



# **CV180X & CV181X**

## **ipcamera 双系统**

## **快启 & 降 Loading**

## **使用手册**

Version: v1.0

Release date: 2024/01/31

©2022 北京晶视智能科技有限公司  
本文件所含信息归北京晶视智能科技有限公司所有。  
未经授权，严禁全部或部分复制或披露该等信息。

# 目录

<b>1</b>	<b>声明</b>	<b>2</b>
<b>2</b>	<b>概述</b>	<b>3</b>
2.1	ipcamera 仓库代码结构说明 . . . . .	4
2.2	小核应用入口说明 . . . . .	5
<b>3</b>	<b>快启</b>	<b>9</b>
3.1	环境搭建 . . . . .	9
3.1.1	构建小核 Alios 编译环境 . . . . .	9
3.1.2	大核快启菜单配置说明 . . . . .	10
3.1.3	小核快启菜单配置说明 . . . . .	12
3.1.4	小核切 pinmux . . . . .	13
3.1.5	小核 Sensor 配置 . . . . .	13
3.2	ipcamera 快启应用编译 . . . . .	16
3.2.1	模块宏定义开关配置 . . . . .	16
3.2.2	ipcamera 快启应用编译方式 . . . . .	17
3.3	ipcamera 快启应用开发 . . . . .	17
3.3.1	ipcamera 应用参数解析过程 . . . . .	18
3.3.2	小核非 Bin 方式参数解析过程 . . . . .	20
3.3.3	小核 Bin 方式参数解析过程 . . . . .	22
<b>4</b>	<b>降 Loading</b>	<b>23</b>

修订记录

版本	日期	功能描述
V1.0	2024/01/31	初稿

# 1 声明



## 法律声明

本数据手册包含北京晶视智能科技有限公司（下称“晶视智能”）的保密信息。未经授权，禁止使用或披露本数据手册中包含的信息。如您未经授权披露全部或部分保密信息，导致晶视智能遭受任何损失或损害，您应对因之产生的损失/损害承担责任。

本文件内信息如有更改，恕不另行通知。晶视智能不对使用或依赖本文件所含信息承担任何责任。本数据手册和本文件所含的所有信息均按“原样”提供，无任何明示、暗示、法定或其他形式的保证。晶视智能特别声明未做任何适销性、非侵权性和特定用途适用性的默示保证，亦对本数据手册所使用、包含或提供的任何第三方的软件不提供任何保证；用户同意仅向该第三方寻求与此相关的任何保证索赔。此外，晶视智能亦不对任何其根据用户规格或符合特定标准或公开讨论而制作的可交付成果承担责任。

## 联系我们

**地址** 北京市海淀区丰豪东路 9 号院中关村集成电路设计园（ICPARK）1 号楼

深圳市宝安区福海街道展城社区会展湾云岸广场 T10 栋

**电话** +86-10-57590723 +86-10-57590724

**邮编** 100094（北京）518100（深圳）

**官方网站** <https://www.sophgo.com/>

**技术论坛** <https://developer.sophgo.com/forum/index.html>

# 2 概述

---

此文件说明 Cvitek IPC 双系统快启动 Turnkey 应用开发流程，旨在为客户提供快速上手文档，提高开发效率。

## 2.1 ipcamera 仓库代码结构说明

└─ build	----编译系统
└─ base.mk	----工具链定义及应用链接到的TARGET-LIB
└─ build.mk	----源码编译过程及应用链接过程
└─ config.mk	----需要用到的SDK INC路径及LIB路径及DEF宏定义
└─ module_config	----勾选 ipcam_dual_fast 应用以及 ipcam_dual 应用需要用到的模块
└─ panel.mk	----MIPI panel宏定义
└─ sensor.mk	----Sensor type宏定义
└─ Makefile	----Makefile入口
└─ modules	----模块文件夹
└─ ai	----AI模块代码，包括MD、FD、PD等
└─ audio	----Audio模块代码
└─ common	----common模块代码，包含linklist，时间戳，json解析，数据压缩
└─ cvi_uac	----UAC模块代码
└─ cvi_uvc	----UVC模块代码
└─ file_recover	----录像恢复模块代码
└─ Makefile	----模块Makefile入口，选择编译哪些模块代码
└─ msg	----双核通信模块代码
└─ network	----网络模块代码
└─ osd	----OSDC模块代码
└─ ota	----网络OTA模块代码
└─ param	----多媒体各模块参数结构体定义
└─ paramparse	----暂无使用
└─ peripheral	----外设模块代码
└─ record	----录像模块代码
└─ ringbuffer	----ringbuffer模块代码
└─ vdec	----视频解码模块代码
└─ video	----多媒体模块代码（包含sys、vi、vpss、venc、rtsp、dump）
└─ vo	----视频输出模块代码
└─ prebuilt	----预编译文件夹
└─ cvitek	----libcvilink 和 libmsg
└─ ffmpeg	----ffmpeg预编译库
└─ jpegturbo	----jpegturbo预编译库
└─ libwebsockets	----websockets预编译库
└─ openssl	----openssl预编译库
└─ rtsp	----rtsp预编译库
└─ thttpd	----thttpd预编译库
└─ README.md	
└─ resource	----资源文件夹
└─ ai_models	----AI模型
└─ bitmap	----老虎图片
└─ file_recover	----H264, H265录像恢复档案
└─ parameter	----应用pipeline参数ini
└─ pipeline	----应用pipeline流程图
└─ www	----web html及js文件
└─ solutions	----应用方案入口
└─ ipcam_dual	----IPC双系统降Loading入口
└─ ipcam_dual_fast	----IPC双系统快启入口

## 2.2 小核应用入口说明

小核启动后会执行应用中的 main 函数

应用入口位于：cvi\_alios/solutions/<PROJECT>/application/app\_main.c

其源码如下所示：

```
int main(int argc, char *argv[])
{
    //board pinmux init
    PLATFORM_IoInit();

    CVI_IPCM_SetRtosSysBootStat();

    YOC_SYSTEM_Init();

    //cli and ulog init
    YOC_SYSTEM_ToolInit();

    //Fs init
    //YOC_SYSTEM_FsVfsInit();
    //load cfg
    PARAM_LoadCfg();
    //video driver init
    MEDIA_VIDEO_SysInit();
#ifdef CONFIG_AUD_DRV_SEL
    //audio driver init
    MEDIA_AUDIO_Init();
#endif
    ipcm_driver_test_init();

    CVI_MSG_Init();

    anon_test_init();
#ifdef CONFIG_RTOS_PRASE_PARAM
    APP_PARAM_Parse();
#endif
#ifdef CONFIG_FAST_BOOT_MODE
    //custom_evenet_pre
    //media video
    MEDIA_VIDEO_Init(0);
#endif

    #if 0
    //network
    #if (CONFIG_APP_ETHERNET_SUPPORT == 1)
    ethernet_init();
    #endif
    #if (CONFIG_APP_HI3861_WIFI_SUPPORT == 1)
    APP_WifiInit();
    #endif
    //cli and ulog init
    // YOC_SYSTEM_ToolInit();
    #if (CONFIG_PQTOOL_SUPPORT == 1)
```

(下页继续)

(续上页)

```

usleep(1000);
isp_daemon2_init(5566);
#endif
LOGI(TAG, "app start.....\n");
APP_CustomEventStart();
#endif
while (1) {
    aos_msleep(3000);
};
}

```

**main()** 函数中主要功能介绍如下：

- **PLATFORM\_IoInit():**

- 主要完成切 pinmux 功能，其调用关系如下：

```

--> PLATFORM_IoInit()
--> _UartPinmux()
--> _MipiRxPinmux()
--> _MipiTxPinmux()
--> _SensorPinmux()
--> JTAG_PinmuxIn()

```

- **CVI\_IPCM\_SetRtosSysBootStat():**

- **YOC\_SYSTEM\_Init():**

- 主要完成了串口初始化
- 其调用关系如下：

```

--> YOC_SYSTEM_Init()
--> cxx_system_init()
--> board_init()
--> board_pinmux_config() # 无效内容
--> stduart_init()
--> console_init(1, 115200, 512)

```

- 上电后小核串口打印 ###YOC###（证明小核启动完毕）

- **YOC\_SYSTEM\_ToolInit():**

- 主要完成 aos\_cli\_init() 初始化 Command Line Interface 模块
- 以及向命令行界面注册调试命令，比如 dumpsys 和 cpuusage 等

- **PARAM\_LoadCfg():**

- 最终的每一个 PARAM\_GET\_XXX\_CFG() 对应的就是 cvi\_alios/solutions/fastboot/customization/turnkey/cv181xc\_evb\_qfn\_param/custom\_xxxparam.c 中的参数结构体
- 其调用关系如下：

```

--> PARAM_LoadCfg()
--> PARAM_INIT_MANAGER_CFG()

```

(下页继续)



(续上页)

```
--> PARAM_SET_MANAGER_CFG_PIPE(0)
--> PARAM_GET_SYS_CFG()
--> PARAM_GET_VI_CFG()
--> PARAM_GET_VPSS_CFG()
--> PARAM_GET_VENC_CFG()
--> PARAM_GET_VO_CFG()
```

- **MEDIA\_VIDEO\_SysInit():**

- 主要完成了多媒体里面各设备驱动的初始化
- 其调用关系如下:

```
-- > MEDIA_VIDEO_SysInit()
--> sys_core_init()
--> cvi_cif_init()
--> cvi_snsr_i2c_probe()
--> vi_core_init()
--> vo_core_init()
--> rgn_core_init()
--> cvi_ldc_probe()
--> mipi_tx_probe()
```

- **MEDIA\_AUDIO\_Init():**

- 主要完成了音频设备驱动的初始化
- 其调用关系如下:

```
-- > MEDIA_AUDIO_Init()
--> _Snd_DriverRegister()
--> snd_card_register()
--> driver_register()
```

- **ipcm\_driver\_test\_init():**

- **CVI\_MSG\_Init():**

- 此步骤初始化 Linux 端核间通信模块

- **anon\_test\_init():**

- **APP\_PARAM\_Parse():**

- 此函数的执行依赖于 CONFIG\_RTOS\_PRASE\_PARAM 宏定义
- CONFIG\_RTOS\_PRASE\_PARAM 该宏定义需要通过 menuconfig 打开

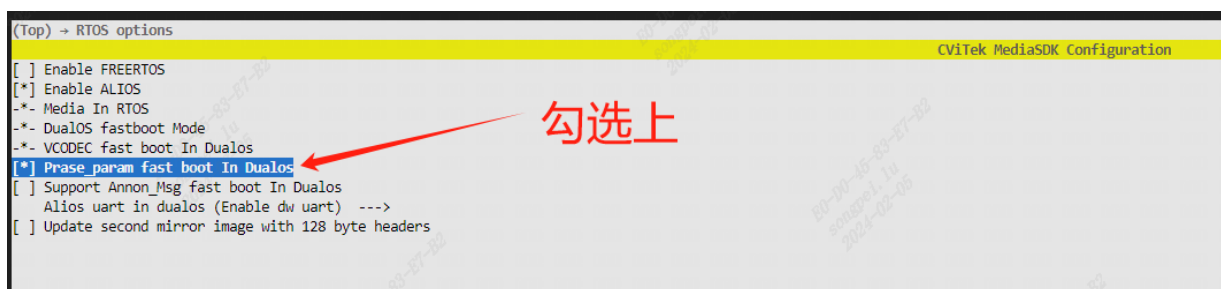


图 2.1: 设置快速启动参数解析

– 其调用关系如下：

```
--> APP_PARAM_Parse()
--> _param_init()
--> PARAM_GET_MANAGER_CFG() # 继承 g_stManagerCtx 的属性
--> pstParam->pstSysCtx = &stSysCtx; # 设置为新的 sys cfg
--> pstParam->pstViCtx = &stViCtx; # 设置为新的 vi cfg
--> pstParam->pstVpssCfg = NULL;
--> pstParam->pstVencCfg = NULL;
--> pstParam->pstVoCfg = NULL;
--> CVI_IPCM_GetParamBinAddr() # 获取 param.bin 的内存地址
--> parse_SysParam() # 解析 param.bin 中 sys模块的参数，更新到 g_stManagerCtx.
--> parse_ViParam() # 解析 param.bin 中 vi模块的参数，更新到 g_stManagerCtx.
--> APP_Param_PqParmParse()
```

#### · MEDIA\_VIDEO\_Init(0):

– 待前面步骤的参数解析完成后，该函数主要完成多媒体各模块的初始化操作

– 其调用关系如下：

```
--> MEDIA_VIDEO_Init()
--> PARAM_Reinit_RawReplay()
--> _MEDIA_VIDEO_SysVbInit()
--> _MEDIA_VIDEO_ViInit()
--> _MEDIA_VIDEO_VpssInit()
--> _MEDIA_VIDEO_VoInit()
--> _MEDIA_VIDEO_VencInit()
--> APP_AiStart()
```

# 3 快启

## 3.1 环境搭建

拉取双系统代码，vscode 中创建一个 workspace

```
cd <workspace>

git clone ssh://<username>@gerrit-ai.sophgo.vip:29418/cvitek/cvi_manifest.git

./cvi_manifest/cvitek_repo_clone.sh --gitclone cvi_manifest/development/dual_os_fast_boot.xml

git clone ssh://<username>@gerrit-ai.sophgo.vip:29418/cvitek/sophapp.git -b dual_os_ipc_dev
```

### 3.1.1 构建小核 Alios 编译环境

#### 安装 Python3

python3 安装地址如下：[python download link](#)

#### 安装 pip 工具

确保环境本地有 PIP 脚本：[pip download link](#)

安装 pip 执行如下命令：

```
python get-pin.py
```

#### 安装 yoctool 工具

```
sudo pip install yoctools -U
```

- 确认 yoctool 版本号：确保版本为 product 1.0.45 或者以上

```
$ product version
```

(下页继续)

(续上页)

```
v1.0.48  
Sep 21 2023,13:01:53
```

- 确认 yoc 工具版本：确保版本为 yoc tool 版本为 2.0.54 或者以上

```
$ yoc --version  
  
2.0.60
```

### 3.1.2 大核快启菜单配置说明

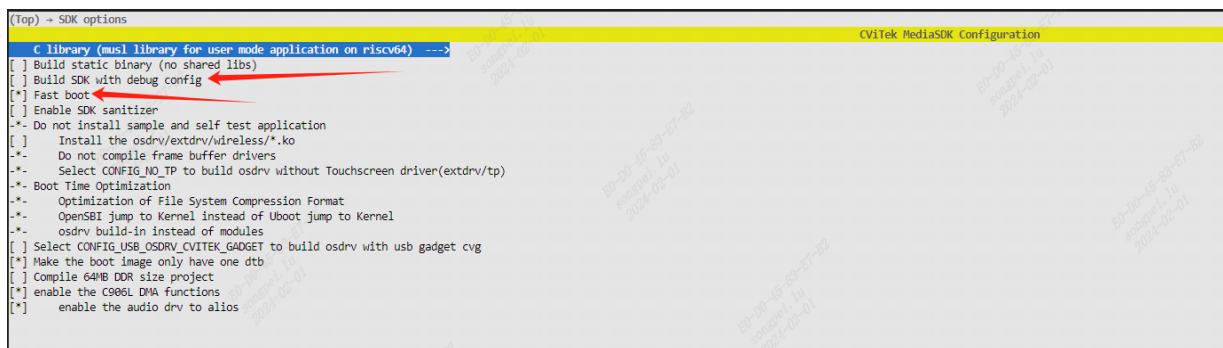


图 3.1: SDK 选项

- **[ ] Build SDK with debug config: 取消勾选**
  - 会选择使用裁剪版的 kernel; kernel size 会变小, log 会变少;
  - 也能增加启动速度; 可以搭配以上开关进行配置;
- **[\*] Fast boot: 确保勾选上**
  - CONFIG\_FASTBOOT 开启后除了会自动打开所有 SDK 级的快启 CONFIG 外, 还会自动打开 Linux 的 CONFIG\_CVITEK\_FASTBOOT 配置项 (Kernel 相关快启 feature 支持)
  - 若关闭 CONFIG\_FASTBOOT, 要使用 Kernel 的相关 feature 要记得在 linux 级的 config 配置中打开 CONFIG\_CVITEK\_FASTBOOT。
  - 此时会选上除了大核压缩的其他所有快启相关选项;

表 3.1: CONFIG\_FASTBOOT 快启总开关 select 的配置参数

CONFIG	描述
CONFIG_FASTBOOT	快启总开关，开启后会自动打开大小核除了 kernel 非压缩的所有快启相关选项
CONFIG_FLASH_SIZE_SHRINK	开启后将不会打包一些 sample 和 self test 应用程序。
CONFIG_NO_FP	不编译 frame buffer 驱动
CONFIG_NO_TP	不编译触摸屏驱动
CONFIG_ROOTFS_FORMAT_OPTIMIZATION	使用 gzip 作为文件系统的压缩算法
CONFIG_SKIP_UBOOT	开启 Opensbi 跳 kernel
CONFIG_OSDRV_BUILD_IN	开启后一些 osdrv 下的驱动将会以 build-in 的形式编译
CONFIG_KERNEL_UNCOMPRESSED	开启后关闭 kernel 压缩

CONFIG\_FASTBOOT 快启总开关对应的 Kconfig 目录如下：

```
# build/Kconfig

config FASTBOOT
bool "Fast boot"
default n
depends on !BUILD_FOR_DEBUG
select FLASH_SIZE_SHRINK
select NO_FB
select NO_TP
select BOOT_TIME_OPTIMIZATION
select ROOTFS_FORMAT_OPTIMIZATION
select SKIP_UBOOT
select OSDRV_BUILD_IN
select MEDIA_IN_RTOS
select FAST_BOOT_MODE
select VCODEC_FIRMWARE_H
# select RTOS_PRAISE_PARAM
# select RTOS_ANNON_MSG
help
    This enables fast boot features.
```

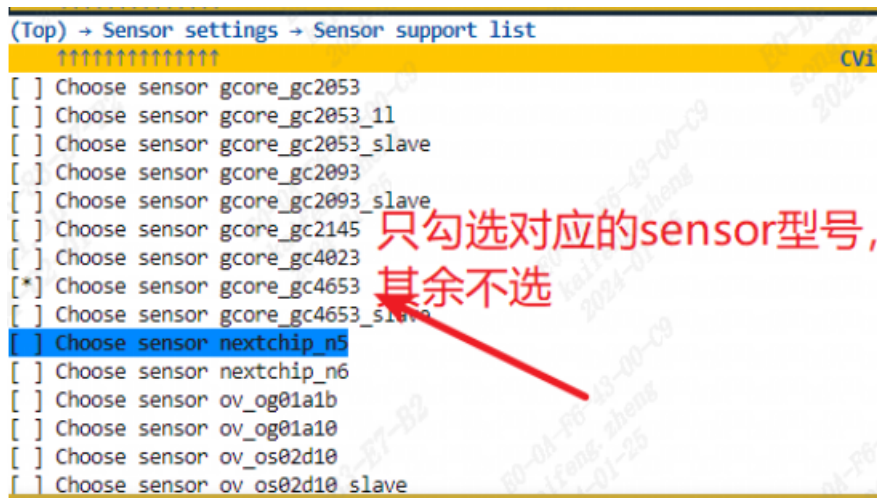


图 3.2: Sensor 选项

只勾选对应的 sensor 型号，其余不选

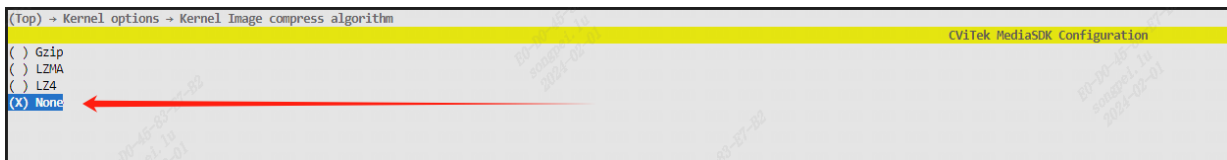


图 3.3: Kernel 选项

关闭 kernel 压缩，减少读 FLASH，Kernel 镜像、参数文件都不使用硬件解压。

### 3.1.3 小核快启菜单配置说明



图 3.4: RTOS 选项

- **[\*] DualOS fastboot Mode：确保勾选上**
  - 用于配置小核是否启动 Media Pipeline 流程，初始化 sensor/VI/VPSS 等多媒体流程；如果是快启方案，建议勾选上。
  - 备注：目前这开关打开后，小核 Alios 会对 sensor 进行初始化，此部分代码是开放的，需要自己适配下对应的 sensor；
- **[\*] VCODEC fast boot In Dualos：确保勾选上**

- 用于配制 venc bin 文件是在小核加载还是在大核加载；
- 如果是快启方案，建议勾选上，把 venc 放到小核来加载；

· **[ ] Prase\_param fast boot In Dualos: 确保没有勾选上**

- 用于控制宏定义 CONFIG\_RTOS\_PRASE\_PARAM
- 如果该选项被勾上，会调用 APP\_PARAM\_Parse(), 参考小核应用入口说明

```
#ifdef CONFIG_RTOS_PRASE_PARAM
    APP_PARAM_Parse();
#endif
```

- 宏定义源码位于: build/Kconfig , 如下所示:

```
config RTOS_PRASE_PARAM
bool "Prase_param fast boot In Dualos"
default n
depends on ENABLE_ALIOS && FAST_BOOT_MODE
help
set fast boot prase param in Dualos.
```

· **[ ] Support Annon\_Msg fast boot In Dualos: 确保没有勾选上**

### 3.1.4 小核切 pinmux

关于 Pinmux 切换更替需要在 cvi\_alios/solutions/fastboot/customization/turnkey/src/custom\_platform.c 中进行对应修改，其调用关系请参考PLATFORM\_IoInit()。

### 3.1.5 小核 Sensor 配置

关于小核 sensor 的配置，需要设置 CONFIG\_SNS0\_TYPE 的值。

该值对应的 package.yaml 位于: cvi\_alios/solutions/fastboot/package\_yamls/package.yaml.turnkey

```

161
162  ## 第五部分：配置信息
163  # def_config: ..... # 组件的可配置项
164  # CONFIG_DEBUG: y
165  # CONFIG_PARAM_NOT_CHECK: y
166  # CONFIG_CLI: y
167  def_config:
168      AOS_COMP_CLI: 1
169      CONFIG_DEBUG: 1
170      CONFIG_DEBUG_MM: 1
171      CONFIG_APP_DEBUG_JTAG: 0
172      CONFIG_APP_TEST: 0
173      CONFIG_EFUSE_TEST: 0
174      CONFIG_SUPPORT_TPU: 0
175      CONFIG_KV_ENABLE_CACHE: 0
176      CONFIG_INIT_TASK_STACK_SIZE: 8192
177      CONFIG_APP_CX_CLOUD_SUPPORT: 0
178      CONFIG_APP_VENC_SUPPORT: 1
179      CONFIG_APP_GUI_SUPPORT: 0
180      CONFIG_APP_SENSOR_IR_USE: 0
181      CONFIG_DEBUG_HOSTMCU_EMU_SUPPORT: 0
182      CONFIG_APP_AI_SUPPORT: 0
183  # UART_MODE_SYNC: 1
184      CONFIG_BOARD_CV181XC: 1
185      CONFIG_BOARD_CV181XH: 0
186  # CONFIG_SD_FATFS_MOUNTPOINT: "/mnt/sd"
187      CLI_CONFIG_STACK_SIZE: 8192
188      #ULOG_CONFIG_POP_FS: 1
189      ULOG_CONFIG_ASYNC: 1 # ulog异步配置项, 1表示开启异步打印, 0表示同步打印
190      ULOG_CONFIG_SYNC_LOG_DETAILS: 1
191      ULOG_CONFIG_ASYNC_BUF_SIZE: 8192
192      ULOG_CONFIG_LOG_SIZE: 512
193      USE_4K_ERASE_SECTION: 0
194      CONFIG_SIMPLE_FONTMOD: 1
195      CONFIG_SUPPORT_NORFLASH: 0
196      CONFIG_SNS0_TYPE: 14
197      CONFIG_SNS1_TYPE: 0
198      CONFIG_PANEL_HX8394: 0
199      CONFIG_PANEL_ILI9488: 0
200      CONFIG_VENC_TEST_CHN: 2
201      CONFIG_POTOL SUPPORT: 0

```

14: GC4653

图 3.5: Sensor 配置

其中 CONFIG\_SNS0\_TYPE 数值是需要参考 `cvi_alios/components/cvi_mmf_sdk/cvi_sensor/sensor_cfg/sensor_cfg.h` 中 SNS\_TYPE\_E 的枚举值

以 GCORE\_GC4653\_MIPI\_4M\_30FPS\_10BIT sensor 为例，其对应的 CONFIG\_SNS0\_TYPE 数值为 14

```

//#if defined(CONFIG_KERNEL_RHINO)
typedef enum SNS_TYPE_E {
    SNS_TYPE_NONE = 0,
    /* ----- LINEAR BEGIN ----- */
    SONY_IMX327_MIPI_2M_30FPS_12BIT,
    SONY_IMX307_MIPI_2M_30FPS_12BIT,
    SONY_IMX307_2L_MIPI_2M_30FPS_12BIT,

```

(下页继续)



(续上页)

```

SONY_IMX307_SLAVE_MIPI_2M_30FPS_12BIT,
GCORE_GC0308_MIPI_1M_30FPS_8BIT,
GCORE_GC1054_MIPI_1M_30FPS_10BIT,
GCORE_GC2053_MIPI_2M_30FPS_10BIT,
GCORE_GC2053_1L_MIPI_2M_30FPS_10BIT,
GCORE_GC2093_MIPI_2M_30FPS_10BIT,
GCORE_GC2145_MIPI_2M_12FPS_8BIT,
GCORE_GC02M1_MIPI_2M_30FPS_10BIT,
GCORE_GC02M1_SLAVE_MIPI_2M_30FPS_10BIT,
GCORE_GC4023_MIPI_4M_30FPS_10BIT,
GCORE_GC4653_MIPI_4M_30FPS_10BIT,
OV_OG01A1B_MIPI_2M_30FPS_10BIT,
OV_OG01A10_MIPI_2M_30FPS_10BIT,
PIXELPLUS_PR2020_1M_25FPS_8BIT,
PIXELPLUS_PR2020_1M_30FPS_8BIT,
PIXELPLUS_PR2020_2M_25FPS_8BIT,
PIXELPLUS_PR2020_2M_30FPS_8BIT,
TECHPOINT_TP9950_1M_30FPS_8BIT,
TECHPOINT_TP9950_2M_30FPS_8BIT,
TECHPOINT_TP9950_1M_25FPS_8BIT,
TECHPOINT_TP9950_2M_25FPS_8BIT,
SMS_SC1336_1L_MIPI_1M_30FPS_10BIT,
SMS_SC1336_1L_MIPI_1M_60FPS_10BIT,
SMS_SC1336_1L_SLAVE_MIPI_1M_30FPS_10BIT,
SMS_SC1336_2L_MIPI_1M_30FPS_10BIT,
SMS_SC1336_2L_MIPI_1M_60FPS_10BIT,
SMS_SC1336_2L_SLAVE_MIPI_1M_30FPS_10BIT,
SMS_SC1346_1L_MIPI_1M_30FPS_10BIT,
SMS_SC1346_1L_MIPI_1M_60FPS_10BIT,
SMS_SC1346_1L_SLAVE_MIPI_1M_30FPS_10BIT,
SMS_SC1346_1L_SLAVE_MIPI_1M_60FPS_10BIT,
SMS_SC2331_1L_MIPI_2M_30FPS_10BIT,
SMS_SC301IOT_MIPI_3M_30FPS_10BIT,
CISTA_C4390_MIPI_4M_30FPS_10BIT,
BYD_BF2253L_MIPI_1200P_30FPS_10BIT,
/* ----- LINEAR END ----- */
SNS_TYPE_LINEAR_BUTT,
/* ----- WDR 2TO1 BEGIN ----- */
SONY_IMX327_MIPI_2M_30FPS_12BIT_WDR2TO1 = SNS_TYPE_LINEAR_BUTT,
SONY_IMX307_MIPI_2M_30FPS_12BIT_WDR2TO1,
SONY_IMX307_2L_MIPI_2M_30FPS_12BIT_WDR2TO1,
SONY_IMX307_SLAVE_MIPI_2M_30FPS_12BIT_WDR2TO1,
GCORE_GC2093_MIPI_2M_30FPS_10BIT_WDR2TO1,
SMS_SC1336_1L_MIPI_1M_30FPS_10BIT_WDR2TO1,
SMS_SC1336_1L_MIPI_1M_60FPS_10BIT_WDR2TO1,
SMS_SC1336_2L_MIPI_1M_30FPS_10BIT_WDR2TO1,
SMS_SC1336_2L_MIPI_1M_60FPS_10BIT_WDR2TO1,
SMS_SC1346_1L_MIPI_1M_30FPS_10BIT_WDR2TO1,
SMS_SC1346_1L_MIPI_1M_60FPS_10BIT_WDR2TO1,
/* ----- WDR 2TO1 END ----- */
SNS_TYPE_WDR_BUTT,
} SNS_TYPE_E;

```

## 3.2 ipcamera 快启应用编译

### 3.2.1 模块宏定义开关配置

参考仓库代码结构说明，ipcamera 快启应用需要编译的模块由客户自定义选择

- 模块编译宏定义开关入口：sophapp/build/module\_config/ipcam\_dual\_fast.mk
- 模块编译宏定义配置如下：

```
CONFIG_MODULE_AI=n          ## 不编译 ai 模块相关代码
CONFIG_MODULE_AI_MD=n       ## 不编译 ai md 模块相关代码
CONFIG_MODULE_AI_PD=n       ## 不编译 ai pd 模块相关代码
CONFIG_MODULE_AI_FD_FACE=n  ## 不编译 ai fd 模块相关代码
CONFIG_MODULE_AI_IRFAECE=n  ## 不编译 ai ir 模块相关代码
CONFIG_MODULE_AI_BABYCRY=n  ## 不编译 ai cry 模块相关代码
CONFIG_MODULE_AUDIO=n       ## 不编译 audio 模块相关代码
CONFIG_MODULE_VIDEODEC=n    ## 不编译 vdec解码 模块相关代码
CONFIG_MODULE_AUDIO_MP3=n   ## 不编译 mp3 模块相关代码
CONFIG_MODULE_MSG=y         ## 编译 双核通信 模块相关代码
CONFIG_MODULE_NETWORK=n     ## 不编译 web 模块相关代码
CONFIG_MODULE_OSD=y         ## 编译 osdc 模块相关代码
CONFIG_MODULE_OTA=n         ## 不编译 ota 模块相关代码
CONFIG_MODULE_PARAMPARSE=y  ## 编译 paramparse 模块相关代码
CONFIG_MODULE_RECORD=n      ## 不编译 录像 模块相关代码
CONFIG_MODULE_RINGBUF=n     ## 不编译 ringbuffer 模块相关代码
CONFIG_MODULE_VIDEO=y       ## 编译 video_
↪模块 (sys、vi、vpss、venc) 相关代码
CONFIG_MODULE_COMMON=y      ## 编译 common_
↪模块 (linklist、printf) 相关代码
CONFIG_MODULE_CVIUAC=n      ## 不编译 uac 模块相关代码
CONFIG_MODULE_CVIUVC=n      ## 不编译 uvc 模块相关代码
CONFIG_MODULE_PERIPHERAL=y  ## 编译 外设 模块相关代码
CONFIG_MODULE_PARAM=y       ## 编译 多媒体结构体参数 模块相关代码
#PERIPHERAL SUB
CONFIG_MODULE_GPIO=y        ## 编译 外设gpio 模块相关代码
CONFIG_MODULE_ADC=n         ## 不编译 外设ADC 模块相关代码
CONFIG_MODULE_IRCUT=y       ## 编译 外设ircut 模块相关代码
CONFIG_MODULE_PWM=y         ## 编译 外设pwm 模块相关代码
CONFIG_MODULE_MIPI_TX=n     ## 不编译 外设mipi-tx 模块相关代码
CONFIG_MODULE_VO=n          ## 编译 VO 模块相关代码
#extern function
CONFIG_MODULE_RTSP=y        ## 编译 rtsp 模块相关代码
CONFIG_MODULE_PQTOOL=n      ## 不编译 pqtool 模块相关代码
CONFIG_PROJECT_ENABLE_FASTBOOT=n
CONFIG_PROJECT_ENABLE_ANONMSG=y
CONFIG_PROJECT_MULTI_PROCESS_SUPPORT=n
CONFIG_PROJECT_STATIC=y
CONFIG_PROJECT_CAPTURE_STARTUP_IMAGE=y
CONFIG_DUAL_OS=y            ## 双系统宏定义
CONFIG_IPCAM_DUAL_FAST=y    ## ipcamera快启应用宏定义

# Add board config
CONFIG_MODULE_CV1810C_IMX307_CONFIG = n    ## 不选择
```

(下页继续)

(续上页)

```

CONFIG_MODULE_CV1810C_GC4653_CONFIG = y    ## 仅解析 sophapp/
↪modules/param/config/cv1810c_config_gc4653 的多媒体参数结构体
CONFIG_MODULE_CV180ZB_GC2053_CONFIG = n    ## 不选择
CONFIG_MODULE_CV180ZB_SC2331_1L_CONFIG = n  ## 不选择
CONFIG_MODULE_CV1800B_GC4653_CONFIG = n    ## 不选择
CONFIG_MODULE_CV1810C_GC4653_PANEL_CONFIG = n    ## ↪
↪不选择
CONFIG_MODULE_CV1812CP_SC035HGS_1L_CONFIG = n    ## ↪
↪不选择
CONFIG_MODULE_CV1812CP_SC035HGS_1L_SC035HGS_1L_CONFIG = n #
↪# 不选择
CONFIG_MODULE_CV180ZB_SC2336_SC2336_CONFIG = n    ## ↪
↪不选择
CONFIG_MODULE_CV180ZB_SC2331_SC2331_1L_CONFIG = n    ## ↪
↪不选择

include $(SRCTREE)/build/config.mk

```

### 3.2.2 ipcamera 快启应用编译方式

参考 README.md 或者输入命令如下：

```

make ipcam_dual_fast clean;
make ipcam_dual_fast;
make ipcam_dual_fast install;

```

## 3.3 ipcamera 快启应用开发

ipcamera 快启应用程序的 main () 函数

- **main()** 程序入口位于：sophapp/solutions/ipcam\_dual\_fast/src/app\_ipcam\_main.c
- **main()** 函数调用关系如下所示：

```

--> main
--> app_ipcam_Param_Load()
--> app_ipcam_Init()
--> app_ipcam_Venc_Start(APP_VENC_ALL)
--> app_ipcam_JpgCapFlag_Set(CVI_TRUE)
--> app_ipcam_mipi_tx_Init()
--> app_ipcam_Vo_Init()
--> app_ipcam_Vdec_Init()
--> app_ipcam_Vdec_Start()
--> app_ipcam_Audio_Init()
--> app_ipcam_Rtsp_Server_Create()
--> app_ipcam_Ai_PD_Start()
--> app_ipcam_Ai_MD_Start()
--> app_ipcam_Ai_FD_Start()

```

(下页继续)

(续上页)

```
--> app_ipcam_Record_Recover_Init()
--> app_ipcam_Record_Init()
--> app_ipcam_Sys_EnableFastBoot()
--> app_ipcam_Ispd_Load()
```

· 主要的函数介绍：

- **app\_ipcam\_Param\_Load()**: 解析 ipcamera 应用的多媒体各模块参数
- **app\_ipcam\_Init()**: 初始化多媒体各模块
- **app\_ipcam\_Venc\_Start()**: 开启编码通道接收输入图像

#### 注意:

ipcamera 快启应用会将 sys、vi、vpss、venc 以及 vo 五个模块的参数解析以及初始化 init 过程放到小核来完成

对于 ipcamera 快启应用，仅需要考虑从 venc 模块开始，设定好 venc 参数（位于 `sophapp/modules/param/config/cv1810c_config_gc4653/src/custom_vencparam.c`）以及后续模块参数（e.g. rtsp、audio、osdc 等）

对于 ipcamera 快启应用，仅需要考虑从 venc 模块开始，调用剩余模块的初始化函数或者启动函数

以 VENC 为例：

- 小核：负责创建 venc 通道，
  - ipcamera 快启应用：负责调用 venc 启动函数
- ```
app_ipcam_Venc_Start(APP_VENC_ALL)
    - CVI_VENC_StartRecvFrame()：唤醒 venc 硬件接收输入图像，并编码成码流
    - CVI_VENC_GetStream()：获取码流，送到后端剩余模块（e.g. rtsp）
```

### 3.3.1 ipcamera 应用参数解析过程

ipcamera 快启应用本身会有一套参数解析过程，该过程除了解析 sys、vi、vpss、venc 以及 vo 等多媒体模块之外，还解析诸如 osd、audio、vdec、ai、record 和 peripheral 等其他模块参数

- 解析函数名称为：`app_ipcam_Param_Load()`
- 解析函数入口为：`sophapp/solutions/ipcam_dual_fast/src/app_ipcam_param.c`
- **app\_ipcam\_Param\_Load()** 函数调用关系如下：

```
--> app_ipcam_Param_Load()
    --> _Load_Param_Vi(app_ipcam_Vi_Param_Get(), PARAM_GET_VI_
    ↪ CFG())
    --> _Load_Param_Vpss(app_ipcam_Vpss_Param_Get(), PARAM_GET_
    ↪ VPSS_CFG())
```

(下页继续)

(续上页)

```

--> _Load_Param_Venc(app_ipcam_Venc_Param_Get(), PARAM_GET_
↪ VENC_CFG())
--> _Load_Param_Rtsp(app_ipcam_Rtsp_Param_Get(), PARAM_GET_
↪ RTSP_CFG())
--> _Load_Param_Osdc(app_ipcam_Osdc_Param_Get(), PARAM_GET_
↪ OSDC_CFG())
--> _Load_Param_Gpio(app_ipcam_Gpio_Param_Get(), PARAM_GET_
↪ GPIO_CFG())
--> _Load_Param_Vdec(app_ipcam_Vdec_Param_Get(), PARAM_GET_
↪ VDEC_CFG())
--> _Load_Param_Audio(app_ipcam_Audio_Param_Get(), PARAM_GET_
↪ AUDIO_CFG())
--> _Load_Param_Ai_MD(app_ipcam_Ai_MD_Param_Get(), PARAM_
↪ GET_AI_MD_CFG())
--> _Load_Param_Ai_PD(app_ipcam_Ai_PD_Param_Get(), PARAM_GET_
↪ AI_PD_CFG())
--> _Load_Param_MipiTx(app_ipcam_MipiTx_Param_Get(), PARAM_GET_
↪ MIPITX_CFG())
--> _Load_Param_Vo(app_ipcam_VO_Param_Get(), PARAM_GET_VO_
↪ CFG())

```

接下来将会分别介绍 `PARAM_GET_xxx_CFG()` 函数、`app_ipcam_xxx_Param_Get()` 函数和 `_Load_Param_xxx()` 函数的作用

· **PARAM\_GET\_XXX\_CFG()** 函数调用关系如下：

- 该类型函数返回了一个结构体指针（以 `venc` 模块为例）

```

PARAM_VENC_CFG_S * PARAM_GET_VENC_CFG(void) {
    return &g_stVencCfg;
}

```

- 该结构体指针指向了某个模块的参数结构体（以 `venc` 模块为例，该结构体定义了 `venc_chn = 1`）

```

PARAM_VENC_CFG_S g_stVencCfg = {
    .s32VencChnCnt = 1,
    .pstVencChnCfg = PARAM_CLASS(VENCCFG,CTX,VENC),
};

```

- `PARAM_GET_xxx_CFG()` 函数入口位于（以 `cv1810c_config_gc4653` 板卡及 `sensor` 为例）：

```

sophapp/modules/param/config/cv1810c_config_gc4653
├── Makefile
└── src
    ├── custom_aimdparam.c
    ├── custom_aipdparam.c
    ├── custom_audioparam.c
    ├── custom_gpioparam.c
    ├── custom_mipitxparam.c
    ├── custom_osdcparam.c
    ├── custom_param.c
    └── custom_rtspparam.c

```

(下页继续)

(续上页)

```

|—— custom_sysparam.c
|—— custom_vdecparam.c
|—— custom_vencparam.c
|—— custom_viparam.c
|—— custom_voparam.c
|—— custom_vpssparam.c

```

- **app\_ipcam\_xxx\_Param\_Get()** 函数调用关系如下：

- 该类型函数返回了一个全局变量结构体指针（以 venc 模块为例）

```

APP_PARAM_VENC_CTX_S *app_ipcam_Venc_Param_Get(void)
{
    return g_pstVencCtx;
}

```

- 该类型函数得到的全局变量结构体初始化过程如下（以 venc 模块为例）：

```

APP_PARAM_VENC_CTX_S g_stVencCtx, *g_pstVencCtx = &g_stVencCtx;

```

- 该类型函数得到的全局变量结构体定义位于：

```

sophapp/modules/video/src/
|—— app_ipcam_dump.c
|—— app_ipcam_rtsp.c
|—— app_ipcam_sys.c
|—— app_ipcam_venc.c
|—— app_ipcam_vi.c
|—— app_ipcam_vpss.c

```

- **\_Load\_Param\_xxx()** 函数调用关系如下：

- 该类型函数定义位于：sophapp/solutions/ipcam\_dual\_fast/src/app\_ipcam\_param.c
- 该函数主要作用是将特定 pipeline 定义好的参数结构体内容 加载到 pipeline 全局变量结构体 当中

### 3.3.2 小核非 Bin 方式参数解析过程

小核将会通过结构体的方式为 sys、vi、vpss、venc 以及 vo 五个模块传递配置参数，具体调用关系请参考 [PARAM\\_LoadCfg\(\)](#)。

#### 多媒体 Manager 结构体

- 结构体声明：cvi\_alios/solutions/fastboot/customization/include/custom\_param.h
- 该 Manager 结构体囊括了各模块（含 sys、vi、vpss、venc 以及 vo）的结构体

```

typedef struct _PARAM_MANAGER_CFG_S {
    PARAM_SYS_CFG_S *pstSysCtx;
    PARAM_VI_CFG_S *pstViCtx;
    PARAM_VPSS_CFG_S *pstVpssCfg;
    PARAM_VENC_CFG_S *pstVencCfg;
}

```

(下页继续)

(续上页)

```
PARAM_VO_CFG_S *pstVoCfg;
} PARAM_MANAGER_CFG_S;
```

- 结构体定义: `cvi_alios/solutions/fastboot/customization/turnkey/cv181xc_evb_qfn_param/custom_param.h`
- 各多媒体模块结构体初始化为空:

```
PARAM_MANAGER_CFG_S g_stManagerCtx = {
    .pstSysCtx = NULL,
    .pstViCtx = NULL,
    .pstVpssCfg = NULL,
    .pstVoCfg = NULL,
    .pstVencCfg = NULL,
};
```

### 多媒体各模块结构体声明

```
cvi_alios/components/cvi_platform/param/include/
|---- cvi_param.h
|---- param_sys.h
|---- param_venc.h
|---- param_vi.h
|---- param_vo.h
|---- param_vpss.h
```

### 多媒体各模块结构体初始化定义

```
cvi_alios/solutions/fastboot/customization/turnkey/cv181xc_evb_qfn_param
|---- custom_param.c
|---- custom_sysparam.c
|---- custom_vencparam.c
|---- custom_viparam.c
|---- custom_voparam.c
|---- custom_vpssparam.c
```

### 例子说明

以 vpss 模块的结构体为例，可以看到该 pipeline 中会定义 1 个 vpss grp，结构体 `PARAM_VPSS_GRP_CFG_S` 指针指向 1 个 vpss grp 属性；

- 其声明如下：

```
typedef struct _PARAM_VPSS_CFG_S {
    CVI_U8 u8GrpCnt;
    PARAM_VPSS_GRP_CFG_S *pstVpssGrpCfg;
} PARAM_VPSS_CFG_S;
```

- 其定义如下：

```
PARAM_VPSS_CFG_S g_stVpssCtx = {
    .u8GrpCnt = 1,
    .pstVpssGrpCfg = PARAM_CLASS(GRPCFG,CTX,GRP),
};
```



- vpss grp 属性定义如下，可见每个 vpss grp 中又会定义 vpss grp 属性及 vpss chn 属性，比如说该 grp 只会有 1 个 chn:

```
PARAM_CLASSDEFINE(PARAM_VPSS_GRP_CFG_S,GRPCFG,CTX,GRP)[] =
{
{
    .bEnable = 1,
    .VpssGrp = 0,
    .u8ChnCnt = 1,
    .pstChnCfgr = PARAM_CLASS(CHNCFG,GRP0,CHN),
    .u8ViRotation = 0,
    .s32BindVidev = 0,
    .stVpssGrpAttr = {
        .u8VpssDev = 1,
        .u32MaxW = 2560,
        .u32MaxH = 1440,
        .enPixelFormat = PIXEL_FORMAT_NV21,
        .stFrameRate.s32SrcFrameRate = -1,
        .stFrameRate.s32DstFrameRate = -1,
    },
    // .bBindMode = CVI_TRUE,
    // .astChn[0] = {
    //     .enModId = CVI_ID_VI,
    //     .s32DevId = 0,
    //     .s32ChnId = 0,
    // },
    .astChn[1] = {
        .enModId = CVI_ID_VPSS,
        .s32DevId = 0,
        .s32ChnId = 0,
    },
},
};
```

### 3.3.3 小核 Bin 方式参数解析过程

需要在 menuconfig 勾选上 CONFIG\_RTOS\_PRASE\_PARAM 宏定义，请参考小核快启菜单配置说明。



# 4 降 Loading

---

TODO