

SIXUENET

To Learn Without Thinking Is To Be Useless, To Think Without Learning Is To Lose
(<http://www.mysixue.com/>)

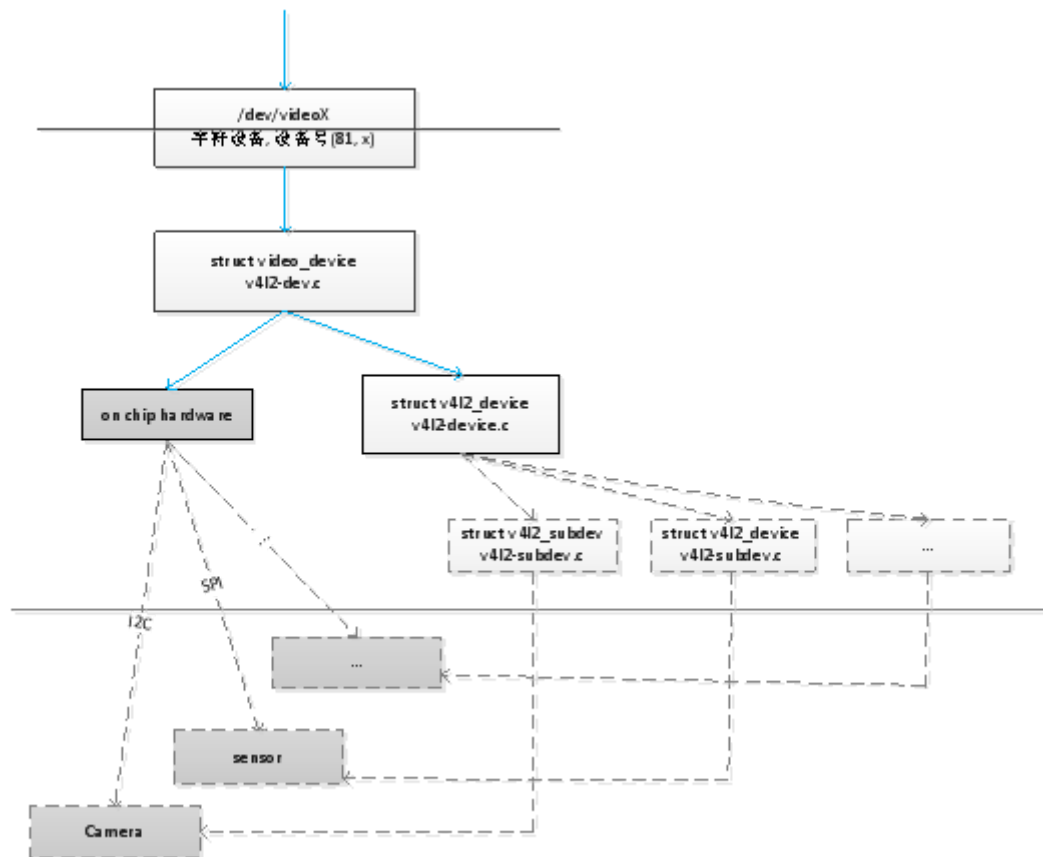
V4L2 SUBSYSTEM

📅 April 20, 2019 ([Http://Www.Mysixue.Com/?P=131](http://Www.Mysixue.Com/?P=131)) 👤 JL
([Http://Www.Mysixue.Com/?Author=1](http://Www.Mysixue.Com/?Author=1)) 🔊

1 Brief description of V4L2 architecture

V4L2 is the Linux community defined Linux multimedia framework kernel , the essence of it is up a character device , then the community defines a set of standard ioctl (<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-ioct.c#L2572>) to interact with the kernel .

1.1 Block Diagram



First, pay attention to the solid line part of the block diagram , which corresponds to the scenario that only needs to drive the on-chip peripherals , such as mtk 's vdec (https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/platform/mtk-vcodec/mtk_vcodec_dec_drv.c#L303) , or atmel 's lcd overlay. In this case, struct video_device is used as a character device driver to respond to user space ioctl . In register struct video_device time , you must create a struct v4l2_device (although no egg with it in this case).

This situation is described in detail in the section ` `video_device " .

Secondly, Note that a block diagram of the solid line + dotted line , corresponds to the need to drive chip peripheral + external hardware scenario , such as the typical C Amara, on-chip peripheral is C Amara controller (e.g. Atmel-isc.c (<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/platform/atmel/atmel-isc.c>)) , external hardware is sensor (such as ov2640.c (<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/i2c/ov2640.c>)) . In this case, struct video_device is still used to abstract the on-chip peripherals , but at the same time, struct v4l2_subdev will be added to abstract the external hardware . The user space ioctl is first transferred to the corresponding driver code of the on-chip peripheral , and then on-chip peripheral code via v4l2_subdev_call calling v4l2_subdev associated hardware to provide an external interface operation .

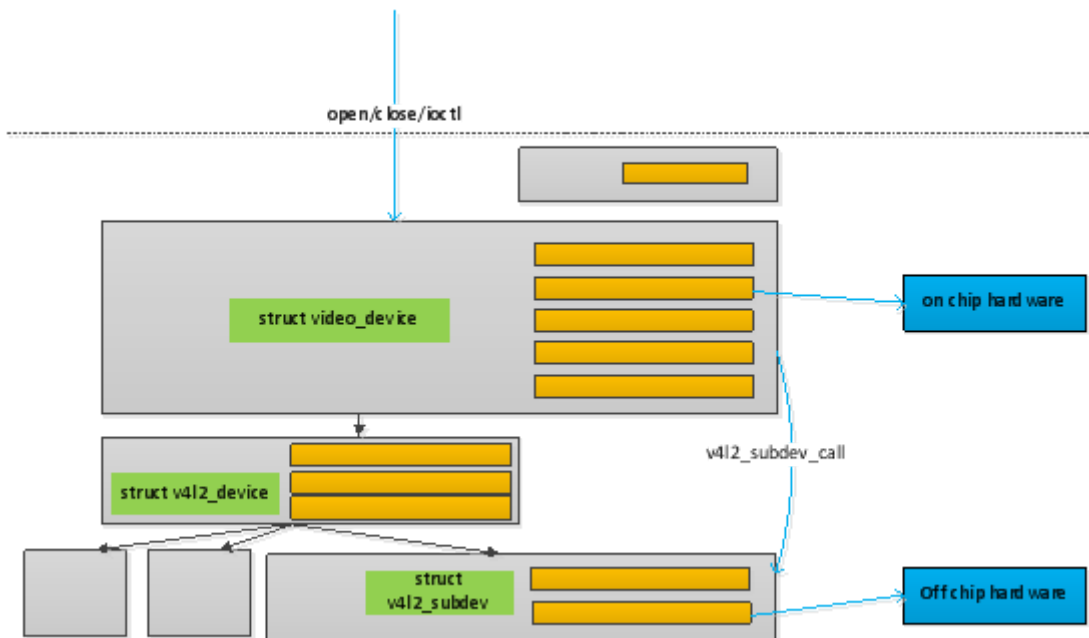
In this case , an on-chip peripherals may have multiple external hardware , each external hardware corresponds to a v4l2_subdev, all subdev are linked in the same v4l2_device next . By v4l2_device, we can traverse all sbudev . V4l2_device this time quite It is the parent device of all subdevs .

The section ` `v4l2_subdev " mainly describes this situation .

There is also a less common situation , such as a radio . In this case , the on-chip peripheral is very simple (that is, I2C) , and we mainly want to drive the external hardware radio. At this time, there is no need to create a character device for the on-chip peripheral (also It is not necessary to register the video_device for on-chip peripherals) , we can directly expose the subdev to the user space in the form of a character device , and the user space is directly sent to the subdev for processing in the ioctl .

The section ``v4l2_subdev as a character device '' describes this situation .

1.2 Relationship diagram



2 video_device

2.1 Core data structure

2.1.1 struct video_device

Header file : `include/media/ v4l2-dev.h` (<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-dev.h#L247>)

Registering a `video_device` with the V4L2 subsystem means that user space will have an extra character device node under `/dev/` . Before registration , you need to prepare the following data structure , so that once the registration is successful , the underlying driver can respond to user space requests .

The data structures that need to be prepared include :

```
struct v4l2_file_operations *fops
```

Header file : `include/media/ v4l2-dev.h` (<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-dev.h#L184>)

The role of this data structure for `video_device` is exactly the same as that of `struct file_operations` for `cdev` . The main purpose is to respond to user space requests such as `open/read/write/ioctl/close` .

```
struct v4l2_ioctl_ops *ioctl_ops
```

Header file : `include/media/ v4l2-ioct.h` (<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-ioct.h#L297>)

User space `ioctl` requests will eventually be forwarded here for processing .

Which `ioctls` can be sent in the user space is specified by the V4L2 standard , so the interface functions that the `v4l2_ioctl_ops` need to implement here are also one-to-one corresponding to the v4l2 standard . For details, please refer to the header file definition .

struct list_head fh_list

Each time a device node is opened , V4L2 core will generate a struct v4l2_fh (see the description herein) corresponding thereto . Thus a video_device case may have a plurality of v4l2_fh. Here's the list head fh_list used to mount all affiliated this video_device of v4l2_fh.

struct v4l2_ctrl_handler *ctrl_handler

Header file : [include/media/v4l2-ctrls.h](https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-ctrls.h#L284) (https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-ctrls.h#L284)

See " v4l2 control related " for details

struct v4l2_prio_state *prio

Header file : [include/media/v4l2-dev.h](https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-dev.h#L95) (https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-dev.h#L95)

The effect is temporarily unknown .

struct vb2_queue *queue

v4l2_ioctl_ops mainly to provide a control interface , and here vb2_queue mainly provides data interface . It is the memory is closely related to the management , memory management in V4L2 is a relatively large frame , so " 1.5 memory management and access," the devoted .

2.1.2 struct v4l2_fh

Header file : [include/media/v4l2-fh.h](https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-fh.h#L47) (https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-fh.h#L47)

Every time the device node is opened , the V4L2 core layer will generate a struct v4l2_fh.

The meaning of this data structure :

On the one hand , it stores some information related to " runtime " , for example, it can store how many events of the currently opened " instance " need to be dequeued , and the priority of the current " instance " .

On the other hand , it is somewhat similar to the C ++ in a derived class , it may be the override ' struct video_device ' some of the information , e.g. Both define struct v4l2_ctrl_handler , the system preferentially used v4l2_fh -> v4l2_ctrl_handler .

2.1.3 v4l2 control related

In V4L2 architecture , user space and kernel mainly through ioctl interaction . V4L2 standard defines the interaction of ' agreement ' .

In the table below for several CTRL type of ioctl (These ioctl mainly some hardware-related settings , such as brightness, saturation, contrast, sharpness and the like), the V4L2 kernel code implements a set of frame , called " v4l2 control " .

IOCTL	Func
VIDIOC_QUERYCTRL	v4l_queryctrl
VIDIOC_QUERY_EXT_CTRL	v4l_query_ext_ctrl
VIDIOC_QUERYMENU	v4l_querymenu
VIDIOC_G_CTRL	v4l_g_ctrl
VIDIOC_S_CTRL	v4l_s_ctrl
VIDIOC_G_EXT_CTRLS	v4l_g_ext_ctrls
VIDIOC_S_EXT_CTRLS	v4l_s_ext_ctrls
VIDIOC_TRY_EXT_CTRLS	v4l_try_ext_ctrls

Of course, you can also choose not to use this set of frameworks , so that the original v4l2_ioctl_ops framework is used . This processing code is typically as follows :

```
static int v4l_queryctrl(const struct v4l2_ioctl_ops *ops,
                        struct file *file, void *fh, void *arg)
{
    struct video_device *vfd = video_devdata(file);
    struct v4l2_queryctrl *p = arg;
    struct v4l2_fh *vfh =
        test_bit(V4L2_FL_USES_V4L2_FH, &vfd->flags) ? fh : NULL;

    if (vfh && vfh->ctrl_handler)
        return v4l2_queryctrl(vfh->ctrl_handler, p);
    if (vfd->ctrl_handler)
        return v4l2_queryctrl(vfd->ctrl_handler, p);
    if (ops->vidioc_queryctrl)
        return ops->vidioc_queryctrl(file, fh, p);
    return -ENOTTY;
}
```

- If you use " the V4L2 Control " framework , the priority call v4l2_fh -> ctrl_handler; followed by video_device -> ctrl_handler.
- Otherwise , use the v4l2_ioctl_ops framework to call the vidioc_xxx function implemented by the underlying driver .

In the " v4l2 control " framework , the kernel system handles a lot of common logic for us , so the code we need to write by ourselves is very simple . Therefore, it is recommended to use this framework first .

Several core data structures related to the " v4l2 control " framework are as follows :

Header file : include/media/v4l2-ctrls.h

struct v4l2_ctrl (<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-ctrls.h#L191>)

struct v4l2_ctrl_ops
(<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-ctrls.h#L69>)

v4l2_ctrl represents an attribute (attribute name / value range, default value, current value) .

v4l2_ctrl_ops represents the operation method of this attribute (getting the attribute value / trying to set the attribute value / setting the attribute value truly).

struct v4l2_ctrl_handler
(<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-ctrls.h#L284>)

Equivalent to a linked list , mount all v4l2_ctrls.

struct video_device has a pointer to this list , its meaning refers to the system up video_device ready Hershey such a list , after you have registered user space can be used directly v4l2 control related functions of .

struct v4l2_fh also has a pointer to this list , when the user space is open when the device node , the pointer will default point video_device -> v4l2_ctrl_handler but the kernel driver can then prepare a new list , then let v4l2_fh -> v4l2_ctrl_handler point to this list . This is called override.

So how to build such a linked list ? See " v 4l2 control APIs " for details .

2.2 APIs

2.2.1 v IDEO _device APIs

struct video_device * [video_device_alloc](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-dev.c#L148>) (void)

Allocate video_device storage space .

static inline int __must_check [video_register_device](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-dev.h#L364>) (struct video_device *vdev, ...)

Register a video_device with the core layer , at this time the core layer will create a character device driver . Later, the user space can interact with the video_device through the character device node .

When the core layer registers a character device , the given ops is [v4l2_fops](#) (<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-dev.c#L459>) , it will forward the request from the user space to video_device->v4l2_file_operations.

2.2.2 v4l2 control APIs

Header file : include/media/v4l2-ctrls.h

Implementation file : drivers/media/v4l2-core/v4l2-ctrls.c

[v4l2_ctrl_handler_init](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-ctrls.h#L422>) (struct v4l2_ctrl_handler *hdl, unsigned int nr_of_controls_hint)

Initialize the v4l2_ctrl_handler linked list , @ nr_of_controls_hint represents the maximum capacity of the linked list

void [v4l2_ctrl_handler_free](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-ctrls.c#L1888>) (struct v4l2_ctrl_handler *hdl)

Clear v4l2_ctrl_handler , release related resources

struct v4l2_ctrl * [v4l2_ctrl_new_std](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-ctrls.c#L2257>) (struct v4l2_ctrl_handler *hdl, const struct v4l2_ctrl_ops *ops , u32 id, s64 min, s64 max, u64 step, s64 def)

Add to the list in a menu-controlled non-variable . @ Min is the minimum value , @ max is the maximum value , @ STEP is the step length , @ DEF is the default .

This function is suitable for control variables that change uniformly within a certain range , such as sound .

struct v4l2_ctrl * [v4l2_ctrl_new_std_menu](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-ctrls.c#L2279>) (struct v4l2_ctrl_handler *hdl, const struct v4l2_ctrl_ops *ops, u32 id, u8 max, u64 mask, u8 def)

Add to the list in a menu control variable . With one API Similarly , @ max is the maximum value , @ DEF is the default value . The difference is that @min IS SET value to 0 and @mask The Determines to Which BE MENU items are Skipped .

This function is suitable for control variables starting from 0 and stepping to 1 .

```
struct                                v4l2_ctrl                                *                                v4l2_ctrl_new_int_menu
(https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-ctrls.c#L2345) (struct v4l2_ctrl_handler *hdl, const struct v4l2_ctrl_ops *ops, u32 id, u8 max, u8 def, const s64 *qmenu_int)
```

With one API similar , but there are significant differences , first @qmenu_int is an array , which stores all the menu selectable values ; secondly @max do not represent the maximum possible value , but represents the maximum array index @def. It represents the default index value .

This function is suitable for control variables whose variable values are discrete and irregular integers .

```
struct                                v4l2_ctrl                                *                                v4l2_ctrl_new_std_menu_items
(https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-ctrls.c#L2312) (struct v4l2_ctrl_handler *hdl, const struct v4l2_ctrl_ops *ops, u32 id, u8 max, u64 mask, u8 def, const char * const *qmenu)
```

A previous API is very similar , the difference is that @qmenu not integer array , but an array of strings . It also more than a @mask, on behalf of the array which item is skip.

```
int v4l2_ctrl_add_handler(struct v4l2_ctrl_handler *hdl, struct v4l2_ctrl_handler *add, v4l2_ctrl_filter filter)
```

The @add all of the control variables are added to @hdl in .

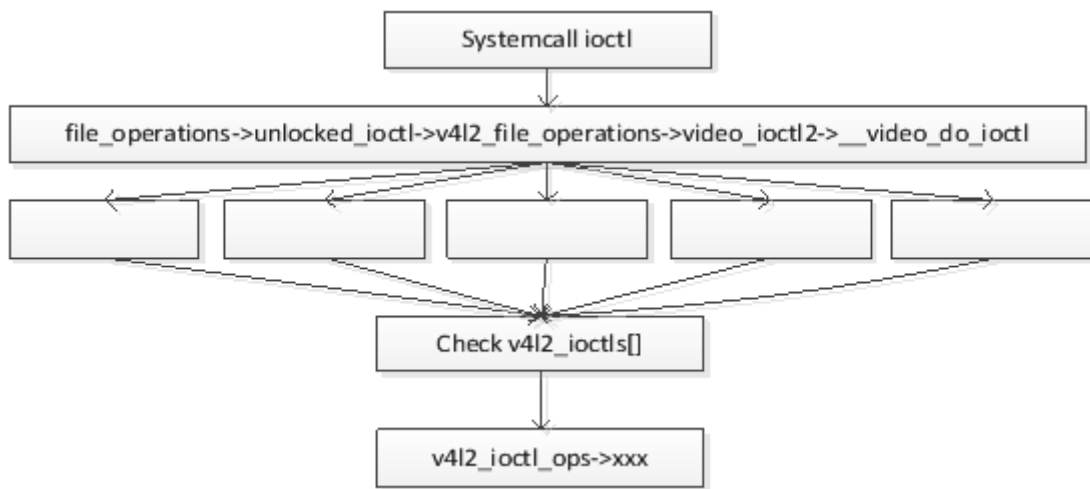
```
int v4l2_ctrl_handler_setup(struct v4l2_ctrl_handler *hdl)
```

The call of this API is optional . Each control variable has its own default value . The purpose of this API is to set the default value of all control variables to the hardware .

After understanding these APIs, it is very simple to see how to construct a v4l2_ctrl_handler linked list . I wo n't go into details here .

2.3 ioctl

User space V4L2 operation of the device is substantially ioctl achieved , I OCT L framework is v4l2_ioctl.c implementation file , the file structure defined in the array v4l2_ioctls (<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-ioctls.c#L2572>) , can be seen as ioctl table command callback function . User Space Call the system call ioctl , pass down the ioctl instruction , and then find the corresponding callback function by looking up this relationship table .



2.4 Demo

Ref MTK 's VDEC (https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/platform/mtk-vcodec/mtk_vcodec_dec_drv.c#L303).

3 v4l2_subdev

Subdev is not actually part of the V4L2 framework , probably because it was added after V4L2 appeared . It defines a set of its own interface function struct v4l2_subdev_ops , all subdev drivers must implement this set of functions .

But this function is V4L2 defined ioctl standard is compatible with the fundamental , we can put a standard user space v4l2 ioctl call onto v4l2_subdev_ops , for example subdev_do_ioctl (<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-subdev.c#L178>) to do this mapping .

3.1 core data structure

3.1.1 struct v4l2_device

Header file : include/media/ v4l2-device.h (<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-device.h#L59>)

v4l2_device there is an element ' struct the kref REF ' , so it functions as a first reference count .

In addition , if there is a system in v4l2_subdev, that v4l2_device played in all v4l2_subdev the parent device , manages registration under which the child devices .

3.1.2 struct v4l2_subdev

Header file : include/media/ v4l2-subdev.h (<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-subdev.h#L842>)

struct v4l2_subdev is used to abstract a sub-device (external hardware). It will be mounted under the v4l2_device linked list .

In addition to this structure , we also need some ops to describe how to operate the hardware , they are :

struct

[v4l2_subdev_ops](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-subdev.h#L736>)

```
struct      v4l2\_subdev\_core\_ops
subdev.h#L197)
struct      v4l2\_subdev\_tuner\_ops
subdev.h#L273)
struct      v4l2\_subdev\_audio\_ops
subdev.h#L313)
struct      v4l2\_subdev\_video\_ops
subdev.h#L424)
struct      v4l2\_subdev\_vbi\_ops
subdev.h#L490)
struct      v4l2\_subdev\_ir\_ops
subdev.h#L608)
struct      v4l2\_subdev\_sensor\_ops
subdev.h#L510)
struct      v4l2\_subdev\_pad\_ops
subdev.h#L683)
```

([https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-](https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-subdev.h#L197)

([https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-](https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-subdev.h#L273)

([https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-](https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-subdev.h#L313)

([https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-](https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-subdev.h#L424)

([https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-](https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-subdev.h#L490)

([https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-](https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-subdev.h#L608)

([https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-](https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-subdev.h#L510)

([https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-](https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-subdev.h#L683)

v4l2_subdev_ops and related several OPS, is mainly used to describe how the hardware , such as registers, etc. is provided .

struct

[v4l2_subdev_internal_ops](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-subdev.h#L764>)

v4l2_subdev_internal_ops provides 4 interfaces : registered , unregistered , open , close . These 4 interfaces are called back to the v4l2 core layer code . When registering / unregistering the subdev with the core layer , the core layer will call back the registered / unregistered here . When the user space opens / closes the device node , the core layer will call back the open/close here .

3.1.3 v4l2 async related

Header file : include/media/ [v4l2-async.h](#) (<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-async.h>)

The async mechanism is somewhat similar to the matching process of device and driver in the device model .

Taking the camera as an example , the hardware is composed of an on-chip camera controller and an external sensor. In the controller-related code, the async mechanism can be used to wait for the external sensor to be initialized . When the sensor initialization code is called , the controller code will be notified through async and then controlled code can begin to create a device node (that is, registered video_device equipment) and so on .

From the point of view of code details , the controller code must first define one or more v4l2_async_subdev, each represents which sensor the controller wants to match , and multiple forms a matching list . Then the controller code needs to implement the callback function defined in v4l2_async_notifier_operations . Finally, both the controller code put together into a package v4l2_async_notifier.

struct

[v4l2_async_subdev](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-async.h#L83>)

struct

[v4l2_async_notifier_operations](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-async.h#L110>)

struct

[v4l2_async_notifier](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-async.h#L134>)

After the data structure is prepared , the controller code can call `v4l2_async_notifier_register` to register the notifier to the system .

When `subdev` initialization , it calls [v4l2_async_notifier_register](#) (<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-async.c#L441>) , for example [ov2640.c](https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/i2c/ov2640.c#L1211) (<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/i2c/ov2640.c#L1211>) , the function scans the notifier list , and match (matching rules See [v4l2_async_find_match](#) (<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-async.c#L92>)) . When a match is found , will call `v4l2_device_register_subdev` (`v4l2_dev`, `sd`) the Sub dev and `v4l2_dev` is associated . Finally , the complete callback function defined in the notifier will be called , and the main controller code will register the `video_device` device in this function . Finally , the device node will appear in the user space .

Later , when the user space operates the device node through `ioctl` , the main controller code will call `v4l2_subdev_call(sd, o, f, args...)` to call the ops function defined in `subdev` , thereby achieving control of `subdev` .

3.2 APIs

3.2.1 v4l2_device APIs

Header file : `include/media/v4l2-device.h`

Implementation file : `drivers/media/v4l2-core/v4l2-device.c`

int

[v4l2_device_register](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-device.c#L33>) (`struct device *dev`, `struct v4l2_device *v4l2_dev`)

The caller allocates a `v4l2_device` space by himself , and then calls this API to initialize the newly allocated space .

void

[v4l2_device_unregister](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-device.c#L104>) (`struct v4l2_device *v4l2_dev`)

Unregister `v4l2_device` and all `v4l2_subdev` mounted under it .

int

[v4l2_device_register_subdev](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-device.c#L154>) (`struct v4l2_device *v4l2_dev`, `struct v4l2_subdev *sd`)

To `v4l2_device` registered a `subdev`.

void

[v4l2_device_unregister_subdev](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-device.c#L291>) (struct v4l2_subdev *sd)

Cancellation of a subdev : it from v4l2_device removed next , if this subdev have created a device node , is called video_unregister_device cancellation node .

static

inline

void

[v4l2_device_get](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-device.h#L81>) (struct v4l2_device *v4l2_dev)

Increase the reference count .

int

[v4l2_device_put](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-device.c#L70>) (struct v4l2_device *v4l2_dev)

Reduce the reference count .

static

inline

void

[v4l2_subdev_notify](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-device.h#L207>) (struct v4l2_subdev *sd, unsigned int notification, void *arg)

Subdev can use this API to notify v4l2_device that an event has occurred .

3.2.2 v4l2_subdev APIs

subdev registration / cancellation of API is v4l2_device_register_subdev / v4l2_device_unregister_subdev . See " v4l2_devices APIs described in the section" .

In addition , v4l2_subdev itself defines the following API:

Header file : include/media/v4l2-subdev.h

Implementation file : drivers/media/v4l2-core/v4l2-subdev.c

void

[v4l2_subdev_init](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-subdev.c#L668>) (struct v4l2_subdev *sd, const struct v4l2_subdev_ops *ops)

Initialize the v4l2_subdev space .

#define

[v4l2_subdev_call](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-subdev.h#L1090>) (sd, o, f, args...)

Call the function defined in v4l2_subdev_xxx_ops .

#define

[v4l2_subdev_has_op](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-subdev.h#L1110>) (sd, o, f)

Check v4l2_subdev_xxx_ops in whether a function is defined .

void

[v4l2_subdev_notify_event](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-subdev.c#L687>) (struct v4l2_subdev *sd, const struct v4l2_event *ev)


The user space can wait for a certain [v4l2_event](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/include/uapi/linux/videodev2.h#L2220>) through ioctl VIDIOC_DQEVENT .

At this time, the v4l2 core code will call v4l2_event_ de queue , block and wait for the event to occur .

(<https://elixir.bootlin.com/linux/v4.18.5/source/include/uapi/linux/videodev2.h#L2220>)

When subdev detects an event (an interrupt usually occurs at this time) , the subdev interrupt handler will call v4l2_subdev_notify_event . v4l2_subdev_notify_event will do two things :

- ✓ First call v4l2_event_Queue , this time will wake the user space of the block, and deliver events to user space .
- ✓  second is to call v4l2_subdev_notify (see " v4l2 device APIs " for details) to notify v4l2_device that an event has occurred .

3.2.3 v4l2 async APIs

Header file : include/media/v4l2-async.h

Implementation file : drivers/media/v4l2-core/v4l2-async.c

int

[v4l2_async_notifier_register](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-async.c#L441>) (struct v4l2_device *v4l2_dev, struct v4l2_async_notifier *notifier)

The on-chip controller-related code calls this API to register a notifier .

void

[v4l2_async_notifier_unregister](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-async.c#L491>) (struct v4l2_async_notifier *notifier)

The on-chip controller-related code calls this API to unregister a notifier.

int

[v4l2_async_register_subdev](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-async.c#L531>) (struct v4l2_subdev *sd)

The external hardware related code calls this API to register a v4l2_subdev with the system , and it will also trigger a matching process with the notifier . If it matches, it will call the callback function defined in v4l2_async_notifier_operations .

void

[v4l2_async_unregister_subdev](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-async.c#L599>) (struct v4l2_subdev *sd)

The external hardware related code calls this API to cancel a v4l2_subdev .

3.3 Demo

Refer Atmel Camera controller code: [atmel-isc.c](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/platform/atmel/atmel-isc.c#L1966>) .

4 v4l2_subdev as a character device

int

[v4l2 device register subdev nodes](#)

(<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-device.c#L225>) (struct v4l2_device *v4l2_dev)

As mentioned earlier , user space interacts with the kernel through device nodes .

Each subdev represents an external hardware , which may be connected to an on-chip controller , such as a camera. In this case, we need to create a device node for this controller , and then pass v4l2_subdev_call in the controller code To operate the external hardware .

In addition , this external hardware may also be directly connected to simple on-chip peripherals such as I2C and SPI , such as radio. In this case, there is no need to create device nodes for these simple peripherals , but directly for the external hardware. Create a device node .

Of course , in the design of on-chip peripherals + external hardware , device nodes can also be created for external hardware . In this way , the external hardware can be operated indirectly through the controller code, or the external hardware can be operated between subdev device nodes .

This API function is for v4l2_dev all mounted under v4l2_subdev, if subdev.flags set V4L2_SUBDEV_FL_HAS_DEVNODE mark , compared with the subdev create a device node (device node creation process is actually registered video_device) .

[subdev_do_ioctl](#) (<https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/v4l2-core/v4l2-subdev.c#L178>) will the v4l2 predetermined ioctl converted v4l2_subdev defined [v4l2_subdev_ops](#) (<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/v4l2-subdev.h#L736>) (subdev defined groups (Core / Tuner / Audio / Video / ...) , each with its own function).

5 Memory Management and Access

5.1 Overview

The previous sections introduced a lot of structures and APIs about control . This section mainly introduces the content of data .

The so-called data , in fact, associated with memory problems . The ultimate goal is to make the kernel and user space can easily exchange data . The V4L2 supports three different data exchange :

- **read and write** : With the help of the read/write interface of the character device , the data in this way needs to be copied between the kernel and the user, and the access speed will be slow . When a large amount of data exchange is involved , do not use this method , which will affect performance .
- **memory-mapped buffer (V4L2_MEMORY_MMAP)** : is the kernel buffer space to open up the application by mmap () is mapped to user address space system calls. These buffers can be large and continuous DMA buffers, virtual buffers created by vmalloc() , or buffers directly opened in the device's IO memory (if

supported by the hardware) .

- **user-space buffer (V4L2_MEMORY_USERPTR)** : is the open space in the user buffer, the buffer pointer exchange between the user and kernel space. Obviously, in this case it is not required mmap () call, but driving the need for additional design , in order to make it accessible to the user space memory .

Read and write methods belong to the frame IO access method , and each frame requires data copy between the user and the kernel ; the latter two are stream IO access methods , which do not require memory copy , and the access speed is relatively fast .

The memory mapped buffer access method is a more commonly used method .

Videobuf management : the videobuf2-core.c , videobuf2-DMA-contig.c , videobuf2-DMA-sg.c , videobuf2-memops.c , videobuf2-vmalloc.c , the V4L2-mem2mem.c and other documents to achieve , complete videobuffer of Assignment, management and cancellation .

5.2 Core data structure

struct vb2_queue

In order to support the flow device IO this manner , the drive needs to achieve struct vb2_queue , structure details are as follows :

Header file : include/media/ [videobuf2-core.h](https://elixir.bootlin.com/linux/v4.18.5/source/include/media/videobuf2-core.h#L509) (https://elixir.bootlin.com/linux/v4.18.5/source/include/media/videobuf2-core.h#L509)

Struct vb2_queue	Comment
unsigned int type	private buffer type whose content is defined by the vb2-core caller. For example, for V4L2, it should match the types defined on @ enum v4l2_buf_type (https://elixir.bootlin.com/linux/v4.18.5/source/include/uapi/linux/videodev2.h#L134)
unsigned int io_modes	Access IO way : mmap , userptr , etc . See enum vb2_io_modes (https://elixir.bootlin.com/linux/v4.18.5/source/include/media/videobuf2-core.h#L195)
...	
const struct vb2_ops *ops	For a set of queue manipulation functions , for example, queues, the queues .
const struct vb2_mem_ops *mem_ops	A collection of operation functions for mem , such as memory allocation and release .
const struct vb2_buf_ops *buf_ops	callbacks to deliver buffer information between user-space and kernel-space.
...	
struct vb2_buffer *bufs[VB2_MAX_FRAME]	Each vb2_buffer represents a piece of memory , which is the storage pool
unsigned int num_buffers	The actual pool inside How many buf
...	

vb2_queue represents a videobuffer queue , vb2_buffer is the basic unit in this queue , vb2_mem_ops is the set of memory operation functions , and vb2_ops is used to manage the queue .

struct vb2_buffer

A vb2_buffer represents a videobuffer in the kernel space . There will be multiple such buffers in a vb2_queue .

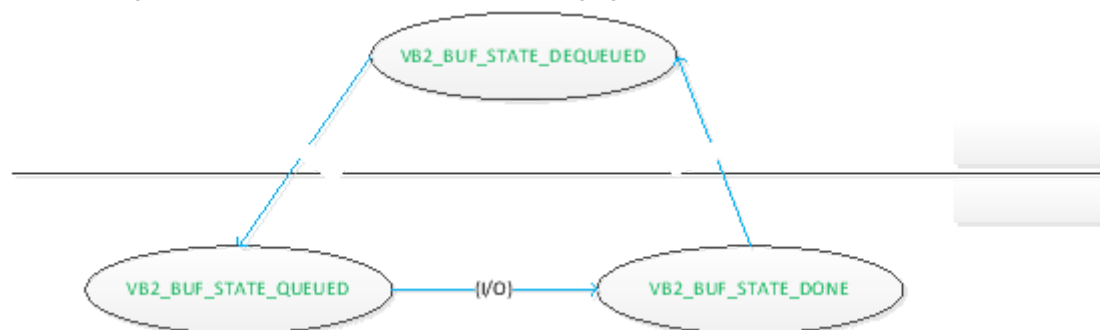
Header file : include/media/ videobuf2-core.h (<https://elixir.bootlin.com/linux/v4.18.5/source/include/media/videobuf2-core.h#L242>)

Struct vb2_buffer	Comment
struct vb2_queue *vb2_queue	Point to the vb2_queue to which the buf belongs
unsigned int memory	the method, in which the actual data is passed
...	
enum vb2_buffer_state (https://elixir.bootlin.com/linux/v4.18.5/source/include/media/videobuf2-core.h#L218) state	Multiple states of a buffer

A buffer may have multiple states , and the common situations are as follows :

- ✓ **driving incoming queue (VB2_BUF_STATE_QUEUED)** : Driver will be processed in this queue buffer , the user space through the IOCTL: VIDIOC_QBUF the input to the queue buffer . For a video capture device , the incoming queue The buffer is empty, and the driver will fill it with data .
- ✓ **driving outgoing queue (VB2_BUF_STATE_DONE)** : These buffers are processed by the drive, for a video capture device, a video buffer has been filled with data, the user is waiting for space to claim
- ✓ **user space queue status (VB2_BUF_STATE_DEQUEUED)** : has passed IOCTL: VIDIOC_DQBUF out to the user buffer space , this time owned by the user buffer space , the drive can not be accessed .

The switching of these three states is shown in the following figure :



struct v4l2_buffer

A v4l2_buffer on behalf of a user space videobuf , it vb2_buffer correspondence . But note that , vb2_buffer which contains the actual storage space ; but v4l2_buffer only contains buffer info, these info, user space can mmap or other The way to get the actual storage address , so as to avoid data copy between user space and the kernel .

Header file : include/uapi/linux/ videodev.h (<https://elixir.bootlin.com/linux/v4.18.5/source/include/uapi/linux/videodev2.h#L917>)

Struct v4l2_buffer	Comment
__u32 index	buffer number
__u32 type	buffer type , see @ enum v4l2_buf_type (https://elixir.bootlin.com/linux/v4.18.5/source/include/uapi/linux/videodev2.h#L134)
__u32 bytesused	Number of bytes used in the buffer
__u32 flags	
__u32 field	
struct timeval timestamp	Timestamp, which represents the time when the frame was captured

struct v4l2_timecode timecode	
__u32 sequence	
/*memory location */ __u32 memory	Indicates whether the buffer is a memory mapped buffer or a user space buffer
union { __u32 offset; unsigned long userptr; struct v4l2_plane *planes; __s32 fd; } m;	//offset represents the location of the kernel buffer //userptr represents the user space address of the buffer
__u32 length	Buffer size , unit byte

When the user space gets v4l2_buffer , you can get the relevant information of the buffer . Byteused is the number of bytes occupied by the image data . If it is the V4L2_MEMORY_MMAP method , m.offset is the starting address of the kernel space image data storage , which will be passed to the mmap function As an offset , a buffer pointer p is returned through mmap mapping , p+byteused is the area occupied by the image data in the virtual address space of the process ; if it is the user pointer buffer mode , the pointer m of the starting address of the image data that can be obtained .userptr , userptr is a pointer to user space , userptr+byteused is the virtual address space occupied , and applications can directly access it .

struct vb2_mem_ops

vb2_mem_ops contains the memory operation methods of memory mapped buffer and user space buffer :

Header file : include/media/ [videobuf2-core.h](https://elixir.bootlin.com/linux/v4.18.5/source/include/media/videobuf2-core.h#L116) (https://elixir.bootlin.com/linux/v4.18.5/source/include/media/videobuf2-core.h#L116)

Struct v b2_mem_ops	Comment
void *(*alloc)(.....)	Allocate video cache
void (*put)(void *buf_priv)	Release video cache
void *(*get_userptr)(.....)	Get user space video buffer pointer
void (*put_userptr)(void *buf_priv)	Free user space video buffer pointer
void (*prepare)(void *buf_priv) void (*finish)(void *buf_priv) void *(*vaddr)(void *buf_priv) void *(*cookie)(void *buf_priv)	Used for cache synchronization
unsigned int (*num_users)(void *buf_priv)	Returns the number of buffers in user space for the current period
int (*mmap)(void *buf_priv, struct vm_area_struct *vma)	Map the buffer to user space

This is a very large structure . So many structures need to be implemented and not exhausted . Fortunately, the kernel has already implemented it for us . Three types of video buffer operation methods are provided : continuous DMA buffer, distribution of DMA buffers and vmalloc buffer created , respectively, by videobuf2-DMA-contig.c , videobuf2-DMA-sg.c and videobuf-vmalloc.c file implementation , it may be used according to the actual situation .

struct vb2_ops

vb2_ops is a collection of functions used to manage the buffer queue , including queue and buffer initialization :

Header file : include/media/ [videobuf2-core.h](https://elixir.bootlin.com/linux/v4.18.5/source/include/media/videobuf2-core.h#L384) (https://elixir.bootlin.com/linux/v4.18.5/source/include/media/videobuf2-core.h#L384)

Struct vb2_ops	Comment
int(*queue_setup)(struct vb2_queue *q, const struct v4l2_format *fmt, unsigned int *num_buffers, unsigned int *num_planes, unsigned int sizes[], void *alloc_ctxs[]);	Queue initialization
void (*wait_prepare)(struct vb2_queue *q) void (*wait_finish)(struct vb2_queue *q)	Release and acquire device operation lock
int (*buf_init)(struct vb2_buffer *vb) int (*buf_prepare)(struct vb2_buffer *vb) void (*buf_finish)(struct vb2_buffer *vb) void (*buf_cleanup)(struct vb2_buffer *vb)	Operation on buffer
int (*start_streaming)(struct vb2_queue *q, unsigned int count)	Start video streaming
int (*stop_streaming)(struct vb2_queue *q)	Stop video streaming
void(*buf_queue)(struct vb2_buffer *vb)	The VB from v4l2 core layer is transmitted to the drive

6 Demo: User space access device

The following access buffer video device kernel mode map (C Apture D evice) process .

1> Open the device file

```
fd = open ( dev_name , O_RDWR /* required */ | O_NONBLOCK , 0 );
dev_name [ /dev/video X ]
```

2> Query the capabilities supported by the device

```
struct v4l2_capability cap ;
ioctl ( fd , VIDIOC_QUERYCAP , & cap )
```

3> Set the video capture format

```
FMT . type = V4L2_BUF_TYPE_VIDEO_CAPTURE ;
fmt . fmt . pix . width      = 640 ;
fmt . fmt . pix . height     = 480 ;
fmt . fmt . pix . pixelformat = V4L2_PIX_FMT_YUYV ; // Pixel format
fmt . fmt . pix . field      = V4L2_FIELD_INTERLACED ;

ioctl ( fd , VIDIOC_S_FMT , & fmt )
```

4> Apply for buffer from the driver

```
struct v4l2_requestbuffers req ;
REQ . COUNT = . 4 ; // buffer number
req . type = V4L2_BUF_TYPE_VIDEO_CAPTURE ;
req . memory = V4L2_MEMORY_MMAP ;

if ( - 1 == ioctl ( fd , VIDIOC_REQBUFS , & req ))
```

5> Get the information of each buffer and map it to user space

```
struct buffer {
```

```

    void * start ;
    size_t length ;
} * buffers ;

buffers = calloc ( req . count , sizeof (* buffers )) ;
for ( n_buffers = 0 ; n_buffers < req . count ; ++ n_buffers ) {
    struct v4l2_buffer buf ;

    buf . type          = V4L2_BUF_TYPE_VIDEO_CAPTURE ;
    buf . Memory        = V4L2_MEMORY_MMAP ;
    buf . index         = n_buffers ;
    if ( - 1 == ioctl ( fd , VIDIOC_QUERYBUF , & buf ))
        errno_exit ( "VIDIOC_QUERYBUF" ) ;

    buffers [ n_buffers ]. length = buf . length ;
    buffers [ n_buffers ]. start = mmap ( NULL /* start anywhere */ ,
        buf . length ,
        PROT_READ | PROT_WRITE /* required */ ,
        MAP_SHARED /* recommended */ ,
        fd , buf . m . offset ) ;
}

```

6> Put the buffer into the incoming queue , open the stream IO , and start video capture

```

for ( i = 0 ; i < n_buffers ; ++ i ) {
    struct v4l2_buffer buf ;

    buf . type = V4L2_BUF_TYPE_VIDEO_CAPTURE ;
    buf . Memory = V4L2_MEMORY_MMAP ;
    buf . index = i ;

    if ( - 1 == ioctl ( fd , VIDIOC_QBUF , & buf ))
        errno_exit ( "VIDIOC_QBUF" ) ;
}

```

```

type = V4L2_BUF_TYPE_VIDEO_CAPTURE ;
if ( - 1 == ioctl ( fd , VIDIOC_STREAMON , & type ))

```

7> Call select to monitor the file descriptor , whether the data in the buffer is filled , and then check the video data

```

for (;;) {
    fd_set fds ;
    struct timeval tv ;
    int r ;
    FD_ZERO (& fds ) ;
    FD_SET ( fd , & fds ) ;
    /* Timeout. */
    tv . tv_sec = 2 ;
    tv . tv_usec = 0 ;

    // Check whether the file description has changed

```

```

r = select ( fd + 1 , & fds , NULL , NULL , & tv );
if ( - 1 == r ) {
    if ( EINTR == errno )
        continue ;
    errno_exit ( "select" );
}

if ( 0 == r ) {
    fprintf ( stderr , "select timeout\n" );
    exit ( EXIT_FAILURE );
}

// Process the video data
if ( read_frame ()
    break ;
/* EAGAIN - continueselect loop. */
}

```

8> a good buffer has been filled , the obtained size of the video data , and then process the data . Here only contain information extracted buffer buffer , no video data copy

```

buf . type = V4L2_BUF_TYPE_VIDEO_CAPTURE ;
buf . Memory = V4L2_MEMORY_MMAP ;

```

```

if ( - 1 == ioctl ( fd , VIDIOC_QBUF , & buf )) // Remove the buffer
    errno_exit ( "VIDIOC_QBUF" );

```

```

process_image ( buffers [ buf . index ]. start , buf . bytesused ); // Video data
processing

```

```

if ( - 1 == ioctl ( fd , VIDIOC_QBUF , & buf )) // and then put it in the incoming queue
    errno_exit ( "VIDIOC_QBUF" );

```

9> Stop video capture

```

type = V4L2_BUF_TYPE_VIDEO_CAPTURE ;
ioctl ( fd , VIDIOC_STREAMOFF , & type );

```

10> Turn off the device

```

close ( fd );

```

7 Soc Camera

In the current code , soc_camera (https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/i2c/soc_camera) seems to have no code in use , as if the kernel is gradually removing it . Camera- related scenes are now the v4l2 framework used in between .
(https://elixir.bootlin.com/linux/v4.18.5/source/drivers/media/i2c/soc_camera)

If you encounter the use of soc_camera in subsequent projects, I will study it in detail .

2 thoughts on “ V4L2 Subsystem ”



Bosspoipoi says:

October 8, 2020 at am11:22 (<http://www.mysixue.com/?p=131#comment-32681>)

Thanks to the blogger for bringing excellent content, the source code and analysis are very attentive. I have a question about one of them, struct v4l2_subdev_ops is undoubtedly a collection of functions that the user ioctl finally parses and executes, then v4l2_subdev_ops is v4l2_subdev alone as the ioctl callback of /dev/v4l-subdev, or v4l2_device as the ioctl callback of /dev/videoX ?



JL says:

October 28, 2020 at am8:22 (<http://www.mysixue.com/?p=131#comment-32712>)

Hello, the two cases you mentioned can be used, depending on what form is used when registering subdev.
