ARK1668E 显示相关说明文档

一、倒车轨迹和雷达信息文件制作及使用说明：

存放路径：linux-arkmicro/tool/arktrack-tool.rar

```
1. arktrack-tool工具使用说明=============================================================================================================

1.1图片命名规则
    1.1.1 轨迹图片：
          存放在track_Images目录下.
          必须以track0.png结尾,字符track为轨迹图片类型，后面的数字(0-99)即为该图片的track_id,
          驱动会根据这个两个重要的标志，去取指定的track图片显示.如：
          norm800_480track0.png ，  norm800_480track99.png


    1.1.2 车辆图片：
          存放在radar_Images目录下.
          必须以car0.png结尾，字符car为车辆图片类型，后面的数字(0-9)即为该图片的car_id,
          驱动会根据这个两个重要的标志，去取指定的car图片显示。如：
          car0.png


    1.1.3 信号图片：
          存放在signal_Images目录下.
          必须以signal0.png结尾，字符signal为信号图片类型，后面的数字(0-9)即为该图片的signal_id,
          驱动会根据这个两个重要的标志，去取指定的signal图片显示。如：
          signal0.png
```

```
   1.1.4 雷达图片：
          存放在radar_Images目录下.

          前左雷达图片显示位置radar0,格式：
          FL_XY.png
          X：数字(0-4)前左雷达信号强度     (radar_Images目录下示例：0-空  1-红  2-黄  3-绿  4-义，用户可自定义)
          Y：数字(0-4)前左中间雷达信号强度 (radar_Images目录下示例：0-空  1-红  2-黄  3-绿  4-义，用户可自定义)

          前右雷达图片显示位置radar1,格式：
          FR_YX.png
          X：数字(0-4)前右雷达信号强度     (radar_Images目录下示例：0-空  1-红  2-黄  3-绿  4-义，用户可自定义)
          Y：数字(0-4)前右中间雷达信号强度 (radar_Images目录下示例：0-空  1-红  2-黄  3-绿  4-义，用户可自定义)

          后左雷达图片显示位置radar2,格式：
          RL_XY.png
          X：数字(0-4)后左雷达信号强度     (radar_Images目录下示例：0-空  1-红  2-黄  3-绿  4-义，用户可自定义)
          Y：数字(0-4)后左中间雷达信号强度 (radar_Images目录下示例：0-空  1-红  2-黄  3-绿  4-义，用户可自定义)

          后右雷达图片显示位置radar3,格式：
          RR_YX.png
          X：数字(0-4)后右雷达信号强度     (radar_Images目录下示例：0-空  1-红  2-黄  3-绿  4-义，用户可自定义)
          Y：数字(0-4)后右中间雷达信号强度 (radar_Images目录下示例：0-空  1-红  2-黄  3-绿  4-义，用户可自定义)


          以上图片名称不能有错，工具中是根据这些名字进行图片id映射.如：
          FL_01.png  ===> 0x01000000
          FR_02.png  ===> 0x00020000
          RL_03.png  ===> 0x00000300
          RR_44.png  ===> 0x00000044
```

```
1.2 config.ini配置文件简要说明
    1.2.1 config.ini作用
        该工具会将配置文件信息写入reversingtrack中，车载系统会在carback驱动中根据配置的信息进行相关设置
        即满足用户自定义的需求，同时做到驱动中的代码兼容.

    1.2.2配置选项说明
          --------------------------------
          ;set file_type ,default:0
          file_type=0

          ;track rect info
          track_posx=0
          track_posy=0       ====>设置轨迹图片在显示屏的X,Y位置
          track_width=800
          track_height=480   ====>轨迹图片的实际大小

          ;car rect info
          car_posx=28
          car_posy=147       ====>设置车辆图片X,Y位置,相对屏幕左上角位置
          car_width=95
          car_height=190     ====>车辆图片的实际大小

          ;radar rect info: radar0==>FL
          radar0_posx=0
          radar0_posy=60     ====>设置雷达图片X,Y位置,相对屏幕左上角位置
          radar0_width=74
          radar0_height=80   ====>雷达前左图片的实际大小，以下依次类推。

          ;radar rect info: radar1==>FR
          radar1_posx=76
          radar1_posy=60
          radar1_width=74
          radar1_height=80

          ;radar rect info: radar2==>RL
          radar2_posx=0
          radar2_posy=345
          radar2_width=74
          radar2_height=80
```

```
;radar rect info: radar3==>3R
radar3_posx=76
radar3_posy=345
radar3_width=74
radar3_height=80

;signal rect info
signal_posx=0
signal_posy=0           ====>设置信号图片X,Y位置,相对屏幕左上角位置
signal_width=800
signal_height=480       ====>信号图片的实际大小
--------------------------
```

```
1.3 reversingtrack生成
    1. 存放路径：linux-arkmicro/tool/arktrack-tool.zip
    2. 将arktrack-tool.zip拷贝到win7下，解压，双击根目录下的backcar.bat,几十秒后会在arktrack-tool目录下生成该文件。

1.4 UI 倒车轨迹及雷达图片要求

    1.4.1.track,car,radar非显示图像区域像素值最好设为为0x00ffffff或0x00000000.
    1.4.2、无信号图片需要设置一张非透明图片，和arktrack-tool/signal_Images里一样即可。


2. reversingtrack升级==========================================================================================================================

    将reversingtrack存放与升级文件一起即可。和其他系统文件升级一样，注意大小不能超过4M.
```

```
3.用户接收解析mcu信息及图片设置
    1.需要将设备树里使能动态轨迹线，以开发板devb编译的为例：
    diff --git a/linux/arch/arm/boot/dts/ark1668e_devb.dts b/linux/arch/arm/boot/dts/ark1668e_devb.dts
    index 1bc27a6..012280f 100644
    --- a/linux/arch/arm/boot/dts/ark1668e_devb.dts
    +++ b/linux/arch/arm/boot/dts/ark1668e_devb.dts
    @@ -5,6 +5,7 @@

     //#define I2S_USE_EXTERNAL_CODEC
     //#define ARK1668E_I2S_FULL_DUPLEX_MODE
    +#define DYNAMIC_TRACK_DISPLAY                  ------>使能动态轨迹线

     / {
           i2c-gpio-0 {
```

```
    2.不同的客户或机型，mcu发过来的方向盘角度和雷达信息的协议是不一样，需要用户根据自己的实际情况进行适配。
    当前代码中实现的是一个简单的demo，仅供参考。 客户可根据自己实际情况改写如下三个函数.

    ark_mcu.c
    --->get_mcu_carback_data(unsigned char ch)

        --->demo_get_wheel_angle(unsigned char ch)  ===>demo, 获取mcu发过的方向盘角度信息,用户根据自己的mcu通讯协议进行修改

        --->demo_get_radar_info(unsigned char ch)   ===>demo, 获取mcu发过的雷达信息,用户根据自己的mcu通讯协议进行修改

            --->demo_parse_mcu_data(unsigned char *buf,unsigned char size)  ===>demo, 解析mcu方向盘角度及雷达信息, 用户根据
                                                                              自己的mcu协议修改,并将方向盘角度及雷达信息转换

                                                                              成要设置图片的的track_id或radar_id


                                                        注意----------------------------------------------------------
                                                        radar_id [31-24] [23-16] [15-8] [7-0]
                                                                  FL      FR      RL     RR

                                                        故radar_id FL FR RL RR 四个通道合成的.
                                                        例如：前左需要显示FL_01.png,前右需要显示FR_02.png,
                                                              后左需要显示RL_13.png,后右需要显示FR_31.png,

                                                               FL_01.png  ===> 0x01000000
                                                               FR_02.png  ===> 0x00020000
                                                               RL_13.png  ===> 0x00001300
                                                               RR_31.png  ===> 0x00000031

                                                               radar_id = 0x01000000 | 0x00020000|0x01001300|0x00000031
                                                                        =0x01021231
                                                        ------------------------------------------------------------

                                                        track_id 就是轨迹图片track字符后的数字.

                                                        ------------------------------------------------------------
注：第一次调试时，可以让mcu循环发送如下demo中的数据让ark1668接收，以便确认串口通信是否正常
    mcu_data[20] = {0x02,0x81,0xee,0x03,0x01,0x01,0x05,0x69};
    mcu_data[20] = {0x02,0x81,0xee,0x09,0x03,0x01,0x03,0x03,0x02,0x02,0x03,0x03,0x02,0x66};
```

二. LIBARKAPI 常用接口函数说明：

（1）常用函数接口说明（函数原型所在头文件路径如下图）

```
jepson@jepson:~/work-xh/test/linux-arkmicro/buildroot-external/package/libarkapi/software/include$ ls
ark_2d.h  ark_api.h  ark_common.h  ark_display.h  ark_list.h  ark_memalloc.h  ark_video.h  ark_vin.h
jepson@jepson:~/work-xh/test/linux-arkmicro/buildroot-external/package/libarkapi/software/include$ pwd
/home/jepson/work-xh/test/linux-arkmicro/buildroot-external/package/libarkapi/software/include
jepson@jepson:~/work-xh/test/linux-arkmicro/buildroot-external/package/libarkapi/software/include$
```

/*

**-->函数功能说明：设置当前的显示模式

**-->参数说明：(1)mode：目前使用的模式主要有：播放多媒体（DISP_PLAYER）、显示手机互联（DISP_PHONELINK）

**-->返回值说明：正确：等于 0　错误：小于 0

```
*/
int arkapi_video_set_display_mode(int mode);

/*
**-->函数功能说明：初始化解码器，显示层，分配内存等
**-->参数说明：(1)stream_type：例如当前的视频流格式是 h264，这里参数设置为 RAW_STRM_TYPE_H264 其余格式可参考 mfcapi.h 里的枚举
**-->返回值说明：正确：返回 video_handle *指针  错误：返回空
*/
video_handle *arkapi_video_init(int stream_type);

/*
**-->函数功能说明:设置当前要显示的信息，比如要显示的宽高，偏移值，源的窗口大小等
**-->参数说明:(1)handle：初始化返回的指针  (2)cfg_vid：参考 ark_api.h 里的 video_cfg 结构体
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_video_set_config(video_handle *handle, video_cfg *cfg_vid);

/*
**-->函数功能说明:获取当前要显示的信息，比如要显示的宽高，偏移值，源的窗口大小等
**-->参数说明:(1)handle：初始化返回的指针  (2)cfg_vid：参考 ark_api.h 里的 video_cfg 结构体
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_video_get_config(video_handle *handle, video_cfg *cfg_vid);

/*
**-->函数功能说明:启动解码器，并调用 scale 模块缩放显示到屏幕上
**-->参数说明:(1)handle：初始化返回的指针 (2)src_addr：视频数据的源地址 (3)len：数据的大小 (4)fps：播放视频流的帧率
**-->返回值说明：正确：返回解码显示的帧数  错误：小于 0
*/
int arkapi_video_play(video_handle *handle, const void *src_addr, int len, int fps);

/*
**-->函数功能说明：针对一些特殊的视频流，会有最后几帧不放不完全的现象，调用这个这个函数会将剩余的帧播放完成
**-->参数说明:(1)handle：初始化返回的指针 (2)fps：播放视频流的帧率
**-->返回值说明：正确：返回 0  错误：小于 0
*/
int arkapi_video_play_eof(video_handle *handle, int fps);
```

```
/*
**-->函数功能说明:打开对应的显示层
**-->参数说明:(1)handle：初始化返回的指针 (2)enable: 1 打开 0 关闭
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_video_show(video_handle *handle, int enable);

/*
**-->函数功能说明:打开对应的显示层并判断当前的显示模式是否和已经设置的模式一样
**-->参数说明:(1)handle：初始化返回的指针 (2)mode：DISP_PLAYER || DISP_PHONELINK
(3)enable：1 打开 0 关闭
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_video_show_mode(video_handle *handle, int mode, int enable);

/*
**-->函数功能说明:释放解码器，内存，显示层等
**-->参数说明:(1)handle：初始化返回的指针
**-->返回值说明：无
*/
void arkapi_video_release(video_handle *handle);

/*
**-->函数功能说明:通知显示库现在已经进入倒车
**-->参数说明:无
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_enter_carback(void);

/*
**-->函数功能说明:通知显示库现在已经退出倒车
**-->参数说明:无
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_exit_carback(void);

/*
**-->函数功能说明:打开显示层
**-->参数说明:(1)PRIMARY_LAYER      -->对应 OSD2 层     //UI
                VIDEO_LAYER        -->对应 VIDEO2 层   //倒车、手机互联、多媒体
                OVER_LAYER         -->对应 OSD1 层     //倒车轨迹、雷达信息
                TVOUT_LAYER        -->对应 VIDEO1 层   //开机动画、LOGO
                AUX_LAYER          -->对应 OSD3 层     //保留
注：(1)1668E 有五个显示层，同时只能用四个层，对于 app 开发者来说主要涉及到需要处
```

理的显示层是 PRIMARY_LAYER，其他层一般由底层处理，一般情况下不要关心。

(2)显示层的优先级： TVOUT_LAYER VIDEO_LAYER OVER_LAYER PRIMARY_LAYER

0 1 2 3

---------->

显示层按以上次序从上往下叠加，数字越大，显示就在越上面。

```
**-->返回值说明：正确：返回空  错误：返回 disp_handle 指针
*/
disp_handle *arkapi_display_open_layer(enum ark_disp_layer layer);

/*
**-->函数功能说明:关闭显示层
**-->参数说明:(1)handle：初始化返回的指针
**-->返回值说明：无
*/
void arkapi_display_close_layer(disp_handle *handle);

/*
**-->函数功能说明:显示对应的显示层
**-->参数说明:(1)handle：初始化返回的指针
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_display_show_layer(disp_handle *handle);

/*
**-->函数功能说明:隐藏对应的显示层
**-->参数说明:(1)handle：初始化返回的指针
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_display_hide_layer(disp_handle *handle);

/*
**-->函数功能说明:强制显示对应的显示层
**-->参数说明:参考 arkapi_display_open_layer 的参数说明
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_display_force_show_layer(enum ark_disp_layer layer);

/*
**-->函数功能说明:强制隐藏对应的显示层
**-->参数说明:参考 arkapi_display_open_layer 的参数说明
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_display_force_hide_layer(enum ark_disp_layer layer);
```

```
/*
**-->函数功能说明:设置显示层位于屏幕的偏移值
**-->参数说明:(1):handle:初始化返回的指针 (2)x: 位于屏幕水平方向的偏移值 (3)y:位于屏幕垂直方向的偏移值
**-->返回值说明：正确：等于0 错误：小于0
*/
int arkapi_display_set_layer_pos(disp_handle *handle, int x, int y);

/*
**-->函数功能说明:设置显示层的大小
**-->参数说明:(1)：handle：初始化返回的指针 (2)width: 水平方向显示的宽度 (3)height：垂直方向显示的高度
**-->返回值说明：正确：等于0 错误：小于0
*/
int arkapi_display_set_layer_size(disp_handle *handle, int width, int height);

/*
**-->函数功能说明:设置显示层的格式
**-->参数说明:(1)：handle：初始化返回的指针 (2)format:参考 ark_api.h 里的ark_disp_format 枚举
**-->返回值说明：正确：等于0 错误：小于0
*/
int arkapi_display_set_layer_format(disp_handle *handle, int format);

/*
**-->函数功能说明:设置显示层的显示地址
**-->参数说明:(1):handle：初始化返回的指针 (2)pyrgbaddr：y地址 (2)pcbaddr: u地址 (3)pcraddr：v地址
**-->返回值说明：正确：等于0 错误：小于0
*/
int arkapi_display_set_layer_addr(disp_handle *handle, u32 pyrgbaddr, u32 pcbaddr, u32 pcraddr);

/*****************************************************************************/

//以下xxx_atomic函数，作用和上述不带atomic函数相同，不同的是下面的函数设置时不会马上生效，需要调用
arkapi_display_layer_update_commit 后才统一生效，主要用于避免视频播放过程中改变显示分辨率造成的花屏。
int arkapi_display_set_layer_pos_atomic(disp_handle *handle, int x, int y);
int arkapi_display_set_layer_size_atomic(disp_handle *handle, int width, int height);
int arkapi_display_set_layer_format_atomic(disp_handle *handle, int format);
int arkapi_display_set_layer_addr_atomic(disp_handle *handle, u32 pyrgbaddr, u32
```

```
pcbaddr, u32 pcraddr);
int arkapi_display_layer_update_commit(disp_handle *handle);

/*************************************************************************/

/*
**-->函数功能说明:设置显示层的亮度饱和度对比度等参数
**-->参数说明:(1)layer：参考 arkapi_display_open_layer 的参数说明 (2)vp：参考 ark_api.h 里的 ark_disp_color 结构体
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_display_layer_set_color(enum ark_disp_layer layer, disp_color* vp);

/*
**-->函数功能说明:获取设置显示层的亮度饱和度对比度等参数
**-->参数说明:(1)layer：参考 arkapi_display_open_layer 的参数说明 (2)vp：参考 ark_api.h 里的 ark_disp_color 结构体
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_display_layer_get_color(enum ark_disp_layer layer, disp_color* vp);

/*
**-->函数功能说明:等待一个帧缓冲
**-->参数说明:(1)handle：初始化返回的指针
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_display_wait_for_vsync(disp_handle *handle);

/*
**-->函数功能说明:获取显示层的显示地址
**-->参数说明:(1):handle：初始化返回的指针 (2)pyrgbaddr：y 地址 (2)pcbaddr: u 地址 (3)pcraddr：v 地址
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_display_get_layer_addr(disp_handle *handle, u32 *pyrgbaddr, u32 *pcbaddr, u32 *pcraddr);

/*
**-->函数功能说明:获取当前的屏幕分辨率
**-->参数说明:(1)screen：包含了屏幕的大小信息的结构体，调用后直接读取该结构体的成员变量的值即可
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_display_get_screen_info(screen_info *screen);
```

```
/*
**-->函数功能说明:内存的初始化
**-->参数说明:无
**-->返回值说明：正确：返回 memalloc_handle 结构体指针  错误：空
*/
memalloc_handle *arkapi_memalloc_init(void);

/*
**-->函数功能说明:申请 buffer
**-->参数说明:(1)handle：初始化返回的指针 (2)reqbuf：需要申请的 buffer 配置，大小，数量等
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_memalloc_reqbuf(memalloc_handle *handle, reqbuf_info *reqbuf);

/*
**-->函数功能说明:释放内存，初始化的句柄
**-->参数说明:(1)handle：初始化返回的指针
**-->返回值说明：无
*/
void arkapi_memalloc_release(memalloc_handle *handle);

/*
**-->函数功能说明:打开 vin 设备
**-->参数说明:无
**-->返回值说明：正确：返回文件描述符  错误：小于 0
*/
int arkapi_open_vin(void);

/*
**-->函数功能说明:关闭 vin 设备
**-->参数说明:(1)vin_fd：初始化返回的描述符
**-->返回值说明：正确：  错误：
*/
int arkapi_close_vin(int vin_fd);

/*
**-->函数功能说明:配置信号以及初始化底层驱动
**-->参数说明:(1)vin_fd：初始化返回的描述符 (2)progressive：信号是隔行 1 还是逐行 0  (3)itu601en：信号是 601 信号 1 不是 0
**-->返回值说明：正确：等于 0 错误：小于 0
*/
int arkapi_vin_config(int vin_fd,int progressive, int itu601en);
```

```
/*
**-->函数功能说明:启动 vin，开始解码显示信号
**-->参数说明:(1)vin_fd：初始化返回的描述符
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_vin_start(int vin_fd);

/*
**-->函数功能说明:停止 vin
**-->参数说明:(1)vin_fd：初始化返回的描述符
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_vin_stop(int vin_fd);

/*
**-->函数功能说明:检测当前配置的通道是否有信号
**-->参数说明:(1)vin_fd：初始化返回的描述符
**-->返回值说明：正确：返回 1 有信号 或者 0 无信号  错误：小于 0
*/
int arkapi_vin_detect_signal(int vin_fd);

/*
**-->函数功能说明:设置解码器的通道
**-->参数说明:(1)vin_fd：初始化返回的描述符  (2)source：解码器对应的通道 参考
ark_api.h 里的 dvr_source 枚举
**-->返回值说明：正确：等于 0  错误：小于 0
*/
int arkapi_vin_switch_channel(int vin_fd, enum dvr_source source);
```

（2）avin 接口使用示例：

进入 avin 的时候调用如下接口：

```
int fd = 0,ret,progressive,itu601en,signal;
int channel = DVR_SOURCE_AUX;

fd = arkapi_open_vin();
if( fd < 0 )
{
        printf("open error.\n");
        return -1;
}

progressive = 0;
itu601en = 0;
arkapi_vin_config(fd,progressive,itu601en);

arkapi_vin_switch_channel(fd,channel);

arkapi_vin_start(fd);
```

退出 AVIN 时调用如下接口

```
arkapi_vin_stop(fd);
arkapi_close_vin(fd);
```

注：调用顺序要先调用 arkapi_vin_config 之后，再调用 arkapi_vin_switch_channel 配置解码芯片的 AVIN 输入通道，然后才能调用 arkapi_vin_start 开始。

（3）基于应用倒车时控制 UI 层的使用示例：

应用和倒车控制 ui 说明：控制 ui 层开启或者关闭分为两种，一种是驱动层开关，一种是应用层开关。具体事是由哪一个去操作，取决于应用是否调用如下接口

```
            arkapi_enter_carback();
            if (0 != ioctl(pThis->carbackFd, CARBACK_IOCTL_APP_ENTER_DONE)) {
                qDebug("ioctl CARBACK_IOCTL_APP_ENTER_DONE fail.");
            }
            arkapi_display_hide_layer(disp_layer_ui);          //close ui
            pThis->CarbackStatusChange(CBS_On);
        } else if (data == CBS_Off) {
            arkapi_exit_carback();
            if (0 != ioctl(pThis->carbackFd, CARBACK_IOCTL_APP_EXIT_DONE)) {
                qDebug("ioctl CARBACK_IOCTL_APP_ENTER_DONE fail.");
            }
            arkapi_display_show_layer(disp_layer_ui);          //open ui
            pThis->CarbackStatusChange(CBS_Off);
        }
    } else {
        qDebug("read /dev/carback fail.");
    }
}
}

void Carback::initialize()
{
    pthread_t pthead;
    int ret = -1;

    carbackFd = open("/dev/carback", O_RDONLY);
    if (carbackFd < 0) {
        qDebug("open /dev/carback fail.");
        return;
    }

    disp_layer_ui = arkapi_display_open_layer(PRIMARY_LAYER); //UI

    if (0 != ioctl(carbackFd, CARBACK_IOCTL_SET_APP_READY)) {
        qDebug("ioctl CARBACK_IOCTL_SET_APP_READY fail.");
        return;
    }

    ret = pthread_create(&pthead, NULL, readCarbckStatus, this);
    if(ret != 0) {
        printf("pthread_create failed!\n");
    }
```

1.1 应用不调用接口的情况：

在倒车的时候，应用不去调用倒车驱动的这三个接口的情况下，此时在进出倒车时的开关 ui 层的操作是在倒车驱动里去做好的，应用不需要关心开关层的处理，底层会自动进倒

车的时候关闭 ui 层，退出倒车时候开 ui 层。

1.2 应用调用接口的情况：

应用调用这三个接口之后，驱动就不会去对 ui 层去做操作， 也就是开关 ui 层都在应用里面去操作。调用 demo 及方法简介请看下面代码：

```
    /*************************************************************************
*********************/
#include "carback.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <QDebug>
#include "ark_api.h"

#define CARBACK_IOCTL_BASE                          0x9A
#define CARBACK_IOCTL_SET_APP_READY                 _IO(CARBACK_IOCTL_BASE, 0)
#define CARBACK_IOCTL_APP_ENTER_DONE                _IO(CARBACK_IOCTL_BASE, 1)
#define CARBACK_IOCTL_APP_EXIT_DONE                 _IO(CARBACK_IOCTL_BASE, 2)
#define CARBACK_IOCTL_GET_STATUS                    _IOR(CARBACK_IOCTL_BASE, 3,
int)
#define CARBACK_IOCTL_DETECT_SIGNAL                 _IOR(CARBACK_IOCTL_BASE, 4,
int)


Carback::Carback(QObject *parent) : QObject(parent)
{
    exitThread = 1;
    carbackFd = -1;
}

Carback::~Carback()
{
    exitThread = -1;
    carbackFd = -1;
}

disp_handle *disp_layer_ui;

void *Carback::readCarbckStatus(void *arg)
```

```
{
    Carback *pThis = (Carback *)arg;

    unsigned char data;

    while (pThis->exitThread) {
        if (1 == read(pThis->carbackFd, &data, 1)) {
            if (data == CBS_On) {
            //告诉显示库此时已经进入倒车，这里必须要调用，否则显示切换会有问题
                arkapi_enter_carback();
            //下面这个 ioctl 就是告诉驱动不要关闭 ui 层，交由应用处理
                if (0 != ioctl(pThis->carbackFd, CARBACK_IOCTL_APP_ENTER_DONE))
                {
                    qDebug("ioctl CARBACK_IOCTL_APP_ENTER_DONE fail.");
                }
        //这里应用就是关闭 ui，看自己需求，如果不需要关闭 ui,需要显示别的 Ui,这里
可以不需要调用
            arkapi_display_hide_layer(disp_layer_ui);           //close ui
                pThis->CarbackStatusChange(CBS_On);
            } else if (data == CBS_Off) {
            //告诉显示库此时已经退出倒车，这里必须要调用，否则显示切换会有问题
                arkapi_exit_carback();
            //下面这个 ioctl 就是告诉驱动不要开启 ui 层，交由应用处理
                if (0 != ioctl(pThis->carbackFd, CARBACK_IOCTL_APP_EXIT_DONE))
                {
                    qDebug("ioctl CARBACK_IOCTL_APP_ENTER_DONE fail.");
                }
    //这里应用就是开启 ui，看自己需求，如果不需要关闭 ui,需要显示别的 Ui,这里可以
不需要调用
        arkapi_display_show_layer(disp_layer_ui);          //open ui
                pThis->CarbackStatusChange(CBS_Off);
            }
        } else {
            qDebug("read /dev/carback fail.");
        }
    }
}

void Carback::initialize()
{
    pthread_t pthead;
    int ret = -1;

    carbackFd = open("/dev/carback", O_RDONLY);
```

```
    if (carbackFd < 0) {
        qDebug("open /dev/carback fail.");
        return;
    }
    //这里是初始化 ui 层句柄，后面检测到进出倒车时需要用到
    disp_layer_ui = arkapi_display_open_layer(PRIMARY_LAYER); //UI
    //必须要在应用上电初始化的时候去调用下面这个 ioctl，代表应用已经准备好了，可以去控制 ui 层了
    if (0 != ioctl(carbackFd, CARBACK_IOCTL_SET_APP_READY)) {
        qDebug("ioctl CARBACK_IOCTL_SET_APP_READY fail.");
        return;
    }

    ret = pthread_create(&pthead, NULL, readCarbckStatus, this);
    if(ret != 0) {
        printf("pthread_create failed!\n");
    }
}
```

（4）显示模式切换应用调用接口说明：

显示模式切换目前主要用于倒车、多媒体视频播放、手机互联之间的相互切换。

1. 倒车调用：进入倒车时调用 arkapi_enter_carback()，退出倒车时调用 arkapi_exit_carback()，具体实现可以参考上面说到的（基于应用倒车时控制 UI 层的使用示例）里的代码。
2. 应用在点击对应 ui 界面进入多媒体视频或者手机互联之前要设置对应的显示模式，进入多媒体视频调用 arkapi_video_set_display_mode(DISP_PLAYER),进入手机互联调用 arkapi_video_set_display_mode(DISP_PHONELINK)

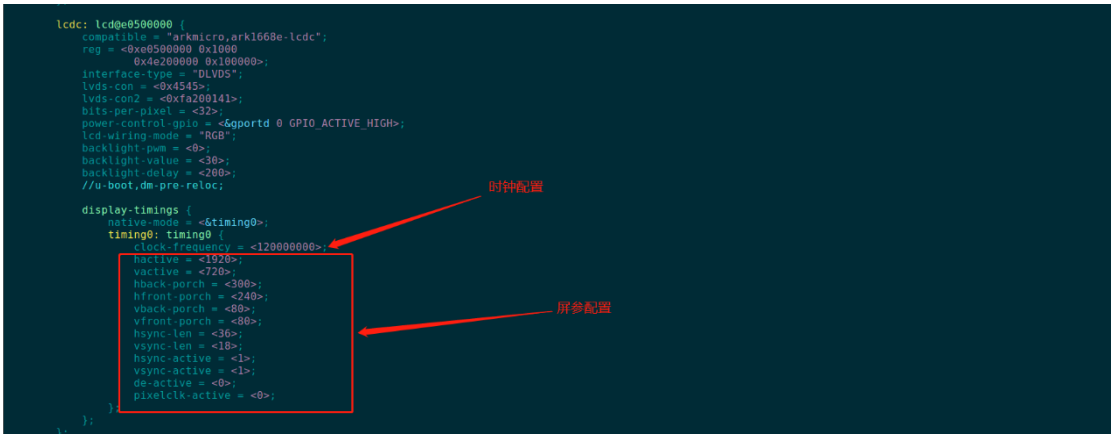
注：如果没有 ui 界面的交互，比如在应用准备调用 mplayer 播放之前就需要设置对应的模式

三．常见问题配置修改：

（1）屏参（以 ark1668e_devb 分支为例）：

1．uboot 屏参修改：

```
vim u-boot/arch/arm/dts/ark1668e_devb.dts
```



2. kernel 屏参修改：

kernel 和 uboot 中一样，kernel 里设备树所在路径如下图所示

```
vim linux/arch/arm/boot/dts/ark1668e_devb.dts
```

（2）GPIO-I2C：参考设备树中对模拟 i2c 的配置，如下图：

```
$ vim linux/arch/arm/boot/dts/ark1668e_devb.dts
```

```
i2c-gpio-1 {
#address-cells = <1>;
#size-cells = <0>;
compatible = "i2c-gpio";
gpios = <&gportf 29 0      /* SDA */
&gportf 28 0        /* SCL */
>;
i2c-gpio,delay-us = <6>;    /* clk freq = 500/delay KHz */
i2c-gpio,scl-output-only;

ark7116: ark7116@82 {
compatible = "arkmicro,ark7116_1668e_devb";
reset-gpio = <&gportf 27 0>;
reg = <0x59>; /* i2c address(7 bits) */
default-channel = <0>;
carback-config = <1>;
port {
ark7116_0: endpoint@0{
remote-endpoint = <&itu_1>;
};
};
};
```