

[4.5. MGC增益配置 16](#_Toc1282)

[5. 代码移植 16](#_Toc22688)

[5.1. 数据类型移植 16](#_Toc20147)

[5.2. Linux系统移植 16](#_Toc16870)

[5.3. 裸机移植 16](#_Toc12156)

# NO-OS介绍

## 配置要求

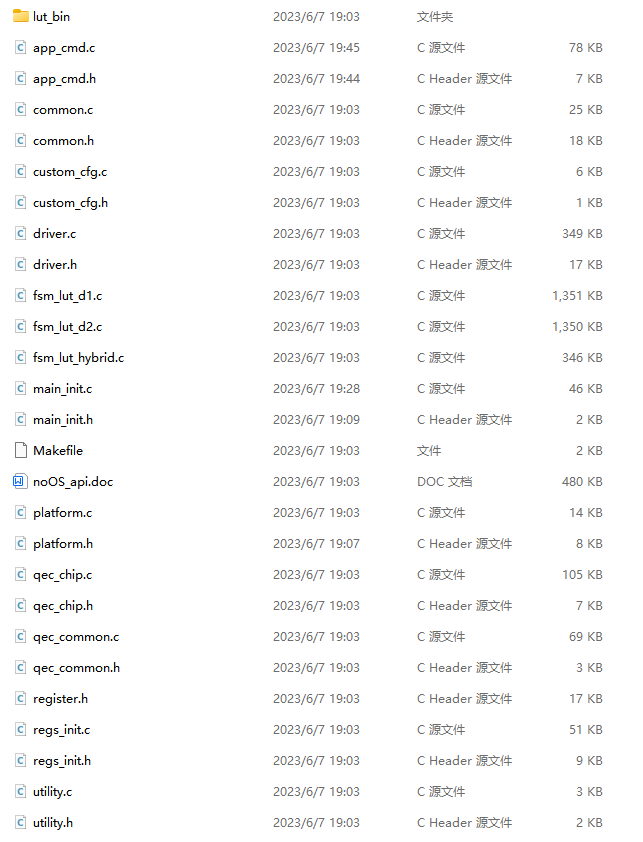
硬件配置：32位CPU，主频108MHz或以上。

操作系统：Linux、Windows、noOS、RTOS系统等。

编译器系统：支持64位数据类型，支持double/float数据类型，带有标准函数库math.h头文件。

## 源码介绍

芯片源码目录如下图示。



Driver.c芯片驱动代码。

Common.c和Common.h芯片默认数据配置，以及芯片phy结构体等定义。

main\_init.c用户接口，包括初始化、LO切换、TX衰减设置等。

custom\_cfg.c用户可配置文件。

app\_cmd.c芯片测试使用的命令。

Platform.c和platform.h硬件平台、软件平台相关的配置，我们移植主要是改这个文件的配置，包括SPI接口的适配、FPGA读写控制的适配等。

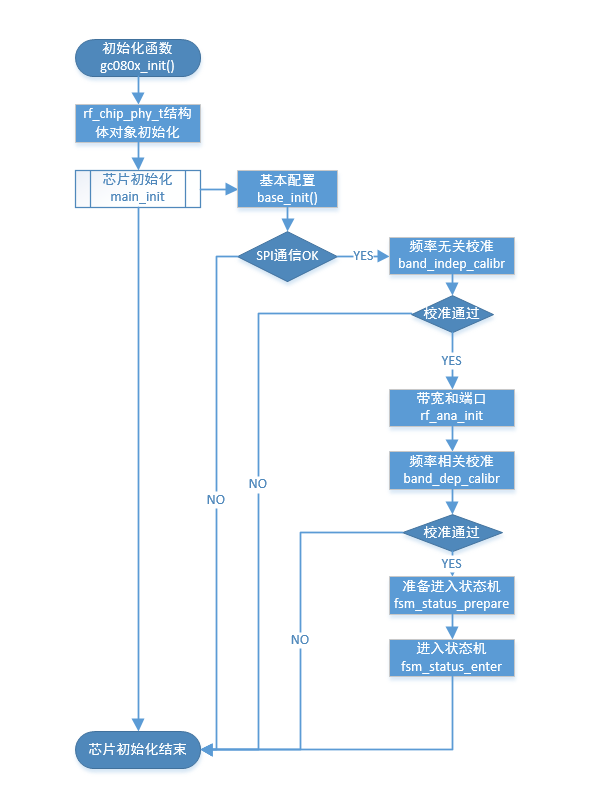
regs\_init.c芯片初始化会用到的寄存器配置。

qec\_\*.\*是芯片镜像抑制的校准算法。

fsm\_lut\_\*.c是芯片基本配置，在初始化的时候会通过SPI下载到芯片中。

## 初始化流程

芯片整体的初始化流程如下图示。



# 芯片配置

用户可配置的文件在custom\_cfg.c里面。这里列举代码中默认的配置如下，这是两颗芯片的配置（RF\_PHY\_NUMBER默认为2，在custom\_cfg.h文件中定义）。在main\_init.c中的gc080x\_init函数会把g\_phy\_ob和g\_phy\_config关联在一起，比如用chip0的配置，我们可以这样初始化芯片gc080x\_init(&g\_phy\_obj[0], &g\_phy\_config[0])。

|  |
| --- |
| short g\_phy\_select = 0;  rf\_chip\_phy\_t g\_phy\_obj[RF\_PHY\_NUMBER];  chip\_config\_t g\_phy\_config[RF\_PHY\_NUMBER] =  {  //chip 0  {  .mode = CH1\_CH2\_TDD,  .use\_bybrid\_mode = 0,  .hybrid\_mode = HYBRID\_FDD\_SXTX,  .dig\_if = LVDS\_IF,  .p0p1\_port = DUAL\_PORT,  .data\_rate = DDR,  .tx\_port[0] = TX\_A,  .tx\_port[1] = TX\_A,  .rx\_port[0] = MIXA\_DIFF,  .rx\_port[1] = MIXA\_DIFF,  .bandwith = BW\_LTE100,  .xtal\_freq = 30720000UL,//64000000UL,  .sys\_fvco = 2949120000ULL,//3072000000ULL,  .x4\_enable = 1,  .fast\_sx\_lock = 0,  .rx\_flo = 2595000000ULL,  .tx\_flo = 2595000000ULL,  .core2\_enable = 0,  .rx\_ext\_loop = 0,  .tx\_ext\_loop = 0,  .qec\_dbfs = 10,  .lol\_dbfs = 40,  .tx\_atten\_chn\_flag = 0,  .rx\_bw\_cal\_flag = 1,  .rx\_dc\_cal\_flag = 1,  .rx\_qec\_flag = 1,  .tx\_dc\_cal\_flag = 1,  .tx\_dac\_cal\_flag = 1,  .tx\_qec\_flag = 1,  .sx\_cal\_flag = 0,  .txlo\_cal\_flag = 0,  .lo\_change\_mode = LO\_CHANGE\_OLD,  .wire\_control\_en = 1,  .wire\_ctrl = PULSE\_CTRL,  .gain\_table\_mode = SPLIT\_TBL,  .gain\_ctrl\_mode = MANUAL\_CTRL,  .gain\_ctrl\_pin\_flag = 0,  .custom\_bandwith\_flag = 0,  .custom\_bandwith = 50000000UL,//50M  .bb\_sample\_rate = 92160000,  .dac\_syspll\_lo\_div = 0x0b,  .adc\_syspll\_lo\_div = 0x0b,  .chip\_sel = 0,  .rxgain\_force\_valid\_flag = 1,  .chip\_ver = 0,  },  //chip 1  {  .mode = CH1\_CH2\_FDD,  .use\_bybrid\_mode = 0,  .hybrid\_mode = HYBRID\_FDD\_SXTX,  .dig\_if = LVDS\_IF,  .p0p1\_port = DUAL\_PORT,  .data\_rate = DDR,  .tx\_port[0] = TX\_A,  .tx\_port[1] = TX\_A,  .rx\_port[0] = MIXA\_DIFF,  .rx\_port[1] = MIXA\_DIFF,  .bandwith = BW\_LTE100,  .xtal\_freq = 30720000UL,//64000000UL,  .sys\_fvco = 2949120000ULL,//3072000000ULL,  .x4\_enable = 1,  .fast\_sx\_lock = 0,  .rx\_flo = 2300000000ULL,  .tx\_flo = 2400000000ULL,  .core2\_enable = 0,  .rx\_ext\_loop = 0,  .tx\_ext\_loop = 0,  .qec\_dbfs = 10,  .lol\_dbfs = 10,  .tx\_atten\_chn\_flag = 0,  .rx\_bw\_cal\_flag = 1,  .rx\_dc\_cal\_flag = 1,  .rx\_qec\_flag = 1,  .tx\_dc\_cal\_flag = 1,  .tx\_dac\_cal\_flag = 1,  .tx\_qec\_flag = 1,  .sx\_cal\_flag = 0,  .txlo\_cal\_flag = 0,  .lo\_change\_mode = LO\_CHANGE\_OLD,  .wire\_control\_en = 1,  .wire\_ctrl = PULSE\_CTRL,  .gain\_table\_mode = SPLIT\_TBL,  .gain\_ctrl\_mode = MANUAL\_CTRL,  .gain\_ctrl\_pin\_flag = 0,  .custom\_bandwith\_flag = 0,  .custom\_bandwith = 50000000UL,//50M  .bb\_sample\_rate = 92160000,  .dac\_syspll\_lo\_div = 0x0b,  .adc\_syspll\_lo\_div = 0x0b,  .chip\_sel = 1,  .rxgain\_force\_valid\_flag = 1,  .chip\_ver = 0,  },  }; |

## 芯片模式配置

在chip\_config\_t中成员mode来表示芯片模式，芯片支持的模式如下：

typedef enum CHIP\_MODE

{

CH1\_FDD=0,

RX1\_TX2\_FDD,

RX2\_TX1\_FDD,

CH2\_FDD,

CH1\_CH2\_FDD,

CH1\_TDD,

CH2\_TDD,

CH1\_CH2\_TDD,

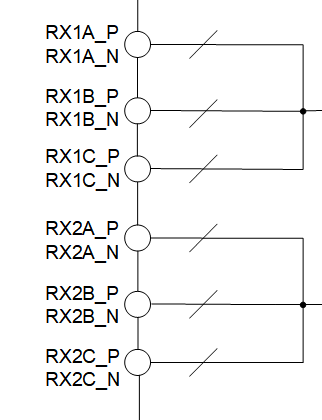
}CHIP\_MODE\_ENUM;

|  |  |
| --- | --- |
| 模式 | 描述 |
| CH1\_FDD | FDD模式，使用芯片RX1和TX1 |
| RX1\_TX2\_FDD | FDD模式，使用芯片RX1和TX2 |
| RX2\_TX1\_FDD | FDD模式，使用芯片RX2和TX1 |
| CH2\_FDD | FDD模式，使用芯片RX2和TX2 |
| CH1\_CH2\_FDD | FDD模式，配置为2T2R模式 |
| CH1\_TDD | TDD模式，使用芯片RX1和TX1 |
| CH2\_TDD | TDD模式，使用芯片RX2和TX2 |
| CH1\_CH2\_TDD | TDD模式，配置为2T2R模式 |

例如，使用2T2R的TDD，则使用CH1\_CH2\_TDD配置。

## 射频端口配置

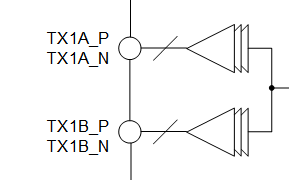
RX的射频端口如下所示，RX1和RX2有A\_P、A\_N、A\_BALANCED（A\_P和A\_N作为一个差分对使用，用A\_BALANCED来表示）。在chip\_config\_t中成员rx\_port[0]来表示RX1的端口配置，rx\_port[1]来表示RX2的端口配置。



下面是在代码中RX端口的配置，MIXAP\_SE对应我们硬件A\_P，MIXAN\_SE对应我们硬件A\_N，MIXA\_DIFF对应我们硬件A\_BALANCED，后面的B和C同理。

|  |
| --- |
| typedef enum RX\_PORT  {  MIXAP\_SE=0,  MIXAN\_SE,  MIXA\_DIFF,  MIXBP\_SE,  MIXBN\_SE,  MIXB\_DIFF,  MIXCP\_SE,  MIXCN\_SE,  MIXC\_DIFF,  MIX\_CNT,  }RX\_PORT\_ENUM; |

如下图示是芯片TX射频口的硬件连接。TX1和TX2可选A和B端口，只有差分连接，不存在单端。在chip\_config\_t中成员tx\_port[0]来表示TX1端口的设置，tx\_port[1]来表示TX2端口设置。

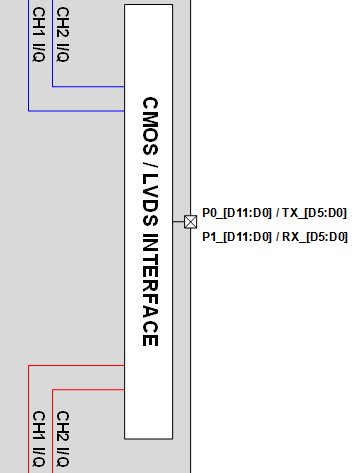


下面是在代码中TX端口的配置，TX\_A对应我们硬件A，TX\_B对应我们硬件B。

|  |
| --- |
| typedef enum TX\_S\_PORT  {  TX\_A=0,  TX\_B,  TX\_PORT\_CNT,  }TX\_S\_PORT\_ENUM; |

## 数据接口配置

数据接口的PIN脚如下所示，芯片支持LVDS和CMOS两种接口。



在chip\_config\_t中成员dig\_if来表示我们选择LVDS接口或CMOS接口，代码中的配置可选如下。

|  |
| --- |
| typedef enum DIG\_IF  {  LVDS\_IF=0,  CMOS\_IF,  }DIG\_IF\_ENUM; |

在chip\_config\_t中成员p0p1\_port来表示我们使用single port或dual port，代码中的配置可选如下。

|  |
| --- |
| typedef enum IF\_TYPE  {  DUAL\_PORT=0,  SINGLE\_PORT,  }IF\_TYPE\_ENUM; |

在chip\_config\_t中成员data\_rate来表示我们使用SDR或DDR时序，代码中的配置可选如下。

|  |
| --- |
| typedef enum DATA\_RATE  {  SDR=0,  DDR,  }DATA\_RATE\_ENUM; |

## 带宽配置

芯片FDD模式下，2T2R支持的最大带宽已经验证过的为55M，这个和接口传输速率等配置有关，TDD下，2T2R支持的最大带宽已经验证过的为100M。目前支持两种方式配置带宽，一种是代码提供常用的带宽，一种客户自定义带宽模式。下面分别阐述两种配置模式。在chip\_config\_t中，有custom\_bandwith\_flag、custom\_bandwith、bb\_sample\_rate、dac\_syspll\_lo\_div、adc\_syspll\_lo\_div、bandwith这些成员和带宽、BB采样率有关。

### 常用带宽

当使用提供常用的带宽，需要把custom\_bandwith\_flag成员设置0，这样custom\_bandwith、bb\_sample\_rate、dac\_syspll\_lo\_div、adc\_syspll\_lo\_div这些参数配置会失效，代码中会自动配置这些参数。bandwith就是用户选择的带宽，目前代码中提供的带宽如下。

|  |
| --- |
| typedef enum BANDWITH  {  BW\_GSM=0,  BW\_BT,  BW\_LTE5,  BW\_LTE10,  BW\_LTE20,  BW\_LTE40,  BW\_LTE60,  BW\_LTE80,  BW\_LTE100,  BW\_CNT,  }BANDWITH\_ENUM; |

### 自定义带宽

当选择这种模式，需要把custom\_bandwith\_flag成员设置1，这样custom\_bandwith、bb\_sample\_rate、dac\_syspll\_lo\_div、adc\_syspll\_lo\_div、bandwith这些参数配置会生效。custom\_bandwith就是实际客户定义的带宽，以HZ为单位，比如55M，那么该值就应该为55000000。bb\_sample\_rate就是BB的采样率是多少，以HZ为单位。dac\_syspll\_lo\_div、adc\_syspll\_lo\_div、bandwith如何设置，请联系我司FAE。

## 系统参考时钟

在chip\_config\_t中成员xtal\_freq和sys\_fvco是配置外部时钟，xtal\_freq是配置外部参考时钟频率，sys\_fvco参数配置，包括2.4.2章节的dac\_syspll\_lo\_div、adc\_syspll\_lo\_div、bandwith需要联系我司FAE。

## 本振设置

在chip\_config\_t中成员rx\_flo和tx\_flo两者是配置RXLO、TXLO，这个仅仅是配置初始化频率，后面切换LO需要用到后面的频点切换API。

## 校准功能配置

芯片支持的校准有：txdc\_offset\_cal、rxdc\_offset\_cal、rx\_bw\_cal、rxqec\_cal、txqec\_lol\_cal、rx tracking。校准功能描述如下。

|  |  |  |
| --- | --- | --- |
| 校准名称 | 在chip\_config\_t中成员 | 描述 |
| txdc\_offset\_cal | tx\_dc\_cal\_flag | TX直流偏移校准 |
| rxdc\_offset\_cal | rx\_dc\_cal\_flag | RX直流偏移校准 |
| tx\_dac\_cal | tx\_dac\_cal\_flag | DAC校准 |
| rx\_bw\_cal | rx\_bw\_cal\_flag | RX的3dB的bandwidth校准 |
| rxqec\_cal | rx\_qec\_flag | RX IQ mismatch校准 |
| txqec\_lol\_cal | tx\_qec\_flag | TX IQmismatch和LOL校准 |
| sx\_cal | sx\_cal\_flag | 在初始化的时候，快速锁定功能需要提前做这个校准存表，后面进行快速锁定的时候用到 |

## 增益控制配置

在chip\_config\_t中成员gain\_table\_mode、gain\_ctrl\_mode、gain\_ctrl\_pin\_flag三者和RX增益控制配置相关。gain\_table\_mode有两种方式，一种是SPLIT TABLE，一种是FULL TABLE，对应到代码中的配置为：

|  |
| --- |
| typedef enum TABLE\_MODE  {  SPLIT\_TBL,  FULL\_TBL,  }TABLE\_MODE\_ENUM; |

增益控制模式有manual、slow、fast，对应代码中的配置如下：

|  |
| --- |
| typedef enum GCTRL\_MODE  {  MANUAL\_CTRL,  SLOW\_ATTACK,  FAST\_ATTACK,  }GCTRL\_MODE\_ENUM; |

gain\_ctrl\_pin\_flag是通过PIN脚来控制增益，这种方式暂未实现，包括AGC里面的一些参数配置说明，后续会更新。

## TDD模式下线控配置

在chip\_config\_t中成员wire\_control\_en、wire\_ctrl只在TDD模式下生效，这两根线可以控制TDD的状态切换，这两根线是ENABLE和TXNRX。wire\_control\_en为1表示TDD下开启线控模式，否则是通过SPI方式进行控制TDD的状态切换。wire\_ctrl只有在wire\_control\_en为1下生效，wire\_ctrl有两种方式控制，一种是脉冲方式控制，一种是电平方式控制，对应代码中的配置如下：

|  |
| --- |
| typedef enum WIRE\_CTRL  {  PULSE\_CTRL=0,  LEVEL\_CTRL,  }WIRE\_CTRL\_ENUM; |

## 锁定模式配置

LO锁定模式有slow、normal、fast、tracking四种模式，四种模式锁定时间从慢到快，fast、tracking两种模式在初始化的时候需要做SX FCAL校准（sx\_cal\_flag设置为1），并把这些校准值存表，后面在扫频的时候会查表，进而实现快速锁定。

# 芯片初始化

## 检测芯片是否正常进入状态机

芯片初始化完成，没有错误信息打印说明初始化成功。也可以通过读取芯片0xda寄存器判断是否进入了FDD/TDD-RX/TDD-TX状态机模式。如下所示为0xda寄存器值对应不同的状态，如果是FDD该值应该是6，如果是TDD该值应该是3（初始化后芯片默认进入RX）。

|  |  |
| --- | --- |
| 0 | WAIT |
| 1 | ALERT |
| 2 | TX |
| 3 | RX |
| 6 | FDD |

## 检测TX和RX通路是否正常工作

RX射频口连接信号源，通过RSSI函数接口来判断RX链路是否正常工作，调用接口实例如下。

|  |
| --- |
| int cmd\_api\_rx\_rssi\_get(TRX\_CHN\_ENUM chn)  {  int rssi\_val;  rssi\_val = rx\_rssi\_get(&g\_phy\_obj[g\_phy\_select], chn);  LOG\_MAIN("rx rssi, chn:%d, RSSI:%d.%03d dBfs.\n", chn, rssi\_val/1000, abs(rssi\_val)%1000);  return rssi\_val;  } |

TX射频端口连接到频谱仪，通过如下接口可以通过TX1或TX2打单音信号出来。chn是通道，on是打开或关闭，freq是单音频率。

|  |
| --- |
| void cmd\_api\_tx\_tone(TRX\_CHN\_ENUM chn, short on, long freq)  {  LOG\_MAIN("tx tone, chn:%d, on:%d, freq=%d\n", chn, on, freq);  //fn\_geo\_tx\_send\_tone(chn, on, freq);  send\_cordic\_signal(&g\_phy\_obj[g\_phy\_select], chn, g\_phy\_obj[g\_phy\_select].config->bandwith, on, freq);  } |

注：上面两个函数是在app\_cmd.c中。

# 芯片API使用

用户使用的函数接口在main\_init.c中。

## 芯片初始化

int gc080x\_init(rf\_chip\_phy\_t \*phy, chip\_config\_t \*config);

功能：这个函数是芯片的初始化入口函数。

参数：

1. phy芯片结构体
2. config芯片配置

## 获取状态机状态

short fsm\_status\_get\_only(rf\_chip\_phy\_t \*phy);

功能：获取状态机状态。

参数：

1、phy芯片结构体

## TRX频点切换

int trx\_lo\_change(rf\_chip\_phy\_t \*phy, unsigned long long txlo, unsigned long long rxlo);

功能：trx的频率切换。

参数：

1. phy芯片结构体
2. Txlo：0表示使用上一次的频率值。
3. Rxlo：0表示使用上一次的频率值。

## TX衰减设置

int tx\_atten\_change(rf\_chip\_phy\_t \*phy, TRX\_CHN\_ENUM chn, int val, short immed);

功能：设置TX的衰减。

参数：

1. phy芯片结构体
2. chn通道（如果可以单独控制每个通道的衰减，我们需要在用户的配置文件custom\_cfg.c中把tx\_atten\_chn\_flag置为1，否则这个参数无效，设置衰减的时候TX1和TX2是同时生效）
3. val衰减index，取值范围是0到50
4. immed立即生效（不需要退出状态机，不会中断数据业务）

## MGC增益配置

int rx\_gain\_mgc\_change(rf\_chip\_phy\_t \*phy, TRX\_CHN\_ENUM chn, RX\_MGC\_GAIN\_ENUM tb, int val);

功能：MGC下通过SPI设置RX增益。

参数：

1. phy芯片结构体
2. chn通道
3. Tb：split表的哪一级，取值有

|  |
| --- |
| typedef enum RX\_MGC\_GAIN  {  LMT\_G,//index取值0到5，步进是6dB  LPF\_G,//index取值0到12，步进是1dB  DIG\_G,//index取值0到200，步进是0.25dB  }RX\_MGC\_GAIN\_ENUM; |

4、val：index值

# 代码移植

代码是使用C语言编写，理论上能移植到所有主流ARM处理器上。

## 数据类型移植

目前代码中使用了C语言的数据类型有：

|  |
| --- |
| 32位：int、unsigned int、long、unsigned long  16位：short、unsigned short  8位：char、unsigned char  枚举类型 |

移植代码的时候要注意平台厂商提供的编译器里面定义的数据类型是否和上面匹配，如果匹配不需要修改代码里面的数据类型，最主要的是要注意编译器中的long long是否是64位。

## Linux系统移植

拿到源码注意修改Makefile文件中的编译器，改为用户使用的编译器。

## 裸机移植

根据Makefile文件的源码和头文件，添加到用户平台开发集成环境中，编译的时候需要注释一些Linux中的头文件。需要把创建rx tracking线程的相关代码注释掉，使用用户平台的创建线程函数接口，如果是裸机则改为定时器来定时执行rx tracking任务。