ALL Winner Technology	文　档　编　号		SW1107REF001		
	版　本	V0.1	密　级	1	共 528 页

MELIS2.0 用户手册

（内部公开/外部公开)

文档作者				创建日期	2019-12-20		
	签　名			日　期			
拟　制							
审　核							
批　准							
分发部门							
[ ]SW	[ ]SD	[ ]AL	[ ]CS	[ ]TST			
[ ]MKT							

All Winner Technology

CopyRight©2011 All Winner Technology, All Right Reserved

版本历史

	修改人	时间	备注
V0.1		2019-12-20	建立初始版本

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

目录

目录....3
1. 编写目的....22
2. Melis2.0 系统概述....23
3. Melis2.0 快速开发....24
  3.1. Melis2.0 SDK 目录结构....24
  3.2. Melis2.0 编译环境....26
  3.3. Melis2.0 固件打包....27
  3.4. Melis2.0 固件烧录....28
  3.5. 串口打印信息....30
  3.6. Melis2.0 添加和调用一个模块....34
    3.6.1. 为什么划分模块？....34
    3.6.2. UART 驱动模块....34
4. 编译工具链使用....39
  4.1. 工具链通用配置....39
  4.2. 模块的工具链配置....41
  4.3. 简单的 makefile....44
5. 固件烧录工具的安装....46
  5.1. PhoenixSuit 的安装步骤....46
  5.2. 检验 USB 驱动安装....50
  5.3. 使用烧录软件 PhoenixSuit....51
6. 固件打包脚本....54
  6.1. 概要描述....54
  6.2. 术语定义....54
    6.2.1. makefile....54
    6.2.2. image.bat....54
  6.3. 工具介绍....54
  6.4. 打包步骤....55
    6.4.1. makefile 部分....55
    6.4.2. image.bat 部分....55
  6.5. 问题与解决方案....56
    6.5.1. 固件由那些文件构成....56
    6.5.2. melis100.fex 文件包含什么内容....56
    6.5.3. ramdisk.iso....57
    6.5.4. udisk.iso....57
    6.5.5. 如何对 sysdata 分区进行添加....57
    6.5.6. 分区对齐设置....58
    6.5.7. 固件烧录后打印提示 spinor 不支持....59
7. 固件修改工具(ImageModify)使用....61
  7.1. 界面说明....61
  7.2. 操作步骤....62

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

7.2.1. 配置平台......62
7.2.2. 选择固件......62
7.2.3. 选择要替换的文件......64
7.2.4. 替换文件......65
7.2.5. 保存固件......65
7.3. 注意事项......66
7.4. 增加固件修改权限设置......68
7.4.1. 概述......68
7.4.2. 操作说明......68
8. 系统启动流程......70
8.1. Shell 部分......70
8.2. Orange 和 desktop 部分......71
8.3. app_root 加载部分......73
8.4. home 加载部分......74
9. 调屏......77
9.1. 调屏步骤简介......77
9.1.1. 判断屏接口。......77
9.1.2. 确定硬件连接。......77
9.1.3. 配置显示部分 sys_config.fex......77
9.1.4. Lcd_panel_cfg.c 初始化文件中配置屏参数......78
9.2. 软件配置说明......78
9.2.1. 屏文件说明......78
9.2.2. 开关屏流程......79
9.2.3. 对屏的初始化......82
9.2.4. 其它函数......85
9.3. TCON 参数说明......86
9.3.1. 接口参数说明......86
9.3.2. 时序参数说明......89
9.3.3. 其他参数说明......91
9.4. 屏文件实例......92
10. 应用程序开发......93
10.1. APP framework 体系结构......93
10.1.1. APP framework 简介......93
10.2. DESKTOP......94
10.2.1. desktop 简介......94
10.3. 10.3.GUI 窗口体系和消息机制......95
10.3.1. 窗口类型......95
10.3.2. 消息机制......96
10.4. 资源使用......96
10.4.1. 应用字符串资源......96
10.4.2. 应用图片资源......98
10.4.3. 背景图资源......100
10.4.4. 启动图资源......100
10.4.5. 按键音资源......100
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

10.4.6. 系统功能资源......100
10.5. 应用程序编写......103
10.5.1. 简单应用编写......103
11. UI 资源工具......110
11.1. 工具的由来......110
11.2. 背景图片的制作工具......111
11.3. 图标制作工具......112
11.4. UI 图标资源包文件 theme.bin......116
11.4.1. 生成 theme.h 和 theme.bin......117
11.4.2. 资源文件 theme.bin 和 theme.h 的使用范例......118
11.5. 语言文字编码工具......120
11.6. 字库制作工具......122
11.6.1. 全字库转换......122
11.6.2. 指定字符集的字库转换......123
11.6.3. 字库使用范例......125
11.7. UI 字符资源包文件 lang.bin......126
11.7.1. 生成 lang.h 和 lang.bin......126
11.7.2. 资源文件 lang.h 和 lang.bin 的使用范例......130
12. 配置文件 sys_config.fex......133
12.1. 配置文件的作用......133
12.2. 配置文件的规则和读取函数接口......134
12.2.1. 配置项格式......134
12.2.2. 注释符号“;”......134
12.2.3. 子健的值的类型......135
12.2.4. 读取配置项的函数接口......135
13. 驱动编程概述......138
13.1. 驱动的挂载和卸载......138
13.2. 驱动访问......138
14. Audio Dirver(CTRL)......139
14.1. 挂载......139
14.2. 卸载......139
14.3. 访问......139
14.4. 功能命令列表......139
14.5. 功能命令描述......140
14.5.1. AUDIO_DEV_CMD_GET_VOLUME......140
14.5.2. AUDIO_DEV_CMD_SET_VOLUME......140
14.5.3. AUDIO_DEV_CMD_CHANGE_IF......141
14.5.4. AUDIO_DEV_CMD_CLOSE_DEV......141
14.5.5. AUDIO_DEV_CMD_SET_PROTECT_VOL......141
14.5.6. AUDIO_DEV_CMD_GET_PROTECT_VOL......142
14.5.7. AUDIO_DEV_CMD_SET_CHAN_MODE......142
14.5.8. AUDIO_DEV_CMD_GET_CHAN_MODE......143
14.5.9. AUDIO_DEV_CMD_ENTER_STANDBY......143
14.5.10. AUDIO_DEV_CMD_EXIT_STANDBY......143
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

14.5.11. AUDIO_DEV_CMD_REG_CALLBACK....144
14.5.12. AUDIO_DEV_CMD_UNREG_CALLBACK....144
14.5.13. AUDIO_DEV_CMD_GET_INTERFACE....144
14.5.14. AUDI_DEV_CMD_SET_HPCOM_DRIVE_MODE....145
14.5.15. AUDIO_DEV_CMD_SWAP_OUTPUT_CHANNELS....145
14.5.16. AUDIO_DEV_CMD_SET_SW_VOL_MAX....145
14.5.17. AUDIO_DEV_CMD_GET_SW_VOL_MAX....146
14.5.18. AUDIO_DEV_CMD_SET_DAC_MAX_GAIN....146
14.5.19. AUDIO_DEV_CMD_GET_DAC_MAX_GAIN....146
14.5.20. AUDIO_DEV_CD_SET_USE_USER_VOLUME_MAP....146
14.5.21. AUDIO_DEV_CMD_MUTE....147
14.5.22. AUDIO_DEV_CMD_GET_DAC_MAX_GAIN....147
15. Audio Dirver(PLAY/PLAY0)....148
15.1. 挂载....148
15.2. 卸载....148
15.3. 访问....148
15.4. 功能命令列表....148
15.5. 功能命令描述....149
15.5.1. AUDIO_DEV_CMD_START....149
15.5.2. AUDIO_DEV_CMD_STOP....149
15.5.3. AUDIO_DEV_CMD_PAUSE....149
15.5.4. AUDIO_DEV_CMD_CONTINUE....150
15.5.5. AUDIO_DEV_CMD_GET_SAMPCNT....150
15.5.6. AUDIO_DEV_CMD_SET_SAMPCNT....150
15.5.7. AUDIO_DEV_CMD_GET_PARA....151
15.5.8. AUDIO_DEV_CMD_SET_PARA....151
15.5.9. AUDIO_DEV_CMD_GET_VOLUME....151
15.5.10. AUDIO_DEV_CMD_SET_VOLUME....152
15.5.11. AUDIO_DEV_CMD_REG_USERMODE....152
15.5.12. AUDIO_DEV_CMD_FLUSH_BUF....152
15.5.13. AUDIO_DEV_CMD_QUERY_BUFSIZE....153
15.5.14. AUDIO_DEV_CMD_RESIZE_BUF....153
15.5.15. AUDIO_DEV_CMD_WRITE_DATA....153
15.5.16. AUDIO_DEV_CMD_DATA_FINISH....154
15.5.17. AUDIO_DEV_CMD_SET_PLAYMODE....154
15.5.18. AUDIO_DEV_CMD_MUTE....154
16. Audio Dirver(REC)....156
16.1. 挂载....156
16.2. 卸载....156
16.3. 访问....156
16.4. 功能命令列表....156
16.5. 功能命令描述....157
16.5.1. AUDIO_DEV_CMD_START....157
16.5.2. AUDIO_DEV_CMD_STOP....157
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

16.5.3. AUDIO_DEV_CMD_PAUSE....157
16.5.4. AUDIO_DEV_CMD_CONTINUE....158
16.5.5. AUDIO_DEV_CMD_GET_SAMPCNT....158
16.5.6. AUDIO_DEV_CMD_SET_SAMPCNT....158
16.5.7. AUDIO_DEV_CMD_GET_PARA....159
16.5.8. AUDIO_DEV_CMD_SET_PARA....159
16.5.9. AUDIO_DEV_CMD_GET_VOLUME....159
16.5.10. AUDIO_DEV_CMD_SET_VOLUME....160
16.5.11. AUDIO_DEV_CMD_REG_USERMODE....160
16.5.12. AUDIO_DEV_CMD_FLUSH_BUF....160
16.5.13. AUDIO_DEV_CMD_QUERY_BUFSIZE....161
16.5.14. AUDIO_DEV_CMD_READ_DATA....161
16.5.15. AUDIO_DEV_CMD_SET_PLAYMODE....161
17. Audio Dirver(FM)....163
17.1. 挂载....163
17.2. 卸载....163
17.3. 访问....163
17.4. 功能命令列表....163
17.5. 功能命令描述....163
17.5.1. AUDIO_DEV_CMD_START....163
17.5.2. AUDIO_DEV_CMD_STOP....164
17.6. 17.6 示例....164
18. RTC Dirver....165
18.1. 挂载....165
18.2. 卸载....165
18.3. 访问....165
18.4. 功能命令列表....165
18.5. 功能命令描述....165
18.5.1. RTC_CMD_GET_TIME....165
18.5.2. RTC_CMD_SET_TIME....166
18.5.3. RTC_CMD_GET_DATE....166
18.5.4. RTC_CMD_SET_DATE....166
18.5.5. RTC_CMD_REQUEST_ALARM....167
18.5.6. RTC_CMD_RELEASE_ALARM....167
18.5.7. RTC_CMD_START_ALARM....167
18.5.8. RTC_CMD_STOP_ALARM....168
18.5.9. RTC_CMD_QUERY_ALARM....168
18.5.10. RTC_CMD_QUERY_INT....168
19. IIC Dirver(TWI0/ TWI1/TWI2)....169
19.1. 挂载....169
19.2. 卸载....169
19.3. 访问....169
19.4. 功能命令列表....169
19.5. 功能命令描述....169
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

19.5.1. TWI_WRITE_SPEC_RS......169
19.5.2. TWI_READ_SPEC_RS......170
19.5.3. TWI_READ_EX_NO_RS......170
19.5.4. TWI_READ_EX_STP_RS......170
19.5.5. TWI_SET_SCL_CLOCK......171
19.6. 数据结构......171
19.6.1. __twi_dev_para_ex_t......171
19.7. 示例......172
20. Key Dirver......173
20.1. 挂载......173
20.2. 卸载......173
20.3. 访问......173
20.4. 功能命令列表......173
20.5. 功能命令描述......174
20.5.1. DRV_KEY_CMD_GETKEY......174
20.5.2. DRV_KEY_CMD_PUTKEY......174
20.5.3. DRV_KEY_CMD_SET_FIRST_DEBOUNCE_TIME......174
20.5.4. DRV_KEY_CMD_SET_FIRST_RPT_TIME......175
20.5.5. DRV_KEY_CMD_SET_SBSEQ_RPT_TIME......175
20.5.6. DRV_KEY_CMD_GET_FIRST_DEBOUNCE_TIME......175
20.5.7. DRV_KEY_CMD_GET_FIRST_RPT_TIME......175
20.5.8. DRV_KEY_CMD_GET_SBSEQ_RPT_TIME......176
20.5.9. DRV_KEY_CMD_GET_IRMASK......176
20.5.10. DRV_KEY_CMD_GET_IRPOWERVALUE......176
20.5.11. DRV_KEY_CMD_GET_HOLDKEYVALUEMAX......177
20.5.12. DRV_KEY_CMD_GET_HOLDKEYVALUEMIN......177
20.6. 数据结构......177
20.6.1. __key_msg_t......177
21. IR Dirver......178
21.1. 挂载......178
21.2. 卸载......178
21.3. 访问......178
21.4. 功能命令列表......178
21.5. 功能命令描述......178
21.5.1. DRV_IRKEY_CMD_PUTKEY......178
21.5.2. DRV_IRKEY_CMD_SET_SBSEQ_RPT_TIME......179
21.5.3. DRV_IRKEY_CMD_GET_SBSEQ_RPT_TIME......179
21.5.4. DRV_IRKEY_CMD_DISPLAY_SCANCODE......179
21.5.5. DRV_KEY_CMD_GET_IRPOWERVALUE......180
21.5.6. DRV_KEY_CMD_GET_IRMASK......180
21.5.7. DRV_IRKEY_CMD_CHG_REMOTER......180
21.6. 数据结构......181
21.6.1. __ir_key_msg_t......181
22. SPI Dirver......182
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

22.1. 挂载......182
22.2. 卸载......182
22.3. 访问......182
22.4. 功能命令列表......182
22.5. 功能命令描述......182
22.5.1. SPI_DEV_CMD_MASTER_RW......182
22.5.2. SPI_DEV_CMD_SLAVE_READ......183
22.5.3. SPI_DEV_CMD_SLAVE_WRITE......184
22.5.4. SPI_DEV_CMD_CHECK_DUAL......184
22.6. 数据结构......184
22.6.1. __spi_dev_set_xfer_t......184
22.7. 7 示例......185
23. UART Dirver(UART0/ UART1/UART2)......187
23.1. 挂载......187
23.2. 卸载......187
23.3. 访问......187
23.4. 功能命令列表......187
23.5. 功能命令描述......188
23.5.1. UART_CMD_SET_PARA......188
23.5.2. UART_CMD_SET_BAUDRATE......188
23.5.3. UART_CMD_FLUSH......188
23.6. 示例......188
24. TP Dirver......190
24.1. 挂载......190
24.2. 卸载......190
24.3. 访问......190
24.4. 功能命令列表......190
24.5. 功能命令描述......190
24.5.1. DRV_TP_CMD_REG......190
24.5.2. DRV_TP_CMD_UNREG......191
24.5.3. DRV_TP_CMD_ADJUST......191
24.5.4. DRV_TP_CMD_SET_OFFSET_INFO......193
24.5.5. DRV_TP_CMD_GET_OFFSET_INFO......194
24.5.6. DRV_TP_CMD_SET_MSG_PERTIME......194
24.5.7. DRV_TP_CMD_GET_MSG_PERTIME......194
24.5.8. DRV_TP_CMD_SET_WORKMODE......194
24.6. 数据结构......195
24.6.1. __ev_tp_msg_t......195
24.6.2. __ev_tp_pos_t......196
24.7. 示例......196
24.7.1. dual 模式......196
24.7.2. fast 模式......197
25. Display Dirver......199
25.1. 挂载......199

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.2. 卸载 199
25.3. 访问 199
25.4. 功能命令列表 199
25.5. 功能命令描述 202
25.5.1. DISP_CMD_SET_BKCOLOR 202
25.5.2. DISP_CMD_GET_BKCOLOR 202
25.5.3. DISP_CMD_SET_COLORKEY 202
25.5.4. DISP_CMD_GET_COLORKEY 203
25.5.5. DISP_CMD_SET_PALETTE_TBL 203
25.5.6. DISP_CMD_GET_PALETTE_TBL 203
25.5.7. DISP_CMD_SCN_GET_WIDTH 203
25.5.8. DISP_CMD_SCN_GET_HEIGHT 204
25.5.9. DISP_CMD_GET_OUTPUT_TYPE 204
25.5.10. DISP_CMD_SET_EXIT_MODE 204
25.5.11. DISP_CMD_START_CMD_CACHE 205
25.5.12. DISP_CMD_EXECUTE_CMD_AND_STOP_CACHE 205
25.5.13. DISP_CMD_SET_BRIGHT 205
25.5.14. DISP_CMD_SET_CONTRAST 206
25.5.15. DISP_CMD_SET_SATURATION 206
25.5.16. DISP_CMD_GET_BRIGHT 206
25.5.17. DISP_CMD_GET_CONTRAST 207
25.5.18. DISP_CMD_GET_SATURATION 207
25.5.19. DISP_CMD_ENHANCE_ON 207
25.5.20. DISP_CMD_ENHANCE_OFF 208
25.5.21. DISP_CMD_GET_ENHANCE_EN 208
25.5.22. DISP_CMD_CLK_ON 208
25.5.23. DISP_CMD_CLK_OFF 208
25.5.24. DISP_CMD_SET_DE_FLICKER 208
25.5.25. DISP_CMD_LAYER_REQUEST 209
25.5.26. DISP_CMD_LAYER_RELEASE 209
25.5.27. DISP_CMD_LAYER_OPEN 210
25.5.28. DISP_CMD_LAYER_CLOSE 210
25.5.29. DISP_CMD_LAYER_SET_FB 211
25.5.30. DISP_CMD_LAYER_GET_FB 211
25.5.31. DISP_CMD_LAYER_SET_SRC_WINDOW 211
25.5.32. DISP_CMD_LAYER_GET_SRC_WINDOW 212
25.5.33. DISP_CMD_LAYER_SET_SCN_WINDOW 212
25.5.34. DISP_CMD_LAYER_GET_SCN_WINDOW 213
25.5.35. DISP_CMD_LAYER_SET_PARA 213
25.5.36. DISP_CMD_LAYER_GET_PARA 213
25.5.37. DISP_CMD_LAYER_ALPHA_ON 214
25.5.38. DISP_CMD_LAYER_ALPHA_OFF 214
25.5.39. DISP_CMD_LAYER_GET_ALPHA_EN 215
25.5.40. DISP_CMD_LAYER_SET_ALPHA_VALUE 215
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.41. DISP_CMD_LAYER_GET_ALPHA_VALUE......215
25.5.42. DISP_CMD_LAYER_CK_ON......216
25.5.43. DISP_CMD_LAYER_CK_OFF......216
25.5.44. DISP_CMD_LAYER_GET_CK_EN......217
25.5.45. DISP_CMD_LAYER_SET_PIPE......217
25.5.46. DISP_CMD_LAYER_GET_PIPE......217
25.5.47. DISP_CMD_LAYER_TOP......218
25.5.48. DISP_CMD_LAYER_BOTTOM......218
25.5.49. DISP_CMD_LAYER_GET_PRIO......219
25.5.50. DISP_CMD_LAYER_SET_SMOOTH......219
25.5.51. DISP_CMD_LAYER_GET_SMOOTH......220
25.5.52. DISP_CMD_LAYER_SET_BRIGHT......220
25.5.53. DISP_CMD_LAYER_SET_CONTRAST......220
25.5.54. DISP_CMD_LAYER_SET_SATURATION......221
25.5.55. DISP_CMD_LAYER_SET_HUE......221
25.5.56. DISP_CMD_LAYER_GET_BRIGHT......222
25.5.57. DISP_CMD_LAYER_GET_CONTRAST......222
25.5.58. DISP_CMD_LAYER_GET_SATURATION......222
25.5.59. DISP_CMD_LAYER_GET_HUE......223
25.5.60. DISP_CMD_LAYER_ENHANCE_ON......223
25.5.61. DISP_CMD_LAYER_ENHANCE_OFF......224
25.5.62. DISP_CMD_LAYER_GET_ENHANCE_EN......224
25.5.63. DISP_CMD_SCALER_REQUEST......224
25.5.64. DISP_CMD_SCALER_RELEASE......225
25.5.65. DISP_CMD_SCALER_EXECUTE......225
25.5.66. DISP_CMD_HWC_OPEN......225
25.5.67. DISP_CMD_HWC_CLOSE......226
25.5.68. DISP_CMD_HWC_SET_POS......226
25.5.69. DISP_CMD_HWC_GET_POS......226
25.5.70. DISP_CMD_HWC_SET_FB......227
25.5.71. DISP_CMD_HWC_SET_PALETTE_TABLE......227
25.5.72. DISP_CMD_VIDEO_START......228
25.5.73. DISP_CMD_VIDEO_STOP......228
25.5.74. DISP_CMD_VIDEO_SET_FB......228
25.5.75. DISP_CMD_VIDEO_GET_FRAME_ID......229
25.5.76. DISP_CMD_VIDEO_GET_DIT_INFO......229
25.5.77. DISP_CMD_LCD_ON......229
25.5.78. DISP_CMD_LCD_OFF......230
25.5.79. DISP_CMD_LCD_SET_BRIGHTNESS......230
25.5.80. DISP_CMD_LCD_GET_BRIGHTNESS......230
25.5.81. DISP_CMD_LCD_SET_COLOR......231
25.5.82. DISP_CMD_LCD_GET_COLOR......231
25.5.83. DISP_CMD_LCD_CPUIF_XY_SWITCH......231
25.5.84. DISP_CMD_LCD_CHECK_OPEN_FINISH......231

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.85. DISP_CMD_LCD_CHECK_CLOSE_FINISH........ 231
25.5.86. DISP_CMD_LCD_SET_SRC........ 231
25.5.87. DISP_CMD_TV_ON........ 232
25.5.88. DISP_CMD_TV_OFF........ 232
25.5.89. DISP_CMD_TV_SET_MODE........ 232
25.5.90. DISP_CMD_TV_GET_MODE........ 233
25.5.91. DISP_CMD_TV_AUTOCHECK_ON........ 234
25.5.92. DISP_CMD_TV_AUTOCHECK_OFF........ 234
25.5.93. DISP_CMD_TV_GET_INTERFACE........ 234
25.5.94. DISP_CMD_TV_SET_SRC........ 235
25.5.95. DISP_CMD_TV_GET_DAC_STATUS........ 235
25.5.96. DISP_CMD_TV_SET_DAC_SOURCE........ 236
25.5.97. DISP_CMD_TV_GET_DAC_SOURCE........ 236
25.6. 数据结构........ 237
25.6.1. __disp_color_t........ 237
25.6.2. __disp_rect_t........ 237
25.6.3. __disp_rectsz_t........ 238
25.6.4. __disp_pos_t........ 238
25.6.5. __disp_fb_t........ 238
25.6.6. __disp_layer_info_t........ 240
25.6.7. __disp_colorkey_t........ 241
25.6.8. __disp_video_fb_t........ 241
25.6.9. __disp_scaler_para_t........ 242
25.7. 示例........ 243
25.7.1. Layer Demo........ 243
25.7.2. Scaler Demo........ 247
25.7.3. HWC Demo........ 252
25.7.4. Video Demo........ 253
26. Orange Module........ 257
26.1. Introduction........ 257
26.1.1. Description........ 257
26.1.2. Purpose........ 257
26.1.3. Reference........ 257
26.1.4. Contact Info........ 257
26.2. Window........ 257
26.2.1. 分类........ 258
26.2.2. 关系........ 258
26.2.3. 图层属性........ 259
26.2.4. 屏幕属性........ 260
26.2.5. 优先级的概念........ 260
26.2.6. Pipe 的概念........ 260
26.2.7. Alpha 的计算公式........ 261
26.2.8. Colorkey 的运算........ 262
26.2.9. Interface........ 262
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.3. Message......279
26.3.1. Interface......280
26.4. Core......283
26.4.1. 2D Graphic Library......284
26.4.2. Displaying Text......292
26.4.3. Bitmap Drawing......300
26.4.4. Fonts......303
26.4.5. Colors......307
26.4.6. Memory Device......309
26.4.7. 2D Accelerate......312
26.4.8. Other......318
26.5. Widget......320
26.6. Resources......321
26.7. Orange Demo......325
27. Cedar Module......336
27.1. Introduction......336
27.1.1. Description......336
27.1.2. Purpose......336
27.1.3. Reference......336
27.2. CEDAR......336
27.2.1. CEDAR_CMD_SET_MEDIAFILE......337
27.2.2. CEDAR_CMD_GET_MESSAGE_CHN......337
27.2.3. CEDAR_CMD_GET_ERROR_TYPE......338
27.2.4. CEDAR_CMD_PLAY......338
27.2.5. CEDAR_CMD_STOP......339
27.2.6. CEDAR_CMD_PAUSE......339
27.2.7. CEDAR_CMD_FF......340
27.2.8. CEDAR_CMD_REV......341
27.2.9. CEDAR_CMD_JUMP......342
27.2.10. CEDAR_CMD_GET_STATUS......343
27.2.11. CEDAR_CMD_AUDIO_RAW_DATA_ENABLE......344
27.2.12. CEDAR_CMD_GET_TOTAL_TIME......345
27.2.13. CEDAR_CMD_GET_CUR_TIME......345
27.2.14. CEDAR_CMD_GET_TAG......346
27.2.15. CEDAR_CMD_SET_FRSPEED......346
27.2.16. CEDAR_CMD_GET_FRSPEED......347
27.2.17. CEDAR_CMD_SET_TAG......347
27.2.18. CEDAR_CMD_GET_ABSTYPE......348
27.2.19. CEDAR_CMD_GET_AUDBPS......349
27.2.20. CEDAR_CMD_GET_SAMPRATE......349
27.2.21. CEDAR_CMD_SET_CHN......350
27.2.22. CEDAR_CMD_GET_CHN......350
27.2.23. CEDAR_CMD_SET_VOL......350
27.2.24. CEDAR_CMD_GET_VOL......351

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

27.2.25. CEDAR_CMD_VOLUP....351
27.2.26. CEDAR_CMD_VOLDOWN....352
27.2.27. CEDAR_CMD_SET_EQ....352
27.2.28. CEDAR_CMD_GET_EQ....353
27.2.29. CEDAR_CMD_SET_VPS....353
27.2.30. CEDAR_CMD_GET_VPS....354
27.2.31. CEDAR_CMD_SET_AB_A....354
27.2.32. CEDAR_CMD_SET_AB_B....355
27.2.33. CEDAR_CMD_SET_AB_LOOPCNT....355
27.2.34. CEDAR_CMD_CLEAR_AB....356
27.2.35. CEDAR_CMD_SET_SPECTRUM....357
27.2.36. CEDAR_CMD_GET_SPECTRUM....357
27.2.37. CEDAR_CMD_SEL_AUDSTREAM....358
27.2.38. CEDAR_CMD_GET_AUDSTREAM....359
27.2.39. CEDAR_CMD_GET_AUDSTREAM_PROFILE....360
27.2.40. CEDAR_CMD_GET_AUDSTREAM_CNT....361
27.2.41. CEDAR_CMD_GET_AUDSTREAM_PROFILE_V2....361
27.2.42. CEDAR_CMD_QUERY_BUFFER_USAGE....362
27.2.43. CEDAR_CMD_SET_AB_A_V2....362
27.2.44. CEDAR_CMD_SET_AB_B_V2....363
27.2.45. CEDAR_CMD_SET_AB_LOOPCNT_V2....363
27.2.46. CEDAR_CMD_CLEAR_AB_V2....364
27.2.47. CEDAR_CMD_ENABLE_AB_V2....364
27.2.48. CEDAR_CMD_SET_AUDIO_AB_MODE_V2....364
27.2.49. CEDAR_CMD_GET_VBSTYPE....365
27.2.50. CEDAR_CMD_GET_VIDBITRATE....366
27.2.51. CEDAR_CMD_GET_VIDFPS....366
27.2.52. CEDAR_CMD_GET_FRAMESIZE....367
27.2.53. CEDAR_CMD_SET_VID_LAYERHDL....368
27.2.54. CEDAR_CMD_SET_VID_WINDOW....370
27.2.55. CEDAR_CMD_GET_VID_WINDOW....371
27.2.56. CEDAR_CMD_SET_VID_SHOW_MODE....372
27.2.57. CEDAR_CMD_GET_VID_SHOW_MODE....373
27.2.58. CEDAR_CMD_SWITCH_VID_SHOW....374
27.2.59. CEDAR_CMD_SET_FRPIC_SHOWTIME....375
27.2.60. CEDAR_CMD_GET_FRPIC_SHOWTIME....375
27.2.61. CEDAR_CMD_SET_ROTATE....375
27.2.62. CEDAR_CMD_INVALID_VIDEOLAYER....376
27.2.63. CEDAR_CMD_SET_FILE_SWITCH_VPLY_MODE....376
27.2.64. CEDAR_CMD_ENABLE_VIDEO_AUTO_SCALE....377
27.2.65. CEDAR_CMD_GET_LBSTYPE....377
27.2.66. CEDAR_CMD_GET_SUB_INFO....378
27.2.67. CEDAR_CMD_GET_SUBTITLE_PROFILE....379
27.2.68. CEDAR_CMD_SELECT_SUBTITLE....380
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

27.2.69. CEDAR_CMD_GET_SUBTITLE........ 381
27.2.70. CEDAR_CMD_GET_SUBTITLE_CNT........ 382
27.2.71. CEDAR_CMD_GET_SUBTITLE_PROFILE_V2........ 382
27.2.72. CEDAR_CMD_SET_SUBTITLE_ITEM_POST_PROCESS........ 382
27.2.73. CEDAR_CMD_ENABLE_EXTERN_SUBTITLE........ 383
27.2.74. CEDAR_CMD_CAPTURE_PIC........ 383
27.2.75. CEDAR_CMD_ASK_PIC_BUFSIZE........ 383
27.2.76. CEDAR_CMD_GET_FRAME_PIC........ 383
27.2.77. CEDAR_DUCKWEED_CMD_OPEN_MEDIAFILE........ 384
27.2.78. CEDAR_DUCKWEED_CMD_CLOSE_MEDIAFILE........ 384
27.2.79. CEDAR_DUCKWEED_CMD_GET_FILE_FORMAT........ 385
27.2.80. CEDAR_DUCKWEED_CMD_GET_FILE_SIZE........ 385
27.2.81. CEDAR_DUCKWEED_CMD_GET_TOTAL_TIME........ 386
27.2.82. CEDAR_DUCKWEED_CMD_GET_VIDSTREAM_CNT........ 386
27.2.83. CEDAR_DUCKWEED_CMD_GET_VIDSTREAM_PROFILE........ 387
27.2.84. CEDAR_DUCKWEED_CMD_GET_AUDSTREAM_CNT........ 387
27.2.85. CEDAR_DUCKWEED_CMD_GET_AUDSTREAM_PROFILE........ 388
27.2.86. CEDAR_DUCKWEED_CMD_GET_PREVIEW_FB........ 389
27.2.87. CEDAR_DUCKWEED_CMD_GET_PREVIEW_FB_BY_PTS........ 390
27.2.88. CEDAR_CMD_SET_USER_FILEOP........ 390
27.2.89. 2.90.CEDAR_CMD_SET_STOP_MODE........ 391
27.2.90. 2.91.CEDAR_CMD_SET_PITCH........ 391
27.2.91. 2.92.CEDAR_CMD_GET_PITCH........ 392
27.2.92. 2.93.CEDAR_CMD_PLAY_AUX_WAV_FILE........ 392
27.2.93. 2.94.CEDAR_CMD_PLAY_AUX_WAV_BUFFER........ 393
27.2.94. 2.95.CEDAR_CMD_SET_AUX_WAV_BUFFER_SIZE........ 393
27.2.95. 2.96.CEDAR_CMD_GET_AUX_WAV_BUFFER_SIZE........ 394
27.2.96. 2.97.CEDAR_CMD_GET_PLY_DATA_STATUS........ 394
27.2.97. 2.98.CEDAR_CMD_STREAM_SET_INFO........ 394
27.2.98. 2.99.CEDAR_CMD_STREAM_QUERY_BUFFER........ 395
27.2.99. 3.10.CEDAR_CMD_STREAM_WRITE_BUFFER........ 395
27.2.100. 3.11.CEDAR_CMD_STREAM_END........ 396
27.2.101. 3.12.CEDAR_CMD_FAST_LOOPPLAY_ENABLE........ 396
27.2.102. 3.13.CEDAR_CMD_FAST_LOOPPLAY_DISENABLE........ 397
27.2.103. 3.14.CEDAR_CMD_SET_FAST_LOOPPLAY_CNT........ 397
27.2.104. 3.15.CEDAR_CMD_GET_FAST_LOOPPLAY_CNT........ 398
27.3. 常用功能 DEMO........ 398
27.3.1. 音视频播放 demo........ 398
27.3.2. Wav/Pcm 音频播放 demo........ 399
27.3.3. H264 裸流数据播放 demo........ 401
28. Willow Module........ 403
28.1. Introduction........ 403
28.1.1. Description........ 403
28.1.2. Purpose........ 403

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

28.1.3. Reference ... 403
28.2. Willow ... 403
28.2.1. WILLOW_CMD_QUERY_STATUS ... 404
28.2.2. WILLOW_CMD_GET_THUMBS ... 404
28.2.3. WILLOW_CMD_SET_ALBUM_ART ... 406
28.2.4. WILLOW_CMD_SHOW_IMG_FROM_FILE ... 409
28.2.5. WILLOW_CMD_SHOW_IMG_FROM_BUFFER ... 410
28.2.6. WILLOW_CMD_SET_SWITCH_MODE ... 412
28.2.7. WILLOW_CMD_GET_SHOW_PARAM ... 413
28.2.8. WILLOW_CMD_GET_IMG_INFO ... 413
28.2.9. WILLOW_CMD_SCALE ... 415
28.2.10. WILLOW_CMD_ROTATE ... 415
28.2.11. WILLOW_CMD_MOVE ... 416
28.2.12. WILLOW_CMD_COME_BACK ... 417
28.2.13. WILLOW_CMD_OPEN_SHOW ... 417
28.2.14. WILLOW_CMD_SHUT_SHOW ... 418
28.2.15. WILLOW_CMD_CHECK_IMG ... 418
28.2.16. WILLOW_CMD_START_DEC ... 419
28.2.17. WILLOW_CMD_START_SHOW ... 419
28.2.18. WILLOW_CMD_STOP ... 420
28.2.19. WILLOW_CMD_SET_SCN ... 420
28.2.20. WILLOW_CMD_START_SHOW_EXT ... 421
28.2.21. WILLOW_CMD_CATCH_SCREEN ... 423
28.2.22. WILLOW_CMD_AUTO_ROTATE ... 424
28.2.23. WILLOW_CMD_AUTO_ROTATE ... 424
28.2.24. WILLOW_CMD_SET_SHOW_MODE ... 425
28.2.25. WILLOW_CMD_CFG_LYR ... 425
28.2.26. WILLOW_CMD_ROT_LYR ... 426
28.2.27. WILLOW_CMD_CFG_OUTPUT ... 427
28.2.28. WILLOW_CMD_CFG_INPUT ... 428
28.2.29. WILLOW_CMD_SET_PLAYLIST ... 429
28.2.30. WILLOW_CMD_MOV_TRACK ... 430
28.2.31. WILLOW_CMD_GET_CUR_INDEX ... 431
28.2.32. WILLOW_CMD_SCALE_EXT ... 431
29. 内核编程指南-多线程编程 ... 433
29.1. Introduction ... 433
29.1.1. Description ... 433
29.1.2. Purpose ... 433
29.1.3. Reference ... 433
29.2. Thread ... 433
29.2.1. CreateThread ... 434
29.2.2. DeleteThread ... 434
29.2.3. DeleteThreadReq ... 435
29.2.4. GetTidCur ... 436

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

29.2.5. SchedLock....436
29.2.6. TimeDly....437
29.2.7. TimeDlyResume....437
29.3. Semaphore....438
29.3.1. SemCreate....438
29.3.2. SemDel....438
29.3.3. SemPend....439
29.3.4. SemPost....440
29.3.5. SemAccept....440
29.3.6. SemQuery....441
29.3.7. SemSet....441
29.4. MsgQ....443
29.4.1. QCreate....443
29.4.2. QDelete....443
29.4.3. QPend....444
29.4.4. QPost....445
29.4.5. QAccept....445
29.4.6. QQuery....446
29.4.7. QFlush....447
29.5. Flag....447
29.5.1. FlagCreate....447
29.5.2. FlagDel....448
29.5.3. FlagPend....449
29.5.4. FlagAccept....450
29.5.5. FlagPost....451
29.5.6. FlagQuery....452
29.6. Socket....452
29.6.1. SktCreate....453
29.6.2. SktDel....454
29.6.3. SktPend....454
29.6.4. SktPost....455
29.6.5. SktAccept....456
29.6.6. SktFlush....456
30. 内核编指南-定时器....458
30.1. Introduction....458
30.1.1. Description....458
30.1.2. Purpose....458
30.1.3. Reference....458
30.1.4. Contact Info....458
30.2. Time....458
30.2.1. TimeGet....459
30.2.2. TimeSet....459
30.2.3. GetRtcTime....459
30.2.4. SetRtcTime....460

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

30.2.5. GetRtcDate....461
30.2.6. SetRtcDate....461
30.2.7. SoftTimer....462
30.2.8. QuerryError....462
30.2.9. CreateTimer....463
30.2.10. DeleteTimer....464
30.2.11. StartTimer....464
30.2.12. StopTimer....465
30.3. HwTimer....465
30.3.1. RequestTimer....466
30.3.2. ReleaseTimer....466
30.3.3. StartTimer....467
30.3.4. StopTimer....468
30.4. Counter....468
30.4.1. RequestCounter....469
30.4.2. ReleaseCounter....469
30.4.3. StartCounter....470
30.4.4. StopCounter....470
30.4.5. PauseCounter....471
30.4.6. ContinueCounter....471
30.4.7. SetValue....472
30.4.8. QuerryCounter....472
30.4.9. SetPrescale....472
30.4.10. QuerryState....473
30.4.11. Demo....473
30.5. Alarm....475
30.5.1. RequestAlarm....476
30.5.2. ReleaseAlarm....476
30.5.3. StartAlarm....477
30.5.4. StopAlarm....477
30.5.5. QuerryAlarm....477
31. 内核编指南-内存管理....478
31.1. Introduction....478
31.1.1. Description....478
31.1.2. Purpose....478
31.1.3. Reference....478
31.2. Palloc....478
31.2.1. PageAlloc....478
31.2.2. PageFree....480
31.3. VmCreate....480
31.3.1. VMCreate....480
31.3.2. VmDelete....481
31.4. Malloc....481
31.4.1. HeapCreate....481
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

31.4.2. HeapDelete......482
31.4.3. Malloc......482
31.4.4. Mfree......483
31.4.5. Realloc......483
31.4.6. VMalloc......484
31.4.7. Vmfree......484
31.5. Cache......484
31.5.1. FlushDCache......485
31.5.2. CleanDCache......485
31.5.3. CleanFlushDCache......486
31.5.4. FlushDCacheRegion......486
31.5.5. CleanDCacheRegion......486
31.5.6. CleanFlushDCacheRegion......487
31.5.7. FlushICache......487
31.5.8. FlushICacheRegion......488
31.5.9. LockICache......488
31.5.10. UnlockICache......489
31.5.11. LockDCache......489
31.5.12. UnlockDCache......489
31.6. Other......490
31.6.1. VA2PA......490
31.6.2. PAContinue......491
31.6.3. GetIOVaByPa......491
31.6.4. TotalMemSize......492
31.6.5. FreeMemSize......492
32. 内核编指南-时钟管理......493
32.1. Introduction......493
32.1.1. Description......493
32.1.2. Purpose......493
32.1.3. Reference......493
32.2. SourceClock......493
32.2.1. SetSrcFreq......493
32.2.2. GetSrcFreq......494
32.3. ModuleClk......495
32.3.1. OpenMclk......495
32.3.2. CloseMclk......495
32.3.3. RegCallback......496
32.3.4. UnregCallback......497
32.3.5. SetMclkSrc......497
32.3.6. GetMclkSrc......498
32.3.7. SetMclkDiv......498
32.3.8. GetMclkDiv......499
32.3.9. MclkOnOff......499
32.3.10. MclkReset......500
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

32.3.11. Demo....500
32.4. PowerMan....502
32.4.1. ReqPwmMode....503
32.4.2. RelPwmMode....503
32.4.3. LockCpuFreq....504
32.4.4. UnlockCpuFreq....504
32.4.5. UsrEventNotify....505
32.4.6. RegDevice....505
32.4.7. UnregDevice....506
32.4.8. EnterStandby....506
33. 内核编指南-模块管理....508
33.1. Introduction....508
33.1.1. Description....508
33.1.2. Purpose....508
33.1.3. Reference....508
33.2. Mechanism....508
33.2.1. Application....509
33.2.2. Module....510
33.2.3. Driver....511
33.3. Application....512
33.3.1. PCreate....512
33.3.2. PDel....513
33.3.3. PDelReq....513
33.3.4. Run....514
33.3.5. Demo....514
33.4. Module....514
33.4.1. MInstall....515
33.4.2. MUninstall....515
33.4.3. MOpen....516
33.4.4. MClose....516
33.4.5. MRead....517
33.4.6. MWrite....517
33.4.7. MIoctrl....518
33.5. Driver....519
34. 内核编指南-PIN 管理....520
34.1. Introduction....520
34.1.1. Description....520
34.1.2. Purpose....520
34.1.3. Reference....520
34.2. Structure....520
34.2.1. PIN Status Structure....520
34.3. PIN Group....521
34.3.1. PIN Group Request....521
34.3.2. PIN Group Release....522
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

34.4. PIN Status....523
- 34.4.1. Get PIN Group Status....523
- 34.4.2. Get PIN Status....523
- 34.4.3. Set PIN Status....524
- 34.4.4. Set PIN IO....525
- 34.4.5. Set PIN Pull....525
- 34.4.6. Set PIN Driver....526

34.5. PIN Data....526
- 34.5.1. Read PIN Data....526
- 34.5.2. Write PIN Data....527

34.6. PIN Config....527

34.7. PIN INT....528

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

1. 编写目的

本文档是全志 Melis2.0 系统的开发指引文档，旨在协助开发者了解和掌握 Melis 系统，快速搭建 Melis 系统的开发环境，将 Melis2.0 系统应用到产品开发中。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

2. Melis2.0 系统概述

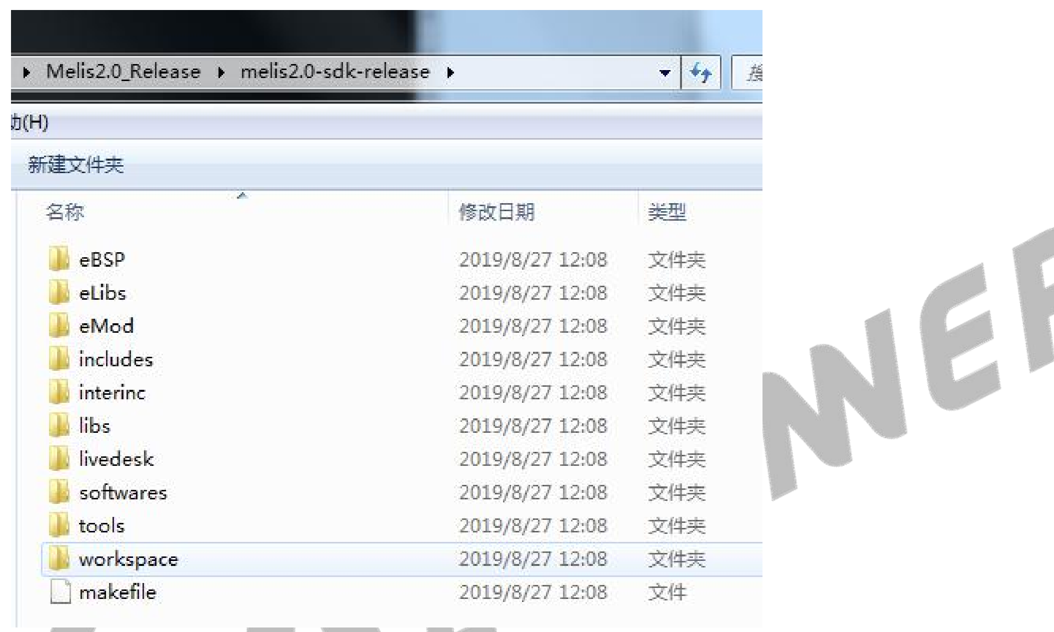
Melis2.0 系统是基于全志芯片平台自主研发的一套软件系统，其内容涵盖 SDK 代码包、资源制作工具组、编译链接脚本、固件打包烧录工具套件、调试工具 5 个部分，本文档将在后续章节向开发者逐一进行介绍。

【图 1】

3. Melis2.0 快速开发

3.1. Melis2.0 SDK 目录结构

Melis2.0 发布版本的 SDK 目录结构如下【图 2】所示，下面按照源码文件夹、打包文件夹、临时库文件夹、工具文件夹的顺序介绍各文件夹的功能，使开发者对 Melis2.0 SDK 的目录结构有一个初步印象。



【图 2】

【eBSP】板级支持包文件夹，该文件夹存放的是 spi、uart、sdio 等驱动的源码文件，包含了各驱动模块对应寄存器的设置代码，生成的目标文件是.a 后缀的库文件，在编译链接 eMod 中的驱动时使用这些.a 库。

【eLIBs】公共库文件夹，生成的目标文件是.a 后缀的库文件，包含了一些公共库函数接口，例如一些本平台自定义的 stdio、string 等接口。

【eMod】驱动模块文件夹，本文件夹主要是以 drv_、mod_开头的子文件夹，各子文件夹代表一个独立的驱动/中间件模块，编译每个子文件夹中的源码会生成一个 ELF 格式的可执行文件。eBSP 库中生成的.a 库大多是在这里的驱动模块编译链接时使用。

【includes】此文件夹用于存放公用的.h 后缀头文件，为使代码结构清晰，这些头文件会按照功能分别放到各个子文件夹中。

【interinc】此文件夹存放的头文件主要是与 ELF 格式可执行文件的解析相关的宏定义和结构体定义等。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

【**livedesk**】本文件夹属于应用层文件夹，桌面、音乐、视频、相册、日历、录音等功能的代码均存放于此。

【**makefile**】SDK 根目录编译脚本文件，打开 cygwin 命令行窗口，进入 SDK 目录，输入 make clean;make 命令后按“回车”键，就会执行此 makefile 脚本把整个 SDK 重新编译一遍。

【**workspace**】编译 eMod、livedesk 文件夹中的源码所生成的驱动、中间件、应用等独立模块的 ELF 可执行文件均存放于此，通过运行 workspace\suniv\beetles\image.bat 脚本，将这些独立的可执行文件，打包合成一个.img 后缀的文件，用于烧录到开发板中的存储设备（比如 Norflash）。

【**libs**】本文件夹主要是存放.a 库，不存放源码，eBSP、eLIBs 文件夹中编译生成的.a 后缀文件均存放在本文件夹中；

【**softwares**】本文件夹用于存放工具软件。

【**tools**】本文件夹主要存放打包脚本中调用到的 windows 应用程序，例如将一些配置信息更新到已经生成的可执行文件中，就会使用到本文件夹中的工具。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

3.2. Melis2.0 编译环境

Melis2.0 的编译环境是 Cygwin + RVDS2.2。

Cygwin 是在 windows 上运行的类 Unix 环境；RVDS2.2 即 RealView Development Suite 2.2，是由 ARM 公司出品的交叉编译工具套件。

双击 cygwin 图标运行 cygwin，通过命令行窗口输入命令。进入 SDK 根目录，输入 make clean;make 命令来执行根目录的 makefile 脚本，重新编译整个 sdk，参考下【图 3】。可使用文本编辑工具打开 makefile 文件查看和修改。

根目录下的 makefile 脚本运行时，逐层编译链接 eBSP、eLIBs、eMOD、livedesk 文件夹中各个子目录，生成 ELF 格式的可执行文件以供打包生成固件。这些 ELF 可执行文件的分别存放到了 workspace/beetles/ramfs 或 workspace/beetles/rootfs 文件夹。

关于 Melis2.0 编译链接工具的具体调用流程，请参考第 4 章《编译工具链使用》。

```
/cygdrive/e/Melis2.0_Release/melis2.0-sdk-release

Administrator@sunweibin-pc ~
$ cd /cygdrive/e/Melis2.0_Release/melis2.0-sdk-release/

Administrator@sunweibin-pc /cygdrive/e/Melis2.0_Release/melis2.0-sdk-release
$ ls -al
总用量 29
drwxrwxr-x+ 1 Administrator None   0 十二 23 16:39 .
drwxrwxr-x+ 1 Administrator None   0 八月 27 12:08 ..
drwxrwxr-x+ 1 Administrator None   0 八月 27 12:08 Documents
drwxrwxr-x+ 1 Administrator None   0 八月 27 12:08 eBSP
drwxrwxr-x+ 1 Administrator None   0 八月 27 12:08 eLibs
drwxrwxr-x+ 1 Administrator None   0 八月 27 12:08 eMod
drwxrwxr-x+ 1 Administrator None   0 八月 27 12:08 includes
drwxrwxr-x+ 1 Administrator None   0 八月 27 12:08 interinc
drwxrwxr-x+ 1 Administrator None   0 八月 27 12:08 libs
drwxrwxr-x+ 1 Administrator None   0 八月 27 12:08 livedesk
-rwxrwxr--+ 1 Administrator None 209 八月 27 12:08 makefile
drwxrwxr-x+ 1 Administrator None   0 八月 27 12:08 softwares
drwxrwxr-x+ 1 Administrator None   0 八月 27 12:08 tools
drwxrwxr-x+ 1 Administrator None   0 八月 27 12:08 workspace

Administrator@sunweibin-pc /cygdrive/e/Melis2.0_Release/melis2.0-sdk-release
$ make clean;make
```

【图 3】

3.3. Melis2.0 固件打包

进入 SDK 的 workspace\suniv\beetles\文件夹，双击 image.bat 运行该脚本，把各驱动、中间件、应用模块等独立的可执行文件合成到 ePDKv100.img 镜像文件，以便 PhoenixSuit 烧录工具烧录到开发板的 Norflash 中。

image.bat 脚本的具体运行流程，可通过文本编辑工具打开查阅和修改，如【图 4】。

【图 4】

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

3.4. Melis2.0 固件烧录

Melis2.0 使用的烧录工具是 PhoenixSuit，通过该工具将固件烧录到开发板的 Norflash 中，安装 PhoenixSuit 的操作文档请参看章节《固件烧录工具的安装》。下面我们介绍其中一种烧录方法。

【step1】双击 PhoenixSuit 图标运行 PhoenixSuit 软件，点击“一键刷机”选项卡，通过“浏览”按钮选择想要烧录的固件的存放路径；

【step2】用 USB 链接线把电脑和开发板链接起来；

【step3】短路 Norflash 的 1、2 引脚，上电 1 到 2 秒钟之后即可松开，当烧录软件出现进度条，表明 PhoenixSuit 通过 USB 检测到开发板，之后烧录软件自动完成剩余的烧录工作；

【step4】等待烧录完成，会有弹窗提示烧录成功，表明烧录完成，之后就可重启设备运行新固件。

烧录期间和重启设备后，都可以通过查看开发板的调试串口输出的打印信息来判断软件的运行状态。

【图 5】

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

【图 6】

【图 7】

3.5. 串口打印信息

安装串口工具（例如 SecureCRT）并打开，通过串口线接收 SoC 输出的 UART 串口打印信息，可以对软件的运行状态做判断，参考【图 9】。

UART 串口波特率是 115200bps，串口收发引脚 Rx/Tx 参看 workspace/eFex/sys_config.fex 文件中的 uart_debug_port、uart_debug_tx、uart_debug_rx 参数，如【图 8】所示为窗口 1UART port1 的 PA2/PA3 脚。

```
100 uart_debug_port = 1
101 uart_debug_tx   = port:PA2<5>
102 uart_debug_rx   = port:PA3<5>
```

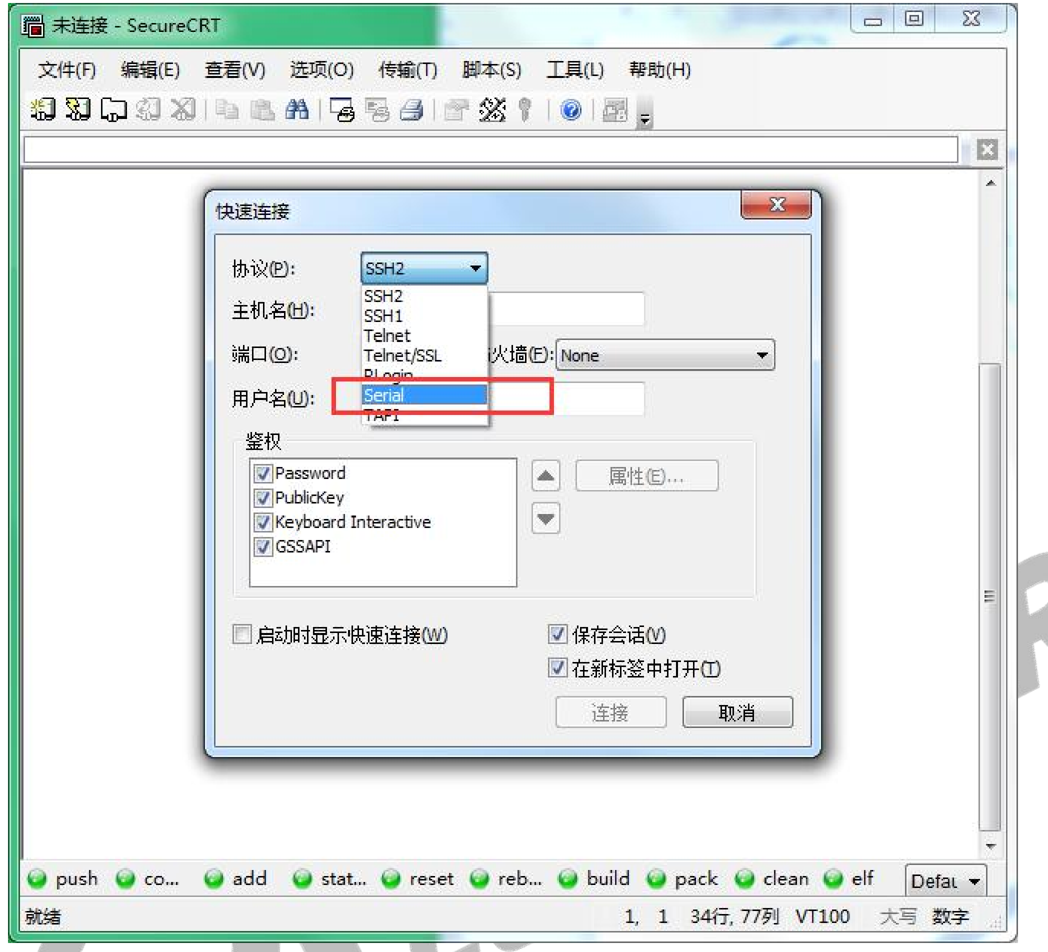
【图 8】

```
BOOT0 is starting
init dram , base         is 0x80000000
init dram , clk          is 156
init dram , access_mode  is 1
init dram , cs_num       is 0x00000001
init dram , ddr8_remap   is 0
init dram , sdr_ddr      is 1
init dram , bwidth       is 16
init dram , col_width    is 10
init dram , row_width    is 13
init dram , bank_size    is 4
init dram , cas          is 3
init dram , size         is 120
dram init successed,size is 64
jump to BOOT1
DBG: boot1 starting!
DBG: init key OK
0
jump to kernal
epos is released at 2018-11-26
commit id is 77dc3cda707b7a7b025da198eb27ac27504d1ebc
DRV_DISP_MInit.
run to esKSRV_OpenLogo
*divider=9
b_interlace=0
lcd_power_used=0 ret=0
Mount Parts Thread running.....
EXTIO ctrl ic communication failed-
exits esKSRV_OpenLogo
BUS EXTIO initialize succeeded!
 dual_output_mode = 1
g_load_sys_part 0x0 set to 0
spinor partition size is 32768
partition [D] plug in..

partition [Z] plug in..
Mount Parts Thread work now.....

partition [E] plug in..
bklight read twi data[0] = 0xFF
bklight on twi data = 0xFF
.Start Shell from partion 'c'.........................................................................
..................
Mount Parts Thread work end....
shell main enter.
.........................................................................................
```

【图 9】

下面简单介绍一下 SecureCRT 的串口使用步骤。

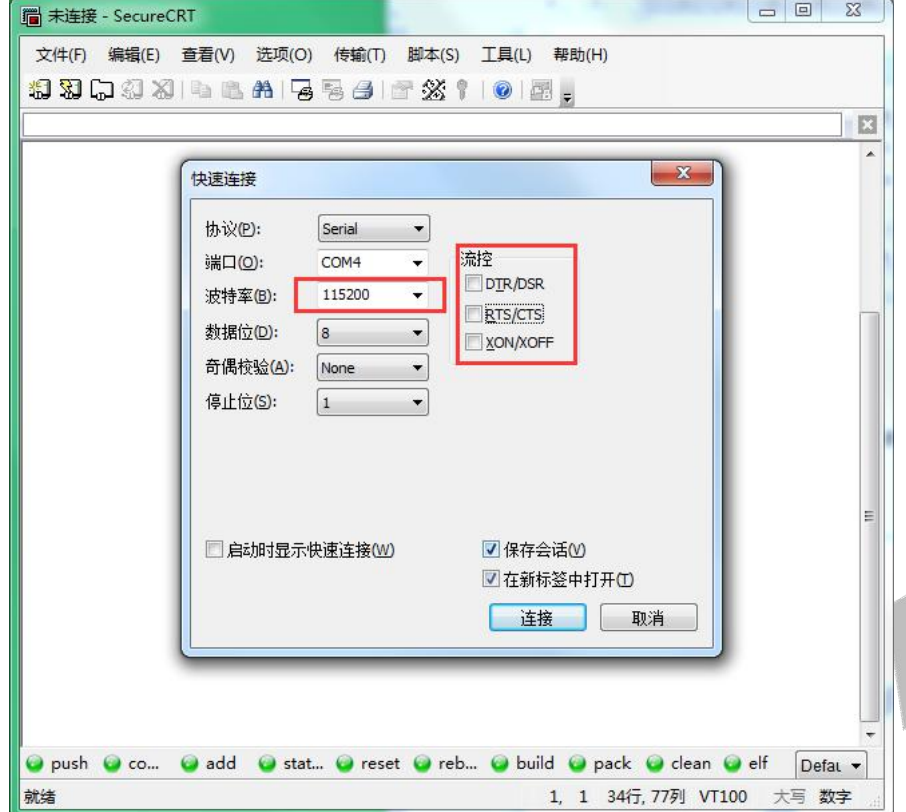
【step1】双击 图标打开 SecureCRT 软件，通过菜单“文件”->“快速链接”，或直接点击工具栏的“快速链接”按钮 来直接启动快速链接，出现如【图 10】的提示窗口，点击“协议”条目出现下拉菜单，选择“Serial”。



【图 10】

【step2】选择 Serial 协议之后，如【图 11】所示，选择指定的串口，把波特率设置成 115200bps，取消流控的勾选项，然后点击“连接”按钮即可。至此串口 SecureCRT 即可用于收发开发板 UART1 串口的数据。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved



【图 11】

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

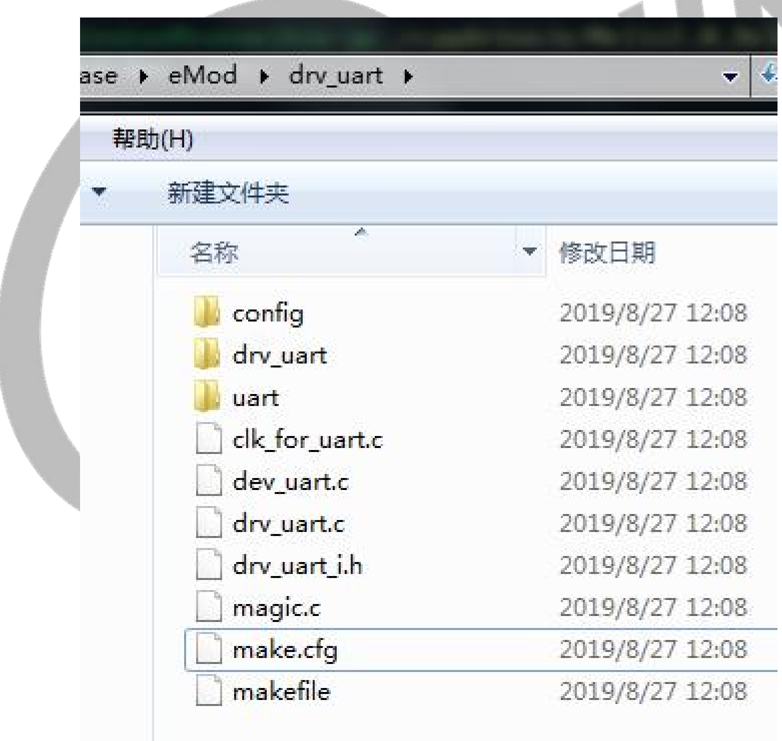
3.6. Melis2.0 添加和调用一个模块

3.6.1. 为什么划分模块？

模块在 Melis2.0 中是一个独立的可执行文件，在介绍前文 SDK 文档目录时提到过模块的概念。一个模块在 SDK 中是如何存在的？它的代码组成基本结构如何？它是如何编译生成的？在固件中如何存在？又如何被加载调用？我们接下来介绍一下。

为使 SDK 结构层次清晰，Melis2.0 将内核操作系统、驱动、中间件、应用等区分开来，单独编译链接生成 ELF 可执行文件，方便各模块独立维护。一个模块需要使用时，要先加载其 ELF 可执行文件，才能使用，不使用时，则可以卸载掉。

3.6.2. UART 驱动模块



【图 12】

例如串口驱动，进入到 eMod/drv_uart 文件夹，可以看到如【图 12】所示的内容。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

3.6.2.1.编译

【make.cfg】

1、通过包含 CROSSTOOL.CFG 指定了编译链接工具为 RVDS 以及 RVDS 工具在电脑上的存储路径；

2、通过 INCLUDES 指定了头文件的包含路径；

3、通过 LIBS 指定了库文件的包含路径；

4、通过 TARGET 指定了输出文件的名称和路径；

5、通过 SRCDIRS 指定了本文件夹及最大 3 层深度的子文件夹中所包含的所有文件名列表；

6、通过 LINK_SCT 指定了链接所使用的文件路径；

7、指定了其他编译选项；

【makefile】通过 SRCCS、SRCSS 过滤得到 SRCDIRS 变量所表示的文件名列表中的.c 和.s 后缀文件参与编译连接，通过 cygwin 进入本文件所在路径，输入 make clean;make 后按“Enter”键,就会执行此 makefile 脚本，并将 ELF 可执行文件输出到 TARGET 所指定的路径下，默认 TARGET 是 SDK 目录下的 workspace\suniv\rootfs\drv\文件夹，客户可按需自行修改。

```
/cygdrive/e/Melis2.0_Release/melis2.0-sdk-release/eMod/drv_uart

Administrator@sunweibin-pc ~
$ cd /cygdrive/e/Melis2.0_Release/melis2.0-sdk-release/eMod/drv_uart/

Administrator@sunweibin-pc /cygdrive/e/Melis2.0_Release/melis2.0-sdk-release/eMod/drv_uart
$ make clean;make
rm ./dev_uart.o ./clk_for_uart.o ./drv_uart.o ./magic.o  __image.axf
rm: 无法删除"./dev_uart.o": No such file or directory
rm: 无法删除"./clk_for_uart.o": No such file or directory
rm: 无法删除"./drv_uart.o": No such file or directory
rm: 无法删除"./magic.o": No such file or directory
rm: 无法删除"__image.axf": No such file or directory
makefile:47: recipe for target 'clean' failed
make: [clean] Error 1 (ignored)
"drv_uart.c", line 32: Warning:  #223-D: function declared implicitly
        memset(&drv_uart, 0, sizeof(__drv_uart_t));
        ^

"drv_uart.c", line 42: Warning:  #223-D: function declared implicitly
        memset(&drv_uart, 0, sizeof(__drv_uart_t));
        ^

drv_uart.c: 2 warnings, 0 errors
"C:/Program Files (x86)/ARM/RVCT/Programs/2.2/349/win_32-pentium"/armlink --noremove --entry 0xe2000000
"C:/Program Files (x86)/ARM/RVCT/Programs/2.2/349/win_32-pentium"/fromelf  --elf --no_debug --output ./.
----------------------------
target make finish
----------------------------

Administrator@sunweibin-pc /cygdrive/e/Melis2.0_Release/melis2.0-sdk-release/eMod/drv_uart
$
```

【图 13】

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

3.6.2.2.加载和使用

uart.drv 生成之后，会在打包时一起合入.img 固件烧录到 Norflash 中，要使用运行该驱动，还需要把该驱动加载到内存中，并注册给系统的设备管理器。

【magic.c】每个模块都有一个 magic.c 文件，本文件定义了一个结构体，需要注意的是“type”和“mif”元素：

【type】type 是一个无符号单字节变量，emod.h 头文件以宏定义的形式为每个模块分配了独占的“type”值，uart 按键驱动的 type 值是 EMOD_TYPE_DRV_UART，开发者如果想添加新模块，可自行添加宏定义，但与其他模块的值不可相同。

【mif】mif 表示模块的接口函数结构体，通过 esDEV_Plugin 函数加载驱动的过程中，依次调用 mif.MInit、mif.MOpen、mif.MIoctrl、mif.MClose 函数，完成 uart.drv 文件的加载、初始化、打开、设备注册和关闭。至此，开发者可通过打开设备注册名所获得的句柄来调用对应的注册函数，下面是一段 UART0 驱动使用代码。

```
{
        ES_FILE *pfuart0 = NULL;
        __uart_para_t uart0_para;

        esDEV_Plugin("\\drv\\uart.drv", 0, NULL, 0);
        pfuart0 = eLIBs_fopen("\\BUS\\UART0", 0);

        if(pfuart0)
        {
            uart0_para.nEvenParity   = 0;   //奇 偶 校 验
            uart0_para.nParityEnable = 0;   //校 验 关 闭
            uart0_para.nStopBit      = 0;   // 1bit 停 止 位
            uart0_para.nDataLen      = 0x03;//8bit 数据发送长度
            eLIBs_fioctrl (pfuart0, UART_CMD_SET_PARA, 0, (void *)&uart0_para);
            eLIBs_fwrite ("hello word!\r\n", 1, sizeof("hello word!\r\n") - 1, pfuart0);
            eLIBs_fclose(pfuart0);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
            pfuart0   = NULL;
        }
}
```

【esDEV_Plugin("\\drv\\uart.drv", 0, NULL, 0)】：如上文所属会根据"\\drv\\uart.drv"加载固件中的 uart.drv 驱动到内存，然后调用 mif.init、mif.open、mif.ioctrol、mif.close 接口，在本串口驱动中则对应的是 DRV_UART_MInit、DRV_UART_MOpen、DRV_UART_MIoctrl、DRV_UART_MClose。需要注意的是函数 DRV_UART_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer)在这个过程中的 4 各参数:

mp：DRV_UART_MOpen 的返回值；

cmd：DRV_CMD_PLUGIN；

aux：对应的是 esDEV_Plugin 函数的第二个参数 0；

pbuffer：对应的是 esDEV_Plugin 函数的第三个参数 NULL；

【pfuart0 = eLIBs_fopen("\\BUS\\UART0", 0)】：需要注意的是，参照源码 DRV_UART_MIoctrl 在处理 DRV_CMD_PLUGIN 消息的过程中调用了 esDEV_DevReg 函数，将为类名为“BUS”、设备名为“UART0”的设备及其接口函数结构体变量 uart_devop 一同注册到内核中。为使 eLIBs_fioctrl 这个接口映射成 uart_devop.Ioctl 接口，需要先通过 eLIBs_fopen 接口打开“\\BUS\\UART0”获得句柄，再将句柄传递给 eLIBs_fioctrl 接口，则此时 eLIBs_fioctrl == uart_devop.Ioctl，传参也一一对应。

eLIBs_fwrite 函数也因为 pfuart0 参数映射成了 uart_devop.Write，eLIBs_fclose 函数也因为 pfuart0 参数映射成了 uart_devop.Close；所以上述示例代码中设置波特率和数据发送的工作的得以实现。

类似的如果调用 eLIBs_fread 附带参数 pfuart0 就会映射成为 uart_devop.Read。

3.6.2.3.UART0 的 PIN 脚配置

考虑到芯片引脚封装、PCBA 成本之类的问题，目前市面上的控制器大多采用了引脚复用的方式来减少芯片管脚的使用，全志平台的芯片也采用了引脚复用的方法。

引脚的配置文件是“sdkroot\workspace\suniv\eFex\sys_config.fex”,uart 部分的配置如【图 14】所示,uart 驱动代码中的 esCFG_GetKeyValue 函数和 esCFG_GetGPIOSecData 函数会通过字符串传参来搜索下面的键值，需要使用的串口要把 uart_used 设置为 1，并将 IO 和复用功能号配置正确。本章侧重于讲解模块的加载和使用以及为配合 UART 测试所涉及到的内容，所以对 sys_config.fex 和 GPIO 不在此详述，相关内容可参考《Melis2.0 文档使用指南.xlsx》。

```
30148: [uart_para0]
30149: uart_used          = 1
30150: uart_port          = 0
30151: uart_type          = 2
30152: uart_debug_tx      = port:PF2<3>
30153: uart_debug_rx      = port:PF4<3>
30154:
30155: [uart_para1]
30156: uart_used          = 0
30157: uart_port          = 1
30158: uart_type          = 2
30159: uart1_tx           = port:PA2<5>
30160: uart1_rx           = port:PA3<5>
30161:
30162: [uart_para2]
30163: uart_used          = 0
30164: uart_port          = 2
30165: uart_type          = 2
30166: uart_tx     = port:PE7<3>
30167: uart_rx     = port:PE8<3>
30168:
```

【图 14】

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

4. 编译工具链使用

4.1. 工具链通用配置

Melis2.0 平台工具链的配置文件是“sdkroot\includes\cfgs\CROSSTOOL.CFG”。 该文件指定了 Melis2.0 平台的编译工具、硬件平台、共用库目录和相关工具的路径，SDK 中所有模块的 makefile 都会引用该配置。

该配置文件中各变量名由平台整合人员统一设置分配，用户可以使用相关的变量，但是不要对变量的名称作修改，以免造成无法编译。相关变量的含义如下：

$(CROSSTOOL)，定义交叉编译工具类型：

“ARMRVDS”，定义交叉编译工具为 RVDS；

“ARMGCC”，定义交叉编译工具为 ARMGCC；

$(EPDK_CHIP)，定义硬件平台的类型，定义在脚本“sdkroot\includes\cfgs\chip.cfg”中：

$(LIBPATH)，定义 Melis2.0 平台的共享库目录。

$(INTERLIBPATH)，定义 Melis2.0 平台内部共享库目录。

$(WORKSPACEPATH)，定义 Melis2.0 平台目标文件及打包工作路径。

$(ESTUDIOROOT)，定义 Melis2.0 平台使用的 PC 工具的路径。

$(RVDSPATH)，定义了 Melis2.0 平台使用的 RVDS 工具的安装路径。

$(CC)，定义 C 语言编译工具：

RVDS 交叉编译工具下为“armcc”；

GCC 交叉编译工具下为“arm-elf-gcc”；

$(CFLAGS)，定义 C 语言编译工具的基本配置参数。

$(AS)，定义汇编器工具：

RVDS 交叉编译工具下为“armasm”；

GCC 交叉编译工具下为“arm-elf-as”；

$(ASFLAGS)，定义汇编器的基本配置参数。

$(LINK)，定义链接工具：

RVDS 交叉编译工具下为“armlink”；

GCC 交叉编译工具下为“arm-elf-ld”；

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

$(LKFLAGS)，定义链接工具基本配置参数。

$(AR)，定义库打包工具：

```
RVDS 交叉编译工具下为“armar”；
GCC 交叉编译工具下为“arm-elf-ar”；
```

$(ARFLAGS)，定义库打包工具的基本配置参数。

$(LOAD)，定义加载器工具：

```
RVDS 交叉编译工具下为“fromelf”；
GCC 交叉编译工具下为“arm-elf-objcopy”；
```

$(LDFLAGS)，定义加载器工具的基本配置参数。

4.2. 模块的工具链配置

CROSSTOOL.CFG 仅配置了一些公共内容,在模块的 make.cfg 使用 include 关键字来使用 CROSSTOOL.CFG 中的配置。包含了公有配置的同时，可在每个模块的 make.cfg 文件中自定义私有配置。虽然是私有配置，但一般都包括下面几个部分：

$(ROOT)，定义当前模块的根目录，一般为“.”。

$(SDKROOT)，定义“sdkroot”目录相对于$(ROOT)的相对路径，此变量必须定义，CROSSTOOL.CFG 配置文件中会通过此变量来引用 SDK 的根目录。

include $(SDKROOT)/includes/cfgs/CROSSTOOL.CFG，引用编译工具通用配置。

$(INCLUDES)，定义所有需要引用的头文件的路径。

$(LIBS)，定义需要引用的库文件。

$(SRCDIRS)，定义所有需要引用的源文件的路径，一般采用自动扫描的方式来定义，不需要逐项列出。

$(TARGET)，定义需要输送出去的目标文件，一般不包含调试信息。

$(LOCALTARGET)，定义本地生成的目标文件，一般命名为“__image.axf”，包含有完整的调试信息，用作调试。

$(LINK_SCT)，定义链接程序使用的链接脚本。

除此以外，还需要对“CFLAGS”、“ASFLAGS”、“LKFLAGS”、“LDFLAGS”等相关工具配置参数做相应的扩展，以满足模块编译的特定需求。

下面是一个 make.cfg 示例：

```
ROOT    = .
SDKROOT = $(ROOT)/../../../
#导入交叉编译器通用配置
include $(SDKROOT)/includes/cfgs/CROSSTOOL.CFG

#头文件路径列表
INCLUDES        =    -I$(SDKROOT)/includes          \
                     -I$(SDKROOT)/includes/emod     \
                     -I$(ROOT)/../include
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
#库文件列表
LIBS            =   $(LIBPATH)/elibs.a  \
                    $(LIBPATH)/app_libs.a\(*.o\) \
                    $(LIBPATH)/elibs_ex.a \
                    $(LIBPATH)/app_views.a \
                    $(LIBPATH)/bsp/bsp_effict.a
#定义生成的目标文件(输出/本地)
TARGET      = $(WORKSPACEPATH)/beetles/rootfs/apps/app_root.axf
LOCALTARGET = __image.axf
#列出该工程下的所有包含源程序的子目录
SRCDIRS  =  $(shell find . -maxdepth 5 -type d)
INCLUDES       :=   $(INCLUDES)       \
                    $(foreach dir,$(SRCDIRS),-I$(dir))
ifeq ($(CROSSTOOL), ARMRVDS)
#================================================================================
#使用 RVDS 编译器
#================================================================================
#程序链接脚本
LINK_SCT    =   --scatter=$(ROOT)/config/config.sct
#编译器扩展参数
CFLAGS     := $(CFLAGS) -O0 --debug -DEPDK_DEBUG_LEVEL=EPDK_DEBUG_LEVEL_LOG_ALL
CFLAGS     := $(CFLAGS) $(INCLUDES)
#汇编器扩展参数
ASFLAGS    := $(ASFLAGS) --debug --keep
ASFLAGS    := $(ASFLAGS) $(INCLUDES)
#链接器扩展参数
LKFLAGS    := $(LKFLAGS) $(LINK_SCT)
#加载器扩展参数
LDFLAGS    := $(LDFLAGS) --elf --no_debug --output
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
else
#================================================================================
#使用 GNU-GCC 编译器
#================================================================================
error:
    $(error GNU Cross-tool is invalid now!!!)
endif
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

4.3. 简单的 makefile

Melis2.0 平台采用 makefile 的隐含规则完成对所有源文件的编译。没有启用完整的依赖规则，只有当源码文件(*.c、*.s)文件发生修改后，才会重新编译该源码文件(未修改的源码文件不会被重新编译)，修改头文件(*.h)不会引发对源码文件的重新编译，因此，当修改了相关的头文件以后，必须先执行 clean，再重新编译。一个基本的 makefile 一般包括以下几个部分：

include make.cfg，引用 makefile 的配置文件。

$(SRCCS)，通过自动扫描获得的*.c 源文件列表。

$(SRCSS)，通过自动扫描获得的*.s 源文件列表。

$(OBJS)，通过后缀替换规则从$(SRCCS)和$(SRCSS)获得的*.o 文件列表，*.o 文件通过 makefile 的隐含规则自动编译$(SRCCS)和$(SRCSS)获得。

$(LOCALTARGET):$(OBJS)，链接相关的*.o 和库文件得到本地目标文件。

all:$(LOCALTARGET)，通过本地文件得到输出目标文件，该符号也是 makefile 的默认入口。

clean，清理生成的临时文件。

一个简单的 makefile 示例文件如下：

```
#导入编译器配置
include make.cfg

#从所有子目录中得到源代码的列表
SRCCS=$(foreach dir,$(SRCDIRS),$(wildcard $(dir)/*.c))
SRCSS=$(foreach dir,$(SRCDIRS),$(wildcard $(dir)/*.s))

#得到源代码对应的目标文件的列表
OBJS=$(SRCCS:.c=.o) $(SRCSS:.s=.o)

#生成输出目标文件
all:
    make builders
    make application
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
buildres:
	$(ESTUDIOROOT)/Softwares/xmlang/xmlang.exe ./res/lang.xml
	$(ESTUDIOROOT)/Softwares/ResCompile/langcompile.exe ./res/lang.cfg
	$(ESTUDIOROOT)/Softwares/ResCompile/themcompile.exe ./res/them.cfg

application:$(LOCALTARGET)
	$(LOAD) $(LDFLAGS) $(TARGET) $(LOCALTARGET)
	@echo ----------------------------
	@echo target make finish
	@echo ----------------------------

#生成本地目标文件
$(LOCALTARGET):$(OBJS)
	$(LINK) $(LKFLAGS) -o $@ $(filter %.o ,$+) $(LIBS)

# 删除生成的中间文件
clean:
	-rm $(OBJS) $(LOCALTARGET)
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

5. 固件烧录工具的安装

5.1. PhoenixSuit 的安装步骤

【step1】双击 图标，开始 PhoenixSuit 的安装。如【图 1】所示，点击安装向导“下一步”按钮，。

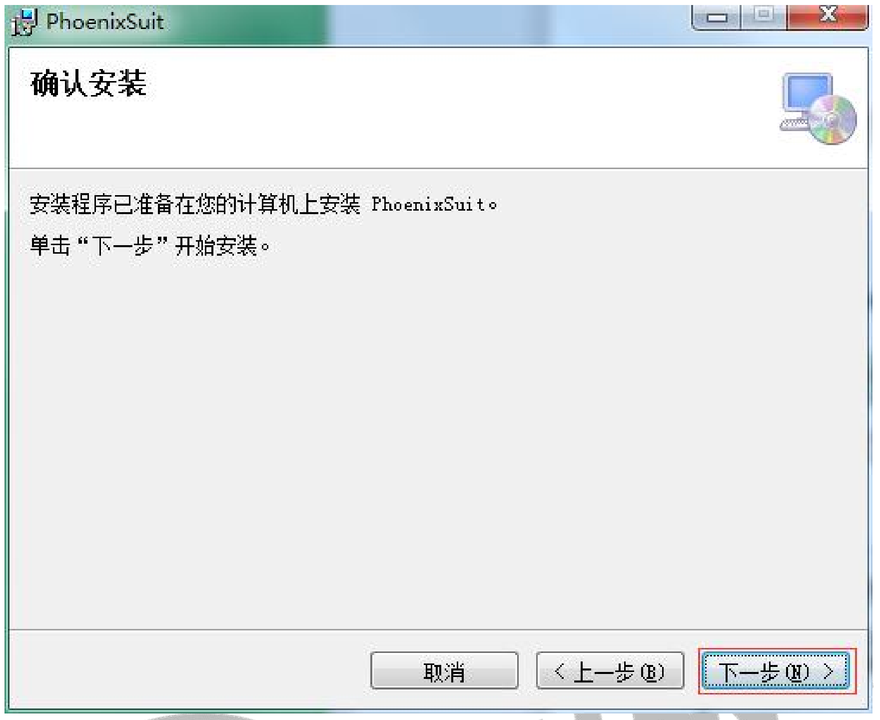
【图 1】

【step2】如【图 2】所示，默认安装路径，也可点击“浏览”按钮自定义安装路径，点击“下一步”按钮；

【图 2】

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

【step3】如【图 3】所示，点击“下一步”按钮确认安装；



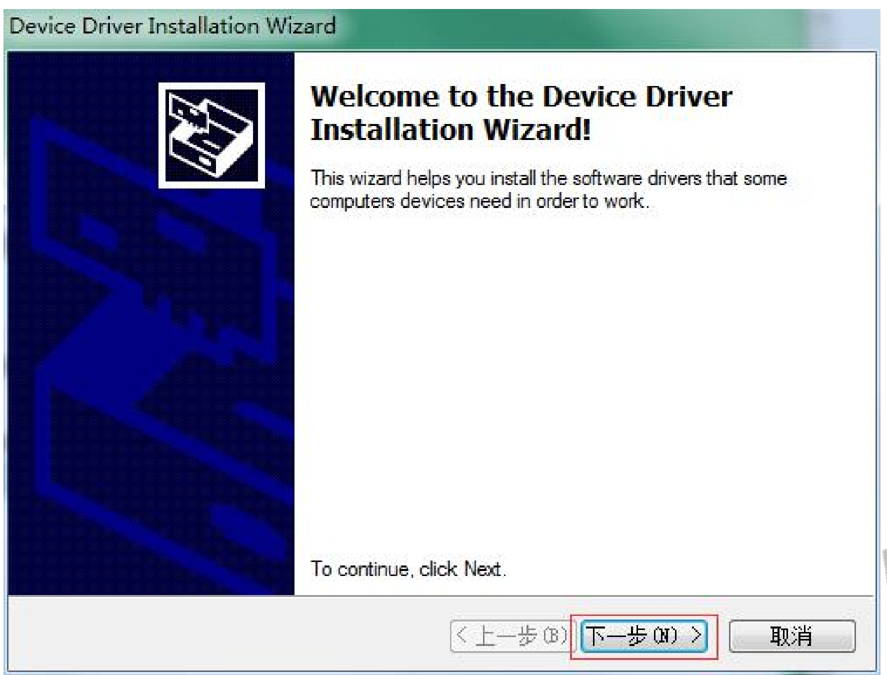
【图 3】

【step4】如【图 4】所示，进入安装，进度条显示安装进度；

【图 4】

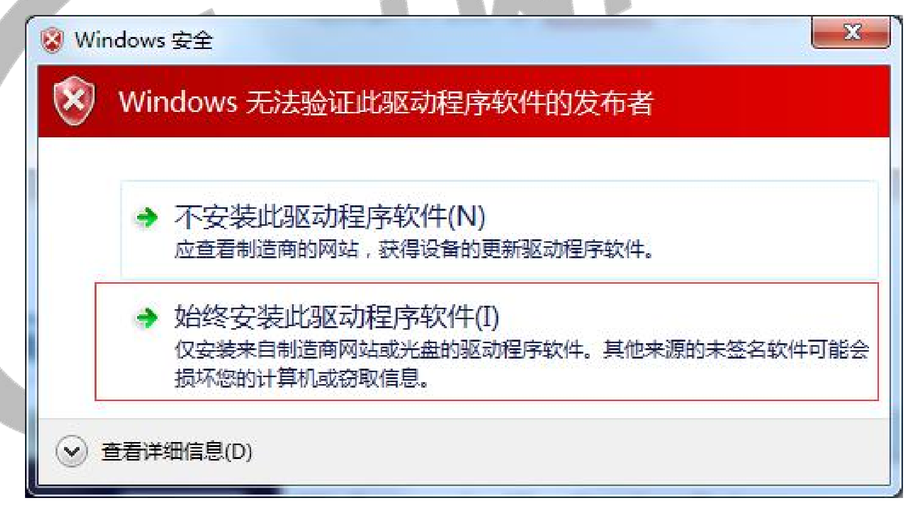
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

【step5】如【图 5】所示，安装过程中会弹出 USB 驱动安装提示窗口，点击“下一步”继续安装；



【图 5】

【step6】如【图 6】所示，如弹出无法验证发布者按钮，请点击“始终安装此驱动程序软件”继续安装；



【图 6】

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

【step7】如【图 7】所示，驱动安装完成后弹出提示窗口，点击“完成”按钮结束 USB 驱动的安装；

【图 7】

【step8】如【图 8】所示，【step7】的 USB 驱动的安装完成后，PhoenixSuit 会继续自动安装，安装完成后会出现“安装完成”提示窗口，点击“关闭”按钮结束安装。

【图 8】

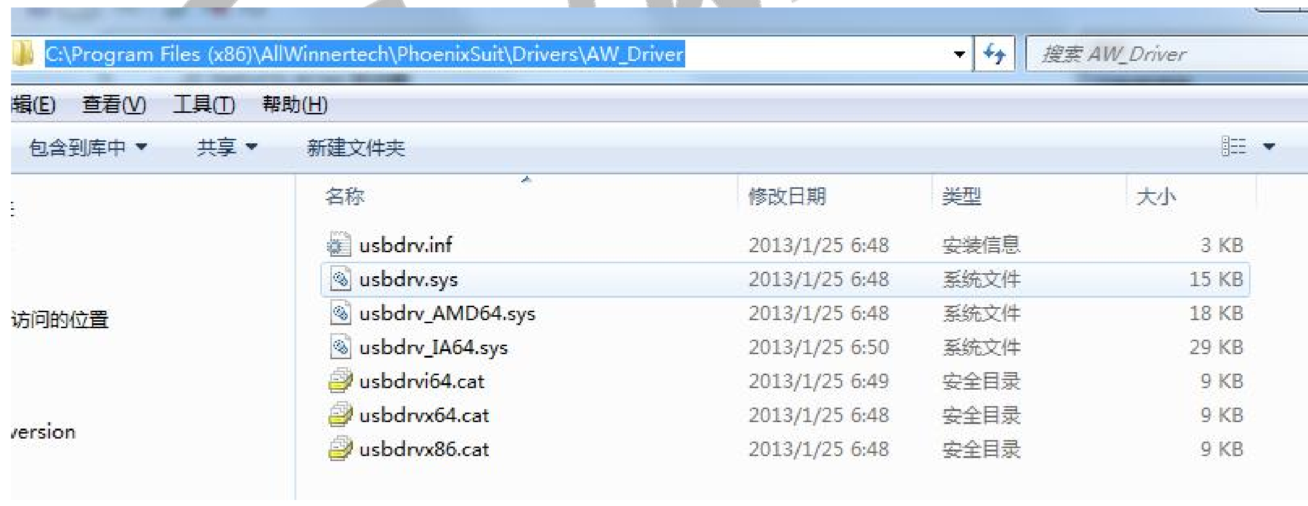
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

5.2. 检验 USB 驱动安装

PhoenixSuit 安装完成后，可打开 windows 的“设备驱动管理器”，点击“通用串行总线控制器”弹出的设备条目，查看是否有“VID_1f3a_PID_efe8”的 USB 设备，如【图 9】所示。

【图 9】

如果开发者想要重新安装驱动，驱动在 PhoenixSuit 的安装文件夹中也有备份，参考下图【图 10】所示。



【图 10】

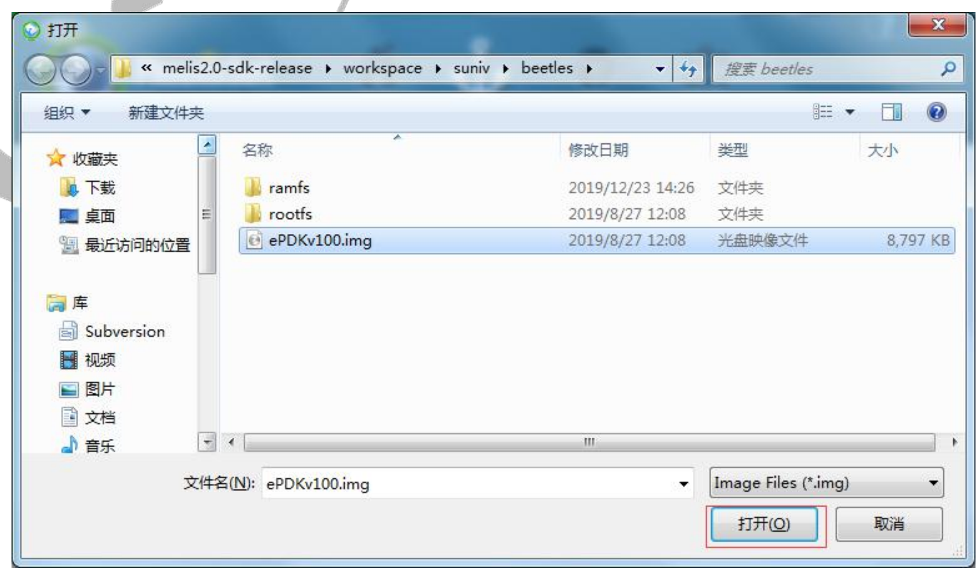
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

5.3. 使用烧录软件 PhoenixSuit

【step1】双击 图标，运行 PhoenixSuit 工具，点击“一键刷机”选项卡，如【图 11】所示

【图 11】

【step2】点击【图 11】的“浏览”按钮来指定固件的存放路径，参考【图 12】，选中.img 后缀的固件文件之后，点击“打开”按钮完成固件选择。



【图 12】

【step4】选择好固件之后如【图 13】所示。接下来参考【图 14】，开发板断电的情况下，短路 Norflash 的 1、2 引脚，通过 USB 线将电脑和开发板相连，再给开发板上电，上电 1~2 秒后松开 Norflash 的短路引脚即可。如开发板本身是通过 USB 的 5V 供电，即 USB 链接电脑和开发板时就已经上电，则按照“先短路，后供电”的原则操作本步骤。

【图 13】

【图 14】

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

【step5】步骤 4【step4】完成之后，开发板就会跟电脑交互通信，自动进入烧录模式，PhoenixSuit 出现进度条提示烧录进度，如【图 15】所示。至此，开发者等待烧录完成即可。

【图 15】

【step6】烧录完成后，会出弹出烧录成功提示窗口，则表示烧录成功完成，开发者即可重启运行新固件。

【图 16】

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

6. 固件打包脚本

6.1. 概要描述

打包固件是下载前的最后一步。系统应用程序、驱动编写完成，经过编译得到输出文件，加上各种中间件、资源文件、配置文件、系统内核等统一打包生成固件，固件下载到开发板上就可以运行了。在 melis2.0 中，是利用批处理 image.bat 进行打包操作。打包主要工作有三方面。一方面在生成的各种源文件中拷贝需要的文件到打包路径下，第二方面利用更新工具对源文件进行更新，第三方面根据配置文件将源文件按照一定规则进行打包操作，最终生成可以烧录的固件。

6.2. 术语定义

6.2.1. makefile

makefile 是自动化编译脚本。makefile 文件描述了工程的编译和连接规则。包括工程中那些源文件需要编译以及如何编译、需要调用那些库文件，设置最后生成文件的路径等等。每个应用程序、模块等工作目录都有几个特定文件，包括 makefile magic.c make.cfg。Magic.c 是模块、驱动的入口，具有统一形式。Make.cfg 是配置文件，目前主要关注的是 target，它指明源文件生成路径以及生成文件的名字。

6.2.2. image.bat

bat 批处理文件是一系列 dos 命令的集合。文件的每一行都是一条 dos 命令。将特定命令编写完成运行就可以简化日常或重复性的任务。Image.bat 文件是打包批处理文件，运行它就可以生成最新的固件。

6.3. 工具介绍

1、update_boot 工具：update_boot 会根据配置文件，修正 boot 中的参数。

2、BurnMBR 工具：BurnMBR 工具主要是用来生成 MBR 数据，其输出文件为 mbr.bin。

3、minfs 工具：制作 minfs 分区。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

4、fsbuild 工具：fsbuild 工具用于制作 fat16 文件系统。

5、fix_file 工具：fix_file 工具较为简单，只是将输入文件延长为指定长度的输出文件，其中的延长部分是 0。

6、dragon 工具：打包。

6.4. 打包步骤

6.4.1. makefile 部分

首先对整个工程进行编译，生成源文件。在 cygwin 环境下在工程根目录使用 make clean;make 命令对整个工程进行编译（如果只修改其中一些部分可以只编译修改部分）。Makefile 会将各部分生成文件分类存放到指定路径。本工程生成路径是 ROOT\c500_net_theater\workspace\suniv。

6.4.2. image.bat 部分

固件打包的工作路径是 ROOT\c500_net_theater\workspace\suniv\beetles。打包固件以前，要确认配置文件正确，并且 makefile 生成的源文件是最新的。

Image.cfg 是配置文件，它指定了那些文件需要打包到固件中，设置固件名字、版本等信息。在生成的 image.txt 文件中可以看到各个被打包文件的信息，包括大小、路径、主键、子键等信息。

Image.bat 的工作流程如下：

1、首先进行一些初始化操作。将原有的镜像删除，设置打包工具路径等。

2、运行 update.bat 批处理文件。查看生成的 update.txt 文件，update.bat 主要是完成文件复制操作，将需要使用的文件按照分类存放到打包路径待用。

3、使用打包工具解析脚本文件（.fex 文件）、更新 boot0 和 boot1 文件头、生成 MBR 文件、更新 fes1、uboot 文件头。如 Boot 阶段使用的__boot0_file_header_t 结构体，在本阶段使用脚本文件更新部分参数，包括 dram、uart、jtag_para、spi 参数等。

4、根据配置文件，生成 MBR 文件，大小为 1kb。

5、更新 usb 烧录文件，根据系统配置文件 sys_config.fex 来修正 fes1 和 uboot 的各项参数。

6、生成文件系统镜像，运行 fsbuild.bat。根据 rootfs.ini、ramfs.ini、udisk.ini 文件将需要的文

件进行打包,分别生成 rootfs.iso、ramfs.iso、udisk.iso。根据不同的命令制作 rootfs.iso、ramfs.iso 两个 minfs 文件系统镜像，和 udisk.iso 的 fat 文件系统镜像。

7、对 boot0 和 boot1 的文件大小进行填充 boot0 为 24kb，boot1 为 120kb。

8、将 boot0.bin boot1.bin mbr.bin rootfs.iso udisk.iso 连接打包成为 melis100.fex 文件，并且生成校验文件 verify.fex。

9、运行 deagon 工具，根据配置信息 image.cfg 生成镜像包文件 ePDKv100.img。

10、删除清理临时文件，可以在脚本中用#注释掉一些删除，观察临时文件的大小是否符合预期。

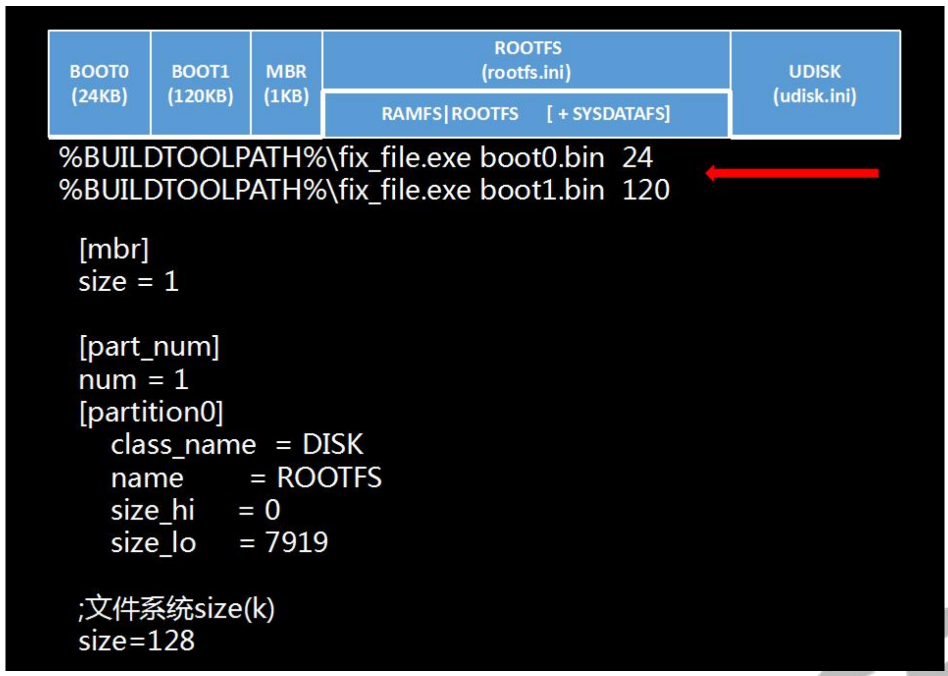
6.5. 问题与解决方案

6.5.1. 固件由那些文件构成

固件由资源文件、配置文件、系统内核、应用程序、驱动、中间件构成。

6.5.2. melis100.fex 文件包含什么内容

以 8M 固件系统未加载 sysdata 分区为例，melis100.fex 是在 do_checksum()中生成的。melis100.fex 是固件里面的重要内容,包含了 boot0.bin boot1.bin mbr.bin rootfs.iso udisk.iso 共计 5 个部分。其中 boot0 大小为 24KB，boot1 大小为 120KB，MBR 大小为 1KB，这三部分固定不变，同时 boot 部分都预留空间，保留。Rootfs 部分为 7919KB，udisk 部分为 128KB。整个文件大小为 8192KB。具体模型如下图：



6.5.3. ramdisk.iso

Ramdisk 在系统启动阶段提供帮助。系统启动时需要挂载根文件系统，根文件系统包含了各种驱动和模块和各种源文件（内核是精简的，不可能把所有的驱动和模块编译进内核，这会让内核很大）。问题就在于挂载根文件系统却需要根文件系统上面的驱动和模块，这是相互矛盾的问题。Ramdisk 包含了系统启动必要的驱动和模块，在启动阶段 ramdisk 和内核都被预先加载到内存中，再通过 ramdisk 里边的驱动和模块帮助启动系统，最终挂载根文件系统。

6.5.4. udisk.iso

Udisk 保存系统运行过程中需要保存的用户数据，包括音量、语言设置等信息。

6.5.5. 如何对 sysdata 分区进行添加

(1)更新打包工具

更新\tools\build_tools 目录下的 BurnMBR.exe 和 script.exe 工具；

(2) 配置 sysdatafs 盘符，修改分区大小

在 sys_config.fex 文件中按需求添加 sysdatafs 盘符，字符串需带引号”” 表示才能解析；打包路径下的 sysdata.ini,rootfs.ini,udisk.ini 里面的 size 大小需与配置文件中对应。并且要注意，如果 sys_config.fex 中不存在 UDISK，这个分区，则在工具中会自动加上，所以，当 part_num 为 2 时，总共有三个分区。

```
504: ;--------------------------------
505: [part_num]
506: num = 2
507:
508: [partition0]
509:     class_name  = "DISK"
510:     name        = "ROOTFS"
511:     size_hi     = 0
512:     size_lo     = 7663
513:
514: [partition1]
515:     class_name  = "DISK"
516:     name        = "SYSDATAFS"
517:     size_hi     = 0
518:     size_lo     = 256
519:
```

(3) 创建 sysdatafs.iso 镜像

fsbuild.bat 文件添加如下语句以创建 sysdatafs.iso，大小在 beetles 文件夹下的 sysdatafs.ini 文件中配置；（注：在第（2）步中定义的分区大小，需要和 ini 后缀文件中定义的大小需一致）

.\..\..\..\tools\build_tools\fsbuild200\fsbuild.exe .\sysdatafs.ini

(4) 添加 sysdatafs.iso 镜像到固件

在打包文件 image.bat 中将 sysdatafs.iso 添加到固件中：

type boot0.bin boot1.bin mbr.bin rootfs.iso sysdatafs.iso udisk.iso > melis100.fex

(5) 更新 spinor.drv 驱动

6.5.6. 分区对齐设置

由问题一可知分区分布情况，其中 sysdata 和 udisk 分区是可读写分区，norfalsh 是必须先将需要写的区域进行擦除，才能正常进行写操作，而擦除 nor 的操作暂时是以 64K 为单位，因此要求 sysdata 和 udisk 分区的起始地址和大小都需要 64K 对齐。（sysdata 和 udisk 分区数据保存请参考文档《用户数据保存》）当分区未按 64K 对齐时，可能引发用户数据保存失败等未知错误。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

6.5.7. 固件烧录后打印提示 spinor 不支持

在首次使用机器烧录固件，或者更换新 NORFLASH 后烧录固件，有可能出现下图打印：

```
Serial-COM44 - SecureCRT
文件(F) 编辑(E) 查看(V) 选项(O) 传输(T) 脚本(S) 工具(L) 窗口(W) 帮助(H)
Serial-COM44
BOOT0 is starting
init dram , base        is 0x80000000
init dram , clk         is 156
init dram , access_mode is 1
init dram , cs_num      is 0x00000001
init dram , ddr8_remap  is 0
init dram , sdr_ddr     is 1
init dram , bwidth      is 16
init dram , col_width   is 10
init dram , row_width   is 13
init dram , bank_size   is 4
init dram , cas         is 3
init dram , size        is 120
dram init successed,size is 32
jump to BOOT1
DBG: boot1 starting!
DBG: init key OK
0
jump to kernal
epos is released at 2018-11-26
commit id is 77dc3cda707b7a7b025da198eb27ac27504d1ebc
DRV_DISP_MInit.
run to esKSRV_OpenLogo
*divider=9
b_interlace=0
lcd_power_used=0 ret=0
Mount Parts Thread running.....
EXTIO ctrl ic communication failed-
exits esKSRV_OpenLogo
BUS EXTIO initialize succeeded!
 dual_output_mode = 1
g_load_sys_part 0x0 set to 0
this spinor is not support, rdid is 0x17701c, rems is 0x0cannot find spinor info
rmation
spinor partition size is -1
```

此时打印提示 spinor 不支持,并且给出了 norflash 的 id 为 0x17701c。因此我们需要在 sys_config.fex 文件中将该型号的 nor 添加上去，具体如何修改，请参考文件中的说明。完成 norflash 型号的添加后如下图：

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
424 ;------------------------------------------------------------
425
426 [spinor_para]
427 spinor_patten_num = 5
428
429 [spinor0]
430 rdid=0x001620c2
431 capaticy=32
432 freq_read=33000000
433 freq=86000000
434
435 [spinor1]
436 rdid=0x00469d7f
437 capaticy=32
438 freq_read=10000000
439 freq=50000000
440
441
442 [spinor2]
443 rdid=0x001820c2
444 capaticy=128
445 freq_read=33000000
446 freq=86000000
447
448 [spinor3]
449 rdid=0x1840ef
450 capaticy=128
451 freq_read=33000000
452 freq=86000000
453
454 [spinor4]
455 rdid=0x17701c
456 capaticy=32
457 freq_read=10000000
458 freq=50000000
459
```

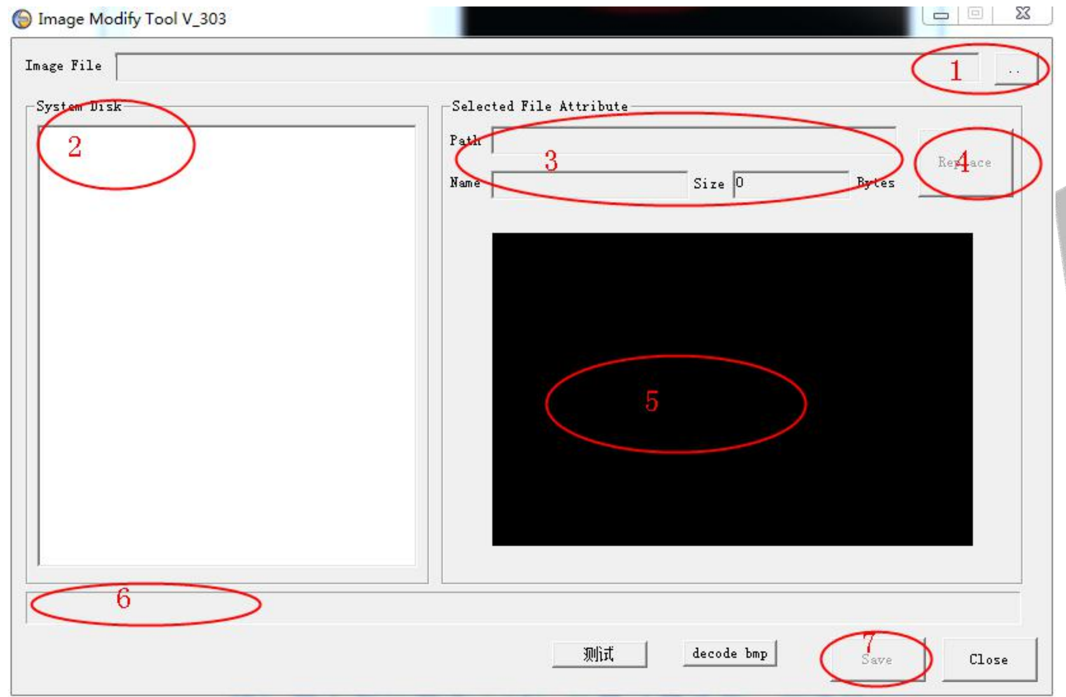
全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

7. 固件修改工具(ImageModify)使用

7.1. 界面说明

ImageModify 工具路径为“SDK\softwares\ImageModify”。ImageModify 的作用是通过该工具直接替换固件内部分文件，生成一个新固件。

主界面如下：



1. 选择固件文件按钮
2. 替换目标的系统盘文件浏览
3. 选中的文件信息
4. 替换按钮
5. 图片浏览区
6. 进度条
7. 保存按钮

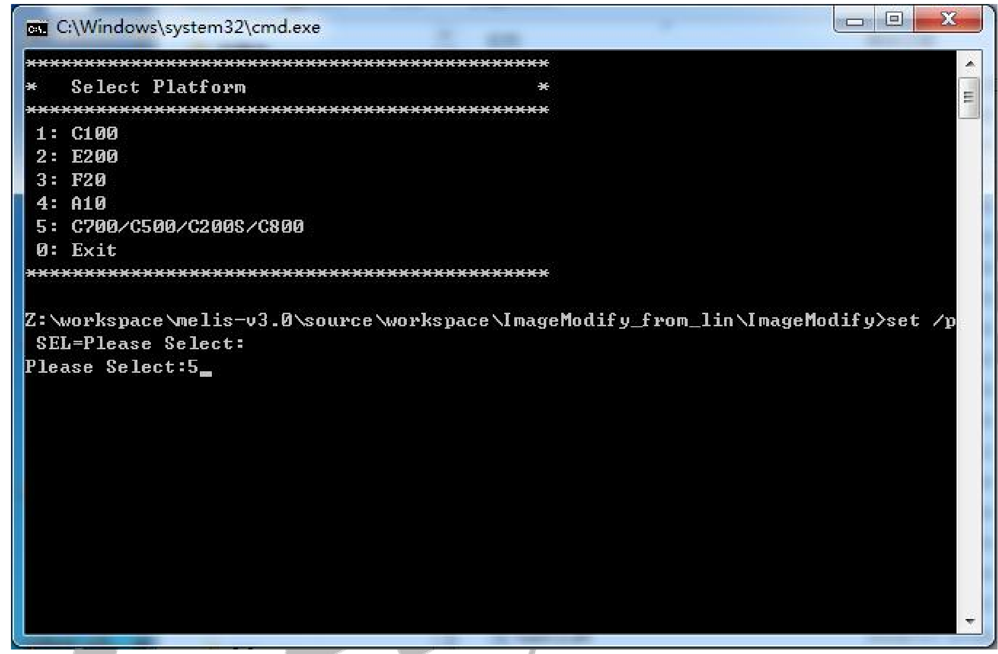
注：由于软件版本不同，界面可能存在一点差异。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

7.2. 操作步骤

7.2.1. 配置平台

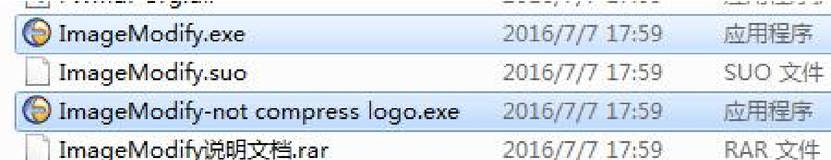
在软件运行目录下运行批处理文件 config.bat。此批处理具体作用可参考软件目录下文档《固件修改工具 V300 说明文档》。



根据平台的类型选择不同的数字，以 C200S 为例子，则选择 5 即可。

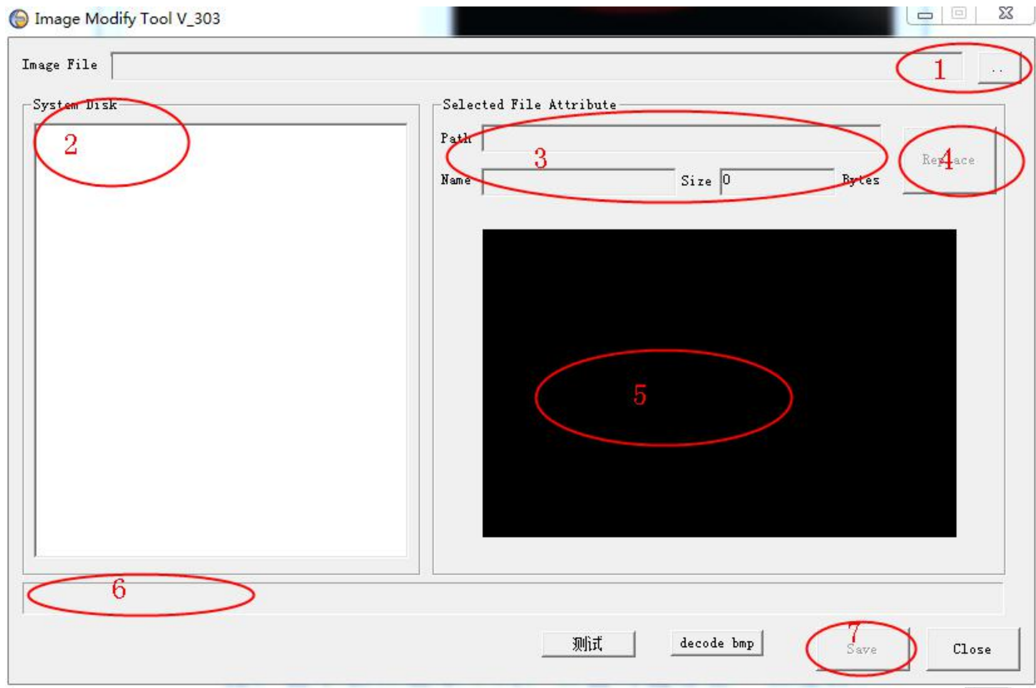
7.2.2. 选择固件

在运行目录下启动固件修改工具 ImageModify.exe 或者 ImageModify-not compress logo.exe。

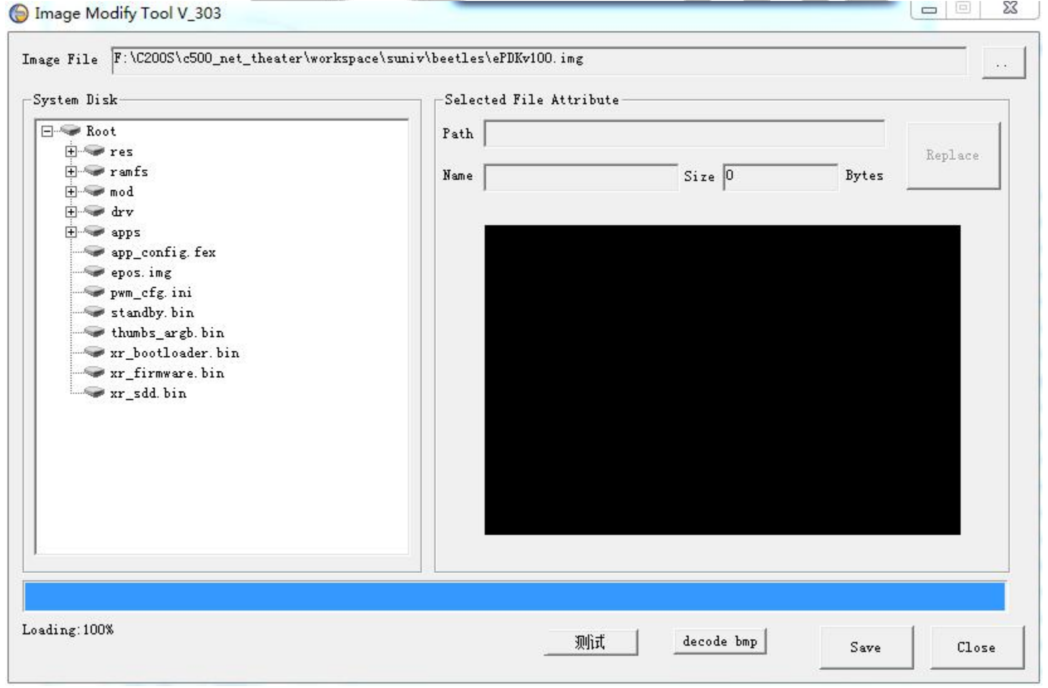


运行界面如下

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

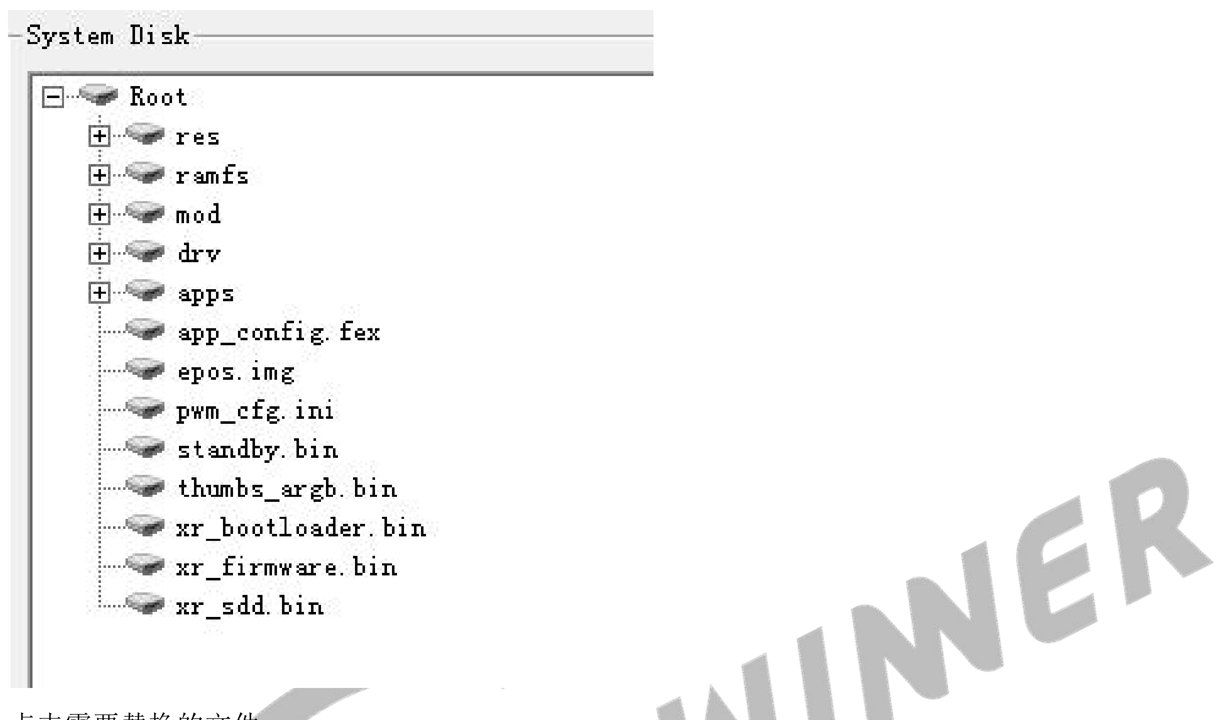


点击，选择要打开的固件文件。固件打开后有如下界面：

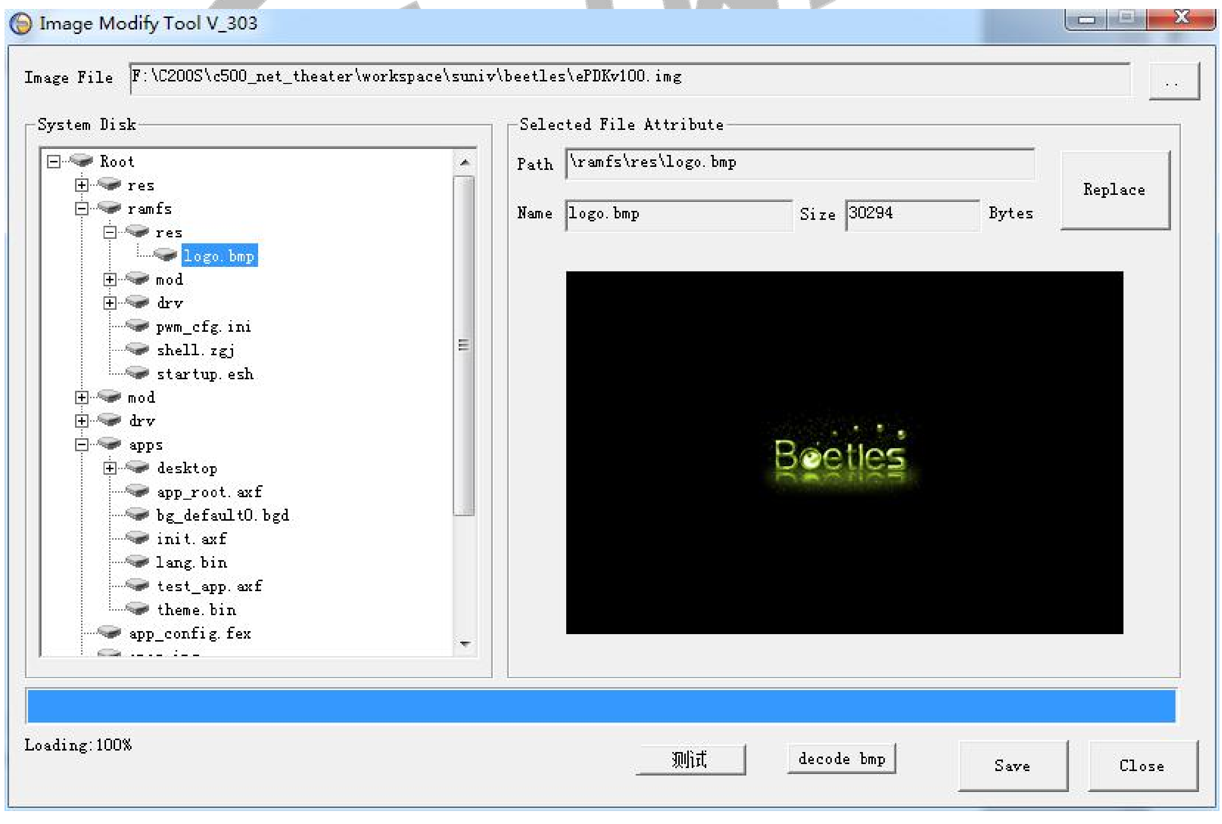


7.2.3. 选择要替换的文件

固件打开后，在目标替换文件浏览区可以看到该分区下的所有目录和文件，如下图



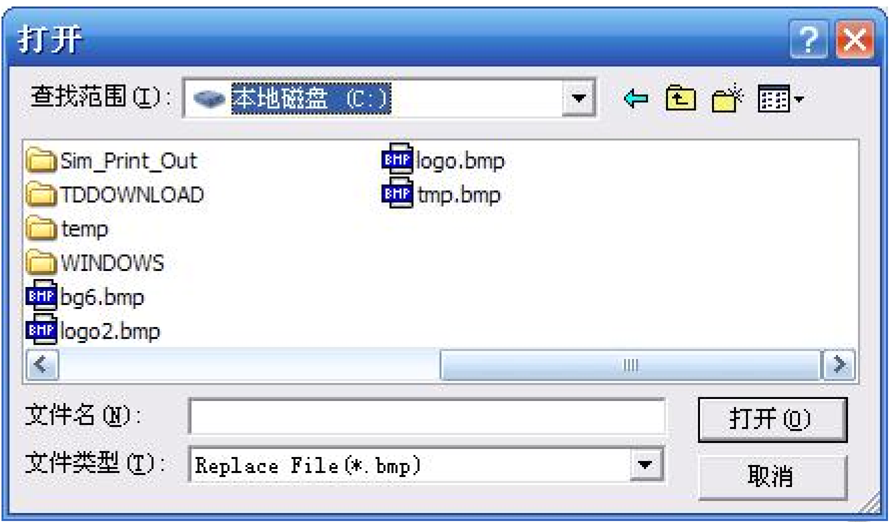
点击需要替换的文件



在上图可以看到选中的文件

7.2.4. 替换文件

在步骤 3 中选中文件后，点击按钮 Replace ，选择新的文件进行替换。



7.2.5. 保存固件

当替换完所有要替换的文件后，点击按钮 Save ，等待进度条提示如下

表示保存成功。会在打开固件目录下会生成一个.img 的新固件。

名称	修改日期	类型	大小
ramfs	2020/4/1 11:16	文件夹	
rootfs	2020/4/1 11:16	文件夹	
udisk	2019/12/13 10:52	文件夹	
ePDKv100.img	2020/4/1 11:17	光盘映像文件	8,797 KB
ePDKv100_20200401_104509.img	2020/4/1 10:45	光盘映像文件	8,797 KB
fsbuild.bat	2019/11/26 9:42	Windows 批处理...	2 KB
hdma	2020/1/13 14:40	文件	0 KB
image.bat	2020/4/1 11:16	Windows 批处理...	7 KB

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

7.3.注意事项

nor 系统的配置脚本要更新一下，参见软件工具目录下“nor 系统配置脚本”文件夹,修改了

1. update.bat 中增加

   @copy .\ramfs.ini .\ramfs\ramfs_ini.tmp

   @copy .\rootfs.ini .\rootfs\rootfs_ini.tmp

2. image.bat 文件中增加

   del .\ramfs\ramfs_ini.tmp

   del .\rootfs\rootfs_ini.tmp

3. beetles\rootfs 里面添加 app_config.fex 文件。可参考修改工具\nor 系统配置脚本下面的修改范例

4. 若 beetles\rootfs 里面 app_config.fex 文件存在则无需添加。app_config.fex 文件内容也有限制，app_config 里面结构基本是类似于主键和子键的结构形式，如下所示

```
1  [key_para]
2  ;;;K14 K12... 这些名称只要与数组g_key_pad_name2val\matrix_keymapping定义一致即可，具体是什么名字不限
3
4
5  SW4=string:EPG
6  SW7=string:RIGHT
7  SW9=string:PREVIEW_LIST
8  SW5=string:SCAN
9  SW8=string:ENTER
10 SW6=string:MENU
11
12
13
14 [system]
15 ;fm search thread hold, (1-63) default is 4
16 fm_th=4
17
18 ;default volume 0-30
19 volume=30
20
21 ;default language 0x400/0x410/0x420 en/chs/cht
22 lang=0x410
23
24 ;default background 0-1
25 style=0
26
27 ;default back light 0-15
28 light=12
29
30 ;default keytone off/on 0/1
31 keytone=0
32
33 ;default cvbs output ntsc/pal 0/1
34 cvbs=0
35
36 [client]
37 clientid=4C4E
```

此处仅为举例，不考虑 client 的实际作用。

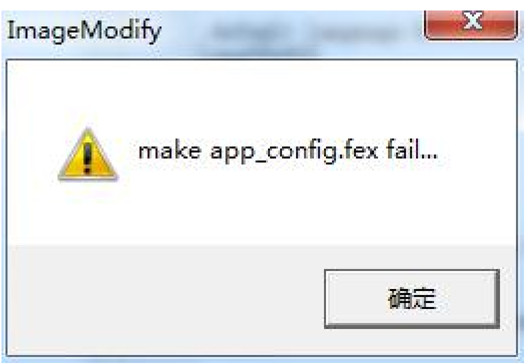
[client]

clientid=4C4E

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

如上例子键等号后面的值如果数字开头不能带有字母，否则在 Save 时，会提示如下错误。



子键等号后面的值如果以字母开头或者 0x 开头则可为任意字符串。正常保存生成新固件。如下图：

```
[key_para]
;;;;K14 K12... 这些名称只要与数组g_key_pad_name2val\matrix_keymapping定义一致即可，具体是什么名字不限


SW4=string:EPG
SW7=string:RIGHT
SW9=string:PREVIEW_LIST
SW5=string:SCAN
SW8=string:ENTER
SW6=string:MENU



[system]
;fm search thread hold, (1-63) default is 4
fm_th=4

;default volume 0-30
volume=30

;default language 0x400/0x410/0x420 en/chs/cht
lang=0x410

;default background 0-1
style=0

;default back light 0-15
light=12

;default keytone off/on 0/1
keytone=0

;default cvbs output ntsc/pal 0/1
cvbs=0

[client]
clientid=0x4C4E
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

7.4. 增加固件修改权限设置

7.4.1. 概述

为了避免用户随意修改或者保护固件修改权限，需要在固件中增加限制修改的方法。通过这种机制，固件发布者可以设置允许修改的文件列表，进入修改权限的密码等信息。

7.4.2. 操作说明

7.4.2.1. 打包

将 modifycfg.fex 文件放在打包路径下，在 image.cfg 文件中的[FILELIST]段中增加如下项

{filename = INPUT_DIR"beetles\\modifycfg.fex", maintype = ITEM_COMMON, subtype = "MDF_CONFIG000000",},

modifycfg.fex 文件是设置修改权限的文件。

7.4.2.2. modifycfg.fex 文件编辑说明

- 设置终端客户验证修改权限的密码

Password="12345"

如果不需要密码，则将该字段设置成 Password=""即可。

- 设置一级代理商验证修改权限的密码

xPassword="12345"

如果不需要密码，则将该字段设置成 Password=""即可。

- 设置是否限制

Limited=1

= 1 表示限制，=0 表示全部可以修改

- 设置允许修改的文件列表

```
FilesTab=
{
    {File=" \\res\\boot_ui\\logo.bmp"},
    {File=" \\apps\\desktop.bin"},
    {File=" \\drv"},                --定义 drv 整个目录都可以修改
}
```

如果整个目录可修改，就定义一个目录名称即可，如例子中的

```
    {File=" \\drv"},
```

备注：
如果是修改 bin 档(非 img 档)，那么权限设置功能无法生效，所有文件将可见并可替换。

8. 系统启动流程

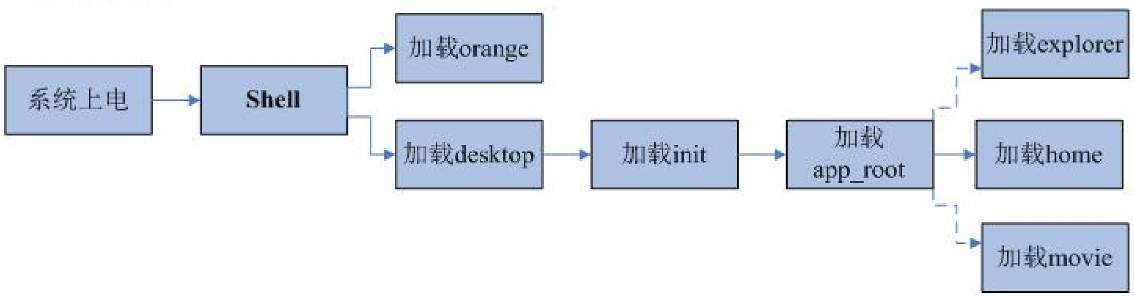
系统加载流程：boot0- boot1-kernel（epos.img）-shell

当硬件目标平台上电后，BROM 就会将 BOOT0 装载到 SRAM 中，并从 SRAM 开始执行，将控制权交给 eBoot。eBoot 的启动流程分为两个阶段：

第一个阶段是 BOOT0 加载 BOOT1，BOOT0 将 BOOT1 装载到 DRAM，在 DRAM 上执行；

第二个阶段是 BOOT1 加载系统内核，BOOT1 将内核装载到 DRAM。

每个阶段分别初始化各自所须的硬件资源，同时也为下一个阶段做好准备工作。本文主要说明从 shell 部分到 home 应用的建立。具体流程如下图：



下面结合代码进行系统加载流程说明：

8.1. Shell 部分

系统初始化完成之后首先进入 Shell 进行初始化操作（路径：suniv\beetles\ramfs\ shell.zgj）。 Shell 部分主要函数是 shellmain()它主要调用三个函数，分别是 Esh_init() 、Esh_StartUp()、Esh_ReaderLoop()。

Esh_init():主要完成一些必要文件的获取路径，申请资源等操作。

```
Esh_strcpy(Esh_Global->PWdir, "C:\\APPS");                /* set work directory */
Esh_strcpy(Esh_Global->AppPath1, BEETLES_APP_ROOT"APPS"); /* set seach app directory */
Esh_strcpy(Esh_Global->StartUpScript, "C:\\startup.esh");      /* set startup script
file */
```

Esh_StartUp():检查并执行 Esh_init()函数获取的 script。（路径：suniv\beetles\ramfs\ startup.esh）

```
StartUp_Script = (const char *)(Esh_Global->StartUpScript);         /* get startup script
file name */
doscript(NULL,   args);                                      /* execute setup script */
```

Startup.esh:脚本执行 startx。查看 Esh_builtin.c 文件中定义：shell 命令 startx 调用的是 dostart.c

```
static const struct builtincmd Commands_Table[] = {              /* builtin commands
table */
     { "startx"            , dostartx           },              /* Eshell execute
commands */
```

dostartx.c:__exec_startx()。该函数完成 desktop.mod 和 orange.mod 两个模块的加载和检查。

```
mid_mod_gui  =  esMODS_MInstall(BEETLES_APP_ROOT"mod\\orange.mod",  0);          /*install
orange.mod*/
mod_gui= esMODS_MOpen(mid_mod_gui, 0);                                 /*openorange.mod*/
mid_mod_desktop    =    esMODS_MInstall(BEETLES_APP_ROOT"mod\\desktop.mod",    0);/*install
```

Esh_ReaderLoop()：主要完成 shell 命令读取和执行。等待模块、驱动、窗口等部分安装初始化完成后，在循环中不断读取串口的调试命令进行处理。可执行的命令均在 Esh_builtin.c 文件中。

```
while( 1 )                                              /* read input commands loop */
{
    if (Esh_Global->ExitFlag == ESH_TURE) break;             /* check need exit or not
first */
    __Esh_getcmd(cmd_buf, ESH_MAX_CMD_LEN);                /* get a command */
    __Esh_Insert_cmd(cmd_buf);                              /* insert command to history
list */
    if (cmd_buf[0])  Esh_execute_one_command(cmd_buf);          /* execute a command */
```

8.2. Orange 和 desktop 部分

模块加载：以 desktop.mod 为例，查看 mod_desktop 文件夹下面的 make.cfg 有

TARGET   = $(WORKSPACEPATH)/beetles/rootfs/mod/desktop.mod

查看入口代码文件 magic.c 找到初始化接口：在 desktop 模块的 MOpen 函数中加载了 init 模块：

```
__mp *DESKTOP_MOpen(__u32 mid, __u32 mod)
{
    __msg("---------------------DESKTOP_MOpen -------------------------\n");
    desktp_data.mid  = mid;
    desktp_data.init_id = esMODS_MInstall(BEETLES_APP_ROOT"apps\\init.axf", 0);
    desktp_data.init_mp = esMODS_MOpen(desktp_data.init_id, 0);
    return (__mp *)&desktp_data;
}
```

模块加载：查看 init 文件夹下面的 make.cfg 有

TARGET   = $(WORKSPACEPATH)/beetles/rootfs/apps/init.axf。

首先查看 magic.c 文件，在 MOpen 函数中创建了一个线程 application_init_process。

```
init_data.init_tid    =    esKRNL_TCreate(application_init_process,    NULL,    0x10000,
KRNL_priolevel4);
```

Desktop 模块加载完成之后加载 init 模块。Init 模块创建一个应用初始化线程 application_init_process。线程首先装载必要的驱动，例如音频驱动、按键驱动等。接下来是卡量产的必要准备工作，这里不做过多研究。该线程最重要的三项工作是：

1、创建主管理窗口，用于消息的接受和预处理。该窗口名称为 init_mainwin。关于窗口的分类以及功能见其他章节。窗口创建时向自身发送 GUI_MSG_CREATE 消息，进行初始化操作。

```
init_mainwin = init_mainwin_create();              /* create mainwin */
GUI_SetActiveManWin(init_mainwin);               /* set active  manwin */
message_loop_win_set_default(init_mainwin);      /* set message loop win */
```

init_mainwin 消息主管理窗口负责消息预处理，新消息先经过本窗口，收到之后在回调函数 init_mainwin_cb()中进行预处理。在调试阶段可以在这里将经过的消息打印出来，查看消息是否被传递到主消息窗口，之后再一级级向下寻找消息传播路径。

2、在消息循环前在_process_init()函数中加载 app_root 应用和注册钩子函数。钩子函数的作用是从 input 输入子系统中拿消息。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
activity_load_app("application://app_root");          /* load main*/
msg_emit_init();                                     /* messge server init*/
emit_hook.key_hook            = key_hook_cb;
emit_hook.tp_hook             = tp_hook_cb;
emit_hook.init_hook               = init_hook_cb;
msg_emit_register_hook(&emit_hook);                  /*register  hook*/
```

3、启动消息接收和分发服务

```
while( GUI_GetMessageEx(&msg, init_mainwin) )        /* message loop*/
{
    ret = GUI_DispatchMessage(&msg);                 // 分发消息到回调
    if( msg.p_arg )                                  // 同步消息回应
    {
        GUI_SetSyncMsgRetVal(&msg, ret);
        GUI_PostSyncSem(&msg);
        }
    }
    GUI_WinThreadCleanup(init_mainwin);
    message_loop_win_set_default(NULL);
```

8.3. app_root 加载部分

在 applets 文件夹下的 make.cfg 有

TARGET   = $(WORKSPACEPATH)/beetles/rootfs/apps/app_root.axf

这正是 inti 部分加载的 app_rootfs.axf 文件。首先创建根管理窗口 APP_ROOT。

首先调用 app_root_wincreat() 函数创建一个管理窗口，其父窗口是 init 创建的根窗口 init_mainwin ，名字是 APP_ROOT，并且有一个重要参数 ManWindowProc（管理窗口消息处理过程），注册的回调函数为

app_root_win_proc()。父窗口 init_mainwin 发送的消息首先在这里被处理，或者将子窗口的消息发送给父窗口。窗口创建时向自身发送 GUI_MSG_CREATE 消息，进行初始化操作。

```
create_info.attr             = activity;                          /*管理窗口的私有属性*/
create_info.hParent      = activity_get_root_win(activity) ;      /*管理窗口的父窗口句柄*/
create_info.hHosting         = NULL;                             / *主窗口的拥有者窗口*/
create_info.ManWindowProc                                                               =
(__pGUI_WIN_CB)esKRNL_GetCallBack((__pCBK_t)app_root_win_proc);
create_info.name            = APP_ROOT;                        /*主窗口名称*/
hManWin                   = GUI_ManWinCreate(&create_info);       /*根据参数创建管理窗口*/
```

app_root_win_proc()函数完成消息处理任务。其中较为重要的是 GUI_MSG_CREATE 和 GUI_MSG_COMMAND 以及 GUI_MSG_KEY 三种消息。不需要处理的消息交给默认流程往下分发。

接收到 GUI_MSG_CREATE 进行应用创建。

接收到 GUI_MSG_COMMAND，处理子窗口向父窗口发送来的消息。根据 app 的 ID 进行各个 app 之间的切换。包括资源的关闭打开等。

接收到 GUI_MSG_KEY 进行按键消息处理，完成按键响应，或者直接拦截按键消息。

8.4. home 加载部分

APP_ROOT 的回调函数接收到 GUI_MSG_CREATE 消息，继续进行桌面创建。首先进行内存资源申请，创建子管理窗口 APP_HOME。创建函数为 app_home_create()。Home 管理窗口的名字是 APP_HOME，父管理窗口为 APP_ROOT，消息处理回调函数为 app_home_proc()。窗口创建时向自身发送 GUI_MSG_CREATE 消息继续进行初始化操作。

```
create_info.name              = APP_HOME;
create_info.hParent            = para->h_parent;
create_info.ManWindowProc      = (__pGUI_WIN_CB)esKRNL_GetCallBack((__pCBK_t)app_home_proc);
create_info.attr               = (void*)para;
create_info.id                  = APP_HOME_ID;
create_info.hHosting           = NULL;
```

app_home_proc()函数主要完成消息处理任务，其中较为重要的是 GUI_MSG_CREATE 和 GUI_MSG_COMMAND 以及 GUI_MSG_KEY 三种消息。不需要处理的消息交给默认流程往下分发。

接收到 GUI_MSG_COMMAND，处理子窗口向本窗口发送的消息。

接收到 GUI_MSG_KEY 进行按键消息处理，完成按键响应，或者直接拦截按键消息。

接收到 GUI_MSG_CREATE 接着完成一系列初始化操作：

```
com_set_palette_by_id(ID_HOME_PAL_BMP);     //创建调色板
gscene_bgd_set_state(BGD_STATUS_HIDE);       //background 隐藏
app_main_menu_create(home_para);              //创建主菜单图层窗口 lyrwin
check_disk(home_para);                      //检查存储设备
//app_sub_menu_create(home_para);             //建立一个子菜单
GUI_LyrWinSetSta(home_para->lyr_mmenu, GUI_LYRWIN_STA_ON);//激活图层
//GUI_LyrWinSetSta(home_para->lyr_smenu, GUI_LYRWIN_STA_ON);
gscene_hbar_set_state(HBAR_ST_SHOW);        //激活状态栏
gscene_bgd_set_bottom();                    //背景图层置底
GUI_WinSetFocusChild(home_para->h_mmenu);   //设置焦点窗口
```

其中 app_main_menu_create()创建主菜单图层窗口，申请图层，图层大小与屏幕大小一致。另外创建 framewin 窗口依赖在图层窗口之上。framewin 管理窗口为 APP_HOME，回调函数为_main_menu_Proc() 创建 framewin 窗口时向自身发送 GUI_MSG_CREATE 和 GUI_MSG_PAINT 消息，进行资源初始化和桌面绘制。

```
home_para->lyr_mmenu = home_layer_32bpp_create(rect, 1);//创建一个图层
mmenu_para.mmenu_font = home_para->main_font;//
mmenu_para.focus_id = home_para->focus_id;
mmenu_para.layer = home_para->lyr_mmenu;
home_para->h_mmenu = main_menu_win_create(home_para->h_app_main, &mmenu_para);//将图层作为
                                                              参数传递给 framewin 窗口
GUI_WinSetFocusChild(home_para->h_mmenu);//设置焦点窗口
```

_main_menu_Proc()函数主要完成消息处理任务，其中较为重要的是 GUI_MSG_CREATE 和 GUI_MSG_PAINT 以及 GUI_MSG_KEY 三种消息。不需要处理的消息交给默认流程往下分发。

接收到 GUI_MSG_PAINT，绘制桌面内容，有些应用该消息未使用，直接在 GUI_MSG_CREATE 完成绘制。

接收到 GUI_MSG_KEY，进行按键消息处理，完成按键响应。

接收到 GUI_MSG_CREATE，初始化 ui 资源，绘制桌面，激活桌面显示：

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
static void paint_mmain_item_all(mmenu_attr_t *mmenu_attr)
{
    __s32 i;
    for(i=0; i < mmenu_attr->item_nr; i++)
    {
        if (i == mmenu_attr->focus_item)
        {
            paint_mmain_item_ex(mmenu_attr, i, 1);
        }
        else
        {
            paint_mmain_item_ex(mmenu_attr, i, 0);
        }
    }
}
```

至此，应用桌面创建完成。

9. 调屏

9.1. 调屏步骤简介

9.1.1. 判断屏接口。

首先要判断屏的接口，是 HV、CPU 还是 LVDS 接口的屏，不同的接口，硬件连线及调屏方法不一样。

1. 询问客户、屏厂或则根据屏的规格书等确定。

2. 通过硬件连线方式确定。

9.1.2. 确定硬件连接。

不同的接口屏，硬件连接不一样，同一接口的屏，连接方式有几种。

1.根据“屏硬件连接”确认。

2.根据原理图确认屏背光、电源、时钟、复位及扩展 IO 连接。

9.1.3. 配置显示部分 sys_config.fex

9.1.3.1.配置屏相关 IO

```
[lcd0_para]
lcd_power_used    = 0
lcd_power         = port:PA0<1><default><default><1>
lcd_bl_en_used    = 0
;lcd_pwm_used     = 0
;lcd_pwm      = port:PE12<4><0><default><default>

;lcdd0        = port:PE0<3><default><default><default>
```

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

```
....
lcdvsync        = port:PD21<2><default><default><default>
```

9.1.4. Lcd_panel_cfg.c 初始化文件中配置屏参数

9.1.4.1.LCD_cfg_panel_info

lcd_panel_para_info(__s32 sel, disp_panel_para *info) 函数中设置相关屏参，包括像素、刷新率、前后肩、RGB 格式等。

9.1.4.2.LCD_open_flow

根据屏的开屏流程函数 LCD_open_flow(__u32 sel) 修改相关函数：

1.static void LCD_power_on(__u32 sel)使能屏，设置复位时序。

2.自定义 LCD_panel_init(__u32 sel) 函数根据屏写数据的不同接口，调用相关接口，写入初始化代码。该函数接在 LCD_power_on 函数之后即可。

比如 hv 屏可能需要 spi 或 iic 初始化，用 LP_TX 模式初始化。无需设置初始化代码的屏，不用设置。

9.2. 软件配置说明

9.2.1. 屏文件说明

lcd_panel_cfg.c 文件，定义了 TCON 的参数，开关屏的流程，还有对屏的初始化操作。

对 IO 位置的定义，包括电源控制，配屏使用的 GPIO，以及 LCD 控制器 IO 的定义在 sys_config.fex 中。

函数：LCD_cfg_panel_info

功能：配置 C200S 的 TCON 基本参数

原型：static void LCD_cfg_panel_info(__panel_para_t * info)

参数的定义见“3 TCON 参数说明”。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

函数：LCD_open_flow

功能：定义开屏的流程

原型：static __s32 LCD_open_flow(__u32 sel)

具体说明见“2.2 开关屏流程”。

函数：LCD_close_flow

功能：定义关屏的流程

原型：static __s32 LCD_close_flow(__u32 sel)

该函数与 LCD_open_flow 对应

函数：LCD_get_panel_funs_0/ LCD_get_panel_funs_1

功能：

原型：void LCD_get_panel_funs_0(__lcd_panel_fun_t * fun)

初始化注册屏的相关操作。

9.2.2. 开关屏流程

开关屏的常见操作流程如图 2-2 所示。

其中，LCD_open_flow 和 LCD_close_flow 称为开关屏流程函数，方框中的函数，如 LCD_power_on，TCON_open 等函数，称为开关屏步骤函数。

部分屏不需要写入屏初始化参数，LCD_panel_init 及 LCD_panel_exit 这两个步骤函数（图中紫色框部分）可以省去。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

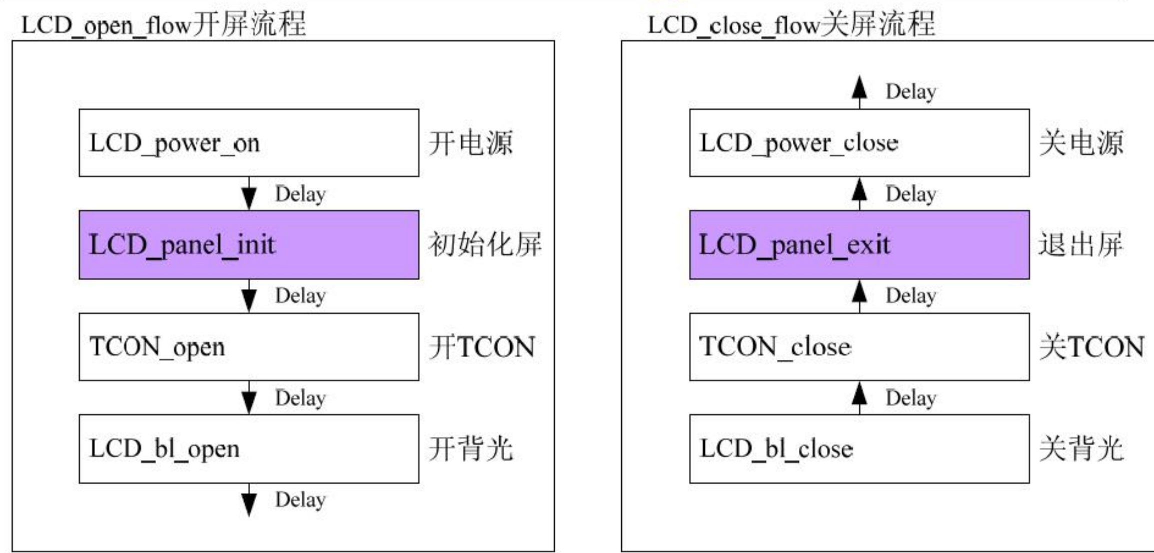


图 2-2 开关屏流程

9.2.2.1.开关屏步骤函数说明

开屏的步骤函数有 LCD_panel_init，TCON_open， LCD_power_on，LCD_bl_open。

函数：**LCD_panel_init**
功能：对屏初始化
原型：static void LCD_panel_init(__u32 sel)
可参考“2.3 对屏的初始化”。部分屏不需要进行初始化操作，LCD_panel_init 及 LCD_panel_exit 这两个步骤函数可以省去。

函数：**TCON_open**
功能：打开 F20 TCON
原型：__s32 TCON0_open(__u32 sel)
该函数由显示驱动提供，用户无需实现。

函数：**LCD_power_on**
功能：打开 LCD 电源
原型：static void LCD_power_on(__u32 sel)
显示驱动提供 LCD_POWER_EN 函数可供调用，用户也可自由实现函数内容。

函数：**LCD_bl_open**
功能：打开 LCD 背光
原型：static void LCD_bl_open(__u32 sel)
显示驱动提供 LCD_PWM_EN 和 LCD_BL_EN 函数可供调用，用户也可自由实现函数内容。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

LCD_PWM_EN，LCD_BL_EN，LCD_POWER_EN 这三个函数是通过 GPIO 控制实现电源和背光的开启关闭，IO 的位置及属性定义在 sys_config.fex 文件中。

函数：LCD_PWM_EN
功能：打开或关闭 LCD 背光调节的 PWM 信号
原型：void LCD_PWM_EN (__u32 sel, __bool b_en)
参数说明：
b_en=0：将 PWM pin 设为输入，并把 PWM 模块关闭
b_en=1：将 PWM pin 设为 PWM，并把 PWM 模块打开
对应于 sys_config.fex 文件的 lcd_pwm。

函数：LCD_BL_EN
功能：打开或关闭 LCD 背光
原型：void LCD_BL_EN (__u32 sel, __bool b_en)
参数说明：
b_en=0：设置 LCD 背光控制 IO 为对应电平，关闭背光
b_en=1：设置 LCD 背光控制 IO 为对应电平，打开背光
对应于 sys_config.fex 文件的 lcd_bl_en;

函数： LCD_POWER_EN
功能：打开或关闭 LCD 电源
原型：void LCD_POWER_EN (__u32 sel, __bool b_en)
参数说明：
b_en=0：设置 LCD 电源控制 IO 为对应电平，关闭 LCD 电源
b_en=1：设置 LCD 电源控制 IO 为对应电平，打开 LCD 电源
对应于 sys_config.fex 文件的 lcd_power。

关屏的步骤函数与开屏的步骤函数相对应。

9.2.2.2.开关屏流程函数说明

函数：LCD_open_flow
功能：初始化开关屏的步骤流程
原型：static __s32 LCD_open_flow(__u32 sel)
函数常用内容为：

```
static __s32 LCD_open_flow(__u32 sel)
{
    LCD_OPEN_FUNC(sel, LCD_power_on,10);
    LCD_OPEN_FUNC(sel, LCD_panel_init, 50);
    LCD_OPEN_FUNC(sel, TCON_open, 100);
    LCD_OPEN_FUNC(sel, LCD_bl_open, 0);
    return 0;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

}
如上，初始化整个开屏的流程步骤为四个：
1、打开 LCD 电源，再延迟 10ms；
2、初始化屏，再延迟 50ms；
3、打开 C200S TCON，再延迟 200ms；
4、打开背光，再延迟 0ms。
LCD_open_flow 函数只会系统初始化的时候调用一次，执行每个 LCD_OPEN_FUNC 即是把对应的开屏步骤函数进行注册，并没有执行该开屏步骤函数。LCD_open_flow 函数的内容必须统一用 LCD_OPEN_FUNC(sel, function, delay_time)进行函数注册的形式，确保正常注册到开屏步骤中。

函数：LCD_OPEN_FUNC
功能：注册开屏步骤函数到开屏流程中
原型：void LCD_OPEN_FUNC(__u32 sel, LCD_FUNC func, __u32 delay)
参数说明：
func 是一个函数指针，其类型是：void (*LCD_FUNC) (__u32 sel)，用户自己定义的函数必须也要用统一的形式。比如：

```
void user_defined_func(__u32 sel)
{
    //do something
}
```

delay 是执行该步骤后，再延迟的时间，时间单位是毫秒。

9.2.3. 对屏的初始化

一部分屏需要进行初始化操作，在开屏步骤函数中，对应于 LCD_panel_init 函数。在 C200S 中，提供了两种方式对屏的初始化。
一种是通过 8080 总线的方式，使用的是 LCDIO（PD,PH）。这种初始化方式，用于 CPU 屏中，其总线的引脚位置定义与 CPU 屏一致。
一种是 SPI 或 IIC 等串行的方式，使用的是 C200S 的 GPIO 引脚模拟实现。模拟 GPIO 的引脚位置定义见于 sys_config.fex 中。

9.2.3.1.IO 模拟串行接口初始化

IO 模拟串行接口初始化可以参考附录中的实例。
IO 的位置（PIN 脚）定义，默认属性（输入输出）定义及默认输出值在 sys_config.fex，具体请参考 2.5.2。
显示驱动提供 3 个接口函数可供使用。说明如下：

函数：LCD_GPIO_read
功能：读取 LCD_GPIO PIN 脚上的电平

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

原型：__s32 LCD_GPIO_read(__u32 sel,__u32 io_index);
参数说明：
io_index = 0：对应于 sys_config.fex 中的 lcd_gpio_0
io_index = 1：对应于 sys_config.fex 中的 lcd_gpio_1
io_index = 2：对应于 sys_config.fex 中的 lcd_gpio_2
io_index = 3：对应于 sys_config.fex 中的 lcd_gpio_3
函数返回值为对应 IO 的输入电平，只用于该 GPIO 定义为输入的情形。

函数：**LCD_GPIO_write**
功能：LCD_GPIO PIN 脚上输出高电平或低电平
原型：__s32 LCD_GPIO_write(__u32 sel,__u32 io_index, __u32 data);
参数说明：
io_index = 0：对应于 sys_config.fex 中的 lcd_gpio_0
io_index = 1：对应于 sys_config.fex 中的 lcd_gpio_1
io_index = 2：对应于 sys_config.fex 中的 lcd_gpio_2
io_index = 3：对应于 sys_config.fex 中的 lcd_gpio_3
data = 0：对应 IO 输出低电平
data = 1：对应 IO 输出高电平
只用于该 GPIO 定义为输出的情形。

函数：**LCD_GPIO_set_attr**
功能：设置 LCD_GPIO PIN 脚为输入或输出模式
原型：__s32 LCD_GPIO_set_attr(__u32 sel,__u32 io_index, __bool b_output);
参数说明：
io_index = 0：对应于 sys_config.fex 中的 lcd_gpio_0
io_index = 1：对应于 sys_config.fex 中的 lcd_gpio_1
io_index = 2：对应于 sys_config.fex 中的 lcd_gpio_2
io_index = 3：对应于 sys_config.fex 中的 lcd_gpio_3
b_output = 0：对应 IO 设置为输入
b_output = 1：对应 IO 设置为输出

9.2.3.2.CPU 屏 8080 总线初始化

CPU 屏的初始化可以参考“附录”的实例。
显示驱动提供 5 个接口函数可供使用。如下：

函数：**LCD_CPU_WR**
功能：设定 CPU 屏的指定寄存器为指定的值
原型：void LCD_CPU_WR(__u32 sel, __u32 index, __u32 data)
函数内容为
void LCD_CPU_WR(__u32 sel, __u32 index, __u32 data)
{

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
LCD_CPU_WR_INDEX(sel, index);
LCD_CPU_WR_DATA(sel, data);
}
```

实现了 8080 总线上的两个写操作。

LCD_CPU_WR_INDEX 实现第一个写操作，这时 PIN 脚 RS（A1）为低电平，总线数据上的数据内容为参数 index 的值。

LCD_CPU_WR_DATA 实现第二个写操作，这时 PIN 脚 RS（A1）为高电平，总线数据上的数据内容为参数 data 的值。

函数：**LCD_CPU_WR_INDEX**
功能：设定 CPU 屏为指定寄存器
原型：void LCD_CPU_WR(__u32 sel, __u32 index, __u32 data)
void LCD_CPU_WR_INDEX(__u32 sel,__u32 index);
具体说明见 LCD_CPU_WR。

函数：**LCD_CPU_WR_DATA**
功能：设定 CPU 屏寄存器的值为指定的值
原型：void LCD_CPU_WR_DATA(__u32 sel, __u32 data);
具体说明见 LCD_CPU_WR。

函数：**LCD_CPU_AUTO_FLUSH**
功能：开启 CPU 屏的刷新
原型：void LCD_CPU_AUTO_FLUSH(__u32 sel, __bool en);
参数说明：
en = 1：8080 总线上开始传送显示 BUFFER 的数据，实现 CPU 屏的刷新

函数：**LCD_cpu_register_irq**
功能：设置 LCD_GPIO PIN 脚为输入或输出模式
原型：void LCD_CPU_register_irq(__u32 sel, void (*Lcd_cpuisr_proc) (void))
注册 cpu 屏的中断处理函数，驱动会在每个 vblanking 中断里调用一下用户注册的中断处理函数 Lcd_cpuisr_proc。

CPU 屏的初始化对应于开屏步骤函数的 LCD_panel_init。在 CPU 屏 LCD_panel_init 函数的最后，需要进行两个操作步骤：

1、使用 LCD_CPU_register_irq 注册 CPU 屏的中断处理函数 Lcd_cpuisr_proc，该函数的内容，可以是 CPU 屏 GRAM 的 X 和 Y 坐标设置为零的操作，以保证异步屏每帧进行一次同步。

2、调用 LCD_CPU_AUTO_FLUSH(sel,1)打开显示数据传送。

示例如下：

```
static void LCD_panel_init(__u32 sel)
{
    kgm281i0_init(sel); //initial lcd panel
    kgm281i0_write_gram_origin(sel); //set gram origin
    LCD_CPU_register_irq(sel,Lcd_cpuisr_proc); //resgister cpu irq func
    LCD_CPU_AUTO_FLUSH(sel,1); //start sent gram data
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
}
```

区别于模拟串行接口的初始化，LCD_open_flow 中，CPU 屏的初始化 LCD_panel_init 放在 TCON_open 之后，示例如下：

```
static __s32 LCD_open_flow(__u32 sel)
{
    LCD_OPEN_FUNC(sel, LCD_power_on,10);
    LCD_OPEN_FUNC(sel, TCON_open, 100);
    LCD_OPEN_FUNC(sel, LCD_panel_init, 50);
    LCD_OPEN_FUNC(sel, LCD_bl_open, 0);
    return 0;
}
```

9.2.4. 其它函数

9.2.4.1.GPIO 操作函数

用户有可能有需要自己对某些 GPIO 进行操作,显示驱动封装了几个函数提供给用户,它们屏避了操作系统间的差异,也就是说在不同的操作系统中都可以使用。

函数：**OSAL_GPIO_Request**
功能：申请 GPIO;
原型：__hdle OSAL_GPIO_Request(user_gpio_set_t *gpio_list, __u32 group_count_max);
参数说明：
gpio_list 为 GPIO 的设置,该结构体如下:

```
typedef struct
{
    char gpio_name[32];
    int port;
    int port_num;
    int mul_sel;
    int pull;
    int drv_level;
    int data;
}user_gpio_set_t;
```

group_count_max: 要设置 GPIO 的个数.
函数返回: 成功返回 GIPO 的句柄, 失败返回 0.

函数：**OSAL_GPIO_Release**
功能：释放 GPIO.
原型：__s32 OSAL_GPIO_Release(__hdle p_handler, __s32 if_release_to_default_status);
参数说明：
p_handler: GPIO 的句柄.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

if_release_to_default_status: 0/1: 表示释放后的GPIO处于输入状态;2: 表示释放后的GPIO状态不变.
函数返回：成功返回 0, 失败返回错误号
将 GIPO PH6 输出高电平，示例如下：

```
static void LCD_vcc_on(__u32 sel)
{
    user_gpio_set_t gpio_list;
    int hdl;
    gpio_list.port = 8;// 1:A; 2:B; 3:C; 4:D;5:E;6:F;7:G;8:H.....
    gpio_list.port_num = 6;
    gpio_list.mul_sel = 1;
    gpio_list.pull = 0;
    gpio_list.drv_level = 0;
    gpio_list.data = 1;
    hdl = OSAL_GPIO_Request(&gpio_list, 1);
    OSAL_GPIO_Release(hdl, 2);
};
```

9.2.4.2.延时函数

驱动提供了毫秒和微秒级的延时给用户使用，不过建议如果延时时间比较长的话可以在开关屏流程里新添新的函数. 因为在 boot 系统里延时是死等的,效率会比较低;如果放在开关屏流程里的话会启用 timmer 去做延时,在延时期间 CPU 可以做其它的工作.

函数：**LCD_delay_ms**
功能：延时 ms 毫秒
原型：void LCD_delay_ms(__u32 ms)

函数：**LCD_delay_us**
功能：延时 us 微秒
原型：void LCD_delay_us(__u32 us)

9.3. TCON 参数说明

9.3.1. 接口参数说明

9.3.1.1.lcd_if

设置相应值的对应含义为：

0：HV(RGB 同步屏)接口
1：CPU(8080)接口
2：TTL
3：LVDS 接口

9.3.1.2.lcd_hv_if

Lcd HV panel Interface
这个参数只有在 lcd_if=0 时才有效。定义 RGB 同步屏下的几种接口类型。
设置相应值的对应含义为：
0：Parallel RGB
1：Serial RGB/ Serial YUV
该参数结合 3.1.3 的 lcd_hv_smode 定义了屏的接口类型。

9.3.1.3.lcd_hv_smode

Lcd HV panel Serial Mode
这个参数只有在 lcd_if=0 且 lcd_hv_if=1 时才有效。定义 RGB 同步串行接口屏的类型。
设置相应值的对应含义为：
0：Serial RGB
1：Serial YUV（CCIR656）
RGB 同步屏的接口类型可参考“附录 5.1.1 HV RGB 同步屏接口”。

9.3.1.4.lcd_hv_s888_if

Lcd HV panel Serial RGB output Interface
这个参数只有在 lcd_if=0 且 lcd_hv_if=1 且 lcd_hv_smode=0（Serial RGB）时才有效。
(lcd_hv_s888_if & 0xC)>>2 得到的值，定义了奇数行 RGB 输出的顺序
0: R→G→B
1: B→R→G
2: G→B→R
(lcd_hv_s888_if & 0x3)得到的值，定义了偶数行 RGB 输出的顺序
0: R→G→B
1: B→R→G
2: G→B→R

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

9.3.1.5.lcd_hv_syuv_if

Lcd HV panel Serial YUV output Interface

这个参数只有在 lcd_if=0 且 lcd_hv_if=2 且 lcd_hv_smode=1（Serial YUV）时才有效。

(lcd_hv_syuv_if & 0xC)>>2 得到的值，定义了 YUV 输出格式

0：YUYV

1：YVYU

2：UYVY

3：VYUY

(lcd_hv_syuv_if & 0x3) 得到的值，定义 CCIR656 编码时 F 相对有效行延迟的行数

0：F toggle right after active video line

1：Delay 2 lines (CCIR NTSC)

2：Delay 3 lines (CCIR PAL)

9.3.1.6.lcd_cpu_if

Lcd CPU panel Interface

这个参数只有在 lcd_if=1 时才有效。

设置相应值的对应含义为：

0：18bit×1cycle parallel (RGB666)

4：16bit×1cycle parallel (RGB565)

5：9bit×2cycle serial (RGB666)

7：8bit×2cycle serial (RGB565)

<table>
<tr><th colspan="2">Interface</th><th>Parameter</th></tr>
<tr><td rowspan="3">同步 RGB 接口
(lcd_if=0)</td><td>Parallel RGB
(lcd_hv_if=0)</td><td></td></tr>
<tr><td>Serial RGB
(lcd_hv_if=1 lcd_hv_smode=0)</td><td>lcd_hv_s888_if</td></tr>
<tr><td>CCIR656
(lcd_hv_if=1 lcd_hv_smode=1)</td><td>lcd_hv_syuv_if</td></tr>
<tr><td rowspan="3">CPU/8080 接口
(lcd_if=1)</td><td>18bit x 1cycle parallel (RGB666)
(lcd_cpu_if=0)</td><td></td></tr>
<tr><td>16bit x 1cycle parallel (RGB565)
(lcd_cpu_if=4)</td><td></td></tr>
<tr><td>9bit x 2cycle serial (RGB666)
(lcd_cpu_if=5)</td><td></td></tr>
</table>

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

	8bit x 2cycle serial (RGB565) (lcd_cpu_if=7)	
LVDS 接口(lcd_if=3)	Single Link	

9.3.1.7.lcd_lvds_bitwidth

Lcd LVDS panel Bit Width
设置相应值对应含义为：
0：24bit
1：18bit
相关说明可参见“附录 5.1.3 LVDS 屏接口”

9.3.1.8.lcd_lvds_mode

Lcd LVDS Mode
这个参数只有在 lcd_lvds_bitwidth=0 时才有效
设置相应值对应含义为：
0：NS mode
1：JEIDA mode
NS mode 和 JEIDA mode 的说明可参见“附录 5.1.3 LVDS 屏接口”。

9.3.2. 时序参数说明

9.3.2.1.lcd_x

显示屏宽的像素个数

9.3.2.2.lcd_y

显示屏高的像素个数

9.3.2.3.lcd_ht

Horizontal Total time
指一行总的 dclk 的 cycle 个数。见图 3-1。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

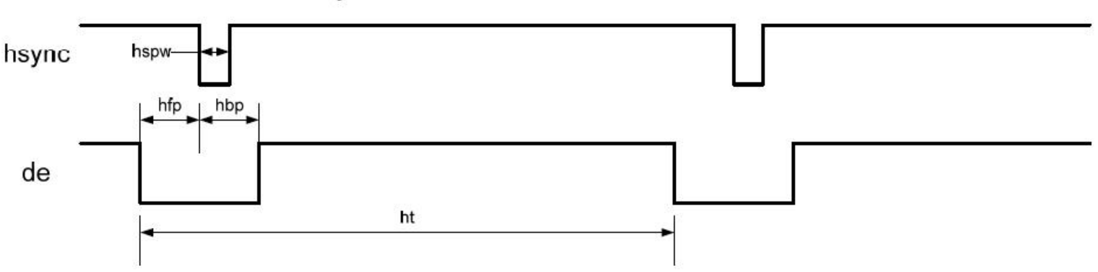


图 3-1 水平方向时序信号图

9.3.2.4.lcd_hbp

Horizontal Back Porch
指有效行间，行同步信号（hsync）开始，到有效数据开始之间的 dclk 的 cycle 个数。见图 3-1。

9.3.2.5.lcd_vt

Vertical Total time
指两场的总行数。见图 3-2。

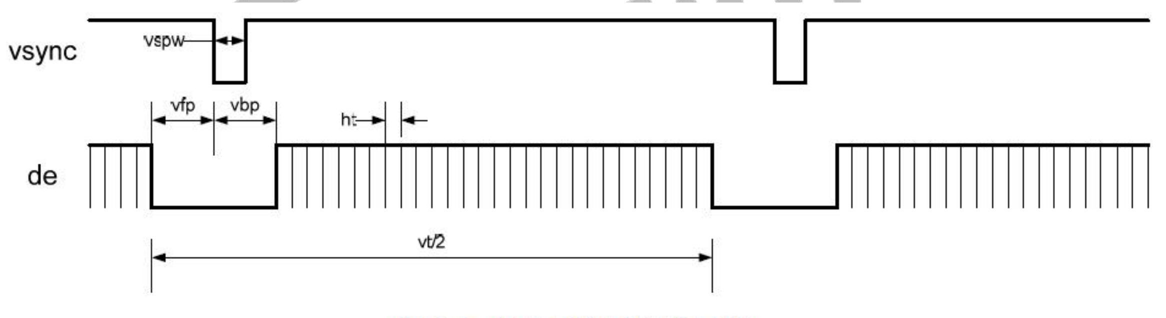


图 3-2 垂直方向时序信号图

9.3.2.6.lcd_vbp

Vertical Back Porch
指场同步信号（vsync）开始，到有效数据行开始之间的行数。见图 3-2。

9.3.2.7.lcd_hv_hspw

Horizontal Sync Pulse Width
指行同步信号的宽度。单位为 1 个 dclk 的时间（即是 1 个 data cycle 的时间）。
见图 3-1。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

9.3.2.8.lcd_hv_vspw

Vertical Sync Pulse Width
指场同步信号的宽度。单位为行。见图 3-2

9.3.2.9.lcd_dclk_freq

Data Clock Frequency
指 PIN 总线上数据的传送频率。单位为 MHz。
屏幕刷新帧数 = (dclk_freq) / (ht×vt/2)

9.3.2.10. lcd_io_cfg0

Lcd IO Configuration0
这个参数提供 RGB 同步屏的相位调节。

lcd_dclk_freq < 40 时，该参数可设置为 0x00000000，0x04000000，0x10000000，0x14000000，0x20000000，0x24000000，对应 LCD DCLK 的六个不同相位。

lcd_dclk_freq > 40 时，该参数可设置为 0x00000000，0x04000000 对应 LCD DCLK 的两个不同相位。

补充说明 1：hbp 在部分屏规格书的定义里并不包括 hspw。这种情况下，要正确配置 AW 的 TCON，hbp(aw)=hbp(panel)+hspw(panel)。vbp 的定义同 hbp。

补充说明 2：F20 的 TCON 中，图 3-1 中的 hfp，图 3-2 中的 vfp 不能为 0。

9.3.3. 其他参数说明

9.3.3.1.lcd_pwm_freq

Lcd backlight PWM Frequency
这个参数配置 lcd_pwm 信号的频率，单位为 KHz。 F20 中可以输出的 PWM 频率为 1KHz-100KHz。

9.3.3.2.lcd_gamma_correction_en

Lcd Gamma Correction Enable
设置相应值的对应含义为：
0：TCON 的 Gamma 校正关闭
1：TCON 的 Gamma 校正打开
设置为 1 时，需要对 lcd_gamma_tbl [256]进行赋值。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

9.3.3.3.lcd_gamma_tbl

Lcd Gamma Table

该参数为一个数组__u32 lcd_gamma_tbl[256];

lcd_gamma_tbl[n] = rout<<16 | gout<<8 | bout<<0 表示：输入 r=n 时，输出 r=rout；输入 g=n 时，输出 g=gout；输入 b=n 时，输出 b=bout。

用户使用 Gamma 校正功能时，可以使用函数 lcd_gamma_gen(__panel_para_t * info)对其赋值，函数内容可自由实现。具体可参考附录中的实例。

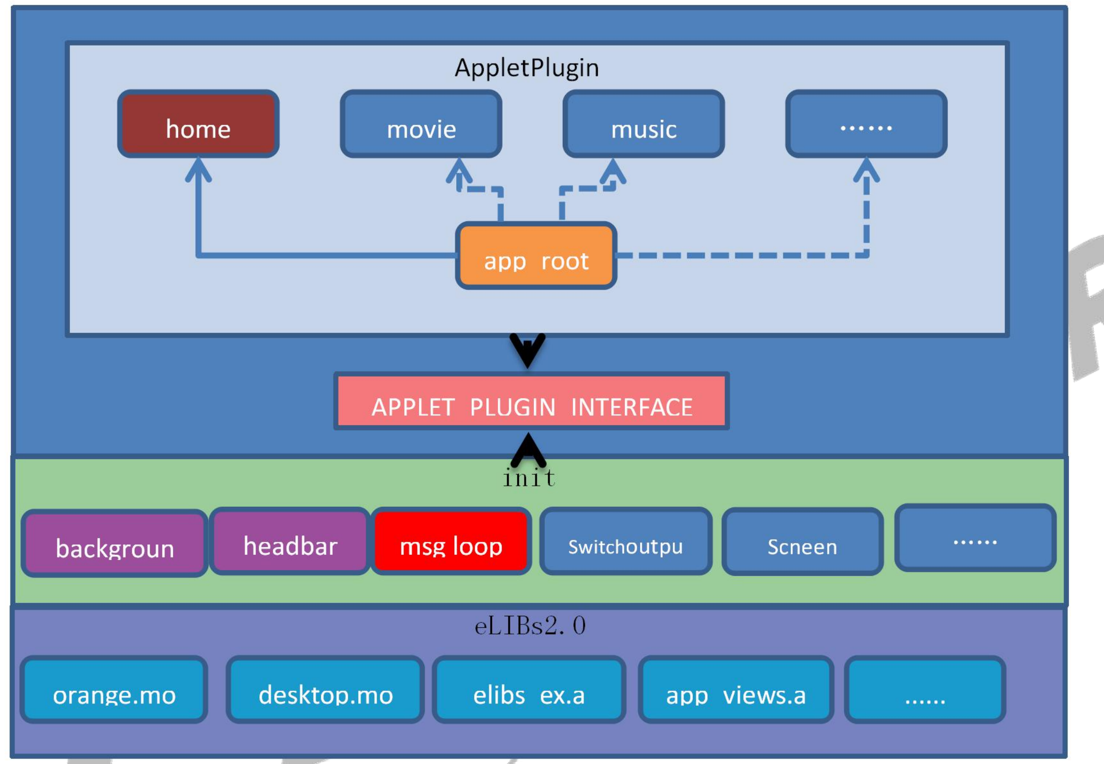
9.4.屏文件实例

路径：\eMod\drv_display\lcd\lcd_bak

10. 应用程序开发

10.1. APP framework 体系结构

10.1.1. APP framework 简介



每个应用程序(AppletPlugin)都是一个插件，实现 APPLET_PLUGIN_INTERFACE（应用程序插件接口），再将应用程序注册到系统中就可以被系统管理。新的方案中只有一个主应用 app_root，再由主应用启动子应用(home、music、movie)以实现多应用切换。主应用 app_root 已经由系统创建，并默认启动子应用 home。

Desktop 是新方案主框架，以动态库形式存在，提供包括应用插件管理、消息循环、系统事件处理、系统服务等功能接口。用户应用可以调用 Desktop 提供的功能接口以使用系统功能。框架实现由 init 模块完成，Desktop 不进行实际操作。

Init 模块调用 Desktop 提供的接口实现应用插件管理、消息循环、系统事件处理，另外提供输出切换，关屏设置的内部实现、背景管理、headbar 管理、系统对话框管理等接口供用户应用使用。

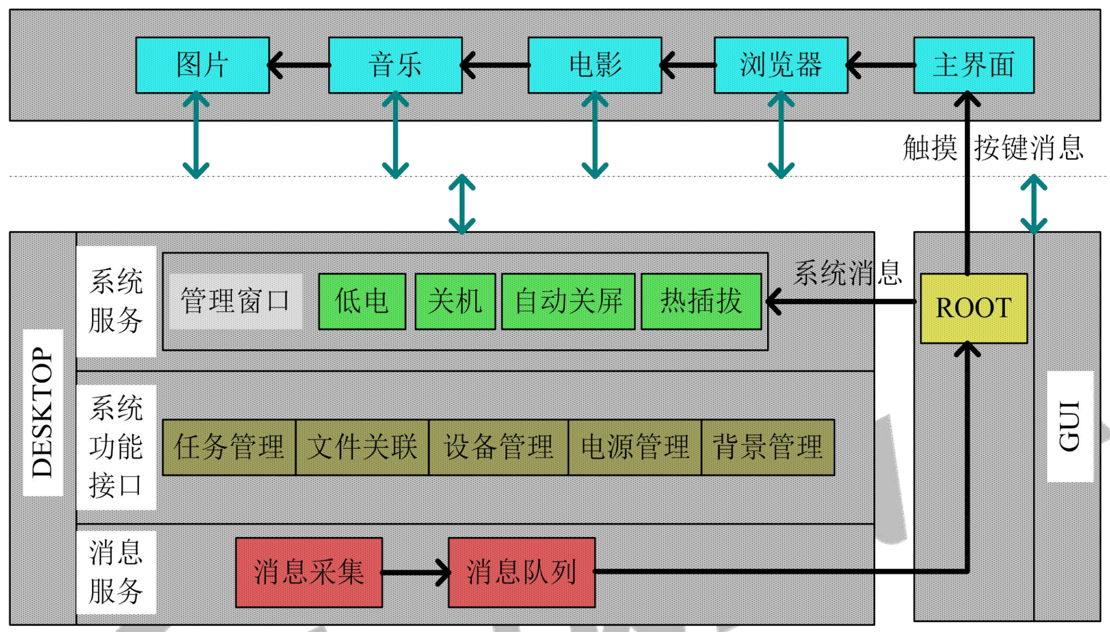
Orange 是 GUI 主体，提供完整的 GUI 绘制方法、图层管理、消息处理和基本控件管理，并封装一系列 API 供用户使用。用户应用可以调用 GUI 绘制方法，添加控件以显示图形界面，再处理 GUI 消息实现交互。

Melis 还提供其他库文件，如 elibs_ex.a 图片功能库 anole、视频音频功能库 robin、文件搜索和播放列表库 rat 等，app_views.a 列表库供应用程序使用。

10.2. DESKTOP

10.2.1. desktop 简介

桌面系统的层次结构如下图：



内核启动后，Shell 开始运行。Shell 执行 startup.esh 脚本，开始初始化 GUI 和 Desktop。Desktop 负责初始化系统资源，包括文字、图片、声音、字体、字库资源，headbar、background、dialog 组件等，为应用提供 API 以使用系统功能。初始化消息循环服务，最后启动应用。

Desktop 系统服务由 init 主管理窗口完成，在窗口回调函数 init_mainwin_cb()中完成具体服务。回调中集中处理了系统消息，用户可以在此回调中处理更多消息。

消息 ID	消息描述
DSK_MSG_POWER_OFF	关机
DSK_MSG_LOW_POWER	低电
DSK_MSG_USBD_PLUG_IN	usb device 插入
DSK_MSG_USBD_PLUG_OUT	usb device 拔出
DSK_MSG_SCREEN_OPEN	开屏
DSK_MSG_SET_SCN_CLOSE	关屏
DSK_MSG_FS_PART_PLUGIN	添加分区，表明 sd 卡插入或 host 加载
DSK_MSG_FS_PART_PLUGOUT	删除分区，表明 sd 卡拔出或 host 卸载
DSK_MSG_HDMI_PLUGIN	hdmi 插入
DSK_MSG_HDMI_PLUGOUT	hdmi 拔出
DSK_MSG_TVDAC_PLUGIN	tvdac 插入
DSK_MSG_TVDAC_PLUGOUT	tvdac 拔出
DSK_MSG_STANDBY	系统进入 standby
DSK_MSG_STANDBY_WAKE_UP	系统从 standby 恢复

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

DSK_MSG_FW_UPDATE	固件升级

Desktop 模块中系统控制接口包含在 functions 文件夹下，是应用程序经常使用的功能集合。主要包含如下部分：

- 显示驱动相关接口：dsk_display.h；
- 固件升级接口：dsk_fw_update.h；
- 按键音接口：dsk_keytone.h；
- 音量接口：dsk_voice.h；
- 音频输出设备接口：dsk_audio_if.h；
- 媒体库接口：dsk_orchid.h；
- 电源管理相关接口：dsk_power.h；
- usb device 相关接口：dsk_usbd.h；
- usb host 相关接口： dsk_usbh.h；
- 字符集设置接口：dsk_charset.h；
- 语言资源操作接口：dsk_langres.h；
- 图片资源操作接口：dsk_theme.h；

Desktop 消息服务在 init 服务线程 application_init_process()中完成，主要采集按键和触摸消息，并将消息发送到 GUI。按键和触摸消息首先会发送到输入子系统，由输入子系统对消息进行封装，再发送到 GUI。用户不需要关心消息服务具体实现，在应用程序中响应系统消息即可。消息机制介绍详见第 4 章。

10.3. 10.3.GUI 窗口体系和消息机制

Melis 应用程序由窗口组成，所有用户的操作（触摸、按键）都通过消息发送到窗口，在窗口的消息处理函数中进行处理。窗口体系和消息传递都在 GUI 实现。

10.3.1. 窗口类型

➢ Manage window(管理窗口)

虚拟窗口，负责消息分发和处理，作为应用程序的入口主窗口，有自己的消息队列。

➢ Layer window(图层窗口)

图层窗口对应的一个图层，是屏幕管理的基本单位，屏幕中的一个实体窗口，有自己的显示区域和 Z 序，也是 frame window 和控件窗口的载体。

➢ Frame window

寄生在 framebuffer 上，实体窗口，有自己的矩形区域，可以再上面写字，画图。

➢ Ctrl window(控件窗口)

封装好的具有特定属性和操作的特殊窗口，如 button、slider 等。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

10.3.2. 消息机制

- 外部事件消息

  主要指由外设触发而由输入子系统传递给 GUI 消息接收器的消息，如触摸、按键消息。

- GUI 系统消息

  主要指由外部事件或相应的函数触发的消息，如绘制、大小改变、设焦等相关的消息。

- 消息优先级

根据消息的优先级的不同又分为同步消息，异步消息（通知消息）。对同步消息而言，发送线程需要等消息执行完成之后再返回，同步消息往往用来处理高优先级的消息。异步消息往往用于非高优先级的消息，对异步消息而言，发送线程只需要将消息投递到指定的消息队列中之后然后返回，并不等消息完全执行完毕。

需要注意的是，区分同步消息和异步消息并不在消息本身，而在选择发送消息的方式，GUI_SendMessage()发送同步消息，GUI_SendNotifyMessage()发送异步消息。

- 消息路由

创建 Manage window 和 Frame window 时需要指定他们的窗口回调函数，即消息处理函数，所有消息的响应均在这个回调函数里实现。

通过 GUI_SetActiveManWin()可以指定哪个 Root 下面第一级管理窗口为 Active window，触摸消息会根据触摸区域选定焦点 Frame window，触摸消息通过 Active window 直接传递给焦点 Frame window；通过 GUI_WinSetFocusChild 可以指定按键消息焦点窗口(Manage window 或 Frame window)，按键消息会从 Active window 逐级传递到焦点窗口。

10.4. 资源使用

10.4.1. 应用字符串资源

字符串资源是打包生成的 lang.bin 文件，包含应用中使用到的所有特定字符串，例如主界面上应用名称“音乐”、“电影”等文字，不包括电影字幕，歌曲歌词。系统调用 bin 文件目的是方便管理和多国语言切换。通过资源 ID 调用对应字符串，调用 Destop 提供的 dsk_langres_set_type()即可切换多国语言。

10.4.1.1. 制作字符串资源

在路径 ROOT\livedesk\beetles\res 中 lang 文件夹存放字符串资源。文件夹中分别存放各个应用使用的字符串资源。下面以 home 主界面为例，介绍字符串资源制作方法。

1、准备字符串资源：在 ROOT\livedesk\beetles\res\lang\下面新建文件夹 home，在 home 下面新建一个 excel 命名为 home.xls，将要添加的字符串按照下面的格式填写到 excel 文件中，包括 ID、中文、繁体中文、英文。将表格另存为 Unicode 文本（home.txt）保存在 home 文件夹下。如果要使用更多语言，则多添加对应列数字符串集，并修改 lang.xml 文件。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

name	ChineseS	ChineseT	English
HOME_INSERT_DISK	请插入磁盘	請插入磁片	Pleaeinsertdisk
HOME_TV	电视	電視	TV
HOME_FM	保留应用	保留應用	reserved
HOME_MOVIE	电影	電影	movie
HOME_PHOTO	相片	相片	photo
HOME_MUSIC	音乐	音樂	music
HOME_OTHERS	附件	附件	others
HOME_SETTING	设置	設置	setting
HOME_TV_PLAY	电视收看	電視收看	play
HOME_CHANNEL_SEARCH	频道搜索	頻道搜索	search
HOME_AV_INPUT	AV 输入	AV 輸入	AVinput

2、修改 lang.xml：打开 ROOT\livedesk\beetles\res\lang\lang.xml。

修改<Groups count="3">并在其后一个行添加

<OSDItem name="home" startid="1001" maxid="2000" file="home\home.txt"/>

其中<Groups count="3">表示应用个数为 3，startid 和 maxid 表示生成资源 ID 可用范围。

其中<LangType count="3">表示有 3 种语言。

```
<OSDTable>
<Groups count="3">
<OSDItem name="home" startid="1001" maxid="2000" file="home\home.txt"/>
<OSDItem name="movie" startid="3001" maxid="4000" file="movie\movie.txt"/>
<OSDItem name="music" startid="4001" maxid="5000" file="music\music.txt"/>
</Groups>
<LangType count="3">
<LangItem name="ChineseS" langId="0x410"/>
<LangItem name="ChineseT" langId="0x420"/>
<LangItem name="English" langId="0x400"/>
</LangType>
<Output bin=".\lang.bin" head=".\lang.h"/>
</OSDTable>
```

3、生成 lang.bin：打开 Cygwin，cd 到 ROOT\livedesk\beetles\res\lang，输入命令 make，回车，即可将准备好的字符串资源打包进 lang.bin，并拷贝到 ROOT\workspace\suniv\beetles\rootfs\apps\lang.bin，相应的字符串资源 ID 见 ROOT\livedesk\beetles\include\res\lang.h。

10.4.1.2. 使用字符串资源

字符串资源的使用参见下面的伪代码：

字体和编码一般在应用初始化和退出时操作，在应用中获取和显示字符串即可。

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

```
char text[128];
GUI_RECT txtrect;
GUI_FONT    *SWFONT;

//创建字体
SWFONT = GUI_SFT_CreateFont(24, "d:\\res\\fonts\\font24.sft");

//设置字符串显示区域
txtrect.x0 = 0;
txtrect.y0 = 0;
txtrect.x1 = 50;
txtrect.y1 = 24;

//通过 ID：STRING_EXPLORE 获取字符串内容，存入 text
dsk_langres_get_menu_text(STRING_EXPLORE, text, 128);
//选择显示的图层，图层 layer 的创建参考 5.1 节
GUI_LyrWinSel(layer);
//设置显示编码（MELIS 上总是 UTF8）
GUI_UC_SetEncodeUTF8();
//选择字体
GUI_SetFont(SWFONT);
//显示字符串，水平居中
GUI_DispStringInRect(text, &txtrect, GUI_TA_HCENTER|GUI_TA_VCENTER);
//释放字体
GUI_SetFont(GUI_GetDefaultFont());
GUI_SFT_ReleaseFont(SWFONT);
```

10.4.2. 应用图片资源

10.4.2.1. 制作应用图片资源

应用图片资源指的是应用中用到的特定图片，例如主界面的应用图标、headbar 充电图标等，不包括歌曲专辑图等。通过图片 ID 即可访问图片资源。下面以 home 主界面为例，介绍图片资源制作方法。

1、准备图片：在 ROOT\livedesk\beetles\res\theme\下面新建文件夹 home，拷贝所有图片到 home 文件夹中。图片资源只能使用 bmp 图片，其他图片文件格式请使用图片转换工具进行转换。系统提供了 png 批量转 bmp 工具：eStudio\Softwares\PngTrans\ PngTrans.exe。转换完成后可根据需要压缩 bmp 图片，系统提供了 bmp 压缩工具\tool\bmp_pack_compress.bat

2、修改 makefile：编辑 ROOT\livedesk\beetles\res\theme\makefile。在# make scripts for every application 和# create theme for the whole case 和# 删除生成的中间文件 三个位置中添加 home 应用信息，其中# make scripts for every application

```
# make scripts for every application
```

```
	$(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./home	1001	2000
	$(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./movie	2001	3000
	$(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./music	4001	5000
# create theme for the whole case
	-rm touchtheme.script
	cat ./config/touchthemehdr.script	\
		./home/home.script	\
		./movie/movie.script	\
		./music/music.script	\
		>> touchtheme.script
#  删除生成的中间文件
clean:
	-rm ./theme.bin
	-rm ./touchtheme.face
	-rm ./theme.h
	-rm ./touchtheme.script
	-rm ./home/home.script
```

3、生成 theme.bin。打开 Cygwin，cd 到 ROOT\livedesk\demo\res\theme，输入命令 make，回车，即可将准备好的图片资源打包进 theme.bin，并拷贝到 ROOT\workspace\suniv\beetles\rootfs\apps\theme.bin，相应的图片资源 ID 见 ROOT\livedesk\beetles\include\res\theme.h。

10.4.2.2. 使用应用图片资源

应用图片资源的使用参见下面的伪代码：

```
HTHEME h_bmp;
void * bmp_buf;

//打开图片资源，获取资源句柄
h_bmp = dsk_theme_open(ID_DESKTOP_CURSOR_8BPP_BMP);
//通过资源句柄获取资源 buffer
bmp_buf = dsk_theme_hdl2buf(h_bmp);
//选择贴图的图层
GUI_LyrWinSel(layer);
//绘制贴图
GUI_BMP_Draw(bmp_buf, 0, 0);
//不需要使用资源时，关闭资源句柄，释放资源（通常是在应用程序退出时）
dsk_theme_close(h_bmp);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

10.4.3. 背景图资源

背景图单独存放于 ROOT\workspace\suniv\beetles\rootfs\apps，命名为 bg_default0.bgd。支持多张背景图并切换。多张背景图命名形如 bg_default1.bgd、bg_default2.bgd，以此类推。

背景图制作方法：

1、背景图仅支持 bmp 格式，如果是其他格式，请先使用图片转换工具转换为 bmp。

2、使用工具 eStudio\Softwares\Convert2YUV\ConvertYUV.exe 将 bmp 转换为.bin 文件，

3、重新将文件命名为.bmp 后缀，使用 bmp 压缩工具将文件压缩。

4、将压缩图片重命名为 bg_default0.bgd，背景制作完成。

5、最后拷贝 bg_default0.bgd 文件至 ROOT\workspace\suniv\beetles\rootfs\apps 即可。

10.4.4. 启动图资源

启动图片存放于 ROOT\workspace\suniv\beetles\rootfs\res\boot_ui，命名为 logo.bmp，在系统启动期间显示，应用启动前关闭启动图片显示。

启动图片仅支持 bmp 格式，使用 bmp 压缩工具压缩后存放至指定路径即可。

使用：在系统配置 sys_config.fex 文件中(路径：ROOT\workspace\suniv\eFex)找到配置：

```
[kernel_logo_para] ;启动图片配置
logo_enable = 1     ;1:启用启动图片  0：关闭启动图片
```

10.4.5. 按键音资源

按键音资源存放于 ROOT\workspace\suniv\beetles\rootfs\res\sounds，命名为 chord.wav，在按下按键或触摸按下时播放对应音效。

使用：Desktop 已经提供按键音操作接口，详见 dsk_keytone.c。系统启动时已经初始化按键音，应用中可直接使用接口播放按键音。

```
dsk_keytone_on();    //播放一次按键音
```

10.4.6. 系统功能资源

10.4.6.1. headbar 标题栏

headbar 标题栏用于显示应用程序信息，如应用名称，显示系统信息如音量、时间、电量、屏幕亮度等，还可以显示用户自定义信息。除了显示信息功能外，headbar 也可以响应触摸或按键消息，这需要在 headbar 的回调函数 cb_shbar_mwin()和 cb_headbar_framewin()中加入触摸或按键消息处理部分。用户也可以修改 headbar 布局，实现自定义 headbar 显示。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

headbar 默认显示信息有标题、亮度、音量、电量。桌面创建时已经初始化 headbar 相关资源，应用程序可以直接使用 Desktop 提供的 headbar 操作接口进行操作：

全局场景 headbar 的相关接口声明在 gscene_headbar.h 中：

```
typedef enum _HBarState
{
    HBAR_ST_SHOW,
    HBAR_ST_HIDE
}HBarState;

typedef enum tag_HBAR_FORMAT
{
    HBAR_FOARMAT_8BPP,
    HBAR_FOARMAT_32BPP
}__hbar_format_t;

typedef struct
{
    char            *name;          // 场景名称
    __u16           sid;            // 场景 id
    H_WIN           parent;         // 父窗口
    H_WIN           layer;          // 图层
    GUI_FONT         *font;         // 字体句柄
}HBarCreatePara;

//创建 headbar（Init 模块调用）
__s32 gscene_hbar_create(HBarCreatePara *para);

//设置 headbar 状态
__s32 gscene_hbar_set_state(HBarState state);

//获取 headbar 状态
__s32 gscene_hbar_get_state(HBarState *p_state);

//title 为 utf8 编码格式字符串， len<=32 ,设置 headbar title 区域字符串
__s32 gscene_hbar_set_title(char *title, __u32 len);

//设置 headbar 音量条
__s32 gscene_hbar_set_volume(void);

//删除 headbar（Init 模块调用）
__s32 gscene_hbar_delete(void);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

10.4.6.2. background 背景

全局场景 background 的相关接口声明在 gscene_backgrd.h 中：

```
typedef enum
{
	BGD_STATUS_SHOW,
	BGD_STATUS_HIDE
}bgd_status_t;

//背景图层初始化(init 模块调用）
__s32		gscene_bgd_init(SIZE *p_size, bgd_status_t status, __fb_type_t ftype);

//设置背景状态
void		gscene_bgd_set_state(bgd_status_t status);

//查询背景状态
bgd_status_t	gscene_bgd_get_state(void);

//背景图层优先级置顶
void		gscene_bgd_set_top(void);

//背景图层优先级置底
void		gscene_bgd_set_bottom(void);

//保存背景图片
void		gscene_bgd_save_fb(FB *fb);

//设置背景图层格式
void		gscene_bgd_set_fb_type(__fb_type_t ftype);

//刷新背景图层，从新绘制背景图片
void		gscene_bgd_refresh(void);

//恢复默认背景图
void		gscene_bgd_restore(void);

//背景图层反初始化(init 模块调用）
__s32		gscene_bgd_deinit(void);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

10.5. 应用程序编写

10.5.1. 简单应用编写

用户程序编写最基本的 6 个步骤：注册应用、创建管理窗口 manwin 并实现回调函数、创建图层、创建 framewin 并实现回调函数。下面进行介绍。

10.5.1.1. 注册应用

新方案中，只有一个主应用，子应用由主应用启动，用户应用程序为子程序。主应用默认启动 home 子应用，并由 home 子应用启动其他子应用。因此 home 子应用定义了要切换的子应用 ID。ROOT\livedesk\beetles\applets\lib\beetles_app.h 文件中定义了子应用 ID 枚举 root_home_id_t，添加项 ID_HOME_TEST_APP。

```
typedef enum
{
	ID_HOME_FM = 0,
	ID_MEDIA_START,
	ID_HOME_RECORD=ID_MEDIA_START,
	ID_HOME_MOVIE,
	ID_MEDIA_END,
	ID_HOME_SETTING = ID_MEDIA_END,
	ID_HOME_TEST_APP,
	ID_MAX_NUM,
}root_home_id_t;
```

为方便窗口、图层创建，在 ROOT\livedesk\beetles\applets\lib\beetles_app.h 加入定义，窗口或图层创建时使用此名称以进行区分。

```
#define APP_TEST			"app_test"
```

10.5.1.2. 创建管理窗口

管理窗口是所有应用的入口，负责接收消息。创建管理窗口时，id 参数请使用注册 ID 即 APP_TEST。用户也可以添加自定义结构体用于参数传递，即 create_info.attr 参数。

```
__s32 app_test_create(root_para_t *para)
{
	__gui_manwincreate_para_t create_info;
	test_app_ctrl_t *test_app_ctrl = NULL;
	__log("*****************************************************************************************\n");
	__log("**********************************	enter test app	***********************************\n");
	__log("*****************************************************************************************\n");
	test_app_ctrl = (test_app_ctrl_t)esMEMS_Balloc(sizeof(test_app_ctrl_t));
```

```
    if( test_app_ctrl == NULL )
    {
        __msg("test app esMEMS_Balloc fail\n");
        return NULL;
    }
    eLIBs_memset(test_app_ctrl, 0, sizeof(test_app_ctrl_t));
    test_app_ctrl->test_app_font = para->font;
    test_app_ctrl->root_type          = para->root_type;
    eLIBs_memset(&create_info, 0, sizeof(__gui_manwincreate_para_t));
    create_info.name                  = APP_TEST;
    create_info.hParent               = para->h_parent;
    create_info.ManWindowProc     = (__pGUI_WIN_CB)esKRNL_GetCallBack((__pCBK_t)_app_test_proc);
    create_info.attr                  = (void*)test_app_ctrl;
    create_info.id                        = APP_TEST;
    create_info.hHosting              = NULL;
    return(GUI_ManWinCreate(&create_info));
}
```

10.5.1.3. 实现管理窗口消息处理回调函数

消息处理函数为管理窗口中设置的回调函数，在发生消息传递时先被调用。主要集中处理 GUI_MSG_CREATE、GUI_MSG_DESTROY、GUI_MSG_CLOSE、GUI_MSG_KEY、GUI_MSG_TOUCH 等系统消息，也可以处理用户自定义消息。

```
static __s32 _app_test_proc(__gui_msg_t *msg)
{
    __s32 ret; switch( msg->id )
    {
    case GUI_MSG_CREATE: /* 创建子窗口*/
    layer = htouch_layer_create(); /* 创建图层*/
    GUI_LyrWinSetTop(layer); /* 图层置顶*/
    htouch_frmwin_create(msg->h_deswin, layer); /* 创建 framewin*/
    return EPDK_OK;
    /*释放在 create 中分配的资源，尽量在此回调中释放资源，而不要在退出消息循环后在释放资源*/
    case GUI_MSG_DESTROY:
    GUI_LyrWinDelete(layer);
    return EPDK_OK;
    /* 需要支持的桌面消息*/
    case DSK_MSG_HOME: /* 回到主界面 */
    case DSK_MSG_KILL: /* 强制杀掉该应用程序 */
    ret = GUI_ManWinDelete(msg->h_deswin);
    return ret;
    case GUI_MSG_CLOSE:
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
        GUI_ManWinDelete(msg->h_deswin);
        dsk_load_app("main.app"); /* 回到主界面 */
        return EPDK_OK;
        case GUI_MSG_KEY: /* 按键响应 */
        if( msg->dwAddData1 == GUI_MSG_KEY_ESCAPE )
        {
        GUI_ManWinDelete(msg->h_deswin);
        dsk_load_app("main.app");
        return EPDK_OK;
        }
        break;
        default:
        break;
        }
        return GUI_ManWinDefaultProc(msg);/*默认处理流程*/
}
```

10.5.1.4. 创建图层

此图层为应用程序显示区域，设置显示矩形区域位置和大小，设置区域格式为 ARGB 或者其他。图层成功创建完毕后将会返回图层句柄。

```
    static H_LYR test_app_32bpp_layer_create(RECT *LayerRect)
    {
        H_LYR layer = NULL;
        RECT LayerRect;
        FB   fb =
        {
            {0, 0},                                             /* size      */
            {0, 0, 0},                                          /* buffer    */
            {FB_TYPE_RGB, {PIXEL_COLOR_ARGB8888, 0, (__rgb_seq_t)0}},    /* fmt      */

        };

        __disp_layer_para_t lstlyr =
        {
            DISP_LAYER_WORK_MODE_NORMAL,                        /* mode      */
            0,                                   /* ck_mode   */
            0,                                   /* alpha_en  */
            0,                                   /* alpha_val */
            1,                                   /* pipe      */
            0xff,                                /* prio      */
            {0, 0, 0, 0},                        /* screen    */
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
            {0, 0, 0, 0},                               /* source    */
            DISP_LAYER_OUTPUT_CHN_DE_CH1,               /* channel   */
          NULL                                          /* fb        */
        };

        __layerwincreate_para_t lyrcreate_info =
        {
            "APP_TEST",
            NULL,
            GUI_LYRWIN_STA_SUSPEND,
            GUI_LYRWIN_NORMAL
    };

        fb.size.width       = LayerRect->width;
        fb.size.height          = LayerRect->height;
        fb.fmt.fmt.rgb.pixelfmt = PIXEL_COLOR_ARGB8888;

        lstlyr.src_win.x        = LayerRect->x;
        lstlyr.src_win.y        = LayerRect->y;
        lstlyr.src_win.width    = LayerRect->width;
        lstlyr.src_win.height   = LayerRect->height;

        lstlyr.scn_win.x        = LayerRect->x;
        lstlyr.scn_win.y        = LayerRect->y;
        lstlyr.scn_win.width    = LayerRect->width;
        lstlyr.scn_win.height   = LayerRect->height;

        lstlyr.pipe = 1;
        lstlyr.fb = &fb;
        lyrcreate_info.lyrpara = &lstlyr;

        layer = GUI_LyrWinCreate(&lyrcreate_info);

        if( !layer )
        {
            __err("test app layer create error !\n");
        }
        return layer;
    }
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

10.5.1.5. 创建 framewin

Framewin 需要传入创建的图层句柄，以操作图层。主要工作是绘图，如创建窗口绘图、按键或触摸改变焦点绘图等。这些工作在回调中实现。另外可以传入一些用户自定义结构体数据，即 framewin_para.attr 参数。

```
static H_WIN test_app_framewin_create(H_WIN h_parent, setting_general_para_t *para)
{
    __gui_framewincreate_para_t framewin_para;
    setting_general_para_t    *general_para;
    FB fb;

    GUI_LyrWinGetFB(para->layer, &fb);

    eLIBs_memset(&framewin_para, 0, sizeof(__gui_framewincreate_para_t));
    framewin_para.name              = "test_app win",
    framewin_para.dwExStyle         = WS_EX_NONE;
    framewin_para.dwStyle           = WS_NONE|WS_VISIBLE;
    framewin_para.spCaption         = NULL;
    framewin_para.hOwner            = NULL;
    framewin_para.id                = GENERAL_MENU_ID;
    framewin_para.hHosting          = h_parent;
    framewin_para.FrameWinPro= (__pGUI_WIN_CB)esKRNL_GetCallBack((__pCBK_t)htouch_frmwin_cb);
    framewin_para.rect.x            = 0;
    framewin_para.rect.y            = 0;
    framewin_para.rect.width        = fb.size.width;
    framewin_para.rect.height       = fb.size.height;
    framewin_para.BkColor.alpha     = 0;
    framewin_para.BkColor.red       = 0;
    framewin_para.BkColor.green     = 0;
    framewin_para.BkColor.blue      = 0;
    framewin_para.attr              = NULL
    framewin_para.hLayer            = para->layer;

    return (GUI_FrmWinCreate(&framewin_para));
}
```

10.5.1.6. 实现 framewin 消息处理回调函数

在此回调中完成图层绘制。可以在 GUI_MSG_CREATE 或 GUI_MSG_PAINT 消息中绘制应用界面，图片、文字的使用和绘制请参考第 5 章中。处理按键消息 GUI_MSG_KEY 或触摸消息 GUI_MSG_TOUCH，更新焦点等。应

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

用程序能处理父窗口发送的 GUI_MSG_COMMAND 自定义命令，并完成相应的绘制或其他响应，此外也能发送响应结果到父窗口或其他目标窗口。

```
static __s32 htouch_frmwin_cb(__gui_msg_t *msg)
{
    switch( msg->id )
    {
        case GUI_MSG_CREATE:
        {
            htoutch_frmw_ctr *ctr;
            button_para_t     *para;
            ctr = esMEMS_Malloc(0, sizeof(htoutch_frmw_ctr));
            if( !ctr )
            {
                __err(" frmwin malloc fail \n");
                return EPDK_FALSE;
            }
            eLIBs_memset(ctr, 0, sizeof(htoutch_frmw_ctr));
            para = &(ctr->para);
            ctr->focus_size         =get_res_them(&(para->focus_bmp), STYLEID,
                                     ID_HELLOTOUCH_FOCUS_PIC_BMP);
            ctr->unfocus_size       = get_res_them(&(para->unfocus_bmp), STYLEID,
                                     ID_HELLOTOUCH_UNFOCUS_PIC_BMP);
            para->bmp_pos.x         = 0;
            para->bmp_pos.y         = 0;
            htouch_static_ctl_create(msg->h_deswin, para);
            GUI_WinSetAddData( msg->h_deswin, (__u32)ctr);
            return EPDK_OK;
        }
        case GUI_MSG_DESTROY:
        {
            htoutch_frmw_ctr *ctr = (htoutch_frmw_ctr *)GUI_WinGetAddData(msg->h_deswin);
            free_res_them(ctr->para.focus_bmp,     ctr->focus_size);
            free_res_them(ctr->para.unfocus_bmp, ctr->unfocus_size);
            esMEMS_Mfree(0, ctr);
            return EPDK_OK;
        }
        case GUI_MSG_CLOSE:
        {
            GUI_FrmWinDelete(msg->h_deswin);
            return EPDK_OK;
        }
        case GUI_MSG_COMMAND:
        {
            switch(LOSWORD(msg->dwAddData1))
            {
                case ID_WIDGET_STATIC:
                {
                    switch( HISWORD(msg->dwAddData1) )
                    {
```

```
                        case BN_CLICKED:
                        {
                            __gui_msg_t msgex;
                            eLIBs_memset(&msgex, 0, sizeof(__gui_msg_t));
                            msgex.id = GUI_MSG_CLOSE;
                            msgex.h_srcwin = 0;
                            msgex.h_deswin = GUI_WinGetManWin(msg->h_deswin);
                            GUI_SendNotifyMessage(&msgex);
                            break;
                        }
                    }
                    break;
                }
            }
            return EPDK_OK;
        }
        default:
            break;
    }

    return GUI_FrmWinDefaultProc(msg);
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

11. UI 资源工具

11.1. 工具的由来

为方便开发者获取 UI 界面的图标素材，同时满足嵌入式产品较为严格的存储容量限制要求，全志科技自主开发了一套制作工具，用来把常规格式的图片文字素材制作、压缩成为 Melis 系统可以识别的二进制数据，使这些数据可以呈现为人机界面的图标、文字。

Melis2.0 系统主要涉及到的图片、文字主要分为 4 种：

1】背景图片资源；

2】图标资源；

3】语言文字编码；

4】文字点阵字库；

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

11.2. 背景图片的制作工具

Convert2YUV

保留（无法给出操作截图。目前的工作环境无法运行本 exe 程序，报 MFC042D.DLL 动态文件缺失，网上下载的该 dll 文件也无法运行，等找一台可运行的机器拷贝其中的 MFC042D.DLL 文件出来）

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

11.3. 图标制作工具

本节介绍图标的制作和转化，将 png 后缀的图片文件转换成 32 位的 bmp 后缀的图片文件，并压缩 bmp 文件的大小。

【step1】进入“SDK\Softwares\bmp_pack_compress”路径，把切好的 png 后缀的 UI 图片拷贝到本文件夹下，如下【图 1】

【step2】双击“8bbp_pack_compress.bat”脚本，弹出如下【图 2】所示的命令行窗口，输入“0”后按“Enter”键，脚本继续运行，把 png 后缀的图片文件转化成 32 位深度的 bmp 后缀文件，并显示运行结果。

如果需要批量处理较多的图片，这些图片的总容量过大，可以接着运行【step3】，把本步骤生成的 32 位 bmp 后缀文件进行压缩，以节省文件容量；

【step3】继续在命令行输入“2”后按“Enter”键，脚本继续运行，把 bmp 后缀的文件进行压缩，并在命令行显示运行结果，请参考【图 3】。

至此，就把 UI 设计师切好的图片转化成了小容量的 bmp 图标文件，在“SDK\livedesk\beetles\res\theme\”文件夹中，各子文件夹下的 bmp 后缀文件就是这样制作的，参考【图 4】。Melis 桌面应用的 C 代码会关联使用这些转化好的文件，关联的流程在此先暂不介绍，此处只侧重于介绍制作方法。

nelis › eStudio › Softwares › bmp_pack_compress　　搜索"bmp_pack_compress"

8bpp_pack_compree　1.png　2.png　3.png　4.png

5.png　6.png　8bpp_pack_compress.bat　convertBIN2BMP.bat　convertBMP2BGD.bat

图【1】

C:\WINDOWS\system32\cmd.exe

```
*********************************************************
*   选择需要的转换                                      *
*********************************************************
 0: 将png转换成32bpp
 1: 将32bpp转换成8bpp
 2: 将bmp压缩
 3: 设置32bpp为透明属性
 4: 清除32bpp为透明属性
 5: 将32bpp转换成8bpp,然后将8bpp压缩
 6: 将32bpp转换成8bpp,然后将8bpp打包
 7: 将32bpp转换成8bpp,然后将8bpp打包,最后将打包文件压缩
 8: 退出
*********************************************************
Please Select:0
1.png to BMP succeed!
2.png to BMP succeed!
3.png to BMP succeed!
4.png to BMP succeed!
5.png to BMP succeed!
6.png to BMP succeed!
```

图【2】

C:\WINDOWS\system32\cmd.exe

```
********************************************************
 0: 将png转换成32bpp
 1: 将32bpp转换成8bpp
 2: 将bmp压缩
 3: 设置32bpp为透明属性
 4: 清除32bpp为透明属性
 5: 将32bpp转换成8bpp,然后将8bpp压缩
 6: 将32bpp转换成8bpp,然后将8bpp打包
 7: 将32bpp转换成8bpp,然后将8bpp打包,最后将打包文件压缩
 8: 退出
********************************************************
Please Select:2
.\8bpp_pack_compree\az100_batch.exe
e
C:\WORK\melis\eStudio\Softwares\bmp_pack_compress\1.bmp
tmp.o
已复制         1 个文件。
.\8bpp_pack_compree\az100_batch.exe
e
C:\WORK\melis\eStudio\Softwares\bmp_pack_compress\2.bmp
tmp.o
已复制         1 个文件。
.\8bpp_pack_compree\az100_batch.exe
e
C:\WORK\melis\eStudio\Softwares\bmp_pack_compress\3.bmp
tmp.o
已复制         1 个文件。
.\8bpp_pack_compree\az100_batch.exe
e
C:\WORK\melis\eStudio\Softwares\bmp_pack_compress\4.bmp
tmp.o
已复制         1 个文件。
.\8bpp_pack_compree\az100_batch.exe
e
C:\WORK\melis\eStudio\Softwares\bmp_pack_compress\5.bmp
tmp.o
```

图【3】

livedesk › beetles › res › theme › home   搜索"home"

名称	日期	类型	大小	标记
home_png	2019/9/30 0:33	文件夹		
BGSELL.bmp	2019/6/25 19:27	BMP 文件	1 KB	
BGSELT.bmp	2019/6/25 19:27	BMP 文件	1 KB	
BGSELU_L.bmp	2019/6/25 19:27	BMP 文件	1 KB	
BGSELU_R.bmp	2019/6/25 19:27	BMP 文件	1 KB	
EBOOK_FOCUS.bmp	2019/6/25 19:27	BMP 文件	1 KB	
EBOOK_LOSE.bmp	2019/6/25 19:27	BMP 文件	1 KB	
MAIN_EBOOK_FOCUS.bmp	2019/6/25 19:27	BMP 文件	7 KB	
MAIN_EBOOK_LOSE.bmp	2019/6/25 19:27	BMP 文件	6 KB	
MAIN_FFM_LOSE.bmp	2019/6/25 19:27	BMP 文件	7 KB	
MAIN_FM_FOCUS.bmp	2019/6/25 19:27	BMP 文件	8 KB	
MAIN_MANGER_FOCUS.bmp	2019/6/25 19:27	BMP 文件	8 KB	
MAIN_MANGER_LOSE.bmp	2019/6/25 19:27	BMP 文件	7 KB	
MAIN_MOVIE_FOCUS.bmp	2019/6/25 19:27	BMP 文件	9 KB	
MAIN_MOVIE_LOSE.bmp	2019/6/25 19:27	BMP 文件	7 KB	
MAIN_MUSIC_FOCUS.bmp	2019/6/25 19:27	BMP 文件	8 KB	

图【4】

11.4. UI 图标资源包文件 theme.bin

上一节讲述了图标资源从 png 文件转换成 32 位 bmp 文件的方法，为了对整个方案的 UI 图标进行集中管理，Melis2.0 系统把所有的图片资源按照自主设计的数据格式集中整合到一个 theme.bin 文件中，在整合期间对每一个图标都分配一个唯一的编号，这些编号以宏定义的形式写到 theme.h 头文件中，以供 C 语言代码调用。如果想要获取某一个图片资源的二进制数据，可以通过宏定义的编号在 theme.bin 文件搜索。

theme.bin 和 theme.h 是在“SDK\livedesk\beetles\res\theme”路径下编译生成的，如下图【5】所示。

```
Administrator@air-win10 /cygdrive/c/WORK/melis
/livedesk/beetles/res/theme
$ make clean;make
```

图【5】

« livedesk › beetles › res › theme   搜索"theme"

名称	修改日期	类型	大小
makefile	2019/9/26 19:51	文件	5 KB
readme.txt	2019/9/26 19:51	TXT 文件	1 KB
theme.bin	2019/9/26 19:51	BIN 文件	1,108 KB
theme.h	2019/9/26 19:51	H 文件	65 KB
touchtheme.face	2019/9/26 19:51	FACE 文件	1,158 KB
touchtheme.script	2019/9/26 19:51	SCRIPT 文件	79 KB

图【6】

11.4.1. 生成 theme.h 和 theme.bin

用文本编辑软件打开“SDK\livedesk\beetles\res\theme\makefile”文件，如下图【7】所示，makefile 脚本中，符号“#”表示注释。

```
00019: ROOT    = .
00020: SDKROOT = $(ROOT)/../../../..
00021:
00022: #导入交叉编译器通用配置
00023: include $(SDKROOT)/includes/cfgs/CROSSTOOL.CFG
00024:
00025: buildtheme:
00026:     # make scripts for every application
00027: #   $(ESTUDIOROOT)/Softwares/Face200/MakeScript/MakeScript.exe ./init      1      1000
00028: #   $(ESTUDIOROOT)/Softwares/Face200/MakeScript/MakeScript.exe ./home      1001   2000
00029: #   $(ESTUDIOROOT)/Softwares/Face200/MakeScript/MakeScript.exe ./movie     2001   3000
00030: #   $(ESTUDIOROOT)/Softwares/Face200/MakeScript/MakeScript.exe ./explorer  3001   4000
00031: #   $(ESTUDIOROOT)/Softwares/Face200/MakeScript/MakeScript.exe ./music     4001   5000
00032: #   $(ESTUDIOROOT)/Softwares/Face200/MakeScript/MakeScript.exe ./photo     5001   6000
00033: #   $(ESTUDIOROOT)/Softwares/Face200/MakeScript/MakeScript.exe ./setting   6001   7000
00034: #   $(ESTUDIOROOT)/Softwares/Face200/MakeScript/MakeScript.exe ./ebook     7001   8000
00035: #   $(ESTUDIOROOT)/Softwares/Face200/MakeScript/MakeScript.exe ./fm        8001   9000
00036: #   $(ESTUDIOROOT)/Softwares/Face200/MakeScript/MakeScript.exe ./calendar  9001   10000
00037: #   $(ESTUDIOROOT)/Softwares/Face200/MakeScript/MakeScript.exe ./dialog    10001  11000
00038: #   $(ESTUDIOROOT)/Softwares/Face200/MakeScript/MakeScript.exe ./record    11001  12000
00039: #   $(ESTUDIOROOT)/Softwares/Face200/MakeScript/MakeScript.exe ./dv            12001  13000
00040:
00041:     $(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./init      1      1000
00042:     $(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./home      1001   2000
00043:     $(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./movie     2001   3000
00044:     $(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./explorer  3001   4000
00045:     $(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./music     4001   5000
00046:     $(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./photo     5001   6000
00047:     $(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./setting   6001   7000
00048: #   $(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./ebook     7001   8000
00049: #   $(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./fm        8001   9000
00050:     $(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./calendar  9001   10000
00051:     $(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./dialog    10001  11000
00052: #   $(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./record    11001  12000
00053:     $(SDKROOT)/tools/build_tools/Face200/MakeScript/MakeScript.exe ./dv            12001  13000
00054:
00055:     # create theme for the whole case
00056:     -rm touchtheme.script
00057:     cat ./config/touchthemehdr.script   \
00058:         ./init/init.script              \
00059:         ./home/home.script              \
00060:         ./movie/movie.script            \
00061:         ./explorer/explorer.script            \
00062:         ./music/music.script            \
00063:         ./photo/photo.script            \
00064:         ./setting/setting.script            \
00065:         ./calendar/calendar.script            \
00066:         ./dialog/dialog.script            \
00067:         ./dv/dv.script            \
00068:         >> touchtheme.script
00069:
00070:     # build the theme
00071:     #$(ESTUDIOROOT)/Softwares/Face200/FaceBuilderCmd/FaceBuilderCmd.exe ./config/config.ini
00072:     $(SDKROOT)/tools/build_tools/Face200/FaceBuilderCmd/FaceBuilderCmd.exe ./config/config.ini
00073:     # copy the theme to workspace
00074:     cp ./theme.h    $(ROOT)/../../include/res/theme.h
00075:     cp ./theme.bin $(WORKSPACEPATH)/beetles/rootfs/apps/theme.bin
```

图【7】

以下是对 makefile 工作流程的分析，以“#”开头的行为注释，在此不做详细说明。

【line19、line20】定义变量，“.”表示当前目录，“..”表示上一级目录，主要用于指定 makefile 调用的 CROSSTOOL.CFG 文件、windows 应用程序 MakeScript.exe 和 FaceBuilderCmd.exe 在 SDK 中的相对路径；

【line23】包含了 SDK 中的 CROSSTOOL.CFG 文件，该文件包含了第 75 行中 WORKSPACEPATH 的定义；

【line25】makefile 语法中的目标；

【line41~line53】调用 MakeScript.exe 对指定的子文件夹的内容进行分析统计。

比如第 41 行，对./init 子文件夹的内容进行分析，同时限定条目数的数的范围为 1~1000。在 make clean;make 运行本脚本期间，MakeScript.exe 会对 init 文件夹下有多少个 bmp 文件进行统计，将统计结果输出到 init.scipt 文件中，以备第 57 行的 cat 命令使用；

开发者掌握本流程之后，可自行按照需求增减和修改子文件夹及子文件夹中的内容，并在此 Makefile 中做好 MakeScript.exe + 文件夹 + 条目数范围的命令配对。

【line56】rm 命令，删除命令，以防止前一次 make clean;make 运行的中间文件 touchtheme.script 遗留下来；

【line57~line68】cat 把 line41~line53 运行的文本集中拷贝到新的 touchtheme.script 文件当中，以备第 72 行的 FaceBuilderCmd.exe 调用，等同于是把所有子文件夹的统计结果集中到一个 script 文本中；

【line72】调用 FaceBuilderCmd.exe 应用程序，touchtheme.script 作为隐性输入文件，./config/config.ini 作为显性配置文件，运行之后输出 theme.bin 和 theme.h。这时就把各个子文件夹下分散的 bmp 图标资源按照一定数据格式正和岛了 theme.bin 文件当中，同时，将每一个 bmp 文件分配到的唯一宏定义输出到 theme.h 头文件，方便 C 语言调用了；

【line74、line75】当前目录只是临时工作区，theme.bin 和 theme.h 最终需要通过 cp 命令拷贝到指定的目录下；

至此，theme 文件夹下的脚本分析完了。

11.4.2. 资源文件 theme.bin 和 theme.h 的使用范例

参考下图【8】和图【9】，是 SDK 中对 charge_energy_0.bmp 图片的调用方式，该代码实现在“SDK\livedesk\beetles\res\theme\theme.h”文件和“SDK\livedesk\beetles\init\headbar\headbar_uipara.c”文件中。

- [theme.h (livedesk\beetles\res\theme)]

tions View Window Help

```
00015:
00016: #define     STYLEID                                   0x0
00017:
00018: #define     NAME_INIT_CHARGE_ENERGY_0_BMP             "INIT_CHARGE_ENERGY_0_BMP"
00019: #define     ID_INIT_CHARGE_ENERGY_0_BMP               1
00020:
00021: #define     NAME_INIT_CHARGE_ENERGY_1_BMP             "INIT_CHARGE_ENERGY_1_BMP"
00022: #define     ID_INIT_CHARGE_ENERGY_1_BMP               2
00023:
00024: #define     NAME_INIT_CHARGE_ENERGY_2_BMP             "INIT_CHARGE_ENERGY_2_BMP"
00025: #define     ID_INIT_CHARGE_ENERGY_2_BMP               3
```

图【8】

[headbar_uipara.c (livedesk\beetles\init\headbar)]

ons View Window Help

```
00157:          headbar_uipara.res_id.vol_id2   = ID_INIT_ENERGY_2_BMP;
00158:          headbar_uipara.res_id.vol_id3   = ID_INIT_ENERGY_3_BMP;
00159:          headbar_uipara.res_id.vol_id4   = ID_INIT_ENERGY_4_BMP;
00160:          headbar_uipara.res_id.vol_id5   = ID_INIT_ENERGY_5_BMP;
00161:
00162:          headbar_uipara.res_id.charge_vol_id0    = ID_INIT_CHARGE_ENERGY_0_BMP;
00163:          headbar_uipara.res_id.charge_vol_id1    = ID_INIT_CHARGE_ENERGY_1_BMP;
00164:          headbar_uipara.res_id.charge_vol_id2    = ID_INIT_CHARGE_ENERGY_2_BMP;
00165:          headbar_uipara.res_id.charge_vol_id3    = ID_INIT_CHARGE_ENERGY_3_BMP;
00166:          headbar_uipara.res_id.charge_vol_id4    = ID_INIT_CHARGE_ENERGY_4_BMP;
00167:          headbar_uipara.res_id.charge_vol_id5    = ID_INIT_CHARGE_ENERGY_5_BMP;
00168:          headbar_uipara.res_id.no_vol_id         = ID_INIT_NO_VOL_BMP;
00169:          headbar_uipara.res_id.usb_connect_vol_id = ID_INIT_USB_CONNECT_VOL_BMP;
```

图【9】

11.5. 语言文字编码工具

（本节待确认——编码转换通过 mod_charset 模块实现，不再使用 charset.bin 作为编码转换源。）

本节主要介绍语言文字编码字符集的生成工具。

要在 LCD 模组上显示一个文字，就需要获取这个文字对应的点阵数据，包含了若干个字符的点阵数据的文件就叫字库。

Melis2.0 系统的点阵字库是采用的是 unicode 编码，所以要想获得一个字符的点阵字库，就要先获得这个字符的 unicode 编码，这就好比在新华字典里用拼音查汉字一样。

但是，我们经常会遇到代码编辑器不用 unicode 编码的情况，比如简体中文的常用字符集编码是 GBK，日语的常用字符集编码是 Shift-JIS，这个时候就需要把一个字符的非 unicode 编码转换成 unicode 编码，再来从字库中搜索到对应的点阵数据。

进入到“SDK\Softwares\多国语言编码转换数据制作步骤\多国语言编码转换数据生成\”运行 TransCode.bat 就可以重新生成新的字符集转换表 charset.bin，这个转换表是按照默认的配置生成的。下面是修改方法。

打开《TransCode 使用说明.txt》看到如下图【10】所示的内容，该文档注明了《TransCode.bat》脚本的格式，该脚本将调用 TransCode.exe 程序，第二个和第三个文件作为 TransCode.exe 的输入配置文件和输出文件名。

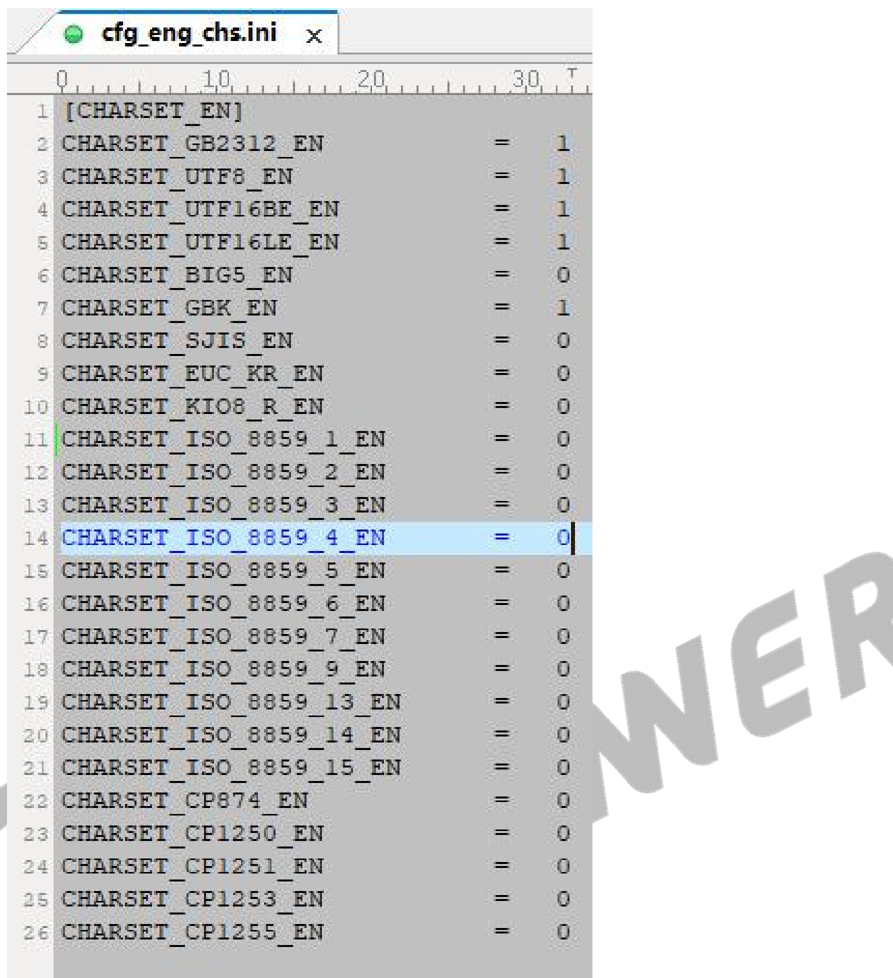
```
;多国语言编码转换工具名称    配置文件          输出文件名
TransCode.exe               cfg_eng_chs.ini  charset.bin
```

图【10】

以 cfg_eng_chs.ini 和 charset.bin 为例，charset.bin 就是我们所需要的字符集转换表，cfg_eng_chs.ini 是配置文件，指定输出文件中包含哪几种字符集的转换，参考图【11】，该配置中等于 1 的条目就是被包含的字符集，包括 GB2312、UTF8、UTF16BE、UTF16LE、GBK 编码，这些编码的字符在生成的 charset.bin 中都有一一对应的 unicode 码。剩下的等于 0 的就是在 charset.bin 文件中的没有对应 unicode 码的字符集编码类型。

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved




```
[CHARSET_EN]
CHARSET_GB2312_EN       =  1
CHARSET_UTF8_EN         =  1
CHARSET_UTF16BE_EN      =  1
CHARSET_UTF16LE_EN      =  1
CHARSET_BIG5_EN         =  0
CHARSET_GBK_EN          =  1
CHARSET_SJIS_EN         =  0
CHARSET_EUC_KR_EN       =  0
CHARSET_KIO8_R_EN       =  0
CHARSET_ISO_8859_1_EN   =  0
CHARSET_ISO_8859_2_EN   =  0
CHARSET_ISO_8859_3_EN   =  0
CHARSET_ISO_8859_4_EN   =  0
CHARSET_ISO_8859_5_EN   =  0
CHARSET_ISO_8859_6_EN   =  0
CHARSET_ISO_8859_7_EN   =  0
CHARSET_ISO_8859_9_EN   =  0
CHARSET_ISO_8859_13_EN  =  0
CHARSET_ISO_8859_14_EN  =  0
CHARSET_ISO_8859_15_EN  =  0
CHARSET_CP874_EN        =  0
CHARSET_CP1250_EN       =  0
CHARSET_CP1251_EN       =  0
CHARSET_CP1253_EN       =  0
CHARSET_CP1255_EN       =  0
```

图【11】

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

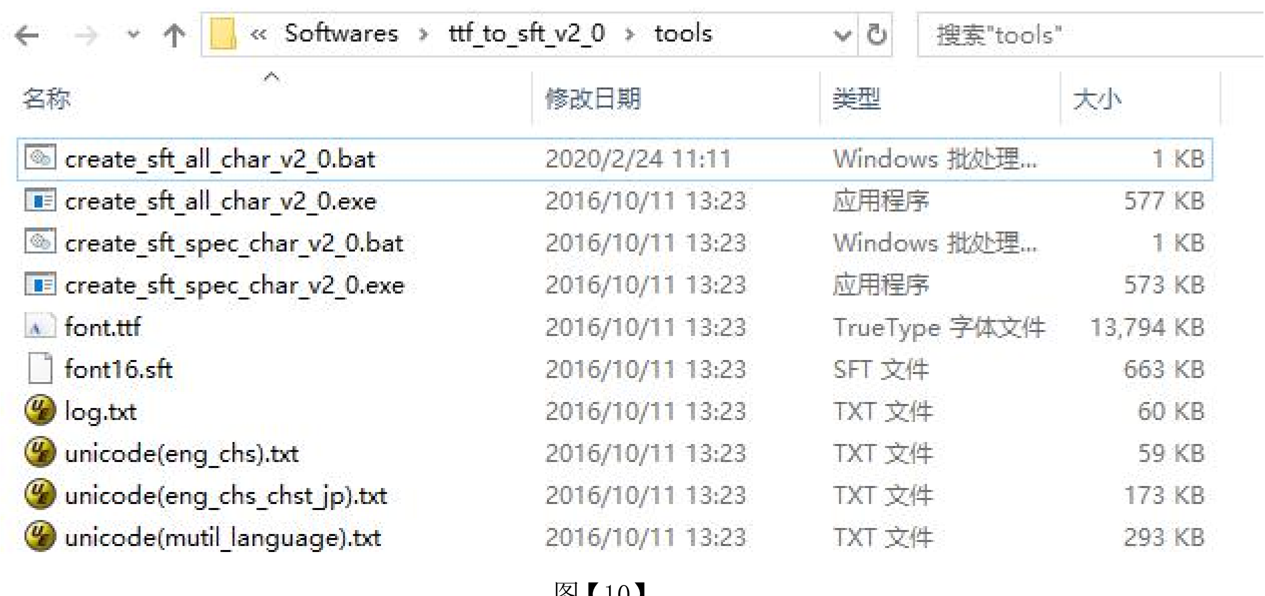
11.6. 字库制作工具

本节主要介绍点阵字库的生成工具，Melis2.0 支持的点阵字库为自主开发的 sft 后缀文件的字库，使用指定的工具，把常用的 ttf 格式字库转换而来，在存储空间和转换时间上达到一个比较好的平衡。

进入“Software\ttf_to_sft_v2_0\tools”文件夹；所包含的文件如下图【10】所示，开发者可以通过运行 create_sft_all_char_v2_0.bat 或 create_sft_spec_char_v2_0.bat 脚本来生成想目标 sft 文件。

【create_sft_all_char_v2_0.bat】把 ttf 文件中的所有字符和点阵数据都转化输出到 sft 文件；

【create_sft_spec_char_v2_0.bat】通过配置文件的形式，把配置文件中指定的字符和点阵数据从 ttf 中转化输出到 sft 文件。



名称	修改日期	类型	大小
create_sft_all_char_v2_0.bat	2020/2/24 11:11	Windows 批处理...	1 KB
create_sft_all_char_v2_0.exe	2016/10/11 13:23	应用程序	577 KB
create_sft_spec_char_v2_0.bat	2016/10/11 13:23	Windows 批处理...	1 KB
create_sft_spec_char_v2_0.exe	2016/10/11 13:23	应用程序	573 KB
font.ttf	2016/10/11 13:23	TrueType 字体文件	13,794 KB
font16.sft	2016/10/11 13:23	SFT 文件	663 KB
log.txt	2016/10/11 13:23	TXT 文件	60 KB
unicode(eng_chs).txt	2016/10/11 13:23	TXT 文件	59 KB
unicode(eng_chs_chst_jp).txt	2016/10/11 13:23	TXT 文件	173 KB
unicode(mutil_language).txt	2016/10/11 13:23	TXT 文件	293 KB

图【10】

11.6.1. 全字库转换

通过文本编辑器打开 create_sft_all_char_v2_0.bat 脚本，如下图【11】所示。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
::   input arguments
::   0: exe file
::   1: source font file
::   2: destination font file
::   3: reverse flag.          set 1 only when the source font file is aw font
::   4: pixel mode.            1 : mono mode, 1-bit bitmap; 2 : gray mode, 8-bit bitmap.
::   5: pixel size 1
::   ...pixel size 2
::   ...pixel size 3
::   ... ...


  create_sft_all_char_v2_0.exe      ^
  font.ttf                         ^
  font16.sft                       ^
  0                                 ^
  1                                 ^
  16                                ^
  32                                ^
  > log.txt
```

图【11】

【参数 0】：create_sft_all_char_v2_0.exe 表示脚本调用的 windows 应用程序；

【参数 1】：source font file 表示转换的源文件，开发人员可参考格式自行更换 ttf 源文件；

【参数 2】：destination font file 表示输出文件名，开发人员可参考格式自行命名；

【参数 3】：reverse flag 为保留参数，默认为 0，开发人员可不做变更；

【参数 4】：pixel mode：表示像素模式，为点阵数据中，每一个点的像素位宽，默认为 1；

【参数 5】：pixel size：表示转换之后的字号大小；

【参数 6】：pixel size：表示转换之后的字号大小，可理解为同一字库中包含了多种字号格式；

log.txt 文件为转换日志输出文件；

11.6.2. 指定字符集的字库转换

通过文本编辑器打开 create_sft_spec_char_v2_0.bat 脚本，如下图【12】所示。

```
::    input arguments
::    0: exe file
::    1: source font file
::    2: destination font file
::    3: reverse flag.           set 1 only when the source font file is aw font
::    4: pixel mode.             1 : mono mode, 1-bit bitmap; 2 : gray mode, 8-bit bitmap.
::    5: unicode list file
::    6: pixel size 1
::    ...pixel size 2
::    ...pixel size 3
::    ... ...


  create_sft_spec_char_v2_0.exe     ^
  font.ttf                        ^
  font16.sft                  ^
  0                                 ^
  1                                 ^
  unicode(eng_chs).txt                        ^
  16                                ^
  > log.txt
```

图【12】

【参数 0】：create_sft_spec_char_v2_0.exe 表示脚本调用的 windows 应用程序；

【参数 1】：source font file 表示转换的源文件，开发人员可参考格式自行更换 ttf 源文件；

【参数 2】：destination font file 表示输出文件名，开发人员可参考格式自行命名；

【参数 3】：reverse flag 为保留参数，默认为 0，开发人员可不做变更；

【参数 4】：pixel mode：表示像素模式，为点阵数据中，每一个点的像素位宽，默认为 1；

【参数 5】：unicode list file：表示配置文件清单，exe 应用程序会按照该清单的编码对 ttf 文件进行点阵数据的转换，输出到【参数 2】所命名的 sft 文件中；

【参数 6】：pixel size：表示转换之后的字号大小；

log.txt 文件为转换日志输出文件；

本工具为只使用了极少量字符的方案所准备的，不必将所有 ttf 字库中的所有字符都附带到方案中去，以此来节省 flash 的空间。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

11.6.3. 字库使用范例

由于字库的具体使用细节会在软件流程中体现(比如现在显示屏上显示一串文字)，所以在此不做过多扩展，仅向刚开始接触 Melis 系统的开发者简单展示一下字库的存放和创建，代码是如何关联到字库文件的。

字库生成之后存放在“SDK\workspae\suniv\beetles\rootfs\res\fonts”文件夹下，如下图【13】，该文件会在固件打包的过程中一起打包进 ePDKv100.img 固件文件中。

图【13】

图【14】是应用对字库文件的调用，仅在此作为示例供开发人员参考理解，该示例代码在 app_root_init.c 文件中。

```
026: __s32 app_root_init_res(void)
027: {
028:     rat_init();
029:
030:     SWFFont = GUI_SFT_CreateFont(16, BEETLES_APP_ROOT"res\\fonts\\font16.sft");
031:     if(SWFFont == NULL)
032:     {
033:         __err("create font fail...\n");
034:         return EPDK_FAIL;
035:     }
036:     GUI_SetFont(SWFFont);
037:
038:     return EPDK_OK;
039: }
040:
```

图【14】

11.7. UI 字符资源包文件 lang.bin

类似于 UI 图标资源的集中管理方式，Melis 系统对 UI 字符资源也采取集中管理的方式，为人机界面中的每一条字符串信息分配一个 ID 号，同时将所有字符串的 unicode 编码集中到 lang.bin 文件中。具体使用时，通过 lang.h 中宏定义的 ID 号在 lang.bin 中搜索出想要的字符串的 unicode 编码，继而按照上一节所述用 unicode 编码在 sft 字库中查找点阵数据将其显示到屏幕上。

lang 是英语 language 的缩写。

lang.h 和 lang.bin 是在“SDK\livedesk\beetles\res\lang”路径下编译生成的，如图【15】所示。

图【16】

11.7.1. 生成 lang.h 和 lang.bin

可使用文本编辑工具打开 makefile 文件，如下图【17】所示

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
00019: ROOT    = .
00020: SDKROOT = $(ROOT)/../../../..
00021:
00022: #导入交叉编译器通用配置
00023: include $(SDKROOT)/includes/cfgs/CROSSTOOL.CFG
00024:
00025:
00026: buildlang:
00027: #   $(ESTUDIOROOT)/Softwares/LangBuild/LangOSDBuild.exe ./lang.xml
00028:     $(SDKROOT)/tools/build_tools/LangBuild/LangOSDBuild.exe ./lang.xml
00029:     cp ./lang.h     $(ROOT)/../../include/res/lang.h
00030:     cp ./lang.bin   $(WORKSPACEPATH)/beetles/rootfs/apps/lang.bin
00031:
00032:
00033: # 删除生成的中间文件
00034: clean:
00035:     -rm ./lang.h
00036:     -rm ./lang.bin
```

图【17】

【line19~line20】定义 ROOT 和 SDKROOT 变量，“.”表示当前目录，“..”表示上一级目录，在第 23 行中使用，用来指定 CROSSTOOL.CFG 在 SDK 中的相对路径；在第 28 行使用，用来指定脚本调用的 windows 应用程序 LangOSDBuild.exe 在 SDK 中的相对路径；
【line23】包含了 SDK 中的 CROSSTOOL.CFG 文件，该文件中定义了第 30 行中的 WORKSPACEPATH 这个变量；
【line26】makefile 语法中的目标；
【line28】调用 LangOSDBuild.exe 来处理 lang.xml 文件，运行结果生成 lang.h 和 lang.bin 到当前文件夹下；
【line29、line30】把 lang.h 和 lang.bin 分别拷贝到指定的目录中取；

到此还不能了解到 lang.bin 生成的规则细节，因为规则在 lang.xml 文件中，可用文本编辑器，打开 lang.xml 文件，如下图【18】所示，将就其中的主要内容进行分析。

```
00001: <OSDTable>
00002: <Groups count="11">
00003: <OSDItem name="init" startid="1" maxid="1000" file="init\init.txt"/>
00004: <OSDItem name="home" startid="1001" maxid="2000" file="home\home.txt"/>
00005: <OSDItem name="explorer" startid="2001" maxid="3000" file="explorer\explorer.txt"/>
00006: <OSDItem name="movie" startid="3001" maxid="4000" file="movie\movie.txt"/>
00007: <OSDItem name="music" startid="4001" maxid="5000" file="music\music.txt"/>
00008: <OSDItem name="calendar" startid="5001" maxid="6000" file="calendar\calendar.txt"/>
00009: <OSDItem name="setting" startid="6001" maxid="7000" file="setting\setting.txt"/>
00010: <OSDItem name="photo" startid="7001" maxid="8000" file="photo\photo.txt"/>
00011: <OSDItem name="dialog" startid="8001" maxid="9000" file="dialog\dialog.txt"/>
00012: <OSDItem name="dv" startid="9001" maxid="10000" file="dv\dv.txt"/>
00013: <OSDItem name="ota" startid="10001" maxid="11000" file="ota\ota.txt"/>
00014: </Groups>
00015: <LangType count="3">
00016: <LangItem name="ChineseS" langId="0x410"/>
00017: <LangItem name="ChineseT" langId="0x420"/>
00018: <LangItem name="English" langId="0x400"/>
00019: </LangType>
00020: <Output bin=".\lang.bin" head=".\lang.h"/>
00021: </OSDTable>
```

图【18】

【line2】数值“11”表示有 11 个条目需要运行，也就是 11 个子文件夹中的 txt 文件需要进行分析；

【line3~line13】以此列举了 11 个 txt 文件的相关信息，name 一般为子文件夹的名字，startid 和 maxid 划定了一个范围，且各行命令中的 id 范围不能重叠，每一个文本中的字符串 id 就在这个范围内分配，原则上是保证各条命令的范围不两两互相重叠，单条命令的范围要大于 txt 文本中的字符串总条目数；

【line15】语言类型总数 3，表示每个字符串的条目有多少种语言类型，适用于支持多语言显示的产品方案；

【line16】语言名称，ChineseS 表示简体中文，0x410 指定了该语言类型在 lang.h 中的宏定义值，此处的内容需要跟“SDK\livedesk\beetles\include\mod_desktop\util\elibs_language.h”头文件定义的适配，如下图【19】’；

【line17】语言名称，ChineseT 表示繁体中文，0x420 指定了该语言类型在 lang.h 中的宏定义值；

【line18】语言名称，English 表示英语，0x400 指定了该语言类型在 lang.h 中的宏定义值；

t - [elibs_language.h (livedesk\beetles\include\mod_desktop\util)]

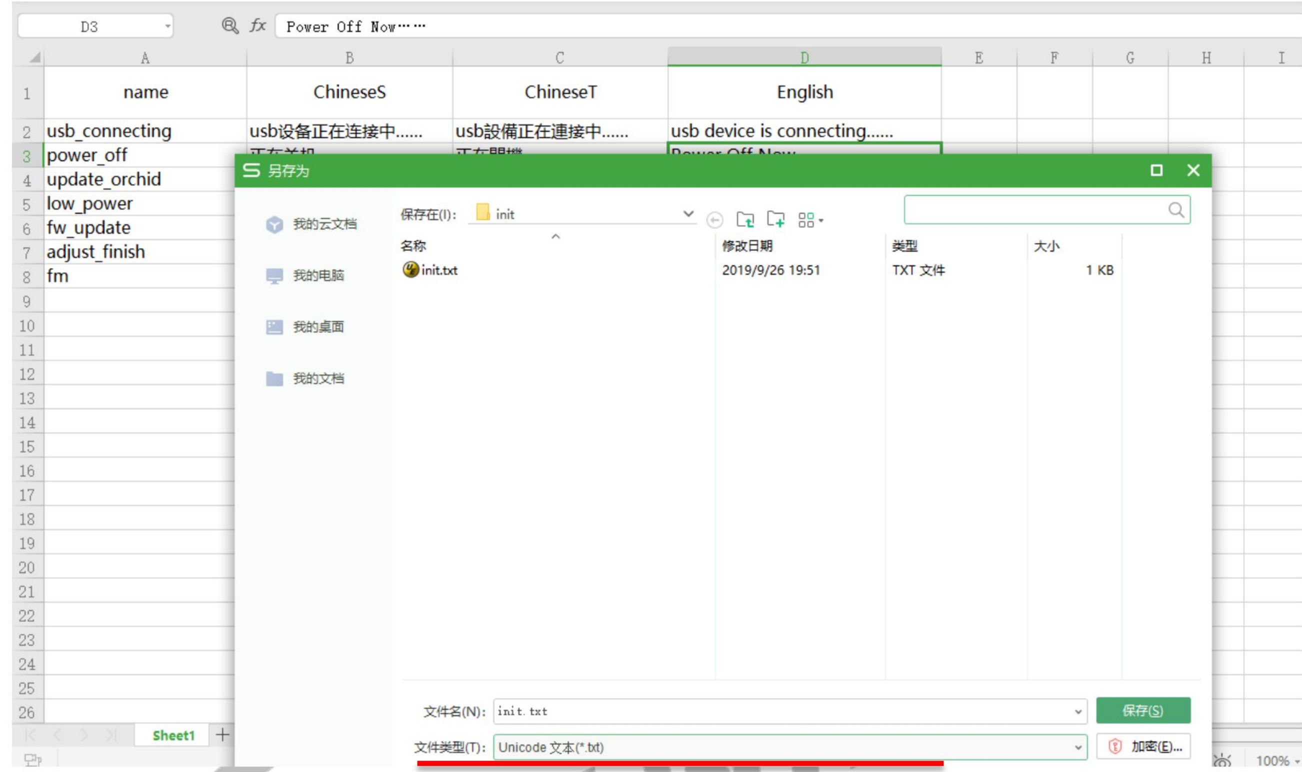
ptions View Window Help

```
00028: */
00029: #ifndef  __ELIBS_LANGUAGE_H__
00030: #define  __ELIBS_LANGUAGE_H__
00031:
00032:
00033:
00034: typedef  enum
00035: {
00036:     EPDK_LANGUAGE_ENM_ENGLISH    = 0x400,     // 英语
00037:     EPDK_LANGUAGE_ENM_CHINESES   = 0x410,     // 简体中文
00038:     EPDK_LANGUAGE_ENM_CHINESET   = 0x420,     // 繁体中文
00039:     EPDK_LANGUAGE_ENM_JAPANESE   = 0x430,     // 日语
00040:     EPDK_LANGUAGE_ENM_KOREAN     = 0x440,     // 韩语
00041:     EPDK_LANGUAGE_ENM_GERMAN     = 0x450,     // 德语
00042:     EPDK_LANGUAGE_ENM_SPANISH    = 0x460,     // 西班牙语
00043:     EPDK_LANGUAGE_ENM_FRENCH     = 0x470,     // 法语
00044:     EPDK_LANGUAGE_ENM_ITALIAN    = 0x480,     // 意大利语
00045:     EPDK_LANGUAGE_ENM_PORTUGUESE = 0x490,     // 葡萄牙语
00046:     EPDK_LANGUAGE_ENM_DUTCH      = 0x4a0,     // 荷兰语
00047:     EPDK_LANGUAGE_ENM_RUSSIAN    = 0x4b0,     // 俄语
00048:     EPDK_LANGUAGE_ENM_POLISH     = 0x4c0,     // 波兰语
00049:     EPDK_LANGUAGE_ENM_TURKISH    = 0x4d0,     // 土耳其语
00050:     EPDK_LANGUAGE_ENM_CZECH      = 0x4e0,     // 捷克语
00051:     EPDK_LANGUAGE_ENM_DANISH     = 0x4f0,     // 丹麦语
00052:     EPDK_LANGUAGE_ENM_HUNGARIAN  = 0x500,     // 匈牙利语
00053:     EPDK_LANGUAGE_ENM_SWEDISH    = 0x510,     // 瑞典语
00054:     EPDK_LANGUAGE_ENM_THAI       = 0x520,     // 泰语
00055:     EPDK_LANGUAGE_ENM_HEBREW     = 0x530,     // 希伯来语
00056:     EPDK_LANGUAGE_ENM_ARABIC     = 0x540,     // 阿拉伯语
00057:
00058:     EPDK_LANGUAGE_ENM_UNKNOWN    = -1         // always the last one
00059: } ? end __epdk_language_enm_e ? __epdk_language_enm_e;
00060:
00061:
00062: #define EPDK_LANGUAGE_NAME_ENGLISH     "ENGLISH"     // 英语
00063: #define EPDK_LANGUAGE_NAME_CHINESES    "CHINESES"    // 简体中文
00064: #define EPDK_LANGUAGE_NAME_CHINESET    "CHINESET"    // 繁体中文
00065: #define EPDK_LANGUAGE_NAME_JAPANESE    "JAPANESE"    // 日语
00066: #define EPDK_LANGUAGE_NAME_KOREAN      "KOREAN"      // 韩语
00067: #define EPDK_LANGUAGE_NAME_GERMAN      "GERMAN"      // 德语
00068: #define EPDK_LANGUAGE_NAME_SPANISH     "SPANISH"     // 西班牙语
00069: #define EPDK_LANGUAGE_NAME_FRENCH      "FRENCH"      // 法语
00070: #define EPDK_LANGUAGE_NAME_ITALIAN     "ITALIAN"     // 意大利语
00071: #define EPDK_LANGUAGE_NAME_PORTUGUESE  "PORTUGUESE"  // 葡萄牙语
00072: #define EPDK_LANGUAGE_NAME_DUTCH       "DUTCH"       // 荷兰语
00073: #define EPDK_LANGUAGE_NAME_RUSSIAN     "RUSSIAN"     // 俄语
00074: #define EPDK_LANGUAGE_NAME_POLISH      "POLISH"      // 波兰语
00075: #define EPDK_LANGUAGE_NAME_TURKISH     "TURKISH"     // 土耳其语
00076: #define EPDK_LANGUAGE_NAME_CZECH       "CZECH"       // 捷克语
00077: #define EPDK_LANGUAGE_NAME_DANISH      "DANISH"      // 丹麦语
00078: #define EPDK_LANGUAGE_NAME_HUNGARIAN   "HUNGARIAN"   // 匈牙利语
00079: #define EPDK_LANGUAGE_NAME_SWEDISH     "SWEDISH"     // 瑞典语
00080: #define EPDK_LANGUAGE_NAME_THAI        "THAI"        // 泰语
00081: #define EPDK_LANGUAGE_NAME_HEBREW      "HEBREW"      // 希伯来语
00082: #define EPDK_LANGUAGE_NAME_ARABIC      "ARABIC"      // 阿拉伯语
00083:
```

图【19】

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

txt 文本中包含的内容就是字符串条目的 unicode 编码，在 excel 文件中编辑后，另存为 unicode 编码格式的文本文件，如下图【20】所示。



图【20】

11.7.2. 资源文件 lang.h 和 lang.bin 的使用范例

参考下图【21】，该代码在“SDK\livedesk\beetles\init\ui\live_init.c”文件中，由于 lang.bin 文件包含了 3 中语言类型，所以在显示文本之前要先设定一种语言类型，以确定搜索该种语言类型的字符串 unicode，Melis 设定语言类型的接口函数为 dsk_langres_set_type，如图【21】所示。

[live_init.c (livedesk\beetles\init\ui)]

ons View Window Help

```
00086:
00087:            dsk_set_auto_off_time(para->poweroff);
00088:            __msg("para->poweroff=%d\n", para->poweroff);
00089:
00090:            dsk_langres_set_type(para->language);
00091:            __msg("para->language=%d\n", para->language);
00092:            para->output = LION_DISP_LCD;
00093:
00094:            if(1 == para->keytone)
00095:            {
00096:                dsk_keytone_set_state(SET_KEYTONE_ON);
00097:            }
00098:            else
00099:            {
00100:                dsk_keytone_set_state(SET_KEYTONE_OFF);
00101:            }
00102:
00103:            if(1 == para->time_set)
00104:            {
00105:                esTIME_SetTime(&para->time);
00106:                esTIME_SetDate(&para->date);
00107:                para->time_set = 0;
00108:            }
00109:        } ? end if para ?
00110:        else
00111:        {
00112:            dsk_display_set_lcd_bright(LION_BRIGHT_LEVEL12);
00113:            dsk_volume_set(20);
00114:            dsk_set_auto_off_time(0);
00115:            g_set_close_scn_time(0);
00116:            dsk_langres_set_type(EPDK_LANGUAGE_ENM_CHINESES);
00117:            dsk_keytone_set_state(SET_KEYTONE_OFF);
00118:        }
```

图【21】

参考下图【22】和图【23】，是 SDK 中对 init.txt 文本中对 usb_connecting 字符串的使用方式，该代码实现在“SDK\livedesk\beetles\res\lang\lang.h”文件和“SDK\livedesk\beetles\init\ui\dialog_scene\dialog_scene.c”文件中。

- [lang.h (livedesk\beetles\res\lang)]

ptions View Window Help

```
00021: #define __LANG__H__
00022:
00023: // LangId
00024: #define LANG_CHINESES_TYPE        0x410
00025: #define LANG_CHINESET_TYPE        0x420
00026: #define LANG_ENGLISH_TYPE         0x400
00027:
00028: // StringID
00029: #define        STRING_USB_CONNECTING        0x01
00030: #define        STRING_POWER_OFF        0x02
00031: #define        STRING_UPDATE_ORCHID        0x03
00032: #define        STRING_LOW_POWER        0x04
```

图【22】

[dialog_scene.c (livedesk\beetles\init\ui\dialog_scene)]

ions View Window Help

```
00204:
00205:             __here__;
00206:             if( p_scene->usb_connect )
00207:             {
00208:                 __here__;
00209:                 p_scene->cur_dialog = USBD_DIALOG;
00210:                 scene_dialog_create(p_scene, STRING_USB_CONNECTING, USBD_DIALOG_ID);
00211:                 __here__;
00212:             }
00213:             if( p_scene->orchid_update )
00214:             {
00215:                 //p_scene->cur_dialog = ORCHID_DIALOG;
00216:                 //GUI_SetTimer(msg->h_deswin, ORCHID_UPDATE_DIALOG_TIME_ID, ORCHID_UPDATE_TOUT , NU
00217:                 //scene_dialog_create(p_scene, STRING_UPDATE_ORCHID, ORCHID_DIALOG_ID);
00218:             }
00219:             if( p_scene->fw_update )
00220:             {
00221:                 p_scene->cur_dialog = FW_UPDATA_DIALOG;
00222:                 scene_dialog_create(p_scene, STRING_FW_UPDATE, FW_UPDATE_DIALOG_ID);
00223:             }
```

图【23】

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

12. 配置文件 sys_config.fex

12.1. 配置文件的作用

Melis2.0 系统的配置文件路径是“SDK\workspace\suniv\eFex\sys_config.fex”，其目的是为客户提供一种不用修改代码，只需修改配置选项，就能把一些常用功能适配到自己方案的方式。这个文件在《Melis2.0 QUICK START.docx》中介绍 UART 串口的引脚配置时提到过，用来配置 TX/RX 引脚，因为不同客户方案分配的串口引脚不尽相同，所以放到配置文件里面来修改。

本文档会介绍 sys_config.fex 的一般规则，再介绍 SDK 代码是用哪些函数接口来读取 sys_config.fex 文件的内容的，这样客户可以更好的理解、变更、添加配置项了。最后本文档还会对 sys_config.fex 文件中已有的一些配置条目的作用进行讲解。

12.2. 配置文件的规则和读取函数接口

12.2.1. 配置项格式

配置文件中的每一组配置项都是按照如下结构组成的：

```
[主键]
子健 1 = 值 1
子健 2 = 值 2
...
子健 N = 值 N
```

一个主键下可包含一个或多个子健。

12.2.2. 注释符号 “;”

分号 “;” 是注释符，注释一行，除注释符 “;” 之后的内容外，其他符号一律不得使用中文符号，参考下图【1】。

```
;----------------------------------------------------------------------------------------------------
; 说明：  脚本中的字符串区分大小写，用户可以修改"="后面的数值，但是不要修改前面的字符串
; 描述gpio的形式：Port:端口+组内序号<功能分配><内部电阻状态><驱动能力><输出电平状态>
; pinName = port:P[A-J]<CFG><PULL><DRV_LEVEL><DATA>
;----------------------------------------------------------------------------------------------------

[update_key]
;--------------------------------------------------------------------------------
;-------key_type  0:不需要强制升级   1:单个按键    2:两个按键组合           -------
;--------------------------------------------------------------------------------
key_type  = 0

;--------------------------------------------------------------------------------
;------------------   以下是只有单个按键(0:拉低  1:拉高)的情况   --------------------
;--------------------------------------------------------------------------------
key_value0  = 0
port0    = 1
port_num0   = 1

```

图【1】

12.2.3. 子健的值的类型

子健的值有如下四种:

1】十进制数;

2】十六进制数，以“0x”开头

3】字符串，以英文双引号表示;

4】GPIO，以 port:P[A-J]portnumber<FuncNumber><PULL><DRV_LEVEL><DATA>的格式表示;

12.2.4. 读取配置项的函数接口

介绍了配置文件的规则之后，以下介绍配置项的读取函数接口，也就是在编写代码时，使用哪些函数来获得 sys_config.fex 中指定的子键的值。

参考下图【2】,是“SDK\includes\emod\sys_config.h”头文件中声明的函数接口,获取 sys_config.fex 的键值即使用的如下接口,这些函数是内核函数，采用软中断的机制实现，在声明时带有 SYSCALL_CFG，在此暂不深入讨论，读者可当一般函数声明即可。

```
00050: SYSCALL_CFG(__s32, esCFG_GetKeyValue       )(char *SecName, char *KeyName, int Value[], int Count);
00051: SYSCALL_CFG(__s32, esCFG_GetSecKeyCount    )(char *SecName);
00052: SYSCALL_CFG(__s32, esCFG_GetSecCount       )(void);
00053: SYSCALL_CFG(__s32, esCFG_GetGPIOSecKeyCount)(char *GPIOSecName);
00054: SYSCALL_CFG(__s32, esCFG_GetGPIOSecData    )(char *GPIOSecName, void *pGPIOCfg, int GPIONum);
```

图【2】

12.2.4.1. esCFG_GetKeyValue

__s32 esCFG_GetKeyValue(char *SecName, char *keyName, int Value[], int Count);

【功能】：获取子键值的函数;

【参数 1】：SecName，主键名;

【参数 2】：keyName，子健名;

【参数 3】：Value，接收子健值的指针地址，读取的内容存放到这里;

【参数 4】：Count，子健的字长，子健的内容以 4 字节为单位表示，以匹配 32bit 的处理器;

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

参考下图【3】，是获取 tvout_para 主键下，tv_en 子健的内容，开发者可以作为参考。

```
[g_display_switch_output.c (livedesk\beetles\init\display)]
ions  View  Window  Help
00105: static __hdle          pc11_hdl = 0;
00106:
00107: static __s32 __app_pullup_tv_en(void)
00108: {
00109:     __s32           ret;
00110:     user_gpio_set_t  gpio_set[1];
00111:
00112:     if(!pc11_hdl)
00113:     {
00114:         /* 申请tv_en */
00115:         eLIBs_memset(gpio_set, 0, sizeof(user_gpio_set_t));
00116:         ret = esCFG_GetKeyValue("tvout_para", "tv_en", (int *)gpio_set, sizeof(user_gpio_set_t)/4);
00117:         if (!ret)
```

图【3】

12.2.4.2. esCFG_GetSecKeyCount

__s32 esCFG_GetSecKeyCount(char *SecName);

【功能】获取主键下的子健个数，没有该主键则返回-1；

【参数 1】SecName，主键名；

12.2.4.3. esCFG_GetSecCount

__s32 esCFG_GetSecCount(void);

【功能】获取 sys_config.fex 中主键的个数；

12.2.4.4. esCFG_GetGPIOSecKeyCount

__s32 esCFG_GetGPIOSecKeyCount(char *GPIOSecName);

【功能】获取主键下 GPIO 类型子健的个数，没有 GPIO 类型的子健就返回 0；

【参数 1】GPIOSecName，主键名；

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

12.2.4.5. esCFG_GetGPIOSecData

```
__s32 esCFG_GetGPIOSecData(char *GPIOSecName, void *pGPIOCfg, int GPIONum);
```

【功能】获取指定逐渐下的 GPIO 类型的子健的键值；

【参数 1】GPIOSecName，主键名；

【参数 2】pGPIOCfg，接收键值的指针地址；

【参数 3】GPIONum，要读取的前 N 个 GPIO 类型子健值；

本函数不用指定子健名，直接读取某个主键下的 GPIO 类型的子健，例如

```
esCFG_GetGPIOSecData("msdet_para", (void *)gpio_set, 1);
```

可直接通过传递参数 msdet_para 和 1 来获取 sys_config.fex 文件 msdet_para 主键下前 1 个 GPIO 类型子健的内容，刚好 msdet_para 下有且只有 1 个 GPIO 类型的子健，且没有其他类型的子健内容。回顾 esCFG_GetKeyValue 函数，如果通过 esCFG_GetKeyValue 来获取 GPIO 类型的子健，则一次只能读取 1 个。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

13. 驱动编程概述

系统提供的驱动包括存储类、按键、触屏、LCD、USB、AUDIO、POWER 和 FM 等等。

13.1. 驱动的挂载和卸载

有些驱动是在系统启动的时候挂载的，有些驱动是应用程序主动挂载的，挂载应用程序调用接口：

```
esDEV_Plugin("\\drv\xxx.drv", 0, 0, 1);
```

卸载驱动调用接口：

```
esDEV_Plugout("\\drv\xxx.drv", 0);
```

注意参数 0 是驱动在固件区的完整文件名，比如安装 FM 驱动：

```
esDEV_Plugin("\\drv\\fm.drv", 0, 0, 1);
```

13.2. 驱动访问

每个驱动定义了若干功能，应用程序可以通过系统 IOCTL 的方式去调用具体的功能，具体接口见头文件目录“D:\winners\ePDK\includes\emod”。

首先打开驱动，获取驱动的句柄：

```
ES_FILE *p_drv = eLIBs_fopen("b:\\1\\2", "r+");
```

1 是驱动注册时候的主设备名，2 是从设备名，比如 FM 驱动注册的时候调用接口：

```
esDEV_DevReg("USER", "FM", &fm_dev_ops, 0);
```

那么访问 fm 驱动的时候实现下面的代码：

```
pfm = eLIBs_fopen("b:\\USER\\FM", "r+");
```

然后通过 ioctl 调用具体的功能，其中第三个参数和第四个参数分别是 aux 和 pbuffer，用于功能命令的附加参数，在功能命令描述中会详加解释：

```
eLIBs_fioctrl(p_drv, DRV_XXX_CMD_XXX, aux, (void *)pbuffer);
```

DRV_XXX_CMD_XXX 就是具体的功能命令，比如打开 FM 的发射功能：

```
eLIBs_fioctrl(pfm, DRV_FM_CMD_SEND_START, 0, (void *)freq);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

14. Audio Dirver(CTRL)

14.1. 挂载

```
esDEV_Plugin("\\drv\\audio.drv", 0, 0, 1);
```

14.2. 卸载

```
esDEV_Plugout("\\drv\\audio.drv", 0);
```

14.3. 访问

Audio 驱动一共注册了 5 个设备，分别是 CTRL，PLAY/PLAY0，REC 和 FM。本节介绍 CTRL 设备的使用。

```
ES_FILE *fctrl = eLIBs_fopen("b:\\AUDIO\\CTRL", "r+");
```

14.4. 功能命令列表

命令	描述
AUDIO_DEV_CMD_GET_VOLUME	获取音量
AUDIO_DEV_CMD_SET_VOLUME	设置音量
AUDIO_DEV_CMD_CHANGE_IF	用户改变播放模式
AUDIO_DEV_CMD_CLOSE_DEV	关闭音设备,会关闭硬件接口，慎用
AUDIO_DEV_CMD_SET_PROTECT_VOL	设置音量保护值
AUDIO_DEV_CMD_GET_PROTECT_VOL	获取音量保护值
AUDIO_DEV_CMD_SET_CHAN_MODE	设置声道模式(左、右、立体声)
AUDIO_DEV_CMD_GET_CHAN_MODE	获取声道模式
AUDIO_DEV_CMD_ENTER_STANDBY	进入 standby 模式
AUDIO_DEV_CMD_EXIT_STANDBY	退出 standby 模式
AUDIO_DEV_CMD_REG_CALLBACK	注册一个回调函数；没有实现
AUDIO_DEV_CMD_UNREG_CALLBACK	卸载一个回调函数；没有实现
AUDIO_DEV_CMD_GET_INTERFACE	取得当前音频播放的接口
AUDI_DEV_CMD_SET_HPCOM_DRIVE_MODE	设置耳机驱动模式
AUDIO_DEV_CMD_SWAP_OUTPUT_CHANNELS	交换左右输出通道
AUDIO_DEV_CMD_SET_SW_VOL_MAX	设置软件音量的最大值
AUDIO_DEV_CMD_GET_SW_VOL_MAX	获取软件音量可设最大值
AUDIO_DEV_CMD_SET_DAC_MAX_GAIN	设置功率放大器增益的最大值
AUDIO_DEV_CMD_GET_DAC_MAX_GAIN	获取功率放大器增益的最大值

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

AUDIO_DEV_CD_SET_USE_USER_VOLUME_MAP	设置用户音量的映射表
AUDIO_DEV_CMD_MUTE	设置设备静音
AUDIO_DEV_CMD_GET_MUTE	获取设备是否静音

14.5. 功能命令描述

14.5.1. AUDIO_DEV_CMD_GET_VOLUME

获取音量。

Parameters

aux

无效，设为 0。

pbuffer

指定音频设备类型__audio_device_type_t：

Value	Description
AUDIO_DEV_PLAY	
AUDIO_DEV_REC	
AUDIO_DEV_FM	

Return Value

__s32

音量。

14.5.2. AUDIO_DEV_CMD_SET_VOLUME

设置音量。

Parameters

aux

音量大小，必须满足 AUDIO_DEVICE_VOLUME_MIN<aux<AUDIO_DEVICE_VOLUME_MAX。

pbuffer

指定音频设备类型__audio_device_type_t：

Value	Description
AUDIO_DEV_PLAY	
AUDIO_DEV_REC	
AUDIO_DEV_FM	

Return Value

__s32

音量。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

14.5.3. AUDIO_DEV_CMD_CHANGE_IF

改变播放模式。

Parameters

aux

指定音频设备接口__audio_dev_interface_t：

Value	Description
AUDIO_DEV_IF_CODEC	
AUDIO_DEV_IF_IIS	
AUDIO_DEV_IF_SPDIF	

pbuffer

无效，设为 0。

Return Value

EPDK_OK

成功。

EPDK_FAIL

非法的音频设备接口或者改变音频设备接口失败。

14.5.4. AUDIO_DEV_CMD_CLOSE_DEV

关闭音设备,会关闭硬件接口，慎用。

Parameters

aux

指定音频设备类型__audio_device_type_t：

Value	Description
AUDIO_DEV_PLAY	
AUDIO_DEV_REC	
AUDIO_DEV_FM	

pbuffer

无效，设为 0。

Return Value

EPDK_OK

成功。

EPDK_FAIL

非法的音频设备类型。

14.5.5. AUDIO_DEV_CMD_SET_PROTECT_VOL

设置音量保护值。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

Parameters

aux

音量保护值。

pbuffer

无效，设为 0。

Return Value

__s32

音量保护值。

14.5.6. AUDIO_DEV_CMD_GET_PROTECT_VOL

获取音量保护值。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

音量保护值。

14.5.7. AUDIO_DEV_CMD_SET_CHAN_MODE

设置声道模式。

Parameters

aux

指定声道模式__audio_dev_chan_mode_t：

Value	Description
AUDIO_DEV_CHANNEL_STEREO	立体声
AUDIO_DEV_CHANNEL_LEFT	左声道
AUDIO_DEV_CHANNEL_RIGHT	右声道

pbuffer

无效，设为 0。

Return Value

EPDK_OK

成功。

EPDK_FAIL

不是播放设备，或者 spidf rawdata 被他人占用，或者指定非法的声道模式。

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

14.5.8. AUDIO_DEV_CMD_GET_CHAN_MODE

获取声道模式。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

Value	Description
AUDIO_DEV_CHANNEL_STEREO	立体声
AUDIO_DEV_CHANNEL_LEFT	左声道
AUDIO_DEV_CHANNEL_RIGHT	右声道

14.5.9. AUDIO_DEV_CMD_ENTER_STANDBY

进入 standby 模式。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

14.5.10. AUDIO_DEV_CMD_EXIT_STANDBY

退出 standby 模式。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

14.5.11. AUDIO_DEV_CMD_REG_CALLBACK

注册一个回调函数。没有实现

Parameters

aux

无效，设为 0。

pbuffer

回调函数。

Return Value

__s32

返回回调函数的序号。当已经注册的回调函数超过 AUDIO_MAX_CALLBACK_COUNT 时返回 EPDK_FAIL。

14.5.12. AUDIO_DEV_CMD_UNREG_CALLBACK

卸载一个回调函数。没有实现

Parameters

aux

注册时返回的回调函数的序号。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

成功。

EPDK_FAIL

非法的回调函数的序号。

14.5.13. AUDIO_DEV_CMD_GET_INTERFACE

取得当前音频播放的接口。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

当前音频播放的接口。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

14.5.14. AUDI_DEV_CMD_SET_HPCOM_DRIVE_MODE

设置耳机驱动模式。

Parameters

aux

0：设置为 AC 模式；1：设置为 DC 模式。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

14.5.15. AUDIO_DEV_CMD_SWAP_OUTPUT_CHANNELS

交换输出通道。

Parameters

aux

1：交换输出通道，左 dac 输出到右通道，右 dac 输出到左通道；0：不交换输出通道。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

14.5.16. AUDIO_DEV_CMD_SET_SW_VOL_MAX

设置软件音量的最大值。

Parameters

aux

设置的音量最大值

pbuffer

无效，设为 0。

Return Value

EPDK_OK

设置成功

EPDK_FAIL

设置失败。一般是因为设置的值太大，超出范围。

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

14.5.17. AUDIO_DEV_CMD_GET_SW_VOL_MAX

获取软件音量可调的最大值。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

音量最大值

14.5.18. AUDIO_DEV_CMD_SET_DAC_MAX_GAIN

设置功率放大器增益的最大值。

Parameters

aux

设置功率放大器增益的最大值，大小必须 <=63。

pbuffer

无效，设为 0。

Return Value

音量最大值

14.5.19. AUDIO_DEV_CMD_GET_DAC_MAX_GAIN

获取功率放大器增益的最大值。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

功率放大器增益的最大值

14.5.20. AUDIO_DEV_CD_SET_USE_USER_VOLUME_MAP

使用用户声音级别映射 pbuffer = 长度为 AUDIO_DEVICE_VOLUME_MAX+1 的数组，数组下标是软件 value,目前不支持 record 声音映射。

Parameters

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

aux

无效，设为 0。

pbuffer

长度为 AUDIO_DEVICE_VOLUME_MAX+1 的数组，数组下标是软件 value。

Return Value

EPDK_OK

14.5.21. AUDIO_DEV_CMD_MUTE

设置当前是否处于静音模式。

Parameters

aux

设置是否处于静音模式：aux：1，设置静音模式；0，取消静音模式。

pbuffer

无效，设为 0。

Return Value

功率放大器增益的最大值

14.5.22. AUDIO_DEV_CMD_GET_DAC_MAX_GAIN

获取当前是否处于静音模式。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

1：静音模式；0：正常模式

15. Audio Dirver(PLAY/PLAY0)

15.1. 挂载

```
esDEV_Plugin("\\drv\\audio.drv", 0, 0, 1);
```

15.2. 卸载

```
esDEV_Plugout("\\drv\\audio.drv", 0);
```

15.3. 访问

```
// 访问 PLAY
ES_FILE *fplay = eLIBs_fopen("b:\\AUDIO\\PLAY", "r+");
// 访问 PLAY0
ES_FILE *fplay0 = eLIBs_fopen("b:\\AUDIO\\PLAY0", "r+");
```

15.4. 功能命令列表

命令	描述
AUDIO_DEV_CMD_START	启动音频设备
AUDIO_DEV_CMD_STOP	停止音频设备
AUDIO_DEV_CMD_PAUSE	暂停音频设备
AUDIO_DEV_CMD_CONTINUE	继续音频设备
AUDIO_DEV_CMD_GET_SAMPCNT	获取音频的采样点数
AUDIO_DEV_CMD_SET_SAMPCNT	设置音频的采样点数
AUDIO_DEV_CMD_GET_PARA	获取用户参数
AUDIO_DEV_CMD_SET_PARA	设置用户参数
AUDIO_DEV_CMD_GET_VOLUME	获取音量
AUDIO_DEV_CMD_SET_VOLUME	设置音量
AUDIO_DEV_CMD_REG_USERMODE	音频用户注册用户模式
AUDIO_DEV_CMD_FLUSH_BUF	清除音频设备用户缓冲区内的音频数据
AUDIO_DEV_CMD_QUERY_BUFSIZE	查询用户缓冲区相关空间参数
AUDIO_DEV_CMD_RESIZE_BUF	重新指定用户缓冲区的大小
AUDIO_DEV_CMD_WRITE_DATA	向音频设备写入数据
AUDIO_DEV_CMD_DATA_FINISH	通知音频设备当前用户的数据已经输入完毕

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

AUDIO_DEV_CMD_SET_PLAYMODE	设置播放模式
AUDIO_DEV_CMD_MUTE	设置开启或关闭静音模式

15.5. 功能命令描述

15.5.1. AUDIO_DEV_CMD_START

启动播放设备。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

15.5.2. AUDIO_DEV_CMD_STOP

停止播放设备。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

15.5.3. AUDIO_DEV_CMD_PAUSE

暂停播放设备。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

成功。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

EPDK_FAIL

播放设备的状态并非处于 AUDIO_DEV_STAT_RUN。

15.5.4. AUDIO_DEV_CMD_CONTINUE

继续播放设备。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

成功。

EPDK_FAIL

播放设备的状态并非处于 AUDIO_DEV_STAT_PAUS。

15.5.5. AUDIO_DEV_CMD_GET_SAMPCNT

获取音频的采样点数。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

音频的采样点数。

15.5.6. AUDIO_DEV_CMD_SET_SAMPCNT

设置音频的采样点数。

Parameters

aux

音频采样的频数。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

15.5.7. AUDIO_DEV_CMD_GET_PARA

获取用户参数。

Parameters

aux

无效，设为 0。

pbuffer

用于返回用户参数__audio_dev_para_t 的指针。

Return Value

EPDK_OK

成功。

EPDK_FAIL

非法指针。

15.5.8. AUDIO_DEV_CMD_SET_PARA

设置用户参数。

Parameters

aux

无效，设为 0。

pbuffer

指定用户参数__audio_dev_para_t 的指针。

Return Value

EPDK_OK

成功。

EPDK_FAIL

非法指针。

15.5.9. AUDIO_DEV_CMD_GET_VOLUME

获取音量。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

当前音量。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

15.5.10. AUDIO_DEV_CMD_SET_VOLUME

设置音量。

Parameters

aux

音量。

pbuffer

无效，设为 0。

Return Value

volume

当前音量。

15.5.11. AUDIO_DEV_CMD_REG_USERMODE

音频用户注册用户模式。

Parameters

aux

用户模式__audio_play_user_t：

Value	Description
AUDIO_PLAY_USR_MASTER	
AUDIO_PLAY_USR_SLAVE	
AUDIO_PLAY_USR_KEY	

pbuffer

无效，设为 0。

Return Value

EPDK_OK

成功。

EPDK_FAIL

非法用户模式。

15.5.12. AUDIO_DEV_CMD_FLUSH_BUF

清除音频设备用户缓冲区内的音频数据。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

EPDK_OK

15.5.13. AUDIO_DEV_CMD_QUERY_BUFSIZE

查询用户缓冲区的大小。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

buffer size

用户缓冲区的大小。

EPDK_FAIL

播放设备的状态并非处于 AUDIO_DEV_STAT_RUN。

15.5.14. AUDIO_DEV_CMD_RESIZE_BUF

重新指定用户缓冲区的大小。

Parameters

aux

指定用户缓冲区的大小。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

成功。

EPDK_FAIL

播放设备的状态并非处于 AUDIO_DEV_STAT_IDLE。

15.5.15. AUDIO_DEV_CMD_WRITE_DATA

向音频设备写入数据。

Parameters

aux

写入双字节数据的数量。

pbuffer

双字节指针，指向写入数据。

Return Value

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

size

实际写入双字节数据的数量。

15.5.16. AUDIO_DEV_CMD_DATA_FINISH

通知音频设备当前用户的数据已经输入完毕。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

EPDK_FAIL

15.5.17. AUDIO_DEV_CMD_SET_PLAYMODE

设置 SPDIF 模式下的播放模式。如果是 RWDATA 播放模式，播放数据不解码直接输出；如果是普通播放模式，播放数据先解码再输出。

Parameters

aux

SPDIF 模式下的播放模式__audio_dev_spdif_playmode_t：

Value	Description
AUDIO_DEV_SPDIF_NORMALPLAY	
AUDIO_DEV_SPDIF_RAWDATAPLAY	

pbuffer

无效，设为 0。

Return Value

EPDK_OK

成功。

EPDK_FAIL

当前模式并非 AUDIO_DEV_IF_SPDIF。

15.5.18. AUDIO_DEV_CMD_MUTE

设置设备是否静音。

Parameters

aux

设置是否处于静音模式：aux：1，设置静音模式；0，恢复音量。

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

pbuffer

无效，设为 0。

Return Value

EPDK_OK

成功。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

16. Audio Dirver(REC)

16.1. 挂载

```
esDEV_Plugin("\\drv\\audio.drv", 0, 0, 1);
```

16.2. 卸载

```
esDEV_Plugout("\\drv\\audio.drv", 0);
```

16.3. 访问

```
ES_FILE *frec = eLIBs_fopen("b:\\AUDIO\\REC", "r+");
```

16.4. 功能命令列表

命令	描述
AUDIO_DEV_CMD_START	启动音频设备
AUDIO_DEV_CMD_STOP	停止音频设备
AUDIO_DEV_CMD_PAUSE	暂停音频设备
AUDIO_DEV_CMD_CONTINUE	继续音频设备
AUDIO_DEV_CMD_GET_SAMPCNT	获取音频的采样点数
AUDIO_DEV_CMD_SET_SAMPCNT	设置音频的采样点数
AUDIO_DEV_CMD_GET_PARA	获取用户参数
AUDIO_DEV_CMD_SET_PARA	设置用户参数
AUDIO_DEV_CMD_GET_VOLUME	获取音量
AUDIO_DEV_CMD_SET_VOLUME	设置音量
AUDIO_DEV_CMD_REG_USERMODE	音频用户注册用户模式
AUDIO_DEV_CMD_FLUSH_BUF	清除音频设备用户缓冲区内的音频数据
AUDIO_DEV_CMD_QUERY_BUFSIZE	查询用户缓冲区相关空间参数
AUDIO_DEV_CMD_READ_DATA	向音频设备读取数据
AUDIO_DEV_CMD_SET_PLAYMODE	设置播放模式

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

16.5. 功能命令描述

16.5.1. AUDIO_DEV_CMD_START

启动录音设备。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

16.5.2. AUDIO_DEV_CMD_STOP

关闭录音设备。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

16.5.3. AUDIO_DEV_CMD_PAUSE

停止录音设备。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

成功。

EPDK_FAIL

录音设备的状态并非处于 AUDIO_DEV_STAT_RUN。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

16.5.4. AUDIO_DEV_CMD_CONTINUE

继续录音设备。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

成功。

EPDK_FAIL

录音设备的状态并非处于 AUDIO_DEV_STAT_PAUS。

16.5.5. AUDIO_DEV_CMD_GET_SAMPCNT

获取音频的采样点数。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

音频的采样点数。

16.5.6. AUDIO_DEV_CMD_SET_SAMPCNT

设置音频的采样点数。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

16.5.7. AUDIO_DEV_CMD_GET_PARA

获取用户参数。

Parameters

aux

无效，设为 0。

pbuffer

用户返回用户参数__audio_dev_para_t 的指针。

Return Value

EPDK_OK

成功。

EPDK_FAIL

非法指针。

16.5.8. AUDIO_DEV_CMD_SET_PARA

设置用户参数。

Parameters

aux

无效，设为 0。

pbuffer

用户参数__audio_dev_para_t 的指针。

Return Value

EPDK_OK

成功。

EPDK_FAIL

非法指针。

16.5.9. AUDIO_DEV_CMD_GET_VOLUME

获取音量。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

当前音量。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

16.5.10. AUDIO_DEV_CMD_SET_VOLUME

设置音量。

Parameters

aux

音量。

pbuffer

无效，设为 0。

Return Value

__s32

当前音量。

16.5.11. AUDIO_DEV_CMD_REG_USERMODE

音频用户注册用户模式。

Parameters

aux

指定录音用户模式__audio_rec_user_t：

Value	Description
AUDIO_REC_USR_LINEIN	
AUDIO_REC_USR_FMIN	
AUDIO_REC_USR_MIC	
AUDIO_REC_USR_MIXER	

pbuffer

无效，设为 0。

Return Value

EPDK_OK

成功。

EPDK_FAIL

非法的录音用户模式。

16.5.12. AUDIO_DEV_CMD_FLUSH_BUF

清除录音设备用户缓冲区内的音频数据。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

Return Value

EPDK_OK

16.5.13. AUDIO_DEV_CMD_QUERY_BUFSIZE

查询用户缓冲区的大小。

Parameters

aux

指定用户缓冲区大小的类型__audio_dev_query_buf_size_type_t：

Value	Description
AUDIO_DEV_QUERY_BUF_SIZE_DATA	
AUDIO_DEV_QUERY_BUF_SIZE_FREE	
AUDIO_DEV_QUERY_BUF_SIZE_TOTAL	

pbuffer

无效，设为 0。

Return Value

0 或者 size

0 表示非法的用户缓冲区。

EPDK_FAIL

非法用户缓冲区大小的类型。

16.5.14. AUDIO_DEV_CMD_READ_DATA

从录音设备读取数据。

Parameters

aux

读取的字节数。

pbuffer

指向接收数据的指针。

Return Value

size

实际读取的数据。

16.5.15. AUDIO_DEV_CMD_SET_PLAYMODE

设置录音时候是否同时需要播放，默认不播放。

Parameters

aux

播放模式__audio_dev_rec_playmode_t：

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

Value	Description
AUDIO_DEV_REC_NOPLAY	录音的时候不需要播放
AUDIO_DEV_REC_WITHPLAY	录音时需要播放

pbuffer

无效，设为 0。

Return Value

EPDK_OK

成功。

EPDK_FAIL

非法的播放模式。

17. Audio Dirver(FM)

17.1. 挂载

```
esDEV_Plugin("\\drv\\audio.drv", 0, 0, 1);
```

17.2. 卸载

```
esDEV_Plugout("\\drv\\audio.drv", 0);
```

17.3. 访问

```
ES_FILE *fm = eLIBs_fopen("b:\\AUDIO\\FM", "r+");
```

17.4. 功能命令列表

命令	描述
AUDIO_DEV_CMD_START	启动音频设备
AUDIO_DEV_CMD_STOP	停止音频设备

17.5. 功能命令描述

17.5.1. AUDIO_DEV_CMD_START

启动音频设备。

Parameters

aux

Value	Description
0	初始化 FM
0xff	初始化 LineIn

pbuffer

无效，设为 0。

Return Value

EPDK_OK

17.5.2. AUDIO_DEV_CMD_STOP

停止音频设备。

Parameters

aux

Value	Description
0	退出 FM
0xff	退出 LineIn

pbuffer

无效，设为 0。

Return Value

EPDK_OK

17.6. 17.6 示例

```
ES_FILE *fm = NULL;

// 挂载 audio dirver
esDEV_Plugin("\\drv\\audio.drv", 0, 0, 1);

// 打开设备
fm = eLIBs_fopen("b:\\AUDIO\\FM", "r+");
if (!fm)
{
    __wrn("cannot open b:\\AUDIO\\FM\n");
    // RETURN
}

// 启动 FM
eLIBs_fioctrl(fm, AUDIO_DEV_CMD_START, 0, 0);

// DO SOMETHING

// 停止 FM
eLIBs_fioctrl(fm, AUDIO_DEV_CMD_STOP, 0, 0);

// 卸载 audio dirver
esDEV_Plugout("\\drv\\audio.drv", 0);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

18. RTC Dirver

18.1. 挂载

```
esDEV_Plugin("\\drv\\rtc.drv", 0, 0, 1);
```

18.2. 卸载

```
esDEV_Plugout("\\drv\\rtc.drv", 0);
```

18.3. 访问

```
ES_FILE *frtc = eLIBs_fopen("b:\\HWSC\\RTC", "r+");
```

18.4. 功能命令列表

命令	描述
RTC_CMD_GET_TIME	获取当前时间
RTC_CMD_SET_TIME	设置当前时间
RTC_CMD_GET_DATE	获取当前日期
RTC_CMD_SET_DATE	设置当前日期
RTC_CMD_REQUEST_ALARM	请求警报器
RTC_CMD_RELEASE_ALARM	释放警报器
RTC_CMD_START_ALARM	启动警报器
RTC_CMD_STOP_ALARM	停止警报器
RTC_CMD_QUERY_ALARM	查询警报器
RTC_CMD_QUERY_INT	查询 RTC 中断

18.5. 功能命令描述

18.5.1. RTC_CMD_GET_TIME

获取当前时间。

Parameters

aux

无效，设为 0。

pbuffer

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

指向__time_t 的指针。

Return Value

EPDK_OK

EPDK_FAIL

18.5.2. RTC_CMD_SET_TIME

设置当前时间。

Parameters

aux

无效，设为 0。

pbuffer

指向__time_t 的指针。

Return Value

EPDK_OK

EPDK_FAIL

18.5.3. RTC_CMD_GET_DATE

获取当前日期。

Parameters

aux

无效，设为 0。

pbuffer

指向__date_t 的指针。

Return Value

EPDK_OK

EPDK_FAIL

18.5.4. RTC_CMD_SET_DATE

设置当前日期。

Parameters

aux

无效，设为 0。

pbuffer

指向__date_t 的指针。

Return Value

EPDK_OK

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

EPDK_FAIL

18.5.5. RTC_CMD_REQUEST_ALARM

请求警报器。

Parameters

aux

警报器的类型，通常设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

返回 0 表示警报器已经被其他人占用，否则返回警报器的句柄。

18.5.6. RTC_CMD_RELEASE_ALARM

释放警报器。

Parameters

aux

无效，设为 0。

pbuffer

警报器的句柄。

Return Value

EPDK_OK

EPDK_FAIL

18.5.7. RTC_CMD_START_ALARM

启动警报器。

Parameters

aux

警报器持续的时间，单位是 s。

pbuffer

警报器的句柄。

Return Value

EPDK_OK

EPDK_FAIL

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

18.5.8. RTC_CMD_STOP_ALARM

停止警报器。

Parameters

aux

无效，设为 0。

pbuffer

警报器的句柄。

Return Value

EPDK_OK

EPDK_FAIL

18.5.9. RTC_CMD_QUERY_ALARM

查询警报器。

Parameters

aux

无效，设为 0。

pbuffer

警报器的句柄。

Return Value

__s32

当前警报器持续的时间，单位是 s。

18.5.10. RTC_CMD_QUERY_INT

查询 RCT 中断。

Parameters

aux

中断源。

pbuffer

无效，设为 0。

Return Value

__s32

0 表示没有中断，1 表示中断已经发生。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

19. IIC Dirver(TWI0/ TWI1/TWI2)

19.1. 挂载

```
esDEV_Plugin("\\drv\\twi.drv", 0, 0, 1);
```

19.2. 卸载

```
esDEV_Plugout("\\drv\\twi.drv", 0);
```

19.3. 访问

用户应该知道设备挂载在哪一路 TWI 总线上，打开没有激活的 TWI 总线会出错。

```
// 访问 TWI0
ES_FILE *ftwi0 = eLIBs_fopen("b:\\BUS\\TWI0", "r+");
// 访问 TWI1
ES_FILE *ftwi1 = eLIBs_fopen("b:\\BUS\\TWI1", "r+");
// 访问 TWI2
ES_FILE *ftwi1 = eLIBs_fopen("b:\\BUS\\TWI2", "r+");
```

19.4. 功能命令列表

命令	描述
TWI_WRITE_SPEC_RS	写命令，支持标准的 twi 写，带 restart
TWI_READ_SPEC_RS	读命令，支持标准的 twi 读，带 restart
TWI_READ_EX_NO_RS	新的 twi 读命令，不带 restart
TWI_READ_EX_STP_RS	新的 twi 读命令，stop 之后再 restart
TWI_SET_SCL_CLOCK	设置 twi 的 scl 时钟，典型的值为 100KHz、400KHz

19.5. 功能命令描述

19.5.1. TWI_WRITE_SPEC_RS

写命令，支持标准的 twi 写，带 restart。

Parameters

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

aux

无效，设为 0。

pbuffer

指向__twi_dev_para_ex_t 结构体的指针。

Return Value

EPDK_OK

EPDK_FAIL

19.5.2. TWI_READ_SPEC_RS

读命令，支持标准的 twi 读，带 restart。

Parameters

aux

无效，设为 0。

pbuffer

指向__twi_dev_para_ex_t 结构体的指针。

Return Value

EPDK_OK

EPDK_FAIL

19.5.3. TWI_READ_EX_NO_RS

新的 twi 读命令，不带 restart。

Parameters

aux

无效，设为 0。

pbuffer

指向__twi_dev_para_ex_t 结构体的指针。

Return Value

EPDK_OK

EPDK_FAIL

19.5.4. TWI_READ_EX_STP_RS

新的 twi 读命令，stop 之后再 restart。

Parameters

aux

无效，设为 0。

pbuffer

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

指向__twi_dev_para_ex_t 结构体的指针。

Return Value

EPDK_OK
EPDK_FAIL

19.5.5. TWI_SET_SCL_CLOCK

设置 twi 的 scl 时钟，典型的值为 100KHz、400KHz。

Parameters

aux

无效，设为 0。

pbuffer

__u32，twi 的 scl 时钟，单位是 Hz。当指定的值大于 400KHz 时默认设为 400KHz。

Return Value

EPDK_OK
EPDK_FAIL

19.6. 数据结构

19.6.1. __twi_dev_para_ex_t

```
typedef struct __TWI_DEV_PARA_EX
{
   __u16   slave_addr;
   __u16   slave_addr_flag;
   __u8   *byte_addr;
   __u16   byte_addr_width;
   __u16   byte_count;
   __u32   reserved;
   __u8   *data;
} __twi_dev_para_ex_t;
```

Members

slave_addr

从设备地址。

slave_addr_flag

从设置地址宽度，0：7bif；1：10bit。

byte_addr

需要读写的数据在从设备中的地址，低字节存放低地址，高字节存放高地址。

byte_addr_width

从设置地址宽度，0 或 1：8bit；其它数字代表字节数。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

byte_count

一次读写要完成的字节数。

reserved

保留位。

data

数据的地址

19.7. 示例

```
ES_FILE *ftwi;
__twi_dev_para_t stwi;
__u8 data[5];
__u8 byte_addr[2];

// 打开 twi 设备
ftwi = eLIBs_fopen("b:\\BUS\\TWI0", "r");

// 填充数据结构
stwi.slave_addr = 0x10;
stwi.slave_addr_flag = 0;
stwi.byte_addr = byte_addr;
stwi.byte_addr_width = 2;
stwi.byte_count = 5;
stwi.data = data;

// 通过 iic 读取数据
eLIBs_fioctrl(ftwi, TWI_READ_SPEC_RS, 0, (void *)&stwi);
// 通过 iic 写入数据
eLIBs_fioctrl(ftwi, TWI_WRITE_SPEC_RS, 0, (void *)&stwi);

// 需要关闭 twi 设备
eLIBs_fclose(ftwi);
```

如果遇到特殊的 twi 从设备，特指不需要指定数据在设备中的寄存器地址的，不需要数据结构中 byte_addr 的值，则将 byte_addr 的地址赋值为 0x0，如 stwi.byte_addr = 0x0，而 byte_addr_width 也设置为 0，其它不需要变化。

很多 iic 设备在直接读取的时候可以不带地址，那样访问出来的就是最后一次设定的地址的内容，需要访问固定不变的地址的时候会用，不过这样的访问也可以用带寄存器的方式代替只是多发了地址字节而已。所以一般建议写上地址字节。

补充说明，stwi.byte_count = 字节数，即设备中数据所在的地址的字节个数。通常一个设备中的地址数在 0 到 0xff 之间，在特殊设备中，地址数可能在 0 到 0xffffff 之间。这时，就需要指明字节数。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

20. Key Dirver

20.1. 挂载

```
esDEV_Plugin("\\drv\\key.drv", 0, 0, 1);
```

20.2. 卸载

```
esDEV_Plugout("\\drv\\key.drv", 0);
```

20.3. 访问

```
ES_FILE *fkey = eLIBs_fopen("b:\\INPUT\\KEY", "r+");
```

20.4. 功能命令列表

命令	描述
DRV_KEY_CMD_GETKEY	没有实现
DRV_KEY_CMD_PUTKEY	向输入子系统发送按键消息
DRV_KEY_CMD_SET_FIRST_DEBOUNCE_TIME	设置按键第一次按下时的响应时间
DRV_KEY_CMD_SET_FIRST_RPT_TIME	设置第一次按键按下后发第二次按键消息的时间间隔
DRV_KEY_CMD_SET_SBSEQ_RPT_TIME	设置按键按住不放时消息发送频率
DRV_KEY_CMD_GET_FIRST_DEBOUNCE_TIME	获取按键第一次按下时的响应时间
DRV_KEY_CMD_GET_FIRST_RPT_TIME	获取第一次按键按下后发第二次按键消息的时间间隔
DRV_KEY_CMD_GET_SBSEQ_RPT_TIME	获取按键按住不放时消息发送频率
DRV_KEY_CMD_GET_IRMASK	没有实现
DRV_KEY_CMD_GET_IRPOWERVALUE	没有实现
DRV_KEY_CMD_GET_HOLDKEYVALUEMAX	没有实现
DRV_KEY_CMD_GET_HOLDKEYVALUEMIN	没有实现

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

20.5. 功能命令描述

20.5.1. DRV_KEY_CMD_GETKEY

没有实现。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_FAIL

20.5.2. DRV_KEY_CMD_PUTKEY

向输入子系统发送按键消息。

Parameters

aux

Value	Description
0	键盘的按键消息
255	虚拟键盘的按键消息

pbuffer

当 aux=255 时，pbuffer 就是按键值；否则，pbuffer 是一个指向__key_msg_t 结构体的指针。

Return Value

EPDK_FAIL

20.5.3. DRV_KEY_CMD_SET_FIRST_DEBOUNCE_TIME

设置按键第一次按下的响应时间，在模块初始化时指定的采样频率作为按键一次按下的响应时间。

Parameters

aux

无效，设为 0。

pbuffer

__u32，按键第一次按下的响应时间。

Return Value

EPDK_FAIL

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

20.5.4. DRV_KEY_CMD_SET_FIRST_RPT_TIME

设置第一次按键按下后发第二次按键消息的时间间隔，同时也是长按键的阈值。

Parameters

aux

无效，设为 0。

pbuffer

__u32，时间间隔。

Return Value

EPDK_FAIL

20.5.5. DRV_KEY_CMD_SET_SBSEQ_RPT_TIME

设置按键按住不放时消息发送频率。

Parameters

aux

无效，设为 0。

pbuffer

__u32，消息发送频率。

Return Value

EPDK_FAIL

20.5.6. DRV_KEY_CMD_GET_FIRST_DEBOUNCE_TIME

获取按键第一次按下时的响应时间。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

按键第一次按下时的响应时间。

20.5.7. DRV_KEY_CMD_GET_FIRST_RPT_TIME

获取第一次按键按下后发第二次按键消息的时间间隔，同时也是长按键的阈值。

Parameters

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

第一次按键按下后发第二次按键消息的时间间隔。

20.5.8. DRV_KEY_CMD_GET_SBSEQ_RPT_TIME

获取按键按住不放时消息发送频率。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

按键按住不放时消息发送频率。

20.5.9. DRV_KEY_CMD_GET_IRMASK

没有实现。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_FAIL

20.5.10. DRV_KEY_CMD_GET_IRPOWERVALUE

没有实现。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

Return Value

EPDK_FAIL

20.5.11. DRV_KEY_CMD_GET_HOLDKEYVALUEMAX

没有实现。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_FAIL

20.5.12. DRV_KEY_CMD_GET_HOLDKEYVALUEMIN

没有实现。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_FAIL

20.6. 数据结构

20.6.1. __key_msg_t

```
typedef struct tag_key_msg
{
    __u32 key_value;
    __u32 flag;
} __key_msg_t;
```

Members

key_value

按键值。

flag

1 表示按键按下，0 表示按键抬起。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

21. IR Dirver

21.1. 挂载

```
esDEV_Plugin("\\drv\\ir.drv", 0, 0, 1);
```

21.2. 卸载

```
esDEV_Plugout("\\drv\\ir.drv", 0);
```

21.3. 访问

```
ES_FILE *fkey = eLIBs_fopen("b:\\INPUT\\IR_KEY", "r+");
```

21.4. 功能命令列表

命令	描述
DRV_IRKEY_CMD_PUTKEY	向输入子系统发送按键消息
DRV_IRKEY_CMD_SET_SBSEQ_RPT_TIME	设置按键连续按下时按键消息发送的频率
DRV_IRKEY_CMD_GET_SBSEQ_RPT_TIME	获取红外按键抬起的检测时间
DRV_IRKEY_CMD_DISPLAY_SCANCODE	显示扫描码
DRV_KEY_CMD_GET_IRPOWERVALUE	获取 POWER 键值
DRV_KEY_CMD_GET_IRMASK	获取用户码或地址码
DRV_IRKEY_CMD_CHG_REMOTER	设置用户码或地址码

21.5. 功能命令描述

21.5.1. DRV_IRKEY_CMD_PUTKEY

向输入子系统发送按键消息。

Parameters

aux

无效，设为 0。

pbuffer

指向 ir_key_msg_t 结构体的指针。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

Return Value

EPDK_FAIL

21.5.2. DRV_IRKEY_CMD_SET_SBSEQ_RPT_TIME

设置按键连续按下时按键消息发送的频率。注意，这不是简单的设置，而是把之前申请的 timer 删除后再重新创建一个 timer 才能设置这个时间。

Parameters

aux

无效，设为 0。

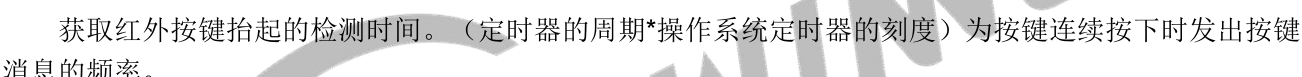
pbuffer

__u32，按键消息发送的频率。

Return Value

EPDK_FAIL

21.5.3. DRV_IRKEY_CMD_GET_SBSEQ_RPT_TIME



获取红外按键抬起的检测时间。（定时器的周期*操作系统定时器的刻度）为按键连续按下时发出按键消息的频率。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

红外按键抬起的检测时间。

21.5.4. DRV_IRKEY_CMD_DISPLAY_SCANCODE

显示扫描码，如果为 1 则控制台打印显示得到的扫描码，如果为 0 则关闭打印显示。主要是方便用户映射扫描码和按键消息。扫描码和用户码一起打印。

Parameters

aux

无效，设为 0。

pbuffer

__u32，1 为打印扫描码，0 为关闭打印扫描码。

Return Value

EPDK_FAIL

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

21.5.5. DRV_KEY_CMD_GET_IRPOWERVALUE

获取红外遥控的 POWER 的键值。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

红外遥控的 POWER 的键值。

21.5.6. DRV_KEY_CMD_GET_IRMASK

获取用户码或地址码

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_FAIL

21.5.7. DRV_IRKEY_CMD_CHG_REMOTER

显设置遥控器的用户码或者地址码，如果为 0 则表示不检测用户码或地址码。用户码或地址码用来区分不同的遥控器。获取地址码或用户码，直接调用 BSP_IR_key_display_scancode 打印出来即可以。

Parameters

aux

无效，设为 0。

pbuffer

__u32，用户码或地址码的值。

Return Value

EPDK_FAIL

21.6. 数据结构

21.6.1. _ir_key_msg_t

```
typedef struct tag_ir_key_msg
{
    __u32 key_value;
    __u32 flag;
} __ir_key_msg_t;
```

Members

key_value

按键值。

flag

1 表示按键按下，0 表示按键抬起。

22. SPI Dirver

22.1. 挂载

```
esDEV_Plugin("\\drv\\spi.drv", 0, 0, 1);
```

22.2. 卸载

```
esDEV_Plugout("\\drv\\spi.drv", 0);
```

22.3. 访问

```
// 访问 SPI0
ES_FILE *fspi0 = eLIBs_fopen("b:\\BUS\\SPI0", "r+");
// 访问 SPI1
ES_FILE *fspi1 = eLIBs_fopen("b:\\BUS\\SPI1", "r+");
```

这里需要判断返回成功还是失败 NULL，用户需要知道设备挂载在哪一个 spi 总线上，打开没有激活的 SPI 路数会出错，返回 NULL，是否激活该路 SPI 由方案的脚本配置决定。

22.4. 功能命令列表

关于 master 和 slave 模式，目前只支持 master 模式。

命令	描述
SPI_DEV_CMD_MASTER_RW	master 模式下的读写操作命令
SPI_DEV_CMD_SLAVE_READ	slave 模式下，接收数据命令，不支持
SPI_DEV_CMD_SLAVE_WRITE	slave 模式下，发送数据命令，不支持
SPI_DEV_CMD_CHECK_DUAL	查询 SPI 处于单线模式或者双线模式

22.5. 功能命令描述

22.5.1. SPI_DEV_CMD_MASTER_RW

master 模式下的读写操作命令。

Parameters

aux

无效，设为 0。

pbuffer

指向__spi_dev_set_xfer_t 结构体的指针。

Return Value

EPDK_OK

EPDK_FAIL

Code Example

```
// 填充__spi_dev_set_xfer_t 结构
__spi_dev_set_xfer_t spi_xfer;
spi_xfer.work_slot = DRV_SPI_SLOT_0;
spi_xfer.work_mode = DRV_SPI_WORK_MODE3;
spi_xfer.work_clk  = 10000000;
// 读操作
__u32 user_buf[10] = {0};
spi_xfer.rx_addr = user_buf;
spi_xfer.rx_count = 10;
spi_xfer.tx_addr = NULL;
spi_xfer.tx_count = 0;
// 写操作
__u32 user_buf[10] = {1};
spi_xfer.rx_addr = NULL;
spi_xfer.rx_count = 0;
spi_xfer.tx_addr = user_buf;
spi_xfer.tx_count = 10;
// 先写后读操作
__u32 rx_buf[10] = {0};
__u32 tx_buf[10] = {1};
spi_xfer.rx_addr = rx_buf;
spi_xfer.rx_count = 10;
spi_xfer.tx_addr = tx_buf;
spi_xfer.tx_count = 10;
// 使用下面命令操作驱动
eLIBs_fioctrl(fspi, SPI_DEV_CMD_MASTER_RW, 0, (void *)&spi_xfer);
```

22.5.2. SPI_DEV_CMD_SLAVE_READ

slave 模式下，接收数据命令，不支持。

Parameters

aux

无效，设为 0。

pbuffer

指向__spi_dev_set_xfer_t 结构体的指针。

Return Value

EPDK_OK

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

EPDK_FAIL

22.5.3. SPI_DEV_CMD_SLAVE_WRITE

slave 模式下，发送数据命令，不支持。

Parameters

aux

无效，设为 0。

pbuffer

指向__spi_dev_set_xfer_t 结构体的指针。

Return Value

EPDK_OK

EPDK_FAIL

22.5.4. SPI_DEV_CMD_CHECK_DUAL

查询 SPI 当前处于 单线 or 双线 模式。

Parameters

aux

无效，设为 0。

pbuffer

指向非 NULL 的指针。

Return Value

0 ： 单线 spi 模式

1 ： 双线 spi 模式

22.6. 数据结构

22.6.1. __spi_dev_set_xfer_t

```
typedef struct __SPI_DEV_SET_XFER
{
    void   *tx_addr;
    __u32   tx_count;
    void   *rx_addr;
    __u32   rx_count;
    __u32   work_mode;
    __u32   work_clk;
    __u32   work_slot;
    void    *reserved;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
} __spi_dev_set_xfer_t;
```

Members

tx_addr

发送的首地址，如果没有置 NULL。

tx_count

发送数据字节数。

rx_addr

接收的首地址，如果没有置 NULL。

rx_count

接收数据字节数。

work_mode

四种工作模式之一，pha 和 pol 的组合，不是 master 或 slave 模式。

Work Mode	Leading Edge	Trailing Edge
DRV_SPI_WORK_MODE0	Rising, Sample	Falling, Setup
DRV_SPI_WORK_MODE1	Rising, Setup	Falling, Sample
DRV_SPI_WORK_MODE2	Falling, Sample	Rising, Setup
DRV_SPI_WORK_MODE3	Falling, Setup	Rising, Sample

work_clk

单位：HZ，仅在 master 下有效；slave 模式下不需要配置这个参数。

work_slot

spi 设备在哪一个 cs 插槽。

Value	Description
DRV_SPI_SLOT_0	cs0
DRV_SPI_SLOT_1	cs1
DRV_SPI_SLOT_2	cs2
DRV_SPI_SLOT_3	cs3

reserved

保留，slave 模式下可做回调。

22.7. 7 示例

```
static ES_FILE *fspi = NULL;
__s32 spic_rw(__u32 spi_no, __u32 tcnt, __u8* txbuf, __u32 rcnt, __u8* rxbuf)
{
    __s32 ret = 0;
    __u32 busy = 0x55;
    __spi_dev_set_xfer_t para;
    eLIBs_memset(&para, 0, sizeof(__spi_dev_set_xfer_t));

    para.tx_addr  = txbuf;
    para.tx_count = tcnt;
    para.rx_addr  = rxbuf;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
    para.rx_count = rcnt;

    para.work_mode = DRV_SPI_WORK_MODE3;
    para.work_clk = 12000000;
    para.work_slot = DRV_SPI_SLOT_0;

    fspi = eLIBs_fopen("b:\\BUS\\SPI1", "r+");
    if (NULL == pFile_spi)
    {
        __inf("open SPI driver failed\n");
        return EPDK_FAIL;
    }

    ret = eLIBs_fioctrl(fspi, SPI_DEV_CMD_MASTER_RW, 0, (void *)&para);
    __inf("return value = %d\n", ret);
    eLIBs_fclose(fspi);
    fspi = NULL;

    return ret;
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

23. UART Dirver(UART0/ UART1/UART2)

23.1. 挂载

```
// 挂载 UART0
esDEV_Plugin("\\drv\\uart.drv", 0, 0, 1);
// 挂载 UART1
esDEV_Plugin("\\drv\\uart.drv", 1, 0, 1);
// 挂载 UART2
esDEV_Plugin("\\drv\\uart.drv", 2, 0, 1);
```

23.2. 卸载

```
// 卸载 UART0
esDEV_Plugout("\\drv\\uart.drv", 0);
// 卸载 UART1
esDEV_Plugout("\\drv\\uart.drv", 1);
// 卸载 UART2
esDEV_Plugout("\\drv\\uart.drv", 2);
```

23.3. 访问

```
// 访问 UART0
ES_FILE *fsuart0 = eLIBs_fopen("b:\\BUS\\UART0", "r+");
// 访问 UART1
ES_FILE *fsuart1 = eLIBs_fopen("b:\\BUS\\UART1", "r+");
// 访问 UART2
ES_FILE *fsuart1 = eLIBs_fopen("b:\\BUS\\UART2", "r+");
```

23.4. 功能命令列表

命令	描述
UART_CMD_SET_PARA	设置参数
UART_CMD_SET_BAUDRATE	设置波特率
UART_CMD_FLUSH	刷新缓冲区

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

23.5. 功能命令描述

23.5.1. UART_CMD_SET_PARA

设置串口通信参数。

Parameters

aux

无效，设为 0。

pbuffer

指向__uart_para_t *的指针。

Return Value

EPDK_OK

23.5.2. UART_CMD_SET_BAUDRATE

设置波特率。

Parameters

aux

无效，设为 0。

pbuffer

__u32 *，指向设置波特率值的指针。

Return Value

EPDK_OK

23.5.3. UART_CMD_FLUSH

刷新缓冲区。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

23.6. 示例

```
__u32 port_num ;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
ES_FILE* pfile_uart ;
char path[32];

__inf(" please input uart port num to test:");
port_num = 0;
__inf(" %d\n",port_num);

// 在挂载 UART 驱动过程中，会初始化 UART，将默认参数设置完成了，因此无需再单独设置参数
esDEV_Plugin("\\drv\\uart.drv", port_num, 0, 1);
esKRNL_TimeDly(10);
eLIBs_sprintf(path,"b:\\BUS\\UART%d",port_num);
__msg("path=%s\n",path);
pfile_uart =  eLIBs_fopen(path,"rb+");
__msg("pfile_uart=%x\n",pfile_uart);

eLIBs_fwrite("hello uart driver", 16, 1, pfile_uart);
while(1)
{
    eLIBs_fread((void*)path, 16, 1, pfile_uart);
    eLIBs_fwrite((void*)path, 16, 1, pfile_uart);

    esKRNL_TimeDly(200);
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

24. TP Dirver

24.1. 挂载

```
esDEV_Plugin("\\drv\\touchpanel.drv", 0, 0, 1);
```

24.2. 卸载

```
esDEV_Plugout("\\drv\\touchpanel.drv", 0);
```

24.3. 访问

```
ES_FILE *ftp = eLIBs_fopen("b:\\INPUT\\TP", "r+");
```

24.4. 功能命令列表

命令	描述
DRV_TP_CMD_REG	挂载回调函数，没有实现
DRV_TP_CMD_UNREG	卸载回调函数，没有实现
DRV_TP_CMD_ADJUST	屏幕校准命令
DRV_TP_CMD_SET_OFFSET_INFO	设置像素偏差
DRV_TP_CMD_GET_OFFSET_INFO	获取像素偏差
DRV_TP_CMD_SET_MSG_PERTIME	设置消息发送间隔
DRV_TP_CMD_GET_MSG_PERTIME	获取消息发送间隔
DRV_TP_CMD_SET_WORKMODE	设置工作模式

24.5. 功能命令描述

24.5.1. DRV_TP_CMD_REG

挂载回调函数，如果前面存在回调函数，会自动替换前一个回调函数。没有实现。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

EPDK_OK

24.5.2. DRV_TP_CMD_UNREG

卸载回调函数，卸载后，如果前面存在回调函数，会自动使用前一个回调函数。没有实现。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

24.5.3. DRV_TP_CMD_ADJUST

屏幕校准命令。

Parameters

aux

步骤。

pbuffer

指向__ev_tp_pos_t 的指针。

Remarks

TP 的屏幕校准 command 只有一条，即 DRV_TP_CMD_ADJUST，要完成这个功能则需要连续使用四次这条命令。需要一个数据结构：

```
__ev_tp_pos_t my_pos;
```

调用命令：

```
eLIBs_fioctrl(hdle, DRV_TP_CMD_ADJUST, 0, (void *)&my_pos);
```

在数据结构 my_pos 的 disp_xpos 和 disp_ypos 中填入屏幕坐标点，在 aux 中填写校准步骤调用此命令后，将等待用户对 step1-4 中提示的点进行点击，直到用户点击后才返回。建议 APP 以此点为中心，绘出一些标志性图标，如圆圈，十字等，并提示用户对该点进行点击。

以上，是对一个点进行校准的标准步骤。通常情况下，屏幕校准应该有四个校准点，并且应该是呈矩形分布的四个点。当对这样的四个点依次进行上面的校准方法后，驱动就可以获取到足够的数据，完成正确的计算。驱动对于这四个点有要求，第一个和第二个点的 x 坐标相同，第二个和第三个的 y 坐标相同，第三个和第四个的 x 坐标相同，第四个和第三个的 y 坐标相同，这样就呈现一个矩形状态 adjust_step 的顺序一定要正确，在驱动中会做校准，即顺序依次是 1，2，3，4，并且与坐标值的变化对应。

以下是 800×480 4 点屏幕校准简单教程：

```
// step1:
//      通知驱动准备校准 TP，驱动开始做相应的工作
eLIBs_fioctrl(hdle, DRV_TP_CMD_ADJUST, 0, 0);
```

```
// step2:
//      等待用户点击，点击后返回，这里不判断成功或失败
my_pos.disp_xpos = 100;
my_pos.disp_ypos = 100;
eLIBs_fioctrl(hdle, DRV_TP_CMD_ADJUST, 1, (void *)&my_pos);

// step3:
//      等待用户点击，点击后返回，这里不判断成功或失败
my_pos.disp_xpos = 700;
my_pos.disp_ypos = 100;
eLIBs_fioctrl(hdle, DRV_TP_CMD_ADJUST, 2, (void *)&my_pos);

// step4:
//      等待用户点击，点击后返回，这里不判断成功或失败
my_pos.disp_xpos = 700;
my_pos.disp_ypos = 400;
eLIBs_fioctrl(hdle, DRV_TP_CMD_ADJUST, 3, (void *)&my_pos);

// step5:
//      等待用户点击，点击后返回，这里不判断成功或失败
my_pos.disp_xpos = 100;
my_pos.disp_ypos = 400;
eLIBs_fioctrl(hdle, DRV_TP_CMD_ADJUST, 4, (void *)&my_pos);
```

到这里已经取得四个数据，驱动首先判断数据时候有效，成功则开始校准。用户需要判断返回值，如果成功，则退出校准，如果失败，应该返回 step2 重新开始读取数据。

5 点校准是在原来 4 点的基础上做的改进，主要是为了解决触摸屏在安装的时候由于某些原有没有安装正确，出现了倾斜等的问题。

当校准到中间的时候，由于某些原因不希望继续向下校准，可以调用以下的函数，通知触摸屏驱动，退出驱动的校准部分，恢复驱动的点触功能。如果需要重新进行校准，需要全部重新来。

```
eLIBs_fioctrl(hdle, DRV_TP_CMD_ADJUST, 6, (void *)&my_pos);
```

以下是 800×480 5 点屏幕校准简单教程：

```
// step1:
//      通知驱动准备校准 TP，驱动开始做相应的工作
//      执行完毕后立即返回
eLIBs_fioctrl(hdle, DRV_TP_CMD_ADJUST, 0, 0);

// step2:
//      用户点击，驱动会判断当前的点击是否正确，如果正确返回成功，否则返回失败
//      应用程序应该在收到失败的时候重新发送这个命令以及同样的参数
my_pos.disp_xpos = 100;
my_pos.disp_ypos = 100;
eLIBs_fioctrl(hdle, DRV_TP_CMD_ADJUST, 1, (void *)&my_pos);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
// step3:
//      用户点击，驱动会判断当前的点击是否正确，如果正确返回成功，否则返回失败
//      应用程序应该在收到失败的时候重新发送这个命令以及同样的参数
my_pos.disp_xpos = 700;
my_pos.disp_ypos = 100;
eLIBs_fioctrl(hdle, DRV_TP_CMD_ADJUST, 2, (void *)&my_pos);

// step4:
//      用户点击，驱动会判断当前的点击是否正确，如果正确返回成功，否则返回失败
//      应用程序应该在收到失败的时候重新发送这个命令以及同样的参数
my_pos.disp_xpos = 700;
my_pos.disp_ypos = 400;
eLIBs_fioctrl(hdle, DRV_TP_CMD_ADJUST, 3, (void *)&my_pos);

// step5:
//      用户点击，驱动会判断当前的点击是否正确，如果正确返回成功，否则返回失败
//      应用程序应该在收到失败的时候重新发送这个命令以及同样的参数
my_pos.disp_xpos = 100;
my_pos.disp_ypos = 400;
eLIBs_fioctrl(hdle, DRV_TP_CMD_ADJUST, 4, (void *)&my_pos);

// step6:
//      用户点击，驱动会判断当前的点击是否正确，如果失败返回失败
//      应用程序应该在收到失败的时候重新发送这个命令以及同样的参数
my_pos.disp_xpos = 400;
my_pos.disp_ypos = 250;
eLIBs_fioctrl(hdle, DRV_TP_CMD_ADJUST, 4, (void *)&my_pos);
```

到这里已经取得五个数据，驱动首先判断数据时候有效，成功则开始校准，否则返回-2。用户需要判断返回值，如果成功，则退出校准，如果是-2，应该返回 step2 重新开始读取数据。

24.5.4. DRV_TP_CMD_SET_OFFSET_INFO

设置像素偏差，x 或者 y 方向超过这个值时认为是一次移动，默认是 TP_DEFAULT_OFFSET。

Parameters

aux

像素偏差值。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

24.5.5. DRV_TP_CMD_GET_OFFSET_INFO

获取像素偏差。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

像素偏差值。

24.5.6. DRV_TP_CMD_SET_MSG_PERTIME

设置消息发送间隔，采样多少次发送一个消息，默认是 TP_DEFAULT_MSG_PERTIME。

Parameters

aux

消息发送间隔。

pbuffer

无效，设为 0。

Return Value

EPDK_OK

24.5.7. DRV_TP_CMD_GET_MSG_PERTIME

获取消息发送间隔。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

消息发送间隔。

24.5.8. DRV_TP_CMD_SET_WORKMODE

设置工作模式，设置普通模式，以及快速模式。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

Parameters

aux

工作模式：

Value	Description
DRV_TP_MODE_NORMAL	适用桌面及一般的应用，提供上/下/左/右的手势及速度的识别
DRV_TP_MODE_FAST	适用手写及快速输入的应用，提供更多的采样数据给上层，没有速度和手势的识别
DRV_TP_MODE_DUAL	适用图片或地图的放大缩小应用，单点或两点触摸均以 EV_TP_PRESS_START 消息表示按下，以 EV_TP_ACTION_NULL 消息表示抬起

pbuffer

无效，设为 0。

Return Value

EPDK_OK

EPDK_FAIL

Remarks

需要注意的是两点触摸时 start 消息(type 值为 0xf0)提供的 x，y 坐标不一定是两点间的中间点坐标值，可能是某个手指先按下的点坐标。

两点触摸中，消息的发送支持单点触摸到两点触摸的切换，即如果用户用一个手指按住触摸屏，随后将另一个手指按住触摸屏的另一个位置，此时会进入两点触摸识别，TP 驱动会根据两点间的距离向上层发送消息，通知是放大还是缩小，或者是保持，提供的消息内容包括：两点之前的中间点坐标值，放大或缩小的比例。

消息的发送不支持从两点触摸到单点触摸的切换，一旦一个动作认为是两点触摸，则在触摸抬起前不会切换到单点模式，防止误操作。

24.6. 数据结构

24.6.1. __ev_tp_msg_t

```
typedef struct __EV_TP_MSG
{
    __u8   msg_pattern;
    __u8   msg_type;
    __u16  xpoint;
    __u16  ypoint;
    __u16  speed_dir;
    __u16  speed_val;
} __ev_tp_msg_t;
```

Members

msg_pattern

消息种类，用于和输入设备兼容，用户暂时不关心，保留。

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

msg_type

消息类型，区分是按下的起点，滑动，按下不动，或者是抬起时候的手势。

xpoint

当前点的 X 坐标值。

ypoint

当前点的 Y 坐标值。

speed_dir

抬起时候移动的方向。

speed_val

抬起时候速度的大小，速度分级：0-6。

24.6.2. __ev_tp_pos_t

```
typedef struct __EV_TP_POS
{
    __u16  disp_xpos;
    __u16  disp_ypos;
} __ev_tp_pos_t;
```

Members

disp_xpos

屏幕对应的 x 坐标值。

disp_ypos

屏幕对应的 y 坐标值。

24.7. 示例

24.7.1. dual 模式

Melis 系统启动后，tp 默认为 normal 模式，此时不支持 dual 模式（即两点模式），桌面接收到的 tp 消息跟 melis 20 保持一致。

如果图片应用或者地图应用需要两点模式的支持，则进入到应用的主界面前要先调用设置 tp 为两点模式，代码如下：

```
ES_FILE *tp_hdl = NULL;
tp_hdl = eLIBs_fopen("b:\\INPUT\\TP", "r+");
if (!tp_hdl)
{
    eLIBs_printf("open tp handle fail\n");
}

ret = eLIBs_fioctrl(tp_hdl, DRV_TP_CMD_SET_WORKMODE, DRV_TP_MODE_DUAL, NULL);
if (ret == EPDK_FAIL)
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
{
    eLIBs_printf("set tp dual mode fail\n");
}

eLIBs_fclose(tp_hdl);
tp_hdl = NULL;
```

在两点模式下，用户单点触摸的消息机制跟 20 的保持一致，如果用户使用两点操作（放大/缩小/保持），则 tp 向上发送的两点消息为（EV_TP_PINCH_OUT/EV_TP_PINCH_IN/ EV_TP_PINCH_HOLD），其中，消息格式如下：

Type	Code	Value
EV_ABS	ABS_MISC	EV_TP_PINCH_OUT/ EV_TP_PINCH_IN/HOLD，放大/缩小/保持。应用依据这个消息来作处理
EV_ABS	ABS_X	两点间的中间点的 X 坐标
EV_ABS	ABS_Y	两点间的中间点的 Y 坐标
EV_ABS	ABS_RUDDER	保留
EV_ABS	ABS_BRAKE	Pinch in/pinch out 消息下为两点间的距离变化值，单位 pixel，应用依据这个值为作放大或缩小的比例
EV_SYN	SYN_REPORT	0

退出的时候要调用下面的代码，恢复到 normal 模式：

```
ES_FILE *tp_hdl = NULL;
tp_hdl = eLIBs_fopen("b:\\INPUT\\TP", "r+");
if (!tp_hdl)
{
    eLIBs_printf("open tp handle fail\n");
}

ret = eLIBs_fioctrl(tp_hdl, DRV_TP_CMD_SET_WORKMODE, DRV_TP_MODE_NORMAL, NULL);
if (ret == EPDK_FAIL)
{
    eLIBs_printf("set tp dual mode fail\n");
}

eLIBs_fclose(tp_hdl);
tp_hdl = NULL;
```

24.7.2. fast 模式

Fast 主要应用在手写场合，提供更多的采样数据给上层，触摸抬起时没有速度和方向值。

Melis 系统启动后，tp 默认为 normal 模式，此时不支持 fast 模式，桌面接收到的 tp 消息跟 melis 20 保持一致。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

如果手写输入应用需要 fast 模式的支持，则进入到应用的主界面前要先调用设置 tp 为 fast 模式，代码如下：

```
ES_FILE *tp_hdl = NULL;
tp_hdl = eLIBs_fopen("b:\\INPUT\\TP", "r+");
if (!tp_hdl)
{
    eLIBs_printf("open tp handle fail\n");
}

ret = eLIBs_fioctrl(tp_hdl, DRV_TP_CMD_SET_WORKMODE, DRV_TP_MODE_FAST, NULL);
if (ret == EPDK_FAIL)
{
    eLIBs_printf("set tp fast mode fail\n");
}

eLIBs_fclose(tp_hdl);
tp_hdl = NULL;
```

在 fast 模式下，用户单点触摸的消息机制跟 20 的保持一致，唯一不同的是提供更多的数据和抬起时没有速度和方向的识别。

退出的时候要调用下面的代码，恢复到 normal 模式：

```
ES_FILE *tp_hdl = NULL;
tp_hdl = eLIBs_fopen("b:\\INPUT\\TP", "r+");
if (!tp_hdl)
{
    eLIBs_printf("open tp handle fail\n");
}

ret = eLIBs_fioctrl(tp_hdl, DRV_TP_CMD_SET_WORKMODE, DRV_TP_MODE_NORMAL, NULL);
if (ret == EPDK_FAIL)
{
    eLIBs_printf("set tp dual mode fail\n");
}

eLIBs_fclose(tp_hdl);
tp_hdl = NULL;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25. Display Dirver

25.1. 挂载

```
esDEV_Plugin("\\drv\\display.drv", 0, 0, 1);
```

25.2. 卸载

```
esDEV_Plugout("\\drv\\display.drv", 0);
```

25.3. 访问

```
ES_FILE *fdisp = eLIBs_fopen("b:\\DISP\\DISPLAY", "r+");
```

25.4. 功能命令列表

命令	描述
DISP_CMD_SET_BKCOLOR	设置背景颜色
DISP_CMD_GET_BKCOLOR	没有实现
DISP_CMD_SET_COLORKEY	设置 color key
DISP_CMD_GET_COLORKEY	没有实现
DISP_CMD_SET_PALETTE_TBL	设置调色板
DISP_CMD_GET_PALETTE_TBL	获取调色板
DISP_CMD_SCN_GET_WIDTH	获取当前屏幕的宽度
DISP_CMD_SCN_GET_HEIGHT	获取当前屏幕的高度
DISP_CMD_GET_OUTPUT_TYPE	获取当前输出类型
DISP_CMD_SET_EXIT_MODE	没有实现
DISP_CMD_START_CMD_CACHE	启动 cache
DISP_CMD_EXECUTE_CMD_AND_STOP_CACHE	执行命令并停止 cache
DISP_CMD_SET_BRIGHT	设置当前屏幕的亮度
DISP_CMD_SET_CONTRAST	设置当前屏幕的对比度
DISP_CMD_SET_SATURATION	设置当前屏幕的饱和度
DISP_CMD_GET_BRIGHT	获取当前屏幕的亮度
DISP_CMD_GET_CONTRAST	获取当前屏幕的对比度
DISP_CMD_GET_SATURATION	获取当前屏幕的饱和度
DISP_CMD_ENHANCE_ON	打开当前屏幕的 enhance 功能
DISP_CMD_ENHANCE_OFF	关闭当前屏幕的 enhance 功能
DISP_CMD_GET_ENHANCE_EN	获取当前屏幕的 enhance 功能是否打

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

	开
DISP_CMD_CLK_ON	没有实现
DISP_CMD_CLK_OFF	没有实现
DISP_CMD_SET_DE_FLICKER	打开或者关闭 flicker 功能
DISP_CMD_LAYER_REQUEST	申请图层
DISP_CMD_LAYER_RELEASE	释放图层
DISP_CMD_LAYER_OPEN	打开图层
DISP_CMD_LAYER_CLOSE	关闭图层
DISP_CMD_LAYER_SET_FB	设置图层 frame buffer
DISP_CMD_LAYER_GET_FB	获取图层 frame buffer
DISP_CMD_LAYER_SET_SRC_WINDOW	设置图层 source window
DISP_CMD_LAYER_GET_SRC_WINDOW	获取图层 source window
DISP_CMD_LAYER_SET_SCN_WINDOW	设置图层 screen window
DISP_CMD_LAYER_GET_SCN_WINDOW	获取图层 screen window
DISP_CMD_LAYER_SET_PARA	设置图层参数
DISP_CMD_LAYER_GET_PARA	获取图层参数
DISP_CMD_LAYER_ALPHA_ON	打开图层的 global alpha 功能
DISP_CMD_LAYER_ALPHA_OFF	关闭图层的 global alpha 功能
DISP_CMD_LAYER_GET_ALPHA_EN	获取图层的 global alpha 功能是否打开
DISP_CMD_LAYER_SET_ALPHA_VALUE	设置图层的 global alpha value
DISP_CMD_LAYER_GET_ALPHA_VALUE	获取图层的 global alpha value
DISP_CMD_LAYER_CK_ON	打开图层的 color key 功能
DISP_CMD_LAYER_CK_OFF	关闭图层的 color key 功能
DISP_CMD_LAYER_GET_CK_EN	获取图层的 color key 功能是否打开
DISP_CMD_LAYER_SET_PIPE	设置图层的 pipe
DISP_CMD_LAYER_GET_PIPE	获取图层的 pipe
DISP_CMD_LAYER_TOP	将图层置顶
DISP_CMD_LAYER_BOTTOM	将图层置底
DISP_CMD_LAYER_GET_PRIO	获取图层的优先级
DISP_CMD_LAYER_SET_SMOOTH	设置图层的 smooth
DISP_CMD_LAYER_GET_SMOOTH	获取图层的 smooth
DISP_CMD_LAYER_SET_BRIGHT	设置 scaler 图层的亮度
DISP_CMD_LAYER_SET_CONTRAST	设置 scaler 图层的对比度
DISP_CMD_LAYER_SET_SATURATION	设置 scaler 图层的饱和度
DISP_CMD_LAYER_SET_HUE	设置 scaler 图层的色调/色度
DISP_CMD_LAYER_GET_BRIGHT	获取 scaler 图层的亮度
DISP_CMD_LAYER_GET_CONTRAST	获取 scaler 图层的对比度
DISP_CMD_LAYER_GET_SATURATION	获取 scaler 图层的饱和度
DISP_CMD_LAYER_GET_HUE	获取 scaler 图层的色调/色度
DISP_CMD_LAYER_ENHANCE_ON	打开 scaler 图层的 enhance 功能

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

DISP_CMD_LAYER_ENHANCE_OFF	关闭 scaler 图层的 enhance 功能
DISP_CMD_LAYER_GET_ENHANCE_EN	获取 scaler 图层的 enhance 功能是否打开
DISP_CMD_SCALER_REQUEST	申请 scaler 用来回写
DISP_CMD_SCALER_RELEASE	释放 scaler
DISP_CMD_SCALER_EXECUTE	执行 scaler
DISP_CMD_HWC_OPEN	打开硬件鼠标
DISP_CMD_HWC_CLOSE	关闭硬件鼠标
DISP_CMD_HWC_SET_POS	设置硬件鼠标在屏幕中的起始坐标
DISP_CMD_HWC_GET_POS	获取硬件鼠标在屏幕中的起始坐标
DISP_CMD_HWC_SET_FB	设置鼠标的 frame buffer 信息
DISP_CMD_HWC_SET_PALETTE_TABLE	设置鼠标的调色板
DISP_CMD_VIDEO_START	开始 video 播放
DISP_CMD_VIDEO_STOP	停止 video 播放
DISP_CMD_VIDEO_SET_FB	设置 video frame buffer 信息
DISP_CMD_VIDEO_GET_FRAME_ID	获取当前正在显示的 frame buffer id
DISP_CMD_VIDEO_GET_DIT_INFO	没有实现
DISP_CMD_LCD_ON	打开 LCD
DISP_CMD_LCD_OFF	关闭 LCD
DISP_CMD_LCD_SET_BRIGHTNESS	设置 LCD 的亮度
DISP_CMD_LCD_GET_BRIGHTNESS	获取 LCD 的亮度
DISP_CMD_LCD_SET_COLOR	设置 LCD 的颜色
DISP_CMD_LCD_GET_COLOR	获取 LCD 的颜色
DISP_CMD_LCD_CPUIF_XY_SWITCH	没有实现
DISP_CMD_LCD_CHECK_OPEN_FINISH	没有实现
DISP_CMD_LCD_CHECK_CLOSE_FINISH	没有实现
DISP_CMD_LCD_SET_SRC	设置 LCD 的 source
DISP_CMD_TV_ON	打开 TV 显示
DISP_CMD_TV_OFF	关闭 TV 显示
DISP_CMD_TV_SET_MODE	设置 TV 的模式
DISP_CMD_TV_GET_MODE	获取当前 TV 的模式
DISP_CMD_TV_AUTOCHECK_ON	打开 TV 的自动检测功能
DISP_CMD_TV_AUTOCHECK_OFF	关闭 TV 的自动检测功能
DISP_CMD_TV_GET_INTERFACE	获取 TV 的 interface
DISP_CMD_TV_SET_SRC	设置 TV 的 source
DISP_CMD_TV_GET_DAC_STATUS	获取 DAC 的连接状态
DISP_CMD_TV_SET_DAC_SOURCE	设置 DAC 的 source
DISP_CMD_TV_GET_DAC_SOURCE	获取 DAC 的 source

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5. 功能命令描述

在 Display Dirver 中，pbuffer 必须是大小为 3 的__u32 数组的首地址。

25.5.1. DISP_CMD_SET_BKCOLOR

设置背景颜色。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	指向__disp_color_t 结构体的指针。
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.2. DISP_CMD_GET_BKCOLOR

没有实现。

25.5.3. DISP_CMD_SET_COLORKEY

设置 color key。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	指向__disp_colorkey_t 结构体的指针。
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.4. DISP_CMD_GET_COLORKEY

没有实现。

25.5.5. DISP_CMD_SET_PALETTE_TBL

设置调色板，系统调色版的总大小是 256*4 byte，可以设置调色版中任意一段连续区域。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	地址
pbuffer[1]	偏移，__u32，DW 对齐
pbuffer[2]	大小，DW 对齐，单位是 byte

Return Value

__s32

成功：0；失败：失败号。

25.5.6. DISP_CMD_GET_PALETTE_TBL

获取调色板。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	地址
pbuffer[1]	偏移，__u32，DW 对齐
pbuffer[2]	大小，DW 对齐，单位是 byte

Return Value

__s32

成功：0；失败：失败号。

25.5.7. DISP_CMD_SCN_GET_WIDTH

获取当前屏幕的宽度。

Parameters

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

屏幕的宽度。

25.5.8. DISP_CMD_SCN_GET_HEIGHT

获取当前屏幕的高度。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

屏幕的高度。

25.5.9. DISP_CMD_GET_OUTPUT_TYPE

获取当前输出类型（LCD/TV）。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

当前输出类型（LCD/TV）。

25.5.10. DISP_CMD_SET_EXIT_MODE

没有实现。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.11. DISP_CMD_START_CMD_CACHE

启动 cache，后面的 IO 命令将不马上执行，而等到调用 DISP_CMD_EXECUTE_CMD_AND_STOP_CACHE 后才一并执行。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

成功：0；失败：失败号。

25.5.12. DISP_CMD_EXECUTE_CMD_AND_STOP_CACHE

执行命令并停止 cache。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

成功：0；失败：失败号。

25.5.13. DISP_CMD_SET_BRIGHT

设置当前屏幕的亮度。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	__u32，屏幕的亮度。
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.14. DISP_CMD_SET_CONTRAST

设置当前屏幕的对比度。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	__u32，屏幕的对比度。
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.15. DISP_CMD_SET_SATURATION

设置当前屏幕的饱和度。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	__u32，屏幕的饱和度。
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.16. DISP_CMD_GET_BRIGHT

获取当前屏幕的亮度。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

Return Value

__s32

当前屏幕的亮度。

25.5.17. DISP_CMD_GET_CONTRAST

获取当前屏幕的对比度。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

当前屏幕的对比度。

25.5.18. DISP_CMD_GET_SATURATION

获取当前屏幕的饱和度。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

当前屏幕的饱和度。

25.5.19. DISP_CMD_ENHANCE_ON

打开当前屏幕的 enhance 功能。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

成功：0；失败：失败号。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.20. DISP_CMD_ENHANCE_OFF

关闭当前屏幕的 enhance 功能。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

成功：0；失败：失败号。

25.5.21. DISP_CMD_GET_ENHANCE_EN

获取当前屏幕的 enhance 功能是否打开。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

成功：是否打开，0 表示关闭，1 表示打开；失败：失败号。

25.5.22. DISP_CMD_CLK_ON

没有实现。

25.5.23. DISP_CMD_CLK_OFF

没有实现。

25.5.24. DISP_CMD_SET_DE_FLICKER

打开或者关闭 flicker 功能。

Parameters

aux

无效，设为 0。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

pbuffer

Value	Description
pbuffer[0]	__u32，0 表示关闭，1 表示打开。
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.25. DISP_CMD_LAYER_REQUEST

申请图层，在驱动中只是分配一个图层出来，没有对该图层作任何设置，故推荐在申请图层后随即调用 DISP_CMD_LAYER_SET_PARA 命令对该图层的参数进行设置。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层工作模式
pbuffer[1]	0
pbuffer[2]	0

图层工作模式：

Value	Description
DISP_LAYER_WORK_MODE_NORMAL	normal work mode
DISP_LAYER_WORK_MODE_PALETTE	palette work mode
DISP_LAYER_WORK_MODE_INTER_BUF	internal frame buffer work mode
DISP_LAYER_WORK_MODE_GAMMA	gamma correction work mode
DISP_LAYER_WORK_MODE_SCALER	scaler work mode

Return Value

__s32

成功：图层的句柄；失败：NULL。

25.5.26. DISP_CMD_LAYER_RELEASE

释放图层，该图层释放后，所有该图层相关的寄存器都被清为默认值，故如果想重新想使用该图层必须重新申请。

Parameters

aux

无效，设为 0。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.27. DISP_CMD_LAYER_OPEN

打开图层，即能看到该图层显示。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.28. DISP_CMD_LAYER_CLOSE

关闭图层。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

25.5.29. DISP_CMD_LAYER_SET_FB

设置图层 frame buffer，仍然保持原来的 source window 设置。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	frame buffer 信息，指向__disp_fb_t 结构体的指针
pbuffer[2]	0

指向一个__u32 数组。该数组有 3 个元素，分别图层句柄，

Return Value

__s32

成功：0；失败：失败号。

25.5.30. DISP_CMD_LAYER_GET_FB

获取图层 frame buffer。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	frame buffer 信息，指向__disp_fb_t 结构体的指针
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.31. DISP_CMD_LAYER_SET_SRC_WINDOW

设置图层 source window。

Parameters

aux

无效，设为 0。

pbuffer

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	指向__disp_rect_t 结构体的指针
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.32. DISP_CMD_LAYER_GET_SRC_WINDOW

获取图层 source window。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	指向__disp_rect_t 结构体的指针
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.33. DISP_CMD_LAYER_SET_SCN_WINDOW

设置图层 screen window。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	指向__disp_rect_t 结构体的指针
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.34. DISP_CMD_LAYER_GET_SCN_WINDOW

获取图层 screen window。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	指向__disp_rect_t 结构体的指针
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.35. DISP_CMD_LAYER_SET_PARA

设置图层参数，图层的所有参数都可以通过该命令进行设置，除了优先级，优先级的调整只能通过 DISP_CMD_LAYER_TOP / DISP_CMD_LAYER_BOTTOM 实现。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	图层的信息，指向__disp_layer_info_t 结构体的指针
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.36. DISP_CMD_LAYER_GET_PARA

获取图层参数。

Parameters

aux

无效，设为 0。

pbuffer

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	图层的信息，指向__disp_layer_info_t 结构体的指针
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.37. DISP_CMD_LAYER_ALPHA_ON

打开图层的 global alpha 功能，即使用面 alpha。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.38. DISP_CMD_LAYER_ALPHA_OFF

关闭图层的 global alpha 功能，即使用点 alpha。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.39. DISP_CMD_LAYER_GET_ALPHA_EN

获取图层的 global alpha 功能是否打开。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

0 表示关闭，1 表示打开。

25.5.40. DISP_CMD_LAYER_SET_ALPHA_VALUE

设置图层的 global alpha value。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	global alpha value.
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.41. DISP_CMD_LAYER_GET_ALPHA_VALUE

获取图层的 global alpha value。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

global alpha value.

25.5.42. DISP_CMD_LAYER_CK_ON

打开图层的 color key 功能。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.43. DISP_CMD_LAYER_CK_OFF

关闭图层的 color key 功能。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.44. DISP_CMD_LAYER_GET_CK_EN

获取图层的 color key 功能是否打开。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

0 表示关闭，1 表示打开。

25.5.45. DISP_CMD_LAYER_SET_PIPE

设置图层的 pipe。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	__u32，图层的 pipe
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.46. DISP_CMD_LAYER_GET_PIPE

获取图层的 pipe。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

图层的 pipe。

25.5.47. DISP_CMD_LAYER_TOP

将图层置顶。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.48. DISP_CMD_LAYER_BOTTOM

将图层置底。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.49. DISP_CMD_LAYER_GET_PRIO

获取图层的优先级。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

图层的优先级。

25.5.50. DISP_CMD_LAYER_SET_SMOOTH

设置图层的 smooth，该图层必须为 scaler 图层。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	图层的 smooth
pbuffer[2]	0

图层的 smooth：

Value	Description
DISP_VIDEO_NATUAL	
DISP_VIDEO_SOFT	
DISP_VIDEO_VERYSOFT	
DISP_VIDEO_SHARP	
DISP_VIDEO_VERYSHARP	

Return Value

__s32

成功：0；失败：失败号。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.51. DISP_CMD_LAYER_GET_SMOOTH

获取图层的 smooth，该图层必须为 scaler 图层。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

图层的 smooth。

25.5.52. DISP_CMD_LAYER_SET_BRIGHT

设置 scaler 图层的亮度。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	亮度值（0~63，默认 32）
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.53. DISP_CMD_LAYER_SET_CONTRAST

设置 scaler 图层的对比度。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

pbuffer[0]	图层的句柄
pbuffer[1]	对比度（0~63，默认 32）
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.54. DISP_CMD_LAYER_SET_SATURATION

设置 scaler 图层的饱和度。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	饱和度（0~63，默认 32）
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.55. DISP_CMD_LAYER_SET_HUE

设置 scaler 图层的色调/色度。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	色调/色度值（0~63，默认 32）
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.56. DISP_CMD_LAYER_GET_BRIGHT

获取 scaler 图层的亮度。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

scaler 图层的亮度。

25.5.57. DISP_CMD_LAYER_GET_CONTRAST

获取 scaler 图层的对比度。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

scaler 图层的对比度。

25.5.58. DISP_CMD_LAYER_GET_SATURATION

获取 scaler 图层的饱和度。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

scaler 图层的饱和度。

25.5.59. DISP_CMD_LAYER_GET_HUE

获取 scaler 图层的色调/色度。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

scaler 图层的色调/色度。

25.5.60. DISP_CMD_LAYER_ENHANCE_ON

打开 scaler 图层的 enhance 功能。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.61. DISP_CMD_LAYER_ENHANCE_OFF

关闭 scaler 图层的 enhance 功能。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.62. DISP_CMD_LAYER_GET_ENHANCE_EN

获取 scaler 图层的 enhance 功能是否打开。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

0 表示 enhance 关闭，1 表示 enhance 打开。

25.5.63. DISP_CMD_SCALER_REQUEST

申请 scaler 用来回写。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

Return Value

__s32

scaler 句柄，失败返回 NULL。

25.5.64. DISP_CMD_SCALER_RELEASE

释放 scaler。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	scaler 句柄。
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.65. DISP_CMD_SCALER_EXECUTE

执行 scaler。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	scaler 句柄。
pbuffer[1]	指向__disp_scaler_para_t 结构体的指针。
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.66. DISP_CMD_HWC_OPEN

打开硬件鼠标。

Parameters

aux

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

成功：0；失败：失败号。

25.5.67. DISP_CMD_HWC_CLOSE

关闭硬件鼠标。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

成功：0；失败：失败号。

25.5.68. DISP_CMD_HWC_SET_POS

设置硬件鼠标在屏幕中的起始坐标。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	指向__disp_pos_t 结构体的指针，表示硬件鼠标在屏幕中的起始坐标。
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.69. DISP_CMD_HWC_GET_POS

获取硬件鼠标在屏幕中的起始坐标。

Parameters

aux

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	指向__disp_pos_t 结构体的指针，表示硬件鼠标在屏幕中的起始坐标。
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.70. DISP_CMD_HWC_SET_FB

设置鼠标的 frame buffer 信息。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	指示__disp_hwc_pattern_t 结构体的指针，表示硬件鼠标的信息。
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.71. DISP_CMD_HWC_SET_PALETTE_TABLE

设置鼠标的调色板。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	地址
pbuffer[1]	偏移，__u32，DW 对齐
pbuffer[2]	大小，DW 对齐，单位是 byte

Return Value

__s32

成功：0；失败：失败号。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.72. DISP_CMD_VIDEO_START

开始 video 播放。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.73. DISP_CMD_VIDEO_STOP

停止 video 播放。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.74. DISP_CMD_VIDEO_SET_FB

设置 video frame buffer 信息。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

pbuffer[0]	图层的句柄
pbuffer[1]	指向__disp_video_fb_t 结构体的指针。
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.75. DISP_CMD_VIDEO_GET_FRAME_ID

获取当前正在显示的 frame buffer id。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	图层的句柄
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：frame buffer id；失败：失败号。

25.5.76. DISP_CMD_VIDEO_GET_DIT_INFO

没有实现。

25.5.77. DISP_CMD_LCD_ON

打开 LCD。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

成功：0；失败：失败号。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.78. DISP_CMD_LCD_OFF

关闭 LCD。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

成功：0；失败：失败号。

25.5.79. DISP_CMD_LCD_SET_BRIGHTNESS

设置 LCD 的亮度。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	LCD 的亮度（DISP_LCD_BRIGHT_LEVEL0~DISP_LCD_BRIGHT_LEVEL15）
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

成功：0；失败：失败号。

25.5.80. DISP_CMD_LCD_GET_BRIGHTNESS

获取 LCD 的亮度。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

LCD 的亮度（DISP_LCD_BRIGHT_LEVEL0~DISP_LCD_BRIGHT_LEVEL15）。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.81. DISP_CMD_LCD_SET_COLOR

没有实现。

25.5.82. DISP_CMD_LCD_GET_COLOR

没有实现。

25.5.83. DISP_CMD_LCD_CPUIF_XY_SWITCH

没有实现。

25.5.84. DISP_CMD_LCD_CHECK_OPEN_FINISH

没有实现。

25.5.85. DISP_CMD_LCD_CHECK_CLOSE_FINISH

没有实现。

25.5.86. DISP_CMD_LCD_SET_SRC

设置 LCD 的 source。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	LCD 的 source
pbuffer[1]	0
pbuffer[2]	0

LCD 的 source：

Value	Description
DISP_LCDC_SRC_DE_CH1	
DISP_LCDC_SRC_DE_CH2	
DISP_LCDC_SRC_DMA	
DISP_LCDC_SRC_WHITE	
DISP_LCDC_SRC_BLACK	

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

DISP_LCDC_SRC_BLUT	

Return Value

__s32

成功：0；失败：失败号。

25.5.87. DISP_CMD_TV_ON

打开 TV 显示。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

成功：0；失败：失败号。

25.5.88. DISP_CMD_TV_OFF

关闭 TV 显示。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

成功：0；失败：失败号。

25.5.89. DISP_CMD_TV_SET_MODE

设置 TV 的模式。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	TV 的模式
pbuffer[1]	0

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

pbuffer[2]	0

TV 的模式：

Value	Description
DISP_TV_MOD_480I	
DISP_TV_MOD_576I	
DISP_TV_MOD_480P	
DISP_TV_MOD_576P	
DISP_TV_MOD_720P_50HZ	
DISP_TV_MOD_720P_60HZ	
DISP_TV_MOD_1080I_50HZ	
DISP_TV_MOD_1080I_60HZ	
DISP_TV_MOD_1080P_24HZ	
DISP_TV_MOD_1080P_50HZ	
DISP_TV_MOD_1080P_60HZ	
DISP_TV_MOD_PAL	
DISP_TV_MOD_PAL_SVIDEO	
DISP_TV_MOD_PAL_CVBS_SVIDEO	
DISP_TV_MOD_NTSC	
DISP_TV_MOD_NTSC_SVIDEO	
DISP_TV_MOD_NTSC_CVBS_SVIDEO	
DISP_TV_MOD_PAL_M	
DISP_TV_MOD_PAL_M_SVIDEO	
DISP_TV_MOD_PAL_M_CVBS_SVIDEO	
DISP_TV_MOD_PAL_NC	
DISP_TV_MOD_PAL_NC_SVIDEO	
DISP_TV_MOD_PAL_NC_CVBS_SVIDEO	

Return Value

__s32

成功：0；失败：失败号。

25.5.90. DISP_CMD_TV_GET_MODE

获取当前 TV 的模式。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

当前 TV 的模式。

25.5.91. DISP_CMD_TV_AUTOCHECK_ON

打开 TV 的自动检测功能。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

成功：0；失败：失败号。

25.5.92. DISP_CMD_TV_AUTOCHECK_OFF

关闭 TV 的自动检测功能。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

成功：0；失败：失败号。

25.5.93. DISP_CMD_TV_GET_INTERFACE

获取 TV 的 interface。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

TV 的 interface。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.94. DISP_CMD_TV_SET_SRC

设置 TV 的 source。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	TV 的 source
pbuffer[1]	0
pbuffer[2]	0

TV 的 source：

Value	Description
DISP_LCDC_SRC_DE_CH1	
DISP_LCDC_SRC_DE_CH2	
DISP_LCDC_SRC_DMA	
DISP_LCDC_SRC_WHITE	
DISP_LCDC_SRC_BLACK	
DISP_LCDC_SRC_BLUT	

Return Value

__s32

成功：0；失败：失败号。

25.5.95. DISP_CMD_TV_GET_DAC_STATUS

获取 DAC 的连接状态。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	DAC 的连接状态。
pbuffer[1]	0
pbuffer[2]	0

Return Value

__s32

0：unconnected; 1:connected; 3:short to ground

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.5.96. DISP_CMD_TV_SET_DAC_SOURCE

设置 DAC 的 source。

Parameters

aux

无效，设为 0。

pbuffer

Value	Description
pbuffer[0]	DAC 的 source
pbuffer[1]	0
pbuffer[2]	0

DAC 的 source：

Value	Description
DISP_TV_DAC_SRC_COMPOSITE	
DISP_TV_DAC_SRC_LUMA	
DISP_TV_DAC_SRC_CHROMA	
DISP_TV_DAC_SRC_Y	
DISP_TV_DAC_SRC_PB	
DISP_TV_DAC_SRC_PR	
DISP_TV_DAC_SRC_NONE	

Return Value

__s32

成功：0；失败：失败号。

25.5.97. DISP_CMD_TV_GET_DAC_SOURCE

获取 DAC 的 source。

Parameters

aux

无效，设为 0。

pbuffer

无效，设为 0。

Return Value

__s32

DAC 的 source。

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

25.6. 数据结构

25.6.1. __disp_color_t

```
typedef struct
{
    __u8 alpha;
    __u8 red;
    __u8 green;
    __u8 blue;
} __disp_color_t;
```

Members

alpha

red

green

blue

25.6.2. __disp_rect_t

```
typedef struct
{
    __s32 x;
    __s32 y;
    __u32 width;
    __u32 height;
} __disp_rect_t;
```

Members

x

x 轴坐标。

y

y 轴坐标。

width

矩形区域的宽。

height

矩形区域的高。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

25.6.3. __disp_rectsz_t

```
typedef struct
{
    __u32 width;
    __u32 height;
} __disp_rectsz_t;
```

Members

width

宽。

height

高。

25.6.4. __disp_pos_t

```
typedef struct
{
    __s32 x;
    __s32 y;
} __disp_pos_t;
```

Members

x

x 坐标。

y

y 坐标。

25.6.5. __disp_fb_t

```
typedef struct
{
    __u32               addr[3];
    __disp_rectsz_t     size;
    __disp_pixel_fmt_t format;
    __disp_pixel_seq_t seq;
    __disp_pixel_mod_t mode;
    __bool             br_swap;
    __disp_cs_mode_t   cs_mode;
} __disp_fb_t;
```

Members

addr

frame buffer 的内容地址，对于 rgb 类型，只有 addr[0]有效。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

size

frame buffer 的大小，单位是 pixel。

format

pixel format.

Value	Description
DISP_FORMAT_1BPP	
DISP_FORMAT_2BPP	
DISP_FORMAT_4BPP	
DISP_FORMAT_8BPP	
DISP_FORMAT_RGB655	
DISP_FORMAT_RGB565	
DISP_FORMAT_RGB556	
DISP_FORMAT_ARGB1555	
DISP_FORMAT_RGBA5551	
DISP_FORMAT_RGB888	
DISP_FORMAT_ARGB8888	
DISP_FORMAT_YUV444	
DISP_FORMAT_YUV422	
DISP_FORMAT_YUV420	
DISP_FORMAT_YUV411	
DISP_FORMAT_CSIRGB	

seq

pixel sequence.

Value	Description
DISP_SEQ_ARGB	for interleave argb8888
DISP_SEQ_BGRA	for interleave argb8888
DISP_SEQ_UYVY	for nterleaved yuv422
DISP_SEQ_YUYV	for nterleaved yuv422
DISP_SEQ_VYUY	for nterleaved yuv422
DISP_SEQ_YVYU	for nterleaved yuv422
DISP_SEQ_AYUV	for interleaved yuv444
DISP_SEQ_VUYA	for interleaved yuv444
DISP_SEQ_UVUV	for uv_combined yuv420
DISP_SEQ_VUVU	for uv_combined yuv420
DISP_SEQ_P10	for 16bpp rgb
DISP_SEQ_P01	for 16bpp rgb
DISP_SEQ_P3210	for planar format or 8bpp rgb
DISP_SEQ_P0123	for planar format or 8bpp rgb
DISP_SEQ_P76543210	for 4bpp rgb
DISP_SEQ_P67452301	for 4bpp rgb
DISP_SEQ_P10325476	for 4bpp rgb

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

DISP_SEQ_P01234567	for 4bpp rgb
DISP_SEQ_2BPP_BIG_BIG	for 2bpp rgb
DISP_SEQ_2BPP_BIG_LITTER	for 2bpp rgb
DISP_SEQ_2BPP_LITTER_BIG	for 2bpp rgb
DISP_SEQ_2BPP_LITTER_LITTER	for 2bpp rgb
DISP_SEQ_1BPP_BIG_BIG	for 1bpp rgb
DISP_SEQ_1BPP_BIG_LITTER	for 1bpp rgb
DISP_SEQ_1BPP_LITTER_BIG	for 1bpp rgb
DISP_SEQ_1BPP_LITTER_LITTER	for 1bpp rgb

mode

pixel mode.

Value	Description
DISP_MOD_INTERLEAVED	interleaved，1 个地址
DISP_MOD_NON_MB_PLANAR	无宏块平面模式，3 个地址，RGB/YUV 每个 channel 分别存放
DISP_MOD_NON_MB_UV_COMBINED	无宏块 UV 打包模式，2 个地址，Y 和 UV 分别存放
DISP_MOD_MB_PLANAR	宏块平面模式，3 个地址，RGB/YUV 每个 channel 分别存放
DISP_MOD_MB_UV_COMBINED	宏块 UV 打包模式，2 个地址，Y 和 UV 分别存放

br_swap

blue red color swap flag, FALSE: RGB; TRUE: BGR, only used in rgb format.

cs_mode

color space.

25.6.6. __disp_layer_info_t

```
typedef struct
{
    __disp_layer_work_mode_t mode;
    __u8                     pipe;
    __u8                     prio;
    __bool                  alpha_en;
    __u16                   alpha_val;
    __bool                  ck_enable;
    __disp_rect_t            src_win;
    __disp_rect_t            scn_win;
    __disp_fb_t             fb;
} __disp_layer_info_t;
```

Members

mode

layer work mode.

pipe

layer pipe, 0/1, if in scaler mode, scaler0 must be pipe0, scaler1 must be pipe1.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

prio

layer priority, can get layer prio, but never set layer prio，从顶至顶，优先级由低至高.

alpha_en

layer global alpha enable.

alpha_val

layer global alpha value.

ck_enable

layer color key enable.

src_win

framebuffer source window, only care x, y if is not scaler mode.

scn_win

screen window.

fb

frame buffer.

25.6.7. __disp_colorkey_t

```
typedef struct
{
    __disp_color_t ck_max;
    __disp_color_t ck_min;
    __u32          red_match_rule;
    __u32          green_match_rule;
    __u32          blue_match_rule;
} __disp_colorkey_t;
```

Members

ck_max

ck_min

red_match_rule

green_match_rule

blue_match_rule

25.6.8. __disp_video_fb_t

```
typedef struct
{
    __s32   id;
    __u32   addr[3];
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
    __bool  interlace;
    __bool  top_field_first;
    __u32   frame_rate;
    __u32   flag_addr;
    __u32   flag_stride;
    __bool  maf_valid;
    __bool  pre_frame_valid;
} __disp_video_fb_t;
```

Members

id

addr

interlace

top_field_first

frame_rate

现在定为 1000。

flag_addr

dit maf flag address.

flag_stride

dit maf flag line stride.

maf_valid

pre_frame_valid

25.6.9. __disp_scaler_para_t

```
typedef struct
{
    __disp_fb_t     input_fb;
    __disp_rect_t  source_regn;
    __disp_fb_t     output_fb;
} __disp_scaler_para_t;
```

Members

input_fb

source_regn

output_fb

25.7. 示例

25.7.1. Layer Demo

```
static __hdle gbinfile0, gbinfile1;

__s32 layer_demo_0(ES_FILE *disphd, __u32 sel)
{
    __disp_layer_info_t layer_para;
    __u32 *mem_in = NULL;
    __hdle hlay[4] = {NULL};
    __s32 j = 0;
    __u32 arg[3];

    mem_in = esMEMS_Balloc(800*480*4);
    eLIBs_memset(mem_in,0,800*480*4);
    esFSYS_fread(mem_in,800*480*4,1,gbinfile0);

    layer_para.fb.addr[0]       = (__u32)mem_in;
    layer_para.fb.size.width    = 800;
    layer_para.fb.mode          = DISP_MOD_INTERLEAVED;
    layer_para.fb.format        = DISP_FORMAT_ARGB8888;
    layer_para.fb.br_swap       = 0;
    layer_para.fb.seq           = DISP_SEQ_ARGB;
    layer_para.ck_enable        = 0;
    layer_para.alpha_en         = 1;
    layer_para.alpha_val        = 0xff;
    layer_para.pipe             = 0;
    layer_para.src_win.x        = 0;
    layer_para.src_win.y        = 0;
    layer_para.src_win.width    = 800;
    layer_para.src_win.height   = 480;
    layer_para.scn_win.x        = 0;
    layer_para.scn_win.y        = 0;
    layer_para.scn_win.width    = 800;
    layer_para.scn_win.height   = 480;

    arg[0] = DISP_LAYER_WORK_MODE_SCALER;
    arg[1] = 0;
    arg[2] = 0;
    hlay[0] = eLIBs_fioctrl(disphd, DISP_CMD_LAYER_REQUEST, sel, (void*)arg);

    layer_para.mode = DISP_LAYER_WORK_MODE_SCALER;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
layer_para.pipe = 0;
arg[0] = hlay[0];
arg[1] = (__u32)&layer_para;
arg[2] = 0;
eLIBs_fioctrl(disphd,DISP_CMD_LAYER_SET_PARA,sel,(void*)arg);

arg[0] = hlay[0];
arg[1] = 0;
arg[2] = 0;
eLIBs_fioctrl(disphd,DISP_CMD_LAYER_OPEN,sel,(void*)arg);

mem_in = esMEMS_Balloc(800*480*4);
eLIBs_memset(mem_in,0x00,800*480*4);
for(j=0;j<800*120;j++)
{
    *(mem_in + j) = 0xffff0000;//red
    *(mem_in + 800*120 +j) = 0xff00ff00;//green
    *(mem_in + 800*120*2 +j) = 0xffffffff;//white
    *(mem_in + 800*120*3 +j) = 0xff0000ff;//blue
}

arg[0] = DISP_LAYER_WORK_MODE_NORMAL;
arg[1] = 0;
arg[2] = 0;
hlay[1] = eLIBs_fioctrl(disphd, DISP_CMD_LAYER_REQUEST, sel, (void*)arg);

layer_para.mode = DISP_LAYER_WORK_MODE_NORMAL;
layer_para.fb.addr[0]       = (__u32)mem_in; //modify
layer_para.fb.size.width    = 800;
layer_para.fb.mode         = DISP_MOD_INTERLEAVED;
layer_para.fb.format       = DISP_FORMAT_ARGB8888;
layer_para.fb.br_swap      = 0;
layer_para.fb.seq          = DISP_SEQ_ARGB;
layer_para.ck_enable       = 0;
layer_para.alpha_en        = 1;
layer_para.alpha_val       = 0xff;
layer_para.pipe            = 0;
layer_para.src_win.x       = 0;
layer_para.src_win.y       = 0;
layer_para.src_win.width    = 800;
layer_para.src_win.height   = 480;
layer_para.scn_win.x       = 0;//modify
layer_para.scn_win.y       = 0;//modify
layer_para.scn_win.width    = 800;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
layer_para.scn_win.height   = 480;
arg[0] = hlay[1];
arg[1] = (__u32)&layer_para;
arg[2] = 0;
eLIBs_fioctrl(disphd,DISP_CMD_LAYER_SET_PARA,sel,(void*)arg);

arg[0] = hlay[1];
arg[1] = 0;
arg[2] = 0;
eLIBs_fioctrl(disphd,DISP_CMD_LAYER_OPEN,sel,(void*)arg);

mem_in = esMEMS_Balloc(800*480*4);
eLIBs_memset(mem_in + 800*60*0,0xa0,800*240);// Y
eLIBs_memset(mem_in + 800*60*1,0x30,800*240);
eLIBs_memset(mem_in + 800*60*2,0xc0,800*240);// CB
eLIBs_memset(mem_in + 800*60*3,0x70,800*240);
eLIBs_memset(mem_in + 800*60*4,0xa0,800*240);// CR
eLIBs_memset(mem_in + 800*60*5,0xf0,800*240);

layer_para.fb.addr[0]       = (__u32)(mem_in + 800*120*0);
layer_para.fb.addr[1]       = (__u32)(mem_in + 800*120*1);
layer_para.fb.addr[2]       = (__u32)(mem_in + 800*120*2);
layer_para.fb.size.width    = 800;
layer_para.fb.size.height   = 480;
layer_para.fb.mode          = DISP_MOD_NON_MB_PLANAR;
layer_para.fb.format        = DISP_FORMAT_YUV444;
layer_para.fb.br_swap       = 0;
layer_para.fb.seq           = DISP_SEQ_P3210;
layer_para.ck_enable        = 0;
layer_para.alpha_en         = 1;
layer_para.alpha_val        = 0xff;
layer_para.pipe             = 0;
layer_para.src_win.x        = 0;
layer_para.src_win.y        = 0;
layer_para.src_win.width    = 800;
layer_para.src_win.height   = 480;
layer_para.scn_win.x        = 0;
layer_para.scn_win.y        = 0;
layer_para.scn_win.width    = 800;
layer_para.scn_win.height   = 480;

arg[0] = DISP_LAYER_WORK_MODE_NORMAL;
arg[1] = 0;
arg[2] = 0;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
hlay[2] = eLIBs_fioctrl(disphd, DISP_CMD_LAYER_REQUEST, sel, (void*)arg);

layer_para.mode = DISP_LAYER_WORK_MODE_NORMAL;
layer_para.pipe = 1;
arg[0] = hlay[2];
arg[1] = (__u32)&layer_para;
arg[2] = 0;
eLIBs_fioctrl(disphd,DISP_CMD_LAYER_SET_PARA,sel,(void*)arg);

arg[0] = hlay[2];
arg[1] = 0;
arg[2] = 0;
eLIBs_fioctrl(disphd,DISP_CMD_LAYER_OPEN,0,(void*)arg);

mem_in = esMEMS_Balloc(800*480*4);
esFSYS_fread(mem_in,800*480*4,1,gbinfile1);
arg[0] = DISP_LAYER_WORK_MODE_NORMAL;
arg[1] = 0;
arg[2] = 0;
hlay[3] = eLIBs_fioctrl(disphd, DISP_CMD_LAYER_REQUEST, sel, (void*)arg);

layer_para.mode = DISP_LAYER_WORK_MODE_NORMAL;
layer_para.fb.addr[0]       = (__u32)mem_in; //modify
layer_para.fb.size.width    = 800;
layer_para.fb.mode          = DISP_MOD_INTERLEAVED;
layer_para.fb.format        = DISP_FORMAT_ARGB8888;
layer_para.fb.br_swap       = 0;
layer_para.fb.seq           = DISP_SEQ_ARGB;
layer_para.ck_enable        = 0;
layer_para.alpha_en         = 1;
layer_para.alpha_val        = 0xff;
layer_para.pipe             = 0;
layer_para.src_win.x        = 0;
layer_para.src_win.y        = 0;
layer_para.src_win.width    = 800;
layer_para.src_win.height   = 480;
layer_para.scn_win.x        = 0;//modify
layer_para.scn_win.y        = 0;//modify
layer_para.scn_win.width    = 800;
layer_para.scn_win.height   = 480;
arg[0] = hlay[3];
arg[1] = (__u32)&layer_para;
arg[2] = 0;
eLIBs_fioctrl(disphd,DISP_CMD_LAYER_SET_PARA,sel,(void*)arg);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
    arg[0] = hlay[3];
    arg[1] = 0;
    arg[2] = 0;
    eLIBs_fioctrl(disphd,DISP_CMD_LAYER_OPEN,sel,(void*)arg);

    return 0;
}
```

25.7.2. Scaler Demo

```
static __hdle gbinfile0, gbinfile1;

__s32 scaler_demo(ES_FILE *disphd, __u32 sel)
{
    __u32 arg[3];
    __u32 scaler_hdl;
    __disp_scaler_para_t para;
    __u32 *mem_in, *mem_out0, *mem_out1;
    __u32 j = 0;
    __disp_layer_info_t layer_para;
    __hdle hlay = NULL;
    __u32 width = 800;
    __u32 height = 480;

    mem_in = esMEMS_Balloc(width*height*4);
    mem_out0 = esMEMS_Balloc(width*height*4);
    mem_out1 = esMEMS_Balloc(width*height*4);
    eLIBs_memset(mem_in,0x00,width*height*4);
    eLIBs_memset(mem_out0,0x00,width*height*4);
    eLIBs_memset(mem_out1,0x00,width*height*4);

    for(j=0;j<width*height/4;j++)
    {
        *(mem_in + j) = 0xffff0000;//red
        *(mem_in + width*height/4 +j) = 0xff00ff00;//green
        *(mem_in + width*height/4*2 +j) = 0xffffffff;//white
        *(mem_in + width*height/4*3 +j) = 0xff0000ff;//blue
    }

    esFSYS_fread(mem_in,800*480*4,1,gbinfile1);

    scaler_hdl = eLIBs_fioctrl(disphd, DISP_CMD_SCALER_REQUEST, sel,0);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
para.input_fb.addr[0]       = (__u32)mem_in;
para.input_fb.size.width    = width;
para.input_fb.size.height   = height;
para.input_fb.mode          = DISP_MOD_INTERLEAVED;
para.input_fb.format        = DISP_FORMAT_ARGB8888;
para.input_fb.br_swap       = 0;
para.input_fb.seq           = DISP_SEQ_ARGB;
para.source_regn.x          = 0;
para.source_regn.y          = 0;
para.source_regn.width      = width;
para.source_regn.height     = height;
para.output_fb.addr[0]      = (__u32)mem_out0 + width*height*0;
para.output_fb.addr[1]      = (__u32)mem_out0 + width*height*1;
para.output_fb.addr[2]      = (__u32)mem_out0 + width*height*2;
para.output_fb.size.width   = width;
para.output_fb.size.height  = height;
para.output_fb.mode         = DISP_MOD_NON_MB_PLANAR;
para.output_fb.format       = DISP_FORMAT_YUV444;
para.output_fb.br_swap      = 0;
para.output_fb.seq          = DISP_SEQ_P3210;
arg[0] = scaler_hdl;
arg[1] = (__u32)&para;
arg[2] = 0;
eLIBs_fioctrl(disphd, DISP_CMD_SCALER_EXECUTE, sel, (void*)arg);

arg[0] = scaler_hdl;
arg[1] = 0;
arg[2] = 0;
eLIBs_fioctrl(disphd, DISP_CMD_SCALER_RELEASE, sel, (void*)arg);

//display src layer
layer_para.fb.addr[0]       = para.input_fb.addr[0];
layer_para.fb.addr[1]       = para.input_fb.addr[1];
layer_para.fb.addr[2]       = para.input_fb.addr[2];
layer_para.fb.size.width    = para.input_fb.size.width;
layer_para.fb.size.height   = para.input_fb.size.height;
layer_para.fb.mode          = para.input_fb.mode;
layer_para.fb.format        = para.input_fb.format;
layer_para.fb.br_swap       = para.input_fb.br_swap;
layer_para.fb.seq           = para.input_fb.seq;
layer_para.ck_enable        = 0;
layer_para.alpha_en         = 1;
layer_para.alpha_val        = 0xff;
layer_para.pipe             = 0;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
layer_para.src_win.x       = 0;
layer_para.src_win.y       = 0;
layer_para.src_win.width    = width;
layer_para.src_win.height   = height;
layer_para.scn_win.x       = 0;
layer_para.scn_win.y       = 0;
layer_para.scn_win.width    = width;
layer_para.scn_win.height   = height;

arg[0] = DISP_LAYER_WORK_MODE_NORMAL;
arg[1] = 0;
arg[2] = 0;
hlay = eLIBs_fioctrl(disphd, DISP_CMD_LAYER_REQUEST, sel, (void*)arg);
if(hlay == NULL)
{
    __wrn(""request layer0 fail\n"");
}
else
{
    layer_para.mode = DISP_LAYER_WORK_MODE_NORMAL;
    layer_para.pipe = 0;
    arg[0] = hlay;
    arg[1] = (__u32)&layer_para;
    arg[2] = 0;
    eLIBs_fioctrl(disphd,DISP_CMD_LAYER_SET_PARA,sel,(void*)arg);

    arg[0] = hlay;
    arg[1] = 0;
    arg[2] = 0;
    eLIBs_fioctrl(disphd,DISP_CMD_LAYER_OPEN,sel,(void*)arg);
}

//display dst0 layer
layer_para.fb.addr[0]      = para.output_fb.addr[0];
layer_para.fb.addr[1]      = para.output_fb.addr[1];
layer_para.fb.addr[2]      = para.output_fb.addr[2];
layer_para.fb.size.width    = para.output_fb.size.width;
layer_para.fb.size.height   = para.output_fb.size.height;
layer_para.fb.mode         = para.output_fb.mode;
layer_para.fb.format       = para.output_fb.format;
layer_para.fb.br_swap      = para.output_fb.br_swap;
layer_para.fb.seq          = para.output_fb.seq;
layer_para.ck_enable       = 0;
layer_para.alpha_en        = 1;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
layer_para.alpha_val        = 0xff;
layer_para.pipe            = 0;
layer_para.src_win.x        = 0;
layer_para.src_win.y        = 0;
layer_para.src_win.width     = width;
layer_para.src_win.height    = height;
layer_para.scn_win.x        = 0;
layer_para.scn_win.y        = 0;
layer_para.scn_win.width     = width;
layer_para.scn_win.height    = height;

arg[0] = DISP_LAYER_WORK_MODE_NORMAL;
arg[1] = 0;
arg[2] = 0;
hlay = eLIBs_fioctrl(disphd, DISP_CMD_LAYER_REQUEST, sel, (void*)arg);
if(hlay == NULL)
{
    __wrn(""request layer0 fail\n"");
}
else
{
    layer_para.mode = DISP_LAYER_WORK_MODE_NORMAL;
    layer_para.pipe = 0;
    arg[0] = hlay;
    arg[1] = (__u32)&layer_para;
    eLIBs_fioctrl(disphd,DISP_CMD_LAYER_SET_PARA,sel,(void*)arg);

    arg[0] = hlay;
    arg[1] = 0;
    arg[2] = 0;
    eLIBs_fioctrl(disphd,DISP_CMD_LAYER_OPEN,sel,(void*)arg);
}

__inf(""press to demo plannel yuv444 --> plannel yuv422\n"");
__getc();

scaler_hdl = eLIBs_fioctrl(disphd, DISP_CMD_SCALER_REQUEST, sel, 0);

para.input_fb.addr[0]      = (__u32)mem_out0 + width*height*0;
para.input_fb.addr[1]      = (__u32)mem_out0 + width*height*1;
para.input_fb.addr[2]      = (__u32)mem_out0 + width*height*2;
para.input_fb.size.width     = width;
para.input_fb.size.height    = height;
para.input_fb.mode          = DISP_MOD_NON_MB_PLANAR;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
para.input_fb.format       = DISP_FORMAT_YUV444;
para.input_fb.br_swap      = 0;
para.input_fb.seq         = DISP_SEQ_P3210;
para.source_regn.x         = 0;
para.source_regn.y         = 0;
para.source_regn.width     = width;
para.source_regn.height    = height;
para.output_fb.addr[0]     = (__u32)mem_out1 + width*height*0;
para.output_fb.addr[1]     = (__u32)mem_out1 + width*height*1;
para.output_fb.addr[2]     = (__u32)mem_out1 + width*height*2;
para.output_fb.size.width   = width;
para.output_fb.size.height  = height;
para.output_fb.mode        = DISP_MOD_NON_MB_PLANAR;
para.output_fb.format      = DISP_FORMAT_YUV422;
para.output_fb.br_swap     = 0;
para.output_fb.seq         = DISP_SEQ_P3210;
arg[0] = scaler_hdl;
arg[1] = (__u32)&para;
arg[2] = 0;
eLIBs_fioctrl(disphd, DISP_CMD_SCALER_EXECUTE, sel, (void*)arg);

arg[0] = scaler_hdl;
arg[1] = 0;
arg[2] = 0;
eLIBs_fioctrl(disphd, DISP_CMD_SCALER_RELEASE, sel, (void*)arg);

//display dst1 layer
layer_para.fb.addr[0]      = para.output_fb.addr[0];
layer_para.fb.addr[1]      = para.output_fb.addr[1];
layer_para.fb.addr[2]      = para.output_fb.addr[2];
layer_para.fb.size.width    = para.output_fb.size.width;
layer_para.fb.size.height   = para.output_fb.size.height;
layer_para.fb.mode         = para.output_fb.mode;
layer_para.fb.format       = para.output_fb.format;
layer_para.fb.br_swap      = para.output_fb.br_swap;
layer_para.fb.seq         = para.output_fb.seq;
layer_para.ck_enable       = 0;
layer_para.alpha_en        = 1;
layer_para.alpha_val       = 0xff;
layer_para.pipe           = 0;
layer_para.src_win.x       = 0;
layer_para.src_win.y       = 0;
layer_para.src_win.width    = width;
layer_para.src_win.height   = height;
```

```
    layer_para.scn_win.x        = 0;
    layer_para.scn_win.y        = 0;
    layer_para.scn_win.width    = width;
    layer_para.scn_win.height   = height;

    arg[0] = DISP_LAYER_WORK_MODE_SCALER;
    arg[1] = 0;
    arg[2] = 0;
    hlay = eLIBs_fioctrl(disphd, DISP_CMD_LAYER_REQUEST, sel, (void*)arg);
    if(hlay == NULL)
    {
        __wrn(""request layer0 fail\n"");
    }
    else
    {
        layer_para.mode = DISP_LAYER_WORK_MODE_SCALER;
        layer_para.pipe = 1;
        arg[0] = hlay;
        arg[1] = (__u32)&layer_para;
        arg[2] = 0;
        eLIBs_fioctrl(disphd,DISP_CMD_LAYER_SET_PARA,sel,(void*)arg);

        arg[0] = hlay;
        arg[1] = 0;
        arg[2] = 0;
        eLIBs_fioctrl(disphd,DISP_CMD_LAYER_OPEN,sel,(void*)arg);
    }

    return 0;
}
```

25.7.3. HWC Demo

```
__s32 hwc_demo(ES_FILE *disphd)
{
    __u32 arg[3];
    __u8  *mem_in = NULL;
    __u32 *palette = NULL;
    __disp_hwc_pattern_t pattern;
    __disp_pos_t pos;
    __u32 i = 0;

    mem_in = esMEMS_Balloc(32*32/2);
    memset(mem_in,0x00,32*32/2/2);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
    memset(mem_in + 32*16/2,0xff,32*32/2/2);

    palette = esMEMS_Balloc(16*4);
    for(i=0; i<16; i++)
    {
        *(palette+i) = (__u32)(0xff<<24) | ((i*16)<<16) | ((i*16)<<8) | (i*16);
    }

    pattern.addr = (__u32)mem_in;
    pattern.pat_mode = DISP_HWC_MOD_H32_V32_4BPP;
    arg[0] = (__u32)&pattern;
    arg[1] = 0;
    arg[2] = 0;
    eLIBs_fioctrl(disphd, DISP_CMD_HWC_SET_FB, 0, (void*)arg);

    pos.x = 100;
    pos.y = 200;
    arg[0] = (__u32)&pos;
    arg[1] = 0;
    arg[2] = 0;
    eLIBs_fioctrl(disphd, DISP_CMD_HWC_SET_POS, 0, (void*)arg);

    arg[0] = (__u32)palette;
    arg[1] = 0;
    arg[2] = 256*4;
    eLIBs_fioctrl(disphd, DISP_CMD_HWC_SET_PALETTE_TABLE, 0, (void*)arg);

    eLIBs_fioctrl(disphd, DISP_CMD_HWC_OPEN, 0, 0);

    return 0;
}
```

25.7.4. Video Demo

```
__s32 video_demo(ES_FILE *disphd, __u32 sel)
{
    __disp_layer_info_t layer_para;
    __disp_video_fb_t video_fb;
    __u32 arg[3];
    __u32 * mem[2];
    __u32 j = 0;
    __hdle hlay;

    mem[0] = esMEMS_Balloc(800*480*4);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
mem[1] = esMEMS_Balloc(800*480*4);
eLIBs_memset(mem[0],0x00,800*480*4);
eLIBs_memset(mem[1],0x00,800*480*4);
for(j=0;j<800*120;j++)
{
    *(mem[0] + j) = 0x12345678;
    *(mem[0] + 800*120 +j) = 0xfedcba98;
    *(mem[0] + 800*120*2 +j) = 0x76543210;
    *(mem[0] + 800*120*3 +j) = 0x75395186;
}
for(j=0;j<800*120;j++)
{
    *(mem[1] + j) = 0xff00ff00;
    *(mem[1] + 800*120 +j) = 0xff00ff00;
    *(mem[1] + 800*120*2 +j) = 0x00ff00ff;
    *(mem[1] + 800*120*3 +j) = 0x00ffff00;
}

layer_para.fb.addr[0]       = (__u32)(mem[0] + 800*120*0);
layer_para.fb.addr[1]       = (__u32)(mem[0] + 800*120*1);
layer_para.fb.addr[2]       = (__u32)(mem[0] + 800*120*2);
layer_para.fb.size.width     = 800;
layer_para.fb.size.height    = 480;
layer_para.fb.mode          = DISP_MOD_MB_UV_COMBINED;
layer_para.fb.format        = DISP_FORMAT_YUV420;
layer_para.fb.br_swap       = 0;
layer_para.fb.seq           = DISP_SEQ_UVUV;
layer_para.ck_enable        = 0;
layer_para.alpha_en         = 1;
layer_para.alpha_val        = 0xff;
layer_para.pipe             = 0;
layer_para.src_win.x        = 0;
layer_para.src_win.y        = 0;
layer_para.src_win.width     = 800;
layer_para.src_win.height    = 480;
layer_para.scn_win.x        = 0;
layer_para.scn_win.y        = 0;
layer_para.scn_win.width     = 800;
layer_para.scn_win.height    = 480;

//layer0
arg[0] = DISP_LAYER_WORK_MODE_SCALER;
arg[1] = 0;
arg[2] = 0;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
hlay = eLIBs_fioctrl(disphd, DISP_CMD_LAYER_REQUEST, sel, (void*)arg);
if(hlay == NULL)
{
    __wrn(""request layer0 fail\n"");
}
else
{
    layer_para.mode = DISP_LAYER_WORK_MODE_SCALER;
    layer_para.pipe = 0;
    arg[0] = hlay;
    arg[1] = (__u32)&layer_para;
    arg[2] = 0;
    eLIBs_fioctrl(disphd,DISP_CMD_LAYER_SET_PARA,sel,(void*)arg);

    arg[0] = hlay;
    arg[1] = 0;
    arg[2] = 0;
    eLIBs_fioctrl(disphd,DISP_CMD_LAYER_OPEN,sel,(void*)arg);
}

arg[0] = hlay;
arg[1] = 0;
arg[2] = 0;
eLIBs_fioctrl(disphd, DISP_CMD_VIDEO_START, sel, (void*)arg);

j = 0;
while(j < 2)
{
    __u32 id = 0;

    eLIBs_memset(&video_fb, 0, sizeof(__disp_video_fb_t));
    video_fb.id = j;
    video_fb.addr[0] = (__u32)(mem[j] + 800*120*0);
    video_fb.addr[1] = (__u32)(mem[j] + 800*120*1);
    video_fb.addr[2] = (__u32)(mem[j] + 800*120*2);
    arg[0] = hlay;
    arg[1] = (__u32)&video_fb;
    arg[2] = 0;
    eLIBs_fioctrl(disphd, DISP_CMD_VIDEO_SET_FB, sel, (void*)arg);

    arg[0] = hlay;
    arg[1] = 0;
    arg[2] = 0;
    id = eLIBs_fioctrl(disphd, DISP_CMD_VIDEO_GET_FRAME_ID, sel, (void*)arg);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
        j++;
        esKRNL_TimeDly(100);
    }

    arg[0] = hlay;
    arg[1] = 0;
    arg[2] = 0;
    eLIBs_fioctrl(disphd, DISP_CMD_VIDEO_STOP, sel, (void*)arg);

    arg[0] = hlay;
    arg[1] = 0;
    arg[2] = 0;
    eLIBs_fioctrl(disphd, DISP_CMD_LAYER_RELEASE, sel, (void*)arg);

    return 0;
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26. Orange Module

26.1. Introduction

26.1.1. Description

Orange1.0 是基于 Melis 操作系统之上的一套 GUI 系统，该系统支持多任务和多图层操作，允许在多个图层上面创建窗口，提供完善的异步和同步窗口消息通讯机制，提供点、线、矩形、扇形、椭圆、圆等基本形状的绘制和填充，支持 TTF 矢量字体、SFT 点阵字体及字体的加边框和加阴影特效，支持文本的显示输出和 BMP 位图的绘制，支持 button、static、listmenu、slider…等十几种控件，提供 alphablending、colorkey 等特效，支持 alpha、copy、fill 的 2D 加速，支持多国语言和内存设备等。在 Orange 中，窗口是 Orange 管理的基本单位，Orange 采用事件驱动编程，窗口是接收事件并分发处理事件的最小单元，在 Orange 中所有的消息响应基于消息循环，窗口不断从消息队列中获取消息并分发给响应的窗口过程处理。

26.1.2. Purpose

本文档主要讲述 Orange1.0 相关的编程接口，让开发者能快速掌握该系统的特性和编程接口，并基于 Melis 系统来开发自己的应用程序。

26.1.3. Reference

读者可以先了解一些 GUI 系统相关的知识，了解一下应用程序的编写流程和组织框架。

26.1.4. Contact Info

如果您发现文档中的描述和实际应用有出入，或是文档描述不清晰让您有疑问，请随时联系我们，我们将最短时间内做出答复。同时，如果您有任何建议或批评，也希望您不吝赐教。
请通过以下联系方式联系我们：
Homepage: http://www.allwinnertech.com
E-mail: laiyandong@allwinnertech.com

26.2. Window

Melis 操作系统上所有内存分配的源头都是基于内存页池的页分配，Melis 内核上一个页(page)的大小为 1Kbyte。内核初始化时，将所有的内存都放置在系统页池中，系统堆的创建、虚拟空间的创建等等用到的内存全部由页分配而来。页分配得到的内存块在物理空间上是连续的，因此，一些对物理空间连续性有要求的用户，必须采用 palloc 来分配内存。

26.2.1. 分类

➢ **管理窗口**

管理窗口是虚拟窗口，负责消息分发和处理，支持跨图层管理，它有两种用途：
1.应用程序的入口窗口，有自己的消息队列，负责消息接收和分发。
2.统筹多图层下的窗口消息处理，形成一个统一的整体。

➢ **Frmwin 和对话框**

Frmwin 寄生在一块 framebuffer 上，是一个实体窗口，有自己的矩形区域。Framewin 有自己的标题栏和客户区。
模式对话框：需要响应关闭之后才能响应其他消息。
非模式对话框：不需要关闭，也可以响应其他消息。

➢ **控件窗口**

封装好的具有特点属性和操作的特殊的实体窗口，其目的是方便应用程序的复制，最大限度减少代码的复用。加快开发效率。

➢ **图层窗口**

图层窗口对应一个图层，是屏幕管理的基本单位，是屏幕中的一个实体窗口，有自己的显示区域和 Z 序，也是其他实体窗口（frmwin 和控件窗口）的载体。

26.2.2. 关系

窗口从属关系如下所示：

➢ **父子兄弟关系**

窗口可以指定父窗口，以此来创建窗口家族树关系，管理窗口的父窗口只能是管理窗口或者空窗口（系统会默认根窗口为其父窗口）。Frmwin 窗口的父窗口只能是管理窗口，控件窗口的父窗口可能为 frmwin，也可能是控件。

➢ **Z 序关系**

窗口在图层中的显示序列称为 Z 序，Z 序值越大的窗口处在图层的位置就越高。

➢ **Owner 关系**

Owner 窗口仅仅对管理窗口和 framewin 窗口才有效。表示管理窗口或者 framewin 窗口的创建时的所属关系。用来对平级的窗口创建进行管理。
引入 Owner 窗口的目的是为了表示窗口创建时的所属管理。并实现模式对话框和模式管理窗口。

➢ **Note**

对管理窗口而言，一个管理窗口创建另外一个管理窗口时有两种状态、一种状态是新创建的管理窗口是该管理窗口的子窗口，此时共享消息队列。并且新创建的管理窗口的消息来源于父窗口的分发。其目的是用来对管理窗口的子单元场景进行管理。
管理窗口创建新管理窗口的另外一种状态是，创建所属窗口。这种情况下，新窗口并不是该窗口的子窗口，而是该窗口的兄弟窗口。因此，尽管共享同一个消息队列，但是消息来源却直接来自于他们共同的父窗口。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

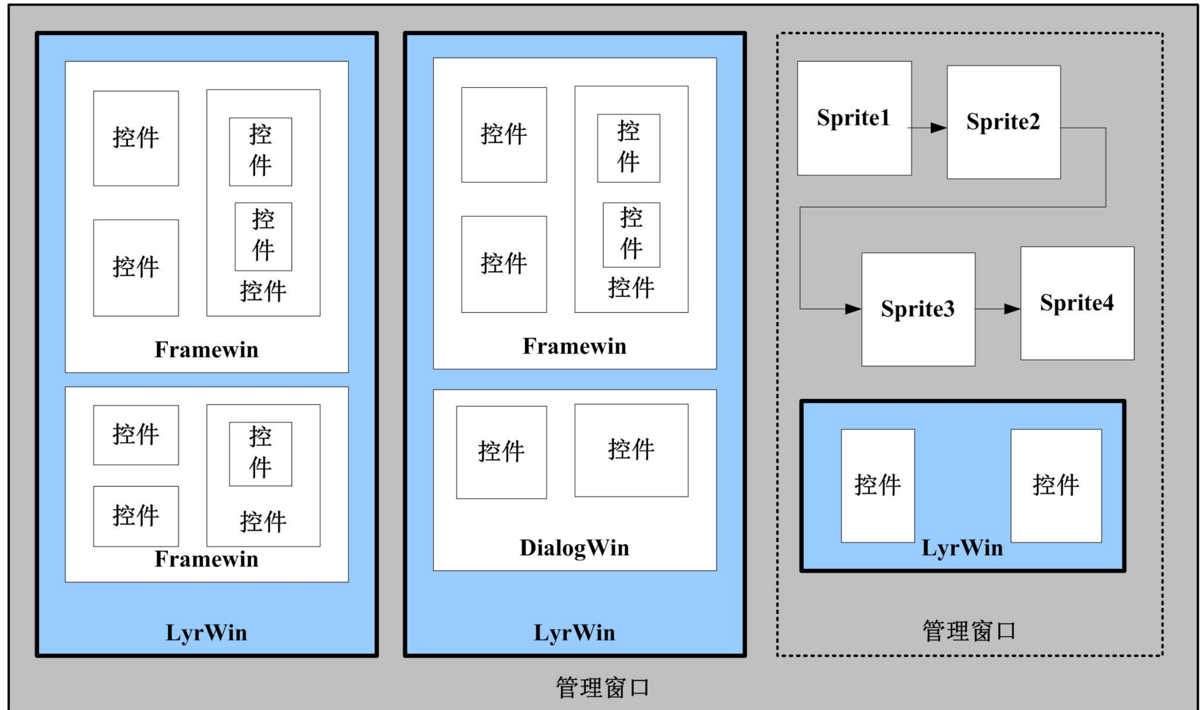


图 2.1

窗口从属关系示意图

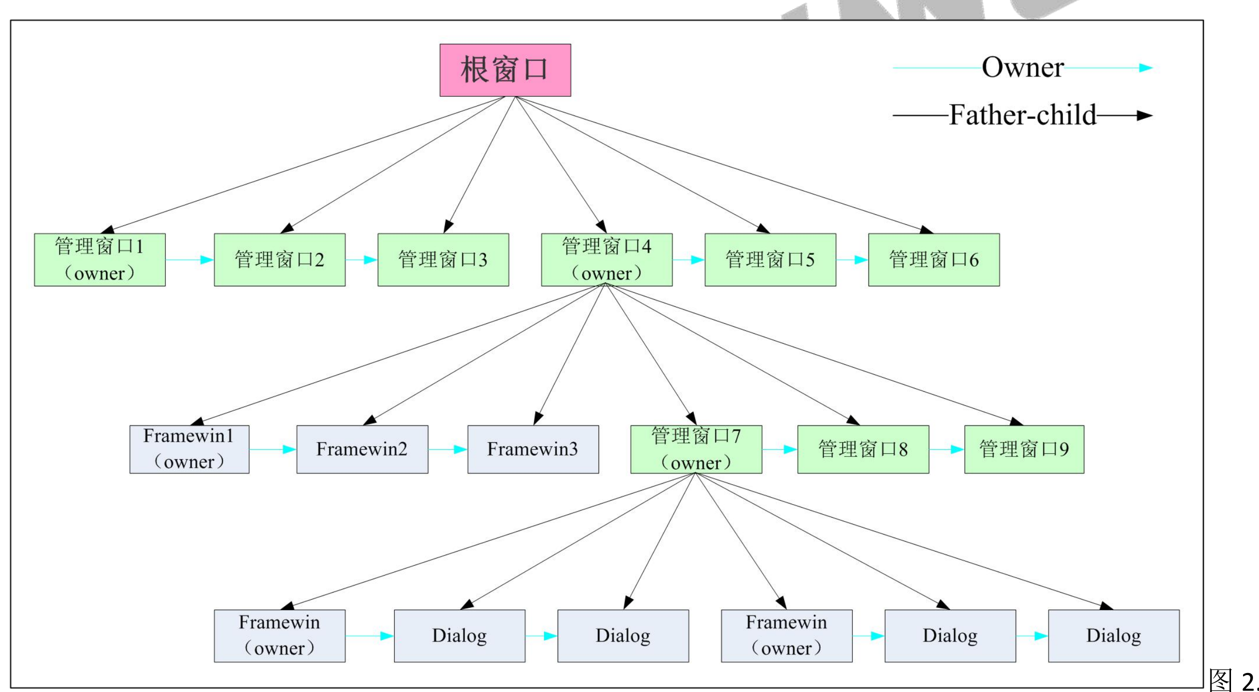


图 2.2

窗口关系树形示意图

26.2.3. 图层属性

- 图层 framebuffer 虚拟地址：应用程序或者中间件在堆中分配的一段连续物理空间的首地址。
- 图层 FrameBuffer 格式

对 normal 图层而言主要有以下几种：

PIXEL_MONO_1BPP PIXEL_MONO_2BPP PIXEL_MONO_4BPP

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

PIXEL_MONO_8BPP PIXEL_COLOR_RGB655 PIXEL_COLOR_RGB565
PIXEL_COLOR_RGB556 PIXEL_COLOR_ARGB1555 PIXEL_COLOR_RGBA5551
PIXEL_COLOR_RGB0888 PIXEL_COLOR_ARGB8888

对 palette/Inter 模式的图层而言，主要有以下几种：
PIXEL_MONO_1BPP PIXEL_MONO_2BPP PIXEL_MONO_4BPP
PIXEL_MONO_8BPP

Scaler 图层的格式主要有以下几种：
PIXEL_YUV444 PIXEL_YUV422 PIXEL_YUV420
PIXEL_YUV411 PIXEL_CSIRGB PIXEL_COLOR_ARGB8888
PIXEL_OTHERFMT

- 图层 Framebuffer 大小

宽度（像素） 高度（像素）

- 图层 Framebuffer 需要显示的区域

包括需要显示的偏移值（X,Y）和需要显示的 framebuffer 区域的大小（width，height）。

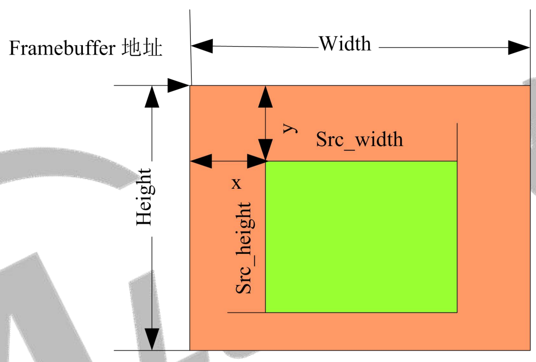


图 2.3 Framebuffer 显示区域示意图

26.2.4. 屏幕属性

- 屏幕区域

包括图层的屏幕坐标（X 坐标，Y 坐标）和图层需要显示的大小（图层的屏幕宽度，图层的屏幕高度）。

- 图层的 interlace 属性（仅仅对 scaler 图层 TV 输出而言）

Progressive or interlace
Frame mode

26.2.5. 优先级的概念

优先级最高的图层处于屏幕的最前面。

26.2.6. Pipe 的概念

对输出通道为 DE_CH1 的图层有效。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

共有两个 pipe 可以选择。

当多个图层选择同一个 pipe 时，它们的公共重叠的地方只有优先级高的图层能够显示，优先级低的图层被覆盖。

下图是优先级分别为 1，2，3 的图层通过同一管道时的示意图：

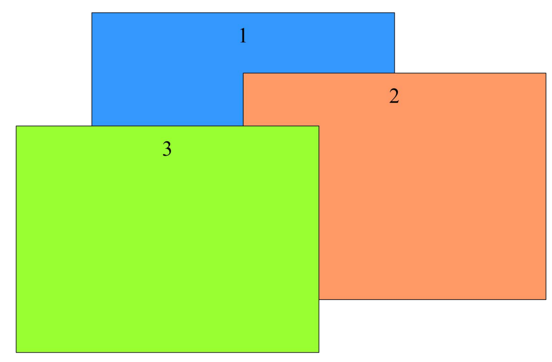


图 2.5 三个图层通过同一个 pipe 示意图

26.2.7. Alpha 的计算公式

当图层通过不同的 pipe 时，图层之间根据 alpha 值做 alpha 运算。Alpha 的计算公式如下：

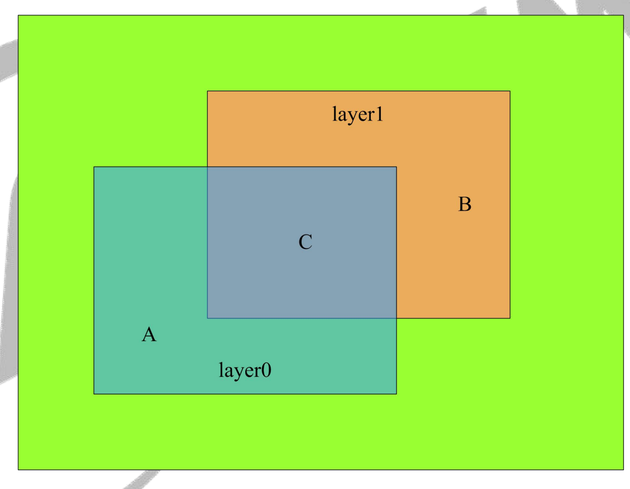


图 2.6 alpha 示意图

如上图所示：

RA = R0*A0+Rg(1-A0);

GA = G0*A0+Gg(1-A0);

BA = B0*A0+Bg(1-A0);

RB = R1*A1+Rg(1-A1);

GB = G1*A1+Gg(1-A1);

BB = B1*A1+Bg(1-A1);

如果 layer0 的优先级高于 layer1 的优先级。

Rc=R0*A0+ (R1*A1+Rg(1-A1))*(1-A0);

Gc=G0*A0+ (G1*A1+Gg(1-A1))*(1-A0);

Bc=B0*A0+ (B1*A1+Bg(1-A1))*(1-A0);

如果 layer0 的优先级低于 layer1 的优先级。

Rc=R1*A1+ (R0*A0+Rg(1-A0))*(1-A1);

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

Gc=G1*A1+ (G0*A0+Gg(1-A0))*(1-A1);

Bc=B1*A1+ (B0*A0+Bg(1-A0))*(1-A1);

其中

R1，G1，B1，A1 为图层 1 的 R 值，G 值，B 值，Alpha 值。

R0，G0，B0，A0 为图层 0 的 R 值，G 值，B 值，Alpha 值。

Rg，Gg，Bg 为背景的 R 值，G 值，B 值。

26.2.8. Colorkey 的运算

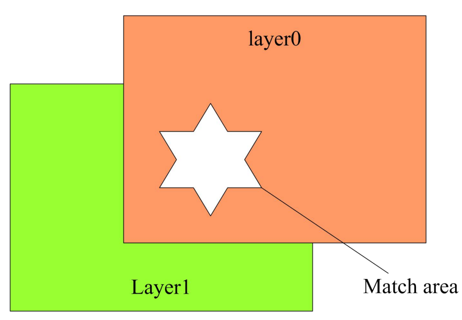


图 2.7 color key 示意图

对匹配区域而言（匹配区域由应用程序指定）。

如果 layer0 的优先级高于 layer1

Layer0 ck_en 设为 True：

Layer1 ck_en 设为 True or False：

故 colorkey match 结果是 layer0 匹配 layer1

R = R0*A0+Rg(1-A0);

G = G0*A0+Gg(1-A0);

B = B0*A0+Bg(1-A0);

如果

Layer0 ck_en 设为 FALSE：

Layer1 ck_en 设为 TRUE：

故 colorkey match 结果是 layer1 匹配 layer0

R = R1*A1+Rg(1-A1);

G = G1*A1+Gg(1-A1);

B = B1*A1+Bg(1-A1);

Colorkey 仅匹配输出选择不同 pipe 的图层。

26.2.9. Interface

26.2.9.1. GUI_FrmWinCreate

- **PROTOTYPE**

```
H_WIN    GUI_FrmWinCreate (pframewincreate create_info);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

- **ARGUMENTS**

create_info　　frmwin 创建信息结构;

- **RETURNS**

Frmwin 句柄;

- **DESCRIPTION**

创建 frmwin。

26.2.9.2. GUI_ManWinCreate

- **PROTOTYPE**

```
H_WIN    GUI_ManWinCreate (pmanwincreate create_info);
```

- **ARGUMENTS**

create_info　　管理窗口创建信息结构

- **RETURNS**

管理窗口句柄

- **DESCRIPTION**

创建管理窗口

26.2.9.3. GUI_FrmWinDelete

- **PROTOTYPE**

```
__s32    GUI_FrmWinDelete(H_WIN   hframewin);
```

- **ARGUMENTS**

hframewin　　frmwin 窗口句柄

- **RETURNS**

ORANGE_OK　删除成功
ORANGE_FAIL 删除失败

- **DESCRIPTION**

删除 frmwin

26.2.9.4. GUI_ManWinDelete

- **PROTOTYPE**

```
__s32    GUI_ManWinDelete (H_WIN   hmanwin);
```

- **ARGUMENTS**

hmanwin　　　管理窗口的句柄

- **RETURNS**

ORANGE_OK　删除成功
ORANGE_FAIL 删除失败

- **DESCRIPTION**

删除管理窗口

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.2.9.5. GUI_CtrlWinCreate

- **PROTOTYPE**

```
H_WIN    GUI_CtrlWinCreate (__gui_ctlwincreate_para_t *create_info);
```

- **ARGUMENTS**

create_info 控件窗口创建信息结构

- **RETURNS**

控件窗口句柄

- **DESCRIPTION**

创建控件窗口

26.2.9.6. GUI_CtrlWinDelete

- **PROTOTYPE**

```
__s32    GUI_CtrlWinDelete (H_WIN    hWnd);
```

- **ARGUMENTS**

hWnd 窗口句柄

- **RETURNS**

ORANGE_OK 删除成功

ORANGE_FAIL 删除失败

- **DESCRIPTION**

删除控件窗口

26.2.9.7. GUI_WinThreadCleanup

- **PROTOTYPE**

```
Void    GUI_WinThreadCleanup (H_WIN hManWnd);
```

- **ARGUMENTS**

hManWnd 窗口句柄

- **RETURNS**

None

- **DESCRIPTION**

gui 窗口线程相关的信息删除，用来在主窗口或者 framewin 窗口结束后，清楚与线程相关的信息。

26.2.9.8. GUI_SetActiveManWin

- **PROTOTYPE**

```
H_WIN    GUI_SetActiveManWin (H_WIN    hWnd);
```

- **ARGUMENTS**

hWnd 系统全局焦点管理窗口句柄。

- **RETURNS**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

上一次的焦点管理窗口的句柄。

- **DESCRIPTION**

通过这个函数设置系统全局的焦点管理窗口句柄。该窗口通常为某个应用程序的入口窗口。

26.2.9.9. GUI_GetActiveManWin

- **PROTOTYPE**

```
H_WIN    GUI_GetActiveManWin (void);
```

- **ARGUMENTS**

void

- **RETURNS**

系统全局焦点管理窗口句柄

- **DESCRIPTION**

通过这个函数获取系统全局的焦点管理窗口句柄。该窗口通常为某个应用程序的入口窗口。

26.2.9.10. GUI_WinGetFocusChild

- **PROTOTYPE**

```
H_WIN    GUI_WinGetFocusChild (H_WIN h_win);
```

- **ARGUMENTS**

h_win　父窗口句柄

- **RETURNS**

h_win　焦点子窗口句柄

- **DESCRIPTION**

获取焦点子窗口句柄

26.2.9.11. GUI_WinSetFocusChild

- **PROTOTYPE**

```
__s32    GUI_WinSetFocusChild (H_WIN h_win);
```

- **ARGUMENTS**

h_win　需要设置的焦点子窗口句柄

- **RETURNS**

ORANGE_OK　设置成功

ORANGE_FAIL 设置失败

- **DESCRIPTION**

设置焦点子窗口

26.2.9.12. GUI_WinGetType

- **PROTOTYPE**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
__s32    GUI_WinGetType (H_WIN hWnd);
```

- **ARGUMENTS**

hWnd　　窗口句柄

- **RETURNS**

窗口类型

- **DESCRIPTION**

获取窗口类型

26.2.9.13. GUI_WinGetMainManWin

- **PROTOTYPE**

```
H_WIN    GUI_WinGetMainManWin (H_WIN hWnd);
```

- **ARGUMENTS**

hWnd　　窗口句柄

- **RETURNS**

窗口的入口管理窗口句柄

- **DESCRIPTION**

通过该窗口获取该应用程序的全局管理窗口的句柄。

26.2.9.14. GUI_WinGetManWin

- **PROTOTYPE**

```
H_WIN    GUI_WinGetManWin (H_WIN hWnd);
```

- **ARGUMENTS**

hWnd　　窗口句柄

- **RETURNS**

窗口的父管理窗口句柄

- **DESCRIPTION**

通过该窗口获取该窗口的局部管理窗口的句柄。

26.2.9.15. GUI_WinGetParent

- **PROTOTYPE**

```
H_WIN    GUI_WinGetParent (H_WIN hWnd);
```

- **ARGUMENTS**

hWnd　　窗口句柄

- **RETURNS**

窗口的父窗口句柄

- **DESCRIPTION**

通过该窗口获取该窗口的父窗口的句柄。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.2.9.16. GUI_WinGetFirstChild

- **PROTOTYPE**

```
H_WIN    GUI_WinGetFirstChild (H_WIN hWnd);
```

- **ARGUMENTS**

hWnd 窗口句柄

- **RETURNS**

窗口的第一个子窗口句柄

- **DESCRIPTION**

通过该窗口获取该窗口的第一个子窗口的句柄。

26.2.9.17. GUI_WinGetNextBro

- **PROTOTYPE**

```
H_WIN    GUI_WinGetNextBro (H_WIN hWnd);
```

- **ARGUMENTS**

hWnd 窗口句柄

- **RETURNS**

窗口的下一个兄弟窗口句柄

- **DESCRIPTION**

通过该窗口获取该窗口的下一个兄弟窗口的句柄。

26.2.9.18. GUI_WinGetNextHostedWin

- **PROTOTYPE**

```
H_WIN       GUI_WinGetNextHostedWin (H_WIN    hWnd);
```

- **ARGUMENTS**

hWnd 窗口句柄

- **RETURNS**

窗口的下一个 Hosted 窗口句柄

- **DESCRIPTION**

通过该窗口获取该窗口的下一个 Hosted 窗口的句柄。

26.2.9.19. GUI_WinGetFirstHostedWin

- **PROTOTYPE**

```
H_WIN    GUI_WinGetFirstHostedWin (H_WIN    hWnd);
```

- **ARGUMENTS**

hWnd 窗口句柄

- **RETURNS**

窗口的第一个 Hosted 窗口句柄

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

➢ **DESCRIPTION**

通过该窗口获取该窗口的第一个 Hosted 窗口的句柄。

26.2.9.20. GUI_WinGetOwnerWin

➢ **PROTOTYPE**

```
H_WIN    GUI_WinGetOwnerWin (H_WIN    hWnd);
```

➢ **ARGUMENTS**

hWnd   窗口句柄

➢ **RETURNS**

窗口的 Owner 窗口句柄

➢ **DESCRIPTION**

通过该窗口获取该窗口的 Owner 窗口的句柄。

26.2.9.21. GUI_WinGetRootWin

➢ **PROTOTYPE**

```
H_WIN    GUI_WinGetRootWin (void);
```

➢ **ARGUMENTS**

void

➢ **RETURNS**

返回根窗口句柄

➢ **DESCRIPTION**

获取根窗口句柄

26.2.9.22. GUI_WinIsAncestor

➢ **PROTOTYPE**

```
__bool    GUI_WinIsAncestor (H_WIN hWnd, H_WIN hChild);
```

➢ **ARGUMENTS**

hWnd   祖先窗口句柄

hChild   子孙窗口句柄

➢ **RETURNS**

满足条件返回 ORANGE_TRUE

否则返回 ORANGE_FALSE

➢ **DESCRIPTION**

判断 *hWnd* 窗口是否是 *hChild* 窗口的祖先窗口。

26.2.9.23. GUI_WinIsChild

➢ **PROTOTYPE**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
    __bool    GUI_WinIsChild (H_WIN hWnd,H_WIN hPraent);
```

- **ARGUMENTS**

  hWnd　　窗口句柄

  hPraent　父窗口句柄

- **RETURNS**

  满足条件返回 ORANGE_TRUE

  否则返回 ORANGE_FALSE

- **DESCRIPTION**

  判断 *hPraent* 窗口是否是 *hWnd* 窗口的父窗口

26.2.9.24. GUI_WinGetDlgItem

- **PROTOTYPE**

```
    H_WIN    GUI_WinGetDlgItem (H_WIN hDlg, __s32 nIDDlgItem);
```

- **ARGUMENTS**

  hDlg　　父窗口句柄

  nIDDlgItem　子窗口 ID

- **RETURNS**

  子窗口句柄

- **DESCRIPTION**

  通过子窗口 id 来获取子窗口句柄。

26.2.9.25. GUI_WinGetItemId

- **PROTOTYPE**

```
    __u32    GUI_WinGetItemId (H_WIN hItem);
```

- **ARGUMENTS**

  hItem　控件窗口句柄

- **RETURNS**

  控件窗口 ID

- **DESCRIPTION**

  获取控件窗口 id。

26.2.9.26. GUI_WinGetHandFromName

- **PROTOTYPE**

```
    H_WIN    GUI_WinGetHandFromName (char * name);
```

- **ARGUMENTS**

  name　窗口名称

- **RETURNS**

  窗口的窗口句柄

- **DESCRIPTION**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

通过该窗口窗口名称该窗口的句柄。

26.2.9.27. GUI_WinGetAddData

- **PROTOTYPE**

```
__u32    GUI_WinGetAddData (H_WIN hWnd);
```

- **ARGUMENTS**

hWnd　窗口句柄

- **RETURNS**

窗口的额外控制信息结构句柄

- **DESCRIPTION**

获取该窗口的额外控制信息结构句柄。

26.2.9.28. GUI_WinSetAddData

- **PROTOTYPE**

```
__s32    GUI_WinSetAddData(H_WIN hWnd, __u32 dwAddData);
```

- **ARGUMENTS**

hWnd　窗口句柄

dwAddData　窗口额外控制信息结构地址

- **RETURNS**

ORANGE_OK　设置成功

ORANGE_FAIL 设置失败

- **DESCRIPTION**

设置该窗口的额外控制信息结构句柄。

26.2.9.29. GUI_WinGetStyle

- **PROTOTYPE**

```
__u32    GUI_WinGetStyle (H_WIN hWnd);
```

- **ARGUMENTS**

hWnd　窗口句柄

- **RETURNS**

窗口的类型信息

- **DESCRIPTION**

获取窗口的类型信息。

26.2.9.30. GUI_WinGetFrmWin

- **PROTOTYPE**

```
H_WIN    GUI_WinGetFrmWin (H_WIN hWnd);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

- **ARGUMENTS**

  hWnd　窗口句柄

- **RETURNS**

  窗口的 frmwin 窗口句柄

- **DESCRIPTION**

  通过该窗口获取该窗口的 frmwin 窗口的句柄。

26.2.9.31. GUI_WinGetAttr

- **PROTOTYPE**

```
void*    GUI_WinGetAttr (H_WIN h_win);
```

- **ARGUMENTS**

  h_win　窗口句柄

- **RETURNS**

  void*　窗口属性地址

- **DESCRIPTION**

  获取窗口属性(用户传递窗口时需要传递的数据)。

26.2.9.32. GUI_WinSetAttr

- **PROTOTYPE**

```
__s32    GUI_WinSetAttr (H_WIN hWnd, void *attr);
```

- **ARGUMENTS**

  hWnd　窗口句柄

  attr　窗口私有属性信息

- **RETURNS**

  ORANGE_OK　设置成功

  ORANGE_FAIL 设置失败

- **DESCRIPTION**

  设置该窗口的窗口私有属性信息。

26.2.9.33. GUI_WinGetLyrWin

- **PROTOTYPE**

```
H_LYR   GUI_WinGetLyrWin(H_WIN h_win);
```

- **ARGUMENTS**

  h_win　窗口句柄

- **RETURNS**

  H_LYR　图层窗口句柄

- **DESCRIPTION**

  获取窗口所在图层的句柄

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.2.9.34. GUI_WinGetName

- **PROTOTYPE**

```
__s32   GUI_WinGetName(H_WIN h_win, char * name);
```

- **ARGUMENTS**

h_win　窗口句柄
name　窗口名字

- **RETURNS**

ORANGE_OK　成功
ORANGE_FAIL 失败

- **DESCRIPTION**

获取窗口名字

26.2.9.35. GUI_ManWinDefaultProc

- **PROTOTYPE**

```
Void        GUI_ManWinDefaultProc(__gui_msg_t * msg);
```

- **ARGUMENTS**

msg　输入消息指针

- **RETURNS**

void

- **DESCRIPTION**

管理窗口默认处理函数

26.2.9.36. GUI_CtrlWinDefaultProc

- **PROTOTYPE**

```
void    GUI_CtrlWinDefaultProc    (__gui_msg_t * msg);
```

- **ARGUMENTS**

msg　输入消息指针指针

- **RETURNS**

void

- **DESCRIPTION**

控件窗口默认处理函数

26.2.9.37. GUI_FrmWinDefaultProc

- **PROTOTYPE**

```
void    GUI_ FrmWinDefaultProc(__gui_msg_t * msg);
```

- **ARGUMENTS**

msg　输入消息指针

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

- **RETURNS**

  void

- **DESCRIPTION**

  Frmwin 窗口默认处理函数

26.2.9.38. GUI_WinSetCallback

- **PROTOTYPE**

```
__pGUI_WIN_CB GUI_WinSetCallback (H_WIN h_win, __pGUI_WIN_CB cb);
```

- **ARGUMENTS**

  h_win　　窗口句柄

  cb　　窗口回调函数

- **RETURNS**

  调整后的回调函数入口地址

- **DESCRIPTION**

  设置窗口回调函数

26.2.9.39. GUI_WinGetCallback

- **PROTOTYPE**

```
__pGUI_WIN_CB GUI_WinGetCallback (H_WIN hWnd);
```

- **ARGUMENTS**

  hWnd　　窗口句柄

- **RETURNS**

  调整后的回调函数入口地址

- **DESCRIPTION**

  获取窗口回调函数

26.2.9.40. GUI_WinSetNotifyCallback

- **PROTOTYPE**

```
NOTIFPROC GUI_WinSetNotifyCallback(H_WIN hwnd, NOTIFPROC notif_proc);
```

- **ARGUMENTS**

  hwnd　　窗口句柄

  notif_proc　　窗口通知回调函数

- **RETURNS**

  调整后的窗口通知回调函数入口地址

- **DESCRIPTION**

  设置窗口后处理回调函数

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.2.9.41. GUI_WinGetNotifyCallback

➢ **PROTOTYPE**

```
NOTIFPROC GUI_WinGetNotifyCallback(H_WIN hwnd);
```

➢ **ARGUMENTS**

hwnd　　窗口句柄

➢ **RETURNS**

调整后的窗口通知回调函数入口地址

➢ **DESCRIPTION**

获取窗口后处理回调函数

26.2.9.42. GUI_LyrWinWinCreate

➢ **PROTOTYPE**

```
H_LYR    GUI_LyrWinWinCreate(__gui_lyrwincreate_info_t * create_info);
```

➢ **ARGUMENTS**

create_info　　图层创建信息结构指针

➢ **RETURNS**

图层句柄

➢ **DESCRIPTION**

创建图层

26.2.9.43. GUI_LyrWinWinDelete

➢ **PROTOTYPE**

```
__s32    GUI_LyrWinWinDelete (H_LYR h_lyr);
```

➢ **ARGUMENTS**

h_lyr　　图层句柄

➢ **RETURNS**

ORANGE_OK　　删除成功

ORANGE_FAIL　　删除失败

➢ **DESCRIPTION**

删除图层

26.2.9.44. GUI_LyrWinSetSrcWindow

➢ **PROTOTYPE**

```
__s32    GUI_LyrWinSetSrcWindow(H_LYR h_lyr, const RECT * rect);
```

➢ **ARGUMENTS**

h_lyr　　图层句柄

rect　　源区域矩形指针

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

➢ **RETURNS**

ORANGE_OK 设置成功

ORANGE_FAIL 设置失败

➢ **DESCRIPTION**

设置图层显示区域在图层中的位置窗口

26.2.9.45. GUI_LyrWinSetScnWindow

➢ **PROTOTYPE**

```
__s32    GUI_LyrWinSetScnWindow(H_LYR h_lyr, const RECT * rect);
```

➢ **ARGUMENTS**

h_lyr 图层句柄

rect 屏幕区域矩形指针

➢ **RETURNS**

ORANGE_OK 设置成功

ORANGE_FAIL 设置失败

➢ **DESCRIPTION**

设置图层显示区域在屏幕中的位置窗口

26.2.9.46. GUI_LyrWinGetSrcWindow

➢ **PROTOTYPE**

```
__s32    GUI_LyrWinGetSrcWindow    (H_LYR h_lyr, RECT * rect);
```

➢ **ARGUMENTS**

h_lyr 图层句柄

rect 源区域矩形指针

➢ **RETURNS**

ORANGE_OK 获取成功

ORANGE_FAIL 获取失败

➢ **DESCRIPTION**

获取图层显示区域在图层中的位置窗口

26.2.9.47. GUI_LyrWinGetScnWindow

➢ **PROTOTYPE**

```
__s32    GUI_LyrWinGetScnWindow(H_LYR h_lyr, RECT * rect);
```

➢ **ARGUMENTS**

h_lyr 图层句柄

rect 屏幕区域矩形指针

➢ **RETURNS**

ORANGE_OK 获取成功

ORANGE_FAIL 获取失败

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

- **DESCRIPTION**

  获取图层显示区域在屏幕中的位置窗口。

26.2.9.48. GUI_LyrMove

- **PROTOTYPE**

```
__s32   GUI_LyrMove(H_LYR h_lyr, __s32 x, __s32 y);
```

- **ARGUMENTS**

  h_lyr　　图层句柄
  x　　x 方向移动的相对坐标
  y　　y 方向移动的相对坐标

- **RETURNS**

  ORANGE_OK　　设置成功
  ORANGE_FAIL　　设置失败

- **DESCRIPTION**

  在屏幕中移动图层。此时需要向窗口发送消息

26.2.9.49. GUI_LyrWinSetFB

- **PROTOTYPE**

```
__s32   GUI_LyrWinSetFB(H_LYR h_lyr, const FB * fb);
```

- **ARGUMENTS**

  h_lyr　　图层句柄
  fb　　指向需设定的 FB 结构体

- **RETURNS**

  ORANGE_OK　　设置成功
  ORANGE_FAIL　　设置失败

- **DESCRIPTION**

  设置图层的 fb，此函数和下面的 get 函数作为特定场合使用的，一般应用程序不会使用这组接口。

26.2.9.50. GUI_LyrWinGetFB

- **PROTOTYPE**

```
__s32   GUI_LyrWinGetFB(H_LYR h_lyr, FB * fb);
```

- **ARGUMENTS**

  h_lyr　　图层句柄
  fb　　指向获得的 FB 结构体

- **RETURNS**

  ORANGE_OK　　获取成功
  ORANGE_FAIL　　获取失败

- **DESCRIPTION**

  获取图层的 framebuffer

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.2.9.51. GUI_LyrWinSel

- **PROTOTYPE**

```
__s32    GUI_LyrWinSel(H_LYR h_lyr);
```

- **ARGUMENTS**

h_lyr　　图层句柄

- **RETURNS**

ORANGE_OK　　设置成功

ORANGE_FAIL　　设置失败

- **DESCRIPTION**

选择图层为当前操作图层

26.2.9.52. GUI_LyrWinSetFocus

- **PROTOTYPE**

```
__s32    GUI_LyrWinSetFocus(H_LYR h_lyr);
```

- **ARGUMENTS**

h_lyr　　图层句柄

- **RETURNS**

ORANGE_OK　　设置成功

ORANGE_FAIL　　设置失败

- **DESCRIPTION**

将图层设置为焦点图层

26.2.9.53. GUI_LyrWinCacheOn

- **PROTOTYPE**

```
__s32    GUI_LyrWinCacheOn(void);
```

- **ARGUMENTS**

None

- **RETURNS**

ORANGE_OK　　设置成功

ORANGE_FAIL　　设置失败

- **DESCRIPTION**

打开命令 cache

26.2.9.54. GUI_LyrWinCacheOff

- **PROTOTYPE**

```
__s32    GUI_LyrWinCacheOff(void);
```

- **ARGUMENTS**

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

None

- **RETURNS**

ORANGE_OK　　设置成功
ORANGE_FAIL　　设置失败

- **DESCRIPTION**

执行 cache 里的所有的命令，并停止 cache。

26.2.9.55. GUI_LyrWinSetTop

- **PROTOTYPE**

```
__s32    GUI_LyrWinSetTop(H_LYR h_lyr);
```

- **ARGUMENTS**

h_lyr　　图层句柄

- **RETURNS**

ORANGE_OK　　设置成功
ORANGE_FAIL　　设置失败

- **DESCRIPTION**

将图层设置到通道的顶部

26.2.9.56. GUI_LyrWinSetBottom

- **PROTOTYPE**

```
__s32    GUI_LyrWinSetBottom(H_LYR h_lyr);
```

- **ARGUMENTS**

h_lyr　　图层句柄

- **RETURNS**

ORANGE_OK　　设置成功
ORANGE_FAIL　　设置失败

- **DESCRIPTION**

将图层设置到通道的底部

26.2.9.57. GUI_LyrWinCKOn

- **PROTOTYPE**

```
__s32 GUI_LyrWinCKOn(H_LYR h_lyr);
```

- **ARGUMENTS**

h_lyr　　图层句柄

- **RETURNS**

ORANGE_OK　　设置成功
ORANGE_FAIL　　设置失败

- **DESCRIPTION**

打开图层的 colorkey 功能

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.2.9.58. GUI_LyrWinCKOff

- **PROTOTYPE**

```
__s32 GUI_LyrWinCKOff(H_LYR h_lyr);
```

- **ARGUMENTS**

h_lyr　　图层句柄

- **RETURNS**

ORANGE_OK　　设置成功
ORANGE_FAIL　　设置失败

- **DESCRIPTION**

关闭 Color key 功能

26.2.9.59. GUI_LyrWinGetSta

- **PROTOTYPE**

```
__gui_lyr_sta_t GUI_LyrWinGetSta(H_LYR h_lyr);
```

- **ARGUMENTS**

h_lyr　　图层句柄

- **RETURNS**

图层状态信息

- **DESCRIPTION**

获取图层的状态

26.2.9.60. GUI_LyrWinSetSta

- **PROTOTYPE**

```
__s32    GUI_LyrWinSetSta(H_LYR h_lyr, __gui_lyr_sta_t status);
```

- **ARGUMENTS**

h_lyr　　图层句柄
status　　图层状态

- **RETURNS**

ORANGE_OK　　设置成功
ORANGE_FAIL　　设置失败

- **DESCRIPTION**

设置图层的状态

26.3. Message

消息管理器主要用来实现窗口的消息分发和处理，包括外部事件发给窗口管理器的消息，和窗口管理器内部触发的消息两种。

- 外部事件消息：主要指由外设触发而由输入设备传递给 GUI 消息接收器的消息，如键盘，鼠标等消息。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

- GUI 系统消息：主要指由外部事件或相应的函数触发的消息，如绘制，大小改变，设焦，等相关的消息。
- GUI 根据消息的优先级的不同又分为同步消息，异步消息，通知消息。
- 对同步消息而言，发送线程需要等消息执行完成之后再返回，同步消息往往用来处理高优先级的消息。同步消息分为两种情况，一种是直接窗口的回调函数，另一种指消息的处理线程和发送线程并不是同一个线程，此时需要等待同步消息信号量 post 之后才能返回。
- 异步消息往往用于非高优先级的消息，对异步消息而言，发送线程只需要将消息投递到指定的消息队列中之后然后返回，并不等消息完全执行完毕。
- Message Router

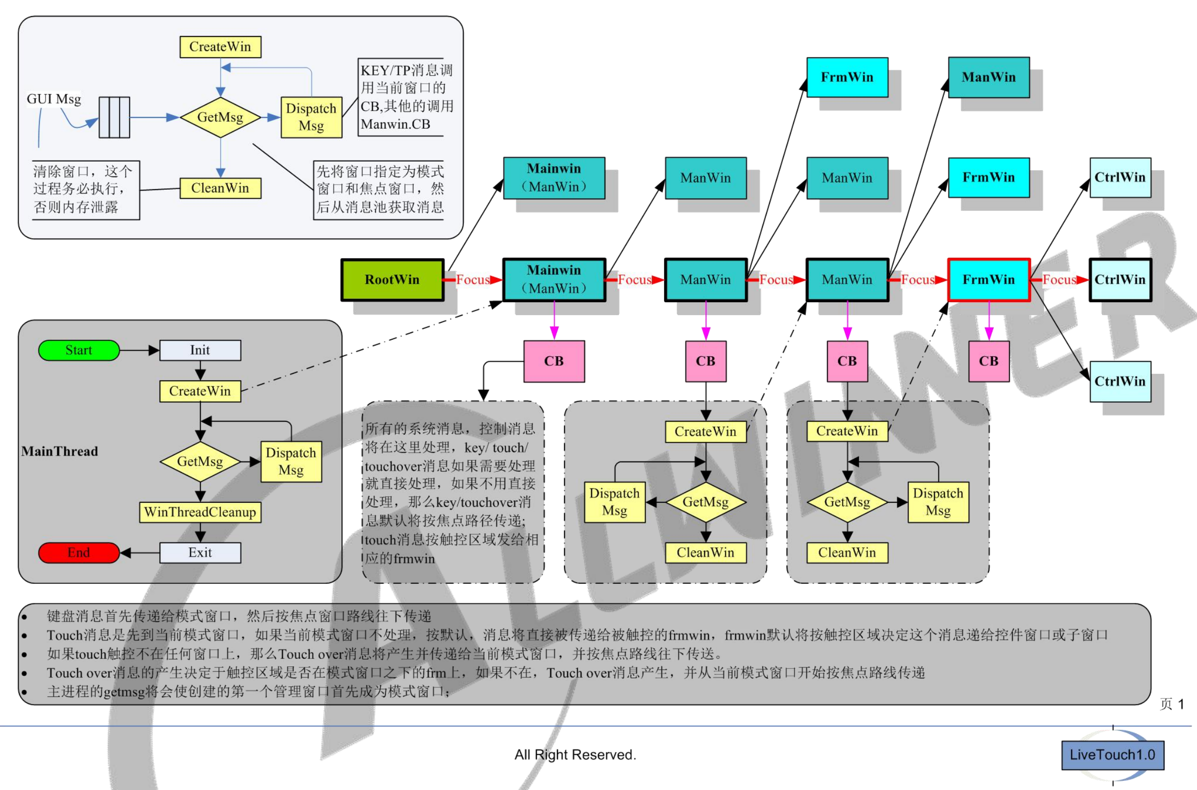


图 3.1 Message router

26.3.1. Interface

26.3.1.1. GUI_NotifyMSGQ

- **PROTOTYPE**

```
void GUI_NotifyMSGQ(__win_msgqueue_t *qmsg);
```

- **ARGUMENTS**

qmsg　　指向需通知的消息队列

- **RETURNS**

None

- **DESCRIPTION**

通知相应的消息队列，有消息来到。

26.3.1.2. GUI_SendMessage

- **PROTOTYPE**

```
__s32 GUI_SendMessage(__gui_msg_t *msg);
```

- **ARGUMENTS**

msg 指向需发送的消息

- **RETURNS**

ORANGE_FAIL 发送失败；否则返回该消息目标窗口回调函数的返回值。

- **DESCRIPTION**

发送消息。如果该消息的目标窗口所在的线程和当前线程是同一线程，则直接发送该消息到目标窗口的回调函数，否则发送同步消息到目标窗口所在线程的消息队列。

26.3.1.3. GUI_PostMessage

- **PROTOTYPE**

```
__s32 GUI_PostMessage(__gui_msg_t *msg);
```

- **ARGUMENTS**

msg 指向需投递的消息

- **RETURNS**

ORANGE_FAIL 投递失败

ORANGE_OK 投递成功

- **DESCRIPTION**

投递消息到消息的目标窗口所在的消息队列。

26.3.1.4. GUI_SendNotifyMessage

- **PROTOTYPE**

```
__s32 GUI_SendNotifyMessage(__gui_msg_t *msg);
```

- **ARGUMENTS**

msg 指向需发送的通知消息

- **RETURNS**

ORANGE_FAIL 发送失败

ORANGE_OK 发送成功

- **DESCRIPTION**

发送通知消息到该消息的目标窗口所在的消息队列。

26.3.1.5. GUI_GetMessageEx

- **PROTOTYPE**

```
__s32 GUI_GetMessageEx(__gui_msg_t *msg, H_WIN    hManWin);
```

- **ARGUMENTS**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

msg　　指向取到的消息

hMainWin　　管理窗口句柄

➢ **RETURNS**

ORANGE_FAIL 取消息失败

ORANGE_TRUE　　取消息成功

➢ **DESCRIPTION**

从管理窗口 *hManWin* 对应的消息队列中取消息。

26.3.1.6. GUI_MsgSetRepeatTimes

➢ **PROTOTYPE**

```
__s32 GUI_MsgSetRepeatTimes(H_WIN    hManWin,__u32 count);
```

➢ **ARGUMENTS**

hManWin　　管理窗口句柄

count　　长按键计数次数

➢ **RETURNS**

ORANGE_FAIL 设置失败

ORANGE_OK　设置成功

➢ **DESCRIPTION**

设置管理窗口 *hManWin* 对应的消息队列的长按键计数次数。

26.3.1.7. GUI_DispatchMessage

➢ **PROTOTYPE**

```
__s32 GUI_DispatchMessage(__gui_msg_t *msg);
```

➢ **ARGUMENTS**

msg　　指向待派发的消息

➢ **RETURNS**

HWND_INVALID　　失败，否则返回该消息目标窗口回调函数的返回值

➢ **DESCRIPTION**

派发 *msg* 到该消息对应的目标窗口的回调函数。

26.3.1.8. GUI_SendAsyncMessage

➢ **PROTOTYPE**

```
__s32 GUI_SendAsyncMessage(__gui_msg_t *msg);
```

➢ **ARGUMENTS**

msg　　指向待发送的消息

➢ **RETURNS**

ORANGE_FAIL 失败，否则返回该消息对应的目标窗口的回调函数的返回值

➢ **DESCRIPTION**

发送异步消息。

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

26.3.1.9. GUI_ThrowAwayMessages

- **PROTOTYPE**

```
__s32 GUI_ThrowAwayMessages (H_WIN hWnd);
```

- **ARGUMENTS**

hWnd　窗口句柄

- **RETURNS**

ORANGE_FAIL 失败，否则返回丢掉的消息个数

- **DESCRIPTION**

丢掉窗口 *hWnd* 对应的消息队列中的所有消息。

26.3.1.10. GUI_SetSyncMsgRetVal

- **PROTOTYPE**

```
__s32 GUI_SetSyncMsgRetVal(__gui_msg_t *msg,__s32 ret);
```

- **ARGUMENTS**

msg　指向需设置返回值的消息

ret　返回值

- **RETURNS**

ORANGE_FAIL 失败

ORANGE_OK　成功

- **DESCRIPTION**

设置同步消息的返回值。

26.3.1.11. GUI_PostSyncSem

- **PROTOTYPE**

```
__s32      GUI_PostSyncSem (__gui_msg_t *msg);
```

- **ARGUMENTS**

msg　指向同步消息

- **RETURNS**

ORANGE_FAIL 失败

ORANGE_OK　成功

- **DESCRIPTION**

释放同步消息的信号量。

26.4. Core

本部分完成 Orange1.0 的绘制和显示功能以及对多线程的处理。其中包括：2D 图形库的绘制、文本显示、绘图、设置字体和颜色、多线程的上下文切换、内存设备、2D 加速等。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.4.1. 2D Graphic Library

26.4.1.1. GUI_ClearRect

- **PROTOTYPE**

```
void GUI_ClearRect(int x0, int y0, int x1, int y1);
```

- **ARGUMENTS**

  x0 矩形左上角 x 坐标.

  y0 矩形左上角 y 坐标.

  x1 矩形右下角 x 坐标.

  y1 矩形右下角 y 坐标

- **RETURNS**

  None

- **DESCRIPTION**

  用背景色去填充当前图层上指定的矩形.

26.4.1.2. GUI_DrawPixel

- **PROTOTYPE**

```
void GUI_DrawPixel(int x, int y);
```

- **ARGUMENTS**

  x 指定像素的 x 坐标.

  y 指定像素的 y 坐标.

- **RETURNS**

  None

- **DESCRIPTION**

  在当前图层的指定位置处渲染一个像素.

26.4.1.3. GUI_DrawPoint

- **PROTOTYPE**

```
void GUI_DrawPoint(int x, int y);
```

- **ARGUMENTS**

  x 绘制点的 x 坐标.

  y 绘制点的 y 坐标.

- **RETURNS**

  None

- **DESCRIPTION**

  用当前的画刷在当前图层的指定位置处绘制一个点.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.4.1.4. GUI_DrawRect

- **PROTOTYPE**

```
void GUI_DrawRect(int x0, int y0, int x1, int y1);
```

- **ARGUMENTS**

x0 矩形左上角 x 坐标.
y0 矩形左上角 y 坐标.
x1 矩形右下角 x 坐标.
y1 矩形右下角 x 坐标.

- **RETURNS**

None

- **DESCRIPTION**

在当前图层的指定位置绘制一个矩形.

26.4.1.5. GUI_DrawRectEx

- **PROTOTYPE**

```
void GUI_DrawRectEx(const GUI_RECT *pRect);
```

- **ARGUMENTS**

pRect 指向一个 GUI_RECT 结构体的矩形，该矩形包括了待绘制矩形在图层上的坐标位置。

- **RETURNS**

None

- **DESCRIPTION**

在当前图层的指定位置绘制一个矩形.

26.4.1.6. GUI_DrawHLine

- **PROTOTYPE**

```
void GUI_DrawHLine(int y, int x0, int x1);
```

- **ARGUMENTS**

y 水平直线在当前图层 y 方向上的位置.
x0 水平直线在当前图层 x 方向上的起始位置.
x1 水平直线在当前图层 x 方向上的终止位置.

- **RETURNS**

None

- **DESCRIPTION**

在当前图层绘制一条水平直线，如果 x1<x0，不进行任何绘制。对于大多数 LCD 控制器，该函数被执行的非常快；该函数画水平直线的话比 GUI_DrawLine()函数快。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.4.1.7. GUI_DrawLine

- **PROTOTYPE**

```
void GUI_DrawLine(int x0, int y0, int x1, int y1);
```

- **ARGUMENTS**

x0 直线在当前图层 x 方向上的起始位置.
y0 直线在当前图层 y 方向上的起始位置.
x1 直线在当前图层 x 方向上的终止位置.
y1 直线在当前图层 y 方向上的终止位置.

- **RETURNS**

None

- **DESCRIPTION**

在当前图层上从指定的起点到终点绘制一条直线.

26.4.1.8. GUI_DrawLineRel

- **PROTOTYPE**

```
void GUI_DrawLineRel(int dx, int dy);
```

- **ARGUMENTS**

dx x 方向上的相对坐标.
dy y 方向上的相对坐标.

- **RETURNS**

None

- **DESCRIPTION**

绘制一条直线，该直线的起点是当前位置，终点由相对于起点位置的相对坐标(dx, dy)指定.

26.4.1.9. GUI_DrawLineTo

- **PROTOTYPE**

```
void GUI_DrawLineTo(int x, int y);
```

- **ARGUMENTS**

x x 方向上的终点坐标.
y y 方向上的终点坐标.

- **RETURNS**

None

- **DESCRIPTION**

在当前图层绘制一条直线，该直线的起点是当前位置，终点坐标是(x,y).

26.4.1.10. GUI_DrawPolyLine

- **PROTOTYPE**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
void GUI_DrawPolyLine(const GUI_POINT* pPoint, int NumPoints, int x0, int y0);
```

- **ARGUMENTS**

pPoint　　指向折线的拐点构成的数组.
NumPoints　拐点个数.
x0　　　指定 x 方向上的原点坐标.
y0　　　指定 y 方向上的原点坐标.

- **RETURNS**

None

- **DESCRIPTION**

在当前图层绘制折线，该折线的拐点由 pPoint 数组中的 point 元素指定，原点由(x0, y0)指定，若 x0=0，y0 = 0，则折线的起始点由 pPoint 数组中的第一个点开始，否则由 pPoint[0].x + x0， 和 pPoint[0].y + y0 作为起始点.

26.4.1.11. GUI_DrawVLine

- **PROTOTYPE**

```
void GUI_DrawVLine(int x, int y0, int y1);
```

- **ARGUMENTS**

x　垂直直线在当前图层 x 方向上的位置.
y0　垂直直线在当前图层 y 方向上的起始位置.
y1　垂直直线在当前图层 y 方向上的终止位置.

- **RETURNS**

None

- **DESCRIPTION**

在当前图层绘制一条垂直直线，如果 y1<y0，不进行任何绘制。对于大多数 LCD 控制器，该函数被执行的非常快；该函数画垂直直线的话比 GUI_DrawLine()函数快.

26.4.1.12. GUI_GetLineStyle

- **PROTOTYPE**

```
U8 GUI_GetLineStyle (void);
```

- **ARGUMENTS**

None

- **RETURNS**

正在使用的绘制风格.

- **DESCRIPTION**

获得正在使用的绘制风格(line 的绘制风格，该绘制风格的设定只有 pen size 为 1 时有效).

26.4.1.13. GUI_MoveRel

- **PROTOTYPE**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
void GUI_MoveRel(int dx, int dy);
```

- **ARGUMENTS**

dx　x 方向上的相对距离.

dy　y 方向上的相对距离.

- **RETURNS**

None

- **DESCRIPTION**

将绘制点的当前位置移动到相对于当前位置的一个坐标点(dx, dy).

26.4.1.14. GUI_MoveTo

- **PROTOTYPE**

```
void GUI_MoveTo(int x, int y);
```

- **ARGUMENTS**

x　x 方向上新的坐标位置.

y　y 方向上新的坐标位置.

- **RETURNS**

None

- **DESCRIPTION**

将绘制点的当前位置移动到坐标点(x, y)处.

26.4.1.15. GUI_SetLineStyle

- **PROTOTYPE**

```
U8 GUI_SetLineStyle(U8 LineStyle);
```

- **ARGUMENTS**

LineStyle　被设定的线的绘制风格(see table below).

GUI_LS_SOLID	实线绘制(default).
GUI_LS_DASH	折线绘制.
GUI_LS_DOT	点线绘制.
GUI_LS_DASHDOT	折线和点线交替绘制.
GUI_LS_DASHDOTDOT	折线和连续的两个点线交替绘制.

- **RETURNS**

老的绘制风格

- **DESCRIPTION**

设置当前线的绘制风格，被使用在 GUI_DrawLine 接口中，该绘制风格的设定只有 pen size 为 1 时有效.

26.4.1.16. GUI_DrawPolygon

- **PROTOTYPE**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
void GUI_DrawPolygon(const GUI_POINT* pPoint, int NumPoints, int x, int y);
```

- **ARGUMENTS**

pPoint 指向多边形各顶点构成的 point 数组.
NumPoints 顶点数.
x 原点在 x 方向上的位置.
y 原点在 y 方向上的位置.

- **RETURNS**

None

- **DESCRIPTION**

在当前图层绘制多边形.

26.4.1.17. GUI_EnlargePolygon

- **PROTOTYPE**

```
void GUI_EnlargePolygon(GUI_POINT* pDest, const GUI_POINT* pSrc, int NumPoints, int Len);
```

- **ARGUMENTS**

pDest 指向目标多边形.
pSrc 指向源多边形.
NumPoints 多边形顶点个数.
Len 多边形放大的长度，以像素为单位.

- **RETURNS**

None

- **DESCRIPTION**

放大多边形，注意：要保证目标数组等于或大于源数组.

26.4.1.18. GUI_FillPolygon

- **PROTOTYPE**

```
void GUI_FillPolygon(const GUI_POINT* pPoint, int NumPoints, int x, int y);
```

- **ARGUMENTS**

pPoint 指向被填充的多边形.
NumPoints 多边形顶点数.
x 原点在 x 方向上的位置.
y 原点在 y 方向上的位置.

- **RETURNS**

None

- **DESCRIPTION**

在当前图层绘制一个被填充了的多边形，填充颜色由 GUI_SetColor 接口指定.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.4.1.19. GUI_DrawCircle

➢ **PROTOTYPE**

```
void GUI_DrawCircle(int x0, int y0, int r);
```

➢ **ARGUMENTS**

x0 圆心在 x 方向上的位置.

y0 圆心在 y 方向上的位置.

r 半径. 最小值：0(一个点)；最大值 180.

➢ **RETURNS**

None

➢ **DESCRIPTION**

在当前图层绘制圆形.

26.4.1.20. GUI_FillCircle

➢ **PROTOTYPE**

```
void GUI_FillCircle(int x0, int y0, int r);
```

➢ **ARGUMENTS**

x0 圆心在 x 方向上的位置.

y0 圆心在 y 方向上的位置.

r 半径. 最小值：0(一个点)；最大值 180.

➢ **RETURNS**

None

➢ **DESCRIPTION**

在当前图层绘制一个被填充了的圆形，填充颜色由 GUI_SetColor 接口指定。该函数不能处理半径超过 180 个像素的圆.

26.4.1.21. GUI_DrawArc

➢ **PROTOTYPE**

```
void GL_DrawArc (int xCenter, int yCenter, int rx, int ry, int a0, int a1);
```

➢ **ARGUMENTS**

xCenter 弧形中心在 x 方向上的位置.

yCenter 弧形中心在 y 方向上的位置.

rx X 方向上的半径(pixels).

ry Y 方向上的半径(pixels)，暂未用.

a0 起始角度(degrees).

a1 终止角度 (degrees).

➢ **RETURNS**

None

➢ **DESCRIPTION**

在当前图层绘制一个弧形，该弧形的半径不能超过 180 个像素.

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

26.4.1.22. GUI_DrawGraph

- **PROTOTYPE**

```
void GUI_DrawGraph(I16 *paY, int NumPoints, int x0, int y0);
```

- **ARGUMENTS**

paY	指向包含图形 Y 值的一个数组.
NumPoints	Y 值的个数.
x0	X 方向上的起始点.
y0	Y 方向上的起始点.

- **RETURNS**

None

- **DESCRIPTION**

在当前图层绘制一张图. 该函数首先通过 x0、y0 和给定数组 paY 中的第一个 Y 值设定当前的显示位置，然后按点(x0 + 1, y0 + *(paY +1))、 点(x0 + 2, y0 + *(paY + 2))依次绘制直线.

26.4.1.23. GUI_RestoreContext

- **PROTOTYPE**

```
void GUI_RestoreContext(const GUI_CONTEXT* pContext);
```

- **ARGUMENTS**

pContext 指向包含新内容的一个 GUI_CONTEXT 结构体.

- **RETURNS**

None

- **DESCRIPTION**

设置 pContext 为当前使用的 GUI Context.

26.4.1.24. GUI_SaveContext

- **PROTOTYPE**

```
void GUI_SaveContext(GUI_CONTEXT* pContext);
```

- **ARGUMENTS**

pContext 指向保存当前内容的一个 GUI_CONTEXT 结构体.

- **RETURNS**

None

- **DESCRIPTION**

保存当前 GUI Context. (See also GUI_RestoreContext).

26.4.1.25. GUI_SetClipRect

- **PROTOTYPE**

```
void GUI_SetClipRect(const GUI_RECT* pRect);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

- **ARGUMENTS**

pRect 指向被用于剪切的矩形，如果是 NULL 则恢复默认的剪切域.

- **RETURNS**

None

- **DESCRIPTION**

为了限制输出范围设置剪切矩形.

26.4.2. Displaying Text

26.4.2.1. GUI_DispChar

- **PROTOTYPE**

```
void GUI_DispChar(U16 c);
```

- **ARGUMENTS**

c 显示的字符.

- **RETURNS**

None

- **DESCRIPTION**

用当前字体在当前图层的当前显示位置绘制单个字符.

26.4.2.2. GUI_DispCharAt

- **PROTOTYPE**

```
void GUI_DispCharAt(U16 c, I16P x, I16P y);
```

- **ARGUMENTS**

c 显示的字符.

x 指定 x 方向上的显示位置.

y 指定 y 方向上的显示位置.

- **RETURNS**

None

- **DESCRIPTION**

用当前字体在当前图层的指定位置绘制单个字符.

26.4.2.3. GUI_DispChars

- **PROTOTYPE**

```
void GUI_DispChars(U16 c, int Cnt);
```

- **ARGUMENTS**

c 显示的字符.

Cnt 重复次数(0 <= Cnt <= 32767).

- **RETURNS**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

None

- **DESCRIPTION**

用当前字体在当前图层的指定位置重复绘制单个字符 Cnt 次.

26.4.2.4. GUI_DispNextLine

- **PROTOTYPE**

```
void GUI_DispNextLine(void);
```

- **ARGUMENTS**

None

- **RETURNS**

None

- **DESCRIPTION**

移动光标到下一行的开始.

26.4.2.5. GUI_DispString

- **PROTOTYPE**

```
void GUI_DispString(const char GUI_FAR *s);
```

- **ARGUMENTS**

s 显示的字符串.

- **RETURNS**

None

- **DESCRIPTION**

用当前字体在当前图层的当前文本显示位置显示一串字符串.

26.4.2.6. GUI_DispStringAt

- **PROTOTYPE**

```
void GUI_DispStringAt(const char GUI_FAR *s, int x, int y);
```

- **ARGUMENTS**

s 显示的字符串.

x 指定 x 方向上的显示位置.

y 指定 y 方向上的显示位置.

- **RETURNS**

None

- **DESCRIPTION**

用当前字体在当前图层的指定位置显示一串字符串.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.4.2.7. GUI_DispStringAtCEOL

- **PROTOTYPE**

```
void GUI_DispStringAtCEOL(const char GUI_UNI_PTR *s, int x, int y);
```

- **ARGUMENTS**

s　显示的字符串.
x　指定 x 方向上的显示位置.
y　指定 y 方向上的显示位置.

- **RETURNS**

None

- **DESCRIPTION**

该函数与 GUI_DispStringAt()接口的参数非常相近，它们都是用当前字体在当前图层的指定位置显示一串字符串，不同的是该函数会清除到行尾的剩余部分。例如某字符串覆盖原先的一行字符串，但该字符串的长度小于原先字符串的长度，这时就可以用该函数将该行剩余部分清除。

26.4.2.8. GUI_DispStringHCenterAt

- **PROTOTYPE**

```
void GUI_DispStringHCenterAt(const char GUI_FAR *s, int x, int y);
```

- **ARGUMENTS**

s　显示的字符串.
x　指定 x 方向上的显示位置.
y　指定 y 方向上的显示位置.

- **RETURNS**

None

- **DESCRIPTION**

用当前字体在当前图层的指定位置的水平中心处显示一串字符串.

26.4.2.9. GUI_DispStringInRect

- **PROTOTYPE**

```
void GUI_DispStringInRect(const char GUI_FAR *s, const GUI_RECT *pRect, int Align);
```

- **ARGUMENTS**

s　显示的字符串.
pRect　指定的矩形区域.
Align　对齐模式(可用的对齐模式见下表).

GUI_TA_LEFT	水平方向靠左
GUI_TA_RIGHT	水平方向靠右
GUI_TA_HCENTER	水平方向中间
GUI_TA_TOP	垂直方向靠上
GUI_TA_BOTTOM	垂直方向靠下

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

GUI_TA_VCENTER	垂直方向中间

➢ **RETURNS**

None

➢ **DESCRIPTION**

用当前字体在当前图层的指定矩形中按指定的对齐模式显示一串字符串.

26.4.2.10. GUI_DispStringInRectWrap

➢ **PROTOTYPE**

```
void GUI_DispStringInRectWrap(const char GUI_UNI_PTR * s, GUI_RECT * pRect, int TextAlign, GUI_WRAPMODE WrapMode);
```

➢ **ARGUMENTS**

s 显示的字符串.

pRect 指定的矩形区域.

Align 对齐模式(see table below).

GUI_TA_LEFT	水平方向靠左
GUI_TA_RIGHT	水平方向靠右
GUI_TA_HCENTER	水平方向中间
GUI_TA_TOP	垂直方向靠上
GUI_TA_BOTTOM	垂直方向靠下
GUI_TA_VCENTER	垂直方向中间

WrapMode 限制方式(see table below).

GUI_WRAPMODE_NONE	无限制.
GUI_WRAPMODE_WORD	Word 方式限制.
GUI_WRAPMODE_CHAR	Char 方式限制.

➢ **RETURNS**

None

➢ **DESCRIPTION**

用当前字体在当前图层的指定矩形中按指定的对齐模式和限制方式(可选的)显示一串字符串.

26.4.2.11. GUI_DispStringLen

➢ **PROTOTYPE**

```
void GUI_DispStringLen(const char GUI_UNI_PTR *s, int MaxNumChars);
```

➢ **ARGUMENTS**

s 显示的字符串. 该字符串必须是以\0 结束的 8bit 字符数组，允许是 NULL.

MaxNumChars显示的字符个数.

➢ **RETURNS**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

None

➢ **DESCRIPTION**

用当前字体在当前图层的当前显示位置显示指定数目的字符. 如果给定字符串的字符个数少于给定的数目则不足的数目由空格补齐; 如果多于给定的数目则只有给定数目的字符被显示.

26.4.2.12. GUI_GetTextMode

➢ **PROTOTYPE**

```
int GUI_GetTextMode(void);
```

➢ **ARGUMENTS**

None

➢ **RETURNS**

文本模式

➢ **DESCRIPTION**

获取当前文本模式.

26.4.2.13. GUI_SetTextMode

➢ **PROTOTYPE**

```
int GUI_SetTextMode(int TextMode);
```

➢ **ARGUMENTS**

TextMode 文本模式(see table below).

GUI_TEXTMODE_NORMAL	正常模式.
GUI_TEXTMODE_REVERSE	相反模式.
GUI_TEXTMODE_TRANSPARENT	透明模式.
GUI_TEXTMODE_XOR	异或模式.

➢ **RETURNS**

先前被选中的文本模式.

➢ **DESCRIPTION**

设置当前文本模式.

26.4.2.14. GUI_SetTextStyle()

➢ **PROTOTYPE**

```
char GUI_SetTextStyle(char Style);
```

➢ **ARGUMENTS**

Style 文本样式(see table below).

GUI_TS_NORMAL	普通样式 (default).
GUI_TS_UNDERLINE	下划线样式.
GUI_TS_STRIKETHRU	中间穿透样式.
GUI_TS_OVERLINE	上划线样式

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

➢ **RETURNS**

先前被选中的文本样式.

➢ **DESCRIPTION**

设置当前文本样式.

26.4.2.15. GUII_SetFontMode

➢ **PROTOTYPE**

```
GUI_FONTMODE GUI_SetFontMode(GUI_FONTMODE fm);
```

➢ **ARGUMENTS**

fm 字体模式

GUI_FONTMODE_8BPP32	用 8bpp 调色板的最后 32 个颜色值作为字体颜色
GUI_FONTMODE_8BPP256	用 8bpp 调色板的 256 个颜色值作为字体颜色
GUI_FONTMODE_8BPP128_1	用 8bpp 调色板的前 128 个颜色值作为字体颜色
GUI_FONTMODE_8BPP128_2	用 8bpp 调色板的后 128 个颜色值作为字体颜色.

➢ **RETURNS**

先前选中的字体模式.

➢ **DESCRIPTION**

仅用在 8bpp 颜色模式的图层上，来为字体设置调色板的颜色索引值.

26.4.2.16. GUI_GetTextAlign

➢ **PROTOTYPE**

```
int GUI_GetTextAlign(void);
```

➢ **ARGUMENTS**

None

➢ **RETURNS**

返回当前文本的对齐方式.

➢ **DESCRIPTION**

获取当前文本的对齐方式.

26.4.2.17. GUI_SetLBorder

➢ **PROTOTYPE**

```
void GUI_SetLBorder(int x);
```

➢ **ARGUMENTS**

X 新的左边界位置.

➢ **RETURNS**

None

➢ **DESCRIPTION**

设置当前图层换行时的左边界起始位置.

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

26.4.2.18. GUI_SetTextAlign

➢ **PROTOTYPE**

```
int GUI_SetTextAlign(int TextAlign);
```

➢ **ARGUMENTS**

TextAlign 文本对齐方式(see table below).

GUI_TA_LEFT	水平方向靠左
GUI_TA_RIGHT	水平方向靠右
GUI_TA_HCENTER	水平方向中间
GUI_TA_TOP	垂直方向靠上
GUI_TA_BOTTOM	垂直方向靠下
GUI_TA_VCENTER	垂直方向中间

➢ **RETURNS**

先前选中的文本对齐方式.

➢ **DESCRIPTION**

为字符串的输出设置文本对齐方式.

26.4.2.19. GUI_GotoXY, GUI_GotoX, GUI_GotoY

➢ **PROTOTYPE**

```
char GUI_GotoXY(int x, int y);
char GUI_GotoX(int x);
char GUI_GotoY(int y);
```

➢ **ARGUMENTS**

x　x 方向上新的位置(以像素为单位，如果是 0 则新位置为左边界).

y　y 方向上新的位置(以像素为单位，如果是 0 则新位置为上边界).

➢ **RETURNS**

通常返回 0，如果非 0，则当前显示位置超出了图层范围，那么接下来的显示操作会被忽略.

➢ **DESCRIPTION**

设置当前文本的显示位置.

GUI_GotoXY() 设置(x,y)的值.

GUI_GotoX()　只设置 x 的值，y 值保持不变.

GUI_GotoY()　只设置 y 的值，x 值保持不变.

26.4.2.20. GUI_GetDispPosX

➢ **PROTOTYPE**

```
int GUI_GetDispPosX(void);
```

➢ **ARGUMENTS**

None

➢ **RETURNS**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

当前 x 方向的显示位置.

- **DESCRIPTION**

获取当前 x 方向的显示位置.

26.4.2.21. GUI_GetDispPosY

- **PROTOTYPE**

```
int GUI_GetDispPosY(void);
```

- **ARGUMENTS**

None

- **RETURNS**

当前 y 方向的显示位置.

- **DESCRIPTION**

获取当前 y 方向的显示位置.

26.4.2.22. GUI_Clear()

- **PROTOTYPE**

```
void GUI_Clear(void);
```

- **ARGUMENTS**

None

- **RETURNS**

None

- **DESCRIPTION**

用背景色清除当前图层.

26.4.2.23. GUI_DispCEOL

- **PROTOTYPE**

```
void GUI_DispCEOL(void);
```

- **ARGUMENTS**

None

- **RETURNS**

None

- **DESCRIPTION**

用背景色清除从当前显示位置到该行行尾的一块矩形，该矩形的高是当前字体的高度.

26.4.2.24. GUI_GetDrawMode

- **PROTOTYPE**

```
GUI_DRAWMODE GUI_GetDrawMode(void);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

➢ **ARGUMENTS**

None

➢ **RETURNS**

当前选择的绘制模式

➢ **DESCRIPTION**

返回当前选择的绘制模式.

26.4.2.25. GUI_SetDrawMode

➢ **PROTOTYPE**

```
GUI_DRAWMODE GUI_SetDrawMode(GUI_DRAWMODE mode);
```

➢ **ARGUMENTS**

mode　绘制模式(see table below).

GUI_DM_NORMAL	正常绘制模式
GUI_DM_XOR	异或绘制模式
GUI_DM_TRANS	透明绘制模式
GUI_DM_REV	反转绘制模式

➢ **RETURNS**

先前被选中的绘制模式.

➢ **DESCRIPTION**

设置绘制模式.

26.4.2.26. GUI_ClearRect

➢ **PROTOTYPE**

```
void GUI_ClearRect(int x0, int y0, int x1, int y1);
```

➢ **ARGUMENTS**

x0　矩形左上角 x 坐标.

y0　矩形左上角 y 坐标

x1　矩形右下角 x 坐标.

y1　矩形右下角 y 坐标

➢ **RETURNS**

None.

➢ **DESCRIPTION**

用背景色清除当前图层上的一块区域，该区域由左上角坐标(x0, y0)和右下角坐标(x1, y1)指定.

26.4.3. Bitmap Drawing

目前 Orange1.0 只支持 BMP 格式图片的绘制，可以绘制 1bpp、2bpp、4bpp、8bpp、16bpp、32bpp 格式的 BMP 图片.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.4.3.1. GUI_BMP_Draw

➢ **PROTOTYPE**

```
int GUI_BMP_Draw(const void* pBMP, int x0, int y0);
```

➢ **ARGUMENTS**

pBMP　指向存放 bmp 文件的 buffer 的起始地址.

x0　当前图层上绘制 bmp 图片的起始位置的 x 坐标.

y0　当前图层上绘制 bmp 图片的起始位置的 y 坐标.

➢ **RETURNS**

ORANGE_OK　成功，否则失败.

➢ **DESCRIPTION**

在当前图层的指定位置处绘制 BMP 文件.

26.4.3.2. GUI_BitString_Draw

➢ **PROTOTYPE**

```
int GUI_BitString_Draw(FB* fb, int x0, int y0);
```

➢ **ARGUMENTS**

fb　指向 FB 结构体.

x0　当前图层上绘制 bmp 图片的起始位置的 x 坐标.

y0　当前图层上绘制 bmp 图片的起始位置的 y 坐标.

➢ **RETURNS**

ORANGE_OK　成功，否则失败.

➢ **DESCRIPTION**

在当前图层的指定位置上绘制一张图片，该图片的数据 buffer 由结构体 FB 中的 addr[0]指定.

26.4.3.3. GUI_BitString_DrawEx

➢ **PROTOTYPE**

```
int GUI_BitString_DrawEx(FB* fb, int x0, int y0);
```

➢ **ARGUMENTS**

fb　指向 FB 结构体.

x0　当前图层上绘制 bmp 图片的起始位置的 x 坐标.

y0　当前图层上绘制 bmp 图片的起始位置的 y 坐标.

➢ **RETURNS**

ORANGE_OK　成功，否则失败.

➢ **DESCRIPTION**

同 GUI_BitString_Draw()功能相同，不同处是该函数是对 GUI_BitString_Draw()接口的加速，但是该函数没有进行边界检查，因此用户必须确保图片 buffer 不超出当前图层的 buffer.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.4.3.4. GUI_ARGB_Draw

- **PROTOTYPE**

```
int GUI_ARGB_Draw(const void * pBMP, int x0, int y0);
```

- **ARGUMENTS**

pBMP　指向存放 bmp 文件的 buffer 的起始地址.
x0　当前图层上绘制 bmp 图片的起始位置的 x 坐标.
y0　当前图层上绘制 bmp 图片的起始位置的 y 坐标.

- **RETURNS**

ORANGE_OK　成功，否则失败.

- **DESCRIPTION**

在当前图层的指定位置处绘制 BMP 文件.与 GUI_BMP_Draw()接口相比该函数在横屏绘制时更快.

26.4.3.5. GUI_ApplyAccelerateRotate

- **PROTOTYPE**

```
__s32 GUI_ApplyAccelerateRotate(void);
```

- **ARGUMENTS**

None

- **RETURNS**

ORANGE_OK　成功
ORANGE_FAIL 失败.

- **DESCRIPTION**

申请 VE 硬件加速，该函数主要用在竖屏绘制时对 GUI_BMP_Draw()和 GUI_ARGB_Draw()接口进行加速，注意，该接口只有在 VE 未被使用时才可进行申请.同时，VE 只对最小为 32*32 的图片进行加速，小于该 size 的图片的绘制用纯软件实现.

26.4.3.6. GUI_EndAccelerateRotate

- **PROTOTYPE**

```
__s32 GUI_EndAccelerateRotate(void);
```

- **ARGUMENTS**

None

- **RETURNS**

ORANGE_OK　成功
ORANGE_FAIL 失败.

- **DESCRIPTION**

结束 VE 硬件加速.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.4.4. Fonts

Orange1.0 支持 SFT 字体和 TTF 字体.

26.4.4.1. GUI_SFT_CreateFont

- **PROTOTYPE**

```
GUI_FONT * GUI_SFT_CreateFont( unsigned int    pixelSize,const char    *font_file );
```

- **ARGUMENTS**

pixelSize 字体大小(像素为单位)

font_file 字库文件

- **RETURNS**

创建成功 创建好的字体句柄.

创建失败 NULL.

- **DESCRIPTION**

创建一种点阵字体.

26.4.4.2. GUI_SFT_ReleaseFont

- **PROTOTYPE**

```
void GUI_SFT_ReleaseFont(GUI_FONT *pFont);
```

- **ARGUMENTS**

pFont 字体句柄

- **RETURNS**

None

- **DESCRIPTION**

释放某种点阵字体.

26.4.4.3. GUI_TTF_CreateFont

- **PROTOTYPE**

```
GUI_FONT *    GUI_TTF_CreateFont( GUI_TTF_ATTR * pTTF_ATTR);
```

- **ARGUMENTS**

pTTF_ATTR 矢量字体创建信息，包括矢量字体显示的大小和字库文件所在路径

- **RETURNS**

创建成功 创建好的字体句柄.

创建失败 NULL.

- **DESCRIPTION**

创建矢量字体.

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

26.4.4.4. GUI_TTF_Done

- **PROTOTYPE**

```
void GUI_TTF_Done(GUI_FONT *pFont);
```

- **ARGUMENTS**

pFont 字体句柄

- **RETURNS**

None

- **DESCRIPTION**

释放矢量字体.

26.4.4.5. GUI_GetFont

- **PROTOTYPE**

```
const GUI_FONT * GUI_GetFont(void);
```

- **ARGUMENTS**

None

- **RETURNS**

当前被选中的字体句柄.

- **DESCRIPTION**

获取当前被选中的字体句柄.

26.4.4.6. GUI_SetFont

- **PROTOTYPE**

```
const GUI_FONT * GUI_SetFont(const GUI_FONT * pNewFont);
```

- **ARGUMENTS**

pNewFont 字体句柄.

- **RETURNS**

先前被选中的字体句柄.

- **DESCRIPTION**

设置字体，一般用在文本显示输出的时候.

26.4.4.7. GUI_GetCharDistX

- **PROTOTYPE**

```
int GUI_GetCharDistX(U16 c);
```

- **ARGUMENTS**

c 指定的字符.

- **RETURNS**

字符宽度.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

➢ **DESCRIPTION**

获取指定字符的宽度，一般用于在当前选定的字体中显示一个指定的字符.

26.4.4.8. GUI_GetFontDistY

➢ **PROTOTYPE**

```
int GUI_GetFontDistY(void);
```

➢ **ARGUMENTS**

None

➢ **RETURNS**

当前选中字体的 Y-spacing.

➢ **DESCRIPTION**

获取当前选中字体的 Y-spacing (Y-spacing 指文本中相邻两行之间的垂直距离)

26.4.4.9. GUI_GetFontSizeY

➢ **PROTOTYPE**

```
int GUI_GetFontSizeY(void);
```

➢ **ARGUMENTS**

None

➢ **RETURNS**

当前选中字体的高度.

➢ **DESCRIPTION**

当前选中字体的高度.该高度小于或等于 Y-spacing.

26.4.4.10. GUI_GetStringDistX

➢ **PROTOTYPE**

```
int GUI_GetStringDistX(const char GUI_FAR *s);
```

➢ **ARGUMENTS**

s　　字符串.

➢ **RETURNS**

字符串的宽度.

➢ **DESCRIPTION**

在当前字体下，获取字符串的宽度.

26.4.4.11. GUI_GetTextExtend

➢ **PROTOTYPE**

```
void GUI_GetTextExtend(GUI_RECT* pRect, const char* s, int Len);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

➢ **ARGUMENTS**

pRect 指向存放显示区域 size 的 rect.

s 给定的字符串.

Len 字符串中字符的个数.

➢ **RETURNS**

None

➢ **DESCRIPTION**

在当前字体下，获取给定字符串显示区域的大小.

26.4.4.12. GUI_GetYDistOfFont

➢ **PROTOTYPE**

```
int GUI_GetYDistOfFont(const GUI_FONT* pFont);
```

➢ **ARGUMENTS**

pFont 字体句柄.

➢ **RETURNS**

指定字体的 Y- spacing.

➢ **DESCRIPTION**

获取指定字体的 Y- spacing.

26.4.4.13. GUI_GetYSizeOfFont

➢ **PROTOTYPE**

```
int GUI_GetYSizeOfFont(const GUI_FONT* pFont);
```

➢ **ARGUMENTS**

pFont 字体句柄.

➢ **RETURNS**

指定字体的高度.

➢ **DESCRIPTION**

获取指定字体的高度.

26.4.4.14. GUI_SetFrameColor8bpp32

➢ **PROTOTYPE**

```
U8 GUI_SetFrameColor8bpp32(U8 frameColor);
```

➢ **ARGUMENTS**

frameColor 字体边框颜色.

➢ **RETURNS**

先前的字体边框颜色.

➢ **DESCRIPTION**

设置字体边框颜色.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.4.5. Colors

Colors 按使用场合分为两种类型：(1)背景色；(2)前景色。背景色主要用于清背景时使用，其它场合均使用前景色，如字体的颜色等

26.4.5.1. GUI_GetBkColor

- **PROTOTYPE**

```
GUI_COLOR GUI_GetBkColor(void);
```

- **ARGUMENTS**

  None

- **RETURNS**

  当前的背景色.

- **DESCRIPTION**

  获取当前的背景色.

26.4.5.2. GUI_GetBkColorIndex

- **PROTOTYPE**

```
int GUI_GetBkColorIndex(void);
```

- **ARGUMENTS**

  None

- **RETURNS**

  当前背景色的索引值.

- **DESCRIPTION**

  获取当前背景色的索引值.

26.4.5.3. GUI_GetColor

- **PROTOTYPE**

```
GUI_COLOR GUI_GetColor(void);
```

- **ARGUMENTS**

  None

- **RETURNS**

  当前的前景色.

- **DESCRIPTION**

  获取当前的前景色.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.4.5.4. GUI_GetColorIndex

- **PROTOTYPE**

```
int GUI_GetColorIndex(void);
```

- **ARGUMENTS**

None

- **RETURNS**

当前前景色的索引值.

- **DESCRIPTION**

获取当前前景色的索引值.

26.4.5.5. GUI_SetBkColor

- **PROTOTYPE**

```
void GUI_SetBkColor(GUI_COLOR color);
```

- **ARGUMENTS**

Color 背景色.

- **RETURNS**

None.

- **DESCRIPTION**

设置当前的背景色.

26.4.5.6. GUI_SetBkColorIndex

- **PROTOTYPE**

```
void GUI_SetBkColorIndex(int Index);
```

- **ARGUMENTS**

Index 颜色索引值.

- **RETURNS**

None.

- **DESCRIPTION**

设置当前的背景色索引值

26.4.5.7. GUI_SetColor

- **PROTOTYPE**

```
void GUI_SetColor(GUI_COLOR Color);
```

- **ARGUMENTS**

Color 前景色

- **RETURNS**

None.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

➢ **DESCRIPTION**
设置当前的前景色.

26.4.5.8. GUI_SetColorIndex

➢ **PROTOTYPE**
```
void GUI_SetColorIndex(int Index);
```
➢ **ARGUMENTS**
Index　颜色索引值.
➢ **RETURNS**
None.
➢ **DESCRIPTION**
设置当前的前景色索引值.

26.4.5.9. GUI_Color2Index

➢ **PROTOTYPE**
```
int GUI_Color2Index(GUI_COLOR Color);
```
➢ **ARGUMENTS**
Color　颜色值.
➢ **RETURNS**
颜色索引值.
➢ **DESCRIPTION**
颜色值到颜色索引值的转换.

26.4.5.10. GUI_Index2Color

➢ **PROTOTYPE**
```
int GUI_Index2Color(int    index);
```
➢ **ARGUMENTS**
Index　颜色索引值
➢ **RETURNS**
颜色值.
➢ **DESCRIPTION**
颜色索引值到颜色值的转换.

26.4.6. Memory Device

内存设备的主要用途是用来防止绘制时的闪烁现象. Orange1.0 支持三种颜色深度的内存设备，分别是 8 bpp ,16bpp and 32 bpp. 内存设备的基本使用方法非常简单：

1. 创建内存设备(用 GUI_MEMDEV_Create()).

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

2. 激活内存设备(用 GUI_MEMDEV_Select()).
3. 执行绘制操作.
4. 复制绘制内容到显示设备(用 GUI_MEMDEV_CopyToLCD()).
5. 如果不再需要改内存设备则删除之(用 GUI_MEMDEV_Delete()).

26.4.6.1. GUI_MEMDEV_CopyToLCD

➢ **PROTOTYPE**

```
void GUI_MEMDEV_CopyToLCD(GUI_MEMDEV_Handle hMem);
```

➢ **ARGUMENTS**

hMem　内存设备句柄.

➢ **RETURNS**

None

➢ **DESCRIPTION**

复制内存设备内容到显示设备.

26.4.6.2. GUI_MEMDEV_CopyToLCDAt

➢ **PROTOTYPE**

```
void GUI_MEMDEV_CopyToLCDAt(GUI_MEMDEV_Handle hMem, int x, int y);
```

➢ **ARGUMENTS**

hMem　内存设备句柄.

x　显示设备 x 方向位置

y　显示设备 y 方向位置

➢ **RETURNS**

None

➢ **DESCRIPTION**

复制内存设备内容到显示设备的指定位置.

26.4.6.3. GUI_MEMDEV_Create

➢ **PROTOTYPE**

```
GUI_MEMDEV_Handle GUI_MEMDEV_Create(int x0, int y0, int XSize, int YSize);
```

➢ **ARGUMENTS**

x0　内存设备起始点 x 坐标.

y0　内存设备起始点 y 坐标.

xsize　内存设备起宽度.

ysize　内存设备起高度.

➢ **RETURNS**

内存设备句柄，如果为 0 则创建失败.

➢ **DESCRIPTION**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

创建内存设备.

26.4.6.4. GUI_MEMDEV_Delete

- **PROTOTYPE**

```
void GUI_MEMDEV_Delete(GUI_MEMDEV_Handle MemDev);
```

- **ARGUMENTS**

hMem　内存设备句柄.

- **RETURNS**

None

- **DESCRIPTION**

删除内存设备.

26.4.6.5. GUI_MEMDEV_GetXSize

- **PROTOTYPE**

```
int GUI_MEMDEV_GetXSize(GUI_MEMDEV_Handle hMem);
```

- **ARGUMENTS**

hMem　内存设备句柄.

- **RETURNS**

内存设备宽度.

- **DESCRIPTION**

获取内存设备宽度.

26.4.6.6. GUI_MEMDEV_GetYSize

- **PROTOTYPE**

```
int GUI_MEMDEV_GetYSize(GUI_MEMDEV_Handle hMem);
```

- **ARGUMENTS**

hMem　内存设备句柄.

- **RETURNS**

内存设备高度.

- **DESCRIPTION**

获取内存设备高度.

26.4.6.7. GUI_MEMDEV_Select

- **PROTOTYPE**

```
void GUI_MEMDEV_Select(GUI_MEMDEV_Handle hMem);
```

- **ARGUMENTS**

hMem　内存设备句柄.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

- **RETURNS**

  None

- **DESCRIPTION**

  激活一个内存设备，如果 hMem = 0，则激活显示设备.

26.4.6.8. GUI_MEMDEV_SetOrg

- **PROTOTYPE**

```
void GUI_MEMDEV_SetOrg(GUI_MEMDEV_Handle hMem, int x0, int y0);
```

- **ARGUMENTS**

  hMem　内存设备句柄
  x0　左上角像素水平位置
  y0　左上角像素垂直位置.

- **RETURNS**

  None

- **DESCRIPTION**

  改变内存设备在显示设备的原点位置. 当同样的内存设别内容被复制到显示设备的不同区域或者相同的内存设备被用于显示设备的不同区域时用该函数比删除当前内存设备再创建更有效.

26.4.7. 2D Accelerate

2D 加速是通过硬件对内存块进行操作，包括内存块的拷贝、alpha 及旋转等.

26.4.7.1. GUI_BlockCreate

- **PROTOTYPE**

```
HBLOCK GUI_BlockCreate(__u32 width,__u32 height, __u8 byte_seq);
```

- **ARGUMENTS**

  width　内存块宽度.
  height　内存块高度
  byte_seq 字节顺序流. 0: argb; 1: bgra

- **RETURNS**

  内存块句柄. 如果为 0 则创建失败.

- **DESCRIPTION**

  创建内存块.

26.4.7.2. GUI_BlockCreateFrom

- **PROTOTYPE**

```
HBLOCK GUI_BlockCreateFrom(RECT *block_rect, __u32 fb_width,__u32 fb_height,void *addr,__u8 byte_seq);
```

- **ARGUMENTS**

  block_rect　创建的内存块矩形的大小，如果为 NULL 则内存块矩形为整个 buffer
  fb_width　已分配 buffer 的宽度
  fb_height　已分配 buffer 的高度
  Addr　已分配 buffer 的起始地址
  byte_seq　字节顺序流. 0: argb; 1: bgra

- **RETURNS**

  内存块句柄. 如果为 0 则创建失败.

- **DESCRIPTION**

  从已分配的 buffer 创建内存块(见 图 4.7.1).

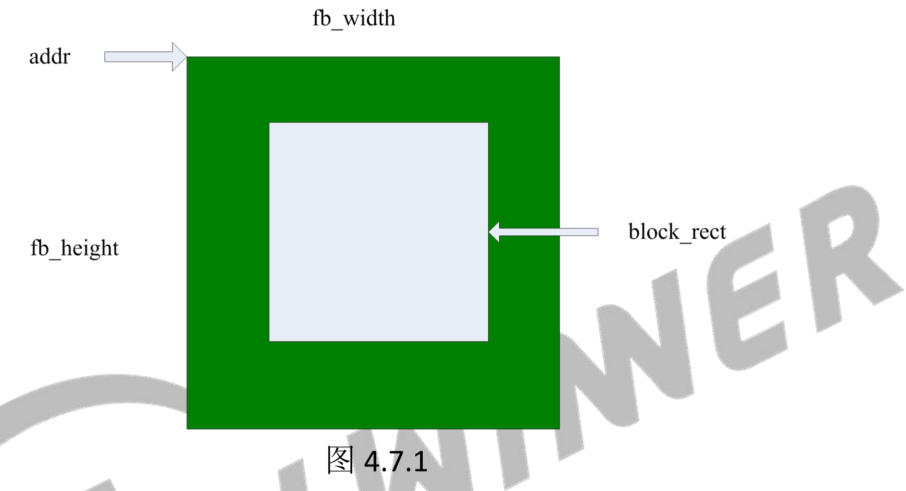


图 4.7.1

26.4.7.3. GUI_BlockDelete

- **PROTOTYPE**

```
__s32 GUI_BlockDelete(HBLOCK hblock);
```

- **ARGUMENTS**

  hblock　内存块句柄.

- **RETURNS**

  ORANGE_OK　成功
  ORANGE_FAIL　失败

- **DESCRIPTION**

  删除内存块.

26.4.7.4. GUI_BlockBitBlit

- **PROTOTYPE**

```
__s32 GUI_BlockBitBlit(HBLOCK hdstblock, __s32 dx, __s32 dy, HBLOCK hsrcblock,RECT *srcrect,__u32 flags, void *value);
```

- **ARGUMENTS**

  hdstblock　目标内存块句柄.
  dx　目标矩形区域左上角 x 坐标

dy 目标矩形区域左上角 y 坐标
hsrcblock 源内存块句柄
srcrect 源内存块操作的矩形区域，如果为 NULL 则为整个内存块
flags 位块操作标记(see table below)

BLITFLAG_COLORSET	复制
BLITFLAG_COLORALPHA	点 alpha
BLITFLAG_ALPHACHANNEL	面 alpha
BLITFLAG_BOTHALPHA	点 alpha 和面 alpha 的混合操作

value 设置面 alpha 值

➢ **RETURNS**

ORANGE_OK 成功
ORANGE_FAIL 失败

➢ **DESCRIPTION**

从源内存块到目标内存块颜色数据的位块转移. 点 alpha 指目标内存块每个像素颜色值 的 alpha, 面 alpha 由用户自己指定(见 图 4.7.2).

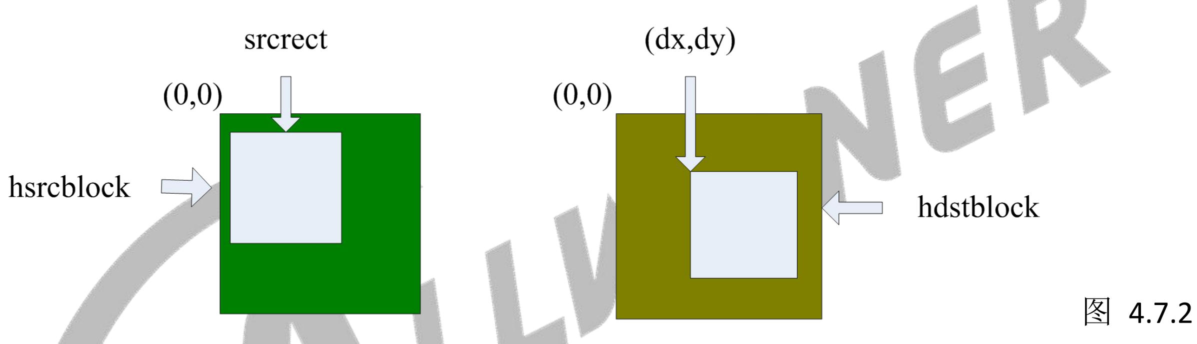


图 4.7.2

26.4.7.5. GUI_BlockLayerBlit

➢ **PROTOTYPE**

```
__s32 GUI_BlockLayerBlit(H_LYR hdstlayer,__s32 dx,__s32 dy, HBLOCK hsrcblock, RECT *srcrect,__u32 flags, void * value);
```

➢ **ARGUMENTS**

hdstlayer 目标图层句柄
dx 目标矩形区域左上角 x 坐标.
dy 目标矩形区域左上角 y 坐标
hsrcblock 源内存块句柄
srcrect 源内存块操作的矩形区域，如果为 NULL 则为整个内存块
flags 位块操作标记(see table below)

BLITFLAG_COLORSET	复制
BLITFLAG_COLORALPHA	点 alpha
BLITFLAG_ALPHACHANNEL	面 alpha
BLITFLAG_BOTHALPHA	点 alpha 和面 alpha 的混合操作

value 设置面 alpha 值

➢ **RETURNS**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

ORANGE_OK 成功
ORANGE_FAIL 失败

➢ **DESCRIPTION**

从源内存块到目标图层颜色数据的位块转移．点 alpha 指目标内存块每个像素颜色值的 alpha，面 alpha 由用户自己指定(见 图 4.7.3).

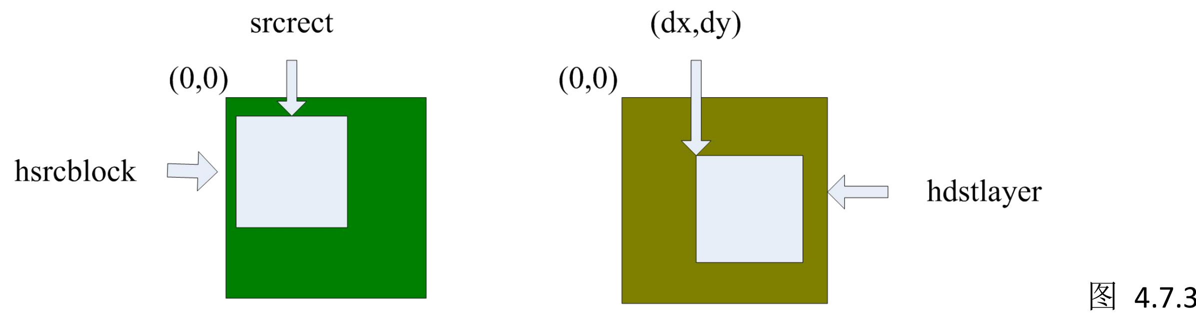


图 4.7.3

26.4.7.6. GUI_BlockGetLayerMem

➢ **PROTOTYPE**

```
__s32 GUI_BlockGetLayerMem(H_LYR hsrclayer, RECT *srcrect,HBLOCK hdstblock, __s32 dx,__s32 dy,__u32 flags, void * value);
```

➢ **ARGUMENTS**

hsrclayer 源图层句柄.
srcrect 源图层操作的矩形区域，如果为 NULL 则为整个图层.
hdstblock 目标内存块句柄
dx 目标矩形区域左上角 x 坐标
dy 目标矩形区域左上角 y 坐标
flags 位块操作标记(see table below)

BLITFLAG_COLORSET	复制
BLITFLAG_COLORALPHA	点 alpha
BLITFLAG_ALPHACHANNEL	面 alpha
BLITFLAG_BOTHALPHA	点 alpha 和面 alpha 的混合操作

value 设置面 alpha 值

➢ **RETURNS**

ORANGE_OK 成功
ORANGE_FAIL 失败

➢ **DESCRIPTION**

从源图层到目标内存块颜色数据的位块转移．点 alpha 指目标内存块每个像素颜色值的 alpha，面 alpha 由用户自己指定(见 图 4.7.4).

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

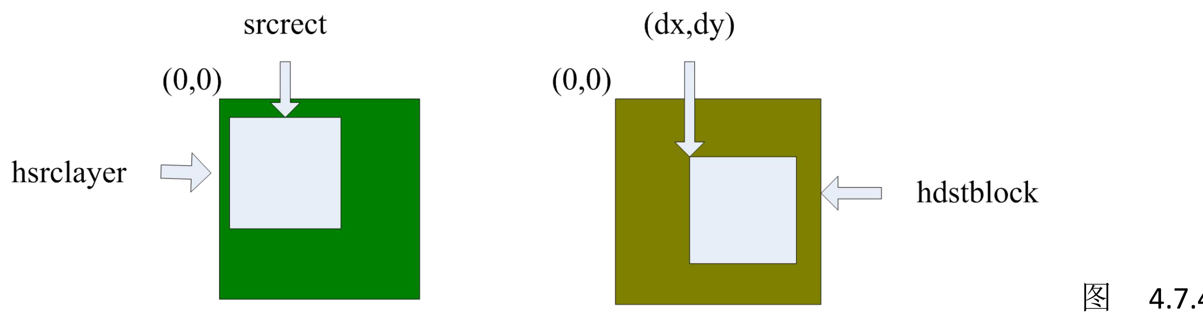


图 4.7.4

26.4.7.7. GUI_BlockRotate

➢ **PROTOTYPE**

```
__s32 GUI_BlockRotate(__u32 srcblock_addr, __u32 dstblock_addr, __u32 srcblock_fb_width, __u32 srcblock_fb_height, __u32 angle_flags);
```

➢ **ARGUMENTS**

srcblock_addr 源 buffer 起始地址
dstblock_addr 目标 buffer 起始地址
srcblock_fb_width 源 buffer 宽度
srcblock_fb_height 源 buffer 高度
angle_flags 旋转角度(see table below)

ROTATE_ANGLE_NONE	Rotate 0 angle
ROTATE_ANGLE_90	Rotate 90 angle
ROTATE_ANGLE_180	Rotate 180 angle
ROTATE_ANGLE_270	Rotate 270 angle
ROTATE_HFLIP	Horizontal Flip
ROTATE_VFLIP	Vertical Flip

➢ **RETURNS**

ORANGE_OK 成功
ORANGE_FAIL 失败

➢ **DESCRIPTION**

旋转一块源 buffer 到目标 buffer.

26.4.7.8. GUI_GetBlockWidth

➢ **PROTOTYPE**

```
__s32 GUI_GetBlockWidth(HBLOCK hblock);
```

➢ **ARGUMENTS**

hblock 内存块句柄.

➢ **RETURNS**

ORANGE_FAIL 失败，否则返回内存块的宽度.

➢ **DESCRIPTION**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

获取内存块的宽度(见 图 4.7.5).

26.4.7.9. GUI_GetBlockHeight

- **PROTOTYPE**

```
__s32 GUI_GetBlockHeight(HBLOCK hblock);
```

- **ARGUMENTS**

hblock 内存块句柄.

- **RETURNS**

ORANGE_FAIL 失败，否则返回内存块的高度.

- **DESCRIPTION**

获取内存块的高度(见 图 4.7.5).

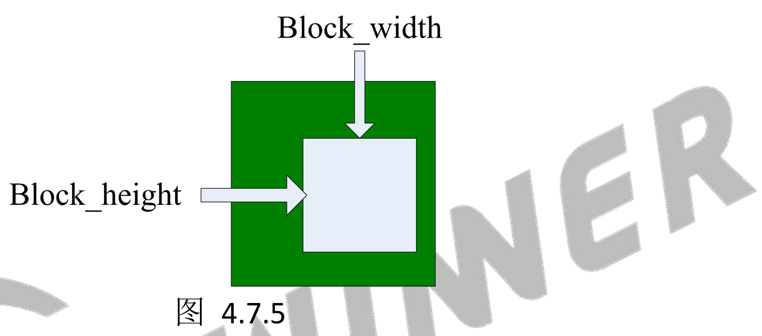


图 4.7.5

26.4.7.10. GUI_GetBlockAddr

- **PROTOTYPE**

```
__s32 GUI_GetBlockAddr(HBLOCK hblock);
```

- **ARGUMENTS**

hblock 内存块句柄.

- **RETURNS**

ORANGE_FAIL 失败，否则返回内存块 buffer 的起始地址

- **DESCRIPTION**

获取内存块 buffer 的起始地址(见 图 4.7.6).

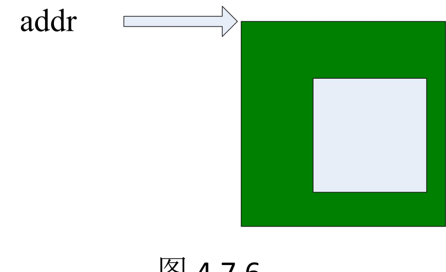


图 4.7.6

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.4.7.11. GUI_GetBlockFBWidth

➢ **PROTOTYPE**

```
__s32 GUI_GetBlockFBWidth(HBLOCK hblock);
```

➢ **ARGUMENTS**

hblock　内存块句柄.

➢ **RETURNS**

ORANGE_FAIL　　失败，否则返回内存块 buffer 的宽度

➢ **DESCRIPTION**

获取内存块 buffer 的宽度(见 图 4.7.7).

26.4.7.12. GUI_GetBlockFBHeight

➢ **PROTOTYPE**

```
__s32 GUI_GetBlockFBHeight(HBLOCK hblock);
```

➢ **ARGUMENTS**

hblock　内存块句柄.

➢ **RETURNS**

ORANGE_FAIL　失败，否则返回内存块 buffer 的高度

➢ **DESCRIPTION**

获取内存块 buffer 的高度(见 图 4.7.7).

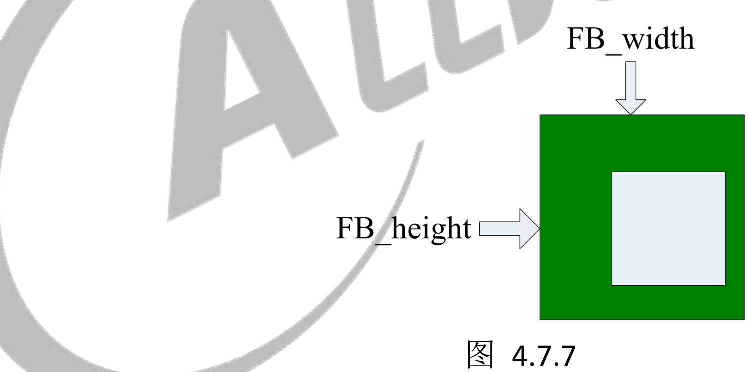


图 4.7.7

26.4.8. Other

26.4.8.1. GUI_OpenAlphaBlend

➢ **PROTOTYPE**

```
void GUI_OpenAlphaBlend();
```

➢ **ARGUMENTS**

None.

➢ **RETURNS**

None

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

- **DESCRIPTION**

打开 alpha 操作.

26.4.8.2. GUI_CloseAlphaBlend

- **PROTOTYPE**

```
void GUI_CloseAlphaBlend();
```

- **ARGUMENTS**

None.

- **RETURNS**

None

- **DESCRIPTION**

关闭 alpha 操作.

26.4.8.3. GUI_CharSetToEncode

- **PROTOTYPE**

```
__s32    GUI_CharSetToEncode( __s32 charset_enm );
```

- **ARGUMENTS**

charset_enm 字符集(see table below).

EPDK_CHARSET_ENM_GB2312	简体中文
EPDK_CHARSET_ENM_UTF8	utf8
EPDK_CHARSET_ENM_UTF16BE	utf16be
EPDK_CHARSET_ENM_UTF16LE	utf16le
EPDK_CHARSET_ENM_BIG5	繁体中文
EPDK_CHARSET_ENM_GBK	中文
EPDK_CHARSET_ENM_SJIS	日文
EPDK_CHARSET_ENM_EUC_JP	日文
EPDK_CHARSET_ENM_EUC_KR	韩文
EPDK_CHARSET_ENM_KIO8_R	俄文
EPDK_CHARSET_ENM_ISO_8859_1	西欧语言
EPDK_CHARSET_ENM_ISO_8859_2	中欧语言
EPDK_CHARSET_ENM_ISO_8859_3	南欧语言
EPDK_CHARSET_ENM_ISO_8859_4	北欧语言
EPDK_CHARSET_ENM_ISO_8859_5	西里尔字母
EPDK_CHARSET_ENM_ISO_8859_6	阿拉伯语
EPDK_CHARSET_ENM_ISO_8859_7	希腊语
EPDK_CHARSET_ENM_ISO_8859_8	希伯来语
EPDK_CHARSET_ENM_ISO_8859_9	土耳其语
EPDK_CHARSET_ENM_ISO_8859_10	北欧斯堪的纳维亚语系
EPDK_CHARSET_ENM_ISO_8859_11	泰语
EPDK_CHARSET_ENM_ISO_8859_12	梵文

ALLWINNER

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

EPDK_CHARSET_ENM_ISO_8859_28	波罗的海语系
EPDK_CHARSET_ENM_ISO_8859_14	凯尔特人语系
EPDK_CHARSET_ENM_ISO_8859_15	扩展了法语和芬兰语的西欧语系
EPDK_CHARSET_ENM_ISO_8859_16	扩展的东南欧语系
EPDK_CHARSET_ENM_CP874	泰文
EPDK_CHARSET_ENM_CP1250	中欧
EPDK_CHARSET_ENM_CP1251	西里尔文
EPDK_CHARSET_ENM_CP1253	希腊文
EPDK_CHARSET_ENM_CP1255	希伯来文
EPDK_CHARSET_ENM_CP1256	阿拉伯文
EPDK_CHARSET_ENM_CP1257	波罗的海文
EPDK_CHARSET_ENM_CP1258	越南

- **RETURNS**

  ORANGE_OK　　　成功

  ORANGE_FAIL　　失败

- **DESCRIPTION**

  设定字符集. 通过设定不同的字符集可以实现多国语言.

26.5. Widget

Orange1.0 支持 button、static、listmenu、slider…等十几种控件，用户也可以创建自己的控件，只要在 Orange1.0 中注册过就可以使用。

26.5.1.1. GUI_AddNewControlClass

- **PROTOTYPE**

```
__s32 GUI_AddNewControlClass (pwinclass   p WndClass);
```

- **ARGUMENTS**

  WndClass　　　注册控件信息.

- **RETURNS**

  ORANGE_OK　　　注册成功

  ORANGE_FAIL　　注册失败

- **DESCRIPTION**

  注册新控件到控件类信息列表.

26.5.1.2. GUI_DeleteControlClass

- **PROTOTYPE**

```
__s32 GUI_DeleteControlClass (const char* szClassName);
```

- **ARGUMENTS**

  szClassName　　　控件名称.

- **RETURNS**

ORANGE_OK 注册成功
ORANGE_FAIL 注册失败

➢ **DESCRIPTION**

从控件类信息列表中卸载控件.

26.5.1.3. GUI_EmptyControlClassInfoTable

➢ **PROTOTYPE**

```
void GUI_EmptyControlClassInfoTable ();
```

➢ **ARGUMENTS**

None.

➢ **RETURNS**

None

➢ **DESCRIPTION**

清除控件类信息列表.

26.5.1.4. GUI_GetControlClassInfo

➢ **PROTOTYPE**

```
pctlclassinfo GUI_GetControlClassInfo (const char* szClassName);
```

➢ **ARGUMENTS**

szClassName 控件名称.

➢ **RETURNS**

控件信息

➢ **DESCRIPTION**

获取某类控件的控件信息.

26.6. Resources

资源包括图片资源和语言资源，在使用任何资源前都要先获取该资源.

26.6.1.1. Lang_Open

➢ **PROTOTYPE**

```
HLANG    Lang_Open(char *szAppFile, __u32 mode);
```

➢ **ARGUMENTS**

szAppFile 文件路径名.
mode 打开模式，目前版本此参数没有意义

➢ **RETURNS**

HLANG 句柄，如果为 NULL 则打开失败.

➢ **DESCRIPTION**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

打开指定路径语言资源文件，返回文件句柄.

26.6.1.2. Lang_Read

- **PROTOTYPE**

```
int            Lang_Read(HLANG hLang, int address, int length, char *buffer);
```

- **ARGUMENTS**

hLang 文件句柄.
address 相对文件起点处的偏移地址
length 读取的数据长度
buffer 输出读取的数据

- **RETURNS**

实际读取的字节数

- **DESCRIPTION**

读取语言资源文件指定地址处的数据.

26.6.1.3. Lang_GetStringAddress

- **PROTOTYPE**

```
int Lang_GetStringAddress(HLANG hLang, short LangID, short StringID);
```

- **ARGUMENTS**

hLang 文件句柄.
LangID 语言 ID
StringID SringID，在 Lang.bat 生成的 Lang.h 文件中定义

- **RETURNS**

地址(相对文件起始偏移量)

- **DESCRIPTION**

查询数据地址.

26.6.1.4. Lang_GetStringSize

- **PROTOTYPE**

```
int Lang_GetStringSize(HLANG hLang, short LangID, short StringID);
```

- **ARGUMENTS**

hLang 文件句柄.
LangID 语言 ID
StringID SringID，在 Lang.bat 生成的 Lang.h 文件中定义

- **RETURNS**

查询数据长度(byte)

- **DESCRIPTION**

查询数据长度.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

26.6.1.5. Lang_GetString

- **PROTOTYPE**

```
int Lang_GetString(HLANG hLang, short LangID, short StringID, char *buffer, int length);
```

- **ARGUMENTS**

hLang	文件句柄.
LangID	语言 ID
StringID	SringID, 在 Lang.bat 生成的 Lang.h 文件中定义
buffer	数据输出缓冲区
length	数据输出缓冲区长度

- **RETURNS**

数据实际长度(byte)

- **DESCRIPTION**

查询数据.

26.6.1.6. Lang_Close

- **PROTOTYPE**

```
int Lang_Close(HLANG hLang);
```

- **ARGUMENTS**

hLang	文件句柄.

- **RETURNS**

0

- **DESCRIPTION**

关闭句柄，释放内存.

26.6.1.7. OpenRes

- **PROTOTYPE**

```
HRES OpenRes( char * szAppFile, __u32 mode);
```

- **ARGUMENTS**

szAppFile	文件路径名.
mode	打开模式，目前版本此参数没有意义.

- **RETURNS**

HLANG 句柄，如果为 NULL 则打开失败

- **DESCRIPTION**

打开指定路径图片文件，返回文件句柄.

26.6.1.8. CloseRes

- **PROTOTYPE**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
int32 CloseRes(HRES hRes);
```

- **ARGUMENTS**

hRes 文件句柄.

- **RETURNS**

ORANGE_OK 成功

ORANGE_FAIL 失败

- **DESCRIPTION**

关闭句柄，释放内存.

26.6.1.9. ReadRes

- **PROTOTYPE**

```
uint32 ReadRes(HRES hRes, uint32 address, uint32 length, void * buffer);
```

- **ARGUMENTS**

hRes 文件句柄.

address 相对文件起点处的偏移地址

length 读取的数据长度

buffer 输出读取的数据

- **RETURNS**

实际读取的字节数

- **DESCRIPTION**

读取图片文件指定地址处的数据.

26.6.1.10. GetResSize

- **PROTOTYPE**

```
uint32 GetResSize(HRES hRes, uint16 StyleID, uint16 ID);
```

- **ARGUMENTS**

hRes 文件句柄.

StyleID 图片样式 ID

ID 某张图片 ID.

- **RETURNS**

资源长度(byte)

- **DESCRIPTION**

获取资源长度.

26.6.1.11. GetResAddr

- **PROTOTYPE**

```
uint32 GetResAddr(HRES hRes, uint16 StyleID, uint16 ID);
```

- **ARGUMENTS**

hRes 文件句柄.

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

StyleID 图片样式 ID

ID 某张图片 ID..

- **RETURNS**

地址(相对文件起始偏移量)

- **DESCRIPTION**

获取资源地址.

26.6.1.12. GetRes

- **PROTOTYPE**

```
int32 GetRes(HRES hRes, uint16 StyleID, uint16 ID, void * Buffer, uint32 Length);
```

- **ARGUMENTS**

hRes 文件句柄.

StyleID 图片样式 ID

ID 某张图片 ID.

buffer 数据输出缓冲区

length 数据输出缓冲区长度.

- **RETURNS**

ORANGE_OK 成功

ORANGE_FAIL 失败

- **DESCRIPTION**

获取资源.

26.7. Orange Demo

该部分是一个简单的应用程序 Demo,

```
/**************************************************************************************************************/
#include "epdk.h"
#include "apps.h"
#include "res\\res.h"

#include "mlang\\fs_charset.h"
#include "mlang\\isn.h"
#include "mlang\\language.h"

/******************************************************************************************************************/
#define ID_WIDGET_STATIC     1000
#define ID_FRMWIN_HTOUCH  2000

typedef struct
{
        __u32                   focus_size;
        __u32                   unfocus_size;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
    char                    text[128];
    button_para_t  para;
}htoutch_frmw_ctr;

static H_LYR       layer;
static GUI_FONT    *SWFONT;
static GUI_FONT    *SWFONT1;
/**************************************************************************************************************/
static __u32 get_res_them(void **buff, __u16 StyleID, __u16 id)
{
    HRES    h_them;
    __u32           size;

    h_them = OpenRes(NULL, 0);

    size = GetResSize(h_them, StyleID, id);
    *buff = (void *)esMEMS_Balloc(size);

    GetRes(h_them,
            StyleID,
            id,
            (void *)((__u32) *buff),
            size);

    CloseRes(h_them);

    return size;
}

static void free_res_them(void *buff, __u32 size)
{
    esMEMS_Bfree(buff, size);
}

static H_WIN htouch_static_ctl_create(H_WIN parent, button_para_t *para)
{
    H_WIN                                           h_ctrl;
    RECT                                        rect;
    __gui_ctlwincreate_para_t       create_para;

    eLIBs_memset(&create_para, 0, sizeof(__gui_ctlwincreate_para_t));

    GUI_WinGetClientRECT(parent, &rect);
```

```
        create_para.dwStyle              = WS_VISIBLE;
        create_para.dwExStyle            = 0;
        create_para.ctl_rect.x           = 0;
        create_para.ctl_rect.y           = 0;
        create_para.ctl_rect.width       = rect.width;
        create_para.ctl_rect.height      = rect.height;
        create_para.spClassName          = CTRL_BUTTON;
        create_para.hParent              = parent;
        create_para.id                   = ID_WIDGET_STATIC;
        create_para.attr                 = para;

        h_ctrl = GUI_CtrlWinCreate(&create_para);
        if( !h_ctrl )
        {
                __err(" static date control win create error \n");
        }

        return h_ctrl;
}

static __s32 htouch_frmwin_cb(__gui_msg_t *msg)
{
        switch( msg->id )
        {
                case GUI_MSG_CREATE:
                {
                        htouch_frmw_ctr *ctr;
                        button_para_t    *para;
                        char out_string[256] = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x50, 0x51, 0x52, 0x0D, 0x0A, 0x49, 0X00};
                        GUI_RECT string_rect;

                        ctr = esMEMS_Malloc(0, sizeof(htoutch_frmw_ctr));
                        if( !ctr )
                        {
                                __err(" frmwin malloc fail \n");
                                return EPDK_FALSE;
                        }
                        eLIBs_memset(ctr, 0, sizeof(htoutch_frmw_ctr));

                        para = &(ctr->para);

                        ctr->focus_size          = get_res_them(&(para->focus_bmp), ID_STYLE0, ID_ICON_BMP_FOCUS_PIC_BMP);
                        ctr->unfocus_size        = get_res_them(&(para->unfocus_bmp), ID_STYLE0, ID_ICON_BMP_UNFOCUS_PIC_BMP);
```

```
        lang_res_open();
        select_language();
        get_menu_text(STRING_HELLO_TOUCH, ctr->text, 128);
        lang_res_close();

        para->bmp_pos.x         = 100;
        para->bmp_pos.y                 = 100;

        para->text_pos.x        = 100;
        para->text_pos.y        = 200;

        para->text                      = ctr->text;
        para->ftxt_color        = GUI_RED;
        para->uftxt_color       = GUI_WHITE;
        para->draw_font                 = SWFONT;
        para->draw_code                 = EPDK_CHARSET_ENM_UTF8;
        para->txt_align         = 0;
        para->bk_color          = 0;
        para->alpha_en          = EPDK_FALSE;

        /*set rect*/
        string_rect.x0          = 50;
        string_rect.y0          = 0;
        string_rect.x1          = 700;
        string_rect.y1          = 200;

        htouch_static_ctl_create(msg->h_deswin, para);

        GUI_BMP_Draw(para->unfocus_bmp, 400, 200);

        GUI_CharSetToEncode(EPDK_CHARSET_ENM_UTF8);
        GUI_SetFont(SWFONT);

        GUI_DispStringAt(out_string, 0, 0);

        GUI_WinSetAddData( msg->h_deswin, (__u32)ctr);
}
        return EPDK_OK;

case GUI_MSG_DESTROY:
{
        htoutch_frmw_ctr *ctr = (htoutch_frmw_ctr *)GUI_WinGetAddData(msg->h_deswin);

        free_res_them(ctr->para.focus_bmp,    ctr->focus_size);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
        free_res_them(ctr->para.unfocus_bmp, ctr->unfocus_size);

        esMEMS_Mfree(0, ctr);
    }
        return EPDK_OK;

    case GUI_MSG_CLOSE:
    {
        GUI_FrmWinDelete(msg->h_deswin);
    }
        return EPDK_OK;

    case GUI_MSG_COMMAND:
    {
        switch(LOSWORD(msg->dwAddData1))
        {
            case ID_WIDGET_STATIC:
            {
                switch( HISWORD(msg->dwAddData1) )
                {
                    case BN_CLICKED:
                    {
                        __gui_msg_t msgex;

                        eLIBs_memset(&msgex, 0, sizeof(__gui_msg_t));

                        __here__
                        msgex.id = GUI_MSG_CLOSE;
                        msgex.h_srcwin = 0;
                        msgex.h_deswin = GUI_WinGetManWin(msg->h_deswin);
                        __here__
                        GUI_SendNotifyMessage(&msgex);
                        __here__
                    }
                    break;
                }
            }
                break;
        }
    }
        return EPDK_OK;

    default:
        break;
```

```
	}

	return GUI_FrmWinDefaultProc(msg);
}

static H_WIN htouch_frmwin_create(H_WIN parent, H_LYR layer)
{
	H_WIN                                   h_win;
	__gui_framewincreate_para_t para;

	eLIBs_memset(&para, 0, sizeof(__gui_framewincreate_para_t));

	para.dwStyle          = WS_VISIBLE;
	para.dwExStyle        = 0;
	para.name                     = NULL;
	para.hOwner                   = 0;
	para.hHosting         = parent;
	para.FrameWinProc     = htouch_frmwin_cb;
	para.id                       = ID_FRMWIN_HTOUCH;
	para.hLayer           = layer;

	para.rect.x                   = 0;
	para.rect.y                   = 0;
	para.rect.width           = 800;
	para.rect.height          = 480;

	para.BkColor.alpha      = 0;
	para.BkColor.red        = 0;
	para.BkColor.green      = 0;
	para.BkColor.blue       = 0;
	para.attr                     = NULL;

	h_win = GUI_FrmWinCreate(&para);
	if( !h_win )
	{
		__err("volume frm win create error ! \n");
	}

	return h_win;
}

/* layer : we can draw res on it */
static H_LYR htouch_layer_create(void)
{
```

```
H_LYR layer = NULL;
FB    fb =
{
    {0, 0},                                              /* size       */
    {0, 0, 0},                                           /* buffer     */
    {FB_TYPE_RGB, {PIXEL_COLOR_ARGB8888, 0, 0}},         /* fmt        */
};

__disp_layer_para_t lstlyr =
{
    DISP_LAYER_WORK_MODE_NORMAL,                         /* mode       */
    0,                                                   /* ck_mode    */
    0,                                                   /* alpha_en   */
    0,                                                   /* alpha_val  */
    1,                                                   /* pipe       */
    0xff,                                                /* prio       */
    {0, 0, 0, 0},                                        /* screen     */
    {0, 0, 0, 0},                                        /* source     */
    DISP_LAYER_OUTPUT_CHN_DE_CH1,                        /* channel    */
    NULL                                                 /* fb         */
};

__layerwincreate_para_t lyrcreate_info =
{
    "hello touch layer",
    NULL,
    GUI_LYRWIN_STA_ON,
    GUI_LYRWIN_NORMAL
};

fb.size.width           = 800;
fb.size.height          = 480;

lstlyr.src_win.x                = 0;
lstlyr.src_win.y                = 0;
lstlyr.src_win.width    = fb.size.width;
lstlyr.src_win.height   = fb.size.height;

lstlyr.scn_win.x        = 0;
lstlyr.scn_win.y        = 0;
lstlyr.scn_win.width    = fb.size.width;
lstlyr.scn_win.height   = fb.size.height;

lstlyr.pipe = 1;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
        lstlyr.fb = &fb;
        lyrcreate_info.lyrpara = &lstlyr;

        layer = GUI_LyrWinCreate(&lyrcreate_info);
        if( !layer )
        {
                __err("app bar layer create error !\n");
        }

        return layer;
}

/*
  * hello touch main win msg proc
  */
static __s32 htouch_mainwin_cb(__gui_msg_t *msg)
{
        __s32 ret;

        switch( msg->id )
        {
                case GUI_MSG_CREATE:
                {
                        layer = htouch_layer_create();

                        GUI_LyrWinSetTop(layer);
                        htouch_frmwin_create(msg->h_deswin, layer);
                }
                        return EPDK_OK;

                case GUI_MSG_DESTROY:
                {
                        GUI_LyrWinDelete(layer);
                }
                        return EPDK_OK;

                case DSK_MSG_HOME:
                case DSK_MSG_KILL:
                        ret = GUI_ManWinDelete(msg->h_deswin);
                        return ret;

                case GUI_MSG_CLOSE:
                        GUI_ManWinDelete(msg->h_deswin);
                        dsk_load_app("main.app");
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
            return EPDK_OK;

        case GUI_MSG_KEY:
        {
            if(    msg->dwAddData1 == GUI_MSG_KEY_ESCAPE )
            {
                GUI_ManWinDelete(msg->h_deswin);
                dsk_load_app("main.app");
                return EPDK_OK;
            }
        }
            break;

        default:
            break;
    }

    return GUI_ManWinDefaultProc(msg);
}

/*
 * hello touch main win create
 */
H_WIN htouch_mainwin_create(void)
{
    H_WIN                                   hManWin;
    __gui_manwincreate_para_t     create_info;

    eLIBs_memset(&create_info, 0, sizeof(__gui_manwincreate_para_t));

    create_info.attr                    = NULL;
    create_info.hParent                 = NULL;
    create_info.hHosting                = NULL;
    create_info.ManWindowProc      = htouch_mainwin_cb;
    create_info.name                    = "APP_HELLOTOUCH";
    hManWin                             = GUI_ManWinCreate(&create_info);

    if(hManWin == NULL)
    {
        __wrn("TouchMain: create main windows failed!\n");
        return NULL;
    }

    return hManWin;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
}

__s32 hello_touch(void *arg)
{
        H_WIN                     hManWin = 0;
        __gui_msg_t     msg;
        __s32                ret = EPDK_FAIL;
        GUI_TTF_ATTR attr;

        __inf("&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\n");
        __inf("&               HelloTouch process enter!                         &\n");
        __inf("&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\n");

        attr.height = 24;
        eLIBs_strcpy(attr.filepath, "d:\\res\\fonts\\font.ttf");
        SWFONT = GUI_TTF_CreateFont(&attr);

        hManWin = htouch_mainwin_create();
        if(hManWin == NULL)
        {
                __wrn("TouchMain: create main windows failed!\n");
                return EPDK_FAIL;
        }

        GUI_SetActiveManWin(hManWin);

        /*  消息循环  */
        while( GUI_GetMessageEx(&msg, hManWin) )
        {
                ret = GUI_DispatchMessage(&msg);              //  分发消息到回调

                if( msg.p_arg )                                          //  同步消息回应
                {
                        GUI_SetSyncMsgRetVal(&msg, ret);
                        GUI_PostSyncSem(&msg);
                }
        }

        GUI_WinThreadCleanup(hManWin);

        /*  释放字体  */
        GUI_SetFont(&GUI_Font6x8);
        GUI_SFT_ReleaseFont(SWFONT);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
    __inf("&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\n");
    __inf("&                HelloTouch process exit!                          &\n");
    __inf("&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\n");

    return EPDK_OK;
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

27. Cedar Module

27.1. Introduction

27.1.1. Description

Melis 操作系统是由全志科技自主研发的一套精简、高效、易用的实时多任务操作系统。Cedar 中间件是基于 Melis 系统上开发的多媒体播放中间件，支持所有常见格式的音视频文件的解码和播放流程控制，为播放器应用程序提供统一的操作接口，使播放器的开发者能够把精力集中在用户操作体验、界面效果上，而不必关心繁杂多样的音视频文件格式、编码格式等。

27.1.2. Purpose

本文档主要讲述 cedar 提供的对外 API，使开发者能迅速了解这些 API 的作用和调用规范，从而基于 cedar 开发自己的多媒体播放器。

27.1.3. Reference

1）读者可以先了解文件封装格式，视频编码格式、音频编码格式的相关内容，然后阅读本文档。
2）读者可以从代码里面了解模块的安装，卸载，打开，关闭以及模块操作命令。

27.2. CEDAR

CEDAR 主要提供以下几个方面的操作接口：

1）音视频文件的播放流程控制，包括开始(包含断点续播)，停止，暂停，快进快退，跳播，换音轨，变速播放，AB 播放。

2）播放过程中获取文件信息，例如得到播放总时间，当前时间，文件音视频编码格式和其他信息；

3）显示效果的操作，提供全屏，满屏，4:3,16:9，裁边等模式，另外提供自定义模式，让应用程序自行设置；

4）解码图像旋转功能，提供 90,180,270,水平镜像，垂直镜像等选项

5）字幕处理，提供字幕的自动检测，解码，切换字幕等功能。

6）截图功能。

7）预览图获取和文件信息预取；

8）杂项。例如 callback 文件读取方式，用于读取加密文件；关闭外挂字幕检测，用于加快打开文件过程；与应用程序的通信管道连接。

本文档只介绍模块操作的控制命令字，如果您希望了解模块的安装，卸载，打开，关闭以及模块操作命令，可以阅读《SW1103REF005_MELIS PROGRAM GUIDE_Module.pdf》文档。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

27.2.1. CEDAR_CMD_SET_MEDIAFILE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_SET_MEDIAFILE；
aux　　= 0；
pbuffer = __cedar_media_file_inf*；

- **RETURNS**

设置结果：

EPDK_OK，
EPDK_FAIL，

- **DESCRIPTION**

设置要播放的文件信息，主要是路径和断点信息。

- **DEMO**

```
//设置文件信息
esMODS_MIoctrl( robin_hced, CEDAR_CMD_SET_MEDIAFILE, 0, &(msg_p->file_info) );
```

27.2.2. CEDAR_CMD_GET_MESSAGE_CHN

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_GET_MESSAGE_CHN；
aux　　= 0；
pbuffer = NULL；

- **RETURNS**

消息通道，类型为__krnl_event_t *

- **DESCRIPTION**

返回一个消息通道，让应用程序和 cedar 能够通信。

- **DEMO**

```
__s32 get_feedback_msgQ( void )
{
robin_cedar_msgQ = (g_Queue) esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_MESSAGE_CHN, 0, NULL );
if( robin_cedar_msgQ == NULL )
{
    __err("Error in getting cedar error channel.\n");
    return -1;
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
return 0;
}
```

27.2.3. CEDAR_CMD_GET_ERROR_TYPE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_GET_ERROR_TYPE；
aux　　　= 0；
pbuffer　= _NULL；

- **RETURNS**

得到错误类型

- **DESCRIPTION**

获取媒体播放器的错误类型,没有实现。

27.2.4. CEDAR_CMD_PLAY

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_PLAY；
aux　　　= 0；
pbuffer　= NULL；

- **RETURNS**

启动播放的结果：

EPDK_OK：启动成功；
EPDK_FAIL：启动失败；

- **DESCRIPTION**

启动播放，在调用这个接口之间，必须先 CEDAR_CMD_SET_MEDIAFILE，设置文件信息。

- **DEMO**

```
ret = esMODS_MIoctrl( robin_hced, CEDAR_CMD_PLAY, 0, NULL );
if( ret != EPDK_OK )
{
    __wrn("Fail in setting play cmd.\n");
    return SYN_OP_RET_SEND_CMD_ERR;
}
```

27.2.5. CEDAR_CMD_STOP

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp   cedar 模块的句柄;
cmd   = CEDAR_CMD_STOP;
aux   = 0;
pbuffer   = NULL;

- **RETURNS**

停止播放的结果:

EPDK_OK: 停止成功;
EPDK_FAIL: 停止失败;

- **DESCRIPTION**

停止播放文件。

- **DEMO**

```
ret = esMODS_MIoctrl( robin_hced, CEDAR_CMD_STOP, 0, NULL );
 if( ret != EPDK_OK )
 {
     __wrn("Fail in setting stop cmd.\n");
     return SYN_OP_RET_SEND_CMD_ERR;
 }
```

27.2.6. CEDAR_CMD_PAUSE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp   cedar 模块的句柄;
cmd   = CEDAR_CMD_PAUSE;
aux   = 0;
pbuffer   = NULL;

- **RETURNS**

暂停的结果:

EPDK_OK: 暂停成功;
EPDK_FAIL: 暂停失败;

- **DESCRIPTION**

暂停播放，如果 cedar 当前已处于暂停状态，或出于正常播放状态，一般会成功。
如果 cedar 处于快进快退状态，暂停无效。

- **DEMO**

```
ret = esMODS_MIoctrl( robin_hced, CEDAR_CMD_PAUSE, 0, NULL );
 if( ret != EPDK_OK )
 {
```

```
        __wrn("Fail in setting pause cmd.\n");
        return SYN_OP_RET_SEND_CMD_ERR;
    }
```

27.2.7. CEDAR_CMD_FF

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_FF；
aux　　　= 0；
pbuffer　= NULL；

➢ **RETURNS**

快进的结果：

EPDK_OK：cedar 接受快进命令；

EPDK_FAIL：cedar 拒绝快进命令；

➢ **DESCRIPTION**

命令 cedar 进入快进状态。返回值为 EPDK_OK 时，只是表明 Cedar 接受了快进命令，但不一定能成功转入快进状态。所以还需要查 cedar 的状态以确认。

➢ **DEMO**

```
__s32 syn_op_ff( void )
{
    __s32  ret;
    __u8   err;
    __s32  msg;
    __u32  counter;

    /* clear cedar message Queue */
    g_QFlush( robin_cedar_msgQ );
    /* send ff command */
    ret = esMODS_MIoctrl( robin_hced, CEDAR_CMD_FF, 0, NULL );
    if( ret != EPDK_OK )
    {
        __wrn("Fail in setting ff cmd.\n");
        return SYN_OP_RET_SEND_CMD_ERR;
    }

    /* check cedar status until some message is gotten */
    counter = 0;
    while( 1 )
    {
        ret = esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_STATUS, 0, NULL );
        if( ret == CEDAR_STAT_FF )
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
        {
            return SYN_OP_RET_OK;
        }
        msg = (__s32)g_QAccept( robin_cedar_msgQ, &err );
        if( msg != NULL )
        {
            /* feedback msg */
            if( msg == CEDAR_FEDBAK_NO_ERROR )
            {
                return SYN_OP_RET_OK;
            }
            else
            {
                g_QFlush( robin_feedbackQ );
                g_QPost( robin_feedbackQ, (void *)msg );          // only feed back error
message
                return SYN_OP_RET_CEDAR_FEEDBACK_ERR;
            }
        }
        if( to_quit_robin )
            return SYN_OP_RET_TO_QUIT_ROBIN;

        if( ++counter >= OVERTIME_THRESHOLD )
            return SYN_OP_RET_OVERTIME;

        g_delay( 5 );
    }
```

27.2.8. CEDAR_CMD_REV

- **PROTOTYPE**

```
    __s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp   cedar 模块的句柄；
cmd   = CEDAR_CMD_REV；
aux   = 0；
pbuffer   = NULL；

- **RETURNS**

快退的结果：

EPDK_OK：cedar 接受快退命令；

EPDK_FAIL：cedar 拒绝快退命令；

- **DESCRIPTION**
- **DEMO**

同 CEDAR_CMD_FF。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

27.2.9. CEDAR_CMD_JUMP

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　cedar 模块的句柄；

cmd　　= CEDAR_CMD_JUMP；

aux　　= nPts；跳播的目的时间，单位毫秒。

pbuffer = 精确跳播的标记；1 表示精确跳播，0 表示可以有误差。精确跳播用于恒定码率的 mp3 文件的跳播，复读机等产品的音频文件一般是 CBR 的 mp3，对跳播精确度要求非常高，故特别加此参数。

➢ **RETURNS**

跳播命令是否被接受：

EPDK_OK：cedar 接受跳播命令；

EPDK_FAIL：cedar 拒绝跳播命令；

➢ **DESCRIPTION**

跳播命令被接受后，cedar 内部的线程会正式执行跳播的相关操作，有可能执行过程中会失败。所以即使 cedar 接受了跳播命令，也不一定能成功跳播到目的时间点。有可能会引发异常而终止播放。

➢ **DEMO**

```
__s32 syn_op_jump( __u32 time )
{
__s32  ret;
__u8   err;
__s32  msg;
__u32  counter;

/* clear cedar message Queue */
g_QFlush( robin_cedar_msgQ );
/* send play command */
ret = esMODS_MIoctrl( robin_hced, CEDAR_CMD_JUMP, time, NULL );
if( ret != EPDK_OK )
{
    __wrn("Fail in setting jump cmd.\n");
    return SYN_OP_RET_SEND_CMD_ERR;
}

/* check cedar status until some message is gotten */
counter = 0;
while( 1 )
{
    ret = esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_STATUS, 0, NULL );
    if( ret == CEDAR_STAT_PLAY || ret == CEDAR_STAT_STOP )
    {
        return SYN_OP_RET_OK;
```

```
        }
        msg = (__s32)g_QAccept( robin_cedar_msgQ, &err );
        if( msg != NULL )
        {
            /* feedback msg */
            if( msg == CEDAR_FEDBAK_NO_ERROR )
            {
                return SYN_OP_RET_OK;
            }
            else
            {
                g_QFlush( robin_feedbackQ );
                g_QPost( robin_feedbackQ, (void *)msg );         // only feed back error
message
                return SYN_OP_RET_CEDAR_FEEDBACK_ERR;
            }
        }
        if( to_quit_robin )
            return SYN_OP_RET_TO_QUIT_ROBIN;

        if( ++counter >= OVERTIME_THRESHOLD )
            return SYN_OP_RET_OVERTIME;

        g_delay( 5 );
    }

    return SYN_OP_RET_OK;
}
```

27.2.10. CEDAR_CMD_GET_STATUS

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄;

cmd　　= CEDAR_CMD_GET_STATUS;

aux　　= 是否等到平稳状态再返回。1:表示立即返回 cedar 的状态，0：表示等 cedar 的状态平稳之后再返回。因为 cedar 的状态如果异常，可能一直不会恢复到平稳状态，所以 aux=0 有可能引起应用程序线程的死锁。

pbuffer　= NULL;

- **RETURNS**

__cedar_status_t：现在定义了 6 种状态：

CEDAR_STAT_PLAY=0,

CEDAR_STAT_PAUSE,

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

CEDAR_STAT_STOP,

CEDAR_STAT_FF,

CEDAR_STAT_RR,

CEDAR_STAT_JUMP,

- **DESCRIPTION**

CEDAR_STAT_JUMP 表示当前处于跳播处理状态。另外，如果当前 cedar 正处于状态转换期，那么返回值是 stat|0x80，以表示是临时状态。

如果参数 aux = 0，那么在这个接口内部会循环查询，直到状态平稳，再返回。

Aux=1，就立即返回。

- **DEMO**

```
ret = esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_STATUS, 0, NULL );
    if( ret == CEDAR_STAT_PLAY )
    {
        return SYN_OP_RET_OK;
    }
```

27.2.11. CEDAR_CMD_AUDIO_RAW_DATA_ENABLE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　 = CEDAR_CMD_AUDIO_RAW_DATA_ENABLE；
aux　　 = 0,1；音频是否以 rawdata 输出
pbuffer = NULL；

- **RETURNS**

设置音频 rawdata 输出的结果：

EPDK_OK：设置成功；

EPDK_FAIL：设置失败；

- **DESCRIPTION**

命令 cedar 以 rawdata 输出音频。必须在播放前设置，播放过程中设置无效。Cedar 并不是所有的音频编码格式都以 rawdata 输出，目前 ac3 和 dts 是会以 rawdata 输出，其他格式仍然是解码后再输出。

- **DEMO**

```
__s32 robin_enable_raw_data( __bool flag )
{
__u8    err;
__s32   ret;

g_pend_mutex( robin_cedar_mutex, &err );

ret = esMODS_MIoctrl( robin_hced, CEDAR_CMD_AUDIO_RAW_DATA_ENABLE, flag, NULL );

g_post_mutex( robin_cedar_mutex );
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
    return ret;
}
```

27.2.12. CEDAR_CMD_GET_TOTAL_TIME

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp   cedar 模块的句柄;
cmd   = CEDAR_CMD_GET_TOTAL_TIME;
aux   = 0
pbuffer   = NULL;

➢ **RETURNS**

文件的总时间：单位毫秒。

➢ **DESCRIPTION**

查询文件的总时间。

➢ **DEMO**

```
__u32  robin_get_total_time( void )
{
    __u32  time;

    if( get_msg_nr_of_q( robin_cmdQ ) != 0 )
        return 0;

    time = (__u32)esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_TOTAL_TIME, 0, NULL );

    return time;
}
```

27.2.13. CEDAR_CMD_GET_CUR_TIME

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp   cedar 模块的句柄;
cmd   = CEDAR_CMD_GET_CUR_TIME;
aux   = 0
pbuffer   = NULL;

➢ **RETURNS**

当前的播放时间：单位毫秒

➢ **DESCRIPTION**

得到当前的播放时间。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

➢ **DEMO**

```
__u32  robin_get_cur_time( void )
{
return  (__u32)esMODS_MIoctrl( robin_hced,  CEDAR_CMD_GET_CUR_TIME,  0,  NULL );
}
```

27.2.14. CEDAR_CMD_GET_TAG

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_GET_TAG；
aux　　　= 0
pbuffer　= __cedar_tag_inf_t*，APP 分配内存。

➢ **RETURNS**

是否获取成功：

EPDK_OK：获取成功；

EPDK_FAIL：获取失败；

➢ **DESCRIPTION**

获取当前播放时间点的标签信息，app 可以将其作为断点信息保存下来，下次再播放该文件时，将其传入 cedar，cedar 就可以从 tag 所标识的时间点开始播放。

➢ **DEMO**

```
__s32  robin_get_tag( __cedar_tag_inf_t  *tag_info_p )
{
if( tag_info_p == NULL )
    return -1;

if( esMODS_MIoctrl( robin_hced,  CEDAR_CMD_GET_TAG,  0,  tag_info_p ) != EPDK_OK )
    return -1;

return 0;
}
```

27.2.15. CEDAR_CMD_SET_FRSPEED

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_SET_FRSPEED；
aux　　　= 快进快退的速度，现在规定在 1~128 倍速之间。
pbuffer　= NULL。

- **RETURNS**

是否设置成功：

EPDK_OK：设置成功；

EPDK_FAIL：设置失败；

- **DESCRIPTION**

设置快进快退的速度。

- **DEMO**

```
__s32  robin_set_ff_rr_speed( __u32 ff_rr_speed )
{
 if( esMODS_MIoctrl( robin_hced, CEDAR_CMD_SET_FRSPEED, ff_rr_speed, NULL ) == EPDK_OK )
     return 0;
 else
     return -1;
}
```

27.2.16. CEDAR_CMD_GET_FRSPEED

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；

cmd　　　= CEDAR_CMD_GET_FRSPEED；

aux　　　= 0

pbuffer　= NULL；

- **RETURNS**

获取当前的快进快退速度：

1~128：倍速；

0：获取失败。

- **DESCRIPTION**

获取当前设置的快进快退的速度值。如果返回值为 0，表示没有设置。

- **DEMO**

```
__u32  robin_get_ff_rr_speed( void )
{
 return esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_FRSPEED, 0, NULL );
}
```

27.2.17. CEDAR_CMD_SET_TAG

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；

cmd = CEDAR_CMD_SET_TAG;
aux = 0
pbuffer = __cedar_tag_inf_t*，APP 分配内存。

- **RETURNS**

设置断点信息是否成功：

EPDK_OK：设置成功；

EPDK_FAIL：设置失败；

- **DESCRIPTION**

将断点信息设入 cedar，之后调用 CEDAR_CMD_PLAY 时，即从断点信息标识的地方开始播放，实现断点续播功能。

一般在接口 CEDAR_CMD_SET_MEDIAFILE 的参数 __cedar_media_file_inf 中附带了断点信息 __cedar_tag_inf_t*，可以在调用 CEDAR_CMD_SET_MEDIAFILE 时顺带输入断点信息。但也可以单独输入，如果想单独输入，CEDAR_CMD_SET_TAG 必须在 CEDAR_CMD_SET_MEDIAFILE 之后调用才有效。

- **DEMO**

27.2.18. CEDAR_CMD_GET_ABSTYPE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp cedar 模块的句柄；
cmd = CEDAR_CMD_GET_ABSTYPE;
aux = 0
pbuffer = NULL;

- **RETURNS**

音频编码类型，__cedar_audio_fmt_t;

- **DESCRIPTION**

获取当前正在播放的音频流的编码类型。

- **DEMO**

```
__cedar_audio_fmt_t  robin_get_audio_encoding( void )
{
    __cedar_audio_fmt_t  ret;

    robin_wait_no_file( );

    ret  =  (__cedar_audio_fmt_t)esMODS_MIoctrl(  robin_hced,  CEDAR_CMD_GET_ABSTYPE,  0,
NULL );

    g_post_mutex( robin_cedar_mutex );
    return ret;
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

27.2.19. CEDAR_CMD_GET_AUDBPS

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp   cedar 模块的句柄；
cmd   = CEDAR_CMD_GET_AUDBPS；
aux   = 0
pbuffer = NULL；

- **RETURNS**

  码率，单位是 Bit/s。

- **DESCRIPTION**

获取音频流的平均码率。

- **DEMO**

```
__u32  robin_get_audio_bps( void )
{
 __u32  ret;
 __u8   err;

 g_pend_mutex( robin_cedar_mutex, &err );
 robin_wait_no_file( );

 ret = (__u32)esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_AUDBPS, 0, NULL );

 g_post_mutex( robin_cedar_mutex );
 return ret;
}
```

27.2.20. CEDAR_CMD_GET_SAMPRATE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp   cedar 模块的句柄；
cmd   = CEDAR_CMD_GET_SAMPRATE；
aux   = 0
pbuffer = NULL；

- **RETURNS**

  采样率：sample/s

- **DESCRIPTION**

获取当前播放的音频流的采样率。

- **DEMO**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
__u32  robin_get_audio_sample_rate( void )
{
 return  (__u32)esMODS_MIoctrl( robin_hced,  CEDAR_CMD_GET_SAMPRATE,  0,  NULL );
}
```

27.2.21. CEDAR_CMD_SET_CHN

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp,  __u32 cmd,  __s32 aux,  void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_SET_CHN;
aux　　　= 0
pbuffer　= NULL;

- **RETURNS**

EPDK_OK：成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

设置声道类型，即把左声道的声音作为双声道输出，或者右声道。
目前该功能由应用程序设置，Cedar 不实现该功能。

- **DEMO**

27.2.22. CEDAR_CMD_GET_CHN

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp,  __u32 cmd,  __s32 aux,  void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_GET_CHN;
aux　　　= 0;
pbuffer　= NULL;

- **RETURNS**

声道数；

- **DESCRIPTION**

获取当前播放的音频流的声道数。

- **DEMO**

27.2.23. CEDAR_CMD_SET_VOL

- **PROTOTYPE**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄;
cmd　　　= CEDAR_CMD_SET_VOL;
aux　　　= 0~30，音量值
pbuffer　= NULL;

- **RETURNS**

设置的音量值;

- **DESCRIPTION**

Cedar 不实现该接口，这个功能由 app 去实现。

- **DEMO**

27.2.24. CEDAR_CMD_GET_VOL

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄;
cmd　　　= CEDAR_CMD_GET_VOL;
aux　　　= 0
pbuffer　= NULL;

- **RETURNS**

当前的音量值

- **DESCRIPTION**

Cedar 不实现该接口，因此调用这个接口得到的返回值无意义。

- **DEMO**

27.2.25. CEDAR_CMD_VOLUP

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄;
cmd　　　= CEDAR_CMD_VOLUP;
aux　　　= 0;
pbuffer　= NULL;

- **RETURNS**

上调一档以后的音量值;

- **DESCRIPTION**

Cedar 不实现该接口。

- **DEMO**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

27.2.26. CEDAR_CMD_VOLDOWN

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_VOLDOWN；
aux　　　= 0；
pbuffer = NULL；

- **RETURNS**

下调一档以后的音量值；

- **DESCRIPTION**

Cedar 不实现该接口。

- **DEMO**

27.2.27. CEDAR_CMD_SET_EQ

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_SET_EQ；
aux　　　= __cedar_audio_eq_t；
pbuffer = __s8 * UsrEq，实际类型是__s8 UsrEq[USR_EQ_BAND_CNT]。

- **RETURNS**

用户设置以后的 EQ 值，__cedar_audio_eq_t。

- **DESCRIPTION**

设置音效模式，音效模式的值为__cedar_audio_eq_t。其中有用户自定义的音效类型，如果是用户自定义，那么 pbuffer 用来传递这个音效的各频段的参数值。

- **DEMO**

```
__s32 robin_set_eq_mode( __cedar_audio_eq_t eq_mode )
{
 if( esMODS_MIoctrl( robin_hced, CEDAR_CMD_SET_EQ, eq_mode, NULL ) == EPDK_OK )
     return 0;
 else
     return -1;
}

__s32 robin_set_eq_value( const __s8 *eq_value_list )
{
 if( eq_value_list == NULL )
     return -1;
```

```
    if( esMODS_MIoctrl( robin_hced,  CEDAR_CMD_SET_EQ,  CEDAR_AUD_EQ_TYPE_USR_MODE,  (void *)eq_value_list ) == EPDK_OK )
        return 0;
    else
        return -1;
}
```

27.2.28. CEDAR_CMD_GET_EQ

➢ **PROTOTYPE**

```
    __s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　cedar 模块的句柄；

cmd　　 = CEDAR_CMD_GET_EQ；

aux　　 = 0

pbuffer = NULL；

➢ **RETURNS**

音效模式，__cedar_audio_eq_t；

➢ **DESCRIPTION**

得到当前设定的音效模式。

➢ **DEMO**

```
    __cedar_audio_eq_t  robin_get_eq_mode( void )
    {
    return   (__cedar_audio_eq_t)esMODS_MIoctrl( robin_hced,  CEDAR_CMD_GET_EQ,  0,  NULL );

    }
```

27.2.29. CEDAR_CMD_SET_VPS

➢ **PROTOTYPE**

```
    __s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　willow 模块的句柄；

cmd　　 = CEDAR_CMD_SET_VPS；

aux　　 = 变速范围，-4~10,关于变速的计算，变速后的播放速度 v = 1 * (1 + aux/10)。所以变速后的播放速度的范围是：0.6 倍速~2 倍速之间。

pbuffer = NULL；

➢ **RETURNS**

设置之后的变速值，正常情况下和 aux 应该相等。

➢ **DESCRIPTION**

设置变速值。

➢ **DEMO**

```
__s32  robin_set_play_speed( __s32 play_speed )
{
 if( esMODS_MIoctrl( robin_hced, CEDAR_CMD_SET_VPS, play_speed, NULL ) == EPDK_OK )
     return 0;
 else
     return -1;
}
```

27.2.30. CEDAR_CMD_GET_VPS

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_GET_VPS;
aux　　= 0
pbuffer =NULL;

➢ **RETURNS**

当前的变速值；

➢ **DESCRIPTION**

获取当前的变速值。。

➢ **DEMO**

```
__s32  robin_get_play_speed( void )
{
 return esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_VPS, 0, NULL );
}
```

27.2.31. CEDAR_CMD_SET_AB_A

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_SET_AB_A;
aux　　= 0
pbuffer = 0

➢ **RETURNS**

EPDK_OK：设置 ab 播放的 a 点成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

这是 ab 播放的第一版接口，用于设置 a 点，第一版 ab 播放只支持音频文件，并且只能以调用该接口时的播放时间作为 a 点，所以不设置 a 点的时间参数。

- **DEMO**

```
__s32  robin_set_ab_a( void )
{
/*-----------------===== to be refreshed =====----------------------*/
if( esMODS_MIoctrl( robin_hced, CEDAR_CMD_SET_AB_A, 0, NULL ) == EPDK_OK )
    return 0;
else
    return -1;
}
```

27.2.32. CEDAR_CMD_SET_AB_B

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp cedar 模块的句柄；
cmd = CEDAR_CMD_SET_AB_B;
aux = 0
pbuffer = NULL

- **RETURNS**

EPDK_OK：成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

这是 ab 播放的第一版接口，用于设置 b 点，第一版 ab 播放只支持音频文件，并且只能以调用该接口时的播放时间作为 b 点。设置成功后，立即启动 ab 播放，即跳转到 a 点开始播放。所以，调用该接口前，必须先调用 CEDAR_CMD_SET_AB_LOOPCNT 设置好循环次数。

- **DEMO**

```
__s32  robin_set_ab_b( void )
{
/*-----------------===== to be refreshed =====----------------------*/
if( esMODS_MIoctrl( robin_hced, CEDAR_CMD_SET_AB_B, 0, NULL ) == EPDK_OK )
    return 0;
else
    return -1;
}
```

27.2.33. CEDAR_CMD_SET_AB_LOOPCNT

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_SET_AB_LOOPCNT；
aux　　= loopcnt，希望的 ab 循环次数。
pbuffer = NULL；

- **RETURNS**

EPDK_OK：设置成功；
EPDK_FAIL：设置失败；

- **DESCRIPTION**

设置 ab 播放的循环次数，该接口只用于 ab 播放第一版，必须在调用 CEDAR_CMD_SET_AB_B 之前设置。

- **DEMO**

```
__s32  robin_set_ab_loop_count( __u32 count )
{
/*-----------------===== to be refreshed =====----------------------*/
if( esMODS_MIoctrl( robin_hced, CEDAR_CMD_SET_AB_LOOPCNT, count, NULL ) == EPDK_OK )
    return 0;
else
    return -1;
}
```

27.2.34. CEDAR_CMD_CLEAR_AB

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_CLEAR_AB；
aux　　= 0；
pbuffer = NULL；

- **RETURNS**

EPDK_OK：清除 AB 播放成功；
EPDK_FAIL：清除 AB 播放失败；

- **DESCRIPTION**

清除 ab 播放，清除以后，视频文件播放到 B 点时，不会在跳转回 A 点，而是继续播放。

- **DEMO**

```
__s32  robin_cancle_ab( void )
{
/*-----------------===== to be refreshed =====----------------------*/
if( esMODS_MIoctrl( robin_hced, CEDAR_CMD_CLEAR_AB, 0, NULL ) == EPDK_OK )
    return 0;
else
    return -1;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
}
```

27.2.35. CEDAR_CMD_SET_SPECTRUM

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_SET_SPECTRUM；
aux　　　= 1 或 0，1 表示打开频谱解析，0 表示关闭频谱解析；
pbuffer　= NULL；

➢ **RETURNS**

打开频谱是否成功。
EPDK_OK：成功；
EPDK_FAIL：失败。

➢ **DESCRIPTION**

如果打开频谱解析，那么在做音频渲染时，音频渲染库就会同时计算频谱参数。一般情况下，只有播放音频文件时，才需要打开频谱解析。所以，如果是视频文件，一般返回值是 EPDK_FAIL。

➢ **DEMO**

```
void robin_enable_spectrum( void )
{
 esMODS_MIoctrl( robin_hced, CEDAR_CMD_SET_SPECTRUM, 1, NULL );
}
```

27.2.36. CEDAR_CMD_GET_SPECTRUM

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_GET_SPECTRUM；
aux　　　= 0；
pbuffer　= __u16*；实际类型为__u16　　uSpectValColumn [SPECTRUM_COLUMN_SIZE]，app 必须分配这么大的内存，然后把指针设给 cedar。

➢ **RETURNS**

是否获取到频谱：
EPDK_OK：获取成功；
EPDK_FAIL：获取失败；

➢ **DESCRIPTION**

获取一组频谱，频谱总共有 1024 行，每行 32 列。这个接口是获取其中一行的 32 列。Cedar 内部会根据当前的音频播放时间，选取合适的一行，把 32 个列的值给到 app 设下来的 buffer 里。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

➢ **DEMO**

```
__s32 __dsk_wkm_get_spectrum_info( __u16 *value_list )
{
__s32  ret;

 if( value_list == NULL )
 {
     return -1;
 }

 ret = esMODS_MIoctrl( dsk_wkm_hced, CEDAR_CMD_GET_SPECTRUM, 0, value_list );
 if( ret == EPDK_OK )
 {
     return 0;
    }
 else                              // no instant spectrum info
 {
     return -1;
    }
}
```

27.2.37. CEDAR_CMD_SEL_AUDSTREAM

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_SEL_AUDSTREAM;
aux　　　= 音频流数组的下标号；
pbuffer　= NULL;

➢ **RETURNS**

Cedar 是否接受换音轨的命令：

EPDK_OK：接受；

EPDK_FAIL：拒绝；

➢ **DESCRIPTION**

通知 cedar 换音轨，aux 为目标音轨在音轨数组里的下标号。Cedar 内部会判断是否能够换音轨。如果可以，那么返回 EPDK_OK。然后把消息发到自己的消息队列里，cedar 的内部线程开始进行换音轨的操作。之所以这样做，是因为换音轨的操作比较复杂，涉及到众多 cedar 的内部模块，容易出现异常，所以不想放在 app 的线程里去做，避免影响 app 的运行。所以，返回值为 EPDK_OK，并不代表换音轨完成，只是表明 cedar 接受了换音轨操作。判断换音轨结束，是通过调用 CEDAR_CMD_GET_STATUS 来完成的，在换音轨过程中，cedar 的状态处于临时态，当换音轨操作结束，才会恢复到平稳态。但是是否换音轨成功，还要再通过 CEDAR_CMD_GET_AUDSTREAM 的接口得到当前正在播放的音轨号，和 app 想换到的音轨号进行比较，如果相同才表示换音轨成功。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

在这里，音轨和音频流意义完全相同。

- **DEMO**

```
__s32  robin_select_track( __u32 track_index )
{
__u8   err;

g_pend_mutex( robin_cedar_mutex, &err );
robin_wait_no_file( );

if( esMODS_MIoctrl( robin_hced, CEDAR_CMD_SEL_AUDSTREAM, track_index, NULL ) == EPDK_OK )
{
    g_post_mutex( robin_cedar_mutex );
    return 0;
}
else
{
    g_post_mutex( robin_cedar_mutex );
    return -1;
}
}
```

27.2.38. CEDAR_CMD_GET_AUDSTREAM

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp   cedar 模块的句柄；
cmd   = CEDAR_CMD_GET_AUDSTREAM;
aux   = 0;
pbuffer   = NULL;

- **RETURNS**

当前正在播放的音频流的下标号：从 0 开始。-1 表示异常。

- **DESCRIPTION**

获取当前正在播放的音频流的下标号，下标号一般从 0 开始。音频流数组也是 cedar 给 app 的。音频流的下标号就是在音频流数组里的下标号。

- **DEMO**

```
__s32  robin_get_track_index( void )
{
__s32  ret;
__u8   err;

g_pend_mutex( robin_cedar_mutex, &err );
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
robin_wait_no_file( );

ret = (__s32)esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_AUDSTREAM, 0, NULL );

g_post_mutex( robin_cedar_mutex );
return ret;
}
```

27.2.39. CEDAR_CMD_GET_AUDSTREAM_PROFILE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_GET_AUDSTREAM_PROFILE；
aux　　= 0
pbuffer　= __audstream_profile_t*，app 分配内存。

- **RETURNS**

是否成功获取文件的音频流信息：
EPDK_OK：成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

获取视频文件的音频流信息，这是第一版的接口，在数据结构__audstream_profile_t 里规定了音频流数组的最多元素为 4 个。所以如果视频文件包含了超过 4 个音频流，就不能全部告诉应用程序了。为克服这个缺点，又设计了第二版接口。两版接口都可以使用，但不能混用。

- **DEMO**

```
__s32 robin_get_track_info( __audstream_profile_t *track_info_p )
{
__u8 err;

if( track_info_p == NULL )
    return -1;

g_pend_mutex( robin_cedar_mutex, &err );
robin_wait_no_file( );

if( esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_AUDSTREAM_PROFILE, 0, track_info_p ) ==
EPDK_OK )
{
    g_post_mutex( robin_cedar_mutex );
    return 0;
}
else
```

```
    {
        g_post_mutex( robin_cedar_mutex );
        return -1;
    }
}
```

27.2.40. CEDAR_CMD_GET_AUDSTREAM_CNT

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp   cedar 模块的句柄；
cmd   = CEDAR_CMD_GET_AUDSTREAM_CNT;
aux   = 0
pbuffer   = NULL;

- **RETURNS**

音轨数，-1 表示异常

- **DESCRIPTION**

获取当前播放的视频文件的音轨数。音频文件一般不调用该接口，如果调用，返回值为-1。

- **DEMO**

27.2.41. CEDAR_CMD_GET_AUDSTREAM_PROFILE_V2

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp   cedar 模块的句柄；
cmd   = CEDAR_CMD_GET_AUDSTREAM_PROFILE_V2;
aux   = 0
pbuffer   = (__audstream_profile_v2_t*);

- **RETURNS**

EPDK_OK：获取成功；
EPDK_FAIL：获取失败；

- **DESCRIPTION**

数据结构__audstream_profile_v2_t 内部含有一个指针__audio_bs_info_t*，由 app 分配内存，分配足够数量的元素，所以一般情况下，app 应该先调用 CEDAR_CMD_GET_AUDSTREAM_CNT 得到音频流的数量，然后分配内存，再调用本接口。

- **DEMO**

```
//得到音频流信息
    result = esMODS_MIoctrl(pCedar, CEDAR_CMD_GET_AUDSTREAM_CNT, 0, NULL);
    WARNING("file audio stream count is [%d]\n", result);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
        pTestApp->AProf.nAudStrmMaxCnt = (__u8)result;
        pTestApp->AProf.nAudStrmNum = 0;
        if(pTestApp->AProf.AudStrmListArray)
        {
            free(pTestApp->AProf.AudStrmListArray);
        }
        pTestApp->AProf.AudStrmListArray                                    =
(__audio_bs_info_t*)malloc(sizeof(__audio_bs_info_t)*pTestApp->AProf.nAudStrmMaxCnt);
        if(pTestApp->AProf.nAudStrmMaxCnt == NULL)
        {
            WARNING("malloc fail\n");
            goto _exit_test_app;
        }
        memset(pTestApp->AProf.AudStrmListArray,                              0,
sizeof(__audio_bs_info_t)*pTestApp->AProf.nAudStrmMaxCnt);
        result    =    esMODS_MIoctrl(pCedar,    CEDAR_CMD_GET_AUDSTREAM_PROFILE_V2,    0,
&pTestApp->AProf);
```

27.2.42. CEDAR_CMD_QUERY_BUFFER_USAGE

➢ **PROTOTYPE**

```
    __s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp   cedar 模块的句柄；
cmd   = CEDAR_CMD_QUERY_BUFFER_USAGE；
aux   = 0
pbuffer   = __buffer_usage_t*, app 分配内存；

➢ **RETURNS**

EPDK_OK：查询成功；
EPDK_FAIL：查询失败；

➢ **DESCRIPTION**

查询当前的视频解码驱动的 vbs buffer 的使用率，和音频解码驱动的 abs buffer 的使用率。

➢ **DEMO**

27.2.43. CEDAR_CMD_SET_AB_A_V2

➢ **PROTOTYPE**

```
    __s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp   cedar 模块的句柄；
cmd   = CEDAR_CMD_SET_AB_A_V2；

aux = a 点的时间点，单位 ms，-1 表示取当前的播放时间作为 a 点的时间点。

pbuffer = NULL；

- **RETURNS**

  EPDK_OK：设置成功；

  EPDK_FAIL：设置失败；

- **DESCRIPTION**

这是 ab 播放第二版的接口，用于设置 a 点，aux 用来指定 a 点的 pts,如果想用当前的播放时间作为 a 点，aux=-1 即可。

- **DEMO**

27.2.44. CEDAR_CMD_SET_AB_B_V2

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp cedar 模块的句柄；

cmd = CEDAR_CMD_SET_AB_B_V2；

aux = b 点的时间点，-1 表示取当前的播放时间作为 b 点时间点。

pbuffer = NULL；

- **RETURNS**

  EPDK_OK：设置成功；

  EPDK_FAIL：设置失败；

- **DESCRIPTION**

这是 ab 播放第二版的接口，用于设置 b 点，aux 用来指定 b 点的 pts,如果想用当前的播放时间作为 b 点，aux=-1 即可。注意与第一版不同的是，设置完 b 点后，ab 播放不会生效，要使 ab 播放生效，还要调用 CEDAR_CMD_ENABLE_AB_V2 接口才可以。

- **DEMO**

27.2.45. CEDAR_CMD_SET_AB_LOOPCNT_V2

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp cedar 模块的句柄；

cmd = CEDAR_CMD_SET_AB_LOOPCNT_V2；

aux = ab 播放的循环次数

pbuffer = NULL；

- **RETURNS**

  设置之后的 ab 播放的循环次数，一般就等于 aux，除非设置失败；

- **DESCRIPTION**

这是 ab 播放第二版的接口，用于设置 ab 播放时的循环次数。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

➢ **DEMO**

27.2.46. CEDAR_CMD_CLEAR_AB_V2

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_CLEAR_AB_V2;
aux　　= 清除 ab 播放
pbuffer = NULL;

➢ **RETURNS**

是否清除成功；
EPDK_OK:成功;
EPDK_FAIL:失败。

➢ **DESCRIPTION**

这是 ab 播放第二版的接口，用于清除 ab 播放。

27.2.47. CEDAR_CMD_ENABLE_AB_V2

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_ENABLE_AB_V2;
aux　　= 0;
pbuffer = NULL;

➢ **RETURNS**

启动 ab 播放：
EPDK_OK：启动成功;
EPDK_FAIL：启动失败;

➢ **DESCRIPTION**

启动 ab 播放。

➢ **DEMO**

27.2.48. CEDAR_CMD_SET_AUDIO_AB_MODE_V2

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_SET_AUDIO_AB_MODE_V2；
aux　　　= 音频文件 ab 播放时，回到 a 点的方式。1：用 fseek 操作回到 a 点；0：用跳播方式跳到 a 点。
pbuffer　= NULL；

- **RETURNS**

  EPDK_OK：设置成功；

  EPDK_FAIL：设置失败；

- **DESCRIPTION**

第一版 ab 播放，只支持音频文件，其实现方式是音频解码库记录设置 a 点时读到的文件位置，和设置 b 点时读到的文件位置。如果当前读到的位置超过 b 点，那么就通过 fseek 回到 a 点继续播放。

第二版 ab 播放，因为要支持视频文件，所以不能采用上述方法，故采用的是记录时间点，通过跳播回到 a 点时间点的办法实现 ab 播放。在播放音频文件时，第二版接口默认采用跳播方式。但如果应用程序要求采用 fseek 方式，那么就调用该接口，cedar 就会使用 fseek 方式对音频文件进行 ab 播放，不推荐。希望统一采用跳播方式。

- **DEMO**

27.2.49. CEDAR_CMD_GET_VBSTYPE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_GET_VBSTYPE；
aux　　　= 0
pbuffer　= NULL；

- **RETURNS**

  视频编码类型，__cedar_video_fmt_t。

- **DESCRIPTION**

获取当前正在播放的视频流的编码格式。

- **DEMO**

```
__cedar_video_fmt_t  robin_get_video_encoding( void )
{
__cedar_video_fmt_t  ret;
__u8   err;

g_pend_mutex( robin_cedar_mutex, &err );
robin_wait_no_file( );

ret  =  (__cedar_video_fmt_t)esMODS_MIoctrl(  robin_hced,  CEDAR_CMD_GET_VBSTYPE,  0,
NULL );

g_post_mutex( robin_cedar_mutex );
```

```
    return ret;
}
```

27.2.50. CEDAR_CMD_GET_VIDBITRATE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄;
cmd　　　= CEDAR_CMD_GET_VIDBITRATE;
aux　　　= 0
pbuffer　= NULL;

- **RETURNS**

比特率;

- **DESCRIPTION**

获取当前正在播放的视频流的比特率。

- **DEMO**

```
__u32  robin_get_video_bps( void )
{
    __u32   ret;
    __u8    err;

    g_pend_mutex( robin_cedar_mutex, &err );
    robin_wait_no_file( );

    ret = (__u32)esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_VIDBITRATE, 0, NULL );

    g_post_mutex( robin_cedar_mutex );
    return ret;
}
```

27.2.51. CEDAR_CMD_GET_VIDFPS

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄;
cmd　　　= CEDAR_CMD_GET_VIDFPS;
aux　　　= 0
pbuffer　= NULL;

- **RETURNS**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

帧率，数值放大1000 倍，即25 帧每秒，返回值是25000。

- **DESCRIPTION**

获取当前正在播放的视频流的帧率。

- **DEMO**

```
__u32  robin_get_video_frame_rate( void )
{
__u32  ret;
__u8   err;

g_pend_mutex( robin_cedar_mutex, &err );
robin_wait_no_file( );

ret = (__u32)esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_VIDFPS, 0, NULL );

g_post_mutex( robin_cedar_mutex );
return ret;
}
```

27.2.52. CEDAR_CMD_GET_FRAMESIZE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_GET_FRAMESIZE;
aux　　　= 0t
pbuffer　= NULL;

- **RETURNS**

数值__s32 的意义为(width << 16) + height；单位为像素。

- **DESCRIPTION**

获取当前正在播放的视频流的帧的宽高，宽高合在一个__s32 的数中给出。

- **DEMO**

```
__s32  robin_get_video_frame_size( __u32 *width_p, __u32 *height_p )
{
__u32  size;
__u8   err;

if( width_p == NULL || height_p == NULL )
    return -1;

g_pend_mutex( robin_cedar_mutex, &err );
robin_wait_no_file( );
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
size = (__u32)esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_FRAMESIZE, 0, NULL );
*width_p  = ( size & ( 0xFFFFU<<16) ) >> 16;
*height_p = size & 0xFFFF;

g_post_mutex( robin_cedar_mutex );
return 0;
}
```

27.2.53. CEDAR_CMD_SET_VID_LAYERHDL

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　cedar 模块的句柄；

cmd　　　= CEDAR_CMD_SET_VID_LAYERHDL;

aux　　　= 0

pbuffer　= vid_layer_hdl;

➢ **RETURNS**

EPDK_OK：设置图层句柄成功；

EPDK_FAIL：设置图层句柄失败；

➢ **DESCRIPTION**

设置图层句柄给 cedar。Cedar 自己不会主动申请图层句柄，图层句柄由 app 申请，并传给 cedar。Cedar 得到句柄后，会自己设置图层属性，一般 scaler 属性一定会设置。

➢ **DEMO**

```
__hdle  robin_request_video_layer( const RECT *rect_p, __s32 pipe, __s32 prio )
{
//__disp_layer_para_t      image_layer_para;
//FB                       image_fb_para;
__disp_layer_info_t      image_layer_info;
__disp_fb_t              image_fb_para;
RECT                     image_win;
__hdle                   hlay = NULL;
__u32 arg[3];

if( rect_p == NULL )
    return NULL;

/* frame buffer attribute is ignored here, that is set by Cedar */

image_fb_para.size.height        = 0;                // DONT'T CARE
image_fb_para.size.width         = 0;                // DONT'T CARE
image_fb_para.addr[0]            = NULL;
image_fb_para.format             = DISP_FORMAT_RGB565;          // DONT'T CARE
```

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

```
image_fb_para.seq                   = DISP_SEQ_ARGB;        // DONT'T CARE
image_fb_para.mode                  = 0;                     // DONT'T CARE
image_fb_para.br_swap               = 0;                 // DONT'T CARE
image_fb_para.cs_mode               = NULL;

image_layer_info.mode               = DISP_LAYER_WORK_MODE_NORMAL;
    image_layer_info.pipe               = pipe;
    image_layer_info.prio               = prio;
    image_layer_info.alpha_en           = 0;
    image_layer_info.alpha_val          = 255;
    image_layer_info.ck_enable          = 0;
    image_layer_info.src_win.x         = 0;
image_layer_info.src_win.y          = 0;
image_layer_info.src_win.width      = rect_p->width ;
image_layer_info.src_win.height = rect_p->height;
    image_layer_info.scn_win.x         = rect_p->x      ;
image_layer_info.scn_win.y          = rect_p->y      ;
image_layer_info.scn_win.width      = rect_p->width ;
image_layer_info.scn_win.height = rect_p->height;
    image_layer_info.fb                 = image_fb_para;

arg[0] = DISP_LAYER_WORK_MODE_NORMAL;
arg[1] = 0;
arg[2] = 0;
hlay = g_fioctrl( robin_hdis, DISP_CMD_LAYER_REQUEST, 0, (void *)arg );

if( hlay == NULL )
{
    __err("Error in applying for the video layer.\n");
    goto error;
}

arg[0] = hlay;
arg[1] = (__u32)&image_layer_info;
arg[2] = 0;
g_fioctrl( robin_hdis, DISP_CMD_LAYER_SET_PARA, 0, (void *)arg );

image_win.x        = rect_p->x;
    image_win.y        = rect_p->y;
    image_win.width    = rect_p->width ;
    image_win.height = rect_p->height;

if(esMODS_MIoctrl(robin_hced, CEDAR_CMD_SET_VID_LAYERHDL, 0, (void *)hlay) != EPDK_OK)
    {
```


Copyright © 2018 by Allwinner. All rights reserved

```
            __wrn("Fail in setting video layer handle to cedar!\n");
            goto error;
        }
        //set video window information to cedar
        if(esMODS_MIoctrl(robin_hced, CEDAR_CMD_SET_VID_WINDOW, 0, &image_win) != EPDK_OK)
        {
            __wrn("Fail in setting video window information to cedar!\n");
            goto error;
        }

    return hlay;

   error:
    if( hlay != NULL )
    {
        arg[0] = hlay;
        arg[1] = 0;
        arg[2] = 0;
        g_fioctrl( robin_hdis, DISP_CMD_LAYER_RELEASE, 0, (void *)arg );
        hlay = NULL;
    }
    return NULL;
   }
```

27.2.54. CEDAR_CMD_SET_VID_WINDOW

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_SET_VID_WINDOW;
aux　　　= 0;
pbuffer　= __rect_t *;

- **RETURNS**

  EPDK_OK：设置成功；
  EPDK_FAIL：设置失败；

- **DESCRIPTION**

设置显示窗口的位置和大小。

- **DEMO**

```
__s32  robin_set_video_win( __s32 x, __s32 y, __s32 width, __s32 height )
{
RECT  image_win;
```

```
    image_win.x       = x;
        image_win.y       = y;
        image_win.width   = width;
        image_win.height = height;
    if( esMODS_MIoctrl( robin_hced, CEDAR_CMD_SET_VID_WINDOW, 0, &image_win ) != EPDK_OK )
    {
        __wrn("Fail in setting video window information to cedar!\n");
        return -1;
    }

    return 0;
}
```

27.2.55. CEDAR_CMD_GET_VID_WINDOW

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp cedar 模块的句柄；
cmd = CEDAR_CMD_GET_VID_WINDOW;
aux = 0;
pbuffer = NULL;

- **RETURNS**

显示窗口的位置和大小，__rect_t*；

- **DESCRIPTION**

获取显示窗口的位置和大小。

- **DEMO**

```
__s32 robin_get_video_win( RECT *rect_p )
{
    RECT  *rect_i;

    if( rect_p == NULL )
        return -1;

    rect_i = (RECT *)esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_VID_WINDOW, 0, NULL );
    if( rect_i == NULL )
        return -1;

    g_memcpy( rect_p, rect_i, sizeof(RECT) );

    return 0;
}
```

27.2.56. CEDAR_CMD_SET_VID_SHOW_MODE

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　cedar 模块的句柄;

cmd　　= CEDAR_CMD_SET_VID_SHOW_MODE;

aux　　= __cedar_vide_window_ratio_mode_t;

pbuffer = NULL;

➢ **RETURNS**

设置之后的 cedar 当前的显示模式;

➢ **DESCRIPTION**

设置显示模式。显示模式本质上是解码出来的 framebuffer 和显示驱动的 sceenbuffer 之间的映射关系。即指定 framebuffer 的某个区域，映射到 sceenbuffer 的某个区域。

➢ **DEMO**

```
__s32  robin_set_zoom( robin_zoom_e zoom )
{
__cedar_vide_window_ratio_mode_t   cedar_zoom;
__u32  screen_width;
__u32  screen_height;
__s32  x;
__s32  y;
__s32  width;
__s32  height;

cedar_zoom = map_crs2cedar( zoom );

get_screen_size( &screen_width, &screen_height );
x      = 0;
y      = 0;
width  = screen_width;
height = screen_height;

if( zoom == ROBIN_ZOOM_FIT_VIEW )
{
    __disp_output_type_t  output;

    output  = (__disp_output_type_t)g_fioctrl( robin_hdis, DISP_CMD_GET_OUTPUT_TYPE, 0,
0 ) ;

    if( output == DISP_OUTPUT_TYPE_TV || output == DISP_OUTPUT_TYPE_HDMI )
    {
        if( screen_width == 1280 && screen_height == 720 )
```

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

```
        {
            x       = (1280-1208)>>1;
            y       = (720-680)>>1;
            width   = 1208;
            height  = 680;
        }
        else if( screen_width == 1920 && screen_height == 1080 )
        {
            x       = (1920-1812)>>1;
            y       = (1080-1020)>>1;
            width   = 1812;
            height  = 1020;
        }
        else if( screen_width == 720 && screen_height == 480 )
        {
            x       = (720-660)>>1;
            y       = (480-440)>>1;
            width   = 660;
            height  = 440;
        }
        else if( screen_width == 720 && screen_height == 576 )
        {
            x       = (720-660)>>1;
            y       = (576-536)>>1;
            width   = 660;
            height  = 536;
        }
    }
}

robin_set_video_win( x, y, width, height );
esMODS_MIoctrl( robin_hced, CEDAR_CMD_SET_VID_SHOW_MODE, cedar_zoom, NULL );

cur_zoom = zoom;

return 0;
}
```

27.2.57. CEDAR_CMD_GET_VID_SHOW_MODE

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　 = CEDAR_CMD_GET_VID_SHOW_MODE;
aux　　 = 0;
pbuffer = NULL;

- **RETURNS**

__cedar_vide_window_ratio_mode_t;

- **DESCRIPTION**

获取当前的显示模式。

- **DEMO**

```
robin_zoom_e  robin_get_zoom( void )
{
__cedar_vide_window_ratio_mode_t  cedar_zoom;

 if( cur_zoom == ROBIN_ZOOM_UNKNOWN )
 {
     cedar_zoom   =   (__cedar_vide_window_ratio_mode_t)esMODS_MIoctrl(   robin_hced,
CEDAR_CMD_GET_VID_SHOW_MODE, 0, NULL );
     return map_cedar2crs( cedar_zoom );
 }
 else
     return cur_zoom;
}
```

27.2.58. CEDAR_CMD_SWITCH_VID_SHOW

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　 = CEDAR_CMD_SWITCH_VID_SHOW;
aux　　 = 是否打开屏。1 表示开屏；0 表示关屏。
pbuffer = NULL;

- **RETURNS**

EPDK_OK：设置成功；
EPDK_FAIL：设置失败；

- **DESCRIPTION**

开关屏的操作。在播放过程中，调用该接口，可以开屏或关屏。关屏后，仍在播放，但屏幕不显示图像了。

- **DEMO**

27.2.59. CEDAR_CMD_SET_FRPIC_SHOWTIME

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_SET_FRPIC_SHOWTIME；
aux　　= 希望的显示持续时间，单位 ms。
pbuffer = NULL；

- **RETURNS**

EPDK_OK：设置成功；
EPDK_FAIL：设置失败；

- **DESCRIPTION**

设置快进快退状态下，希望的一帧显示的持续时间。Psr 模块就以这个持续时间为准选取关键帧。因为快进快退状态下，时间轴是倍速与正常时间轴的，一般是 64 倍速。那么意味着一帧如果显示 30ms，这个持续时间等于正常情况下 30*64ms，在这么长的时间里，可以显示很多帧，那么在快进快退下，这些帧就应跳过，选下一帧。这就是快进快退一帧显示时间的含义。默认为 30ms。

- **DEMO**

27.2.60. CEDAR_CMD_GET_FRPIC_SHOWTIME

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_GET_FRPIC_SHOWTIME；
aux　　= 0
pbuffer = NULL；

- **RETURNS**

得到当前设置的快进快退状态下，一帧的显示持续时间；

- **DESCRIPTION**

Cedar 没有实现该接口。

- **DEMO**

27.2.61. CEDAR_CMD_SET_ROTATE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；

cmd　　　= CEDAR_CMD_SET_ROTATE;

aux　　　= 旋转角度设置，0：原图；1：顺时针旋转 90 度；2：顺时针 180 度；3：顺时针 270 度；4：水平镜像；5：垂直镜像。

pbuffer　= NULL;

- **RETURNS**

  EPDK_OK：设置成功

  EPDK_FAIL：设置失败;

- **DESCRIPTION**

  设置解码出的图像的旋转角度。必须在文件播放前设置，即调用 CEDAR_CMD_PLAY 前设置。并且播放过程中不允许再调用该接口改变旋转角度，如果调用，无效。设置完成后，只要 cedar 主控模块不退出，就一直生效。也就是说对所有文件生效，而不仅是当前要播放的文件。

- **DEMO**

27.2.62. CEDAR_CMD_INVALID_VIDEOLAYER

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　　cedar 模块的句柄;

cmd　　　= CEDAR_CMD_INVALID_VIDEOLAYER;

aux　　　= 是否设置图层无效，1：通知 cedar，图层句柄无效；0：通知 cedar，图层句柄恢复有效。

pbuffer　= NULL;

- **RETURNS**

  EPDK_OK：设置成功;

  EPDK_FAIL：设置失败;

- **DESCRIPTION**

  当播放视频过程中，可能在一些情况下，应用程序要把之前设给 cedar 的图层收回做其他用途，但希望 cedar 仍然能继续播放。对于 cedar 而言，没有图层，是可以继续播放的，只是解码出来的图像不再显示而已。所以 cedar 提供了这个接口，允许在播放过程中，将图层设为无效。等 app 使用完毕后，调用 CEDAR_CMD_SET_VID_LAYERHDL 将图层重设给 cedar，再调用本接口，恢复图层。

- **DEMO**

27.2.63. CEDAR_CMD_SET_FILE_SWITCH_VPLY_MODE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　　cedar 模块的句柄;

cmd　　　= CEDAR_CMD_SET_FILE_SWITCH_VPLY_MODE;

aux　　　= CedarFileSwitchVplyMode;

pbuffer　= NULL;

- **RETURNS**

设置文件切换模式的结果：

EPDK_OK：设置成功；

EPDK_FAIL：设置失败；

- **DESCRIPTION**

在文件切换时，cedar 提供了两种模式，一是文件切换过程中关屏；二是切换过程中始终显示上一个文件的切换之前正在显示的帧。第二种模式又称无缝切换。现在默认采用无缝切换模式。如果应用程序希望采用第一种模式，调用本接口设置即可。

- **DEMO**

27.2.64. CEDAR_CMD_ENABLE_VIDEO_AUTO_SCALE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_ENABLE_VIDEO_AUTO_SCALE；
aux　　= 1：允许在解码过程中自动做 scale；0：不允许自动做 scale。
pbuffer = NULL；

- **RETURNS**

EPDK_OK：设置成功；

EPDK_FAIL：设置失败；

- **DESCRIPTION**

因为芯片性能的原因，在解 1080p 等高分辨率视频文件时，常常带宽不够造成屏幕闪烁，这个接口允许应用程序定制 cedar 内部的视频解码驱动的行为：遇到高分辨率文件时，是否做 scale 以减少带宽和内存消耗。所谓 scale 是指在解码时压缩图像，例如一幅图像原本正常解码出来的宽高是 1920*1080，经过 scale，解码出来变为 960*1080 等，这样图像变小了，输出的显示带宽自然变小了，内存消耗也小了。

Cedar 默认不做自动 scale。

- **DEMO**

27.2.65. CEDAR_CMD_GET_LBSTYPE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_GET_LBSTYPE；
aux　　= 0；
pbuffer = NULL；

- **RETURNS**

字幕文件格式：__cedar_lyric_fmt_t；

- **DESCRIPTION**

获取当前的字幕文件格式，对于内置字幕，也定义了一套文件格式的值。需要注意的是：对于文本字幕，其内部文字的编码格式，是另有一套枚举类型定义值的，就是__cedar_subtitle_encode_t。

- **DEMO**

```
__cedar_lyric_fmt_t  robin_get_subtitle_format( void )
{
__cedar_lyric_fmt_t  ret;
__u8   err;

g_pend_mutex( robin_cedar_mutex, &err );
robin_wait_no_file( );

ret  =  (__cedar_lyric_fmt_t)esMODS_MIoctrl(  robin_hced,  CEDAR_CMD_GET_LBSTYPE,  0,
NULL );

g_post_mutex( robin_cedar_mutex );
return ret;
}
```

27.2.66. CEDAR_CMD_GET_SUB_INFO

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_GET_SUB_INFO；
aux　　= nPts,单位 ms；
pbuffer　= __cedar_get_sub_inf_t，字幕条目的类型；具体有：
　CEDAR_GET_SUB_INF_ALL：一次拿到所有的字幕条目。
　CEDAR_GET_SUB_INF_ITEM：根据送入的 nPts 选一条合适的字幕条目给出来。
　类型不同，返回值的类型也不同。

- **RETURNS**

如果类型是 CEDAR_GET_SUB_INF_ALL，返回值为__cedar_buf_inf_t *,
如果类型是 CEDAR_GET_SUB_INF_ITEM ，返回值为__cedar_subtitle_item_t*

- **DESCRIPTION**

获取当前字幕流的字幕条目，或者根据 PTS 拿一个合适的字幕条目,或者拿全部的。因为内存是 cedar 内部分配的，所以应用程序要注意，不要改动内存的内容。

- **DEMO**

```
__s32  robin_get_subtitle_item( __u32 time, __cedar_subtitle_item_t *subtitle_item_p )
{
__cedar_subtitle_item_t  *p = NULL;
__u8   err;
```

```
    if( subtitle_item_p == NULL )
        return -1;

    g_pend_mutex( robin_cedar_mutex, &err );
    robin_wait_no_file( );

    p = (__cedar_subtitle_item_t *)esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_SUB_INFO,
time,
                                                    (void *)CEDAR_GET_SUB_INF_ITEM );
    if( p == NULL )            // no subtitle information at present
    {
        g_post_mutex( robin_cedar_mutex );
        return -1;
    }
    else
    {
        g_memcpy( subtitle_item_p, p, sizeof(__cedar_subtitle_item_t) );
        g_post_mutex( robin_cedar_mutex );
        return 0;
    }
}
```

27.2.67. CEDAR_CMD_GET_SUBTITLE_PROFILE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_GET_SUBTITLE_PROFILE;
aux　　= 0;
pbuffer = __subtitle_profile_t *, app 分配内存;

- **RETURNS**

EPDK_OK：获取成功；
EPDK_FAIL：获取失败；

- **DESCRIPTION**

获取所有的字幕流信息。这是第一版接口，规定了最多只能有 8 个字幕流。第二版接口做了扩展，可以获取的字幕流数量不受限制。

- **DEMO**

```
__s32 robin_get_subtitle_list( __subtitle_profile_t *subtitle_info_p )
{
    __u8   err;

    if( subtitle_info_p == NULL )
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
        return -1;

    g_pend_mutex( robin_cedar_mutex, &err );
    robin_wait_no_file( );

    if( esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_SUBTITLE_PROFILE, 0, subtitle_info_p ) ==
EPDK_OK )
    {
        g_post_mutex( robin_cedar_mutex );
        return 0;
    }
    else
    {
        g_post_mutex( robin_cedar_mutex );
        return -1;
    }
   }
```

27.2.68. CEDAR_CMD_SELECT_SUBTITLE

- **PROTOTYPE**

```
    __s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄;
cmd　　　= CEDAR_CMD_SELECT_SUBTITLE;
aux　　　= 字幕流的下标;
pbuffer　= NULL;

- **RETURNS**

EPDK_OK：设置成功;
EPDK_FAIL：设置失败;

- **DESCRIPTION**

通知 cedar 换字幕流。

- **DEMO**

```
    __s32  robin_select_subtitle( __u32 subtitle_index )
    {
    __u8   err;

    g_pend_mutex( robin_cedar_mutex, &err );
    robin_wait_no_file( );

    if( esMODS_MIoctrl( robin_hced, CEDAR_CMD_SELECT_SUBTITLE, subtitle_index, NULL ) ==
EPDK_OK )
    {
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
        g_post_mutex( robin_cedar_mutex );
        return 0;
    }
    else
    {
        g_post_mutex( robin_cedar_mutex );
        return -1;
    }
}
```

27.2.69. CEDAR_CMD_GET_SUBTITLE

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_GET_SUBTITLE；
aux　　　= 0
pbuffer　= NULL；

➢ **RETURNS**

字幕流下标；

➢ **DESCRIPTION**

获取当前正在播放的字幕流下标。下标从 0 开始。-1 表示字幕还没有检测完毕。遇到这种情况，应用程序应该 delay 一段时间继续查。

➢ **DEMO**

```
__s32  robin_get_subtitle_index( void )
{
 __s32  ret;
 __u8   err;

 g_pend_mutex( robin_cedar_mutex, &err );
 robin_wait_no_file( );

 ret = (__s32)esMODS_MIoctrl( robin_hced, CEDAR_CMD_GET_SUBTITLE, 0, NULL );

 g_post_mutex( robin_cedar_mutex );
 return ret;
}
```

27.2.70. CEDAR_CMD_GET_SUBTITLE_CNT

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp cedar 模块的句柄;
cmd = CEDAR_CMD_GET_SUBTITLE_CNT;
aux = 0
pbuffer = NULL;

➢ **RETURNS**

字幕流的数量, -1 表示字幕解析过程还没完成。

➢ **DESCRIPTION**

该接口配合 CEDAR_CMD_GET_SUBTITLE_PROFILE_V2 一起使用;*-1 表示字幕解析过程还没完成,所以得不到字幕流的数量。应用程序遇到这种情况应 delay 一段时间继续调用本接口。*

➢ **DEMO**

27.2.71. CEDAR_CMD_GET_SUBTITLE_PROFILE_V2

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp cedar 模块的句柄;
cmd = CEDAR_CMD_GET_SUBTITLE_PROFILE_V2;
aux = 0
pbuffer = __subtitle_profile_v2_t*; app 分配内存。

➢ **RETURNS**

EPDK_OK:成功;
EPDK_FAIL:失败;

➢ **DESCRIPTION**

获取视频文件所有的音频流信息。如果是音频文件,返回值为 EPDK_FAIL。该接口一般和 CEDAR_CMD_GET_SUBTITLE_CNT 联合使用,CEDAR_CMD_GET_SUBTITLE_CNT 得到音轨数,然后为 __subtitle_profile_v2_t 分配内存,再调用该接口。

➢ **DEMO**

27.2.72. CEDAR_CMD_SET_SUBTITLE_ITEM_POST_PROCESS

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp cedar 模块的句柄;

全志科技版权所有,侵权必究
Copyright © 2018 by Allwinner. All rights reserved

cmd　　　= CEDAR_CMD_SET_SUBTITLE_ITEM_POST_PROCESS;
aux　　　= 1:对文本字幕条目进行后处理，所谓后处理就是去除一些特殊字符；0：不进行后处理。
pbuffer　= NULL;

- **RETURNS**

  EPDK_OK：成功；

  EPDK_FAIL：失败；

- **DESCRIPTION**

设置是否对解码出来的文本字幕条目进行后处理，去除特殊字符。

- **DEMO**

27.2.73. CEDAR_CMD_ENABLE_EXTERN_SUBTITLE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_ENABLE_EXTERN_SUBTITLE;
aux　　　= 1：进行外挂字幕文件检测；0：禁止检测外挂字幕文件。
pbuffer　= NULL;

- **RETURNS**

图像操作的结果：

  EPDK_OK：成功；

  EPDK_FAIL：失败；

- **DESCRIPTION**

设置 cedar 是否进行外挂字幕检测，不进行外挂字幕检测可以加快文件打开速度。Cedar 默认检测，如果应用程序不想检测字幕，可以调用该接口设置。

- **DEMO**

27.2.74. CEDAR_CMD_CAPTURE_PIC

已废弃。

27.2.75. CEDAR_CMD_ASK_PIC_BUFSIZE

已废弃。

27.2.76. CEDAR_CMD_GET_FRAME_PIC

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved　

➢ **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_CMD_GET_FRAME_PIC；
aux　　　= 0
pbuffer　= FB*；APP 分配内存，必须用 ARGB 格式。FB 的参数也必须配置正确，因为 cedar 不查错。

➢ **RETURNS**

EPDK_OK：成功；
EPDK_FAIL：失败；

➢ **DESCRIPTION**

截图，app 指定截取之后的图的类型，大小，并给定 buffer。准确的说，这个接口并不会给出一张完整的图。例如 BMP，这个接口只给出 bmp 图的内容部分，bmp 的头信息还是由应用程序自己制作。这个接口是同步的，一旦返回就表示截图完成。

➢ **DEMO**

27.2.77. CEDAR_DUCKWEED_CMD_OPEN_MEDIAFILE

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_DUCKWEED_CMD_OPEN_MEDIAFILE；
aux　　　= 0
pbuffer　= NULL；

➢ **RETURNS**

操作的结果：

EPDK_OK：成功；
EPDK_FAIL：失败；

➢ **DESCRIPTION**

打开文件。这个接口是做预览图的接口之一。Cedar 后期集成了预览图中间件，所以也继承了原预览图中间件几乎所有的接口，并和原中间件的操作尽量保持一致，以方便应用程序的修改。

➢ **DEMO**

```
//打开文件
    result = esMODS_MIoctrl(pCedar, CEDAR_DUCKWEED_CMD_OPEN_MEDIAFILE, 0, NULL);
```

27.2.78. CEDAR_DUCKWEED_CMD_CLOSE_MEDIAFILE

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_DUCKWEED_CMD_CLOSE_MEDIAFILE；
aux　　　= 0

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

pbuffer = NULL;

- **RETURNS**

操作的结果：

EPDK_OK：成功；

EPDK_FAIL：失败；

- **DESCRIPTION**

关闭媒体文件。

- **DEMO**

```
    //关闭文件
        result = esMODS_MIoctrl(pCedar, CEDAR_DUCKWEED_CMD_CLOSE_MEDIAFILE, 0, NULL);
```

27.2.79. CEDAR_DUCKWEED_CMD_GET_FILE_FORMAT

- **PROTOTYPE**

```
    __s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；

cmd　　= CEDAR_DUCKWEED_CMD_GET_FILE_FORMAT;

aux　　= 0

pbuffer = NULL;

- **RETURNS**

文件格式；__cedar_media_file_fmt_t。

- **DESCRIPTION**

获取文件格式。

- **DEMO**

```
    //得到文件格式
        result = esMODS_MIoctrl(pCedar, CEDAR_DUCKWEED_CMD_GET_FILE_FORMAT, 0, NULL);
```

27.2.80. CEDAR_DUCKWEED_CMD_GET_FILE_SIZE

- **PROTOTYPE**

```
    __s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；

cmd　　= CEDAR_DUCKWEED_CMD_GET_FILE_SIZE;

aux　　= 0

pbuffer = __s64*;

- **RETURNS**

EPDK_OK：成功；

EPDK_FAIL：失败；

- **DESCRIPTION**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

获取文件大小，因为返回值是__s32，而有的文件长度超出__s32 的范围，所以文件长度放在 pbuffer 中返回给应用程序。

- **DEMO**

```
//得到文件大小
    result = esMODS_MIoctrl(pCedar, CEDAR_DUCKWEED_CMD_GET_FILE_SIZE, 0, &file_size);
```

27.2.81. CEDAR_DUCKWEED_CMD_GET_TOTAL_TIME

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_DUCKWEED_CMD_GET_TOTAL_TIME;
aux　　　= 0
pbuffer　= NULL;

- **RETURNS**

视频的总时间，单位 ms；

- **DESCRIPTION**

获取视频播放的总时间。

- **DEMO**

```
//得到播放总时间
    result = esMODS_MIoctrl(pCedar, CEDAR_DUCKWEED_CMD_GET_TOTAL_TIME, 0, NULL);
```

27.2.82. CEDAR_DUCKWEED_CMD_GET_VIDSTREAM_CNT

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_DUCKWEED_CMD_GET_VIDSTREAM_CNT;
aux　　　= 0
pbuffer　= NULL;

- **RETURNS**

视频文件的视频流数量；

- **DESCRIPTION**

获取视频文件的视频流数量。

- **DEMO**

27.2.83. CEDAR_DUCKWEED_CMD_GET_VIDSTREAM_PROFILE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_DUCKWEED_CMD_GET_VIDSTREAM_PROFILE；
aux　　= 0,
pbuffer　= __vidstream_profile_v2_t*；app 分配内存

- **RETURNS**

EPDK_OK：成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

获取视频流信息。

- **DEMO**

```
    //得到视频流信息
        result = esMODS_MIoctrl(pCedar, CEDAR_DUCKWEED_CMD_GET_VIDSTREAM_CNT, 0, NULL);
        pTestApp->VProf.nVidStrmMaxCnt = (__u8)result;
        pTestApp->VProf.nVidStrmNum = 0;
        if(pTestApp->VProf.VidStrmListArray)
        {
            free(pTestApp->VProf.VidStrmListArray);
        }
        pTestApp->VProf.VidStrmListArray                                                    =
(__video_bs_info_t*)malloc(sizeof(__video_bs_info_t)*pTestApp->VProf.nVidStrmMaxCnt);
        if(pTestApp->VProf.VidStrmListArray == NULL)
        {
            WARNING("malloc fail\n");
            goto _exit_test_app;
        }
        memset(pTestApp->VProf.VidStrmListArray,                                           0,
sizeof(__video_bs_info_t)*pTestApp->VProf.nVidStrmMaxCnt);
        result   =   esMODS_MIoctrl(pCedar,   CEDAR_DUCKWEED_CMD_GET_VIDSTREAM_PROFILE,   0,
&pTestApp->VProf);
```

27.2.84. CEDAR_DUCKWEED_CMD_GET_AUDSTREAM_CNT

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_DUCKWEED_CMD_GET_AUDSTREAM_CNT；

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

aux = 0;
pbuffer = NULL;

- **RETURNS**

  EPDK_OK：成功；

  EPDK_FAIL：失败；

- **DESCRIPTION**

获取视频文件的音频流数量。

- **DEMO**

27.2.85. CEDAR_DUCKWEED_CMD_GET_AUDSTREAM_PROFILE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp cedar 模块的句柄；
cmd = CEDAR_DUCKWEED_CMD_GET_AUDSTREAM_PROFILE；
aux = 0；
pbuffer = __audstream_profile_v2_t*；app 分配内存

- **RETURNS**

  EPDK_OK：成功；

  EPDK_FAIL：失败；

- **DESCRIPTION**

获取音频流信息。

- **DEMO**

```
    //得到音频流信息
        result = esMODS_MIoctrl(pCedar, CEDAR_DUCKWEED_CMD_GET_AUDSTREAM_CNT, 0, NULL);
        WARNING("file audio stream count is [%d]\n", result);
        pTestApp->AProf.nAudStrmMaxCnt = (__u8)result;
        pTestApp->AProf.nAudStrmNum = 0;
        if(pTestApp->AProf.AudStrmListArray)
        {
            free(pTestApp->AProf.AudStrmListArray);
        }
        pTestApp->AProf.AudStrmListArray =
(__audio_bs_info_t*)malloc(sizeof(__audio_bs_info_t)*pTestApp->AProf.nAudStrmMaxCnt);
        if(pTestApp->AProf.nAudStrmMaxCnt == NULL)
        {
            WARNING("malloc fail\n");
            goto _exit_test_app;
        }
        memset(pTestApp->AProf.AudStrmListArray, 0,
sizeof(__audio_bs_info_t)*pTestApp->AProf.nAudStrmMaxCnt);
```

```
        result  =  esMODS_MIoctrl(pCedar,  CEDAR_DUCKWEED_CMD_GET_AUDSTREAM_PROFILE,  0, &pTestApp->AProf);
```

27.2.86. CEDAR_DUCKWEED_CMD_GET_PREVIEW_FB

- **PROTOTYPE**

```
    __s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄；
cmd　　　= CEDAR_DUCKWEED_CMD_GET_PREVIEW_FB;
aux　　　= 0;
pbuffer = FB*; app 分配内存

- **RETURNS**

EPDK_OK：成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

获取视频文件的第一帧图像，

- **DEMO**

```
    memset(fb, 0, sizeof(FB));
        fb->size.width = PREVIEW_WIDTH;
        fb->size.height = PREVIEW_HEIGHT;
        fb->addr[0] = esMEMS_Palloc(((fb->size.width  * fb->size.height * 4 + 1023)>>10), MEMS_PALLOC_MODE_BND_NONE | MEMS_PALLOC_MODE_BNK_NONE);
        if(NULL == fb->addr[0])
        {
            WARNING("palloc fail\n");
            return EPDK_FAIL;
        }
        fb->addr[1] = NULL;
        fb->addr[2] = NULL;
        fb->fmt.type = FB_TYPE_RGB;
        fb->fmt.cs_mode = BT601;
        fb->fmt.fmt.rgb.pixelfmt = PIXEL_COLOR_ARGB8888;
        fb->fmt.fmt.rgb.br_swap = 0;
        fb->fmt.fmt.rgb.pixseq = RGB_SEQ_ARGB;
        fb->fmt.fmt.rgb.palette.addr = NULL;
        fb->fmt.fmt.rgb.palette.size = 0;
    esMODS_MIoctrl(pCedar, CEDAR_DUCKWEED_CMD_GET_PREVIEW_FB, 0, &fb);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

27.2.87. CEDAR_DUCKWEED_CMD_GET_PREVIEW_FB_BY_PTS

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_DUCKWEED_CMD_GET_PREVIEW_FB_BY_PTS；
aux　　= 时间点，单位 ms；
pbuffer　= FB*, APP 分配内存。

- **RETURNS**

EPDK_OK：成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

在指定的时间点附件选一个关键帧，解码获取其图像。转为 FB*指定的规格。

- **DEMO**

```
result = esMODS_MIoctrl(pCedar, CEDAR_DUCKWEED_CMD_GET_PREVIEW_FB_BY_PTS, 3000, &pTestApp->previewFB);
```

27.2.88. CEDAR_CMD_SET_USER_FILEOP

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_SET_USER_FILEOP；
aux　　= 0
pbuffer　= __cedar_usr_file_op_t *, app 分配内存。

- **RETURNS**

EPDK_OK：成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

如果应用程序想通过 callback 方式，由自己去读文件，那么就通过这个接口，将 callback 函数设置给 cedar，cedar 读取文件时，实际上是调用 app 的函数去读取。这个方式一般用于读取加密文件，cedar 不知道解密方法，而 app 知道，所以 app 在自己的函数里读取加密文件并解密，这样 cedar 就不需要关心解密的操作了。

- **DEMO**

```
__cedar_usr_file_op_t   UsrFileOp;
            memset(&UsrFileOp, 0, sizeof(__cedar_usr_file_op_t));
            pApp->pFile = eLIBs_fopen(pApp->media_file.file_path, "r");
        if(!pApp->pFile)
        {
            return EPDK_FAIL;
```

```
            }
            fseek(pAppp->pFile, 0, SEEK_END);
                UsrFileOp.usr_fread = esKRNL_GetCallBack(usr_file_read);//inka_file_read
            UsrFileOp.fp = (__u32)pApp->pFile;
                UsrFileOp.media_fmt = CEDAR_MEDIA_FILE_FMT_AVI; //这里用普通的 AVI 文件做
demo,
                UsrFileOp.file_size = ftell(pApp->pFile);
                UsrFileOp.flag = 0; //没有用到
            UsrFileOp.usr_fseek = NULL; //没有用到
            fseek(pApp->pFile, 0, SEEK_SET);
            esMODS_MIoctrl(pCedar, CEDAR_CMD_SET_USER_FILEOP, 0, &UsrFileOp);
```

27.2.89. 2.90.CEDAR_CMD_SET_STOP_MODE

- **PROTOTYPE**

```
    __s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄;

cmd　　　= CEDAR_CMD_SET_STOP_MODE;

aux　　　=CedarStopMode 。stop 时是保留所有插件，还是卸载所有插件； .aux = CEDAR_STOP_MODE_KEEP_PLUGINS:保留插件,会加快文件切换速度,默认使用保留插件的模式.aux = CEDAR_STOP_MODE_UNINSTALL_PLUGINS:卸载所有插件,在这种模式下,无缝切换无效。

pbuffer　= NULL。

- **RETURNS**

EPDK_OK：成功;

EPDK_FAIL：失败;

- **DESCRIPTION**

设置 stop play 时播放器是否卸载所有插件，包括 parser、decoder、playbak 等插件和驱动。

- **DEMO**

```
    result = esMODS_MIoctrl(pCedar, CEDAR_CMD_SET_STOP_MODE, cedar_stop_mode, NULL);
```

27.2.90. 2.91.CEDAR_CMD_SET_PITCH

- **PROTOTYPE**

```
    __s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　cedar 模块的句柄;

cmd　　　= CEDAR_CMD_SET_PITCH;

aux　　　=音频变调范围，-12~12。

pbuffer　= NULL。

- **RETURNS**

EPDK_OK：成功；

EPDK_FAIL：失败；

- **DESCRIPTION**

设置音频播放时候变调播放，并设置变调速度

- **DEMO**

```
result = esMODS_MIoctrl(pCedar, CEDAR_CMD_SET_PITCH, aux, NULL);
```

27.2.91. 2.92.CEDAR_CMD_GET_PITCH

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；

cmd　　= CEDAR_CMD_GET_PITCH；

aux　　= 0。

pbuffer = NULL。

- **RETURNS**

音调速度

- **DESCRIPTION**

获取音频变调速度

- **DEMO**

```
result = esMODS_MIoctrl(pCedar, CEDAR_CMD_GET_PITCH, 0, NULL);
```

27.2.92. 2.93.CEDAR_CMD_PLAY_AUX_WAV_FILE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；

cmd　　= CEDAR_CMD_PLAY_AUX_WAV_FILE；

aux　　= __cedar_play_aux_wav_mode_t；

pbuffer = wav 数据播放文件路径。

- **RETURNS**

EPDK_OK：成功；

EPDK_FAIL：失败；

- **DESCRIPTION**

用于在音视频播放的时候播放一些附加的 wav 声音数据，该数据跟解码音频数据混音播放

- **DEMO**

```
result = esMODS_MIoctrl(pCedar, CEDAR_CMD_PLAY_AUX_WAV_FILE, aux, pbuffer);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

27.2.93. 2.94.CEDAR_CMD_PLAY_AUX_WAV_BUFFER

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_PLAY_AUX_WAV_BUFFER；
aux　　= __cedar_play_aux_wav_mode_t；
pbuffer = wav 数据播放参数__cedar_pcm_info_t。

➢ **RETURNS**

EPDK_OK：成功；
EPDK_FAIL：失败；

➢ **DESCRIPTION**

用于在音视频播放的时候播放一些附加的 wav 声音数据，该数据跟解码音频数据混音播放，功能和 CEDAR_CMD_PLAY_AUX_WAV_FILE 一样，只是参数不一样

➢ **DEMO**

```
eLIBs_memset(&pcm_info, 0, sizeof(__cedar_pcm_info_t));
pcm_info.Chan = wav.uChannels;
pcm_info.PcmLen = wav.uSampDataSize/(wav.uBitsPerSample/8)
                                    /wav.uChannels;
pcm_info.preamp = 0;
pcm_info.SampleRate = wav.uSampleRate;
pcm_info.PCMPtr = (__u16*)pwav_buf;
ret = MIoctrl(pCedar, CEDAR_CMD_PLAY_AUX_WAV_BUFFER, aux, &pcm_info);
```

27.2.94. 2.95.CEDAR_CMD_SET_AUX_WAV_BUFFER_SIZE

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_SET_AUX_WAV_BUFFER_SIZE；
aux　　= 设置掌声数据播放 buffer 大小，aux=buffer size；
pbuffer = NULL。

➢ **RETURNS**

EPDK_OK：成功；
EPDK_FAIL：失败；

➢ **DESCRIPTION**

➢ CEDAR_CMD_PLAY_AUX_WAV_BUFFER 播放的附加声音数据的 buffer 大小，音频驱动初始化 时就会按此值分配好内存.

➢ **DEMO**

```
result= esMODS_MIoctrl(pCedar, CEDAR_CMD_SET_AUX_WAV_BUFFER_SIZE, aux, NULL);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

27.2.95. 2.96.CEDAR_CMD_GET_AUX_WAV_BUFFER_SIZE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄;
cmd　　= CEDAR_CMD_GET_AUX_WAV_BUFFER_SIZE;
aux　　= 0;
pbuffer　= NULL。

- **RETURNS**

返回掌声数据播放 buffer 大小

- **DESCRIPTION**

CEDAR_CMD_PLAY_AUX_WAV_BUFFER 播放的附加声音数据的 buffer 大小，音频驱动初始化 时就会按此值分配好内存.

- **DEMO**

```
result= esMODS_MIoctrl(pCedar, CEDAR_CMD_GET_AUX_WAV_BUFFER_SIZE, 0, NULL);
```

27.2.96. 2.97.CEDAR_CMD_GET_PLY_DATA_STATUS

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄;
cmd　　= CEDAR_CMD_GET_PLY_DATA_STATUS;
aux　　= 0;
pbuffer　= NULL。

- **RETURNS**

非 1：没有数据
1：有数据

- **DESCRIPTION**

获取 playback 中一定时间内是否有数据，即是否有音视频数据马上要播放

- **DEMO**

```
result= esMODS_MIoctrl(pCedar, CEDAR_CMD_GET_PLY_DATA_STATUS, 0, NULL);
```

27.2.97. 2.98.CEDAR_CMD_STREAM_SET_INFO

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄;
cmd　　= CEDAR_CMD_STREAM_SET_INFO;
aux　　= 0;

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

pbuffer = 设置 h264/h265 裸流数据信息，__stream_inf_t。

- **RETURNS**

  EPDK_OK：成功；

  EPDK_FAIL：失败；

- **DESCRIPTION**

  应用传输 h264/h265 裸流数据播放前，设置视频信息

- **DEMO**

```
__s32 robin_set_stream_info(__stream_inf_t* streaminf)
{
    return esMODS_MIoctrl( robin_hced, CEDAR_CMD_STREAM_SET_INFO, 0, (void*)streaminf);
}
```

27.2.98. 2.99.CEDAR_CMD_STREAM_QUERY_BUFFER

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp cedar 模块的句柄；

cmd = CEDAR_CMD_STREAM_QUERY_BUFFER；

aux = 0；

pbuffer = NULL。

- **RETURNS**

  返回播放器缓存区域还有几帧数据没有解码，个数

- **DESCRIPTION**

  专用于 h264/h265 裸流数据播放接口

- **DEMO**

```
__s32 robin_set_stream_info(__stream_inf_t* streaminf)
{
    return esMODS_MIoctrl( robin_hced, CEDAR_CMD_STREAM_QUERY_BUFFER, 0, (void*)streaminf);
}
```

27.2.99. 3.10.CEDAR_CMD_STREAM_WRITE_BUFFER

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp cedar 模块的句柄；

cmd = CEDAR_CMD_STREAM_WRITE_BUFFER；

aux = 一帧数据的长度；

pbuffer = 帧数据 buffer,注意数据前面 4 字节代表帧长度，可能需要应用在数据头加上。

- **RETURNS0**

  EPDK_OK：成功；

  EPDK_FAIL：失败；

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

➢ **DESCRIPTION**

专用于 h264/h265 裸流数据播放接口。

➢ **DEMO**

```
__s32 robin_write_buffer(__u32 buffer_len, __u8 * buffer)
{
    return esMODS_MIoctrl( robin_hced, CEDAR_CMD_STREAM_WRITE_BUFFER, buffer_len, (void *)buffer );
}
```

27.2.100. 3.11.CEDAR_CMD_STREAM_END

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp cedar 模块的句柄；
cmd = CEDAR_CMD_STREAM_END；
aux = 0；
pbuffer = NULL。

➢ **RETURNS0**

EPDK_OK：成功；
EPDK_FAIL：失败；

➢ **DESCRIPTION**

专用于 h264/h265 裸流数据播放接口。通知播放器数据传输完毕。

➢ **DEMO**

```
__s32 robin_set_stop(void)
{
    return esMODS_MIoctrl( robin_hced, CEDAR_CMD_STREAM_END, 0, NULL );
}
```

27.2.101. 3.12.CEDAR_CMD_FAST_LOOPPLAY_ENABLE

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp cedar 模块的句柄；
cmd = CEDAR_CMD_FAST_LOOPPLAY_ENABLE；
aux = 0；
pbuffer = NULL。

➢ **RETURNS0**

EPDK_OK：成功；
EPDK_FAIL：失败；

➢ **DESCRIPTION**

使能音频循环播放模式，只支持音频格式。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

➢ **DEMO**

```
void robin_loop_play_enble()
{
    esMODS_MIoctrl(robin_hced, CEDAR_CMD_FAST_LOOPPLAY_ENABLE, 0, NULL);
}
```

27.2.102. 3.13.CEDAR_CMD_FAST_LOOPPLAY_DISENABLE

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp cedar 模块的句柄；
cmd = CEDAR_CMD_FAST_LOOPPLAY_DISENABLE；
aux = 0；
pbuffer = NULL。

➢ **RETURNS0**

EPDK_OK：成功；
EPDK_FAIL：失败；

➢ **DESCRIPTION**

关闭音频循环播放模式，只支持音频格式。

➢ **DEMO**

```
void robin_loop_play_disenble()
{
    esMODS_MIoctrl(robin_hced, CEDAR_CMD_FAST_LOOPPLAY_DISENABLE, 0, NULL);
}
```

27.2.103. 3.14.CEDAR_CMD_SET_FAST_LOOPPLAY_CNT

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp cedar 模块的句柄；
cmd = CEDAR_CMD_SET_FAST_LOOPPLAY_CNT；
aux = 循环播放次数；
pbuffer = NULL。

➢ **RETURNS0**

EPDK_OK：成功；
EPDK_FAIL：失败；

➢ **DESCRIPTION**

设置音频循环播放次数，只支持音频格式。

➢ **DEMO**

```
void robin_set_loop_play_cnt(__s32 count)
{
```

```
    esMODS_MIoctrl(robin_hced, CEDAR_CMD_SET_FAST_LOOPPLAY_CNT, count, NULL);
}
```

27.2.104. 3.15.CEDAR_CMD_GET_FAST_LOOPPLAY_CNT

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　cedar 模块的句柄；
cmd　　= CEDAR_CMD_GET_FAST_LOOPPLAY_CNT；
aux　　= 0；
pbuffer = NULL。

- **RETURNS0**

返回循环次数

- **DESCRIPTION**

获取音频循环播放次数，只支持音频格式。

- **DEMO**

```
__s32 robin_get_loop_play_cnt()
{
    return esMODS_MIoctrl(robin_hced, CEDAR_CMD_GET_FAST_LOOPPLAY_CNT, 0, NULL);
}
```

27.3. 常用功能 DEMO

27.3.1. 音视频播放 demo

```
__s32 test_video(const char* path)
{
    RECT rect;
    __hdle video_layer;

    if( path == NULL || eLIBs_strlen(path) == 0 )
        return -1;

    __log("play video:%s\n", path);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
if( robin_hdis == NULL )
{
        //esDEV_Plugin("\\drv\\audio.drv", 0, 0, 1);
        esKRNL_TimeDly(100);
        volume_module_init();

        disp_open();
        __log("disp_open\n");
        cedar_open();
        __log("cedar_open\n");
        volume_set_volume(AUDIO_DEV_PLAY, 20);

        rect.x          = 0;
        rect.y          = 0;
        rect.width      = eLIBs_fioctrl(robin_hdis, DISP_CMD_SCN_GET_WIDTH, 0, NULL);
        rect.height = eLIBs_fioctrl(robin_hdis, DISP_CMD_SCN_GET_HEIGHT, 0, NULL);
        video_layer = robin_request_video_layer(&rect, 1, 0xff);
        if(video_layer == NULL)
        {
                __log("request video layer failed!\n");
        }
        else
        {
                __log("request video layer OK!\n");
        }
}
//cedar_play_mp4("E:\\20000373.mp4");
//cedar_play_mp4("http://bbhlt.shoujiduoduo.com/bb/video/first/20000373.mp4");
cedar_play_mp4((char*)path);

return 0;
}
```

Path 为音频或者视频文件路径。具体实现流程可以参考 livedesk\beetles\shell\Esh_builtins\commands\dohelp.c，或者 app_movie.c。

27.3.2. Wav/Pcm 音频播放 demo

```
__s32 dsk_keytone_init( const char *keytone_file )
{
ES_FILE                 *pfile2;
__audio_dev_para_t      pbuf2;
__wave_header_t1        wav;

    #if 0==EPDK_AUDIO_READY
```

```
    return EPDK_OK;
#endif

keytone = (DKTone *)g_malloc(sizeof(DKTone));
if( !keytone )
{
    __err(" DKTone malloc error \n");
    return EPDK_FAIL;
}

keytone->f_audiodev = eLIBs_fopen("b:\\AUDIO\\PLAY", "r+");

keytone->state      = SET_KEYTONE_ON;

pfile2 = eLIBs_fopen(keytone_file, "rb");
    if(pfile2 == 0)
    {
    __err("%s cannot open \n", keytone_file);
    return EPDK_OK;
    }

eLIBs_fread(&wav, 1, sizeof(__wave_header_t1), pfile2);
    g_wav_sample_size = wav.uSampDataSize;
keytone->wavsize = wav.uSampDataSize;

    keytone->tonebuf     =    (__u8    *)esMEMS_Palloc(((g_wav_sample_size+1023)    >>    10),
MEMS_PALLOC_MODE_BND_NONE | MEMS_PALLOC_MODE_BNK_NONE);
if( !keytone->tonebuf )
{
    __err(" tonebuf malloc error \n");
    return EPDK_FAIL;
}

    eLIBs_fread(keytone->tonebuf, 1, g_wav_sample_size, pfile2);
    eLIBs_fclose(pfile2);

pbuf2.bps    = wav.uBitsPerSample;
pbuf2.chn    = wav.uChannels;
pbuf2.fs     = wav.uSampleRate;
eLIBs_fioctrl(keytone->f_audiodev, AUDIO_DEV_CMD_SET_PARA, 0, &pbuf2);
eLIBs_fioctrl(keytone->f_audiodev, AUDIO_DEV_CMD_REG_USERMODE, AUDIO_PLAY_USR_KEY, 0);
eLIBs_fioctrl(keytone->f_audiodev, AUDIO_DEV_CMD_START, 0, 0);
    return EPDK_OK;
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
__s32 dsk_keytone_on(void)
{
    #if 0==EPDK_AUDIO_READY
    return EPDK_OK;
 #endif

 if( (keytone->state == SET_KEYTONE_ON) && (keytone->tonebuf) && (keytone->f_audiodev) )
    {
        eLIBs_fioctrl(keytone->f_audiodev, AUDIO_DEV_CMD_FLUSH_BUF, 0, 0);
        eLIBs_fwrite(keytone->tonebuf, 1, keytone->wavsize, keytone->f_audiodev);
    }
    return EPDK_OK;
}
```

Pcm/Wav 音频播放可以直接通过往音频播放设备写数据实现，流程简单。具体参考 livedesk\beetles\mod_desktop\functions\keytone\dsk_keytone.c。

27.3.3. H264 裸流数据播放 demo

```
__s32 video_decode_play(__u8 *buf, __u32 size)
{
 __s32 ret;
 __u8 *head = NULL;
 __u8  *file_buf = NULL;
 eLIBs_printf("P-V: %d\n", size);

 file_buf = esMEMS_Malloc(0, 1*1024*1024);
 if(file_buf == NULL)
 {
       __log("error file_buf == NULL\n");
 }
 eLIBs_memset(file_buf, 0x0, 1*1024*1024);

    while(1)
    {
        ret = robin_query_buffer();
        //__msg("ret==0x%x==%d\n", ret, ret);
        //if(ret >= (len+4))
        if(ret <= 10)
        {
            break;
        }
        __log(" **********robin_query_buffer failed*************\n");
        esKRNL_TimeDly(2);
```

```
    }

head = (__u8*)&size;
memcpy(file_buf,head,4);
memcpy(file_buf+4,buf,size);
    ret = robin_write_buffer(size+4, file_buf);
    //__log("app time[%d]\n", esKRNL_TimeGet()*10);
    //esKRNL_TimeDly(5);
    esMEMS_Mfree(0, file_buf);

return EPDK_OK;
}
```

具体流程参考\livedesk\beetles\applets\play\ios_h264.c

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

28. Willow Module

28.1. Introduction

28.1.1. Description

Melis 操作系统是由全志科技自主研发的一套精简、高效、易用的实时多任务操作系统。Willow 中间件是基于 Melis 系统上开发的图片解码/显示插件，用于支持应用对图片上的各种功能的需求，免除应用对具体图片格式的了解，隔离和底层系统和驱动的操作，让图片的相关的应用程序变得简单。

28.1.2. Purpose

本文档主要讲述 Willow 插件的编程接口，让开发者能快速掌握 Melis 系统上的图片解码，图片显示，图像处理的操作，并基于 Willow 来开发自己的图片相关应用。

28.1.3. Reference

1）读者可以先了解一些关于颜色空间，像素格式等基本概念，以便更加快速的掌握 Willow 中间件的相关接口，加快应用开发速度。

2）读者可以从《SW1103REF005_MELIS PROGRAM GUIDE_Module.pdf》文档了解模块的安装，卸载，打开，关闭以及模块操作命令。

28.2. Willow

图片中间件 willow 用于支持应用对图片上的各种功能的需求，免除应用对具体图片格式的了解，隔离和底层系统和驱动的操作，让图片的相关的应用程序变得简单。包括以下几个方面的功能：

1）支持对多种图片格式的解析、解码支持，包括 jpeg/bmp/png/gif 格式，并能够方便的添加其他格式的图片解码支持；支持 jpeg 图片硬件解码，提供高速解码性能。

2）支持从音乐文件中解码出专辑图片；

3）支持图片的显示，并支持对图片的缩小/放大/旋转功能；

4）支持图片缩略图的获取，图片基本信息的获取；

5）在图片浏览时支持多种图片切换效果，包括：淡入淡出、马赛克、缩小、放大、水平/垂直百叶窗、向上/向下/向左/向右滑入、向上/向下/向左/向右展开；

6）支持图片的自动旋转功能；

7）在图片浏览时支持多种浏览模式，包括：原始尺寸，剪裁全屏模式，拉伸全屏模式，自动模式。

本文档只介绍模块操作的控制命令字，如果您希望了解模块的安装，卸载，打开，关闭以及模块操作命令，可以阅读《SW1103REF005_MELIS PROGRAM GUIDE_Module.pdf》文档。

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

28.2.1. WILLOW_CMD_QUERY_STATUS

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　willow 模块的句柄；
cmd　　= WILLOW_CMD_QUERY_STATUS;
aux　　= 0;
pbuffer　= 0;

- **RETURNS**

Willow 当前的状态：

WILLOW_STATUS_INIT，　　加载后的初始化状态；
WILLOW_STATUS_BUSY，　　Willow 有命令未执行完毕；
WILLOW_STATUS_READY，　　Willow ready，目前未使用该状态；
WILLOW_STATUS_FINISH，　Willow 所有命令执行完毕；

- **DESCRIPTION**

查询 Willow 中间件当前的状态；

- **DEMO**

```
//查询 willow 状态，等待命令执行完毕
static void WaitWillowReady(__mp    * hWillow)
{
    __u8 status;
    while(1)
    {
        status = esMODS_MIoctrl(hWillow, WILLOW_CMD_QUERY_STATUS, 0, 0);
        if (status != WILLOW_STATUS_BUSY)
        {
            break;
        }
        esKRNL_TimeDly(1);
    }
}
```

28.2.2. WILLOW_CMD_GET_THUMBS

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　willow 模块的句柄；
cmd　　= WILLOW_CMD_GET_THUMBS;
aux　　= 0;

pbuffer = __willow_get_thumbs_param_t *， 输入获取缩略图的信息，包括文件名，缩略图的尺寸，buffer 地址等；

- **RETURNS**

设置缩略图信息的结果：

EPDK_OK：设置成功；

EPDK_FAIL：设置失败；

- **DESCRIPTION**

设置获取缩略图所需要的信息；

注意，该命令必须和 WILLOW_CMD_START_SHOW(WILLOW_CMD_START_SHOW_EXT) 或者 WILLOW_CMD_GET_IMG_INFO 配合使用；连续单独调用两次 WILLOW_CMD_GET_THUMBS 命令，第二次调用会返回失败。

- **DEMO**

```
/**************************************************************
* Function name: demoGetThumbnail
* Description:
*     获取图片缩略图
*
* Parameters:
*     @param hWillow:         willow 模块的句柄
*
* Return:
*     @return: on success return EPDK_OK otherwise return EPDK_FAIL.
* Time: 2011/06/24
**************************************************************/
__s32 demoGetThumbnail(__mp *hWillow)
{
    __willow_get_thumbs_param_t thumbInfo;
    __s32 ret;
    __s32 bufSize;
    char filename[512];

    //从播放列表获取文件名
    ret = GetFileFromPlaylist(filename, 0);
    if (ret != EPDK_OK)
    {
        return EPDK_FAIL;
    }

    /*根据实际应用初始化参数*/
    thumbInfo.filename = filename;
    thumbInfo.format = RGB_SEQ_ARGB;
    thumbInfo.size.width = 100;
    thumbInfo.size.height = 76;
    bufSize = thumbInfo.size.width * thumbInfo.size.height * 4;
```

```
    thumbInfo.buf = (__u8*) esMEMS_Palloc((bufSize + 1023)/1024, 0);
    if (thumbInfo.buf == NULL)
    {
        __wrn("err while palloc!\n");
        return EPDK_FAIL;
    }

    /*解码图片，获取缩略图*/
    WaitWillowReady();
    ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_GET_THUMBS, 0, &thumbInfo);
    if (EPDK_FAIL == ret)
    {
        __wrn("set thumbs info err!\n");
        goto EXIT_GETALBUMN;
    }
    ret = esMODS_MIoctrl(hWillow,WILLOW_CMD_CHECK_IMG,0,0);
    if (EPDK_FAIL == ret)
    {
        __wrn("image not support!\n");
        goto EXIT_GETALBUMN;
    }
    ret = esMODS_MIoctrl(hWillow,WILLOW_CMD_START_SHOW,0,0);
    WaitWillowReady();

    /*此处可以对缩略图操作*/
    ret = ShowThumbs(&thumbInfo);

EXIT_GETTHUMB:
    if (thumbInfo.buf != NULL)
    {
        esMEMS_Free(thumbInfo.buf, (bufSize + 1023)/1024);
    }
    return ret;
}
```

28.2.3. WILLOW_CMD_SET_ALBUM_ART

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　willow 模块的句柄；

cmd　　= WILLOW_CMD_SET_ALBUM_ART；

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

aux = 0;

pbuffer = __willow_get_albumart_param_t *， 专辑图片的信息，包括专辑图片在文件中的偏移，专辑图片的格式等;

- **RETURNS**

设置专辑图片信息的结果:

EPDK_OK: 设置成功;

EPDK_FAIL: 设置失败;

- **DESCRIPTION**

设置获取专辑图片缩略图所需要的信息;

注意，该命令必须和 WILLOW_CMD_START_SHOW(WILLOW_CMD_START_SHOW_EXT) 或者 WILLOW_CMD_GET_IMG_INFO 配合使用; 连续单独调用两次 WILLOW_CMD_SET_ALBUM_ART 命令，第二次调用会返回失败。

- **DEMO**

```
/*****************************************************************
* Function name: demoGetAlbumart
* Description:
*     获取专辑图片的缩略图
*
* Parameters:
*     @param hWillow:         willow 模块的句柄
*
* Return:
*     @return: on success return EPDK_OK otherwise return EPDK_FAIL.
* Time: 2011/06/24
*****************************************************************/
__s32 demoGetAlbumart(__mp *hWillow)
{
    __willow_get_albumart_param_t albumartInfo;
    __s32 ret;
    __s32 bufPage;
    char filename[512];

    //从播放列表获取文件名
    ret = GetFileFromPlaylist(filename, 0);
    if (ret != EPDK_OK)
    {
        return EPDK_FAIL;
    }

    /*根据实际应用初始化参数*/
    albumartInfo.thumbs.filename = filename;
    albumartInfo.thumbs.format = RGB_SEQ_ARGB;
    albumartInfo.thumbs.size.width = 100;
```

```
    albumartInfo.thumbs.size.height = 76;
    bufPage = albumartInfo.thumbs.size.width
                * albumartInfo.thumbs.size.height * 4;
    bufPage = (bufPage + 1023)/1024;
    albumartInfo.thumbs.buf = (__u8*)esMEMS_Palloc(bufPage, 0);
    if (albumartInfo.thumbs.buf == NULL)
    {
        __wrn("err while palloc!\n");
        return EPDK_FAIL;
    }
    /*以下内容根据具体的音乐文件 id3 信息解析获得*/
    albumartInfo.album_art_info.enable = EPDK_TRUE;
    albumartInfo.album_art_info.img_type = IMG_FORMAT_JPG;
    albumartInfo.album_art_info.length = 0x1000;
    albumartInfo.album_art_info.offset = 0x200;

    /*解码图片，获取缩略图*/
    WaitWillowReady();
    ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_SET_ALBUM_ART,
                0, &albumartInfo);
    if (EPDK_FAIL == ret)
    {
        __wrn("set thumbs info err!\n");
        goto EXIT_GETALBUMN;
    }
    ret = esMODS_MIoctrl(hWillow,WILLOW_CMD_CHECK_IMG,0,0);
    if (EPDK_FAIL == ret)
    {
        __wrn("image not support!\n");
        goto EXIT_GETALBUMN;
    }
    ret = esMODS_MIoctrl(hWillow,WILLOW_CMD_START_SHOW,0,0);
    WaitWillowReady();

    /*此处可以对缩略图操作*/
    ret = ShowThumbs(&albumartInfo.thumbs);

EXIT_GETALBUMN:
    if (albumartInfo.thumbs.buf != NULL)
    {
        esMEMS_Free(albumartInfo.thumbs.buf, bufPage);
    }
    return ret;
}
```

28.2.4. WILLOW_CMD_SHOW_IMG_FROM_FILE

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　willow 模块的句柄；

cmd　　　= WILLOW_CMD_SHOW_IMG_FROM_FILE；

aux　　　= 0；

pbuffer　= __willow_show_file_param_t *，　浏览图片的文件信息；

➢ **RETURNS**

设置文件信息的结果：

EPDK_OK：设置成功；

EPDK_FAIL：设置失败；

➢ **DESCRIPTION**

设置浏览图片的文件的信息；

注意，该命令必须和 WILLOW_CMD_START_SHOW(WILLOW_CMD_START_SHOW_EXT) 或者 WILLOW_CMD_GET_IMG_INFO 配合使用；连续单独调用两次 WILLOW_CMD_SHOW_IMG_FROM_FILE 命令，第二次调用会返回失败。

➢ **DEMO**

```
/*****************************************************************
* Function name: demoShowFile
* Description:
*     浏览图片文件
*
* Parameters:
*     @param hWillow:         willow 模块的句柄
*
* Return:
*     @return: on success return EPDK_OK otherwise return EPDK_FAIL.
* Time: 2011/06/24
*****************************************************************/
__s32 demoShowFile(__mp *hWillow)
{
    __willow_show_file_param_t fileInfo;
    __s32 retval;
    char filename[512];

    fileInfo.filename = filename;
    esMODS_MIoctrl(hWillow, WILLOW_CMD_SET_SWITCH_MODE,
                FADE_IN_OUT, NULL);
```

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved　

```
        retval = EPDK_OK;
        fileInfo.img_no = 0;
        while(retval == EPDK_OK)
        {
            //从播放列表获取文件名
            retval = GetFileFromPlaylist(filename, fileInfo.img_no);
            if (retval != EPDK_OK)
            {
                break;
            }
            /*显示图片*/
            WaitWillowReady();
            esMODS_MIoctrl(hWillow, WILLOW_CMD_SHOW_IMG_FROM_FILE,
                        0, &fileInfo);
            retval = esMODS_MIoctrl(hWillow, WILLOW_CMD_CHECK_IMG, 0, 0);
            if (retval == EPDK_OK)
            {
                esMODS_MIoctrl(hWillow, WILLOW_CMD_START_SHOW, 0, 0);
                WaitWillowReady();
            }
            retval = EPDK_OK;
            fileInfo.img_no++;
        }
        return EPDK_OK;
    }
```

28.2.5. WILLOW_CMD_SHOW_IMG_FROM_BUFFER

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　willow 模块的句柄；
cmd　　　= WILLOW_CMD_SHOW_IMG_FROM_BUFFER;
aux　　　= 0;
pbuffer = __willow_show_file_param_t *, 浏览图片的文件信息；

- **RETURNS**

设置 ARGB 数据信息的结果：

EPDK_OK：设置成功；

EPDK_FAIL：设置失败；

- **DESCRIPTION**

设置浏览图片的文件的信息；

注意，该命令必须和 WILLOW_CMD_START_SHOW(WILLOW_CMD_START_SHOW_EXT) 或者 WILLOW_CMD_GET_IMG_INFO 配合使用；连续单独调用两次 2.5. WILLOW_CMD_SHOW_IMG_FROM_BUFFER 命令，第二次调用会返回失败。

- **DEMO**

```
/*****************************************************************
* Function name: demoShowARGB
* Description:
*     浏览 ARGB 数据
*
* Parameters:
*     @param hWillow:        willow 模块的句柄
*     @param pdata:          ARGB 数据 buffer
*     @param psize:          ARGB 数据的宽高信息
*
* Return:
*     @return: on success return EPDK_OK otherwise return EPDK_FAIL.
* Time: 2011/06/24
*****************************************************************/
__s32 demoShowARGB(__mp *hWillow, __u8* pdata, __rectsz_t *psize)
{
    __willow_show_buffer_param_t ARGBInfo;
    __s32 retval;

    eLIBs_memset(&ARGBInfo, 0, sizeof(ARGBInfo));
    ARGBInfo.img_size.height = psize->height;
    ARGBInfo.img_size.width = psize->width;
    ARGBInfo.img_buf = pdata;
    ARGBInfo.buf_size = psize->height * psize->width;

    /*显示 ARGB 数据*/
    WaitWillowReady();
    esMODS_MIoctrl(hWillow, WILLOW_CMD_SHOW_IMG_FROM_BUFFER,
                0, &ARGBInfo);
    retval = esMODS_MIoctrl(hWillow, WILLOW_CMD_CHECK_IMG, 0, 0);
    if (retval == EPDK_OK)
    {
        esMODS_MIoctrl(hWillow, WILLOW_CMD_START_SHOW, 0, 0);
        WaitWillowReady();
    }
    return retval;
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

28.2.6. WILLOW_CMD_SET_SWITCH_MODE

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp willow 模块的句柄；
cmd = WILLOW_CMD_SET_SWITCH_MODE;
aux = __willow_switch_mode_t;
pbuffer = 0;

其中，切换效果具体包含以下几种

DEFAULT_SWITCH:	无切换效果
FADE_IN_OUT:	淡入淡出
PERSIAN_BLIND_H :	水平百叶窗
PERSIAN_BLIND_V:	垂直百叶窗
SLID_UP:	向上滑动
SLID_DOWN:	向下滑动
SLID_LEFT:	向左滑动
SLID_RIGHT:	向右滑动
STRETCH_UP:	向上展开
STRETCH_DOWN:	向下展开
STRETCH_LEFT:	向左展开
STRETCH_RIGHT:	向右展开
MOSAIC:	马赛克
ROOM_IN:	缩小
ROOM_OUT:	放大
TOUCH_SLID_H:	水平方向拖动
TOUCH_SLID_V:	垂直方向拖动

➢ **RETURNS**

设置切换模式的结果：

EPDK_OK：设置成功；

EPDK_FAIL：设置失败；

➢ **DESCRIPTION**

设置浏览图片时，图片切换时的切换效果；

➢ **DEMO**

```
//设置切换模式
__s32   result;
if(hWillow != NULL)
{
    result = esMODS_MIoctrl(hWillow, WILLOW_CMD_SET_SWITCH_MODE,
                FADE_IN_OUT, NULL);
    if(result != EPDK_OK)
    {
        __wrn("Try to set switch mode failed!\n");
```

```
        }
    }
```

28.2.7. WILLOW_CMD_GET_SHOW_PARAM

- **PROTOTYPE**

```
    __s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　willow 模块的句柄；
cmd　　　= WILLOW_CMD_GET_SHOW_PARAM；
aux　　　= 0；
pbuffer　= __willow_showing_img_info_t *, 当前显示的信息；

- **RETURNS**

获取显示参数的结果：

EPDK_OK：成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

获取当前显示的参数，包括 display 驱动句柄，图层，当前显示部分在原图中的位置等。

- **DEMO**

```
    //设置切换模式
    __s32    result;
    __willow_showing_img_info_t showInfo;
    if(hWillow != NULL)
    {
        result = esMODS_MIoctrl(hWillow, WILLOW_CMD_GET_SHOW_PARAM,
                    0, &showInfo);
        if(result != EPDK_OK)
        {
            __wrn("Try to get show para failed!\n");
        }
    }
```

28.2.8. WILLOW_CMD_GET_IMG_INFO

- **PROTOTYPE**

```
    __s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　willow 模块的句柄；
cmd　　　= WILLOW_CMD_GET_IMG_INFO；
aux　　　= 0；
pbuffer　= __willow_img_info_t *, 图片信息；

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

- **RETURNS**

  获取图片信息的结果：

  EPDK_OK：成功；

  EPDK_FAIL：失败；

- **DESCRIPTION**

  获取图片的信息，图片的文件名由 WILLOW_CMD_GET_THUMBS / WILLOW_CMD_SHOW_IMG_FROM_FILE / WILLOW_CMD_SET_ALBUM_ART 命令设置。

- **DEMO**

```
/****************************************************************
* Function name: DemoGetImageInfo
* Description:
*     获取图片信息
*
* Parameters:
*     @param hWillow:         willow 模块的句柄
*
* Return:
*     @return: on success return EPDK_OK otherwise return EPDK_FAIL.
* Time: 2011/06/24
****************************************************************/
__s32 DemoGetImageInfo(__mp *hWillow)
{
    __willow_show_file_param_t fileInfo;
    __s32 ret;
    char filename[512];
    __willow_img_info_t imgInfo;

    fileInfo.filename = filename;

    ret = EPDK_OK;
    fileInfo.img_no = 0;
    //从播放列表获取文件名
    ret = GetFileFromPlaylist(filename, fileInfo.img_no);
    if (ret != EPDK_OK)
    {
        break;
    }

    /*获取图片信息*/
    WaitWillowReady();
    esMODS_MIoctrl(hWillow, WILLOW_CMD_SHOW_IMG_FROM_FILE,
                0, &fileInfo);
    ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_GET_IMG_INFO, 0, &imgInfo);
    if (ret == EPDK_OK)
```

```
    {
        __wrn("filename:%s, image size(w-h):%d-%d\n",
                    imgInfo.name,
                    imgInfo.size.width,
                    imgInfo.size.height);
    }
    return ret;
}
```

28.2.9. WILLOW_CMD_SCALE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　willow 模块的句柄；
cmd　　= WILLOW_CMD_SCALE；
aux　　= -3,-2,-1,0,1,2,3；分别代表缩小 8/4/2 倍/原始大小/放大 2/4/8 倍
pbuffer　= NULL；

- **RETURNS**

图像操作的结果：

EPDK_OK：成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

放大/缩小当前显示的图片。

- **DEMO**

```
// 将正在浏览的图片放大 4 倍显示
ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_SCALE, 2, NULL);
if (ret == EPDK_FAIL)
{
    __wrn("scale image failed!\n");
}
```

28.2.10. WILLOW_CMD_ROTATE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　willow 模块的句柄；
cmd　　= WILLOW_CMD_ROTATE；
aux　　= -270, -180, -90, 0, 90, 180, 270，负数表示逆时针,正数表示顺时针

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

pbuffer = NULL;

- **RETURNS**

图像操作的结果：

EPDK_OK：成功；

EPDK_FAIL：失败；

- **DESCRIPTION**

放大/缩小当前显示的图片。

- **DEMO**

```
// 将正在浏览的图片顺时针旋转 90 度
ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_ROTATE, 90, NULL);
if (ret == EPDK_FAIL)
{
    __wrn("rotate image failed!\n");
}
```

28.2.11. WILLOW_CMD_MOVE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp willow 模块的句柄；

cmd = WILLOW_CMD_MOVE;

aux = 0

pbuffer = (__pos_t *)，移动图像的偏移；

- **RETURNS**

图像操作的结果：

EPDK_OK：成功；

EPDK_FAIL：失败；

- **DESCRIPTION**

移动图片显示的窗口，在图片放大后有效。

- **DEMO**

```
// 将放大后的图片移动向右移动 20 像素，向下移动-40 像素
__pos_t offset;
offset.x = 20;
offset.y = -40;
ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_MOVE, 0, &offset);
if (ret == EPDK_FAIL)
{
    __wrn("move image failed!\n");
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

28.2.12. WILLOW_CMD_COME_BACK

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　willow 模块的句柄；
cmd　　= WILLOW_CMD_COME_BACK；
aux　　= 0
pbuffer = NULL；

- **RETURNS**

图像操作的结果：

EPDK_OK：成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

图片放大后，返回到图片原始显示的大小，等效于 WILLOW_CMD_SCALE 输入为 0。

- **DEMO**

```
//图片放大后，返回到图片原始显示的大小
ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_COME_BACK, 0, NULL);
if (ret == EPDK_FAIL)
{
    __wrn("comeback to original size failed!\n");
}
```

28.2.13. WILLOW_CMD_OPEN_SHOW

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　willow 模块的句柄；
cmd　　= WILLOW_CMD_OPEN_SHOW；
aux　　= 0
pbuffer = NULL；

- **RETURNS**

图像操作的结果：

EPDK_OK：成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

打开图片浏览，该命令会申请图片显示时需要的 frame buffer 内存资源。应用也可以不调用该命令，willow 中间件会在显示图片前检查资源是否分配，若资源没有分配，会自动分配资源。

- **DEMO**

```
//准备图片浏览资源
ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_OPEN_SHOW, 0, NULL);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
if (ret == EPDK_FAIL)
{
    __wrn("open show failed!\n");
}
```

28.2.14. WILLOW_CMD_SHUT_SHOW

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp   willow 模块的句柄；
cmd   = WILLOW_CMD_SHUT_SHOW；
aux   = 0
pbuffer = NULL；

- **RETURNS**

图像操作的结果：

EPDK_OK：成功；

EPDK_FAIL：失败；

- **DESCRIPTION**

释放图片浏览占用的资源，注意，该命令仅仅只会释放图片浏览时需要的资源，解码图片需要的内存并不会被释放。

- **DEMO**

```
//准备图片浏览资源
ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_SHUT_SHOW, 0, NULL);
if (ret == EPDK_FAIL)
{
    __wrn("shut show failed!\n");
}
```

28.2.15. WILLOW_CMD_CHECK_IMG

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp   willow 模块的句柄；
cmd   = WILLOW_CMD_CHECK_IMG；
aux   = 0
pbuffer = NULL；

- **RETURNS**

图片是否支持：

EPDK_OK：支持；

EPDK_FAIL：不支持；

- **DESCRIPTION**

检查设置的图片是否支持。

- **DEMO**

```
//准备图片浏览资源
/*显示图片*/
esMODS_MIoctrl(hWillow, WILLOW_CMD_SHOW_IMG_FROM_FILE,
            0, &fileInfo);
retval = esMODS_MIoctrl(hWillow, WILLOW_CMD_CHECK_IMG, 0, 0);
if (retval != EPDK_OK)
{
    __wrn("image not supported!\n");
}
```

28.2.16. WILLOW_CMD_START_DEC

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**
- **RETURNS**
- **DESCRIPTION**

废弃的命令，未使用。

28.2.17. WILLOW_CMD_START_SHOW

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　willow 模块的句柄；
cmd　　= WILLOW_CMD_START_SHOW；
aux　　= 0
pbuffer　= NULL；

- **RETURNS**

显示图片/获取缩略图的结果：

EPDK_OK：成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

开始显示图片或者解码缩略图。

- **DEMO**

参考 WILLOW_CMD_GET_THUMBS/ WILLOW_CMD_SHOW_IMG_FROM_FILE 命令的 demo

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

28.2.18. WILLOW_CMD_STOP

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　willow 模块的句柄；
cmd　　 = WILLOW_CMD_STOP；
aux　　 = 0
pbuffer = NULL；

- **RETURNS**

操作的结果：

EPDK_OK：成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

停止解码/显示图片。

- **DEMO**

```
//准备图片浏览资源
/*显示图片*/
esMODS_MIoctrl(hWillow, WILLOW_CMD_STOP, 0, NULL);
//卸载 willow 插件…
```

28.2.19. WILLOW_CMD_SET_SCN

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　willow 模块的句柄；
cmd　　 = WILLOW_CMD_MOVE；
aux　　 = 0
pbuffer = (__rect_t *)，显示区域；

- **RETURNS**

图像操作的结果：

EPDK_OK：成功；
EPDK_FAIL：失败；

- **DESCRIPTION**

设置图片浏览时候的显示区域。

- **DEMO**

```
// 设置图片显示的区域为(0,0,400,240)
__rect_t displayRect;
displayRect.x = 0;
displayRect.y = 0;
displayRect.width = 400;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
    displayRect.height = 240;
    ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_SET_SCN, 0, &displayRect);
    if (ret == EPDK_FAIL)
    {
        __wrn("set display area failed!\n");
    }
```

28.2.20. WILLOW_CMD_START_SHOW_EXT

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　willow 模块的句柄;

cmd　　　= WILLOW_CMD_START_SHOW_EXT;

aux　　　= 0

pbuffer　= (__u32 *)(fill_mode | clear_flag),

　fill_mode 可以为:

　　WILLOW_SCALE_RATIO, 按比例缩小至指定尺寸，不能溢出,图片尺寸小于指定尺寸时，保持图片尺寸不变　;

　　WILLOW_SCALE_STRETCH, 以窗口的比例缩放图片至满窗口显示，可能会变形

　　WILLOW_SCALE_RATIO_SCALE, 按比例缩小至指定尺寸，不能溢出，图片尺寸小于指定尺寸时，放大图片以适应尺寸

　　WILLOW_SCALE_RATIO_CUTOFF，以图片的比例缩放图片至满窗口显示，可能会有部分图片溢出窗口(不可见);

　clear_flag 可以为　:

　　WILLOW_SCALE_CLEAR_BK，缩略图未填充满时，填充剩余部分为黑色

　　WILLOW_SCALE_CLEAR_NONE，缩略图未填充满时，不填充背景色，由应用自己填充

➢ **RETURNS**

EPDK_OK：成功;

EPDK_FAIL：失败;

➢ **DESCRIPTION**

开始获取缩略图，和 WILLOW_CMD_START_SHOW 命令的区别在于：1）只能用于缩略图；2）可以对缩略图的属性进行设置。

➢ **DEMO**

```
    /****************************************************************
    * Function name: demoGetThumbnailExt
    * Description:
    *    以 WILLOW_SCALE_RATIO_SCALE 模式获取图片缩略图，填充背景色为透明
    *
    * Parameters:
    *    @param hWillow:         willow 模块的句柄
    *
    * Return:
```

```
*    @return: on success return EPDK_OK otherwise return EPDK_FAIL.
* Time: 2011/06/24
*****************************************************************************/
__s32 demoGetThumbnailExt(__mp *hWillow)
{
    __willow_get_thumbs_param_t thumbInfo;
    __s32 ret;
    __s32 bufSize;
    char filename[512];

    //从播放列表获取文件名
    ret = GetFileFromPlaylist(filename, 0);
    if (ret != EPDK_OK)
    {
        return EPDK_FAIL;
    }

    /*根据实际应用初始化参数*/
    thumbInfo.filename = filename;
    thumbInfo.format = RGB_SEQ_ARGB;
    thumbInfo.size.width = 100;
    thumbInfo.size.height = 76;
    bufSize = thumbInfo.size.width * thumbInfo.size.height * 4;
    thumbInfo.buf = (__u8*) esMEMS_Palloc((bufSize + 1023)/1024, 0);
    if (thumbInfo.buf == NULL)
    {
        __wrn("err while palloc!\n");
        return EPDK_FAIL;
    }
    /*将所有的值清为 0xFF，即背景色为白色，全透明*/
    eLIBs_memset(thumbInfo.buf, 0xFF, bufSize）;

    /*解码图片，获取缩略图*/
    WaitWillowReady();
    ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_GET_THUMBS, 0, &thumbInfo);
    if (EPDK_FAIL == ret)
    {
        __wrn("set thumbs info err!\n");
        goto EXIT_GETALBUMN;
    }
    ret = esMODS_MIoctrl(hWillow,WILLOW_CMD_CHECK_IMG,0,0);
    if (EPDK_FAIL == ret)
    {
        __wrn("image not support!\n");
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
        goto EXIT_GETALBUMN;
    }
    ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_START_SHOW_EXT, 0,
        (__u32 *)(WILLOW_SCALE_RATIO_SCALE | WILLOW_SCALE_CLEAR_NONE));
    WaitWillowReady();

    /*此处可以对缩略图操作*/
    ret = ShowThumbs(&thumbInfo);

EXIT_GETTHUMB:
    if (thumbInfo.buf != NULL)
    {
        esMEMS_Free(thumbInfo.buf, (bufSize + 1023)/1024);
    }
    return ret;
}
```

28.2.21. WILLOW_CMD_CATCH_SCREEN

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　willow 模块的句柄；
cmd　　= WILLOW_CMD_CATCH_SCREEN；
aux　　= 0
pbuffer　= (FB *)；

- **RETURNS**

  EPDK_OK：成功；
  EPDK_FAIL：失败；

- **DESCRIPTION**

获取当前显示图像的 framebuffer 信息，目前只能转换为 yuv444 planar 格式或者 ARGB 格式的 framebuffer。

- **DEMO**

```
// 获取当前显示图像的 framebuffer
FB curShowFrame;
curShowFrame.fmt.type = FB_TYPE_YUV;
ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_CATCH_SCREEN, 0, & curShowFrame);
if (ret == EPDK_FAIL)
{
    __wrn("get display frame failed!\n");
}
```

28.2.22. WILLOW_CMD_AUTO_ROTATE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp   willow 模块的句柄;
cmd   = WILLOW_CMD_AUTO_ROTATE;
aux   = __willow_rotate_mode_t
pbuffer = NULL;

- **RETURNS**

EPDK_OK：成功;
EPDK_FAIL：失败;

- **DESCRIPTION**

当高度 > 宽度时，是否需要自动旋转显示。

- **DEMO**

```
//设置当高度 > 宽度时，自动顺时针旋转 90 度显示
ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_AUTO_ROTATE,
    PSHOW_ROTATE_RIGHT, NULL);
if (ret == EPDK_FAIL)
{
    __wrn("set auto rotate failed!\n");
}
```

28.2.23. WILLOW_CMD_AUTO_ROTATE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp   willow 模块的句柄;
cmd   = WILLOW_CMD_AUTO_ROTATE;
aux   = __willow_rotate_mode_t
pbuffer = NULL;

- **RETURNS**

EPDK_OK：成功;
EPDK_FAIL：失败;

- **DESCRIPTION**

当高度 > 宽度时，是否需要自动旋转显示。

- **DEMO**

```
//设置当高度 > 宽度时，自动顺时针旋转 90 度显示
ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_AUTO_ROTATE,
    PSHOW_ROTATE_RIGHT, NULL);
if (ret == EPDK_FAIL)
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
    {
        __wrn("set auto rotate failed!\n");
    }
```

28.2.24. WILLOW_CMD_SET_SHOW_MODE

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　willow 模块的句柄；

cmd　　　= WILLOW_CMD_SET_SHOW_MODE；

aux　　　= __willow_image_show_mode_t，包括

WILLOW_IMAGE_SHOW_ORISIZE：以图片原始大小在窗口内显示，不能溢出窗口图片尺寸小于窗口时，保持图片尺寸不变；

WILLOW_IMAGE_SHOW_STRETCH：以窗口的比例缩放图片至满窗口显示，可能会变形；

WILLOW_IMAGE_SHOW_CUTOFF：以图片的比例缩放图片至满窗口显示，可能会有部分图片溢出窗口(不可见)；

WILLOW_IMAGE_SHOW_AUTO：

WILLOW_IMAGE_SHOW_RATIO,以图片的比例缩放图片至满窗口显示，不能溢出窗口，图片尺寸小于窗口时，会放大至合适尺寸

pbuffer　= NULL；

- **RETURNS**

EPDK_OK：成功；

EPDK_FAIL：失败；

- **DESCRIPTION**

设置显示的模式。

- **DEMO**

```
// 以 WILLOW_IMAGE_SHOW_RATIO 模式显示图片
ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_SET_SHOW_MODE,
    WILLOW_IMAGE_SHOW_RATIO, NULL);
if (ret == EPDK_FAIL)
{
    __wrn("set image show mode failed!\n");
}
```

28.2.25. WILLOW_CMD_CFG_LYR

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　　willow 模块的句柄；

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

cmd   = WILLOW_CMD_CFG_LYR;
aux   = __willow_lyr_cfg_t;
pbuffer = NULL;

- **RETURNS**

    EPDK_OK：成功；

    EPDK_FAIL：失败；

- **DESCRIPTION**

设置 willow 使用的图层的属性。

- **DEMO**

```
// 设置 willow 使用的图层在 pipe0
ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_CFG_LYR,
    WILLOW_LYR_PIPE_0, NULL);
if (ret == EPDK_FAIL)
{
    __wrn("set image pipe failed!\n");
}
```

28.2.26. WILLOW_CMD_ROT_LYR

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp   willow 模块的句柄;
cmd   = WILLOW_CMD_ROT_LYR;
aux   = __willow_rotate_mode_t;
pbuffer = NULL;

- **RETURNS**

    EPDK_OK：成功；

    EPDK_FAIL：失败；

- **DESCRIPTION**

将图片旋转一个角度后显示，注意图片最终显示的旋转方向为 WILLOW_CMD_ROT_LYR 和 WILLOW_CMD_AUTO_ROTATE 两者共同作用的结果，如果需要设置 WILLOW_CMD_ROT_LYR，建议将 WILLOW_CMD_AUTO_ROTATE 设置为 none。

- **DEMO**

```
// 图片显示时为竖屏设置
esMODS_MIoctrl(hWillow, WILLOW_CMD_AUTO_ROTATE, PSHOW_ROTATE_NONE, NULL);
ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_ROT_LYR,
    PSHOW_ROTATE_RIGHT, NULL);
if (ret == EPDK_FAIL)
{
    __wrn("set image pipe failed!\n");
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

28.2.27. WILLOW_CMD_CFG_OUTPUT

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　willow 模块的句柄；

cmd　　= WILLOW_CMD_CFG_OUTPUT;

aux　　= 0;

pbuffer = (__willow_output_t *)

➢ **RETURNS**

EPDK_OK：成功；

EPDK_FAIL：失败；

➢ **DESCRIPTION**

在获取缩略图时，将原始数据输出到指定的 fb 中（不再获取缩略图）。

➢ **DEMO**

```
//将图片原始数据输出到指定的 fb 中
__s32 ret;
__s32 bufSize;
char filename[512];
__willow_get_thumbs_param_t thumbInfo;
__willow_output_t willowOut;
//从播放列表获取文件名
ret = GetFileFromPlaylist(filename, 0);
if (ret != EPDK_OK)
{
    return EPDK_FAIL;
}

ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_CFG_OUTPUT, 0, &willowOut);
/*将图片解码到 output 中*/
thumbInfo.filename = filename;
WaitWillowReady();
ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_GET_THUMBS, 0, &thumbInfo);
if (EPDK_FAIL == ret)
{
    __wrn("set thumbs info err!\n");
    goto EXIT_GETALBUMN;
}
ret = esMODS_MIoctrl(hWillow,WILLOW_CMD_CHECK_IMG,0,0);
if (EPDK_FAIL == ret)
{
    __wrn("image not support!\n");
    goto EXIT_GETALBUMN;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
}
ret = esMODS_MIoctrl(hWillow,WILLOW_CMD_START_SHOW,0,0);
WaitWillowReady();
if (ret == EPDK_FAIL || willowOut.is_supported = EPDK_FALSE)
{
    __wrn("image not supported!\n");
}
```

28.2.28. WILLOW_CMD_CFG_INPUT

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　willow 模块的句柄;
cmd　　= WILLOW_CMD_CFG_INPUT;
aux　　= __willow_input_src_e，可取值为:
　WILLOW_INPUT_FILENAME：输入为本地文件的文件名
　WILLOW_INPUT_BUFFER：输入为 buffer，所有文件预先读入到内存
　WILLOW_INPUT_RAWDATA：输入为解码后的数据
pbuffer　当 aux = WILLOW_INPUT_BUFFER 时,pbuffer = (__willow_buf_info_t *),其它时间取值为 0

- **RETURNS**

EPDK_OK：成功;
EPDK_FAIL：失败;

- **DESCRIPTION**

设置指定的输入方式

- **DEMO**

```
//将图片原始数据输出到指定的 fb 中
__willow_buf_info_t inputData;
/*初始化 input buffer，申请 buffer 空间*/
/*将文件读入到 input buffer 中来*/
ret = esMODS_MIoctrl(hWillow, WILLOW_CMD_CFG_INPUT,
    WILLOW_INPUT_BUFFER, & inputData);
retval = esMODS_MIoctrl(hWillow, WILLOW_CMD_CHECK_IMG, 0, 0);
if (retval == EPDK_OK)
{
    esMODS_MIoctrl(hWillow, WILLOW_CMD_START_SHOW, 0, 0);
    WaitWillowReady();
}
```

28.2.29. WILLOW_CMD_SET_PLAYLIST

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　willow 模块的句柄；

cmd　　= WILLOW_CMD_SET_PLAYLIST;

aux　　= 0

pbuffer = __willow_img_plst_info_t *

➢ **RETURNS**

EPDK_OK：成功；

EPDK_FAIL：失败；

➢ **DESCRIPTION**

拖拽功能初始化，需要设置获取文件名的回调函数，播放列表中文件总数，正在解码时显示的 framebuffer，图片不支持时显示的 framebuffer

➢ **DEMO**

```
/* 拖拽功能初始化 */

/* 根据文件索引值获取文件名的回调函数*/
static __s32 __get_file_name_by_index(__willow_show_file_param_t *param);
{
    __s32 result = EPDK_OK;
    /* 假设图片播放列表共有 4 张图片，为 1.jpg ， 2.jpg ， 3.jpg ， 4.jpg */
    if (param->img_no >= 0 && param->img_no <= 3)
    {
        eLIBs_strncpy(param->filename,"e:\\1.jpg",50);
        param->filename[4] = param->img_no + '0';
    }
    else
    {
        result = EPDK_FAIL;
    }
    return EPDK_OK;
}
/*初始化函数*/
__s32 anole_slider_init(FB *decoding_fb, FB *unsupported_fb)
{
    __willow_img_plst_info_t playlist;

    playlist.total = 4;
    playlist.get_imagename_by_index =
            esKRNL_GetCallBack((__pCBK_t)__get_file_name_by_index);
    playlist.decoding_fb[0] = decoding_fb;
```

```
    playlist.decoding_fb[1] = NULL;
    playlist.decoding_fb[2] = NULL;
    playlist.unsupported_fb = unsupported_fb;
    esMODS_MIoctrl(mod_willow, WILLOW_CMD_SET_PLAYLIST, 0, &playlist);

    return 0;
}
```

28.2.30. WILLOW_CMD_MOV_TRACK

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　willow 模块的句柄；

cmd　　= WILLOW_CMD_MOV_TRACK;

aux　　= 0

pbuffer = slider_step_para_t *，拖拽的轨迹，整个过程必须包含 SLIDER_STEP_START 和 SLIDER_STEP_FINISH 两条轨迹开始和结束的命令。

- **RETURNS**

EPDK_OK：成功；

EPDK_FAIL：失败；

- **DESCRIPTION**

设置拖拽运动的轨迹。必须有结束命令，否则中间件会挂死。

- **DEMO**

```
/* 模拟拖拽，图片由（20，30）拖拽到(50,30)  */
slider_step_para_t slider_cmd;
__u8 i;
__u32 switch_mode;

/*显示一张图片*/
……
anole_slider_init(pdecoding_fb, punsupported_fb);
switch_mode = TOUCH_SLID_H;
 esMODS_MIoctrl(mod_willow, WILLOW_CMD_SET_SWITCH_MODE, switch_mode, 0);
// 开始拖拽
slider_cmd.cmd        = SLIDER_STEP_START;
slider_cmd.offset.x = 20;
slider_cmd.offset.y = 30;
esMODS_MIoctrl(mod_willow, WILLOW_CMD_MOV_TRACK, 0, &slider_cmd);

// 模拟拖拽移动，在实际实现中(x,y)坐标由触摸消息获取
slider_cmd.cmd        = SLIDER_STEP_MOVE;
for (i = 20; i < 50; i += 5)
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
{
    slider_cmd.offset.x = i;
    esMODS_MIoctrl(mod_willow, WILLOW_CMD_MOV_TRACK, 0, &slider_cmd);
}

// 结束拖拽
slider_cmd.cmd        = SLIDER_STEP_FINISH
slider_cmd.offset.x = 50;
slider_cmd.offset.y = 30;
esMODS_MIoctrl(mod_willow, WILLOW_CMD_MOV_TRACK, 0, &slider_cmd);
```

28.2.31. WILLOW_CMD_GET_CUR_INDEX

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　willow 模块的句柄；
cmd　　= WILLOW_CMD_GET_CUR_INDEX；
aux　　= 0
pbuffer = 0

- **RETURNS**

>= 0：成功，返回拖拽结束后当前显示的图片的 index；
< 0：失败；

- **DESCRIPTION**

拖拽结束后当前显示的图片的 index。由于在拖拽时，由 willow 在控制图片播放列表前进/后退，在托拽结束后，如果应用继续以其它切换效果播放图片，需要获取 index，以确定当前播放列表中的位置。

- **DEMO**

```
/* 模拟拖拽，图片由（20，30）拖拽到(50,30)  */
__s32 index;
index = esMODS_MIoctrl(mod_willow,WILLOW_CMD_GET_CUR_INDEX, 0, 0);
__wrn("current show image index is %d\n",index);
```

28.2.32. WILLOW_CMD_SCALE_EXT

- **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

- **ARGUMENTS**

mp　　willow 模块的句柄；
cmd　　= WILLOW_CMD_SCALE_EXT；
aux　　= 放大倍数 * WILLOW_SCALER_BASE_RATIO，2300 表示放大 2.3 倍

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

pbuffer = 0

- **RETURNS**

  EPDK_OK：成功；

  EPDK_FAIL：失败；

- **DESCRIPTION**

无级放大图片，可以任意倍数放大图片

- **DEMO**

```
/* 将正在显示的图片放大 2.3 倍，参考 WILLOW_CMD_SCALE 命令  */
__s32 retval;
retval = esMODS_MIoctrl(mod_willow, WILLOW_CMD_SCALE_EXT, 2300, 0);
```

29. 内核编程指南-多线程编程

29.1. Introduction

29.1.1. Description

Melis 操作系统是由全志科技自主研发的一套精简、高效、易用的实时多任务操作系统，有着完善的内存管理、模块管理、中断管理、时钟管理、设备管理以及功耗管理。开发者可以基于 Melis 操作系统规划自己的方案，自由地开发应用程序和扩展设备驱动。

Melis 内核采用基于优先级抢占式调度机制，不支持时间片轮转，除非高优先级的线程因等待事件挂起或主动延时释放 CPU，否则低优先级的线程将永远得不到执行权。 Melis 内核支持的线程间通信机制有以下几种：

- ✓ 信号量(Semaphore)
  信号量常用于线程间的同步、资源保护，以及中断和线程间的唤醒机制等。
- ✓ 消息队列(MsgQ)
  消息队列常用于线程和线程间，或中断和线程间的消息传递等。
- ✓ 事件标志组(Flag)
  事件标志组常用于对一组共享资源的管理。
- ✓ 插槽(Socket)
  插槽常用于两个线程间共享数据流的处理。

29.1.2. Purpose

本文档主要讲述 Melis 内核的线程管理及线程间通信机制的编程接口，让开发者可以快速掌握基于 Melis 内核实现多线程编程。

29.1.3. Reference

使用本文档的开发者需要具备操作系统原理的基本知识，以及多线程编程的一些基本概念。推荐大家阅读《操作系统原理》(汤子赢注)和《嵌入式实时操作系统 uC/OS-II》(Jean J.Labrosse 注，邵贝贝等译)。

29.2. Thread

Melis 内核采用基于优先级抢占式线程调度机制。内核按优先级高低将线程依次划分为八个层级（Level0~Level7，Level0 层级的线程优先级最高，Level7 层级的线程优先级最低），最多支持 254 个线程。Level0 和 Level7 保留给 Melis 内核使用，Level1~Level6 开放给驱动、模块及应用程序使用。除 Level0 和 Level7 层级外，其它每个层级都可以支持 32 个线程。采用同一 Level 创建的线程，原则上先创建的线程的优先级比后创建的优先级要高，但是，也会出现反转的情况（即后创建的线程优先级可能会比先创建的线程的优先级要高）。 Melis 内核的线程管理支持以下几种操作接口：

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

- ✓ 创建线程
- ✓ 删除线程
- ✓ 请求删除线程
- ✓ 获取当前线程的 ID 号
- ✓ 禁止线程调度
- ✓ 允许线程调度
- ✓ 线程延时
- ✓ 撤消线程延时

29.2.1. CreateThread

- **PROTOTYPE**

```
__u8 esKRNL_TCreate(__pTHD_t thread,
                    void*    p_arg,
                    __u32    stksize,
                    __u16    priolevel);
```

- **ARGUMENTS**

Thread 待创建的线程的主函数入口；
p_arg 线程的输入参数；
Stksize 线程的栈的大小，4bytes 为单位；
Priolevel 高 8 位必须为 0，低 8 位为线程的优先级级别，对于非内核线程，可以是：KRNL_priolevel1、KRNL_priolevel2、KRNL_priolevel3、KRNL_priolevel4、KRNL_priolevel5、KRNL_priolevel6

- **RETURNS**

如果创建线程成功，则返回线程的 ID；否则，返回 0。

- **DESCRIPTION**

该函数用于创建一个线程，创建线程时除了指定线程的入口函数，还需要指定线程的栈空间大小以及线程的优先级别。

- **DEMO**

```
#define STACK_SIZE (1024*16) // 定义线程栈空间为 16K bytes
extern void thread_main(void *arg); // 线程的主函数
__u8 TestTid;
// 创建优先级别为 4 的线程
TestTid = esKRNL_TCreate(thread_main, 0, (STACK_SIZE/4), KRNL_priolevel4);
if(TestTid == 0)
{
    __wrn( “Create thread faild!\n” );
}
```

29.2.2. DeleteThread

- **PROTOTYPE**

```
__s8 esKRNL_TDel(__u8 tid);
```

- **ARGUMENTS**

tid　　　　待删除线程的 ID；
　　　　　OS_PRIO_SELF 为删除当前线程；

- **RETURNS**

EPDK_OK，删除线程成功；
EPDK_FAIL，删除线程失败；

- **DESCRIPTION**

该函数可以用来删除指定的其它线程，也可以用于删除线程自己。Melis 平台的线程不能直接返回，线程结束时，必须调用该函数删除自己，否则会出现程序异常。删除线程时，必须确保线程自己申请的资源已全部释放，否则会导致资源泄露。该函数是一个线程的终结，不会返回。

- **DEMO**

```
// 删除 TID 为 TestTid 的线程
esKRNL_TDel(TestTid);

// 删除当前线程
esKRNL_TDel(OS_PRIO_SELF);
```

29.2.3. DeleteThreadReq

- **PROTOTYPE**

```
__s8 esKRNL_TDelReq(__u16 tid_ex);
```

- **ARGUMENTS**

tid_ex　　　低 8 位为待删除线程的 ID，高 8 位为 0；
　　　　　OS_PRIO_SELF 为查询是否有其它线程请求删除当前线程；

- **RETURNS**

当 tid 为其它线程时：
　OS_NO_ERR， 向 ID 为 tid 的线程发送删除请求成功；
　OS_TASK_NOT_EXIST， ID 为 tid 的线程不存在；
　OS_TASK_DEL_ERR， 发生未知错误；
　OS_TASK_DEL_IDLE， tid 为 IDLE 线程的 ID，不允许删除；
　OS_PRIO_INVALID，tid 不合法，可能是一个不允许删除的内核线程；
当 tid 为 OS_PRIO_SELF 时：
　OS_NO_ERR， 没有其它线程请求删除当前线程；
　OS_TASK_DEL_REQ， 有其它线程正在请求删除当前线程

- **DESCRIPTION**

该函数用于请求删除其它线程，或者查询是否有其它线程请求删除当前线程。

- **DEMO**

```
void thread_demo(void *arg)
{
    while(1)
    {
        // 请求并等待 TestTid 线程退出
        while(esKRNL_TDelReq(TestTid) != OS_TASK_NOT_EXIST)
        {
        // 等待其它线程退出
```

```
            esKRNL_TimeDly(1);
        }
        // 检查是否有其它线程请求删除当前线程
        if(esKRNL_TDelReq(EXEC_prioself) == OS_TASK_DEL_REQ)
        {
            // 退出当前线程
            esKRNL_TDel(EXEC_prioself);
        }
        esKRNL_TimeDly(1);
    }
}
```

29.2.4. GetTidCur

- **PROTOTYPE**

```
__u8 esKRNL_GetTIDCur (void);
```

- **ARGUMENTS**

none;

- **RETURNS**

当前线程的 ID;

- **DESCRIPTION**

该函数用于获取当前线程的 ID，多用于调试;

- **DEMO**

```
// 获取当前线程的 ID
__u8 tmpTidCurrent = esKRNL_GetTIDCur();
printf(" Tid of current thread is:%x\n" , tmpTidCurrent);
```

29.2.5. SchedLock

- **PROTOTYPE**

```
void esKRNL_SchedLock(void);
```

- **ARGUMENTS**

none;

- **RETURNS**

none;

- **DESCRIPTION**

该函数用于禁止内核线程调度，通常用来保护多线程操作的临界区,支持多层嵌套。必须和 esKRNL_SchedUnlock 配对使用。

- **DEMO**

```
// 禁止线程调度
esKRNL_SchedLock();
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

29.2.6. TimeDly

➢ **PROTOTYPE**

```
void esKRNL_TimeDly (__u16 ticks);
```

➢ **ARGUMENTS**

ticks　　线程需要延时的 tick 数，tick 以 10ms 为单位;

➢ **RETURNS**

none;

➢ **DESCRIPTION**

该线程延时函数，该函数会将当前线程挂起 ticks*10ms；

➢ **DEMO**

```
// 延时 100ms
esKRNL_TimeDly (10);
```

29.2.7. TimeDlyResume

➢ **PROTOTYPE**

```
__u8 esKRNL_TimeDlyResume (__u16 tid_ex);
```

➢ **ARGUMENTS**

tid_ex　　线程的 ID 扩展，低 8 位为线程 ID，高 8 位为 0;

➢ **RETURNS**

恢复线程运行的结果：

OS_NO_ERR，　线程恢复运行成功；
OS_TIME_NOT_DLY，　线程为非等待状态；
OS_TASK_NOT_EXIST，　ID 为 tid 的线程不存在；
OS_PRIO_INVALID，　tid 非法；

➢ **DESCRIPTION**

该函数用于恢复处于等待状态的线程继续运行，该线程可以是调用 esKRNL_TimeDly 进入等待状态，也可以是通过带超时的等待事件而入等待状态(此时，线程等待事件的结果是超时)；

➢ **DEMO**

```
// 请求并等待 tid 线程退出
while(esKRNL_TDelReq(tid) != OS_TASK_NOT_EXIST)
{
    // tid 的线程可能处于等待状态，恢复运行
    esKRNL_TimeDlyResume(tid);
    // 等待其它线程退出
    esKRNL_TimeDly(1);
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

29.3. Semaphore

Melis 内核的 Semaphore 常用于线程之间、线程和中断之间的同步及唤醒，也可以用于多线程间共享资源的互斥访问保护。

Melis 内核的信号量管理支持以下几种编程接口：

- ✓ 创建信号量
- ✓ 删除信号量
- ✓ 等待信号量
- ✓ 释放信号量
- ✓ 获取信号量
- ✓ 查询信号量
- ✓ 设置信号量

29.3.1. SemCreate

- **PROTOTYPE**

```
__krnl_event_t* esKRNL_SemCreate(__u16 cnt);
```

- **ARGUMENTS**

cnt 信号量的初始值;

- **RETURNS**

创建信号量成功返回信号量句柄，否则返回 0;

- **DESCRIPTION**

该函数用于创建一个信号量，信号量的初始值由参数 cnt 指定。

- **DEMO**

```
// 创建一个初始值为 0 的信号量
__krnl_event_t* tmpSem0 = esKRNL_SemCreate(0);
if(tmpSem0 == (__krnl_event_t *)0)
{
    __wrn( “create semaphore failed!\n” );
}
```

29.3.2. SemDel

- **PROTOTYPE**

```
__krnl_event_t* esKRNL_SemDel( __krnl_event_t *pevent,
                               __u8 opt,
                               __u8 *err);
```

- **ARGUMENTS**

Tpevent 待删除的信号量句柄;

opt 删除信号量的可选项：

OS_DEL_NO_PEND 当没有其它线程在等待当前信号量时方可删除；

OS_DEL_ALWAYS 无条删除当前信号量；

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

err　　信号量删除操作产生的错误类型：

OS_NO_ERR　　成功删除当前信号量；

OS_ERR_DEL_ISR　　当前的删除操作是在 ISR 中执行，Melis 不允许在 ISR 中进行信号量删除操作；

OS_ERR_INVALID_OPT opt　　参数不合法；

OS_ERR_TASK_WAITING　　有其它线程正在等待当前信号量，当 opt 选择 OS_DEL_NO_PEND 时，则会报告该错误类型；

OS_ERR_EVENT_TYPE　　当前句柄指向的不是一个信号量；

OS_ERR_PEVENT_NULL　　正在试图删除一个句柄为 NULL 的信号量；

- **RETURNS**

如果删除信号量失败，返回当前信号量句柄，否则，返回 NULL；

- **DESCRIPTION**

该函数用于删除一个信号量,Melis 系统不允许在 ISR（中断服务程序）中进行信号量删除操作。

- **DEMO**

```
// 无条件删除信号量 TestSem
__u8 err;
TestSem = esKRNL_SemDel(TestSem, OS_DEL_ALWAYS, &err);
if(TestSem != (__krnl_event_t*)0)
{
    __wrn(“Try to delete TestSem failed! Error type:%d\n”, err);
}
```

29.3.3. SemPend

- **PROTOTYPE**

```
void esKRNL_SemPend(__krnl_event_t *pevent, __u16 timeout, __u8 *err);
```

- **ARGUMENTS**

pevent　　等待的信号量句柄;

timeout　　等待信号量超时的时间限制，如果该值为 0 则无限时等待，直至能成功获取一个信号量，timeout 的时间单位 tick(10ms)；

err　　信号量等待操作产生的错误类型：

OS_NO_ERR　　等待信号量成功；

OS_TIMEOUT　　等待信号量超；

OS_ERR_EVENT_TYPE　　当前的句柄不是指向一个信号量；

OS_ERR_PEND_ISR　　试图在 ISR 中执行信号量等待操作，Melis 系统不允许在 ISR 中等待信号量；

OS_ERR_PEVENT_NULL　　正在试图等待一个句柄为 NULL 的信号量

- **RETURNS**

none；

- **DESCRIPTION**

该函数用于等待一个信号量，Melis 系统不允许在 ISR 中等待信号量。当使用信号量作为资源互斥访问保护时，一定要判断信号量等待的结果，否则可能会导致保护失效。

- **DEMO**

```
// 等待信号量 TestSem，有效时间限制为 2 秒
```

```
__u8 err;
esKRNL_SemPend(TestSem, 200, &err);
if(err != OS_NO_ERR)
{
    __wrn( “Try to wait TestSem failed! Error type:%d\n” , err);
}
```

29.3.4. SemPost

➢ **PROTOTYPE**

```
__u8 esKRNL_SemPost(__krnl_event_t *pevent);
```

➢ **ARGUMENTS**

释放信号量的结果：

OS_NO_ERR	释放信号量成功；
OS_SEM_OVF	信号量记数器溢出，最多允许 65535 个;
OS_ERR_EVENT_TYPE pevent	不是指向一个信号量；
OS_ERR_PEVENT_NULL	信号量句柄为 NULL；

➢ **RETURNS**

如果创建线程成功，则返回线程的 ID；否则，返回 0。

➢ **DESCRIPTION**

该函数用于释放一个信号量，Melis 系统支持在线程或 ISR 中释放信号量。

➢ **DEMO**

```
// 释放信号量 TestSem
__u8 result = esKRNL_SemPend(TestSem);
if(err != OS_NO_ERR)
{
    __wrn( “Try to post TestSem failed! Error type:%d\n” , err);
}
```

29.3.5. SemAccept

➢ **PROTOTYPE**

```
__u16 esKRNL_SemAccept(__krnl_event_t *pevent);
```

➢ **ARGUMENTS**

pevent　　等待的信号量句柄;

➢ **RETURNS**

当前信号量的记数值，如果该值不为零，表示等待信号量成功，否则表示失败；

➢ **DESCRIPTION**

该函数用于无等待获取一个信号量，当等待信号量成功时，当前信号量的记数器会减 1。
如：信号量的记数值为 8，执行该函数后，信号量的记数值变为 7。

➢ **DEMO**

```
// 无等待的获取信号量 TestSem
__u16 result = esKRNL_SemAccept(TestSem);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
if(result == 0)
{
    __wrn("Try to accept semaphore TestSem failed!\n");
}
```

29.3.6. SemQuery

- **PROTOTYPE**

```
__u8 esKRNL_SemQuery(__krnl_event_t      *pevent,
                     OS_SEM_DATA         *p_sem_data);
```

- **ARGUMENTS**

pevent 待查询的信号量句柄;

p_sem_data 指向存放 OS_SEM_DATA 的指针，作为返回值的一部分；

- **RETURNS**

查询当前信号量的结果：

OS_NO_ERR	查询信号量成功；
OS_ERR_EVENT_TYPE	句柄 pevent 指向的不是一个信号量；
OS_ERR_EVENT_NULL	句柄 pevent 为 NULL；
OS_ERR_PDATA_NULL	p_sem_data 指针为 NULL；

查询成功后，信号量 pevent 的数据结果被拷贝到 p_sem_data 中。

- **DESCRIPTION**

该函数用于获取信号量的数据结构，通常用该接口来查询信号量记数器当前值。

- **DEMO**

```
// 查询信号量 TestSem 的当前值
OS_SEM_DATA tmpSemData;
__u8 result;
result = esKRNL_SemQuery(TestSem, &tmpSemData);
if(result == OS_NO_ERR)
{
__inf("Events count of TestSem is:%d\n", tmpSemData. OSCnt);
}
```

29.3.7. SemSet

- **PROTOTYPE**

```
void esKRNL_SemSet( __krnl_event_t  *pevent,
                    __u16           cnt,
                    __u8            *err);
```

- **ARGUMENTS**

pevent 待设置的信号量的句柄;

cnt 设置给信号量记数器的值；

err 设置信号量时的错误返回值：

OS_NO_ERR	设置信号量的记数器成功；
OS_ERR_EVENT_TYPE	句柄 pevent 指向的不是一个信号量；

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

OS_ERR_PEVENT_NULL　　　信号量句柄 pevent 为 NULL；
OS_ERR_TASK_WAITING　　有其它线程正在等待当前信号量；

➢ **RETURNS**

none。

➢ **DESCRIPTION**

该函数用于设置信号量的记数器的值，不推荐使用该函数。

➢ **DEMO**

```
__u8 err;
OS_SEM_DATA tmpSemData;
// 设置信号量 TestSem 的值
esKRNL_SemSet(TestSem, 100, &err);
if(err != OS_NO_ERR)
{
    __wrn( “Set envent count of TestSem failed, Error Type:%d!\n” , err);
}
// 测试信号量的记数器
result = esKRNL_SemQuery(TestSem, &tmpSemData);
if(result == OS_NO_ERR)
{
    __inf( “Events count of TestSem is:%d\n” , tmpSemData. OSCnt);
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

29.4. MsgQ

MsgQ 常用于多线程间交换消息。
Melis 内核的 MsgQ 支持以下几种编程接口：

- ✓ 创建消息队列
- ✓ 删除消息队列
- ✓ 等待消息
- ✓ 发送消息
- ✓ 获取消息
- ✓ 查询消息队列
- ✓ 清空消息队列

29.4.1. QCreate

➢ **PROTOTYPE**

```
__krnl_event_t* esKRNL_QCreate(__u16 size);
```

➢ **ARGUMENTS**

size　　消息队列的深度;

➢ **RETURNS**

创建消息队列成功返回消息队列句柄，否则返回 0;

➢ **DESCRIPTION**

该函数用于创建一个消息队列，消息队列的深度由参数 size 指定。

➢ **DEMO**

```
// 创建一个深度为 8 的消息队列
__krnl_event_t* tmpMsgQ = esKRNL_QCreate(8);
if(tmpMsgQ== (__krnl_event_t *)0)
{
    __wrn("create message queue failed!\n");
}
```

29.4.2. QDelete

➢ **PROTOTYPE**

```
__krnl_event_t* esKRNL_QDel( __krnl_event_t  *pevent,
                             __u8            opt,
                             __u8            *err);
```

➢ **ARGUMENTS**

pevent　　待删除的消息队列句柄;
opt　　删除消息队列时的选择项:
　　OS_DEL_NO_PEND　　没有其它线程正在等待该消息队列方允许删除;
　　OS_DEL_ALWAYS　　无条件删除该消息队列;
err　　删除消息队列操作产生的错误类型:

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

OS_NO_ERR	删除消息队列成功，未见异常；
OS_ERR_DEL_ISR	当前删除操作位于 ISR 程序中，Melis 内核不允在 ISR 中执行消息除列的删除操作；
OS_ERR_INVALID_OPT opt	参数不合法；
OS_ERR_TASK_WAITING	当 opt 为 OS_DEL_NO_PEND 时，如果有其它线程正在等待当前消息队列，则删除操作会报错；
OS_ERR_EVENT_TYPE	pevent 指向的不是一个消息队列；
OS_ERR_PEVENT_NULL	pevent 的值为 NULL；

- **RETURNS**

如果删除消息队列成功，返回 NULL，否则返回 pevent；

- **DESCRIPTION**

该函数用于删除一个指定的消息队列。

- **DEMO**

```
// 删除消息队列 TestMsgQ
__krnl_event_t *tmpMsgQ;
__u8 err;
tmpMsgQ = esKRNL_QDel(TestMsgQ, OS_DEL_ALWAYS, &err);
if(tmpMsgQ != (__krnl_event_t *)0)
{
    __wrn(“delete message queue TestMsgQ failed! Error type:%d\n”, err);
}
```

29.4.3. QPend

- **PROTOTYPE**

```
void* esKRNL_QPend( __krnl_event_t      *pevent,
                    __u16               timeout,
                    __u8                *err);
```

- **ARGUMENTS**

pevent	等待的消息队列的句柄；
timeout	等待消息队列的有效时限（以 tick 为单位），如果 timeout 为 0 则表示无限时地等待直至获得消息为止；
err	等待消息队列操作产生的错误类型：

OS_NO_ERR	*等待消息队列成功，未见异常；*
OS_TIMEOUT	*等待消息队列超时；*
OS_ERR_EVENT_TYPE	*pevent 句柄指向的不是一个消息队列；*
OS_ERR_PEND_ISR	*试图在 ISR 中执行消息队列等待操作，Melis 系统不允许在 ISR 中等待信号量；*
OS_ERR_PEVENT_NULL	*pevent 句柄为 NULL；*

- **RETURNS**

从消息队列中获取的消息，通常是一个指向用户自定义数据结构的指针；如果等待消息队列失败则返回 NULL。

- **DESCRIPTION**

该函数用于从消息队列中获取一个消息。

- **DEMO**

```
// 从消息队列 TestMsgQ 中获取一个消息，时限为 2 秒
__u8      err;
void      *tmpMsg;
tmpMsg = esKRNL_QPend(tmpMsg, 200, &err);
if(err== OS_NO_ERR)
{
        __inf("Get a message from TestMsgQ:%x\n!\n", tmpMsg);
}
Else
{
        __wrn("Try to get message from TestMsgQ failed! Error type:%d\n", err);
}
```

29.4.4. QPost

- **PROTOTYPE**

```
__u8 esKRNL_QPost(__krnl_event_t *pevent, void *msg);
```

- **ARGUMENTS**

pevent　　消息队列的句柄;
msg　　释放到消息队列 pevent 中的消息;

- **RETURNS**

释放消息的结果:

OS_NO_ERR　　*释放消息成功，未见异常;*
KRNL_Q_FULL　　*消息队列已满(消息队列的深度在创建时指定);*
OS_ERR_EVENT_TYPE　　*pevent 句柄指向的不是一个消息队列;*
OS_ERR_PEVENT_NULL　　*pevent 句柄的值为 NLL;*

- **DESCRIPTION**

该函数用于向消息队列 pevent 中释放一个消息，该消息可以是一个指向某数据结构的指针，也可以是一个值，由消息释放方和消息获取方约定使用;

- **DEMO**

```
// 向消息队列 TestMsgQ 中释放一个消息 msg
__u8      result;
result = esKRNL_QPost(TestMsgQ, msg);
if(result!= OS_NO_ERR)
{
        __wrn("post message to TestMsgQ failed! Error type:%d\n", result);
}
```

29.4.5. QAccept

- **PROTOTYPE**

```
void *esKRNL_QAccept(__krnl_event_t *pevent, __u8 *err);
```

➢ **ARGUMENTS**

pevent　　　等待的消息队列的句柄;

err　　　　等待消息时产生的错误的类型;

OS_NO_ERR　　　　*等待消息队列成功，未见异常;*

OS_ERR_EVENT_TYPE　　*pevent 句柄指向的不是一个消息队列;*

OS_ERR_PEVENT_NULL　　*pevent 句柄为 NULL;*

KRNL_Q_EMPTY　　　　*消息队列里当前状态为空，没有任何消息;*

➢ **RETURNS**

当 err 的值为 OS_NO_ERR 时，返回获取到的消息;

否则，返回 NULL;

➢ **DESCRIPTION**

该函数为无等待地从消息队列中获取一个消息，该操作可以由线程执行，也可以在 ISR 中执行。

➢ **DEMO**

```
// 无等待地从消息队列 TestMsgQ 中获取一个消息
void        *tmpMsg;
__u8        err;
tmpMsg = esKRNL_QAccept(TestMsgQ, &err);
if(err== OS_NO_ERR)
{
        __inf("get a message from TestMsgQ, value is:%x\n", tmpMsg);
}
```

29.4.6. QQuery

➢ **PROTOTYPE**

```
__u8 esKRNL_QQuery( __krnl_event_t   *pevent,
                    __krnl_q_data_t  *p_q_data);
```

➢ **ARGUMENTS**

pevent　　　待查询的消息队列的句柄;

p_q_data　　指向存放消息队列数据结构的指针;

➢ **RETURNS**

查询消息队列的结果:

OS_NO_ERR　　　　*查询消息队列数据结构成功，数据结果存放于 p_q_data;*

OS_ERR_EVENT_TYPE　　*pevent 句柄指向的不是一个消息队列;*

OS_ERR_PEVENT_NULL　　*pevent 句柄为 NULL;*

OS_ERR_PDATA_NULL　　*p_a_data 为指针的值为 NULL;*

➢ **DESCRIPTION**

该函数用于获取消息队列的数据，一般用来查询消息队列当前的消息数。

➢ **DEMO**

```
// 查询消息队列 TestMsgQ
__u8                result;
__krnl_q_data_t     tmpMsgQData;
result = esKRNL_QQuery(TestMsgQ, &tmpMsgQData);
if(result == OS_NO_ERR)
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
{
        __inf("Count of event in TestMsgQ is:%d\n", tmpMsgQData. OSNMsgs);
}
```

29.4.7. QFlush

➢ **PROTOTYPE**

```
__u8 esKRNL_QFlush(__krnl_event_t *pevent);
```

➢ **ARGUMENTS**

pevent　　待清空的消息队列的句柄；

➢ **RETURNS**

清空消息队列操作的结果：

OS_NO_ERR　　*清空消息队列的操作成功；*
OS_ERR_EVENT_TYPE　　*pevent 句柄指向的不是一个消息队列；*
OS_ERR_PEVENT_NULL　　*pevent 句柄为 NULL；*

➢ **DESCRIPTION**

该函数用于清空一个消息队列内的所有消息。

➢ **DEMO**

```
// 清空消息队列 TestMsgQ
__u8        result;
result = esKRNL_QFlush(TestMsgQ);
if(result!= OS_NO_ERR)
{
        __wrn("Flush message queue TestMsgQ failed! Error type:%d\n", result);
}
```

29.5. Flag

Melis 内核的事件标志组常用于对一组资源的管理。当多个用户对一组资源集合中的某些资源同时都有需求的时候，事件标志组将会是对资源进行管理的最好的选择。要么同时抢占到所有需要的资源，要么一个都不占用，从而避免因只抢夺到部分资源而导致彼此互锁的情况。

Melis 内核的事件标志组支持以下几种编程接口：

- ✓ 创建事件标志组
- ✓ 删除事件标志组
- ✓ 等待事件
- ✓ 释放事件
- ✓ 获取事件
- ✓ 查询事件

29.5.1. FlagCreate

➢ **PROTOTYPE**

```
__krnl_flag_grp_t* esKRNL_FlagCreate( __krnl_flags_t      flags,
                                      __u8                *err);
```

➢ **ARGUMENTS**

flags　　事件标志组的初始值；

err　　创建事件标志组的错误类型：

OS_NO_ERR　　*创建事件标志组成功；*

OS_ERR_CREATE_ISR　　*在 ISR 中创建事件标志组，Melis 内核不允许在 ISR 中创建事件标志组；*

KRNL_FLAG_GRP_DEPLETED　　*创建事件标志组时分配内存失败；*

➢ **RETURNS**

如果创建事件标志组成功则返回事件标志组的句柄；

否则，返回 NULL；

➢ **DESCRIPTION**

该函数用于创建一个事件标志组，事件标志组的初始值有参数 flags 给出。

➢ **DEMO**

```
// 创建一个事件标志组，初始值为 0x00000ff0
__krnl_flag_grp_t         *TestFlag;
__u8                      err;
TestFlag = esKRNL_FlagCreate((__krnl_flags_t)0x00000ff0, &err);
if(err != OS_NO_ERR)
{
    __wrn(“create flag failed! Error type:%d\n”, err);
}
```

29.5.2. FlagDel

➢ **PROTOTYPE**

```
__krnl_flag_grp_t* esKRNL_FlagDel( __krnl_flag_grp_t      *pgrp,
                                   __u8                   opt,
                                   __u8                   *err);
```

➢ **ARGUMENTS**

pgrp　　待删除的事件标志组的句柄；

opt　　删除事件标志组的选择项；

OS_DEL_NO_PEND　　*没有其它线程正在等待该事件标志组方允许删除；*

OS_DEL_ALWAYS　　*无条件删除该事件标志组；*

err　　删除事件标志组的错误类型；

OS_NO_ERR　　*删除事件标志组成功，未见异常；*

OS_ERR_DEL_ISR　　*当前删除操作位于 ISR 程序中，Melis 内核不允许在 ISR 中执行事件标志组的删除操作；*

OS_ERR_INVALID_OPT　　*opt 参数不合法；*

OS_ERR_TASK_WAITING　　*当 opt 为 OS_DEL_NO_PEND 时，如果有其它线程正在等待当前事件标志组，则删除操作会报错；*

OS_ERR_EVENT_TYPE　　*pgrp 指向的不是一个消息队列；*

OS_FLAG_INVALID_PGRP　　*pgrp 的值为 NULL；*

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

➢ **RETURNS**

如果删除事件标志组成功，则返回 NULL；

否则，返回 pgrp；

➢ **DESCRIPTION**

该函数用于删除一个事件标志组。

➢ **DEMO**

```
// 删除事件标志组 TestFlag
__krnl_flag_grp_t     *tmpFlag;
__u8                  err;
tmpFlag = esKRNL_FlagDel(TestFlag,OS_DEL_ALWAYS, &err);
if(err != OS_NO_ERR)
{
    __wrn( "Try to delete TestFlag failed! Error type:%d\n" , err);
}
```

29.5.3. FlagPend

➢ **PROTOTYPE**

```
__krnl_flags_t esKRNL_FlagPend( __krnl_flag_grp_t    *pgrp,
                                __krnl_flags_t       flags,
                                __u32                wtype_tout,
                                __u8                 *err)
```

➢ **ARGUMENTS**

pgrp　　　　等待的事件标志组的句柄；

flags　　　　等待的事件类型，是一些事件的集合；

wtype_tout　　一个组合参数：

高 16Bit，等待事件的方式：

OS_FLAG_WAIT_SET_ALL　　所有要等待的事件都发生才算等待成功；

OS_FLAG_WAIT_SET_ANY　　任一个要等待的事件发生就算等待成功；

低 16Bit，等待事件的时效；

err　　　　等待事件标志组的错误类型；

OS_NO_ERR　　　　等待事件标志组成功，未见异常；

OS_ERR_PEND_ISR　　当前等待操作位于 ISR 程序中，Melis 内核不允在 ISR 中执行事件标志组的等待操作；

OS_TIMEOUT　　　　等事件事件超时；

OS_ERR_EVENT_TYPE　pgrp 指向的不是一个消息队列；

OS_FLAG_ERR_WAIT_TYPE　等待事件的方式参数不合法；

OS_FLAG_INVALID_PGRP　　pgrp 的值为 NULL；

➢ **RETURNS**

如果等待事件成功，则返回得到的事件类型；

否则，返回 0；

➢ **DESCRIPTION**

该函数用于等待一组事件，事件类型由 flags 指定。

➢ **DEMO**

```
// 等待事件标志组 TestFlag，等待的事件类型为 0x00000330，
// 不限时，所有事件都必须满足
__u8                err;
__krnl_flags_t      tmpFlag;
tmpFlag = esKRNL_FlagPend( TestFlag,
                           (__krnl_flags_t) 0x00000330,
                           (OS_FLAG_WAIT_SET_ALL << 16)|(0<<0);
                           &err);
if(err != OS_NO_ERR)
{
    __wrn(“Wait flag from TestFlag failed! Error type:%d\n”, err);
}
```

29.5.4. FlagAccept

- **PROTOTYPE**

```
__krnl_flags_t esKRNL_FlagAccept( __krnl_flag_grp_t     *pgrp,
                                  __krnl_flags_t        flags,
                                  __u8                  wait_type,
                                  __u8                  *err);
```

- **ARGUMENTS**

pgrp　　等待的事件标志组的句柄；
flags　　等待的事件类型；
wait_type　　等待事件的方式：
OS_FLAG_WAIT_SET_ALL　　所有要等待的事件都发生才算等待成功；
OS_FLAG_WAIT_SET_ANY　　任一个要等待的事件发生就算等待成功；
err　　等待事件产生的错误类型：
OS_NO_ERR　　等待事件标志组成功，未见异常；
OS_ERR_EVENT_TYPE　　pgrp 指向的不是一个消息队列；
OS_FLAG_ERR_WAIT_TYPE　　等待事件的方式参数不合法；
OS_FLAG_INVALID_PGRP　　pgrp 的值为 NULL；
OS_FLAG_ERR_NOT_RDY　　等待的事件类型不满足；

- **RETURNS**

如果等待事件成功，返回得到的事件类型；
否则，返回 0；
否则，返回 NULL；

- **DESCRIPTION**

该函数用于无等待地从事件标志组中获取事件类型，支持在线程及 ISR 中使用。

- **DEMO**

```
// 等待事件标志组 TestFlag，等待的事件类型为 0x00000550，
// 任一事件满足及可
__u8                err;
__krnl_flags_t      tmpFlag;
tmpFlag = esKRNL_FlagAccept( TestFlag,
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
                                    （__krnl_flags_t）0x00000550,
                                    OS_FLAG_WAIT_SET_ANY,
                                    &err);
    if(err != OS_NO_ERR)
    {
        __wrn("Accept flag from TestFlag failed! Error type:%d\n", err);
    }
    else
    {
        __inf("Accept TestFlag successed! Event type:%x\n", tmpFlag);
    }
```

29.5.5. FlagPost

➢ **PROTOTYPE**

```
    __krnl_flags_t esKRNL_FlagPost( __krnl_flag_grp_t    *pgrp,
                                    __krnl_flags_t       flags,
                                    __u8                 opt,
                                    __u8                 *err);
```

➢ **ARGUMENTS**

pgrp　　　事件标志组的句柄；

flags　　　要设置的事件类型；

opt　　　设置事件的操作类型：

　　OS_FLAG_SET　　*设置各事件有效；*

　　OS_FLAG_CLR　　*设置各事件无效；*

err　　　设置标志组操作的错误类型：

　　OS_NO_ERR　　*删除事件标志组成功，未见异常；*

　　OS_ERR_INVALID_OPT　　*opt 参数不合法；*

　　OS_ERR_EVENT_TYPE　　*pgrp 指向的不是一个消息队列；*

　　OS_FLAG_INVALID_PGRP　　*pgrp 的值为 NULL；*

➢ **RETURNS**

如果设置事件标志组成功，返回事件标志组当前有效的事件类型；

否则，返回 0；

➢ **DESCRIPTION**

该函数用于设置事件标志组中的事件类型，事件类型参数 flags 给出，设置方式(设置事件有效或是无效)由参数 opt 给出。

➢ **DEMO**

```
    // 设置事件标志组 TestFlag，事件类型为 0x00000030，设置事件有效
    __u8                    err;
    __krnl_flags_t          tmpFlag;
    tmpFlag = esKRNL_FlagPost( TestFlag,
                               (__krnl_flags_t)0x00000030,
                               OS_FLAG_SET,
                               &err);
```

```
if(err != OS_NO_ERR)
{
    __wrn("Post TestFlag failed! Error type:%d\n", err);
}
else
{
    __inf("Post TestFlag Successed! Event type:%x\n", tmpFlag);
}
```

29.5.6. FlagQuery

- **PROTOTYPE**

```
__krnl_flags_t esKRNL_FlagQuery(__krnl_flag_grp_t *pgrp, __u8 *err);
```

- **ARGUMENTS**

pgrp　　事件标志组的句柄;

err　　创建事件标志组的错误类型:

OS_NO_ERR	*查询事件标志组成功;*
OS_ERR_EVENT_TYPE	*pgrp 指向的不是一个消息队列;*
OS_FLAG_INVALID_PGRP	*pgrp 的值为 NULL;*

- **RETURNS**

如果查询事件标志组成功，则返回事件标志组中当前的事件类型;

否则，返回 0;

- **DESCRIPTION**

该函数用于查询事件标志组当前各事件的状态。

- **DEMO**

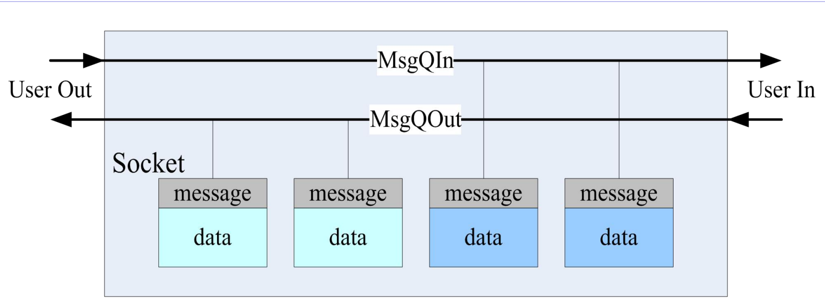
```
// 查询事件标志组 TestFlag 当前的事件状态
__u8              err;
__krnl_flags_t    tmpFlag;
tmpFlag = esKRNL_FlagQuery(TestFlag, &err);
if(err != OS_NO_ERR)
{
    __wrn("Query TestFlag failed! Error type:%d\n", err);
}
else
{
    __inf("Event type of TestFlag is:%x\n", tmpFlag);
}
```

29.6. Socket

Socket 也是 Melis 内核线程间通信的一种机制。Melis 内核上的 Socket 常用于多个线程对共享数据流进行处理。Socket 内部设计思想如下图所示(该 Socket 的缓冲深度为 4)，其由两个 MsgQ 组合实现，消息和数据以帧为单位进行管理（每个帧带有自己的消息缓冲区和数据缓冲区）。在 Socket 创建时，所有的缓冲区被发

送到 MsgQIn 中，用户 UserIn 通过 MsgQIn 有序地获取从用户 UserOut 发送来的数据，UserIn 处理完数据以后，再将缓冲区或数据通过 MsgQOut 有序地传送给 UserOut，从而实现对共享数据的交换和处理。message 用来传递用户自定义的消息，一般是对数据缓冲区内的数据的描述信息，也可以是其它消息，由 UserIn 和 UserOut 自行约定。

Socket内部组织结构图



Melis 内核的 Socket 通信机制支持以下几种编程接口：

- ✓ 创建插槽
- ✓ 删除插槽
- ✓ 等待插槽帧
- ✓ 发送插槽帧
- ✓ 获取插槽帧
- ✓ 清空插槽帧

29.6.1. SktCreate

➢ **PROTOTYPE**

```
__hdle esKRNL_SktCreate( __u32       depth,
                         __u32       dbuf_attr,
                         __u32       mbuf_attr);
```

➢ **ARGUMENTS**

depth　　　待创建的插槽的深度；

dbuf_attr　　插槽的数据缓冲区的描述参数：

低 24bit，描述数据缓冲区的块大小，以字节为单位；

高 8bit，　描述数据缓冲区是否为物理空间连续：

KRNL_SKT_BUF_PHY_UNSEQ　　　不需要物理空间连续；

KRNL_SKT_BUF_PHY_SEQ　　　　必须物理空间边续；

mbuf_attr　　插槽的消息缓冲区的描述参数：

低 24bit，描述数据消息缓冲区的大小，以字节为单位；

高 8bit，　描述消息缓冲区是否为物理空间连续：

KRNL_SKT_BUF_PHY_UNSEQ　　　不需要物理空间连续；

KRNL_SKT_BUF_PHY_SEQ　　　　必须物理空间边续；

➢ **RETURNS**

如果创建插槽成功，则返回插槽的句柄；

否则，返回 NULL；

➢ **DESCRIPTION**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

该函数用于创建一个插槽，插槽的数据缓冲区和消息缓冲区由内核负责根据给定的属性描述参数进行分配。

➢ **DEMO**

```
// 创建一个插槽，深度为 4，数据缓冲区为 8K，要求物理空间连续，消息缓冲
// 区是一个类型为__test_skt_t 的数据结构，不要求物理空间连续
__hdle   TestSkt;
TestSkt = esKRNL_SktCreate( 4,
                            KRNL_SKT_BUF_PHY_SEQ | 0x2000,
                            KRNL_SKT_BUF_PHY_UNSEQ| sizeof(__test_skt_t));
if(TestSkt == (__hdle)0)
{
    __wrn( "create socket failed!\n" );
}
```

29.6.2. SktDel

➢ **PROTOTYPE**

```
__s32  esKRNL_SktDel(__hdle hSkt, __u8 opt );
```

➢ **ARGUMENTS**

hSkt　　待删除的插槽的句柄；

opt　　未定义：

➢ **RETURNS**

如果删除插槽成功，则返回 EPDK_OK；

否则，返回 EPDK_FAIL；

➢ **DESCRIPTION**

该函数用于删除一个插槽，无论插槽上的帧处于何种状态，删除插槽时会释放所有的帧缓冲区。

➢ **DEMO**

```
// 删除插槽 TestSkt
__s32        result;
result = esKRNL_SktDel(TestSkt, 0);
if(result != EPDK_OK)
{
    __wrn( "delete socket TestSkt failed!\n" );
}
```

29.6.3. SktPend

➢ **PROTOTYPE**

```
__krnl_sktfrm_t *esKRNL_SktPend( __hdle       hSkt,
                                 __u8         user,
                                 __u32        timeout);
```

➢ **ARGUMENTS**

hSkt　　等待的插槽的句柄；

user　　　等待的方式：

　　　　　KRNL_SKT_USR_IN　　*以输入用户方式等待；*

　　　　　KRNL_SKT_USR_OUT　 *以输出用户方式等待；*

timeout　 等待插槽的时限，以 tick 为单位；

- **RETURNS**

如果等待插槽消息成功，则返回等到的插槽消息帧；

否则，返回 NULL；

- **DESCRIPTION**

该函数用于从插槽 hSkt 中等待一个消息帧。

- **DEMO**

```
// 从插槽 TestSkt 上以输出用户方式等待一个消息帧，以 2 秒为时限
__krnl_sktfrm_t       tmpSktFrm;
tmpSktFrm = esKRNL_SktPend(TestSkt, KRNL_SKT_USR_OUT, 200);
if(tmpSktFrm == (__krnl_sktfrm_t *)0)
{
    __wrn("Pend TestSkt failed!\n");
}
```

29.6.4. SktPost

- **PROTOTYPE**

```
__s32 esKRNL_SktPost( __hdle              hSkt,
                      __u8                user,
                      __krnl_sktfrm_t     *frm);
```

- **ARGUMENTS**

hSkt　　　待释放消息帧的插槽的句柄；

user　　　用户模式：

　　　　　KRNL_SKT_USR_IN　　*以输入用户方式释放消息帧；*

　　　　　KRNL_SKT_USR_OUT　 *以输出用户方式释放消息帧；*

frm　　　 等释放的消息帧；

- **RETURNS**

释放插槽消息帧的结果：

OS_NO_ERR	*释放消息成功，未见异常；*
KRNL_Q_FULL	*消息队列已满(消息队列的深度在创建时指定)；*
OS_ERR_EVENT_TYPE	*pevent 句柄指向的不是一个消息队列；*
OS_ERR_PEVENT_NULL	*pevent 句柄的值为 NLL；*

- **DESCRIPTION**

该函数用于向插槽 hSkt 中释放一个消息帧。

- **DEMO**

```
// 向插槽 TestSkt 中以输入用户方式释放消息帧 pFrm
__s32         result;
result = esKRNL_SktPost(TestSkt, KRNL_SKT_USR_IN, pFrm);
if(result != OS_NO_ERR)
{
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
        __wrn( "Post frame to TestSkt failed! Error type:%d\n", result);
    }
```

29.6.5. SktAccept

➢ **PROTOTYPE**

```
    __krnl_sktfrm_t *esKRNL_SktAccept(__hdle  pSkt, __u8 user);
```

➢ **ARGUMENTS**

hSkt　　　等待消息帧插槽的句柄；
user　　　用户模式：
　　　　　KRNL_SKT_USR_IN　　*以输入用户方式等待消息帧；*
　　　　　KRNL_SKT_USR_OUT　　*以输出用户方式等待消息帧；*

➢ **RETURNS**

如果等待消息帧成功，则返回消息帧指针；
否则，返回 NULL；

➢ **DESCRIPTION**

该函数用于无等待地从插槽 pSkt 中获取一个消息帧，支持在线程及 ISR 程序中使用；

➢ **DEMO**

```
    // 无等待地从 TestSkt 中以输出用户模式获取消息帧
    __krnl_sktfrm_t       *tmpSktFrm;
    tmpSktFrm = esKRNL_SktAccept(TestSkt, KRNL_SKT_USR_OUT);
    if(tmpSktFrm == (__krnl_sktfrm_t *)0)
    {
        __wrn( "Try to get frame from TestSkt failed!);
    }
```

29.6.6. SktFlush

➢ **PROTOTYPE**

```
    __s32 esKRNL_SktFlush (__hdle hSkt, __u8 user);
```

➢ **ARGUMENTS**

hSkt　　　需要清空的插槽的句柄；
user　　　用户模式：
　　　　　KRNL_SKT_USR_IN　　*以输入用户方式清空插槽；*
　　　　　KRNL_SKT_USR_OUT　　*以输出用户方式清空插槽；*

➢ **RETURNS**

清空插槽操作的结果：
　　OS_NO_ERR　　　　*清空插槽的操作成功；*

➢ **DESCRIPTION**

该函数用于用于清空插槽中的消息帧；

➢ **DEMO**

```
    // 以输出用户模式清空插槽 TestSkt
    __s32          result;
```

```
result = esKRNL_SktFlush(TestSkt, KRNL_SKT_USR_OUT);
if(result != OS_NO_ERR)
{
    __wrn("Try to flush TestSkt failed!);
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

30. 内核编指南-定时器

30.1. Introduction

30.1.1. Description

Melis 操作系统是由全志科技自主研发的一套精简、高效、易用的实时多任务操作系统，有着完善的内存管理、模块管理、中断管理、时钟管理、设备管理以及功耗管理。开发者可以基于 Melis 操作系统规划自己的方案，自由地开发应用程序和扩展设备驱动。
Melis 内核的定时器支持软件定时器（SoftTimer）和硬件定时器(HardwareTimer)两种类型。软件定时器由定时器 Server 线程来处理所有的软件定时器，精度较低，数量没有限制；硬件定时器则是采用硬件定时器单元来实现，精度较高，但是定时器个数受硬件电路单元限制。一般来说，只允许用户使用软件定时器，硬件定时器由操作系统固定分配给一些有特定需求的设备驱动来使用。

30.1.2. Purpose

本文档主要讲述 Melis 内核和时间及定时器相关的编接口,让开发者能快速掌握 Melis 系统上的定时器操作，并基于定时器来开发自己的程序。

30.1.3. Reference

读者可以先了解一些定时器的基本功能和操作，明白定时器的工作原理，以加深对定时器编程接口的理解。

30.1.4. Contact Info

如果您发现文档中的描述和实际应用有出入，或是文档描述不清晰让您有疑问，请随时联系我们，我们将最短时间内做出答复。同时，如果您有任建议或批评，也希望您不吝赐教。
请通过以下联系方式联系我们：
Homepage: http://www.allwinnertech.com
E-mail: kevin@allwinnertech.com

30.2. Time

Melis 操作系统上的时间处理包含两个方面：内核运行的时间和 RTC 时间。内核运行的时间以内核调度器的 tick(10ms)为单位，有于记录系统启动后运行的时间；RTC 时间是基于硬件 RTC 模块得到的时间。
Melis 内核关于时间的操作支持以下几个接口：

- ✓ 获取系统运行时间
- ✓ 设置系统运行时间

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

- ✓ 获取 RTC 时间
- ✓ 设置 RTC 时间
- ✓ 获取日期
- ✓ 设置日期

30.2.1. TimeGet

- **PROTOTYPE**

```
__u32 esKRNL_TimeGet (void);
```

- **ARGUMENTS**

none;

- **RETURNS**

系统运行的时间，以 tick 为单位（Melis 内核的 tick 为 10ms）；

- **DESCRIPTION**

该函数用于获取系统运行的时间(从系统启动到现在的时间)，时间值以内核的 tick 为单位，melis 内核的 tick 为 10ms。例如，如果返回值为 5000，则表示系统运行时间为 5000*10ms；

- **DEMO**

```
// 获取内核启动后的时间
__u32 time = esKRNL_TimeGet();
__inf(“Time from kernel boot is %d seconds.\n”,time/100);
```

30.2.2. TimeSet

- **PROTOTYPE**

```
void esKRNL_TimeSet (__u32 ticks);
```

- **ARGUMENTS**

ticks　需要设置的时间值，以 tick 为单位;

- **RETURNS**

none;

- **DESCRIPTION**

该函数用于设置系统运行的时间，以内核的 tick 为单位;

- **DEMO**

```
// 设置系统运行的时间
__inf(“set time of system run to %d seconds.\n”,time/100);
esKRNL_TimeSet(time);
```

30.2.3. GetRtcTime

- **PROTOTYPE**

```
__s32 esTIME_GetTime(__time_t *time);
```

- **ARGUMENTS**

time　指向存放 RTC 时间结构体的指针;

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

- **RETURNS**

获取 RTC 时间的结果：

EPDK_OK，获取 RTC 时间成功，时间存放于 time 所指向的单元；

EPDK_FAIL，获取 RTC 时间失败；

- **DESCRIPTION**

该函数用于获取当前的 RTC 时间；

- **DEMO**

```
// 获取当前的 RTC 时间
__time_t     tmpTime;
__s32        result;
result = esTIME_GetTime(&tmpTime);
if(result == EPDK_OK)
{
    __inf("Current rtc time:%d:%d:%d!\n", tmpTime.hour,
                    tmpTime.minute,tmpTime.second);
}
```

30.2.4. SetRtcTime

- **PROTOTYPE**

```
__s32 esTIME_SetTime(__time_t *time);
```

- **ARGUMENTS**

time　　指向需要设置的 RTC 时间结构的指针；

- **RETURNS**

设置 RTC 时间的结果：

EPDK_OK，设置 RTC 时间成功；

EPDK_FAIL，设置 RTC 时间失败；

- **DESCRIPTION**

该函数用于设置 RTC 时间，包含时、分、秒；

- **DEMO**

```
// 设置系统 RTC 时间
__time_t     tmpTime;
__s32        result;
__inf("set rtc time to 19:38:57.\n");
tmpTime.hour    = 19;
tmpTime.minute = 38;
tmpTime.second = 57
result = esTIME_SetTime(&tmpTime);
if(result != EPDK_OK)
{
    __wrn("Try to set rtc time failed!\n");
}
```

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

30.2.5. GetRtcDate

- **PROTOTYPE**

```
__s32 esTIME_GetDate(__date_t *date);
```

- **ARGUMENTS**

date　　指向存放 RTC 日期的指针;

- **RETURNS**

获取 RTC 日期的结果:

EPDK_OK，获取 RTC 日期成功;

EPDK_FAIL，获取 RTC 日期失败;

- **DESCRIPTION**

该函数用于获取系统的 RTC 日期;

- **DEMO**

```
// 获取系统的 RTC 日期
__date_t     tmpDate;
__s32        result;
result = esTIME_GetDate(&tmpDate);
if(result == EPDK_OK)
{
    __inf("Current date is:%d-%d-%d\n", tmpDate.year,
                      tmpDate.month, tmpDate.day);
}
```

30.2.6. SetRtcDate

- **PROTOTYPE**

```
__s32 esTIME_SetDate(__date_t *date);
```

- **ARGUMENTS**

date　　指向要设置的 RTC 日期的指针;

- **RETURNS**

设置 RTC 日期的结果:

EPDK_OK，设置 RTC 日期成功;

EPDK_FAIL，设置 RTC 日期失败;

- **DESCRIPTION**

该函数用于设置 RTC 日期，日期格式为 xxxx-xx-xx;

- **DEMO**

```
// 设置 RTC 日期
__date_t     tmpDate;
__s32        result;
__inf("Set rtc date to: 2011-03-10\n");
tmpDate.year  = 2011;
tmpDate.month = 3;
tmpDate.day   = 10;
```

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

```
result = esTIME_SetDate(&tmpDate);
if(result != EPDK_OK)
{
    __wrn( “Try to set rtc date failed!\n” );
}
```

30.2.7. SoftTimer

Melis 平台上的 SoftTimer 是通过软件定时器的 Server 线程来实现的。SoftTimer 的定时单位为毫秒(ms)，时间精度为 5ms，即：1~5 毫秒的实际定时为 5ms，6~10 毫秒的实际定时为 10ms。
SoftTimer 分为两种类型：

- ✓ OS_TMR_OPT_PRIO_LOW
- ✓ OS_TMR_OPT_PRIO_HIGH

OS_TMR_OPT_PRIO_LOW 类型的软件定时器的 Server 线程的优先级较低，适时性较差，只能在线程中进行创建、删除、启动和停止等操作，不支持在中断 ISR 中进行相关操作，一般用于应用程序或对定时的适时性要求不是很高的场合;OS_TMR_OPT_PRIO_HIGH 类型的软件定时器的 Server 线程的优先级较高,适时性较好,既支持在线程中进行相关操作，也支持在中断的 ISR 中进行相关操作，一般用于对定时的适时性要求较高的设备驱动。
Melis 平台的 SoftTimer 支持以下几种操作接口：

- ✓ 创建定时器
- ✓ 销毁定时器
- ✓ 启动定时器
- ✓ 停止定时器
- ✓ 获取操作错误类型

30.2.8. QuerryError

➢ **PROTOTYPE**

```
__u16 esKRNL_TmrError(__krnl_stmr_t *ptmr);
```

➢ **ARGUMENTS**

ptmr　　软件定时器句柄;

➢ **RETURNS**

软件定时器操作失败的错误类型：

OS_NO_ERR，	*操作定时器成功，未发现异常;*
KRNL_ERR_TMR_INVALID_PERIOD，	*定时器周期参数非法;*
KRNL_ERR_TMR_INVALID_OPT，	*定时器类型配置参数非法;*
KRNL_ERR_TMR_NON_AVAIL，	*分配定时器失败;*
KRNL_ERR_TMR_INACTIVE，	*软件定时器未启动;*
KRNL_ERR_TMR_INVALID_TYPE，	*句柄指向的不是一个合法的定时器;*
KRNL_ERR_TMR_INVALID，	*定时器句柄为空;*
KRNL_ERR_TMR_ISR，	*在中断 ISR 中操作定时器，低优先级的软件定时器不允许在 ISR 中进行相关操作;*
KRNL_ERR_TMR_INVALID_STATE，	*定时器状态不合法;*

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

KRNL_ERR_TMR_STOPPED，　　　定时器处于停止未运行状态；
KRNL_ERR_TMR_NO_CALLBACK，　　定时器没有注删合法的回调函数；

➢ **DESCRIPTION**

查询软件定时器相关操作失败的错误类型。当使和定时器的其它接口失败时，可以通过该接口来查询导致操作失败的错误原因，如果软件定时器句柄为空，则查询的是定时器上次操作失败的全局错误类型；

➢ **DEMO**

```
//查询定时器的全局错误类型
__u16    tmpErrType;
tmpErrType = esKRNL_TmrError(0);
__inf(”Soft timer error type:%d\n”, tmpErrType);

//查询定时器 hSoftTmr 的错误类型
tmpErrType = esKRNL_TmrError(hSoftTmr);
__inf(”Error type of soft timer hSoftTmr is:%d\n”, tmpErrType);
```

30.2.9. CreateTimer

➢ **PROTOTYPE**

```
__krnl_stmr_t *esKRNL_TmrCreate( __u32        period,
                                 __u8         opt,
                                 OS_TMR_CALLBACK     callback,
                                 void         *arg);
```

➢ **ARGUMENTS**

period　　定时器的时间周期，以 ms 为单位，精度为 1ms；
opt　　　定时器的类型配置参数：
　　bit0~bit3，定时器周期类型配置：
　　　OS_TMR_OPT_PERIODIC　自动周期性触发型定时器，定时时间到达时，定时器自动触发进行下一次定时；
　　　OS_TMR_OPT_ONE_SHOT　单次触发型定时器，定时时间到达时，定时器自动停止，不再定时；
　　bit4~bit7，定时器优先级别配置：
　　　OS_TMR_OPT_PRIO_LOW　低优先级定时器；
　　　OS_TMR_OPT_PRIO_HIGH 高优先级定时器；
callback　　定时器服务的回调函数；
arg　　　传给定时服务回调函数的参数；

➢ **RETURNS**

如果创建定时器成功返回定时器句柄，否则返回 NULL。如果创建定时器失败，可以通过 QuerryError 来获取导致失败的原因；

➢ **DESCRIPTION**

该函数用于创建软件定时器，定时器的回调由参数 callback 传入，当定时期到达时，定时器的 Server 线程通过调用 callback 来执行用户的定时器处理函数。

➢ **DEMO**

```
// 创建一个定时周期为 100ms 的周期性触发低优先级软件时器
extern void SoftTmrHdl(void *arg);
```

```
__krnl_stmr_t    *hSoftTmr;
hSoftTmr = esKRNL_TmrCreate(100,
                            OS_TMR_OPT_PERIODIC|OS_TMR_OPT_PRIO_LOW,
                            SoftTmrHdl,
                            0);
if(hSoftTmr == (__krnl_stmr_t *)0)
{
    __wrn("Create soft timer failed!\n");
}
```

30.2.10. DeleteTimer

➢ **PROTOTYPE**

```
__s32 esKRNL_TmrDel(__krnl_stmr_t     *ptmr);
```

➢ **ARGUMENTS**

ptmr　　软件定时器句柄；

➢ **RETURNS**

删除软件定时器的结果：

EPDK_OK，　　删除软件定时器成功；

EPDK_FAIL，删除软件定时器失败；

➢ **DESCRIPTION**

删除软件定时器。低优先级的定时器不允许在定时器的回调函数中删除软件定时器，否则，可能会导致系统死锁。如果删除定时器失败，可以通过 QuerryError 函数来查询错误的原因；

➢ **DEMO**

```
//删除软件定时器 hSoftTmr
__s32    result;
result = esKRNL_TmrDel(hSoftTmr);
if(result != EPDK_OK)
{
    __u16    tmpErrType;
    tmpErrType = esKRNL_TmrError(hSoftTmr);
    __wrn("Try to delete hSoftTmr failed! Error Type:%d\n", tmpErrType);
}
```

30.2.11. StartTimer

➢ **PROTOTYPE**

```
__s32 esKRNL_TmrStart(__krnl_stmr_t   *ptmr);
```

➢ **ARGUMENTS**

ptmr　　软件定时器句柄；

➢ **RETURNS**

启动软件定时器的结果：

EPDK_OK，　　　启动软件定时器成功；

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

EPDK_FAIL， 启动软件定时器失败；

- **DESCRIPTION**

启动软件定时器。低优先级的定时器不允许在定时器回调函数中启动定时器，否则可能会导致系统死锁。如果启动定时器失败，可以通过 QuerryError 函数来查询错误的原因；

- **DEMO**

```
//启动软件定时器 hSoftTmr
__s32    result;
result = esKRNL_TmrStart(hSoftTmr);
if(result != EPDK_OK)
{
        __u16    tmpErrType;
        tmpErrType = esKRNL_TmrError(hSoftTmr);
        __wrn(“Try to start hSoftTmr failed! Error type:%d\n”, hSoftTmr);
}
```

30.2.12. StopTimer

- **PROTOTYPE**

```
__s32 esKRNL_TmrStop(__krnl_stmr_t    *ptmr);
```

- **ARGUMENTS**

ptmr　　软件定时器句柄；

- **RETURNS**

停止软件定时器的结果：

EPDK_OK，　　　停止软件定时器成功；
EPDK_FAIL，　停止软件定时器失败；

- **DESCRIPTION**

停止软件定时器。低优先级的定时器不允许在定时器回调函数中停止定时器，否则可能会导致系统死锁。如果停止定时器失败，可以通过 QuerryError 函数来查询错误的原因；

- **DEMO**

```
//停止软件定时器 hSoftTmr
__s32    result;
result = esKRNL_TmrStop(hSoftTmr);
if(result != EPDK_OK)
{
        __u16    tmpErrType;
        tmpErrType = esKRNL_TmrError(hSoftTmr);
        __wrn(“Try to stop hSoftTmr failed! Error type:%d\n”, hSoftTmr);
}
```

30.3. HwTimer

Melis 平台也提供 HardwareTimer 给用户使用，但是由于 HardwareTimer 的资源有限，所以 HardwareTimer 一般预先分配给有特定需求的设备驱动来使用。HardwareTimer 定时器的适时性不会受系统负荷的影响，定

时精度定时器用户自己指定，可以达到 1ms。HardwareTimer 定时器的精度定得过高会对系统性能有很大的负面影响。
Melis 平台 HardwareTimer 定时器支持以下几种编程接口：

- ✓ 申请定时器
- ✓ 释放定时器
- ✓ 启动定时器
- ✓ 停止定时器

30.3.1. RequestTimer

➢ **PROTOTYPE**

```
__hdle esTIME_RequestTimer( __hw_tmr_type_t  *pType,
                            __pCBK_t          pHdlr,
                            void              *pArg,
                            char              *pUsr);
```

➢ **ARGUMENTS**

pType　定时器的特性描述参数：
　pType->precision　定时器的精度，秒/毫秒/微秒；
　pType->leastCount　定时器的定时长度要求，以 pType->precision 为单位；
pHdlr　硬件定时器的回调处理函数；
pArg　需要传给硬件定时器回调函数的参数；
pUsr　用户身份描述信息；

➢ **RETURNS**

如果成功申请到硬件定时器，返回定时器的句柄；否则，返回 NULL。

➢ **DESCRIPTION**

该函数用于申请一个硬件定时器，硬件定时器的特性由用户根据自己的实际需来指定。用户可以指定定时器的定时粗度、定时的时间长度等。

➢ **DEMO**

```
//申请一个硬件定时器，定时精度为毫秒，定时能力不低于 500 毫秒
extern   __s32 HwTmrHdlr(void *arg)
__hdle   TestHwTmr;
__hw_tmr_type_t  tmpTmrType;
tmpTmrType.precision = CSP_TMRC_TMR_PRECISION_MILLI_SECOND;
tmpTmrType. leastCount = 500;
TestHwTmr = esTIME_RequestTimer(&tmpTmrType, HwTmrHdlr, 0, "DEMO");
if(TestHwTmr == (__hdle)0)
{
    __wrn("Try to request hardware timer failed!\n");
}
```

30.3.2. ReleaseTimer

➢ **PROTOTYPE**

```
__s32 esTIME_ReleaseTimer(__hdle hTmr);
```

- **ARGUMENTS**

hTmr　　硬件定时器句柄；

- **RETURNS**

释放硬件定时器的结果：

EPDK_OK，　释放硬件定时器成功；

EPDK_FAIL，释放硬件定时器失败；

- **DESCRIPTION**

释放成功申请到的硬件定时器。

- **DEMO**

```
//释放硬件定时器 TestHwTmr
__s32    result;
result = esTIME_ReleaseTimer(TestHwTmr);
if(result != EPDK_OK)
{
    __wrn("Release hardware timer TestHwTmr failed!\n");
}
```

30.3.3. StartTimer

- **PROTOTYPE**

```
__s32 esTIME_StartTimer( __hdle       hTmr,
                         __u32        nPeriod,
                         __hw_tmr_mode_e nMode);
```

- **ARGUMENTS**

hTmr　　硬件定时器句柄；

nPeriod　硬件定时器的定时周期，以用户指定的精度为单位；

nMode　　定时器的触发类型：

CSP_TMRC_TMR_MODE_CONTINU,　　*硬件定时器周期性自动触发;*

CSP_TMRC_TMR_MODE_ONE_SHOOT,　*硬件定时器单次触发;*

- **RETURNS**

启动硬件定时器的结果：

EPDK_OK，　启动硬件定时器成功;

EPDK_FAIL，启动硬件定时器失败;

- **DESCRIPTION**

启动硬件定时器，用户需要指定定时器的运行周期，以申请定时器时指定的精度为单位。此外，用户还需要指定定时器的触发类型是周期性自动触发（定时器定时到时，硬件自动启动进行下次定时）还是单次触发（定时器定时到时，硬件自动停止）；

- **DEMO**

```
//启动硬件定时器 TestHwTmr，定时周期为 100ms，自动周期性触发
__s32    result;
result = esTIME_StartTimer(TestHwTmr, 100, CSP_TMRC_TMR_MODE_CONTINU);
if(result != EPDK_OK)
{
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
        __wrn("Try to start timer TestHwTmr failed!\n");
    }
```

30.3.4. StopTimer

- **PROTOTYPE**

```
    __s32 esTIME_StopTimer(__hdle hTmr);
```

- **ARGUMENTS**

hTmr　　硬件定时器句柄；

- **RETURNS**

停止硬件定时器的结果：

EPDK_OK，　停止硬件定时器成功；

EPDK_FAIL，停止硬件定时器失败；

- **DESCRIPTION**

停止硬件定时器。

- **DEMO**

```
    //启动硬件定时器 TestHwTmr
    __s32    result;
    result = esTIME_StopTimer(TestHwTmr);
    if(result != EPDK_OK)
    {
        __wrn("Try to stop timer TestHwTmr failed!\n");
    }
```

30.4. Counter

Melis 平台提供了一组 Counter 用于计时，用户可以基于 Counter 来实现自己的同步时钟或统计某时间段的长度。例如播放视频时作为视频帧播放的参考时钟，程序段运行时间统计等等。
Melis 内核 Counter 支持以下一些编程接口：

- ✓ 申请计时器
- ✓ 释放计时器
- ✓ 启动计时器
- ✓ 停止计时器
- ✓ 暂停计时器
- ✓ 恢复计时器
- ✓ 设置计时器时间
- ✓ 查询计时器时间
- ✓ 设置计时器精度
- ✓ 查询计时器状态

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

30.4.1. RequestCounter

- **PROTOTYPE**

```
__hdle esTIME_RequestCntr(__pCB_ClkCtl_t cb, char *pUsr);
```

- **ARGUMENTS**

cb　　时钟变更通知回调函数，兼容旧系统的无用参数，固定为 NULL 即可；
pUsr　　申请计时器的用户身份信息；

- **RETURNS**

如果成功申请到计时器，返回计时器句柄，否则，返回 NULL；

- **DESCRIPTION**

申请计时器资源。

- **DEMO**

```
//申请计时器
__hdle       hTestCntr;
hTestCntr = esTIME_RequestCntr(0,  "DEMO");
if(hTestCntr == (__hdle)0)
{
    __wrn("Try to request counter failed!\n");
}
```

30.4.2. ReleaseCounter

- **PROTOTYPE**

```
__hdle esTIME_ReleaseCntr(__hdle hCntr);
```

- **ARGUMENTS**

hCntr　　计时器句柄；

- **RETURNS**

释放计时器的结果：

EPDK_OK，　　释放计时器成功；
EPDK_FAIL，　释放计时器失败；

- **DESCRIPTION**

释放计时器资源。

- **DEMO**

```
//释放计时器 hTestCntr
__s32   result;
result = esTIME_ReleaseCntr(hTestCntr);
if(result != EPDK_OK)
{
    __wrn("Try to release counter hTestCntr failed!\n");
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

30.4.3. StartCounter

- **PROTOTYPE**

```
__s32 esTIME_StartCntr(__hdle hCntr);
```

- **ARGUMENTS**

hCntr　　计时器句柄；

- **RETURNS**

启动计时器的结果：

EPDK_OK，　　　启动计时器成功；

EPDK_FAIL，　启动计时器失败；

- **DESCRIPTION**

启动计时器。启动计时器前需要先设置好计时器的精度和初始值，并确保计时没有处理 pause 状态。

- **DEMO**

```
//启动计时器 hTestCntr
__s32    result;
result = esTIME_StartCntr(hTestCntr);
if(result != EPDK_OK)
{
    __wrn(“Try to start counter hTestCntr failed!\n”);
}
```

30.4.4. StopCounter

- **PROTOTYPE**

```
__s32 esTIME_StopCntr(__hdle hCntr);
```

- **ARGUMENTS**

hCntr　　计时器句柄；

- **RETURNS**

停止计时器的结果：

EPDK_OK，　　　停止计时器成功；

EPDK_FAIL，　停止计时器失败；

- **DESCRIPTION**

停止计时器。

- **DEMO**

```
//停止计时器 hTestCntr
__s32    result;
result = esTIME_StopCntr(hTestCntr);
if(result != EPDK_OK)
{
    __wrn(“Try to stop counter hTestCntr failed!\n”);
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

30.4.5. PauseCounter

- **PROTOTYPE**

```
__s32 esTIME_PauseCntr(__hdle hCntr);
```

- **ARGUMENTS**

hCntr　　计时器句柄；

- **RETURNS**

暂停计时器的结果：

EPDK_OK，　　　暂停计时器运行成功；

EPDK_FAIL，　暂停计时器运行失败；

- **DESCRIPTION**

暂停计时器运行。

- **DEMO**

```
//停止计时器 hTestCntr
__s32   result;
result = esTIME_PauseCntr(hTestCntr);
if(result != EPDK_OK)
{
    __wrn(“Try to pause counter hTestCntr failed!\n”);
}
```

30.4.6. ContinueCounter

- **PROTOTYPE**

```
__s32 esTIME_ContiCntr(__hdle hCntr);
```

- **ARGUMENTS**

hCntr　　计时器句柄；

- **RETURNS**

恢复计时器的结果：

EPDK_OK，　　　恢复计时器运行成功；

EPDK_FAIL，　恢复计时器运行失败；

- **DESCRIPTION**

恢复计时器运行。

- **DEMO**

```
//恢复计时器 hTestCntr
__s32   result;
result = esTIME_ContiCntr(hTestCntr);
if(result != EPDK_OK)
{
    __wrn(“Try to continue counter hTestCntr failed!\n”);
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

30.4.7. SetValue

- **PROTOTYPE**

```
__s32 esTIME_SetCntrValue(__hdle hCntr, __u32 value);
```

- **ARGUMENTS**

hCntr　　计时器句柄；

value　　要设置给计时器的值；

- **RETURNS**

设置计时器值的结果：

EPDK_OK，　　　设置计时器值成功；

EPDK_FAIL，　设置计时器值失败；

- **DESCRIPTION**

设置计时器的值。

- **DEMO**

```
//设置计时器 hTestCntr 的值为 0x8888
__s32    result;
result = esTIME_SetCntrValue(hTestCntr, 0x8888);
if(result != EPDK_OK)
{
    __wrn("Try to set counter hTestCntr's value failed!\n");
}
```

30.4.8. QuerryCounter

- **PROTOTYPE**

```
__u32 esTIME_QueryCntr (__hdle hCntr);
```

- **ARGUMENTS**

hCntr　　计时器句柄；

- **RETURNS**

查询到的计时器的值，如果 hCntr 非法，查询值返回为 0；

- **DESCRIPTION**

查询计时器的值。

- **DEMO**

```
//计时器 hTestCntr 的值为 0x8888
__u32    tmpCntrValue;
tmpCntrValue = esTIME_QuerryCntr(hTestCntr);
__inf("Value of counter hTestCntr is:%d\n", tmpCntrValue);
```

30.4.9. SetPrescale

- **PROTOTYPE**

```
__s32 esTIME_SetCntrPrescale (__hdle hCntr, __s32 prescl);
```

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

- **ARGUMENTS**

hCntr　计时器句柄；
prescl　计时器的分频系数；

- **RETURNS**

设置计时器的精度的结果：

EPDK_OK，　　设置计时器精度成功；
EPDK_FAIL，　设置计时器精度失败；

- **DESCRIPTION**

设置计时器有精度。计时器的时钟源为 24MHz，内部存在 1/2 的预分频，设置精度时，需要结合时钟源和预分频来处理。例如：设置计时器的精度为 1us，则计时器的分频系数为 12；如果精度设置为 1ms，则分频系数为 12*1024。

- **DEMO**

```
//设置计时器 hTestCntr 的计时精度为 1us
__s32    result;
result = esTIME_SetCntrPrescale(hTestCntr, 12);
if(result != EPDK_OK)
{
    __wrn("Try to set prescale for hTestCntr to 1us failed!\n");
}
```

30.4.10. QuerryState

- **PROTOTYPE**

```
__s32 esTIME_QuerryCntrStat(__hdle hCntr);
```

- **ARGUMENTS**

hCntr　计时器句柄；

- **RETURNS**

计时器的当前状态：

TMR_CNT_STAT_INVALID，计时器状态非法；
TMR_CNT_STAT_STOP，计时器处于停止状态；
TMR_CNT_STAT_RUN，　计时器处于运行状态；
TMR_CNT_STAT_PAUSE，　计时器处于暂停状态；

- **DESCRIPTION**

查询计时器当前的运行状态。

- **DEMO**

```
//查询计时器 hTestCntr 当前的运行状态
__s32    tmpStat;
tmpStat = esTIME_QuerryCntrStat(hTestCntr);
__inf("Stat of counter hTestCntr is:%d\n", tmpStat);
```

30.4.11. Demo

采用 Melis 平台提供的 counter 来测试某段程序运行的实际执行时间。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
__hdle  hTestTimer  =   NULL;
__u32   StartTime   =   0;
__u32   FinishTime  =   0;

void TestTimerInit(void)
{
    /* request one counter */
    hTestTimer = esTIME_RequestCntr(0, "USRTEST");
    if (NULL == hTestTimer) {
        __wrn("Request counter fail\n");
        return ;
    }

    /* set counter prescale : source counter preacale depend on platform */
    esTIME_SetCntrPrescale(hTestTimer, 12);
    esTIME_SetCntrValue(hTestTimer, 0);

    StartTime   = 0;
    FinishTime  = 0;
}

void TestTimerExit(void)
{
    /* release timer counter */
    if (hTestTimer)
    {
        esTIME_ReleaseCntr(hTestTimer);
    }
}

void TestTimerStart(void)
{
    if (hTestTimer)
    {
        StartTime = esTIME_QuerryCntr(hTestTimer);
    }
}

void TestTimerReport(void)
{
    __u32 ElapsedTime;

    if (hTestTimer)
    {
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
        FinishTime = esTIME_QuerryCntr(hTestTimer);
    }
    else
    {
        FinishTime = 0;
    }

    ElapsedTime = FinishTime - StartTime;
    eLIBs_Printf("Elapsed time is:%d\n", ElapseTime);
}


void Test(void)
{
    __s32   i;

    //初始化计时器资源
    TestTimerInit();
    //启动计时器开时计时
    TestTimerStart();
    for(i=0; i<0x10000000; i++)
    {
        ;
    }
    //报告程序实际运行的时间
    TestTimerReport();
    //释放计时器资源
    TestTimerExit();
}
```

30.5. Alarm

Melis 平台提供闹钟服务，支持开机状态下闹钟定时提醒，待机状态下闹钟定时唤醒系统，关机状态下闹钟定时开机等功能。系统只提供一个闹钟资源，如果想同时实现多个闹钟，需要在用户程序中自行组合来实现。

Melis 平台提供的闹钟服务编程接口有以下：

- ✓ 申请闹钟
- ✓ 释放闹钟
- ✓ 启动闹钟
- ✓ 查询闹钟

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

30.5.1. RequestAlarm

➢ **PROTOTYPE**

```
__hdle esTIME_RequestAlarm(__u32 mode);
```

➢ **ARGUMENTS**

mode　　申请的闹钟类型，目前只支持 ALARM_MODE_TIMER;

➢ **RETURNS**

如果申请闹钟成功，返回闹钟句柄，否则，返回 NULL。

➢ **DESCRIPTION**

申请闹钟资源;

➢ **DEMO**

```
//申请闹钟
__hdle        hTestAlarm;
hTestAlarm = esTIME_RequestAlarm(ALARM_MODE_TIMER);
if(hTestAlarm == (__hdle)0)
{
    __wrn("Try to request alarm failed!\n");
}
```

30.5.2. ReleaseAlarm

➢ **PROTOTYPE**

```
__s32 esTIME_ReleaseAlarm(__hdle alarm);
```

➢ **ARGUMENTS**

alarm　　　闹钟资源的句柄;

➢ **RETURNS**

如果释放闹钟资源的结果:

EPDK_OK，　　　释放闹钟资源成功;

EPDK_FAIL，　释放闹钟资源失败;

➢ **DESCRIPTION**

释放闹钟资源;

➢ **DEMO**

```
//释放闹钟 hTestAlarm
__s32    result;
result = esTIME_ReleaseAlarm(hTestAlarm);
if(result != EPDK_OK)
{
    __wrn("Try to release alarm hTestAlarm failed!\n");
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

30.5.3. StartAlarm

- **PROTOTYPE**

```
__s32 esTIME_StartAlarm(__hdle alarm, __u32 time);
```

- **ARGUMENTS**

Alarm　　闹钟资源句柄；
time　　闹钟定时时间，基于当前时间的偏移，以秒为单位；

- **RETURNS**

如果申请闹钟成功，返回闹钟句柄，否则，返回 NULL。

- **DESCRIPTION**

申请闹钟资源；

- **DEMO**

```
//启动闹钟，闹钟定时为 30 分钟以后
__s32    result;
Result = esTIME_StartAlarm()
```

30.5.4. StopAlarm

- **PROTOTYPE**

```
__hdle esTIME_RequestAlarm(__u32 mode);
```

- **ARGUMENTS**

mode　　；

- **RETURNS**

如果申请闹钟成功，返回闹钟句柄，否则，返回 NULL。

- **DESCRIPTION**

申请闹钟资源；

- **DEMO**

30.5.5. QuerryAlarm

- **PROTOTYPE**

```
__hdle esTIME_RequestAlarm(__u32 mode);
```

- **ARGUMENTS**

mode　　；

- **RETURNS**

如果申请闹钟成功，返回闹钟句柄，否则，返回 NULL。

- **DESCRIPTION**

申请闹钟资源；

- **DEMO**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

31. 内核编指南-内存管理

31.1. Introduction

31.1.1. Description

Melis 操作系统是由全志科技自主研发的一套精简、高效、易用的实时多任务操作系统，有着完善的内存管理、模块管理、中断管理、时钟管理、设备管理以及功耗管理。开发者可以基于 Melis 操作系统规划自己的方案，自由地开发应用程序和扩展设备驱动。

Melis 内核支持虚拟内存管理，内存分配灵活宜用。用户可以根据自己的实际需求，向内核申请可以满足自己特定需求的内存块。

Melis 内核支持用户建立自己的堆，实现独立的内存分配管理。用户释放资源时，销毁自己建立的堆，可以将占用的内存资源一次性全部归还给系统，以避免内存资源丢失。

Melis 内核还提供了一些 Cache 相关的操作，用户可以将自己的一段数据或代码锁定在 Cache 中，以实现局部加速。

31.1.2. Purpose

本文档主要讲述 Melis 内核内存管理相关的编程接口，让开发者能快速掌握 Melis 系统上的内存管理特性和编程接口，并基于系统提供的内存管理来开发自己的程序。

31.1.3. Reference

读者可以先了解一些基于 MMU 的虚拟内存管理相关的知识，了解一些内存分配和释放常见的接口。

31.2. Palloc

Melis 操作系统上所有内存分配的源头都是基于内存页池的页分配，Melis 内核上一个页(page)的大小为 1Kbyte。内核初始化时，将所有的内存都放置在系统页池中，系统堆的创建、虚拟空间的创建等等用到的内存全部由页分配而来。页分配得到的内存块在物理空间上是连续的，因此，一些对物理空间连续性有要求的用户，必须采用 palloc 来分配内存。

31.2.1. PageAlloc

- **PROTOTYPE**

```
void *esMEMS_Palloc(__u32 npage, __u32 mode);
```

- **ARGUMENTS**

npage　　请求分配的内存页面数，页面大小为 1Kbytes;

mode　　请求分配的内存的模式，常见于一些对内存访问有特定限制的硬件模块：

bit0~bit3，分配的内存块的地址对齐要求：

MEMS_PALLOC_MODE_BND_NONE，　内存块对地址对齐没有要求；
MEMS_PALLOC_MODE_BND_2K，　内存块要求地址以 2Kbyte 对齐；
MEMS_PALLOC_MODE_BND_4K，　内存块要求地址以 4Kbyte 对齐；
MEMS_PALLOC_MODE_BND_8K，　内存块要求地址以 8Kbyte 对齐；
MEMS_PALLOC_MODE_BND_16K，　内存块要求地址以 16Kbyte 对齐；
MEMS_PALLOC_MODE_BND_32K，　内存块要求地址以 32Kbyte 对齐；

Bit4~bit7，分配的内存块的边界要求：

MEMS_PALLOC_MODE_BNK_NONE，　内存块对边界没有限制；
MEMS_PALLOC_MODE_BNK_2M，　内存块不能跨越 2Mbyte 的边界；
MEMS_PALLOC_MODE_BNK_4M，　内存块不能跨越 4Mbyte 的边界；
MEMS_PALLOC_MODE_BNK_8M，　内存块不能跨越 8Mbyte 的边界；
MEMS_PALLOC_MODE_BNK_16M，　内存块不能跨越 16Mbyte 的边界；
MEMS_PALLOC_MODE_BNK_32M，　内存块不能跨越 32Mbyte 的边界；
MEMS_PALLOC_MODE_BNK_64M，　内存块不能跨越 64Mbyte 的边界；
MEMS_PALLOC_MODE_BNK_128M，　内存块不能跨越 128Mbyte 的边界；
MEMS_PALLOC_MODE_BNK_256M，　内存块不能跨越 256Mbyte 的边界；
MEMS_PALLOC_MODE_BNK_512M，　内存块不能跨越 512Mbyte 的边界；

Bit8~bit12，分配的内存块在物理地址空间上的要求：

MEMS_PALLOC_MODE_AREA_NONE，内存块在物理地址空间上无限制；
MEMS_PALLOC_MODE_AREA_2M，　内存块必须位于物理内存的低 2Mbyte 空间以内；
MEMS_PALLOC_MODE_AREA_4M，　内存块必须位于物理内存的低 4Mbyte 空间以内；
MEMS_PALLOC_MODE_AREA_8M，　内存块必须位于物理内存的低 8Mbyte 空间以内；
MEMS_PALLOC_MODE_AREA_16M，　内存块必须位于物理内存的低 16Mbyte 空间以内；
MEMS_PALLOC_MODE_AREA_32M，　内存块必须位于物理内存的低 32Mbyte 空间以内；
MEMS_PALLOC_MODE_AREA_64M，　内存块必须位于物理内存的低 64Mbyte 空间以内；
MEMS_PALLOC_MODE_AREA_128M，内存块必须位于物理内存的低 128Mbyte 空间以内；
MEMS_PALLOC_MODE_AREA_256M，内存块必须位于物理内存的低 256Mbyte 空间以内；

- **RETURNS**

如果申请分配内存成功，返回申请到的内存首地址；否则，返回 NULL；

- **DESCRIPTION**

该函数用于从内存页池中申请分配内存，页池中的内存资源管理以页为单位。当申请内存给硬件访问时，必须使用该接口来分配内存，因为只有该接口分配到的内存可以保证分配到的内存块在物理地址空间上是连续的。通过 esMEMS_Palloc 申请到的内存必须通过 esMEMS_Pfree 接口进行释放，否则会导致内存泄露。

- **DEMO**

```
// 申请 512Kbyte 内存，地址必须 4Kbyte 对齐，位于物理地址空间的低
// 32Mbyte 以内，对跨边界没有要求
void     *pTestBuf;
pTestBuf = esMEMS_Palloc(512,
                         MEMS_PALLOC_MODE_BND_4K
                         | MEMS_PALLOC_MODE_BNK_NONE
                         |  MEMS_PALLOC_MODE_AREA_32M);
```

```
if(pTestBuf == 0)
{
    __wrn("Try to request memory from pagepool failed!\n");
}
```

31.2.2. PageFree

- **PROTOTYPE**

```
void  esMEMS_Pfree(void *pblk, __u32 npage);
```

- **ARGUMENTS**

pblk　　要释放的内存块的首地址;
npage　　要释放的内存块的大小，以页为单位;

- **RETURNS**

none;

- **DESCRIPTION**

该函数用于释放通过 esMEMS_Palloc 申请到的内存资源，释放内存块时必须指定内存块的大小。

- **DEMO**

```
// 释放内存块 pTestBuf
esMEMS_Pfree(pTestBuf, 512);
```

31.3. VmCreate

Melis 平台提供虚拟内存创建功能，该功能从内存页池请求空闲内存，并在指空间上创建出连续内存空间。Melis 系统除了内核代码区及系统页池所占用的内存为系统预先固定分配，其它所有的内存空间全部由 VmCreate 创建得到。VmCreate 创建的内存区在虚拟地址空间上是连续的，物理地址空间不连续。

31.3.1. VMCreate

- **PROTOTYPE**

```
__s32 esMEMS_VMCreate(__mems_page_t *pblk, __u32 size, __u8 domain);
```

- **ARGUMENTS**

pblk　　待创建的内存空间的指针;
size　　待创建的内存空间的大小，以页为单位（1kbytes/page）;
domain　待创建的内存空间的访问权限;

- **RETURNS**

创建虚拟地址空间的结果:

EPDK_OK，　　创建内存空间成功;
EPDK_FAIL，　创建内存空间失败;

- **DESCRIPTION**

该函数用于在指定的地址上创建内存空间，指定地址空间时，必须确保待创建的地址空间未被使用，只有创建了内存空间的地址空间才能被 CPU 访问，否则会导致系统异常。该函数一般限制只允许内核使用，由该函数创建的内存空间只能用 VmDelete 进行销毁。

➢ **DEMO**

```
//在 0x60000000 处建 2Mbyte 的内存空间
__s32        result;
result = esMEMS_VMCreate((__mems_page_t *)0x60000000, 2048, 0);
if(result != EPDK_OK)
{
    __wrn("Try to create memory at 0x60000000 failed!\n");
}
```

31.3.2. VmDelete

➢ **PROTOTYPE**

```
void  esMEMS_VMDelete(__mems_page_t *pblk, __u32 size);
```

➢ **ARGUMENTS**

pblk　　要销毁的内存空间的地址；

size　　要销毁的内存空间的大小，以页为单位；

➢ **RETURNS**

none；

➢ **DESCRIPTION**

该函数用于销毁由 VmCreate 函数创建的内存空间。

➢ **DEMO**

```
// 销毁 0x60000000 处的 2Mbyte 的内存空间
esMEMS_VMDelete((__mems_page_t *)0x60000000, 2048);
```

31.4. Malloc

Melis 系统采用了高效的堆管理算法，可以快速地申请、释放内存。

31.4.1. HeapCreate

➢ **PROTOTYPE**

```
__s32  esMEMS_HeapCreate(void *heap, __u32 npage);
```

➢ **ARGUMENTS**

heap　　堆地址，待建立的堆的首地址；

npage　　堆的初始大小，以页为单位，堆可以动态增长；

➢ **RETURNS**

如果创建堆成功，返回 EPDK_OK，否则，返回 EPDK_FAIL。

➢ **DESCRIPTION**

该函数用于在指定的地址空间上创建堆，Melis 内核启动时，会创建系统堆。目前，所有基于 malloc 函数分配的内存全部来源于系统堆。对一些有特殊需求的用户，也可以在指定空间上建立自己的堆，堆所占用的内存在堆销毁时会全部被释放，避免内泄露。创建堆时，必须确保所指定的空间没有被使用，否则可能会导致系统异常。

➢ **DEMO**

```
//在地址 0x60000000 处建立初始大小为 512Kbyte 的堆
__s32    result;
result = esMEMS_HeapCreate((void *)0x60000000, 512);
if(result != EPDK_OK)
{
    __wrn("Try to create heap at 0x60000000 failed!\n");
}
```

31.4.2. HeapDelete

➢ **PROTOTYPE**

```
void  esMEMS_HeapDel(__mems_heap_t * heap);
```

➢ **ARGUMENTS**

heap　　待销毁的堆的首地址；

➢ **RETURNS**

none;

➢ **DESCRIPTION**

根据指定的堆首地址销毁指定的堆，堆销毁时，堆占用的所有内存会全部被释放。

➢ **DEMO**

```
//销毁 0x60000000 处的堆
esMEMS_HeapDel((__mems_heap_t *)0x60000000);
```

31.4.3. Malloc

➢ **PROTOTYPE**

```
void *esMEMS_Malloc(__mems_heap_t *heap, __u32 size);
```

➢ **ARGUMENTS**

heap　　待申请的内存块所在的堆，如果从系统堆中申请内存，该值为 0；

size　　待申请的内存块的大小，以 byte 为单位；

➢ **RETURNS**

如果申请内存块成功，返回内存块的首地址；否则，返回 NULL；

➢ **DESCRIPTION**

该函数用于从指定堆中申请内存块，该内存块只保证在虚拟空间上连续，不保证在物理地址空间上连续，因此不适用于和硬件交换数据的场合（如 FrameBuffer 等）。Melis 系统未启用局部堆，所有的 malloc 申请的内存都来源于系统堆，参数 heap 固定为 0；

➢ **DEMO**

```
//申请 198302bytes 的内存块
void    *pTestBuf;
pTestBuf = esMEMS_Malloc(0, 198302);
if(!pTestBuf)
{
    __wrn("Try to request memory failed!\n");
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
    }
```

31.4.4. Mfree

➢ **PROTOTYPE**

```
void  esMEMS_Mfree(__mems_heap_t *heap, void *buf);
```

➢ **ARGUMENTS**

heap　　待释放的内存块所在的堆的首地址，系统堆固定为 0;

buf　　待释放的内存块的首地址;

➢ **RETURNS**

none;

➢ **DESCRIPTION**

释放由堆中分配的内存块，如果内存块是从系统堆中分配，则 heap 参数固定为 0。

➢ **DEMO**

```
//释放内存块 pTestBuf
esMEMS_Mfree(0, pTestBuf);
```

31.4.5. Realloc

➢ **PROTOTYPE**

```
void* esMEMS_Realloc(__mems_heap_t *heap, void *buf, __u32 size);
```

➢ **ARGUMENTS**

heap　　内存块所在的堆的首地址，系统堆固定为 0;

buf　　待重新分配内存的内存块首地址;

size　　待重新分配内存的内存块大小，以 Byte 为单位;

➢ **RETURNS**

如果分配内存成功，返回内存块的新首地址；否则，返回 NULL;

➢ **DESCRIPTION**

该函数用于重新分配指定的内存块，旧内存块中的数据会被拷贝到新的内存块中。重新分配的内存块的首地址和旧内存块的首地址不是同一地址。如果重新分配内存成功，旧的内存块会被释放，返回新内存块的首地址；如果分配失败，旧的内存块不会被释放，返回 NULL 指针。

➢ **DEMO**

```
//重新分配内存块 pTestBuf, 大小为 201103
void     *tmpBuf;
tmpBuf = esMEMS_Realloc(0, pTestBuf, 201103);
if(tmpBuf)
{
    pTestBuf = tmpBuf;
}
else
{
    __wrn(“Try to realloc buffer pTestBuf failed!\n”);
}
```

31.4.6. VMalloc

- **PROTOTYPE**

```
void  *esMEMS_VMalloc(__u32 size);
```

- **ARGUMENTS**

size　　待分配内存的内存块大小，以 Byte 为单位；

- **RETURNS**

如果分配内存成功，返回内存块的首地址；否则，返回 NULL;

- **DESCRIPTION**

该函数用于申请大块内存，所分配的内存块在虚拟地址空间上连续，在物理地址空间上不连续。如果用户需要分配大块内存（如大于 64Kbyte 以上的内存块），且对内存的物理连续性没有要求，可以通过该接口来请求分配。通过 Malloc 接口申请分配内存时，如果内存块大于 64Kbyte，内核也会采用 Vmalloc 来进行分配。建议用户申请分配不需要物理空间连续的内存块时，统一使用 Malloc 接口。

- **DEMO**

```
//分配内存块 pTestBuf，大小为 201103
void    *pTestVBuf = esMEMS_VMalloc(201103);
if(!pTestVBuf)
{
    __wrn(“Try to request buffer failed!\n”);
}
```

31.4.7. Vmfree

- **PROTOTYPE**

```
void  esMEMS_VMfree(void *ptr);
```

- **ARGUMENTS**

ptr　　待释放的内存块的首地址，该内存块儿由 VMalloc 分配得来；

- **RETURNS**

none;

- **DESCRIPTION**

该函数用于释放由 VMalloc 分配得来的内存块。

- **DEMO**

```
// 释放内存块 pTestVBuf
esMEMS_VMfree(pTestVBuf);
```

31.5. Cache

Melis 系统提供了一组 Cache 相关的操作，用来处理 CPU 和硬件之间的数据交换。同时，Melis 内核还提供了 Cache 的锁定等操作，以支持用户对局部代码执行的加速。

Melis 平台的 Cache 操作提供以下编程接口：

- ✓ 无效 D-Cache
- ✓ 回写 D-Cache

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

- ✓ 回写并无效 D-Cache
- ✓ 无效 D-Cache 区域
- ✓ 回写 D-Cache 区域
- ✓ 回写并无效 D-Cache 区域
- ✓ 无效 I-Cache
- ✓ 无效 I-Cache 区域
- ✓ 锁定 I-Cache 区域
- ✓ 解锁 I-Cache 区域
- ✓ 锁定 D-Cache 区域
- ✓ 解锁 D-Cache 区域

31.5.1. FlushDCache

- **PROTOTYPE**

```
void esMEMS_FlushDCache(void);
```

- **ARGUMENTS**

none;

- **RETURNS**

none;

- **DESCRIPTION**

该函数用于无效 D-Cache，如果 D-Cache 中存在比主存更新的数据复本的话，执行此操作以后，D-Cache 中的这些更新将全部丢失，D-Cache 的映射关系也全部丢弃。一般不使用此函数，建议使用 FlushDCacheRegion 函数。

- **DEMO**

```
//清除所有 DCache 中的内容
esMEMS_FlushDCache();
```

31.5.2. CleanDCache

- **PROTOTYPE**

```
void esMEMS_CleanDCache(void);
```

- **ARGUMENTS**

none;

- **RETURNS**

none;

- **DESCRIPTION**

该函数用于回写 D-Cache 中的数据到内存，如果 D-Cache 中存在比主存更新的数据复本的话，执行此操作以后，D-Cache 中的数据将被回写到主存，Cache 和主存的映射关系维持不变，此函数一般很少用到，建议使用 CleanDCacheRegion 函数。

- **DEMO**

```
//回写 D-Cache 中的所有数据到主存
esMEMS_CleanDCache();
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

31.5.3. CleanFlushDCache

- **PROTOTYPE**

```
void esMEMS_CleanFlushDCache(void);
```

- **ARGUMENTS**

none;

- **RETURNS**

none;

- **DESCRIPTION**

回写并无效 D-Cache，如果 D-Cache 中存在比主存更新的数据复本，执行此函数以后，D-Cache 中的数据更新将会被回写到主存，并清除所有 D-Cache 和主存间的映射信息。该函数一般很少用，建议使用 CleanFlushDCacheRegion 函数。

- **DEMO**

```
//回写并无效所有的 D-Cache
esMEMS_CleanFlushDCache();
```

31.5.4. FlushDCacheRegion

- **PROTOTYPE**

```
void esMEMS_FlushDCacheRegion(void *addr, __u32 bytes);
```

- **ARGUMENTS**

addr　　需要无效 D-Cache 的区域的空间的首地址；
bytes　　需要无效 D-Cache 的区域的空间大小；

- **RETURNS**

none。

- **DESCRIPTION**

该函数用于无效一块内存区域的 D-Cache，如果一块内存区域上存在主存和 Cache 间的映射，执行此函数对该区域操作以后，该区域上的 D-Cache 映射信息将会丢失，如果 D-Cache 存在比主存更新的数据复本，该数据复本将会丢失。在启动 DMA 传输数据之前，一般先调用此函数对 DMA 传输的目标缓冲区进行处理，以防止 DMA 传输完数据以后出现 Cache 和主存数据不一致的情况。

- **DEMO**

```
//无效 0xc3800000 处 512Kbytes 的内存的 Cache
esMEMS_FlushDCacheRegion((void *)0xc3800000, 512*1024);
```

31.5.5. CleanDCacheRegion

- **PROTOTYPE**

```
void esMEMS_CleanDCacheRegion(void *addr, __u32 bytes);
```

- **ARGUMENTS**

addr　　待回写 Cache 的数据区首地址；
bytes　　待回写 Cache 的数据区数据长度；

- **RETURNS**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

none。

➢ **DESCRIPTION**

该函数用于回写某块区域上Cache的数据到主存。如果在某段内存区上，Cache和主存存在关联，且Cache存在比主存数据更新的复本，执行此操作以后，Cache 上的数据会被回写到主存，Cache 和主存间的映射关系维持不变。在启动 DMA 进行数据传之前，一般先执行此操作对 DMA 的源数据缓冲区进行处理，回写 Cache 中更新的数据复本到主存，防止 DMA 传送的是主存上旧的数据。

➢ **DEMO**

```
//回写 0xc2080000 处 256Kbytes 的数据到主存
esMEMS_CleanDCacheRegion((void *)0xc2080000, 256*1024);
```

31.5.6. CleanFlushDCacheRegion

➢ **PROTOTYPE**

```
void esMEMS_CleanFlushDCacheRegion(void *addr, __u32 bytes);
```

➢ **ARGUMENTS**

addr　　待回写并无效 Cache 的数据区首地址；

bytes　　待回写并无效 Cache 的数据区数据长度；

➢ **RETURNS**

none。

➢ **DESCRIPTION**

该函数用于回写某块区域上的 Cache 的数据到主存并无效 Cache 和主存的关联。如果在某段内存区上，Cache 和主存存在关联，且 Cache 存在比主存数据更新的复本，执行此操作以后，Cache 上的数据会被回写到主存，并丢弃 Cache 和主存间的关联。

➢ **DEMO**

```
//回写 0xc2060000 处 256Kbytes 的数据到主存并无效 Cache
esMEMS_CleanFlushDCacheRegion((void *)0xc2060000, 256*1024);
```

31.5.7. FlushICache

➢ **PROTOTYPE**

```
void esMEMS_FlushICache (void);
```

➢ **ARGUMENTS**

none;

➢ **RETURNS**

none。

➢ **DESCRIPTION**

该函数用于无效所有 I-Cache，执行此操作后，I-Cache 和主存间的所有关联将全部被清除，一般不用此函数，建议使用 FlushICacheRegion。

➢ **DEMO**

```
//无效 I-Cache
esMEMS_FlushICache();
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

31.5.8. FlushICacheRegion

➢ **PROTOTYPE**

```
void esMEMS_FlushICacheRegion (void *addr, __u32 bytes);
```

➢ **ARGUMENTS**

addr　　待无效 I-Cache 的内存区首地址；

bytes　　待无效 I-Cache 的内存区数据长度；

➢ **RETURNS**

none。

➢ **DESCRIPTION**

该函数用于无效某块内存区上关联的 I-Cache，执行此操作后，该区域上 I-Cache 和主存间的关联将被清除，该函数一般用于加载器加载完一块代码以后，对该带码区执行该操作，以保证读取指令的有效性。

➢ **DEMO**

```
//加载器加载 64100 字节的代码到 0xe0300000 处
……
//回写并无效 0xe0300000 处 64100 字节的数据到主存
esMEMS_CleanFlushDCacheRegion((void *)0xe0300000, 64100);
//无效 0xe0300000 处 64100 字节的内存区的 I-Cache 关联
esMEMS_FlushICacheRegion((void *)0xe0300000, 64100);
```

31.5.9. LockICache

➢ **PROTOTYPE**

```
__s32 esMEMS_LockICache(void *addr, __u32 size);
```

➢ **ARGUMENTS**

addr　　待锁定到 I-Cache 中的内存数据首地址；

bytes　　待锁定到 I-Cache 中的内存数据长度；

➢ **RETURNS**

锁定 I-Cache 的结果：

EPDK_OK，　　锁定 I-Cache 操作成功；

EPDK_FAIL，　锁定 I-Cache 操作失败。

➢ **DESCRIPTION**

该函数用于将指定地址上指定长度的代码锁定到 I-Cache 中，以实现对该段代码的执行进行加速。锁定 I-Cache 必须以 4Kbytes 为单位，不足 4Kbytes 的以 4Kbytes 进行处理，最多允许锁定 12Kbytes 到 I-Cache。锁定 I-Cache 可能会导致系统的整体性能下降。

➢ **DEMO**

```
//锁定 0xe0301000 处的 3Kbytes 到 I-Cache 以实现对该段代码进行加速
esMEMS_LockICache((void *)0xe0300400, 3*1024);
//注：实际锁存了 0xe0301000~0xe0301fff 区间 4Kbytes 的代码到 I-Cache 中
```

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

31.5.10. UnlockICache

- **PROTOTYPE**

```
__s32 esMEMS_UnlockICache(void *addr);
```

- **ARGUMENTS**

addr　　被锁定到 I-Cache 中的内存数据首地址；

- **RETURNS**

解除锁定 I-Cache 的结果：

EPDK_OK，　　　解除锁定 I-Cache 操作成功；

EPDK_FAIL，　解除锁定 I-Cache 操作失败。

- **DESCRIPTION**

该函数用于解除对 I-Cache 的锁定，被解除锁定的内存区域必须是曾被锁定到 I-Cache 中的区域；

- **DEMO**

```
//解除对 0xe0301000 处锁定的 I-Cache
esMEMS_UnlockICache((void *)0xe0301000);
```

31.5.11. LockDCache

- **PROTOTYPE**

```
__s32 esMEMS_LockDCache(void *addr, __u32 size);
```

- **ARGUMENTS**

addr　　待锁定到 D-Cache 中的内存数据首地址；

bytes　　待锁定到 D-Cache 中的内存数据长度；

- **RETURNS**

锁定 D-Cache 的结果：

EPDK_OK，　　　锁定 D-Cache 操作成功；

EPDK_FAIL，　锁定 D-Cache 操作失败。

- **DESCRIPTION**

该函数用于将指定地址上指定长度的代码锁定到 D-Cache 中，以实现对该段代码的执行进行加速。锁定 D-Cache 必须以 4Kbytes 为单位，不足 4Kbytes 的以 4Kbytes 进行处理，最多允许锁定 12Kbytes 到 D-Cache。锁定 D-Cache 可能会导致系统的整体性能下降。

- **DEMO**

```
//锁定 0xe030e000 处的 3Kbytes 到 D-Cache 以实现对该区域的数据访问加速
esMEMS_LockDCache((void *)0xe030e000, 3*1024);
//注：实际锁存了 0xe030e000~0xe030efff 区间 4Kbytes 的数据到 D-Cache 中
```

31.5.12. UnlockDCache

- **PROTOTYPE**

```
__s32 esMEMS_UnlockDCache(void *addr);
```

- **ARGUMENTS**

addr　　被锁定到 D-Cache 中的内存数据首地址；

➢ **RETURNS**

解除锁定 D-Cache 的结果：

EPDK_OK，　　解除锁定 D-Cache 操作成功；

EPDK_FAIL，　解除锁定 D-Cache 操作失败。

➢ **DESCRIPTION**

该函数用于解除对 D-Cache 的锁定，被解除锁定的内存区域必须是曾被锁定到 D-Cache 中的区域；

➢ **DEMO**

```
//解除对 0xe030e000 处锁定的 D-Cache
esMEMS_UnlockDCache((void *)0xe030e000);
```

31.6. Other

Melis 内核关于内存管理相关的一些其它常用接口：

- ✓ 虚拟地址转物理地址
- ✓ 判断内存块是物理连续
- ✓ 通过 IO 空间的物理地址获取虚拟地址
- ✓ 查询内存总容量
- ✓ 查询内存当前空闲容量

31.6.1. VA2PA

➢ **PROTOTYPE**

```
void  *esMEMS_VA2PA(void *addr);
```

➢ **ARGUMENTS**

addr　　需要转换物理地址的虚拟地址；

➢ **RETURNS**

如果地址转换成功，返回实际的物理地址；否则返回 NULL。

➢ **DESCRIPTION**

该函数用于将指定的虚拟地址转换成物理地址，通常用于配置硬件。如配置 DMA 传输数据时，需要配置数据块的物理地址给 DMA 控制器。待转换的虚拟地址必须是一个 CPU 可以访问的有效地址，否则地址转换会报错。

➢ **DEMO**

```
//转换虚拟地址 0xc3008000 的物理地址
void     *tmpBuf;
tmpBuf = esMEMS_VA2PA((void *)0xc3008000);
if(tmpBuf)
{
     __inf( “Physical address of 0xc3008000 is:%d\n” , tmpBuf);
}
```

31.6.2. PAContinue

- **PROTOTYPE**

```
__s32 esMEMS_PhyAddrConti(void *mem);
```

- **ARGUMENTS**

mem 待判断的内存块的虚拟地址；

- **RETURNS**

如果地址转换成功，返回实际的物理地址；否则返回 NULL。

- **DESCRIPTION**

该函数用于判断一个给定的内存块是否在物理地址空间上连续，一般用于块设备判断数据缓冲区是否可以直接供硬件访问，以优化数据操作效率。

- **DEMO**

```
//判断 0xc3008000 处 16Kbytes 的内存块是否在物理空间上连续
__s32   result;
result = esMEMS_PhyAddrConti((void *)0xc3008000, 16384);
if(result == EPDK_TRUE)
{
    __inf("Physical address of 0xc3008000 is continuouse\n");
}
```

31.6.3. GetIOVaByPa

- **PROTOTYPE**

```
__u32 esMEMS_GetIoVAByPA(__u32 phyaddr, __u32 size);
```

- **ARGUMENTS**

phyaddr IO 模块的物理地址；

size IO 模块的地址大小；

- **RETURNS**

如果 IO 模块物理地址有效，返回有效的虚拟地址，否则返回 NULL。

- **DESCRIPTION**

该函数用于通过 IO 模块的物理地址来获取 IO 模块的虚拟地址，一般用于设备驱动的开发。IO 模块的地址映射由 Melis 内核管理，模块只知道自己的物理地址，当操作 IO 空间时，需要通过内核查询到虚拟地址来访问。该函数只适用于对 IO 地址空间的查询，不支持普通内存地址空间查询。

- **DEMO**

```
//查询 IO 地址 0x01c20000 的 1024 字节空间的虚拟地址
__u32   tmpAddr;
tmpAddr = esMEMS_GetIOVAByPA((void *)0x01c20000, 1024);
if(tmpAddr)
{
    __inf("Virtual address of 0x01c20000 is:%d\n", tmpAddr);
}
```

全志科技版权所有，侵权必究

Copyright © 2018 by Allwinner. All rights reserved

31.6.4. TotalMemSize

- **PROTOTYPE**

```
__u32 esMEMS_TotalMemSize(void);
```

- **ARGUMENTS**

none;

- **RETURNS**

系统当前使用的内存容量大小，以字节为单位。

- **DESCRIPTION**

该函数用于查询系统当前使用的内存的总容量。

- **DEMO**

```
//查询系统内存总容量
__u32    tmpSize;
tmpSize = esMEMS_TotalMemSize();
__inf("Total memory size is:%d\n", tmpSize);
```

31.6.5. FreeMemSize

- **PROTOTYPE**

```
__u32 esMEMS_FreeMemSize(void);
```

- **ARGUMENTS**

none;

- **RETURNS**

系统当前空闲的内存容量大小，以字节为单位。

- **DESCRIPTION**

该函数用于查询系统当前空闲的内存容量。

- **DEMO**

```
//查询当前空闲内存容量
__u32    tmpFreeSize;
tmpFreeSize = esMEMS_FreeMemSize();
__inf("Free memory size is:%d\n", tmpFreeSize);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

32. 内核编指南-时钟管理

32.1. Introduction

32.1.1. Description

Melis 操作系统是由全志科技自主研发的一套精简、高效、易用的实时多任务操作系统，有着完善的内存管理、模块管理、中断管理、时钟管理、设备管理以及功耗管理。开发者可以基于 Melis 操作系统规划自己的方案，自由地开发应用程序和扩展设备驱动。
Melis 内核提供了灵活完善的时钟管理机制，用户在开发设备驱动时，不需要太多关注硬件时钟的处理细节，只需要理清硬件模块和时钟模块间的逻辑关系，即可基于 Melis 的时钟管理体系实现对特定硬件模块的时钟管理。

32.1.2. Purpose

本文档主要讲述 Melis 内核的时钟管理相关的编程接口，以期让用户能快速掌握在 Melis 平台开发设备驱动时对于时钟的处理方法以及控制设备的功耗。

32.1.3. Reference

读者可以先了解一些设备驱动对时钟的处理，包括设备的工作时钟设置、功耗控制等等。

32.2. SourceClock

Melis 系统将时钟划分两大类：源时钟和模块时钟。源时钟，顾名思义，也就是一些硬件模块工作时钟的来源，一般有晶振、PLL 锁相环等；模块时钟，即某一个特定硬件模块正常工作所需配置的时钟，一般有总线访问时钟、硬件工作时钟等相关配置。
Melis 系统对于源时钟提供以下几种操作接口：

- ✓ 设置源时钟的频率
- ✓ 获取源时钟的频率

32.2.1. SetSrcFreq

➢ **PROTOTYPE**

```
__s32 esCLK_SetSrcFreq(__u32 nSclkNo, __u32 nFreq);
```

➢ **ARGUMENTS**

nSclkNo　需要设置的源时钟的时钟号，不同硬件平台的时钟号定义会有异，具体请参考 CSP_CCM_sysClkNo_t 的定义；

nFreq　　需要设置给源时钟的频率值，以 Hz 为单位，由于时钟源能设置的值都是一些特定的离散值，请参考具体硬件编程手册给定合法的频率值；

- **RETURNS**

设置源时钟频率的结果：

EPDK_OK，　　设置源时钟的频率成功；

EPDK_FAIL，　设置源时钟的频率失败；

- **DESCRIPTION**

该函数用于设置源时钟的频率，源时钟根据不同的硬件平台，定义会有差异，具体需要参考具体硬件平台 CSP 关于源时钟号的定义。此外，每一个时钟源能支持的频率都是一些特定值，如 CORE_PLL 能支持的值为（6*n+30）Mhz（n=0,1,2,…），设置频率时必须根据硬件编程手册的要求给定合法的频率值，否则设置频率将会失败。

- **DEMO**

```
// 设置 CSP_CCM_SYS_CLK_CORE_PLL 源时钟的频率为 150Mhz
__s32    result;
result = esCLK_SetSrcFreq(CSP_CCM_SYS_CLK_CORE_PLL, 150*1000*1000);
if(resulg != EPDK_OK)
{
    __wrn("Try to set core pll to 150Mhz failed!\n");
}
```

32.2.2. GetSrcFreq

- **PROTOTYPE**

```
__u32 esCLK_GetSrcFreq(__u32 nSclkNo);
```

- **ARGUMENTS**

nSclkNo　　要获取频率的源时钟的时钟号；

- **RETURNS**

如果查询成功，返加源时钟的当前频率，否则，返回 0；

- **DESCRIPTION**

该函数用于查询某指定的源时钟的频率，如果源时钟没有打开或给定的时钟号非法，返回 0。

- **DEMO**

```
// 查询源时钟 CSP_CCM_SYS_CLK_CORE_PLL 的频率
__u32    tmpFreq;
tmpFreq = esCLK_GetSrcFreq(CSP_CCM_SYS_CLK_CORE_PLL);
if(tmpFreq != 0)
{
    __inf("Frequency of core pll is:%d\n", tmpFreq);
}
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

32.3. ModuleClk

Melis 平台的时钟管理体系对每一个硬件模块及总线访问进行定义，用户在开发某一硬件模块的驱动程序时，指定自己关心的模块时钟号，并实现自己的时钟频率调节回调函数，即可实现对硬件模块的时钟配置和控制。

Melis 内核时钟管理体系关于模块时钟管理提供的编程接口如下：

- ✓ 打开模块时钟
- ✓ 关闭模块时钟
- ✓ 注册时钟回调
- ✓ 注销时钟回调
- ✓ 设置模块时钟源时钟
- ✓ 获取模块时钟源时钟
- ✓ 设置模块时钟分频比
- ✓ 获取模块时钟分频比
- ✓ 设置硬件时钟状态
- ✓ 设置硬件模块工作状态

32.3.1. OpenMclk

- **PROTOTYPE**

```
__hdle esCLK_OpenMclk(__u32 nMclkNo);
```

- **ARGUMENTS**

nMclkNO　　待打开的模块时钟号，具体参见 CSP_CCM_modClkNo_t 的定义；

- **RETURNS**

如果打开模块时钟成功，返回模块时钟操作的有效句柄，否则，返回 NULL；

- **DESCRIPTION**

该函数用于打开模块时钟获取模块时钟操作的句柄，其它所有的模块时钟相关操作都基于该句柄来实现。打开动作只是软件管理层面上的操作，分配指定硬件模块时钟的操作权限给用户，不涉及具体的硬件设置。

- **DEMO**

```
//打开 MACC 的模块时钟
__hdle   hMAccMClk;
hMAccMClk = esCLK_OpenMclk(CSP_CCM_MOD_CLK_VE);
if(!hMAccMClk)
{
    WARNING("Open MACC module clock failed!\n");
    return EPDK_FAIL;
}
```

32.3.2. CloseMclk

- **PROTOTYPE**

```
__s32 esCLK_CloseMclk(__hdle hMclk);
```

- **ARGUMENTS**

hMclk　　待关闭的模块时钟的句柄；

- **RETURNS**

关闭模块时钟的结果：

EPDK_OK，　　　关闭模块时钟成功；

EPDK_FAIL，　关闭模块时钟失败；

- **DESCRIPTION**

该函数用于关闭给定的模块时钟句柄，关闭模块时钟句柄之前须确保硬件模块已关闭，避免造成功耗浪费。关闭模块时钟句柄，只是软件管理层面上的资源释放，只有关闭句柄以后，其它用户才能再次打开该模块时钟。

- **DEMO**

```
// 关闭 MACC 的模块时钟
__s32    result;
result = __s32 esCLK_CloseMclk(hMAccClk);
if(result != EPDK_OK)
{
    __wrn("Try to close MACC module clock failed!\n");
}
```

32.3.3. RegCallback

- **PROTOTYPE**

```
__s32 esCLK_MclkRegCb(__u32 nMclkNo, __pCB_ClkCtl_t pCb);
```

- **ARGUMENTS**

nMclkNo　　待注册时钟调节回调函数的模块时钟号；

pCb　　　　源时钟调节时的回调函数，由用户自己实现；

- **RETURNS**

注册时钟调节回调函数的结果：

EPDK_OK，　　　注册时钟回调成功；

EPDK_FAIL，　注册时钟回调失败；

- **DESCRIPTION**

该函数用于注册模块时钟源时钟调节的回调函数。由于源时钟是多个硬件模块共用的时钟源，当有用户请求调节源时钟的频率时，为确保所有共享该源时钟的硬件模块能够正常工作，必须通过回调函数来能知相关联的硬件模块时钟源调节的信息。如果某些硬件模块对源时钟的频率调节不敏感，也可以不用注册此回调函数。

- **DEMO**

```
// 注册 MACC 模块的时钟回调函数
extern __s32 cb_MACC_ClockChange(__u32 cmd, __s32 aux);
__s32    result;
result = esCLK_MclkRegCb(CSP_CCM_MOD_CLK_VE, cb_MACC_ClockChange);
if(result != EPDK_OK)
{
    __wrn("Register macc clock call-back failed!\n");
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
}
```

32.3.4. UnregCallback

- **PROTOTYPE**

```
__s32 esCLK_MclkUnregCb(__u32 nMclkNo, __pCB_ClkCtl_t pCb);
```

- **ARGUMENTS**

nMclkNo 待注销时钟调节回调函数的模块时钟号；

pCb 要注销的时钟回调函数；

- **RETURNS**

注销时钟调节回调函数的结果：

EPDK_OK， *注销时钟回调成功；*

EPDK_FAIL， *注销时钟回调失败；*

- **DESCRIPTION**

该函数用于注销模块的源时钟调节回调函数，设备驱动退出时，必须注销自己注册给时钟管理体系的回调函数，否则可能会导致系统异常。

- **DEMO**

```
// 注销 MACC 模块的时钟回调函数
extern __s32 cb_MACC_ClockChange(__u32 cmd, __s32 aux);
__s32    result;
result = esCLK_MclkUnregCb(CSP_CCM_MOD_CLK_VE, cb_MACC_ClockChange);
if(result != EPDK_OK)
{
    __wrn("unregister macc clock call-back failed!\n");
}
```

32.3.5. SetMclkSrc

- **PROTOTYPE**

```
__s32 esCLK_SetMclkSrc(__hdle hMclk, __s32 nSclkNo);
```

- **ARGUMENTS**

hMclk 待设置源时钟的模块时钟句柄；

nSclkNo 待设置给模块时钟的源时钟号；

- **RETURNS**

设置模块源时钟的结果：

EPDK_OK， *设置模块源时钟成功；*

EPDK_FAIL， *设置模块源时钟失败；*

- **DESCRIPTION**

该函数用于设置硬件模块工作的时钟源，一些硬件模块可选的时钟源可能有多个，驱动必须告诉时钟管理体系选用哪一个，时钟管理体系根据用户的选择将硬件模块纳入具体时钟源的体系以统一管理。

- **DEMO**

```
// 设定 MACC 模块的源时钟为 VEPLL
__s32    result;
```

```
result = esCLK_SetMclkSrc(hMAccMClk, CSP_CCM_SYS_CLK_VE_PLL);
if(result != EPDK_OK)
{
    __wrn("Try to set source clock for MACC failed!\n");
}
```

32.3.6. GetMclkSrc

➢ **PROTOTYPE**

```
__s32 esCLK_GetMclkSrc(__hdle hMclk);
```

➢ **ARGUMENTS**

hMclk　　待查询源时钟的模块时钟句柄;

➢ **RETURNS**

硬件模块源时钟的时钟号。

➢ **DESCRIPTION**

该函数用于查询硬件模块源时钟的时钟号。

➢ **DEMO**

```
// 查询 MACC 工作的源时钟
__s32    tmpSclk;
tmpSclk = esCLK_GetMclkSrc(hMAccMClk);
__inf("Source clock of MACC is:%d\n", tmpSclk);
```

32.3.7. SetMclkDiv

➢ **PROTOTYPE**

```
__s32 esCLK_SetMclkDiv(__hdle hMclk, __s32 nDiv);
```

➢ **ARGUMENTS**

hMclk　待设置时钟分频系数的模块时钟句柄;

nDiv　　待设置的分频系数;

➢ **RETURNS**

设置模块时钟分频系数的结果:

EPDK_OK,　　　*设置模块时钟分频系数成功;*

EPDK_FAIL,　*设置模块时钟分频系数失败;*

➢ **DESCRIPTION**

该函数用于设置硬件模块工作时钟相对于源时钟的分频系数，硬件模块时钟分频系数一般是有限的一些值，用户必须根据硬件编程手册给定合法的值，硬件模块工作的实际频率为 SclkFreq/nDiv。

➢ **DEMO**

```
//设置 MACC 工作的时钟为 VEPLL 的 1 分频
__s32    result;
result = esCLK_SetMclkDiv(hMAccMClk, 1);
if(result != EPDK_OK)
{
    __wrn("Try to set division for MACC clock failed!\n");
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
}
```

32.3.8. GetMclkDiv

- **PROTOTYPE**

```
__s32 esCLK_GetMclkDiv(__hdle hMclk);
```

- **ARGUMENTS**

hMclk　　模块时钟句柄；

- **RETURNS**

模块时钟的分频系数；

- **DESCRIPTION**

该函数用于查询硬件模块工作时钟的分频系数。

- **DEMO**

```
//查询 MACC 工作的时钟的分频系数
__s32    tmpDiv;
tmpDiv = esCLK_SetMclkDiv(hMAccMClk, 1);
__inf("Division of MACC clock is:%d\n", tmpDiv);
```

32.3.9. MclkOnOff

- **PROTOTYPE**

```
__s32 esCLK_MclkOnOff(__hdle hMclk, __s32 bOnOff);
```

- **ARGUMENTS**

hMclk　　模块时钟句柄；

bOnOff　 硬件模块时钟状态：

　　CLK_ON　- 打开时钟；

　　CLK_OFF - 关闭时钟；

- **RETURNS**

设置硬件模块时钟状态的结果：

　EPDK_OK，　　　设置硬件模块时钟状态成功；

　EPDK_FAIL，　设置硬件模块时钟状态失败；

- **DESCRIPTION**

该函数有于打开或关闭硬件模块的时钟，该函数会直接对硬件产生影响。

- **DEMO**

```
//打开 MACC 模块时钟
__s32    result;
result = esCLK_MclkOnOff(hMAccMClk, CLK_ON);
if(result != EPDK_OK)
{
    __wrn("Try to on MACC module clock failed!\n");
}
//关闭 MACC 模块时钟
result = esCLK_MclkOnOff(hMAccMClk, CLK_OFF);
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
if(result != EPDK_OK)
{
    __wrn("Try to on MACC module clock failed!\n");
}
```

32.3.10. MclkReset

- **PROTOTYPE**

```
__s32 esCLK_MclkReset(__hdle hMclk, __s32 bReset);
```

- **ARGUMENTS**

hMclk　模块时钟句柄；

bReset　硬件工作状态：

　0　- 硬件正常工作；

　!0　- 硬件处于自动 reset；

- **RETURNS**

设置硬件工作状态的结果：

EPDK_OK，　　设置硬件工作状态成功；

EPDK_FAIL，　设置硬件工作状态失败；

- **DESCRIPTION**

该函数用于置硬件模块为正常工作或自动 reset，当硬件处于自动 reset 状态时，会进入低功耗模式，但是硬件的寄存器配置不会丢失。一般在强制硬件模块进入低功耗或硬件模块出错需要复位时使用。

- **DEMO**

```
//reset MACC 模块
//置 MACC 为自动 reset 状态
esCLK_MclkReset(hMAccClk, 1);
//置 MACC 为正常工作状态
esCLK_MclkReset(hMAccClk, 0);
```

32.3.11. Demo

```
//配置 VE 模块时钟的 Demo 程序

//MACC 时钟调节通知函数
static __s32 cb_MACC_ClockChange(__u32 cmd, __s32 aux)
{
    __s32    result;

    switch(cmd)
    {
        case CLK_CMD_SCLKCHG_REQ:
        {
            ...
            break;
        }

        case CLK_CMD_SCLKCHG_DONE:
        {
            //config mpeg acc clock
```

```
            ...
            break
        }
    }

    return result;
}

//open MACC clock
__s32 MACC_OpenClk(void)
{
    //open macc module clock
    hMAccMClk = esCLK_OpenMclk(CSP_CCM_MOD_CLK_VE);
    if(!hMAccMClk)
    {
        WARNING("Open MACC module clock failed!\n");
        return EPDK_FAIL;
    }
    //register call-back fucntion for adjust ve clock
    esCLK_MclkRegCb(CSP_CCM_MOD_CLK_VE, cb_MACC_ClockChange);
    //set macc clock source to video pll
    esCLK_SetMclkSrc(hMAccMClk, CSP_CCM_SYS_CLK_VE_PLL);
    //set macc clock division
    esCLK_SetMclkDiv(hMAccMClk, 1);
    //enable macc module clock
    esCLK_MclkOnOff(hMAccMClk, CLK_ON);
    //disable ve module reset
    esCLK_MclkReset(hMAccMClk, 0);

    //open macc ahb clock
    hMaccAhbClk = esCLK_OpenMclk(CSP_CCM_MOD_CLK_AHB_VE);
    if(!hMaccAhbClk)
    {
        WARNING("Open MACC ahb clock failed!\n");
        return EPDK_FAIL;
    }
    esCLK_MclkOnOff(hMaccAhbClk, CLK_ON);

    //open macc dram access clock
    hDramMaccClk = esCLK_OpenMclk(CSP_CCM_MOD_CLK_SDRAM_VE);
    if(!hDramMaccClk)
    {
        WARNING("Open MACC Dram access clock failed!\n");
        return EPDK_FAIL;
    }
    esCLK_MclkOnOff(hDramMaccClk, CLK_ON);

    return EPDK_OK;
}

//close MACC clock
__s32 MACC_CloseClk(void)
{
    __u8        err;

    //unregister ve module clock
    if(hMAccMClk)
    {
        //enable ve module reset
        esCLK_MclkReset(hMAccMClk, 1);
```

```
            //disable ve module clock
            esCLK_MclkOnOff(hMAccMClk, CLK_OFF);
            esCLK_CloseMclk(hMAccMClk);
            //register call-back fucntion for adjust ve clock
            esCLK_MclkUnregCb(CSP_CCM_MOD_CLK_VE, cb_MACC_ClockChange);
            hMAccMClk = 0;
        }

        //close macc ahb clock
        if(hMaccAhbClk)
        {
            //disable mpeg acc ahb clock
            esCLK_MclkOnOff(hMaccAhbClk, CLK_OFF);
            esCLK_CloseMclk(hMaccAhbClk);
            hMaccAhbClk = 0;
        }

        //close macc dram access clock
        if(hDramMaccClk)
        {
            //disable mpeg acc module clock
            esCLK_MclkOnOff(hDramMaccClk, CLK_OFF);
            esCLK_CloseMclk(hDramMaccClk);
            hDramMaccClk = 0;
        }

        return EPDK_OK;
    }
```

32.4. PowerMan

Melis 系统有自己完善的功耗管理系统，负责控制系统在各种应用场合下的系统功耗。系统内核会时刻监控系统的负载，根据负载的变化情况适当调整 CPU 的运行速度，在不影响系统处理用户任务的情况下，尽可能低的降低 CPU 速度，以获得最小的系统功耗。

Melis 系统将系统功耗模式分为几个不同的 Level:

- SYS_PWM_MODE_LEVEL0：全速模式，系统运行在硬件可运行的最高频率状态，此时，系统功耗较大，系统瞬间响应速度最好，一般适用于对功耗不敏感（如 USB 连接）的场合；
- SYS_PWM_MODE_LEVEL1：高速模式，系统允许 CPU 运行的最低频率较高，可以满足一些对 CPU 瞬间响应能力有较高要求的应用需求，如视频播放时，CPU 必须能及时在规定时间段内处理完 LCD 控制器的中断等。
- SYS_PWM_MODE_LEVEL2：普通模式，系统允许 CPU 运行的最低频率比较低，在用户任务较轻的时候，CPU 的频率会下调较多，适用于大部分常见应用场景。
- SYS_PWM_MODE_LEVEL3：低速模式，系统允许 CPU 运行的最低频率是硬件可调节的最小值，一般适用于系统空闲状态。

各应用场景根据自身的实际需求向系统的功耗管理体系申请适当的功耗模式，系统内核根据所有用户的需求，来协调 CPU 运行速度的调节策略。同时，功耗管理体系和事件响应系统相结合，来改善事件响应的速度，从而能更适时快速地响应用户操作。

关于系统功耗模式的配置，具体请参见系统功耗配置脚本 pwm_cfg.ini 的定义。

Melis 系统功耗管理体系除了对 CPU 功耗的控制外，还对所有硬件设备作了管理，以降低硬件设备空闲时的功耗。

Melis 内核功耗管理系统提供以下几个编程接口：

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

- ✓ 申请功耗模式
- ✓ 释放功耗模式
- ✓ 锁定 CPU 频率
- ✓ 解除 CPU 频率锁定
- ✓ 用户事件通知
- ✓ 注册硬件设备
- ✓ 注销硬件设备
- ✓ 进入系统待机

32.4.1. ReqPwmMode

- **PROTOTYPE**

```
__s32 esPWM_RequestPwmMode(__s32 mode);
```

- **ARGUMENTS**

mode　　用户需要申请的功耗模式；

- **RETURNS**

如果向功耗管理系统申请功耗模式成功，返回 EPDK_OK，否则返回 EPDK_FAIL。

- **DESCRIPTION**

该函数用于向功耗管理系统申请功耗模式，功耗管理系统会综合考虑各用户的不同需求来协调处理。

- **DEMO**

```
//申请系统功耗模式 SYS_PWM_MODE_LEVEL1
__s32        result;
result = esPWM_RequestPwmMode(SYS_PWM_MODE_LEVEL1);
if(result != EPDK_OK)
{
    __wrn("Try to request pwm mode failed!\n");
}
```

32.4.2. RelPwmMode

- **PROTOTYPE**

```
__s32 esPWM_ReleasePwmMode(__s32 mode);
```

- **ARGUMENTS**

mode　　待释放的功耗模式；

- **RETURNS**

释放功耗模式的结果：

EPDK_OK，　释放功耗模式成功；

EPDK_FAIL，释放功耗模式失败；

- **DESCRIPTION**

释放指定的功耗模式。

- **DEMO**

```
//释放系统功耗模式 SYS_PWM_MODE_LEVEL1
__s32        result;
```

```
result = esPWM_ReleasePwmMode(SYS_PWM_MODE_LEVEL1);
if(result != EPDK_OK)
{
    __wrn("Try to release pwm mode failed!\n");
}
```

32.4.3. LockCpuFreq

- **PROTOTYPE**

```
__s32 esPWM_LockCpuFreq(void);
```

- **ARGUMENTS**

none;

- **RETURNS**

锁定 CPU 频率的结果：

EPDK_OK，　锁定 CPU 频率成功；

EPDK_FAIL，锁定 CPU 频率失败；

- **DESCRIPTION**

该函数用于锁定 CPU 频率。Melis 内核会实时地对 CPU 频率进行调节，如果用户希望 CPU 频率在一段时间内不变化，可以通过该函数锁定 CPU 频率。执行此操作以后，CPU 频率被锁定在当前允许运行最高频率值。该函数需要和 esPWM_UnlockCpuFreq 配对使用。

- **DEMO**

```
//锁定 CPU 频率
__s32       result;
result = esPWM_LockCpuFreq();
if(result != EPDK_OK)
{
    __wrn("Try to lock cpu frequency failed!\n");
}
```

32.4.4. UnlockCpuFreq

- **PROTOTYPE**

```
__s32 esPWM_UnlockCpuFreq(void);
```

- **ARGUMENTS**

none;

- **RETURNS**

解除 CPU 频率锁定的结果：

EPDK_OK，　解除 CPU 频率锁定成功；

EPDK_FAIL，解除 CPU 频率锁定失败；

- **DESCRIPTION**

该函数用于解除对 CPU 频率的锁定。

- **DEMO**

```
//解除 CPU 频率锁定
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
__s32       result;
result = esPWM_UnlockCpuFreq();
if(result != EPDK_OK)
{
    __wrn("Try to unlock cpu frequency failed!\n");
}
```

32.4.5. UsrEventNotify

- **PROTOTYPE**

```
void esPWM_UsrEventNotify(void);
```

- **ARGUMENTS**

none;

- **RETURNS**

none;

- **DESCRIPTION**

该函数用于通知功耗管理系统当前有用记事件触发，需要系统提高速度，快速对用户事件作出反应。例如有户触发按键时，按键消息处理程序可以通过此接口通知系统提速来对用户按键消息作处理。

- **DEMO**

```
//按键消息处理器接收到按键消息，通知道系统
esPWM_UsrEventNotify();
```

32.4.6. RegDevice

- **PROTOTYPE**

```
__s32 esPWM_RegDevice( __sys_pwm_dev_e    device,
                       __pCB_DPMCtl_t     cb,
                       void               *parg);
```

- **ARGUMENTS**

device	设备 ID，由系统定义，参见__sys_pwm_dev 的定义;
cb	功耗响应回调函数，用于处理功耗变化请求;
parg	功耗响应回调函数的参数;

- **RETURNS**

注册设备的结果：

EPDK_OK， 向功耗管理系统注册设备成功;

EPDK_FAIL，向功耗管理系统注册设备失败;

- **DESCRIPTION**

该函数用于向功耗管理系统注册设备，设备 ID 由系统预先定义。当系统工作模式发生变化（如需要进入待机等）时，功耗管理系统通过此接口通知设备驱动对设备状态作相应的处理。

- **DEMO**

```
//向功耗管理系统注册设备 PWM_DEV_DISPLAY
extern __s32 cb_DisplayPwm(__u32 cmd, void *arg);
__s32    result;
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
result = esPWM_RegDevice(PWM_DEV_DISPLAY, cb_DisplayPwm, 0);
if(result != EPDK_OK)
{
    __wrn("Try register pwm device failed!\n");
}
```

32.4.7. UnregDevice

- **PROTOTYPE**

```
__s32 esPWM_UnregDevice(__sys_pwm_dev_e device, __pCB_DPMCtl_t cb);
```

- **ARGUMENTS**

device　　设备 ID，由系统定义，参见__sys_pwm_dev 的定义；
cb　　功耗响应回调函数；

- **RETURNS**

注销设备的结果：
EPDK_OK，　向功耗管理系统注销设备成功；
EPDK_FAIL，向功耗管理系统注销设备失败；

- **DESCRIPTION**

该函数用于向功耗管理系统注销设备，和 esPWM_RegDevice 函数配对使用。

- **DEMO**

```
//向功耗管理系统注销设备 PWM_DEV_DISPLAY
extern __s32 cb_DisplayPwm(__u32 cmd, void *arg);
__s32    result;
result = esPWM_UnregDevice(PWM_DEV_DISPLAY, cb_DisplayPwm);
if(result != EPDK_OK)
{
    __wrn("Try unregister pwm device failed!\n");
}
```

32.4.8. EnterStandby

- **PROTOTYPE**

```
void esPWM_EnterStandby(__u32 time);
```

- **ARGUMENTS**

time　　自动关机的时间，以秒为单位。如果该值为 0，则表示系统进入 Standby 模式以后不用定时关机。

- **RETURNS**

none;

- **DESCRIPTION**

该函数用于通知系统进入 Standby，一般由桌面管理系统调用该函数通知系统进入待机模式以节省系统功耗。桌面管理系统在调用该函数进入 Standby 以前，需要自行处理诸如关闭 LCD、卸载 USB 设备等等。

- **DEMO**

```
//通知道系统进入 Standby，关机时间为 60 秒
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
esPWM_EnterStandby(60);
```

33. 内核编指南-模块管理

33.1. Introduction

33.1.1. Description

Melis 操作系统是由全志科技自主研发的一套精简、高效、易用的实时多任务操作系统，有着完善的内存管理、模块管理、中断管理、时钟管理、设备管理以及功耗管理。开发者可以基于 Melis 操作系统规划自己的方案，自由地开发应用程序和扩展设备驱动。
Melis 内核提供了简洁易用的应用程序、模块以及驱动程序框架，开发者只需要按照系统定义的规则来套用这些框架，就可以很容易开发出自己的应用和驱动。

33.1.2. Purpose

本文档主要讲述 Melis 内核对应用程序、模块及驱动的管理，以及基本框架和用法，以期能让用户快速掌握 Melis 平台的应用程序、模块及驱动的模型，开发自己的应用和相关驱动。

33.1.3. Reference

读者可以先了解一些关于 Linux 平台的应用程序以及驱动程序框架作为参考，以加深对 Melis 内核模块管理机制的理解。

33.2. Mechanism

Melis 平台上程序分为三大类：应用程序、用户模块和驱动程序，内核对三类程序提供了不同的管理机制和程序框架。Melis 平台不支持动态链接库，所有的程序都由静态链接产生。应用程序和用户模块采用虚拟地址，即段内地址；驱动程序采用实地址，即全局地址。应用程序和用户模块实际加载的地址由系统内核动态分配，而驱动程序的加载地址是唯一确定的。图 2-1 为系统内存空间分布图。

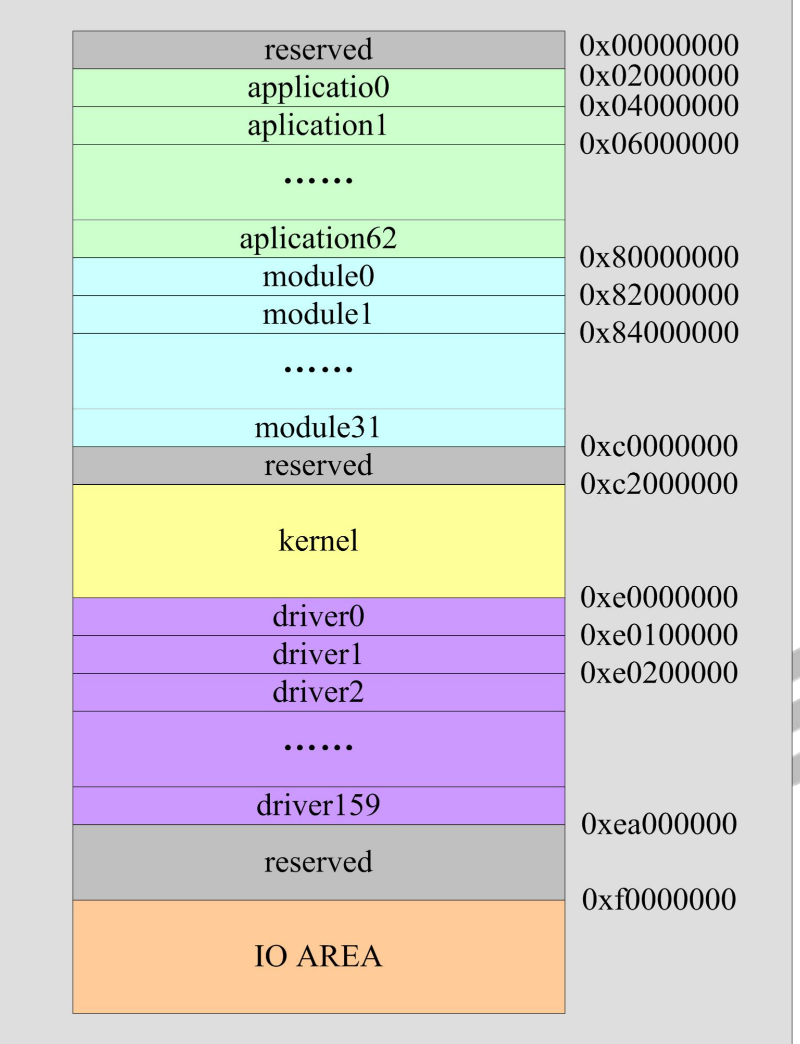


图 2-1　内存空间分布图

33.2.1. Application

Melis 平台的应用程序实际上就是一个进程，该类程序的特点是没有接口供其它程序调用，有唯一的程序入口点在加载时由内核加载器来负责启动。每一个应用程序占用一个独立的 32Mbytes 的段，起始地址均以 0 开始，应用程序之间不能直接互相访问。Melis 内核最多支持 63 个应用程序并存。

应用程序的 magic 文件:

```
extern __s32 main_entry(void *p_arg);
#pragma arm section rodata="MAGIC"
const __exec_mgsec_t exfinfo =
{
  {'e','P','D','K','.','e','x','f'},       //模块合法性标识
  0x01000000,          //版本号 1.00.0000
  0,                   //模块类型为应用程序
  0x1F00000,           //进程堆的地址（暂未启用）
  0x400,               //进程堆的初始大小（暂未启用）
  main_entry,          //进程的主线程入口
```

```
    0x40000,            //进程主线程栈大小（以 word 为单位）
    KRNL_priolevel3     //进程主线程的优先级
};
#pragma arm section
```

内核加载应用程序时，会找到”MAGIC”段，检查程序的 magic 是否合法，如果合法，则加载应用程序，否则，加载失败。必须将程序的类型指定为应用程序模块。如果加载应用程序成功，内核根据指定的主线程入口、线程栈大小以及线程优先级为应用程序创建主线程，此时应用程序被激活。

同一个应用程序可以被加载多次，彼此互不干扰。

应用程序的 scatter 文件：

```
LO_FIRST 0x0000 0x2000000        ;//应用程序整体不能起过 32Mbytes
{
    EXEC_APP  0x00000000         ;//应用程序以 0 地址作为起始地址
    {
        * (+RO)
        * (+RW)
        * (+ZI)
    }
    MAGIC  0xFFFF0000            ;//指定 MAGIC 段的加载地址，该段仅作为内核解
    {                            ;//解析 MAGIC 用，实际并不加载
        * (MAGIC)
    }
}
```

33.2.2. Module

Melis 平台的用户模块的空间管理和应用程序相似，每个用户模块占用一个独产的 32Mbytes 的段，以 0 地址作为程序的起始地址。用户模块没有程序主入口，而是提供一组接口交由其它程序调用。Melis 内核最多支多 32 个用户模块并存。

用户模块的 magic 文件:

```
#pragma arm section rodata="MAGIC"
const __mods_mgsec_t modinfo =
{
    {'e','P','D','K','.','m','o','d'},     //文件合法性标识
    0x01000000,         //用户模块的版本号 1.00.0000
    EMOD_TYPE_USER,     //程序类型为用户模块
    0xF0000,            //用户模块的堆地址（未启用）
    0x400,              //用户模块的堆的初始大小（未启用）
  {                     //用户模块的接口
      &MInit,           //模块初始化接口
      &MExit,           //模块退出接口
      &MOpen,           //模块打开接口
      &MClose,          //模块关闭接口
      &MRead,           //模块读取接口
      &MWrite,          //模块写入接口
```

```
        &MIoctrl          //模块控制接口
    }
};
#pragma arm section
```

内核加载用户模块时，会找到”MAGIC”段，检查程序的 magic 是否合法，如果合法，则加载用户模块，否则，加载失败。必须将程序的类型指定为用户模块。如果加载用户模块成功，则记录模块的 ID 及模块的接口，所有对模块的访问都基于注册的模块接口来实现。加载器加载用户模块后会执行模块的 MInit 函数来对模块进行初始化。

同一个用户模块可以被加载多次，彼此互不干扰。

用户模块的 scatter 文件：

```
LO_FIRST 0x0000 0x2000000       ;//用户模块整体不能起过 32Mbytes
{
    EXEC_MOD   0x00000000       ;//用户模块以 0 地址作为起始地址
    {
        * (+RO)
        * (+RW)
        * (+ZI)
    }
    MAGIC   0xFFFF0000          ;//指定 MAGIC 段的加载地址，该段仅作为内核解
    {                           ;//解析 MAGIC 用，实际并不加载
        * (MAGIC)
    }
}
```

33.2.3. Driver

Melis 平台的驱动程序的访问方式和用户模块相似，但是空间管理方式有很大差异。Melis 内核将驱动程序空间划分为以 1Mbytes 为单位的段，一共 196 个，每个驱动程序所占用的空间由内核预定义。一个驱动程序一般只占用一个段，也允许占用多个段，但是要保证代码空间不会重叠覆盖。关于各个驱动空间的分配情参见头文件《emod.h》。

显示驱动的 magic 文件:

```
#pragma arm section rodata="MAGIC"
const __mods_mgsec_t modinfo =
{
    {'e','P','D','K','.','m','o','d'},     //文件合法性标识
    0x01000000,              //驱动程序的版本号 1.00.0000
    EMOD_TYPE_DRV_DISP,      //程序为显示驱动
    0xF0000,             //驱动程序的堆地址（未启用）
    0x400,               //驱动程序的堆的初始大小（未启用）
  {                      //驱动程序的接口
      &MInit,            //驱动初始化接口
      &MExit,            //驱动退出接口
      &MOpen,            //驱动打开接口
      &MClose,           //驱动关闭接口
```

```
        &MRead,         //驱动读取接口
        &MWrite,        //驱动写入接口
        &MIoctrl        //驱动控制接口
    }
};
#pragma arm section
```

内核加载驱动程序时，会找到”MAGIC”段，检查程序的 magic 是否合法，如果合法，则加载驱动程序，否则，加载失败。必须将程序的类型指定为实际的驱动类型（如显示驱动等）。如果加载驱动程序成功，则记录驱动的 ID 及驱动访问接口，所有对驱动的访问都基于注册的访问接口来实现。加载器加载驱动程序后会执行模块的 MInit 函数来对模块进行初始化。

由于驱动程序的加载地址是唯一确定的，所以驱动只允许加载一次。

显示驱动的 scatter 文件：

```
LO_FIRST 0x0000 0x100000        ;//驱动程序整体不能起过 1Mbytes
{
    EXEC_MOD  0xe0300000        ;//显示驱动以 0xe0300000 地址作为起始地址
    {
        * (+RO)
        * (+RW)
        * (+ZI)
    }
    MAGIC  0xFFFF0000           ;//指定 MAGIC 段的加载地址，该段仅作为内核解
    {                           ;//解析 MAGIC 用，实际并不加载
        * (MAGIC)
    }
}
```

33.3. Application

Melis 平台的应用程序管理模块支持以下几个编程接口：

- ✓ 创建进程
- ✓ 删除进程
- ✓ 请求删除进程
- ✓ 启动应用程序

33.3.1. PCreate

➢ **PROTOTYPE**

```
__u8  esEXEC_PCreate(const char *pfilename);;
```

➢ **ARGUMENTS**

pfilename　　待创建的进程的文件名；

➢ **RETURNS**

如果创建进程成功，返回进程 ID，否则，返回 0；

- **DESCRIPTION**

该函数用于根据用户指定的应用程序文件名创建一个进程。

- **DEMO**

```
//创建 shell 进程，文件名为“shell.zjg”
__u8      uShellId;
uShellId = esEXEC_PCreate(“e:\\shell.zgj”);
if(uShellId == 0)
{
    __wrn(”Try to create process of shell failed!\n”);
}
```

33.3.2. PDel

- **PROTOTYPE**

```
__s32 esEXEC_PDel(__u8 pid);
```

- **ARGUMENTS**

pid　　待删除进程的 ID，由创建进程时得到；
　　　　如果指定 pid 为 EXEC_pidself，则删除当前进程；

- **RETURNS**

删除进程的结果：

EPDK_OK，　删除进程成功；
EPDK_FAIL，删除进程失败；

- **DESCRIPTION**

该函数用于删除指定 ID 的进程，进程也可以自己删除自己。删除进程时，内核会先请求杀死所有该进程创建的线程，如果尝试杀死线程失败，则会导致删除进程失败。

- **DEMO**

33.3.3. PDelReq

- **PROTOTYPE**

```
__s32 esEXEC_PDelReq(__u8 pid);
```

- **ARGUMENTS**

pid　　请求删除进程的 ID，由创建进程时得到；

- **RETURNS**

请求删除进程的结果：

EPDK_OK，　成功向进程管理器发送删除进程的消息；
EPDK_FAIL，向进程管理器发送删除请求失败；

- **DESCRIPTION**

该函数用于向进程管理器发送删某个进程的请求，仅仅是发送删除请求，而不直接删除进程，该消息需要等待内核守护线程来异步处理。

- **DEMO**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

33.3.4. Run

➢ **PROTOTYPE**

```
__u8  esEXEC_Run(const char     *pfilename,
                 void           *p_arg,
                 __u32          mode,
                 __u32          *ret);
```

➢ **ARGUMENTS**

pfilename　待启动的应用程序的文件名；

p_arg　需要传递给应用程序主线程的参数；

mode　应用程序运行的同步模式：

0　　，启动应用程序后控制权交回给启动者；

EXEC_CREATE_WAIT_RET，待被启动的应用程序退出以后才将控制权交回启动者，即启动者要等待被启动的应用程序退出；

ret　应用程序返回给启动者的运行结果；

➢ **RETURNS**

如果启动应用程序成功，返回进程 ID；否则，返回 0；

➢ **DESCRIPTION**

该函数用于向进程管理器发送删某个进程的请求，仅仅是发送删除请求，而不直接删除进程，该消息需要等待内核守护线程来异步处理。

➢ **DEMO**

33.3.5. Demo

```

```

33.4. Module

Melis 内核提供的模块管理机制允许用户将自己的程序插件化，只需要按标准的框架实现自己的模块程序，即可以通过系统提供的通用接口进行程序访问和调用。模块管理提供以下编程接口供用户使用：

- ✓ 安装模块
- ✓ 卸载模块
- ✓ 打开模块
- ✓ 关闭模块
- ✓ 模块读取
- ✓ 模块写入
- ✓ 模块操作

33.4.1. MInstall

➢ **PROTOTYPE**

```
__u32 esMODS_MInstall(const char *mfile, __u8 mode);
```

➢ **ARGUMENTS**

mfile　　待安装的模块的文件名；
mode　　模块的安装模式，暂未启用；

➢ **RETURNS**

如果安装模块成功，返回模块 ID；否则，返回 0。

➢ **DESCRIPTION**

该函数用于安装一个模块，模块名由 mfile 给出。模块安装时，模块管理器会调用模块的 MInit 函数对模块进行初始化。

➢ **DEMO**

```
//安装媒体播放中间件的主控模块
__u32    uCedarMid;

uCedarMid = esMODS_MInstall( “d:\mod\cedar.mod”, 0);
if(uCedarMid == 0)
{
    __wrn( “Try to install cedar.mod failed!\n” );
}
```

33.4.2. MUninstall

➢ **PROTOTYPE**

```
__s32 esMODS_MUninstall(__u8 mid);
```

➢ **ARGUMENTS**

mid　　待卸载的模块的 ID 号；

➢ **RETURNS**

卸载模块的结果：

EPDK_OK，　　卸载模块成功；
EPDK_FAIL，　卸载模块失败。

➢ **DESCRIPTION**

该函数用于根据指定的模块 ID 号来卸载模块。

➢ **DEMO**

```
//卸载 cedar 模块
__s32    result;

if(uCedarMid != 0)
{
    result = esMODS_MUninstall(uCedarMid);
     if(result != EPDK_OK)
     {
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

```
            __wrn("Try to uninstall cedar.mod failed!\n");
        }
        else
        {
            uCedarMid = 0;
        }
    }
```

33.4.3. MOpen

- **PROTOTYPE**

```
__mp  *esMODS_MOpen(__u8 mid, __u8 mode);
```

- **ARGUMENTS**

mid　　要打开的模块的 ID 号;

mode　　打开模块的模式;

- **RETURNS**

如果打开模块成功，返回模块句柄；否则，返回 NULL。

- **DESCRIPTION**

该函数用于打开模块，获取模块句柄，模块的其它所有操作接口都是基于该模块句柄来实现的，该函数会调用被打开模块的 MOpen 函数。模块句柄由被打开的模块返回，模块定义句柄的数据结构时，第一个 word 必须是模块的 ID 号，并在 MOpen 函数中，将模块的 ID 号保存到模块句柄的结构中。模块一般只允许打开一次，是否允许多次被多次打开，由模块自行处理。

- **DEMO**

```
//打开 cedar 模块
__mp     *hCedar;
hCedar = esMODS_MOpen(uCedarMid, 0);
if(hCedar == NULL)
{
    __wrn("Try to open cedar failed!\n");
}
```

33.4.4. MClose

- **PROTOTYPE**

```
__s32 esMODS_MClose(__mp *mp);
```

- **ARGUMENTS**

mp　　待关闭的模块句柄;

- **RETURNS**

关闭模块句柄的结果:

　　EPDK_OK，　　　关闭模块成功;

　　EPDK_FAIL，　关闭模块失败;

- **DESCRIPTION**

该函数用于关闭模块，该函数会调用被关闭模块的 MClose 函数。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

➢ **DEMO**

```
//关闭 cedar 模块
__s32    result;

if(hCedar)
{
    result = esMODS_MClose(hCedar);
    if(result == EPDK_OK)
    {
        hCedar = NULL;
    }
    else
    {
        __wrn("Try to close cedar failed!\n");
    }
}
```

33.4.5. MRead

➢ **PROTOTYPE**

```
__u32 esMODS_MRead(void *pdata, __u32 size, __u32 n, __mp *mp);
```

➢ **ARGUMENTS**

pdata　存放从模块读取的数据的缓冲区；
size　从模块读取数据的块大小；
n　从模块读取数据的块数；
mp　要读取的模块的句柄；

➢ **RETURNS**

从模块中实际读取到的数据的块数。

➢ **DESCRIPTION**

该函数用于从模块中读取数据，该函数会调用被访问模块的 MRead 函数。

➢ **DEMO**

```
//从 cedar 模块中读取数据
__u32    tmpCnt;
__u8     tmpBuf[1024*4];
tmpCnt = esMODS_MRead(tmpBuf, 4, 1024, hCedar);
__inf("Actual size of data read from cedar is:%dbytes\n", tmpCnt*4);
```

33.4.6. MWrite

➢ **PROTOTYPE**

```
__u32 esMODS_MWrite(const void *pdata, __u32 size, __u32 n, __mp *mp);
```

➢ **ARGUMENTS**

pdata　存放要写入到模块中的数据的缓冲区；

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

size　　写入到模块的数据的块大小；
n　　　写入到模块的数据的块数；
mp　　　待写入数据的模块的句柄；

➢ **RETURNS**

实际写入到模块中的数据的块数。

➢ **DESCRIPTION**

该函数用于向被访问模块中写入数据，该函数会调用被访问模块的 MWrite 函数。

➢ **DEMO**

```
//向 Cedar 模块写入数据
__u32    tmpCnt;
__u8     tmpBuf[1024*4];
tmpCnt = esMODS_MWrite(tmpBuf, 4, 1024, hCedar);
__inf("Actual size of data write cedar is:%dbytes\n", tmpCnt*4);
```

33.4.7. MIoctrl

➢ **PROTOTYPE**

```
__s32 esMODS_MIoctrl(__mp *mp, __u32 cmd, __s32 aux, void *pbuffer);
```

➢ **ARGUMENTS**

mp　　　待操作的模块的句柄；
cmd　　　模块的操作命令；
aux　　　模块自定义参数；
pbuffer　模块自定义参数；

➢ **RETURNS**

模块自定义结果。

➢ **DESCRIPTION**

该函数用于操作被访问的模块，会调用模块的 Mioctrl 接口，允许模块自由扩展命令集，各命令的参数及返回值由模块自行定义。

➢ **DEMO**

```
//启动 cedar 播放媒体文件
__s32    result;
if(hCedar != NULL)
{
    result = esMODS_MIoctrl(hCedar, CEDAR_CMD_PLAY, 0, 0);
    if(result != EPDK_OK)
    {
        __wrn("Try to start cedar failed!\n");
    }
}
```

33.5. Driver

Melis 系统对驱动的管理方式和模块相同，差别在于模块使用虚拟地址，而驱动使用实地址。模块的地址空间限定为 32Mbyte，驱动一般限制为 1Mbyte，也允许一个驱动占用多个驱动段。设备驱动的访问方式和普通驱动有很大差异，具体请参考《MELIS PROGRAM GUID_Device.doc》。

34. 内核编指南-PIN 管理

34.1. Introduction

34.1.1. Description

Melis 操作系统是由全志科技自主研发的一套精简、高效、易用的实时多任务操作系统，有着完善的内存管理、模块管理、中断管理、时钟管理、设备管理以及功耗管理。开发者可以基于 Melis 操作系统规划自己的方案，自由地开发应用程序和扩展设备驱动。
Melis 内核提供了完整的 PIN 管理机制，用户可针对不同方案灵活的配置 PIN 信息，用户可以以 PIN 组为单位操作 PIN 的相关属性，简单易用。

34.1.2. Purpose

本文档主要讲述 Melis 内核的 PIN 管理相关的编程接口，以期让用户能快速掌握在 Melis 平台开发设备驱动时对于 PIN 的操作使用。

34.1.3. Reference

读者可以先了解一些 PIN 相关的硬件知识作为参考，以加深对 Melis 内核 PIN 管理机制的理解。

34.2. Structure

34.2.1. PIN Status Structure

- **PROTOTYPE**

```
typedef struct
{
    char gpio_name[32];
    int  port;
    int  port_num;
    int  mul_sel;
    int  pull;
    int  drv_level;
    int  data;
} user_gpio_set_t;
```

- **MEMBERS**

pin_name	代表当前 PIN 的名称

port	PIN 使用的端口号
port_num	PIN 在当前端口的序号
mul_sel	PIN 的功能选择
pull	PIN 的内置电阻状态
drv_level	PIN 的驱动能力
data	PIN 的电平

- **DESCRIPTION**

user_gpio_set_t 用于描述一个 PIN 的属性信息。

34.3. PIN Group

Melis 平台上对同一逻辑功能的一组 PIN 抽象为一个 PIN 组，用户对于 PIN 的申请、释放等操作均以 PIN 组为基本单位，同时用户也可对单独一个 PIN 进行相关操作。

34.3.1. PIN Group Request

- **PROTOTYPE**

```
__hdle esPINS_PinGrpReq(user_gpio_set_t *pGrpStat, __u32 GrpSize);
```

- **ARGUMENTS**

pGrpStat　数据地址，保存 PIN 属性，来自于配置脚本或者用户自定义；
GrpSize　用户保存 GPIO 数据的结构体的个数；

- **RETURNS**

如果申请成功，返回 PIN 组操作的有效句柄，否则，返回 NULL；

- **DESCRIPTION**

该函数用于申请一组 PIN，PIN 的属性信息可通过配置系统获取，也可以由用户自定义；函数调用成功后，设置的 PIN 组属性会立即生效。

- **DEMO**

```
//通过系统配置信息获取 PIN 属性申请 PIN 组
    __hdle          hPin;
    user_gpio_set_t  PinStat[2];
    if (esCFG_GetGPIOSecData("twi_para", (void *)(&PinStat), 2))
    {
        __wrn("Get pins config information failed\n");
        return ;
    }
    hPin = esPINS_PinGrpReq(&PinStat, 2);
    if (hPin == NULL)
    {
        __wrn("Request twi pins failed\n");
        return ;
    }
    __wrn("Request twi pins succeeded\n");
```

```
//用户自定义属性申请PIN组
    __hdle          hPin;
    user_gpio_set_t  PinStat[1];

    //initialize pin status information
    PinStat.gpio_name = "twi_scl";
    PinStat.port      = 2;
    PinStat.port_num  = 0;
    PinStat.mul_sel   = 2;
    PinStat.pull      = 1;
    PinStat.drv_level = 1;
    PinStat.data      = 1;

    hPin = esPINS_PinGrpReq(&PinStat, 1);
    if (hPin == NULL)
    {
        __wrn("Request twi clock pin failed\n");
        return ;
    }
    __wrn("Request twi clock pin succeeded\n");
```

34.3.2. PIN Group Release

- **PROTOTYPE**

```
__s32  esPINS_PinGrpRel(__hdle hPin, __bool bRestore);
```

- **ARGUMENTS**

hPin 待关闭的模块时钟的句柄;

bRestore PIN 组释放后是否恢复到默认状态;

- **RETURNS**

释放 PIN 组的结果:

EPDK_OK，　释放 PIN 组成功;

EPDK_FAIL，　释放 PIN 组失败;

- **DESCRIPTION**

该函数用于释放给定的 PIN 组，释放 PIN 组句柄，只是软件管理层面上的资源释放，只有 PIN 组释放以后，其它用户才能再次申请该 PIN 组。

- **DEMO**

```
//释放PIN组
    if (esPINS_DevPinsRel(hPin, 1) != EPDK_OK)
    {
        __wrn("release pin group failed\n");
        return ;
    }
    __inf("release pin group succeeded\n");
```

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

34.4. PIN Status

Melis 平台提供获取或设置 PIN 属性的操作接口，用户可同时配置或获取一组 PIN 的属性信息，也可设置 PIN 的一个具体属性，便于设备驱动灵活操作 PIN 相关的属性。

34.4.1. Get PIN Group Status

- **PROTOTYPE**

```
__s32 esPINS_GetPinGrpStat(__hdle hPin, user_gpio_set_t *pGrpStat, __u32 GrpSize, __bool bFromHW);
```

- **ARGUMENTS**

hPin　　　待获取属性的 PIN 组句柄；
pGrpStat　　用于存放获取的 PIN 组属性首地址，大小由 GrpSize 指定；
GrpSize　　PIN 组属性数据结构的个数；
bFromHW 是否获取当前实际硬件的属性信息；

- **RETURNS**

获取 PIN 组属性的结果：

EPDK_OK，　　获取 PIN 组属性*成功；*
EPDK_FAIL，　获取 PIN 组属性*失败；*

- **DESCRIPTION**

该函数用于获取一个 PIN 组的所有 PIN 的属性信息，获取的 PIN 属性可以是当前实际的硬件属性信息，也可以用户通过配置系统设置的属性信息。

- **DEMO**

```
//获取 PIN 组属性信息
    user_gpio_set_t HWPinStat[2];
    if (esPINS_GetPinGrpStat(hPin, HWPinStat, 2, 1) != EPDK_OK)
    {
        __wrn("get pin group hardware status failed\n");
        return ;
    }
    __inf("get pin group hardware status succeeded\n");
```

34.4.2. Get PIN Status

- **PROTOTYPE**

```
__s32 esPINS_GetPinStat(__hdle hPin, user_gpio_set_t *pPinStat, const char *pPinName, __bool bFromHW);
```

- **ARGUMENTS**

hPin　　　待获取属性的 PIN 组句柄；
pGrpStat　　用于存放获取的 PIN 属性信息；
pPinName　　PIN 名称；
bFromHW 是否获取当前实际硬件的属性信息；

➢ **RETURNS**

获取 PIN 属性的结果：

EPDK_OK， 获取 PIN 属性*成功*；

EPDK_FAIL， 获取 PIN 属性*失败*；

➢ **DESCRIPTION**

该函数用于获取一个 PIN 组中的一个 PIN 的属性信息，获取的 PIN 属性可以是当前实际的硬件属性信息，也可以用户通过配置系统设置的属性信息。PIN 名称可以来源于配置系统，也可以由用户自己定义，但是必须与 PIN 申请时的传入的 PIN 名称一致。

➢ **DEMO**

```
//获取 PIN 组中一个 PIN 的属性信息
    user_gpio_set_t HWPinStat[1];
    if (esPINS_GetPinStat(hPin, HWPinStat, "twi_scl", 1) != EPDK_OK)
    {
        __wrn("get twi clock pin hardware status failed\n");
        return ;
    }
    __inf("get twi clock pin hardware status succeeded\n");
```

34.4.3. Set PIN Status

➢ **PROTOTYPE**

```
__s32  esPINS_SetPinStat(__hdle hPin, user_gpio_set_t *pPinStat,
                        const char *pPinName, __bool bSetUserStat);
```

➢ **ARGUMENTS**

hPin 待设置属性的 PIN 组句柄；

pGrpStat 用于存放设置的 PIN 属性信息；

pPinName PIN 名称；

bSetUserStat 是否设置用户输入的 PIN 属性信息；

➢ **RETURNS**

设置 PIN 属性的结果：

EPDK_OK， 设置 PIN 属性*成功*；

EPDK_FAIL， 设置 PIN 属性*失败*；

➢ **DESCRIPTION**

该函数用于设置一个 PIN 组中的一个 PIN 的属性信息，设置的 PIN 属性信息可以由用户自定义，也可以设置为配置系统中配置的属性信息。

➢ **DEMO**

```
//设置 PIN 组中的一个 PIN 属性信息
    user_gpio_set_t HWPinStat[1];

    //initialize pin status information
    HWPinStat.port        = 2;      //PIOC0,C 组 PIN 脚
    HWPinStat.port_num    = 0;      //PIOC0,PIN 在 C 组内偏移为 0
    HWPinStat.mul_sel     = 2;      //PIN 功能选择，根据具体 PIN 设定
    HWPinStat.pull        = PIN_PULL_UP;          //设置为上拉
```

```
        HWPinStat.drv_level = PIN_MULTI_DRIVING_1;   //驱动力为LEVEL1
        HWPinStat.data      = PIN_DATA_HIGH;         //DATA 位为高

        if (esPINS_SetPinStat(hPin, HWPinStat, "twi_scl", 1) != EPDK_OK)
        {
            __wrn("set twi clock pin hardware status failed\n");
            return ;
        }
        __inf("set twi clock pin hardware status succeeded\n");
```

34.4.4. Set PIN IO

- **PROTOTYPE**

```
__s32  esPINS_SetPinIO(__hdle hPin, __bool bOut, const char *pPinName);
```

- **ARGUMENTS**

hPin　　　待设置属性的 PIN 组句柄；
bOut　　　PIN IO 属性是否设置为输出；
pPinName　　PIN 名称；

- **RETURNS**

设置 PIN IO 属性的结果：
　　EPDK_OK，　　　设置 PIN IO 属性*成功*；
　　EPDK_FAIL，　设置 PIN IO 属性*失败*；

- **DESCRIPTION**

该函数用于设置一个 PIN 组中的一个 PIN 的 IO 属性。

- **DEMO**

34.4.5. Set PIN Pull

- **PROTOTYPE**

```
__s32  esPINS_SetPinPull(__hdle hPin, __u32 PullStat, const char *pPinName);
```

- **ARGUMENTS**

hPin　　　待设置属性的 PIN 组句柄；
PullStat　　PIN 内部电阻状态属性；
　　内部电阻状态：
　　PIN_PULL_DISABLE　　高阻态
　　PIN_PULL_UP　　　　上拉
　　PIN_PULL_DOWN　　　下拉
pPinName　　PIN 名称；

- **RETURNS**

设置 PIN IO 属性的结果：
　　EPDK_OK，　　　设置 PIN IO 属性*成功*；
　　EPDK_FAIL，　设置 PIN IO 属性*失败*；

- **DESCRIPTION**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

该函数用于设置一个 PIN 组中的一个 PIN 的内部电阻状态属性。

- **DEMO**

34.4.6. Set PIN Driver

- **PROTOTYPE**

```
__s32  esPINS_SetPinDrive(__hdle hPin, __u32 DriveLevel, const char *pPinName);
```

- **ARGUMENTS**

hPin　　　　待设置属性的 PIN 组句柄；
DriveLevel　设置的 PIN 驱动能力等级；
　　　　　　PIN 驱动能力级别：
　　　　　　PIN_MULTI_DRIVING_0 LEVEL0 驱动能力
　　　　　　PIN_MULTI_DRIVING_1 LEVEL1 驱动能力
　　　　　　PIN_MULTI_DRIVING_2 LEVEL2 驱动能力
　　　　　　PIN_MULTI_DRIVING_3 LEVEL3 驱动能力
pPinName　　PIN 名称；

- **RETURNS**

设置 PIN IO 属性的结果：
EPDK_OK，　　　设置 PIN IO 属性*成功*；
EPDK_FAIL，　设置 PIN IO 属性*失败*；

- **DESCRIPTION**

该函数用于设置一个 PIN 组中的一个 PIN 的驱动能力属性。驱动能力表示 PIN 在驱动外部设备时候的能力。驱动能力越高，表示在推动电平变化的能力越强，波形上会看到从低到高或者从高到低变化很陡峭，但是会出现过冲；驱动能力等级低，波形上看到电平变化平缓，不会出现过冲，但是变化的时间会比较长。通常，使用驱动能力 1 就能满足要求，某些苛刻的设备可能需要其他的驱动能力等级。

- **DEMO**

34.5. PIN Data

Melis 系统提供对 PIN 端口电平状态的操作接口，用户可获取和设置一个 PIN 的端口电平状态，对于端口电平的设置或获取，系统只支持对单独一个 PIN 的操作，不支持同时操作多个 PIN 的端口电平。

34.5.1. Read PIN Data

- **PROTOTYPE**

```
    __s32  esPINS_ReadPinData(__hdle hPin, const char *pPinName);
```

- **ARGUMENTS**

hPin　　　　待读取电平状态的 PIN 组句柄；
pPinName　　PIN 名称；

- **RETURNS**

指定 PIN 的端口电平状态，-1 表示读取指定 PIN 的端口电平状态失败。

- **DESCRIPTION**

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

该函数用于读取一个PIN的端口电平状态，所获取端口电平状态PIN的IO属性必须为输入。

- **DEMO**

34.5.2. Write PIN Data

- **PROTOTYPE**

```
__s32 esPINS_WritePinData(__hdle hPin, __u32 Value, const char *pPinName);
```

- **ARGUMENTS**

hPin　　　待写入电平状态的PIN组句柄；
Value　　　待写入的端口电平值，0代表低电平，1代表高电平；
pPinName　　PIN名称；

- **RETURNS**

设置PIN端口电平的结果：

EPDK_OK，　　设置PIN端口电平*成功*；
EPDK_FAIL，　设置PIN端口电平*失败*；

- **DESCRIPTION**

该函数用于设置一个PIN的端口电平状态，0代表低电平，1代表高电平，所设置端口电平状态PIN的IO属性必须为输出。

- **DEMO**

34.6. PIN Config

由于不同的方案板对于PIN的使用可能会有所不同，为了适应用户的需求，MELIS系统对于可配置PIN的使用操作不再固定，而是通过配置系统获取PIN的属性信息。这样用户不同方案不再需要修改源代码，仅需要修改PIN的配置属性即可，方便用户使用，降低方案开发的难度。

PIN配置属性信息是使用类似标准INI文件格式，一个PIN的配置项信息如下：

```
[pinGrpName]
pinName = port:P[A-J]<CFG><PULL><DRV_LEVEL><DATA>
```

PIN的配置项说明：
段(pinGrpName)表示一个PIN组名称；
键(pinName) 表示代表PIN的名称；
值(port:P[A-J]<CFG><PULL><DRV_LEVEL><DATA>)表示PIN的属性信息；

PIN的属性信息配置项说明：
[A-J]表示组信息；
CFG表示PIN的复用属性；
PULL表示内部电阻状态；
DRV_LEVEL表示驱动力；
DATA表示PIN的端口电平。

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved

34.7. PIN INT

全志科技版权所有，侵权必究
Copyright © 2018 by Allwinner. All rights reserved