

IDE:

创建工程

打开 Eclipse, 选择菜单 “File/New”, 然后选择 “RDA Project”。项目配置窗口如下, 自定义工程名, 选择项目代码路径:

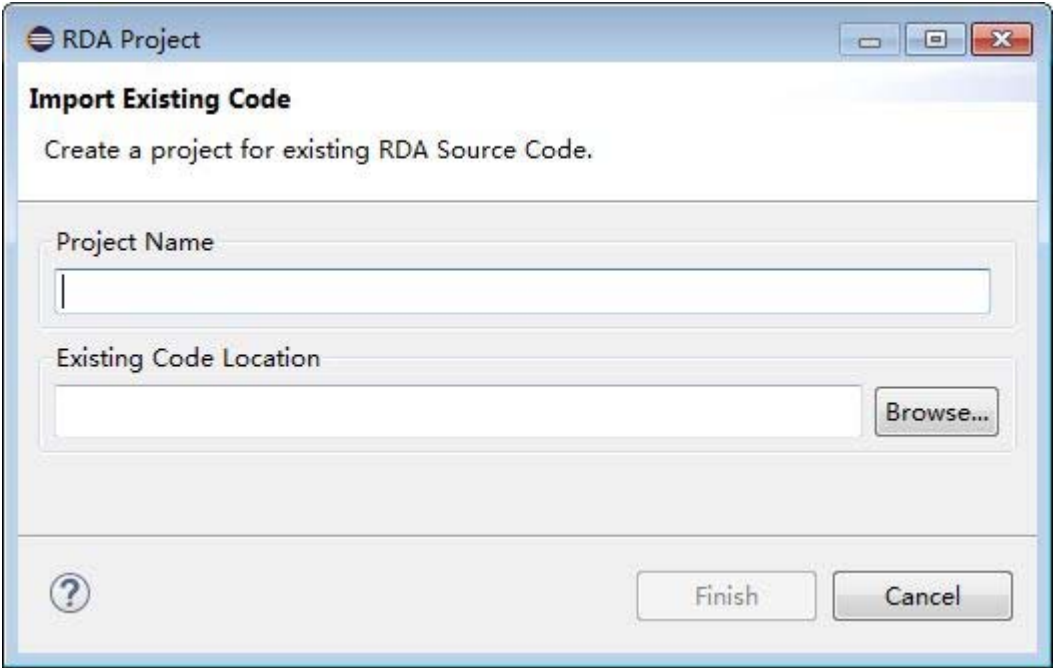


图 2-1 创建项目

项目基础代码由 RDA 提供, 用户可以在基础代码之上实现自己特有的功能和应用。文档中示例工程名为 “iot”, 选择代码路径后单击 “Finish” 按钮, 生成工程如下:

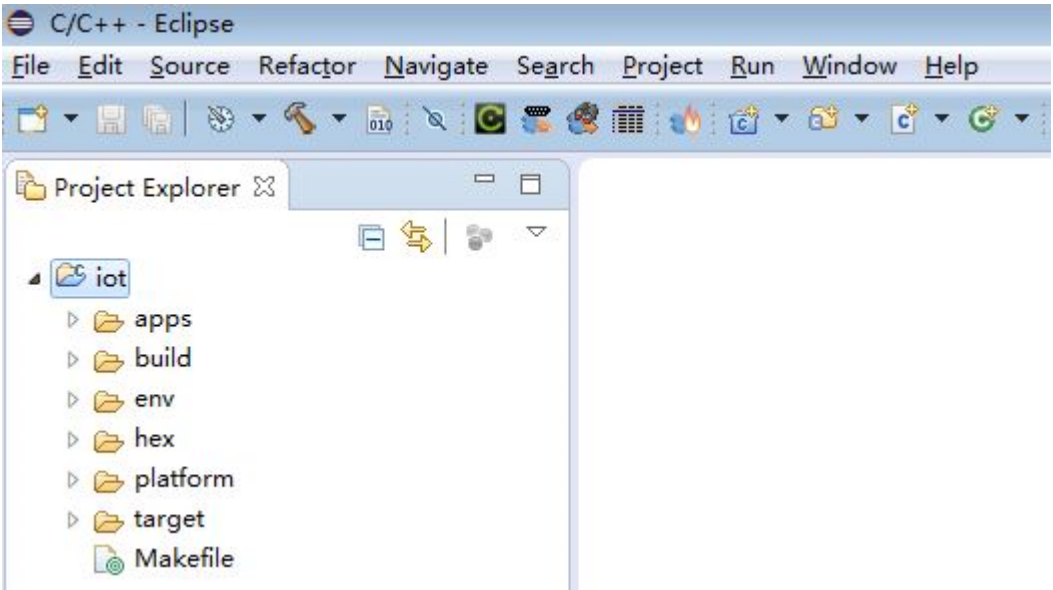


图 2-2 项目浏览器

配置工程

在“Project Explorer”中点击右键，选择“Properties”，进入“Properties”窗口，选择“C/C++ Build”，然后选择“RDA Project”，配置目标：

Select Target	选择目标 Target
Target Wizard	项目向导，配置项目特性、模块等
Remove	删除 Target
New Target	新建 Target
Select Release	选择生成“release”还是“debug”版本
Base Build Command	-
Custom Build Option	-

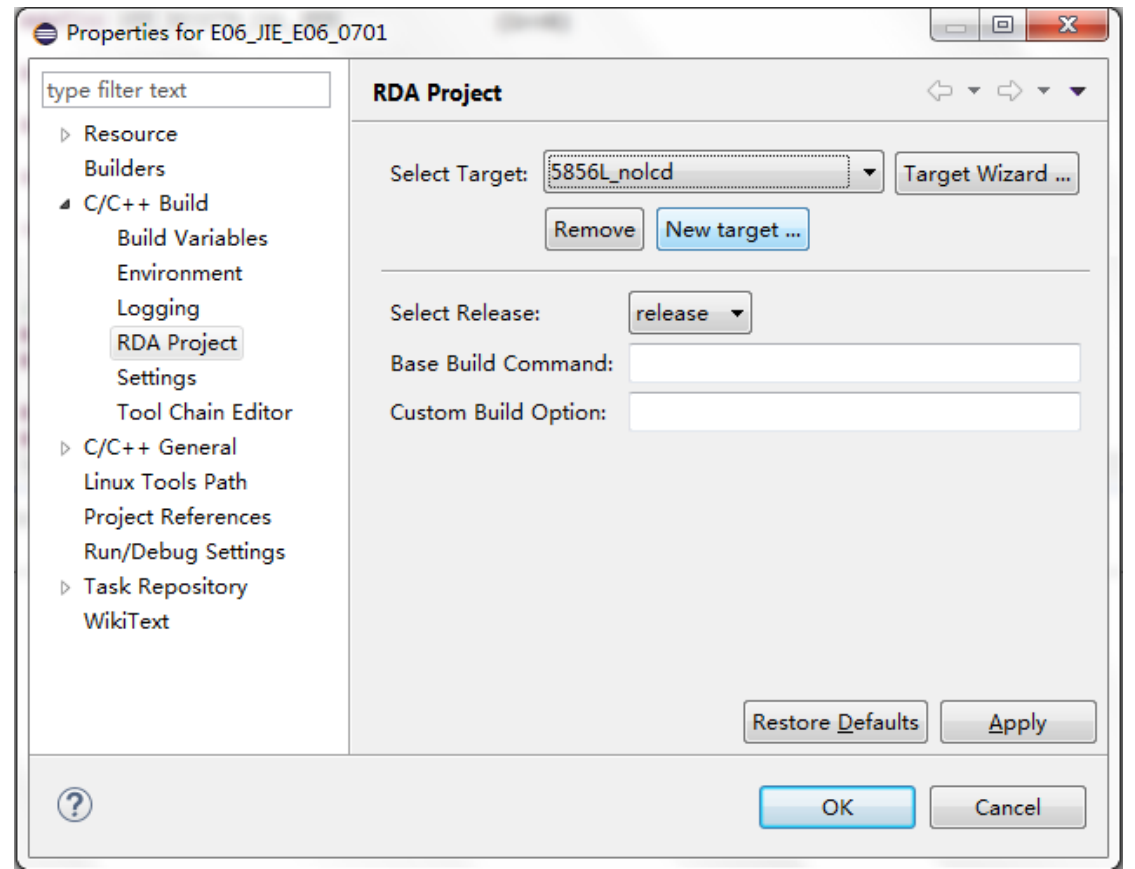


图 2-3 项目配置

备注：5856T: 24PIN 5856L:48PIN 5856Q:64PIN

编译工程

在“Project Explorer”中点击右键，选择“RDA Tools”，然后可以看到两个编译选项：

Build Flash	编译 Ramrun 文件
Build Image	编译 Flash 文件

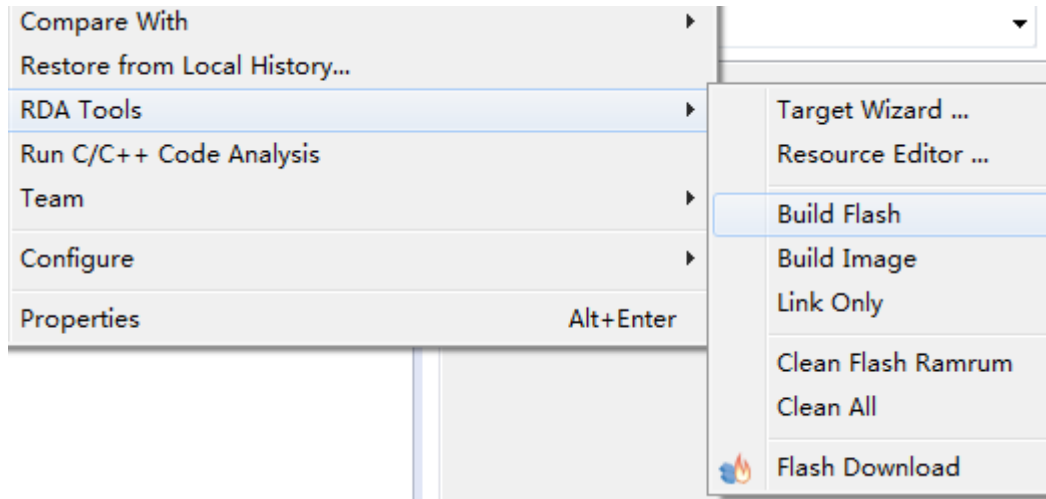
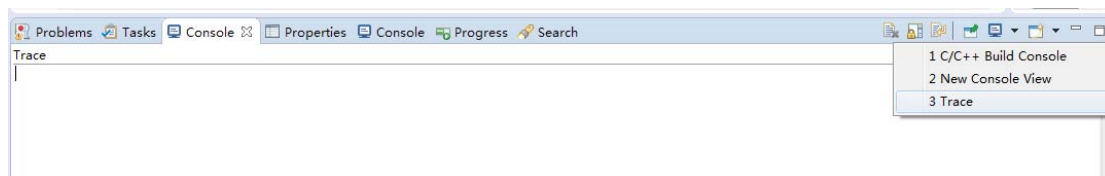


图 2-4 编译项目

打开 TRACE



Trace 选择:

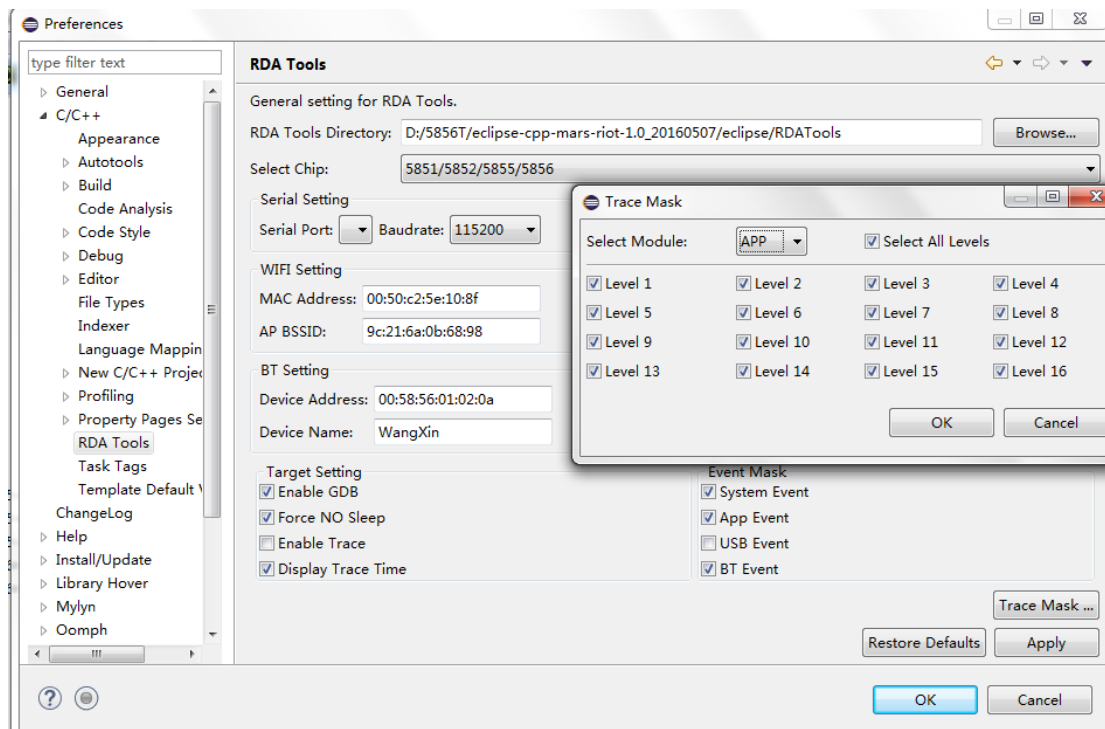
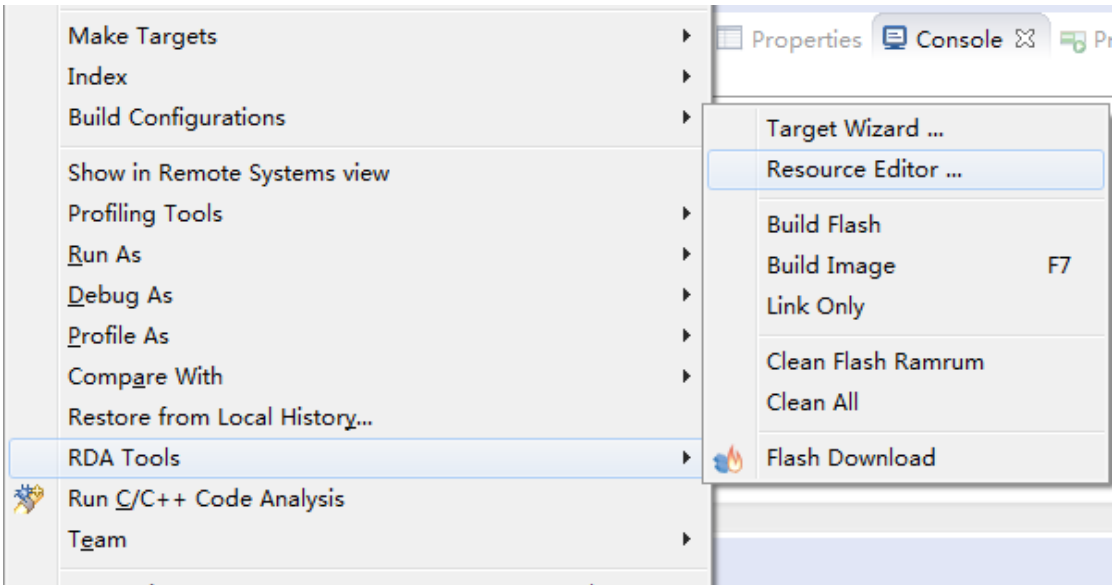


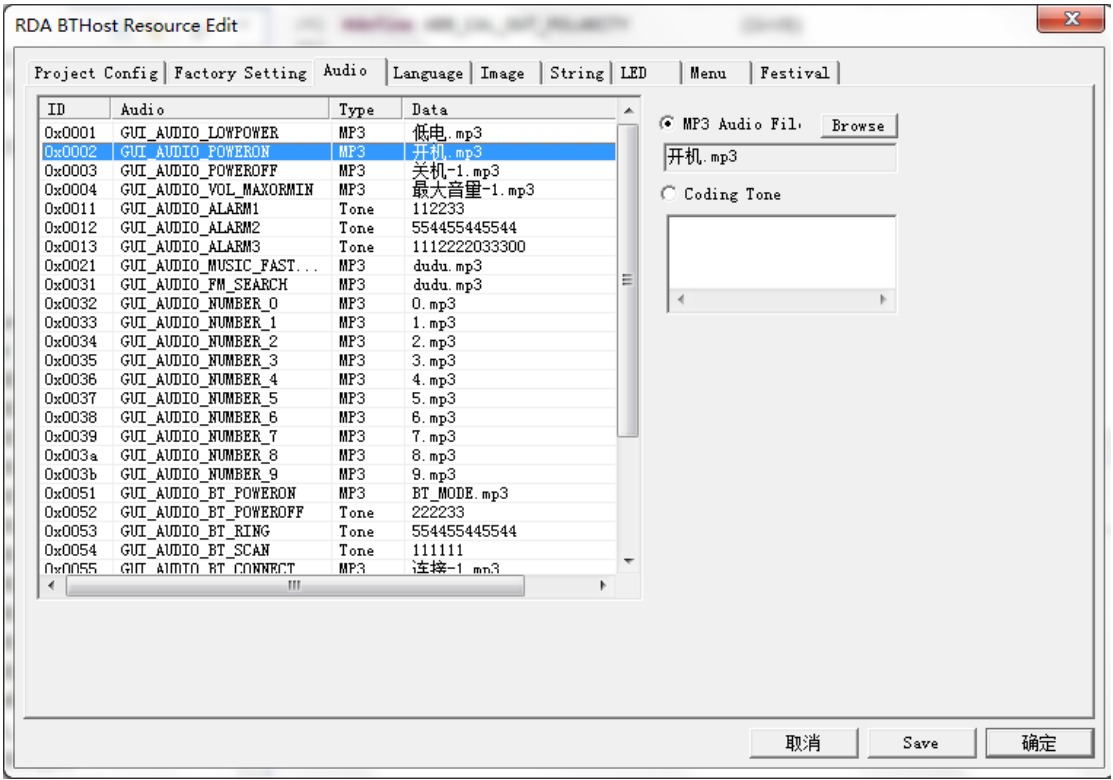
图 2-5 打开 trace

提示音

在“Project Explorer”中点击右键，选择“RDA Tools”，可以看到 Resource Editor 选项：



接着选择 Audio（目前支持 MP3 和 tone 音），最后在代码选中播放相应的音频 ID 号即可。

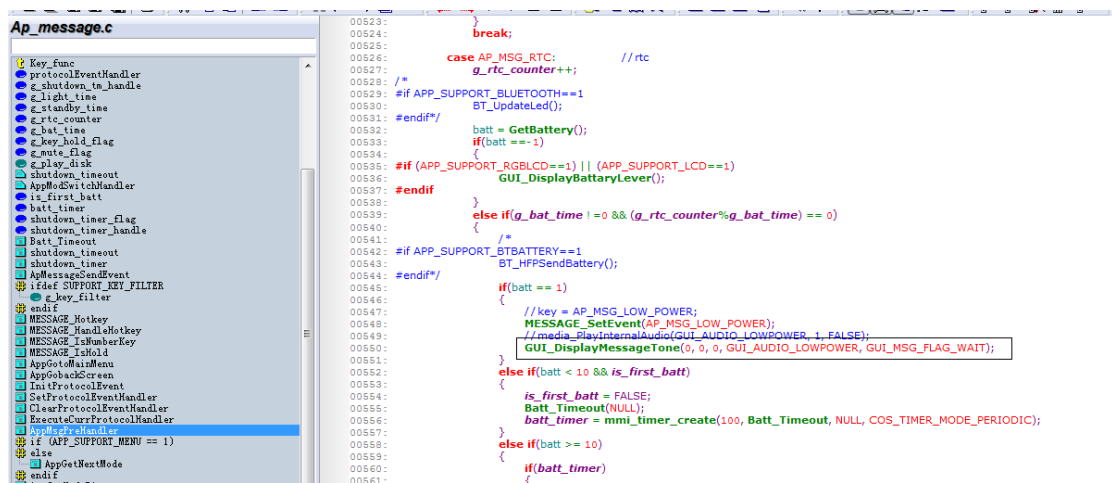


如果需要添加新的音频，可以在 resource_id.h 里添加新的 ID，然后重新打开 Resource Editor，再指定下目标音频就可以了。

一般加提示音有两种方式，一个是调用 `media_PlayInternalAudio()` 函数，这种方式是先播，然后等 `FINISH` 的标志（此时是没有其他音频在播放）

```
00516: *****/
00517: BOOL AppPowerOnMsgHandler(COS_EVENT *pEvent)
00518: {
00519:     APP_ASSERT(pEvent);
00520:
00521:     switch(pEvent->nEventId)
00522:     {
00523:         case EV_UI_FW_ON_START:
00524:             if(AP_Support_LCD())
00525:             {
00526:                 GUI_ClearScreen(NULL);
00527:                 GUI_UpdateScreen(NULL);
00528:             }
00529:             #if APP_SUPPORT_LED==1
00530:                 LED_SetPattern(GUI_LED_POWERON, 1);
00531:             #endif
00532:             media_PlayInternalAudio(GUI_AUDIO_POWERON, 1, 0);
00533:             break;
00534:
00535:         case EV_ADUIO_INTERNAL_PLAY_FINISH_IND:
00536:             if((g_displayconfig.log_image_time | g_displayconfig.log_image_count) == 0)
00537:             {
00538:                 // mmi_timer_create(1, GobackAllHistory,NULL,COS_TIMER_MODE_SINGLE);
00539:                 GobackAllHistory();
00540:             }
00541:             else
00542:             {
00543:                 APP_DisplayLogo_TimeOutHandler(NULL);
00544:             }
00545:             break;
00546:
```

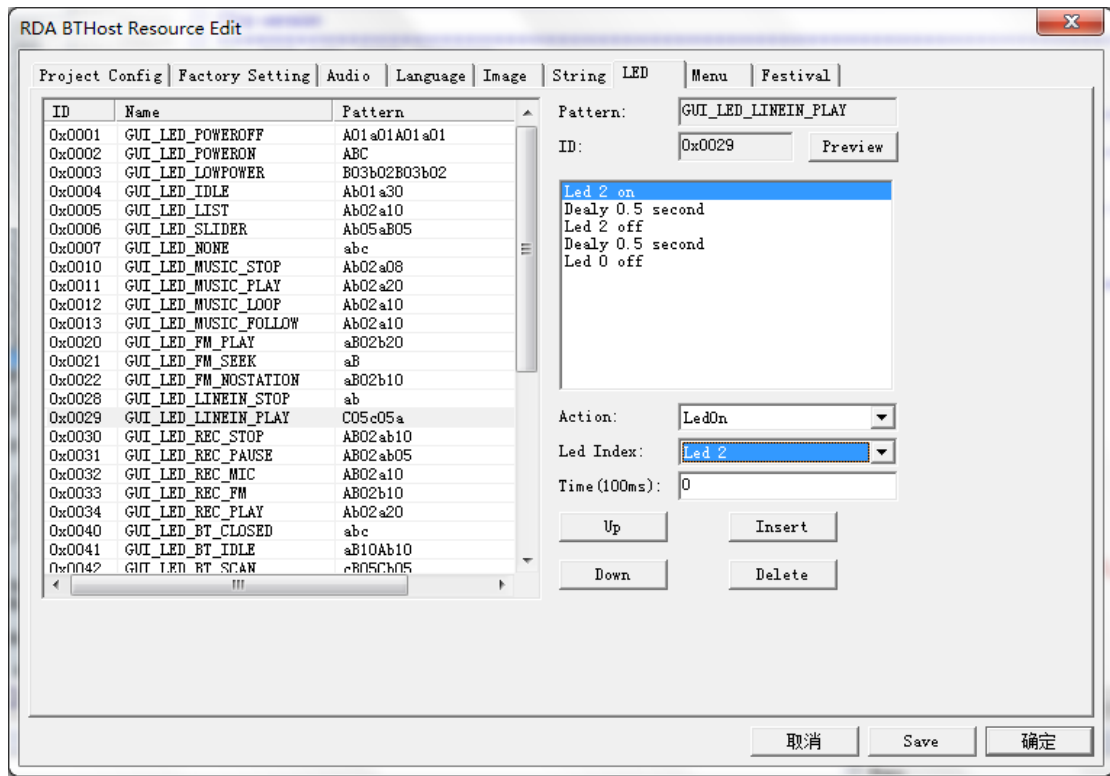
另外一个 是调用 `GUI_DisplayMessageTone()`，这个是会暂停当前的音频，先播放提示音，播完提示音再重新播放之前的音频（会先发 `pause` 消息，等提示音播完，再发 `resume` 消息）。



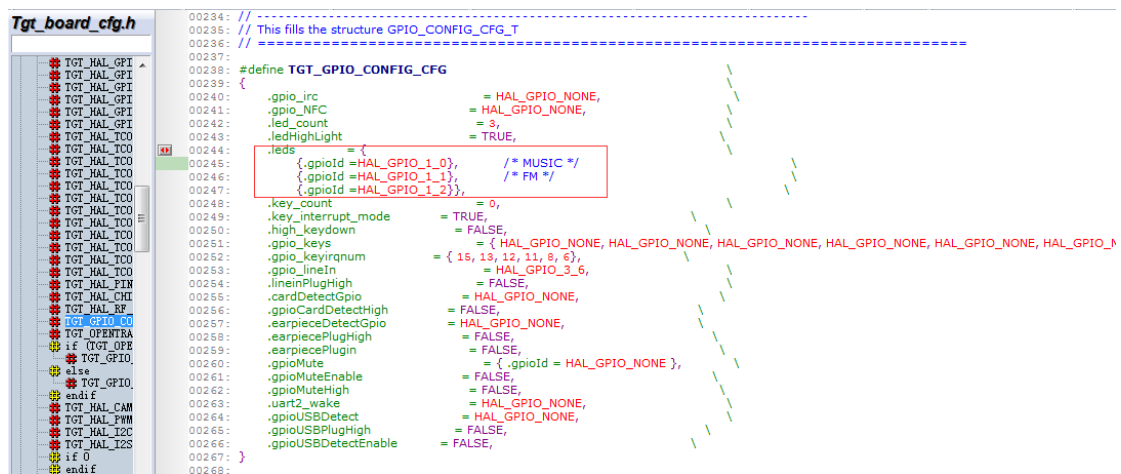
Led

在“Project Explorer”中点击右键，选择“RDA Tools”，可以看到 `Resource Editor` 选项，再选择 `LED` 资源，就可以编辑 `LED` 状态了。

通过 `Insert` 可以添加新的 `action`，和添加需要的 `delay`。



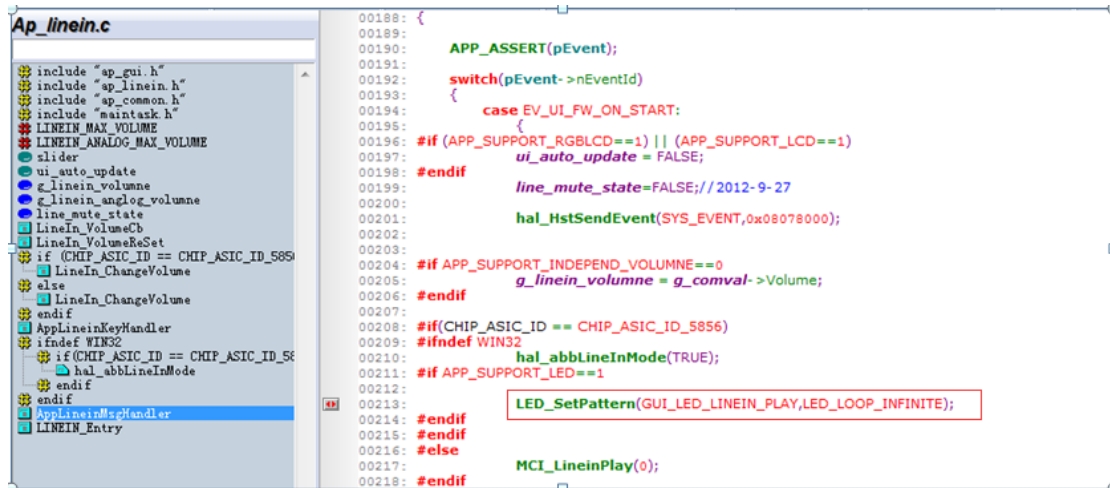
Led Index 关系，就是对应下图定义的顺序。P10 对应 led0，P11 对应 led1，P12 对应 led2.



另外需要注意下相应的 IO 必须配成 output，还有 cfg 必须配成 0.

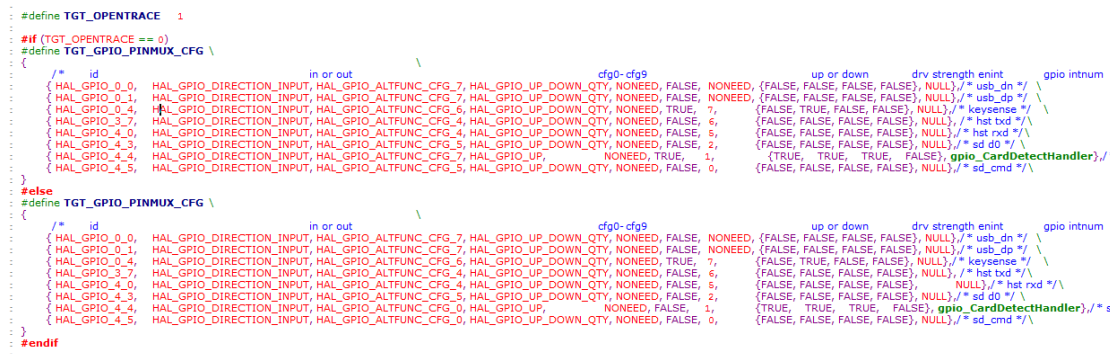
```
#define TGT_GPIO_PINMUX_CFG \
{
    /* id in or out \ cfg0-cfg9 up or down
    { HAL_GPIO_0_0, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_7, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FALS
    { HAL_GPIO_0_1, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_7, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FALS
    { HAL_GPIO_0_4, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_6, HAL_GPIO_UP_DOWN_QTY, NONEED, TRUE, 7, {FALSE, TRUE, FALSE, FALSE
    { HAL_GPIO_1_0, HAL_GPIO_DIRECTION_OUTPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 0, {TRUE, TR
    { HAL_GPIO_1_1, HAL_GPIO_DIRECTION_OUTPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 15, {TRUE, TR
    { HAL_GPIO_1_2, HAL_GPIO_DIRECTION_OUTPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FAI
    { HAL_GPIO_1_3, HAL_GPIO_DIRECTION_OUTPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 14, {FALSE, FALSE
    { HAL_GPIO_1_4, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_3, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FALS
    { HAL_GPIO_2_0, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 10, {FALSE, FALSE, FA
    { HAL_GPIO_3_0, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_3, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FALS
    { HAL_GPIO_3_1, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_3, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FALS
    { HAL_GPIO_3_2, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_3, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FALS
    { HAL_GPIO_3_3, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_3, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FALS
    { HAL_GPIO_3_4, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_3, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 9, {FALSE, FALSE, FALS
    { HAL_GPIO_3_5, HAL_GPIO_DIRECTION_OUTPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FAI
    { HAL_GPIO_3_6, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_3, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 8, {FALSE, FALSE
    { HAL_GPIO_3_7, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 6, {FALSE, FALSE
    { HAL_GPIO_4_0, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 5, {FALSE, FALSE
    { HAL_GPIO_4_1, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_4, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 4, {FALSE, FALSE
    { HAL_GPIO_4_2, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_4, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 3, {FALSE, FALSE
    { HAL_GPIO_4_3, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_5, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 2, {FALSE, FALSE
    { HAL_GPIO_4_4, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_5, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 1, {FALSE, FALSE
    { HAL_GPIO_4_5, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_5, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 0, {FALSE, FALSE,
}
}
```


调用方法:



GPIO

现在 GPIO 的 pinmux 配置都在 tgt_board_cfg.h 里。



一般设为 IO，配成 CFG0，设为外部中断，配成 CFG7。

typedef struct

```
{
    HAL_GPIO_GPIO_ID_T id;
    HAL_GPIO_DIRECTION_T direction;
    HAL_GPIO_ALTFUNC_CFG_ID_T altcfg;
    HAL_GPIO_UP_DOWN_ID_T updown;
    UINT8 drv;
    UINT8 enint; /* enable interrupt or not */
    UINT8 gpio_intrnum; /* gpio interrupt num */
    HAL_GPIO_IRQ_MASK_T irqMask;
    HAL_GPIO_IRQ_HANDLER_T irqHandler;
}BoardSetup_GPIO_CFG;
```

typedef struct

```
{
    /// Defines whether an interruption will be triggered on a rising
```



```

    /// edge on the GPIO
    BOOL rising;

    /// Defines whether an interruption will be triggered on a falling
    /// edge on the GPIO
    BOOL falling;

    /// Defines if the GPIO's signal will be debounced before the interrupt
    /// is triggered
    BOOL debounce;

    /// Defines if the interruption is on level (\c TRUE), or on edge (\c FALSE)
    BOOL level;
} HAL_GPIO_IRQ_MASK_T;

```

比较特殊的是 P37 和 P40，默认是内部 I2C 的功能，要配成外部功能，必须先使能外部 IO 功能，（下图第一条语句）然后再配成所需要的 CFG。CFG0 是 gpio 功能，CFG4 是配成 trace 功能。

```

PUBLIC VOID pmd_I2C_ExtEn(void)
{
    pmd_RDAWrite(RDA_ADDR_SYS_CTRL, pmd_RDARRead(RDA_ADDR_SYS_CTRL)&(~RDA_PMU_SYS_CTRL_I2C_EXT_EN)); // i2c_ext_en
    hal_GpioSetPinMux(HAL_GPIO_3_7, HAL_GPIO_ALTFUNC_CFG_0);
    hal_GpioSetPinMux(HAL_GPIO_4_0, HAL_GPIO_ALTFUNC_CFG_0);
}

```

红外

红外的 pin 脚是 P10，目前仅 5856L, 5856Q 支持，5856T 不支持。

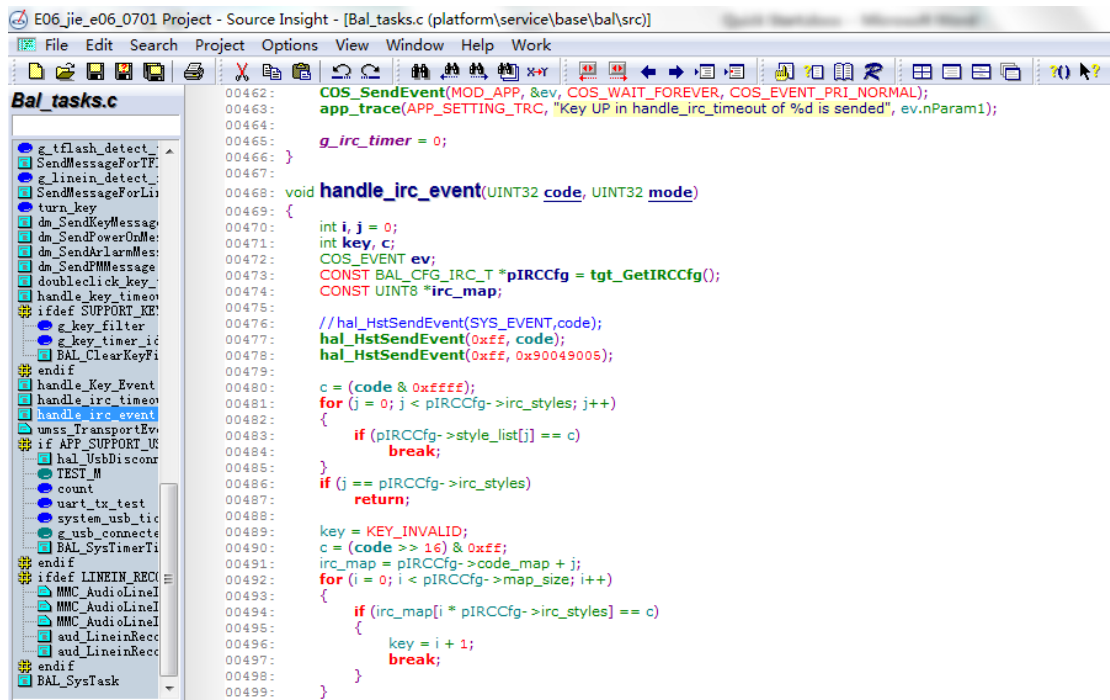
```

#if (TGT_OPENTRACE == 0)
#define TGT_GPIO_PINMUX_CFG \
{
    /* id in or out cfg0-cfg9 up or dov
    { HAL_GPIO_0_0, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_7, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FAL
    { HAL_GPIO_0_1, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_7, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FAL
    { HAL_GPIO_0_4, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_6, HAL_GPIO_UP_DOWN_QTY, NONEED, TRUE, 7, {FALSE, TRUE, FALSE, FAL
    { HAL_GPIO_1_0, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_3, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 0, {TRUE, TI
    { HAL_GPIO_1_1, HAL_GPIO_DIRECTION_OUTPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 15, {TRUE, TI
    { HAL_GPIO_1_2, HAL_GPIO_DIRECTION_OUTPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, F
    { HAL_GPIO_1_3, HAL_GPIO_DIRECTION_OUTPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 14, {FALSE, FAL
    { HAL_GPIO_1_4, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FAL
    { HAL_GPIO_2_0, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 10, {FALSE, FALSE,
    { HAL_GPIO_3_0, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_3, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FAL
    { HAL_GPIO_3_1, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_3, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FAL
    { HAL_GPIO_3_2, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_3, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FAL
    { HAL_GPIO_3_3, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_3, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FAL
    { HAL_GPIO_3_4, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_3, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, FAL
    { HAL_GPIO_3_5, HAL_GPIO_DIRECTION_OUTPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, NONEED, {FALSE, FALSE, F
    { HAL_GPIO_3_6, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_3, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 8, {FALSE, FAL
    { HAL_GPIO_3_7, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 6, {FALSE, FAL
    { HAL_GPIO_4_0, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_0, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 5, {FALSE, FALSI
    { HAL_GPIO_4_1, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_4, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 4, {FALSE, FALSI
    { HAL_GPIO_4_2, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_4, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 3, {FALSE, FALSI
    { HAL_GPIO_4_3, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_5, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 2, {FALSE, FALSI
    { HAL_GPIO_4_4, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_5, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 1, {FALSE, FALSI
    { HAL_GPIO_4_5, HAL_GPIO_DIRECTION_INPUT, HAL_GPIO_ALTFUNC_CFG_5, HAL_GPIO_UP_DOWN_QTY, NONEED, FALSE, 0, {FALSE, FALSI
}

```

Pinmux 设成 5 即可。

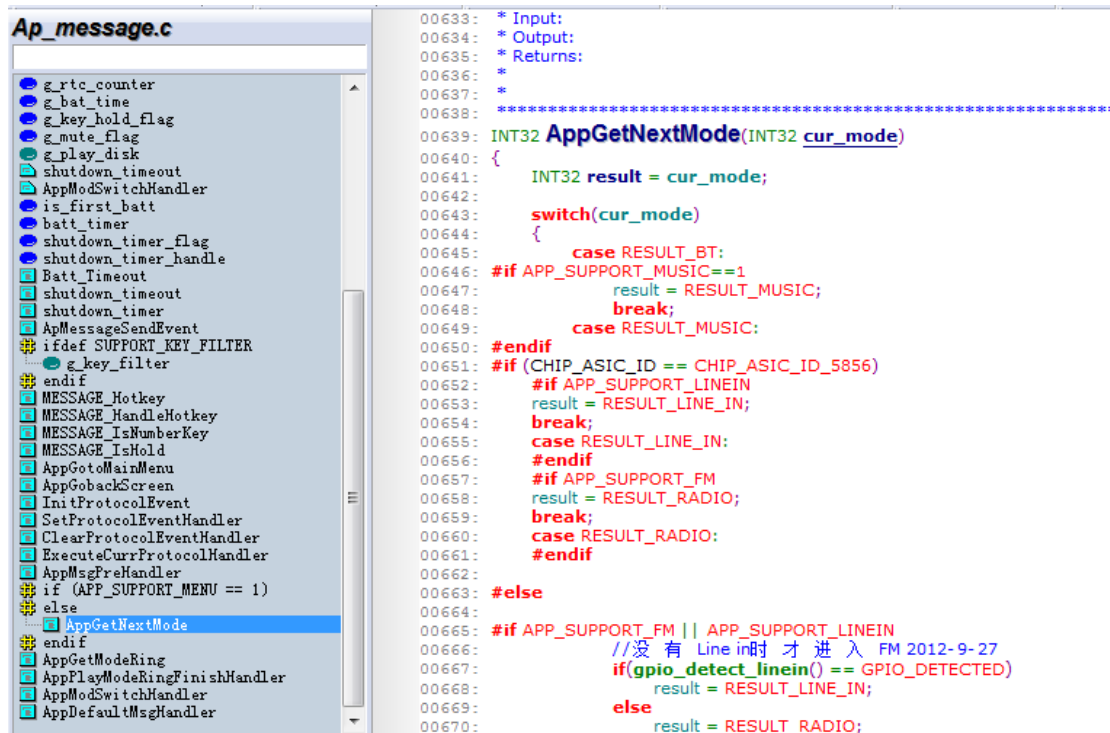
红外的处理函数是 `hand_irc_event()`。



```
00462: COS_SendEvent(MOD_APP, &ev, COS_WAIT_FOREVER, COS_EVENT_PRI_NORMAL);
00463: app_trace(APP_SETTING_TRC, "Key UP in handle_irc_timeout of %d is send", ev.nParam1);
00464:
00465: g_irc_timer = 0;
00466: }
00467:
00468: void handle_irc_event(UINT32 code, UINT32 mode)
00469: {
00470:     int i, j = 0;
00471:     int key, c;
00472:     COS_EVENT ev;
00473:     CONST BAL_CFG_IRC_T *pIRCCfg = tgt_GetIRCCfg();
00474:     CONST UINT8 *irc_map;
00475:
00476:     // hal_HstSendEvent(SYS_EVENT, code);
00477:     hal_HstSendEvent(0x11, code);
00478:     hal_HstSendEvent(0x11, 0x90049005);
00479:
00480:     c = (code & 0xffff);
00481:     for (j = 0; j < pIRCCfg->irc_styles; j++)
00482:     {
00483:         if (pIRCCfg->style_list[j] == c)
00484:             break;
00485:     }
00486:     if (j == pIRCCfg->irc_styles)
00487:         return;
00488:
00489:     key = KEY_INVALID;
00490:     c = (code >> 16) & 0xff;
00491:     irc_map = pIRCCfg->code_map + j;
00492:     for (i = 0; i < pIRCCfg->map_size; i++)
00493:     {
00494:         if (irc_map[i] * pIRCCfg->irc_styles == c)
00495:         {
00496:             key = i + 1;
00497:             break;
00498:         }
00499:     }
00499: }
```

模式切换的顺序

主要在 AppGetNextMode 函数里，去获取下一个模式。



```
00633: * Input:
00634: * Output:
00635: * Returns:
00636: *
00637:
00638: *****
00639: INT32 AppGetNextMode(INT32 cur_mode)
00640: {
00641:     INT32 result = cur_mode;
00642:
00643:     switch(cur_mode)
00644:     {
00645:         case RESULT_BT:
00646:             #if APP_SUPPORT_MUSIC==1
00647:                 result = RESULT_MUSIC;
00648:                 break;
00649:             case RESULT_MUSIC:
00650:             #endif
00651:             #if (CHIP_ASIC_ID == CHIP_ASIC_ID_5856)
00652:                 #if APP_SUPPORT_LINEIN
00653:                     result = RESULT_LINE_IN;
00654:                     break;
00655:                 case RESULT_LINE_IN:
00656:                 #endif
00657:                 #if APP_SUPPORT_FM
00658:                     result = RESULT_RADIO;
00659:                     break;
00660:                 case RESULT_RADIO:
00661:                 #endif
00662:             #else
00663:             #if APP_SUPPORT_FM || APP_SUPPORT_LINEIN
00664:                 //没有 Line in时 才进入 FM 2012-9-27
00665:                 if(gpio_detect_linein() == GPIO_DETECTED)
00666:                     result = RESULT_LINE_IN;
00667:                 else
00668:                     result = RESULT_RADIO;
00669:             #endif
00670:             }
```

目前模式顺序是：

插着 T 卡，也插着 USB

BT -> T 卡 -> USB -> Linein -> FM

不插 T 卡和 USB

BT -> Linein -> FM

插 T 卡/USB

BT ->T 卡/USB->Linein ->FM

模拟按键的识别

选择好阻值，配好最大值和最小值

```
//
: #ifndef TGT_TSD_CONFIG
: #define TGT_TSD_CONFIG
: {
:     .penGpio          = HAL_GPIO_0_4,
:     .debounceTime     = 5*HAL_TICK1S/1000,
:     .downPeriod       = 3,
:     .upPeriod         = 3,
:     .maxError         = 0x50,
:     .keyCount         = 7,
:     .minVolt          = 0,
:     .maxVolt          = 1200,
: }
: #endif // TGT_TSD_CONFIG
:

03853: #endif
03854:
03855:
03856: UINT16 pmd_GetKeyValue()
03857: {
03858:     UINT16 key_Data;
03859:     UINT32 button_high4;
03860:     UINT32 button_low4;
03861:     UINT8 button;
03862:     //hal_I2cGetData(HAL_I2C_BUS_ID_2, 0x5f, 0x1e, &key_Data, 2);
03863:     pmd_RegRead(RDA_ADDR_TOUCH_SCREEN_RESULTS1, &key_Data);
03864:
03865:     hal_HstSendEvent(BOOT_EVENT, 0x00009999);
03866:     hal_HstSendEvent(BOOT_EVENT, key_Data);
03867:     HAL_ANA_GPADC_MV_T mv = hal_AnaGpadcGpadc2Volt(key_Data&0x3ff);
03868:     hal_HstSendEvent(BOOT_EVENT, mv);
03869:     button = tsd_GetKeyIndex(mv);
03870:     hal_HstSendEvent(BOOT_EVENT, button);
03871:     return button;
03872: }
```

根据打印的 button 值，填入相应的 key 值。

```
#define KEY_MAP
{
/* power key */ KEY_PLAY,
/* gpio keys */ KEY_VOL_UP, KEY_PLAY, KEY_NEXT, KEY_VOL_DOWN, KEY_MODE, KEY_PREV, KEY_INVALID, KEY_INVALID,
/* TSC keys */ KEY_PREV, KEY_PLAY, KEY_INVALID, KEY_VOL_UP, KEY_INVALID, KEY_NEXT, KEY_VOL_DOWN, KEY_VOL_D
KEY_MUTE, KEY_PREV, KEY_MODE, KEY_REC, KEY_INVALID, KEY_EQ, KEY_INVALID, KEY_INVALID,
KEY_VOL_UP, KEY_VOL_DOWN, KEY_INVALID, KEY_INVALID, KEY_INVALID, KEY_INVALID, KEY_INVALID, KEY_INVALID,
}
```

上图中 Powerkey 就是对应软开关机的按键，可以根据自己的需求选择相应的按键。

开机时间：

因为 hst_debug 口和 sd 的引脚复用了，所以下载只在开机 3000ms 内有效，这个时间有点长，客户可以根据需要缩短下时间，设置成 500ms 即可。

```
Hal_config.c
#include "cfg_regs.h"
#include "gpio.h"
#include "tcu.h"
#include "uart.h"
#include "halp_sys.h"
#include "halp_debug.h"
#include "halp_gpio.h"
#include "halp_config.h"
#include "halp_pwm.h"
#include "hal_config.h"
#include "hal_gouda.h"
#include "hal_gpio.h"
#include "hal_pwm.h"
#include "tgt_hal_cfg.h"
#include "boot_map.h"
#include "pmd_m.h"
#include "iomux.h"
#include "gpio_config.h"
#include "tgt_board_cfg.h"
LFG_FWL_PWT_PWM_USED
FWMUSED

00241: // =====
00242: // hal_BoardSetup
00243: // =====
00244: /// Apply board dependent configuration to HAL
00245: /// @param halCfg Pointer to HAL configuration structure (from the target
00246: /// module).
00247: // =====
00248: PROTECTED VOID hal_BoardSetup(CONST HAL_CFG_CONFIG_T* halCfg)
00249: {
00250:     // Store the config pointer for later use in hal
00251:     g_halCfg = halCfg;
00252:
00253:     #if defined(_FLASH_PROGRAMMER) && defined(FORCE_GPIO_INPUT_GPO_LOW)
00254:         hal_BoardSetupFp(halCfg);
00255:     #else // ! (_FLASH_PROGRAMMER && FORCE_GPIO_INPUT_GPO_LOW)
00256:         pmd_SoftStart();
00257:         hal_TimDelay(500 MS_WAITING);
00258:         hal_BoardSetupGeneral();
00259:     #endif
00260:
00261:
00262:
00263: }
```

为减轻 pop 音，在 hal_abb_config.c 也加了 2500ms 的延时，如果有 mute 脚可以 mute 功放，这个延时也可以去掉。

```
//digital part
hal_AbbRegWrite(CODEC_CLK_CTRL, g_halAbbCodecClkCtrl);
hal_AbbRegWrite(CODEC_EP_DET_CTRL, 0xf8a7);
hal_AbbRegWrite(CODEC_ADC_SR, 0x03a8);
hal_AbbRegWrite(CODEC_VCOM_VREF_IBIAS, 0x1647);
hal_AbbRegWrite(CODEC_DAC_VOLUME, AUDIO_S_MUTE_R | AUDIO_S_MUTE_L);
hal_AbbRegWrite(CODEC_ADC_CLK, AUDIO_CNT_ADC_CLK_INT(24) | AUDIO_S_ADC_OSR_SEL(1));

hal_AbbRegWrite(CODEC_DAC_SETTING1, ABB_DAC_MODE_R_EN | ABB_DAC_MODE_L_EN | ABB_RESET_DAC(3));

hal_AbbRegWrite(CODEC_DAC_SETTING1, ABB_DAC_MODE_R_EN | ABB_DAC_MODE_L_EN);
hal_AbbRegWrite(CODEC_DAC_SETTING2, ABB_HP_GAIN_BIT(0) | ABB_RESET_MUX_EN | ABB_RESET_PA_HP);
hal_AbbRegWrite(CODEC_DAC_SETTING2, ABB_HP_GAIN_BIT(1) | ABB_RESET_PA_HP);
hal_TimDelay(2500 MS_WAITING);
hal_AbbRegWrite(CODEC_DAC_SETTING2, ABB_HP_GAIN_BIT(2) | ABB_PA_EN_HP);
```

Maintask 程序流程

2. Maintask 消息主框架

如图 1 Maintask 消息主框架所示，Maintask 主要分为：

1. 预处理消息
2. Timer 回调
3. App 自定义消息处理
4. 默认消息处理
5. 动态注册消息处理

```

        if(AppCurMsgHandler)
        {
            if(AppCurMsgHandler(&ev))
            {
                break;
            }
        }
        else
        {
            ;
        }

        if(0)
        {
        }
    }
    if 0//defined(__SLIMMMI_TCPIP__)
    else if(app_SocketEventcb(&ev))
    {
    }
    endif /* __SLIMMMI_TCPIP__ */
    if defined(__AT_SUPPORT__)
    else if(AT_AsyncEventProcess(&ev))
    {
    }
    endif /* __AT_SUPPORT__ */
    else if(ExecuteCurrProtocolHandler(&ev))
    {
    }
    else if(AppDefaultMsgHandler(&ev))
    {
    }
    else
    {
        // Ignore some ev
        if(! (ev.nEventId == 0 || ev.nEventId == AP_MSG_RTC || ev.nEventId == EV_UI_FW_REDRAW))
            app_trace(APP_MAIN_TRC, "Ignore event:%d, nparam=%d", ev.nEventId, ev.nParam1);
    }
}

```

```

while(1)
{
    COS_WaitEvent(MOD_APP, &ev, COS_WAIT_FOREVER);

    //Ignore some ev
    if(! (ev.nEventId == 0 || ev.nEventId == AP_MSG_RTC || ev.nEventId == EV_UI_FW_REDRAW))
        app_trace(APP_MAIN_TRC, "MainTask ev.id=%d, param=%d", ev.nEventId, ev.nParam1);
    if(AppMsgPreHandler(&ev))
        continue;

    if(0 == ev.nEventId)// mmi clock
    {
        mmi_handle_expired_timers();
    }

    switch(ev.nEventId)
    {
        case EV_DM_POWER_ON_IND:
        {
            UINT32 ret;

            //Display power on screen
            g_mmi_poweron_cause = ev.nParam1;
            AppPowerOnEntry();
        }
        break;
        default:
        {
            if(AppCurMsgHandler)
            {
                if(AppCurMsgHandler(&ev))
                {
                    break;
                }
            }
            else
            {
                ;
            }

            if(AppDefaultMsgHandler(&ev))
            {

```

注意：

每个消息处理函数需要返回一个布尔值，以表明消息是否需要继续往后面传递

正常情况下，一条消息只会在预处理消息和对应的处理函数处理 2 次，所以在某条消息处理完后需要在处理函数返回 `TRUE`，如果不处理消息，则需要返回一个 `FALSE` 值，否则后面的消息处理函数则不会再处理。

·3. 预处理消息

目前主要是在按键操作、插拔设备（T 卡，USB）后，重置定时器（包括关机定时器、超时定时器等）、开关背光灯，检查电池电量以及过滤一些特殊消息等。

·4. Mmi timer

处理 mmi 的定时器，精度是 0.1s

常用函数有：

1. mmi_timer_create 创建定时器
2. mmi_change_timer 修改定时时间
3. mmi_cancel_timer 取消定时器

·5. App 自定义消息处理

App 自定义消息处理的入口函数统一为 AppCurMsgHandler。

调用一个 app 的入口函数后，会调用

EntryNewScreen(SCR_ID, MsgHandlerFunc, Gui_buffer_ptr, 0, 0);

来注册当前 app 的消息处理函数（见图 3 AppCurMsgHandler 的值），调用完成后，图 2 主框架代码截图中的 AppCurMsgHandler 指向当前运行的 app 的消息处理函数。

所以，压栈内的其他 app 是无法收到消息的，如果后台 app 需要处理消息，可以通过注册动态注册消息机制来处理。

● EV_UI_FW_ON_START

在调用 EntryNewScreen 之后调用，此消息只会收到一次，除非退出 App。

- **EV_UI_FW_REDRAW**

这是一条通用的重画界面消息，经常是 App 自己给自己发。

- **EV_UI_FW_ON_PAUSE**

在进入一个新的 App 调用 EntryNewScreen 时，历史管理器会首先给当前的 App 的消息处理函数发这条消息，让当前 App 准备进入后台挂起状态，然后才是给新进入的 App 消息处理函数发 **EV_UI_FW_ON_START** 消息。

- **EV_UI_FW_ON_RESUME**

当被暂时挂起的 App 重新恢复运行状态会收到此消息，一般收到这条消息需要重画界面。

- **EV_UI_FW_ON_EXIT**

退出 App 时收到消息。

·6. 默认消息处理

固定消息(如切换应用 **EV_UI_FW_SWITCH_MOD**)、热键处理

·7. 动态注册消息处理

用于在 app 内部动态处理一些特殊消息，一般在退出 app 后取消注册，可以在 app 代码运行任意位置注册消息处理

这种机制在 App 后台处理时，比较有用。

1. SetProtocolEventHandler
2. ClearProtocolEventHandler