Rockchip VICAP 开发指南

文件标识：RK-KF-YF-402

发布版本：V1.0.1

日期：2020-05-27

文件密级：□绝密 □秘密 □内部资料 ■公开

福州瑞芯微电子股份有限公司

Fuzhou Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

VICAP（video capture unit）用于对并口数据进行接收，主要支持以下功能：

- BT601 YCbCr 422 8bit input
- BT656 YCbCr 422 8bit input
- YUYV/YVYU/UYVY/VYUY input
- RAW 8/10/12 bit input
- JPEG input
- Window cropping
- Virtual stride when write to DDR
- Different stored address for Y and UV
- YUV 422/420 output
- Configurable for the polarity of pixel_clk, hsync, vsync

产品版本

芯片名称	版本
RK2108	RT-Thread & HAL
PISCES	RT-Thread & HAL
RK2206	RKOS & HAL

读者对象

本文档主要适用以下工程师：

技术支持工程师

软件开发工程师

修订记录

日期	版本	作者	修改说明
2019-07-16	V1.0.0	黄江龙	初版
2020-05-27	V1.0.1	钟勇汪	修正格式

目录

1 HAL的使用

1.1 HAL的说明

HAL层主要是将VICAP控制器的各个具体功能，在寄存器层面封装成单独的函数接口，以供上层的os进行调用。 OS层直接调用HAL层的函数接口以实现目标功能，同时负责HAL层相应寄存器的竟态管理，而HAL不负责，比如进行锁操作。 VICAP各项功能设置详见相对应的芯片手册。

1.2 HAL的主要函数接口

HAL各函数接口按照VICAP的寄存器功能进行展开。

1.2.1 Register: VICAP_DVP_CTRL

```
HAL_Status HAL_VICAP_SetAxiBurstType(struct VICAP_REG *pReg,
eVICAP_axiBurstType type);
HAL_Status HAL_VICAP_SetCaptureEnable(struct VICAP_REG *pReg, bool enable);
HAL_Status HAL_VICAP_SetWorkmode(struct VICAP_REG *pReg, eVICAP_workMode
workMode);
```

1.2.2 Register: VICAP_DVP_INTEN/VICAP_DVP_INTSTAT

中断使能/禁能函数：

```
HAL_Status HAL_VICAP_SetIrqEnable(struct VICAP_REG *pReg);
HAL_Status HAL_VICAP_SetIrqDisable(struct VICAP_REG *pReg);
```

中断状态获取/清零函数：

```
uint32_t HAL_VICAP_GetIrqStatus(struct VICAP_REG *pReg);
HAL_Status HAL_VICAP_ClearIrqStatus(struct VICAP_REG *pReg, uint32_t mask);
```

1.2.3 Register: VICAP_DVP_FOR

VICAP_DVP_FOR寄存器是VICAP控制器重要的寄存器涉及了数据输入输出格式的相关配置，需要仔细确认。

```
HAL_Status HAL_VICAP_SetUvStoreOrder(struct VICAP_REG *pReg,
eVICAP_uvStoreOrder order);
HAL_Status HAL_VICAP_SetRawEnd(struct VICAP_REG *pReg, eVICAP_rawEnd type);
HAL_Status HAL_VICAP_SetOut420Order(struct VICAP_REG *pReg,
eVICAP_out420Order type);
HAL_Status HAL_VICAP_SetOutFormat(struct VICAP_REG *pReg,
eVICAP_outputFormat type);
HAL_Status HAL_VICAP_SetYmodeOnly(struct VICAP_REG *pReg, bool enable);
HAL_Status HAL_VICAP_SetRawWidth(struct VICAP_REG *pReg, eVICAP_rawWidth
width);
HAL_Status HAL_VICAP_SetJpegMode(struct VICAP_REG *pReg, eVICAP_jpegMode
mode);
HAL_Status HAL_VICAP_SetFieldOrder(struct VICAP_REG *pReg, eVICAP_fieldOrder
order);
HAL_Status HAL_VICAP_SetYuvInOrder(struct VICAP_REG *pReg, eVICAP_yuvInOrder
order);
HAL_Status HAL_VICAP_SetInputMode(struct VICAP_REG *pReg, eVICAP_inputMode
mode);
HAL_Status HAL_VICAP_SetHrefVsynPol(struct VICAP_REG *pReg, eVICAP_hrefPol
hpol,  eVICAP_vsyncPol vpol);
```

1.2.4 Register: VICAP_DVP_FRM0_ADDR_Y/VICAP_DVP_FRM0_ADDR_UV

VICAP使用oneframe mode或者frame pingpong mode时需要设置。

```
HAL_Status HAL_VICAP_SetFrm0YAddr(struct VICAP_REG *pReg, uint32_t yAddr);
HAL_Status HAL_VICAP_SetFrm0UvAddr(struct VICAP_REG *pReg, uint32_t uvAddr);
```

1.2.5 Register: VICAP_DVP_FRM1_ADDR_Y/VICAP_DVP_FRM1_ADDR_UV

VICAP使用oneframe mode或frame pingpong mode时需要设置。

```
HAL_Status HAL_VICAP_SetFrm1YAddr(struct VICAP_REG *pReg, uint32_t yAddr);
HAL_Status HAL_VICAP_SetFrm1UvAddr(struct VICAP_REG *pReg, uint32_t uvAddr);
```

1.2.6 Register: VICAP_DVP_VIR_LINE_WIDTH

虚宽的定义是存储图像数据时，相邻两行行数据存储首地址的差值，该寄存器可读可写。

```
HAL_Status HAL_VICAP_SetVirtualLineWidth(struct VICAP_REG *pReg, uint32_t
width);
uint32_t HAL_VICAP_GetVirtualLineWidth(struct VICAP_REG *pReg);
```

1.2.7 Register: VICAP_DVP_SET_SIZE

VICAP_DVP_SET_SIZE寄存器用于设置上层application需要VICAP输出的width和height。

```
HAL_Status HAL_VICAP_SetReceivedSize(struct VICAP_REG *pReg, uint32_t height,
uint32_t width);
```

1.2.8 Register: VICAP_DVP_BLOCK_LINE_NUM

VICAP使用block pingpong mode时，必须对该寄存器进行设置。 计算公式： $num = height \div blocknum$ height是指VICAP实际输出的高，与VICAP_DVP_SET_SIZE寄存器的height一致。 blocknum是指VICAP实际输出的完整图像被分割的总block数。 必须注意的是VICAP_DVP_SET_SIZE寄存器设置的height必须要能被blocknum整除,以保证分配的buf地址能够对齐。

```
HAL_Status HAL_VICAP_SetBlockLineNum(struct VICAP_REG *pReg, uint32_t num);
```

1.2.9 Register: VICAP_DVP_BLOCK0_ADDR_Y/VICAP_DVP_BLOCK0_ADDR_UV

VICAP使用block pingpong mode时需要设置。

```
HAL_Status HAL_VICAP_SetBlock0YAddr(struct VICAP_REG *pReg, uint32_t yAddr);
HAL_Status HAL_VICAP_SetBlock0UvAddr(struct VICAP_REG *pReg, uint32_t
uvAddr);
```

1.2.10 Register: VICAP_DVP_BLOCK1_ADDR_Y/VICAP_DVP_BLOCK1_ADDR_UV

VICAP使用block pingpong mode时需要设置。

```
HAL_Status HAL_VICAP_SetBlock1YAddr(struct VICAP_REG *pReg, uint32_t yAddr);
HAL_Status HAL_VICAP_SetBlock1UvAddr(struct VICAP_REG *pReg, uint32_t
uvAddr);
```

1.2.11 Register: VICAP_DVP_BLOCK_STATUS

VICAP_DVP_BLOCK_STATUS寄存器用于指示block pingpong mode模式下，VICAP的block状态，需要在每个block采样完成后相应清零。

```
uint32_t HAL_VICAP_GetBlockStatus(struct VICAP_REG *pReg);
uint32_t HAL_VICAP_ClearBlockStatus(struct VICAP_REG *pReg, uint32_t mask);
```

1.2.12 Register: VICAP_DVP_CROP

VICAP_DVP_CROP寄存器用于对输出图像进行裁剪时进行设置。

```
HAL_Status HAL_VICAP_SetCropOrdinate(struct VICAP_REG *pReg, uint32_t startY,
uint32_t startX);
```

1.2.13 Register: VICAP_DVP_FRAME_STATUS

VICAP_DVP_FRAME_STATUS寄存器用于指示frame oneframe mode或者frame pingpong mode模式下VICAP的采样状态。frame pingpong mode 不需要对此寄存器清零，frame oneframe mode则需要。

```
uint32_t HAL_VICAP_GetFrameStatus(struct VICAP_REG *pReg);
uint32_t HAL_VICAP_ClearFrameStatus(struct VICAP_REG *pReg, uint32_t mask);
```

2 RTOS VICAP Framework

VICAP framework 是基于RT-Thread/RKOS的设备对象将其抽象为struct rt_VICAP_device来实现的。 为实现通过VICAP采集所连接的camera的图像数据，需要完成以下三部分的工作：

- 完成camera的驱动实现
- 完成VICAP driver所需的自定义功能
- 完成Application端的应用采集

具体说明如下所述。

2.1 Adapter layer

Adapter layer做为适配层，将RT-Thread和RKOS的相关的系统变量和接口做桥接，为vicap驱动在这两套系统的实现提供了统一的接口和变量，具体实现用户可以不用关心。其主要包含以下三个文件：

file name	description
adapter.c	实现了系统接口的桥接
adapter.h	实现了系统对象变量的桥接
adapter_type.h	实现系统整形变量的桥接

2.2 Camera Device Driver

Camera device driver的整体框架是基于RT-Thread/RKOS的设备对象而实现。该框架基于内核对象struct rk_device/struct _DEVICE_CLASS封装了struct rt_camera_device对象，该camera对象为VICAP驱动实现camera类型的设备驱动提供了统一的实现方式。主要包含以下几个文件：

filename	description
camera.c	实现了camera框架
camera.h	声明了camera device的相关对象
camera_mediabus.h	声明了media-bus相关参数及对象

2.2.1 创建和注册camera设备

camera对象struct rt_camera_device具体说明如下。

```
struct rk_camera_device
{
    rk_device parent;
    char name[RK_CAMERA_DEVICE_NAME_SIZE];
    struct rk_camera_info info;
    const struct rk_camera_ops *ops;
    char i2c_name[RK_CAMERA_I2C_NAME_SIZE];
    rk_i2c_bus_device *i2c_bus;
#if defined(__RK_OS__)
    uint8_t class_id;
    uint8_t object_id;
#endif
};
```

field	description
parent	camera对象派生自内核的设备对象，rk_device 由adapter层实现
name	用于设置具体camera的名称，具有唯一性，是内核查找设备的唯一标识。由在camera设备注册时由驱动具体设置。
info	用于设置camera输出的分辨率、mediabus等信息，可为上层应用所获取用于图像配置。由驱动具体设置。
ops	用于实现具体camera的访问和控制。由驱动具体设置。
i2c_name	用于指定具体camera挂载所在的i2c总线的名称，该名称需与对应的系统的i2c总线驱动名称一致。由驱动具体设置。
i2c_bus	由系统接口rk_device_find(...)通过i2c_name获得，进而控制camera。一般在init阶段进行赋值。由驱动具体设置。

其中struct rk_camera_ops *ops，是用户实现具体camera驱动的重点，相关函数由用户自行实现对应回调函数。具体说明如下：

```
struct rk_camera_ops
{
    ret_err_t (*init)(struct rk_camera_device *dev);
    ret_err_t (*open)(struct rk_camera_device *dev, uint16_t oflag);
    ret_err_t (*close)(struct rk_camera_device *dev);
    ret_err_t (*control)(struct rk_camera_device *dev, dt_cmd_t cmd, void
*arg);
    ret_err_t (*rx_indicate)(struct rk_camera_device *dev, ret_size_t size);
};
```

field	description
init	该回调函数必须实现，通过系统接口rk_device_init(...)或者rk_device_open(...)被回调。camera的struct rk_camera_info、i2c_name等相关配置，一般在此函数完成。
open	该回调函数必须实现，通过系统接口rk_device_open(...)被回调。可在此函数完成相关参数检查或者初始设置，与close成对使用。
close	该回调函数必须实现，通过系统接口rk_device_close(...)被回调。可在此函数完成相关参数检查或者初始设置，与open成对使用。
control	该回调函数必须实现，通过系统接口rk_device_control(...)被回调。camera的各种自定义控制用户可在此进行定义。
rx_indicate	该回调函数可选，用于设备在接收到数据时进行相关的额外处理。

对于上述的control回调函数，目前框架针对应用只定义了三种控制命令，若用户有额外的控制需求，可自行添加实现。已有控制如下表格。

command name	description
RK_DEVICE_CTRL_CAMERA_STREAM_ON	该命令将stream on 设备
RK_DEVICE_CTRL_CAMERA_STREAM_OFF	该命令将stream off 设备
RK_DEVICE_CTRL_CAMERA_GET_FORMAT	该命令将获取设备相关的format信息

基于struct rk_camera_device创建的camera实例，通过框架的接口rk_camera_register（...）注册到RTOS设备管理器当中。struct rk_camera_device实例可以是静态的，也可以是动态的。

一般是将struct rk_camera_device对象嵌入到一个用户自定义的设备对象中去，再调用rk_camera_register(...)进行注册。示例如下：

```
struct gc2145_dev
{
    struct rk_camera_device parent;
    char name[RK_CAMERA_DEVICE_NAME_SIZE];

#if RT_USING_GC2145_OPS
    struct gc2145_ops *ops;
#endif

    int32_t pin_rst;
    int32_t pin_pwdn;
    int32_t pin_clkout;
    struct mclk mclk;
    char i2c_name[RK_CAMERA_I2C_NAME_SIZE];
    rk_i2c_bus_device *i2c_bus;
    struct rk_mutex mutex_lock;
};
```

在vicap驱动中，所有注册的camera必须有设备名，目前统一的命名格式为:“sensor_index”,其中index是camera设备在系统内的索引，第一个camera叫sensor_0,第二个叫sensor_1,以此类推。目前vicap驱动只支持单sensor，故统一camera设备叫sensor_0。 在实现完camera设备的驱动后，需要将camera设备注册到VICAP device driver中去，以实现VICAP驱动对camera操作。注册方式为：

```
ret_err_t rk_camera_register(struct rk_camera_device *camera, const char
*name, void *data);
```

2.2.2 访问Camera设备

对于camera而言，上层应用通过调用RT-Thread的系统I/O设备管理接口进行访问，进而回调struct rk_camera_ops *ops相应的控制函数，实现对具体camera的硬件操作。上层应用操作camera设备流程如下:

- step 1: 查找camera设备,以获取注册到内核的camera device

```
dev = rk_device_find(name);
```

- step 2: 在step1获取到设备句柄后，初始化设备

```
rk_device_init(dev);
```

- step 3: 在step2初始化设备成功后，打开设备

```
rk_device_open(dev);
```

- step 4: 在step3打开设备成功后, 控制设备

```
rk_device_control(dev, cmd, &arg)
```

- step 5: 在完成设备的各类控制后，关闭设备

```
rk_device_close(dev);
```

2.3 VICAP Driver Framework

2.3.1 VICAP Driver Framework的说明

VICAP driver framework是基于RTOS的设备对象封装了struct rk_VICAP_device对象，进而实现了VICAP驱动框架以实现具体的VICAP device driver。

目前的VICAP设备驱动已实现其大部分主要功能，用户可以不用关心VICAP controller的具体使用用法而去实现VICAP驱动。用户只要关心application端对VICAP的控制就可以了。

2.3.2 VICAP Application

应用层使用VICAP功能需要按照以下流程进行操作： 其中必须注意的是： 在使用vicap的block模式时，通过RK_DEVICE_CTRL_VICAP_SET_FMT命令设置的height，必须能被通过RK_DEVICE_CTRL_VICAP_SET_BLOCK_NUM命令设置的block num整除，否则会导致地址不对齐而出错。

- step1: Find out the VICAP device by device name:

```
vicapdev = rk_device_find(name);
```

- step2: Open the found VICAP device in step1 to init device:

```
ret = rk_device_open(vicapdev, RT_DEVICE_OFLAG_RDWR);
```

- step3: Set the work mode of VICAP device after opening device:

```
ret = rk_device_control(vicapdev, RK_DEVICE_CTRL_VICAP_SET_WORKMODE,
&workmode);
```

- step4: Set the format for outputing:

```
ret = rk_device_control(vicapdev, RK_DEVICE_CTRL_VICAP_SET_FMT, &format);
```

- step5: Set the crop information if it is required:

```
ret = rk_device_control(vicapdev, RK_DEVICE_CTRL_VICAP_CROP_IMAGE, &crop);
```

- step6: Set the block num if the VICAP's block mode is required:

```
ret = rk_device_control(vicapdev, RK_DEVICE_CTRL_VICAP_SET_BLOCK_NUM, &num);
```

- step7: Set the buf num required by application to capture image:

```
ret = rk_device_control(vicapdev, RK_DEVICE_CTRL_VICAP_REQBUF, &reqbuf);
```

- step8: Query the buffers have been allocated in step 7):

```
ret = rk_device_control(vicapdev, RK_DEVICE_CTRL_VICAP_QUERYBUF, &buf);
```

- step9: Queue all the buffers have been checked in step 8 into VICAP drivers:

```
ret = rk_device_control(vicapdev, RK_DEVICE_CTRL_VICAP_QBUF, &buf);
```

- step10: Stream on the VICAP device to capture image:

```
ret = rk_device_control(vicapdev, RK_DEVICE_CTRL_VICAP_STREAM_ON, RK_NULL);
```

- step11: Dqueue the buf filled with image data from driver for processing in application:

```
ret = rk_device_control(vicapdev, RK_DEVICE_CTRL_VICAP_DQBUF, &buf);
```

- step12: Queue the buf has been processed in application into driver:

```
ret = rk_device_control(vicapdev, RK_DEVICE_CTRL_VICAP_QBUF, &buf);
```

- step13:Loop step11 and step 12 untill application stops streaming;

```
do something user needs.
```

- step14: Stop streaming:

```
ret = rk_device_control(vicapdev, RK_DEVICE_CTRL_VICAP_STREAM_OFF, RK_NULL);
```

- step15: Close device

```
ret = rk_device_close(vicapdev);
```

2.4 VICAP 测试

vicap在系统中实现了vicap_test命令，用于提供vicap的功能展示。 通过在窗口输出vicap_test help 可获得相应的用法提示。 目前测试demo app只支持采集nv12。

2.4.1 RT-Thread 的测试

2.4.1.1 config 的配置

- step1：打开vicap test

```
| Symbol: RT_USING_COMMON_TEST_VICAP [=y]
| Type  : boolean
| Prompt: Enable BSP Common VICAP TEST
|   Location:
|     -> RT-Thread bsp test case
|       -> RT-Thread Common Test case
| (1)     -> Enable BSP Common TEST (RT_USING_COMMON_TEST [=y])
|   Defined at ./../common/tests/Kconfig:117
|   Depends on: RT_USING_COMMON_TEST [=y] && RT_USING_VICAP [=y] &&
RT_USING_CAMERA [=y]

```

- step2: 设置串口命令长度

```
| Symbol: RT_CONSOLEBUF_SIZE [=512]
| Type  : integer
| Prompt: the buffer size for console log printf
|   Location:
|     -> RT-Thread Kernel
|       -> Kernel Device Object
| (1)   -> Using console for rt_kprintf (RT_USING_CONSOLE [=y])
|   Defined at ../../../src/Kconfig:300
|  Depends on: RT_USING_CONSOLE [=y]
```

```
| Symbol: FINSH_ARG_MAX [=12]
| Type  : integer
| Prompt: The command arg num for shell
|   Location:
|     -> RT-Thread Components
|       -> Command shell
|         -> finsh shell (RT_USING_FINSH [=y])
| (1)       -> Using module shell (FINSH_USING_MSH [=y])
|   Defined at ../../../components/finsh/Kconfig:74
|   Depends on: RT_USING_FINSH [=y] && FINSH_USING_MSH [=y]
```

```
| Symbol: FINSH_CMD_SIZE [=256]
| Type  : integer
| Prompt: The command line size for shell
|   Location:
|     -> RT-Thread Components
|       -> Command shell
| (1)     -> finsh shell (RT_USING_FINSH [=y])
|   Defined at ../../../components/finsh/Kconfig:41
|   Depends on: RT_USING_FINSH [=y]
```

2.4.1.2 打开vicap

- step1: 设置vicap格式

具体设置格式和用法，可通过执行vicap_test help获得具体用法。 若是block模式，设置的height或者crop的height必须是block num的整数倍。

```
vicap_test dev_set --set-dev=vicap_0 --set-workmode=block --set-blocks=6 --
set-format=fourcc=NV12,width=320,height=240 --stream-buf=7 --stream-count=2 -
-stream-mode=photo
```

- step2: 打开vicap

```
vicap_test dev_streamon
```

在photo模式下，正常结束会打印log：release buf;在preview模式下，正常执行会连续打印log：fps:帧率，提示帧率。

2.4.2 RKOS的测试

2.4.2.1 config的配置

打开vicap shell选项

```
COMPONENTS_SHELL_VICAP(=y) "Enable VICAP shell command"
```

2.4.2.2 打开vicap

- step1: 创建文件存储目录

```
file.setpath A:\\
```

- step2: 创建图像存储文件

```
file.mf cif.yuv
```

- step3: 创建设备

```
vicap_test dev_create
```

- step4: 设置vicap格式

具体设置格式和用法，可通过执行vicap_test help获得具体用法。 若是block模式，设置的height或者crop的height必须是block num的整数倍。

```
vicap_test dev_set --set-dev=vicap_0 --set-workmode=oneframe --set-blocks=6 --set-format=fourcc=NV12,width=640,height=480 --stream-buf=3 --stream-count=2 --stream-mode=photo
```

- step5: 打开vicap

```
vicap_test dev_streamon
```

- step6: 删除图像存储文件

```
file.df cif.yuv
```