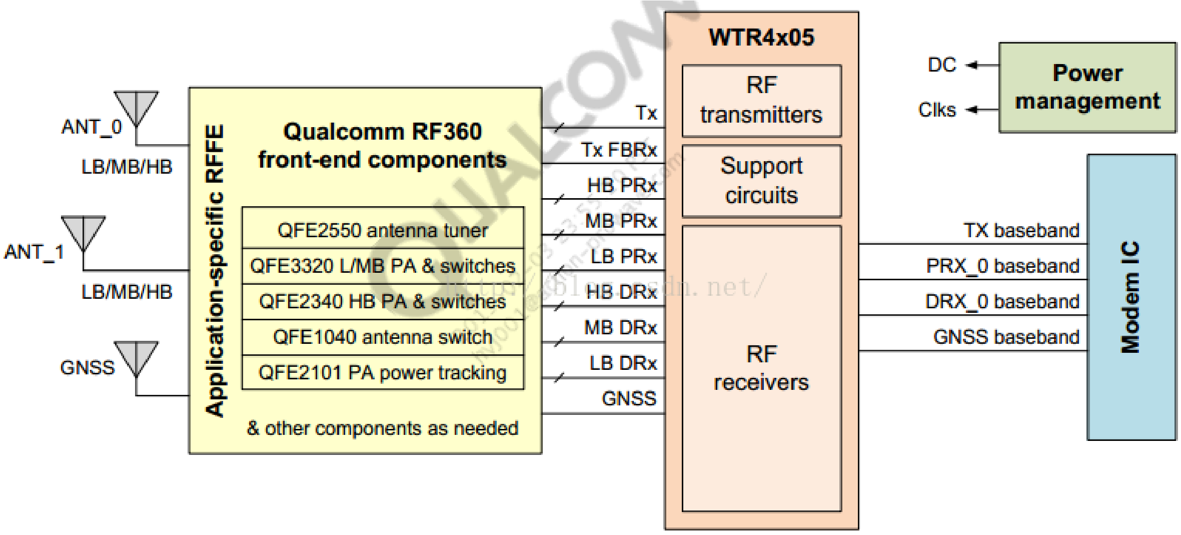
0014-RFC的基本配置流程

# RF总体的连接图如下



左边为天线开关模块跟功放的集成体，如RF360,QFE23XX,SKYXXXX

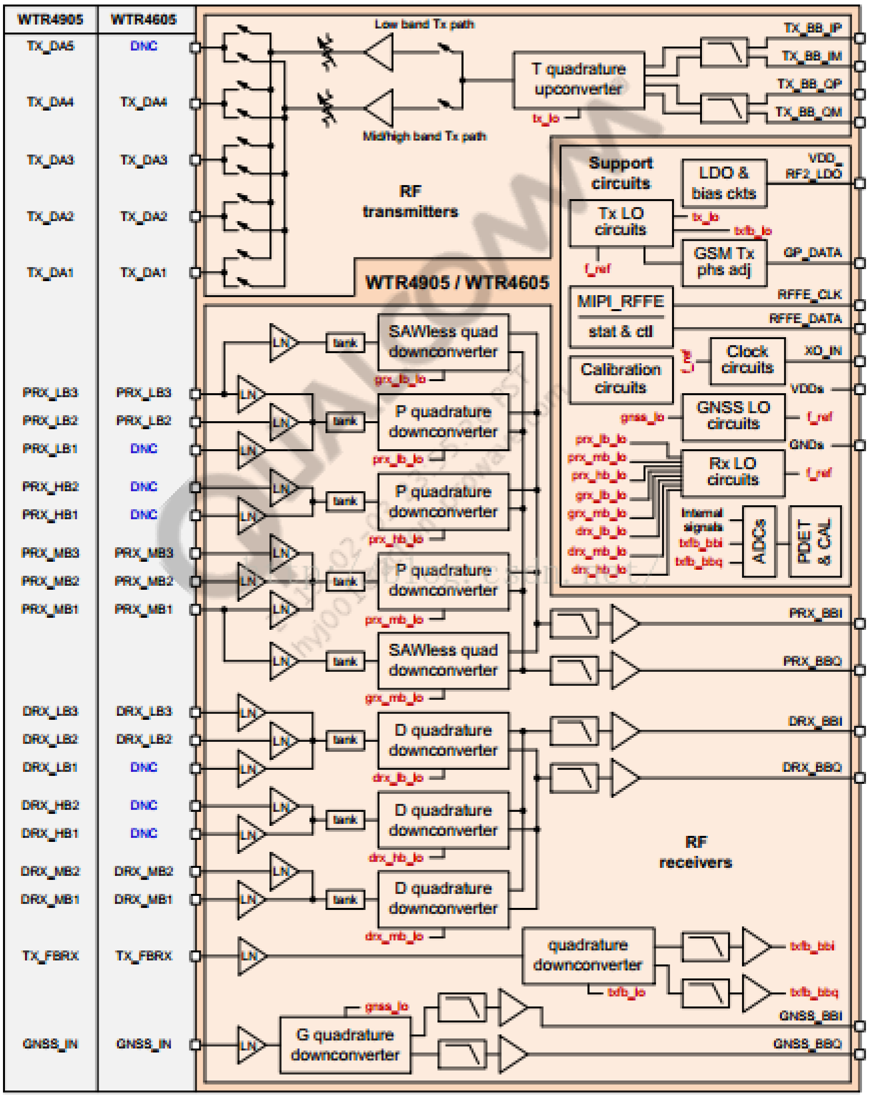
# 下边以**SKY**简称

中间的WTR4X05作用：射频收发器，工作过程为（基带信号<--->上/下变频<--->滤波<--->放大信号）　# 下边以**WTR**简称

最右边modem_ic指处理器芯片，我们这里可以假设为高通的MSM8909处理器，

# 下边就以**MSM**简称

WTR4X05（射频收发器）的内部结构图如下图:



- 上方为`RF transmitters`发送模块框图 ，quadrature upconverter为上变频，高频率才能发送
- 下方为`RF receivers`接收模块框图，quadrature downconverter为下变频，变为低频率后才能有MSMcpu芯片处理

通过开关标号来实现通路的选择

最左边的接口分为三波：发送一波，主接收PRX一波，副接收DRX一波。一半副接收可以节省电量，在不接受的时候可以处于休眠状态

# RF Driver主要设计用到的器件

RFFE(RF Front End)

1. PA
2. ASM
3. RF Card

Transceiver基本是以高通参考设计采用的芯片为主，为WTRXXX系列，但是RFFE由于成本问题，往往不采用参考设计中的芯片，RFFE这边只讨论**PA**和**ASM**，一般分为**MiPi** 和**GRFC**

MIPI 设备通过寄存器配置PA以及ASM，GRFC则是通过GPIO配置。

关于**RFC**涉及的代码目录有

1. `modem_proc/rfc_jolokia//` 核心目录，存放参考设计的RF driver，我们自己添加的也在这里
2. `modem_proc/rfcdevice_pa//` 存放MIPI pa 设备
3. `modem_proc/rfcdevice_asm//` 存放MIPI asm 设备
4. `modem_proc/rfctarget_jolokia/common/qcn//` 存放RF的部分nv
5. `modem_proc/rfcnv//` 存放modem版本下定义nv项的文件nvdefintion.xml

# 如何添加一个新的**MIPI PA**芯片

如果RF前端采用了一款新的MIPI 的pa，需要在modem_proc/rfcdevice_pa新建一个PA设备，在添加之前我们需要查看pa的datasheet，主要有两部分需要着重查看：端口结构图，寄存器，以sky77638为例，这款芯片复用了PA 以及ASM

端口结构图：

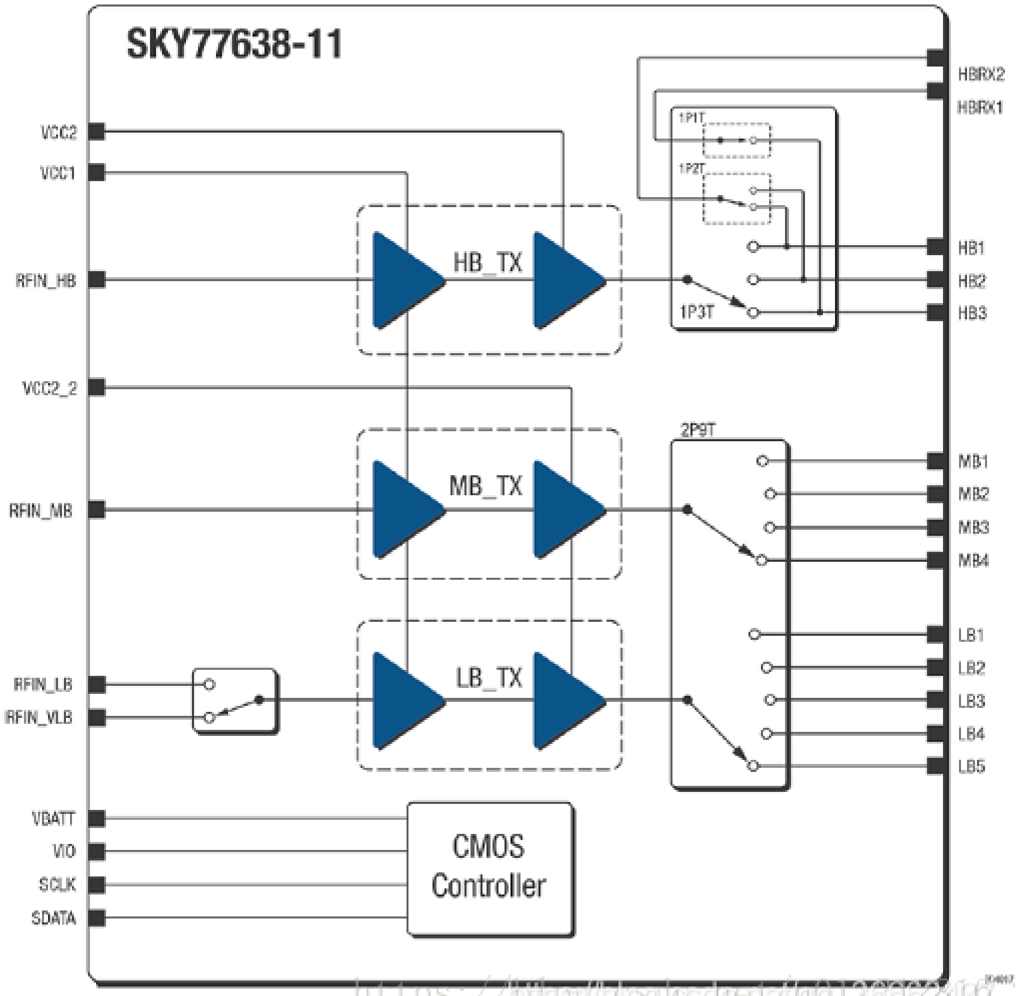


Figure 1. SKY77638-11 Functional Block Diagram

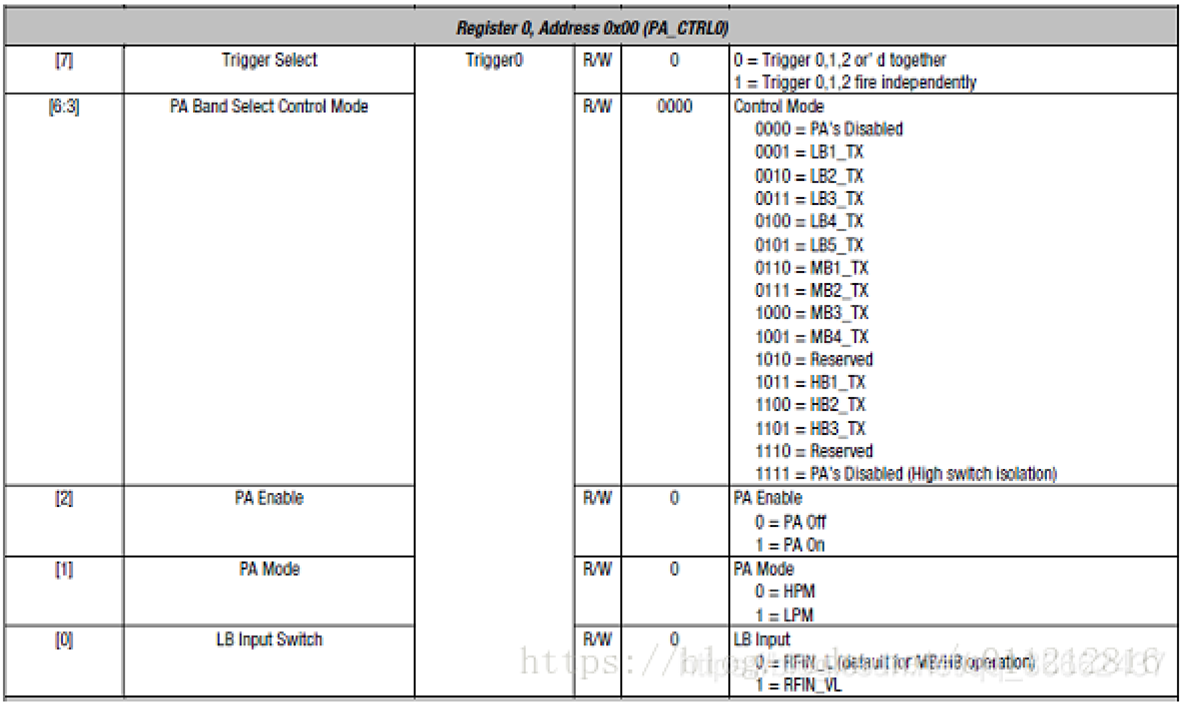
该器件的输入端口主要有RFIN_HB,RFIN_MB,RFIN_LB,RFIN_VLB,输出端口有HB1、HB2、HB3、MB1、MB2、MB3、MB4、LB1、LB2、LB3、LB4、LB5

ASM部分包括HBRX1和HBRX2

寄存器：

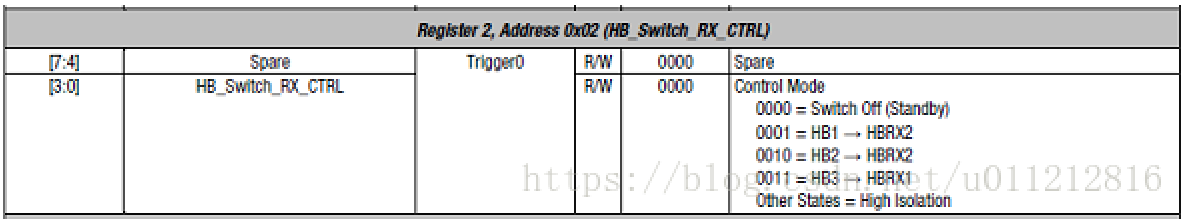
寄存器表比较重要的有：0x00 0x02 0x1d 0x1e 0x1f

0X00：输出端口，输入端口，以及PA模式



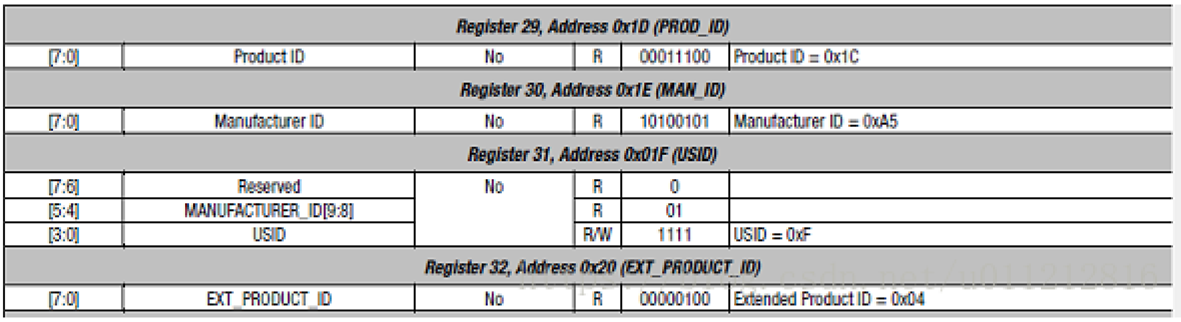
Register 0, Address 0x00 (PA_CTRL0)					
[7]	Trigger Select	Trigger0	R/W	0	0 = Trigger 0,1,2 or' d together 1 = Trigger 0,1,2 fire independently
[6:3]	PA Band Select Control Mode		R/W	0000	Control Mode 0000 = PA's Disabled 0001 = LB1_TX 0010 = LB2_TX 0011 = LB3_TX 0100 = LB4_TX 0101 = LB5_TX 0110 = MB1_TX 0111 = MB2_TX 1000 = MB3_TX 1001 = MB4_TX 1010 = Reserved 1011 = HB1_TX 1100 = HB2_TX 1101 = HB3_TX 1110 = Reserved 1111 = PA's Disabled (High switch isolation)
[2]	PA Enable		R/W	0	PA Enable 0 = PA Off 1 = PA On
[1]	PA Mode		R/W	0	PA Mode 0 = HPM 1 = LPM
[0]	LB Input Switch		R/W	0	LB Input 0 = RFIN_L (default for MB/HB operation) 1 = RFIN_VL

0X02：ASM的端口



Register 2, Address 0x02 (HB_Switch_RX_CTRL)					
[7:4]	Spare	Trigger0	R/W	0000	Spare
[3:0]	HB_Switch_RX_CTRL		R/W	0000	Control Mode 0000 = Switch Off (Standby) 0001 = HB1 → HBRX2 0010 = HB2 → HBRX2 0011 = HB3 → HBRX1 Other States = High Isolation

0X1D 0X1E 0X1F pid mid



Register 29, Address 0x1D (PROD_ID)					
[7:0]	Product ID	No	R	00011100	Product ID = 0x1C
Register 30, Address 0x1E (MAN_ID)					
[7:0]	Manufacturer ID	No	R	10100101	Manufacturer ID = 0xA5
Register 31, Address 0x01F (USID)					
[7:6]	Reserved	No	R	0	
[5:4]	MANUFACTURER_ID[9:8]		R	01	
[3:0]	USID		R/W	1111	USID = 0xF
Register 32, Address 0x20 (EXT_PRODUCT_ID)					
[7:0]	EXT_PRODUCT_ID	No	R	00000100	Extended Product ID = 0x04

看完datasheet，我们需要修改目录下`modem_proc/rfcdevice_pa/src`

1. Create the rfdevice_pa_XXX_data_ag.h.复制已有的文件，修改类的名称
2. Create the rfdevice_pa_XXX_data_ag.cpp
   - 填写PA_SET_BIAS_REG/DATA
   - 填写PA_set_range_REG/DATA
   - 填写PA_ON_REG/DATA
   - 填写PA_OFF_REG/DATA
   - 填写PA_TRIGGER/DATA
   - 填写PID MIDPRO_REV

```
#define RFDEVICE_PA_SKY_XXX_NUM_PORTS 16//查看reg0x00中port口为16（0000-1111）
```

```
#define RFDEVICE_PA_SKY_XXX_PA_SET_BIAS_NUM_REGS 2//reg中有primary bias 和 second bias 在0x01 和0x03，前面没有列出来
static uint8
rfdevice_pa_sky_XXX_pa_set_bias_regs[RFDEVICE_PA_SKY_XXX_PA_SET_BIAS_NUM_REGS] =  {0x01, 0x03};寄存器
static int16
rfdevice_pa_sky_XXX_pa_set_bias_data[RFDEVICE_PA_SKY_XXX_NUM_PORTS][4]
[RFDEVICE_PA_SKY_XXX_PA_SET_BIAS_NUM_REGS] =
{
......具体值没有通用性
};
#define RFDEVICE_PA_SKY_XXX_PA_SET_RANGE_NUM_REGS 1//pa range
static uint8
rfdevice_pa_sky_XXX_pa_set_range_regs[RFDEVICE_PA_SKY_XXXX_PA_SET_RANGE_NUM_REGS] =  {0x00, };
static int16
rfdevice_pa_sky_XXX_pa_set_range_data[RFDEVICE_PA_SKY_XXXX_NUM_PORTS]
[4][RFDEVICE_PA_SKY_XXX_PA_SET_RANGE_NUM_REGS] =
{
  { /* PORT NUM: 0 *//* PA's Disable */
    { 0x00, },  /* PA Range: 0 */HPM
    { 0x00, },  /* PA Range: 1 */LPM
    { 0x00, },  /* PA Range: 2 */LPM
    { 0x00, },  /* PA Range: 3 */LPM
  },
.......
};
#define RFDEVICE_PA_SKY_XXX_PA_ON_NUM_REGS 1//一般不设置
static uint8
rfdevice_pa_sky_XXX_pa_on_regs[RFDEVICE_PA_SKY_XXX_PA_ON_NUM_REGS] =
{RFFE_INVALID_REG_ADDR /*Warning: Not Specified*/, };
static int16
rfdevice_pa_sky_XXX_pa_on_data[RFDEVICE_PA_SKY_XXX_NUM_PORTS]
[RFDEVICE_PA_SKY_XXXPA_ON_NUM_REGS] =
{
  { /* PORT NUM: 0 */
    RF_REG_INVALID,
  },
  .....
};
#define RFDEVICE_PA_SKY_XXX_PA_OFF_NUM_REGS 1
```

```
static uint8
rfdevice_pa_sky_XXX_pa_off_regs[RFDEVICE_PA_SKY_XXX_PA_OFF_NUM_REGS] =
{0x00, };
static int16
rfdevice_pa_sky_XXX_pa_off_data[RFDEVICE_PA_SKY_XXX_NUM_PORTS]
[RFDEVICE_PA_SKY_XXX_PA_OFF_NUM_REGS] =
{
  { /* PORT NUM: 0 */
    0x02, //需要将PA disable PA 改为LOW POWER,所以0X00寄存器设置为2


  },
.........
};
#define RFDEVICE_PA_SKY_XXX_PA_TRIGGER_NUM_REGS 1
static uint8
rfdevice_pa_sky_XXX_pa_trigger_regs[RFDEVICE_PA_SKY_XXX_PA_TRIGGER_NUM_
REGS] =  {0x1C, };
static int16
rfdevice_pa_sky_XXX_pa_trigger_data[RFDEVICE_PA_SKY_XXX_NUM_PORTS]
[RFDEVICE_PA_SKY_XXX_PA_TRIGGER_NUM_REGS] =
{
  { /* PORT NUM: 0 */
    0x07, //默认都打开，所以为7
  },
............
};
boolean rfdevice_pa_sky_XXX_v3_data_ag::device_info_get( rfdevice_pa_info_type
*pa_info )
{
  {
    pa_info->mfg_id = 0x01A5;//MID
    pa_info->prd_id = 0x1C;//PID
    pa_info->prd_rev = 2;//自定义
    pa_info->num_ports = RFDEVICE_PA_SKY_XXX_NUM_PORTS;
    pa_info->num_pa_ranges = 4;
    ret_val = TRUE;


}
```

对以上的代码PA RANGE的配置，值得一提的是，一般虽然设置为4个range，但是其实用不到，具体参考`NV #2029 NV_WCDMA_PA_RANGE_MAP_I`，一般设置为{1, 0, 0, 0}，如下图，所以我们在PA RANGE中其实第一个是HPM，后面是LPM

3. Modify the `rfdevice_pa_factory.cpp` file

```
#include"rfdevice_pa_XXX_data_ag.h"    // 添加器件XYZ的头文件
.....
  else if ( mfg_id ==  0x01A5 && prd_id == 0x1C  &&prd_rev == 0)
  {//填加这款PA的mid pid prd_rev
     pa_data = rfdevice_pa_XXX_data_ag::get_instance();

  }
```

# 如何添加一个新的**MIPI ASM**芯片

mipi asm 配置比pa简单很多，毕竟只是一个单刀多掷的开关，首先依然查看datasheet。

结构图：

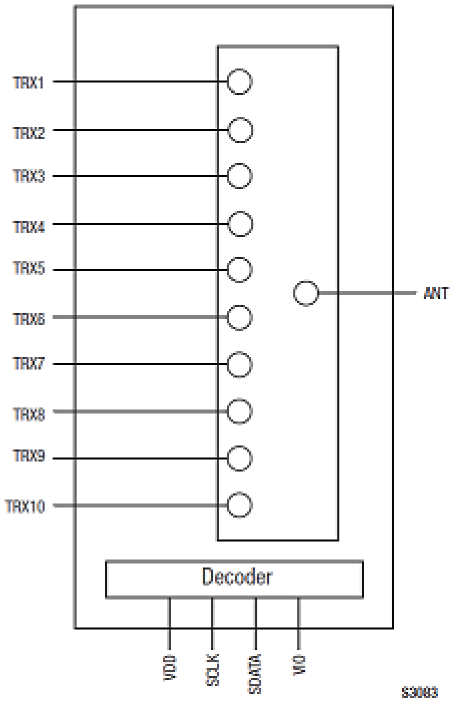
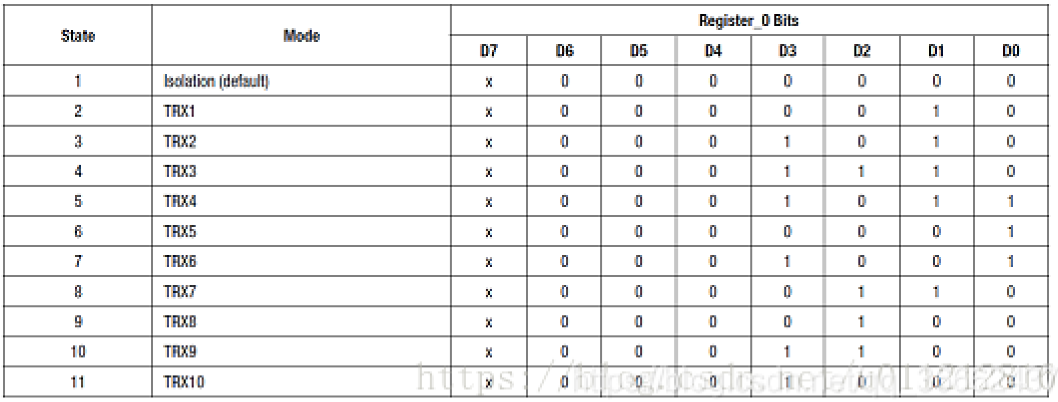


Figure 1. SKY13473–569LF Block Diagram

寄存器：

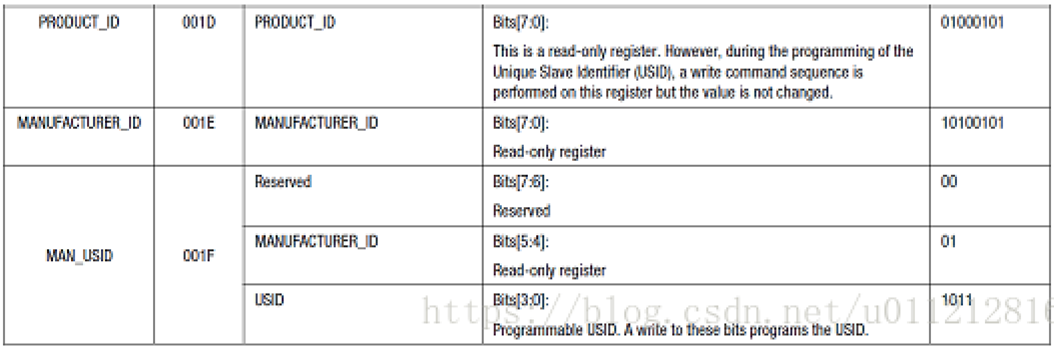
0X00是开关真值表

Table 11. Register_0 Truth Table



State	Mode	Register_0 Bits							
		D7	D6	D5	D4	D3	D2	D1	D0
1	Isolation (default)	x	0	0	0	0	0	0	0
2	TRX1	x	0	0	0	0	0	1	0
3	TRX2	x	0	0	0	1	0	1	0
4	TRX3	x	0	0	0	1	1	1	0
5	TRX4	x	0	0	0	1	0	1	1
6	TRX5	x	0	0	0	0	0	0	1
7	TRX6	x	0	0	0	1	0	0	1
8	TRX7	x	0	0	0	0	1	1	0
9	TRX8	x	0	0	0	0	1	0	0
10	TRX9	x	0	0	0	1	1	0	0
11	TRX10	x	0	0	0	1	0	0	0

001D 001E 001F 依然是 pid mid



PRODUCT_ID	001D	PRODUCT_ID	Bits[7:0]: This is a read-only register. However, during the programming of the Unique Slave Identifier (USID), a write command sequence is performed on this register but the value is not changed.	01000101
MANUFACTURER_ID	001E	MANUFACTURER_ID	Bits[7:0]: Read-only register	10100101
MAN_USID	001F	Reserved	Bits[7:6]: Reserved	00
		MANUFACTURER_ID	Bits[5:4]: Read-only register	01
		USID	Bits[3:0]: Programmable USID. A write to these bits programs the USID.	1011

接下来修改目录下`modem_proc/rfcdevice_asm/src`

1. Create the `rfdevice_asm_XXX_data_ag.h` //复制已有的文件，修改类的名称
2. Create the `rfdevice_asm_XXX_data_ag.cpp` file
    - 填写ASM_ON_REG
    - 填写ASM_OFF_REG
    - 填写ASM_TRIGGER_REG
    - 填写PID MIDPRO_REV

```
#define RFDEVICE_ASM_XXX_NUM_PORTS 11//真值表个数


#define RFDEVICE_ASM_XXX_ASM_ON_NUM_REGS 1
static uint8
rfdevice_asm_XXX_asm_on_regs[RFDEVICE_ASM_XXX_ASM_ON_NUM_REGS] =
{0x00, };//寄存器地址
static int16 rfdevice_asm_XXX_asm_on_data[RFDEVICE_ASM_XXX_NUM_PORTS]
[RFDEVICE_ASM_XXX_ASM_ON_NUM_REGS] =
{
  { /* PORT NUM: 0 *//* Isolation*/
    0x00,
  },
  { /* PORT NUM: 1 *//* TRX1 */查看真值表
    0x02,
  },
  { /* PORT NUM: 2 *//* TRX2 */
    0x0A,
  },
  { /* PORT NUM: 3 *//* TRX3 */
    0x0E,
  },
.....
};


#define RFDEVICE_ASM_XXX_ASM_OFF_NUM_REGS 1
static uint8
rfdevice_asm_XXX_asm_off_regs[RFDEVICE_ASM_XXX_ASM_OFF_NUM_REGS] =
{0x00, };
static int16 rfdevice_asm_XXX_asm_off_data[RFDEVICE_ASM_SXXX_NUM_PORTS]
[RFDEVICE_ASM_XXX_ASM_OFF_NUM_REGS] =
{
  { /* PORT NUM: 0 */
    0x00, //第一个port为关
  },
  ....
};


#define RFDEVICE_ASM_XXX_ASM_TRIGGER_NUM_REGS 1
static uint8
rfdevice_asm_XXX_asm_trigger_regs[RFDEVICE_ASM_XXX_ASM_TRIGGER_NUM_R
EGS] =  {0x1C, };//trigger寄存器
static int16
rfdevice_asm_XXX_asm_trigger_data[RFDEVICE_ASM_SXXX_NUM_PORTS]
[RFDEVICE_ASM_XXX_ASM_TRIGGER_NUM_REGS] =
{
  { /* PORT NUM: 0 */
    0x07, //默认为7
  },
......
};
boolean rfdevice_asm_sky_XXX_data_ag::device_info_get( rfdevice_asm_info_type
*asm_info )
{
    asm_info->mfg_id = 0x1A5;//PID
    asm_info->prd_id = 0x45;//MID
    asm_info->prd_rev = 0;
    asm_info->num_ports = RFDEVICE_ASM_XXX_NUM_PORTS;
    ret_val = TRUE;
}
3、Modify the rfdevice_asm_factory.cpp file

    #include"rfdevice_asm_XXX_data_ag.h"     // 添加器件XYZ的头文件
.....
  else if ( mfg_id ==  0x01A5 && prd_id == 0x1C  &&prd_rev == 0)
    {//填加这款ASM的mid pid prd_rev
       pa_data = rfdevice_asm_XXX_data_ag::get_instance();

    }
```

# SKY到WTR的通路

寄存器值表格用来决定左边的硬件电路连接图用哪个来作为输入，在代码中的表示形式如下：

代码所在文件

`\modem_proc\rfdevice_asm\src\rfdevice_asm_sky13455_data_ag.cpp`

# **asm(Antenna switch matrix)**即天线开关模块

sky13455部分代码如下：

```
#define RFDEVICE_ASM_SKY13455_ASM_ON_NUM_REGS 1

static uint8
rfdevice_asm_sky13455_asm_on_regs[RFDEVICE_ASM_SKY13455_ASM_ON_NUM_REGS] = {0x00, };

static int16
rfdevice_asm_sky13455_asm_on_data[RFDEVICE_ASM_SKY13455_NUM_PORTS]
[RFDEVICE_ASM_SKY13455_ASM_ON_NUM_REGS] =

{

{ /* PORT NUM: 0 */

0x0A, 此值与寄存器值表格中的值相对应，比如 LTX 对应的值为 x0001010=0x0A
```

```
},
{ /* PORT NUM: 1 */
0x08,
},
{ /* PORT NUM: 2 */
0x04,
},
{ /* PORT NUM: 3 */
0x05,
},
{ /* PORT NUM: 4 */
0x06,
},
{ /* PORT NUM: 5 */
0x07,
},
{ /* PORT NUM: 6 */
0x09,
},
{ /* PORT NUM: 7 */
0x0B,
},

{ /* PORT NUM: 8 */

0x0C,

},

{ /* PORT NUM: 9 */

0x01,

},

{ /* PORT NUM: 10 */

0x02,

},

{ /* PORT NUM: 11 */

0x03,

},

};
```

RF card部分的代码会用到以上port，代码如下

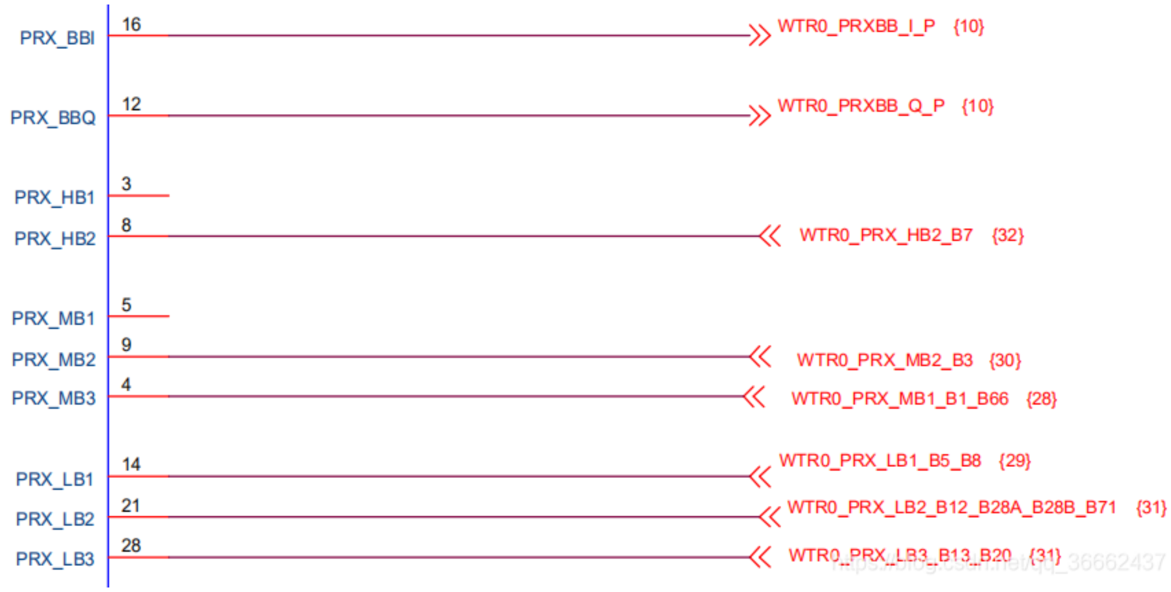
`\modem_proc\rfc_jolokia\rf_card\rfc_wtr4905_om\lte\src\rfc_wtr4905_om_lte_config_data_ag.c`

```
rfc_device_info_type rf_card_wtr4905_om_rx0_lte_b1_device_info =

{

RFC_ENCODED_REVISION,

RFC_RX_MODEM_CHAIN_0, /* Modem Chain */

0, /* NV Container */
```

```
RFC_INVALID_PARAM /* Warning: Not Specified */, /* Antenna */

2, /* NUM_DEVICES_TO_CONFIGURE */

{

{

RFDEVICE_TRANSCEIVER,

WTR4905, /* NAME */

0, /* DEVICE_MODULE_TYPE_INSTANCE */

0, /* PHY_PATH_NUM */

{

0 /*Warning: Not specified*/, /* INTF_REV */

(int)WTR4905_LTEFDD_PRXLGY1_BAND1_PMB2, /* PORT */ 此PORT是MSM处理器端的接口，具体选取见附录1分析

( RFDEVICE_PA_LUT_MAPPING_INVALID ), /* RF_ASIC_BAND_AGC_LUT_MAPPING */

FALSE, /* TXAGC_LUT */

WTR4905_FBRX_ATTN_DEFAULT, /* FBRX_ATTN_STATE */

0, /* Array Filler */

},

},

{

RFDEVICE_ASM,

GEN_ASM, /* NAME */
0, /* DEVICE_MODULE_TYPE_INSTANCE */

0 /*Warning: Not specified*/, /* PHY_PATH_NUM */

{

0 /* Orig setting: */, /* INTF_REV */

(0x01A5 << 22)/*mfg_id*/ | (0x41 << 14)/*prd_id*/ | (5)/*port_num*/, /* PORT_NUM */ //此处的port num会对应以上的rfdevice_asm_sky13455_asm_on_data[PORT_NUM]数组成员，在数组中用红色字体来表示

0, /* Array Filler */

0, /* Array Filler */

0, /* Array Filler */

0, /* Array Filler */

},

},

},

};
```

附录

以下步骤跟据硬件给的图来决定



WTR4905_WCDMA_DRXLGY1_BAND8_PLB1,//band8， 主接收， 对应的LB1。

- 先决定是哪个BAND //决定于下表右方的数字
- 再决定是否为主接受
  - D代表rx1，副接收，用rx1表示 如：rf_card_wtr4905_om_rx1_wcdma_b8_device_info //对应硬件文档的RF port部分的DRX_xBx
  - P代表rx0表示主接收 用rx0表示 如：rf_card_wtr4905_om_rx0_wcdma_b8_device_info //对应硬件文档的RF port部分的PRX_xBx,

# 下边是MSM跟WTR的通路

```
rfc_signal_info_type
rfc_wtr4905_china_ct_4m_sig_info[RFC_WTR4905_CHINA_CT_4M_SIG_NUM + 1] =


{


{ RFC_MSM_TIMING_PA_CTL , RFC_LOW, DAL_GPIO_NO_PULL, DAL_GPIO_2MA,
(DALGpioIdType)NULL }, /* RFC_WTR4905_CHINA_CT_4M_TIMING_PA_CTL*/


{ RFC_MSM_TIMING_PA_RANGE , RFC_LOW, DAL_GPIO_NO_PULL,
DAL_GPIO_2MA, (DALGpioIdType)NULL }, /*
RFC_WTR4905_CHINA_CT_4M_TIMING_PA_RANGE */


{ RFC_MSM_TIMING_ASM_CTL , RFC_LOW, DAL_GPIO_NO_PULL, DAL_GPIO_2MA,
(DALGpioIdType)NULL }, /* RFC_WTR4905_CHINA_CT_4M_TIMING_ASM_CTL */
{ RFC_MSM_TIMING_PAPM_CTL , RFC_LOW, DAL_GPIO_NO_PULL,
DAL_GPIO_2MA, (DALGpioIdType)NULL }, /*
RFC_WTR4905_CHINA_CT_4M_TIMING_PAPM_CTL */

{ RFC_MSM_TIMING_TX_TX_RF_ON0, RFC_LOW, DAL_GPIO_NO_PULL,
DAL_GPIO_2MA, (DALGpioIdType)NULL}, /*
RFC_WTR4905_CHINA_CT_4M_TIMING_TX_TX_RF_ON0 */

.......红色部分代表WTR和MSM的连接，此定义在WTR相关的文件中，这里
RFC_MSM_TIMING_xx_xx就是MSM的一个引脚



};
```

以上结构体于下边结构体对应，后边用来控制管脚的初始化

```
typedef enum

{

RFC_WTR4905_OM_TIMING_PA_CTL, 你看看，顺序都一样，指的肯定是同一个引脚
了，下边的程序中会用到

RFC_WTR4905_OM_TIMING_PA_RANGE,

RFC_WTR4905_OM_TIMING_ASM_CTL,

RFC_WTR4905_OM_TIMING_PAPM_CTL,

RFC_WTR4905_OM_TIMING_TX_TX_RF_ON0,

RFC_WTR4905_OM_TIMING_TX_RX_RF_ON0,



rfc_sig_info_type rf_card_wtr4905_om_tx0_lte_b1_sig_cfg =

{

RFC_ENCODED_REVISION,

{
```

```
{ (int)RFC_WTR4905_OM_PA0_R0_ALT1, { RFC_CONFIG_ONLY, 0 }, {RFC_LOW, 0 } }, 这里用到上边结构体里定义的引脚，这里用来初始化引脚

{ (int)RFC_WTR4905_OM_PA0_R1_ALT1, { RFC_CONFIG_ONLY, 0 }, {RFC_LOW, 0 } },

{ (int)RFC_SIG_LIST_END, { RFC_LOW, 0 }, {RFC_LOW, 0 } }

}, 因为选定特定的band的话，需要有些引脚需要初始化为特定电平，这个结构体来包含相应band情况下所有需要初始的引脚

};
```

至此，软件的通路就打通了