Table of Contents

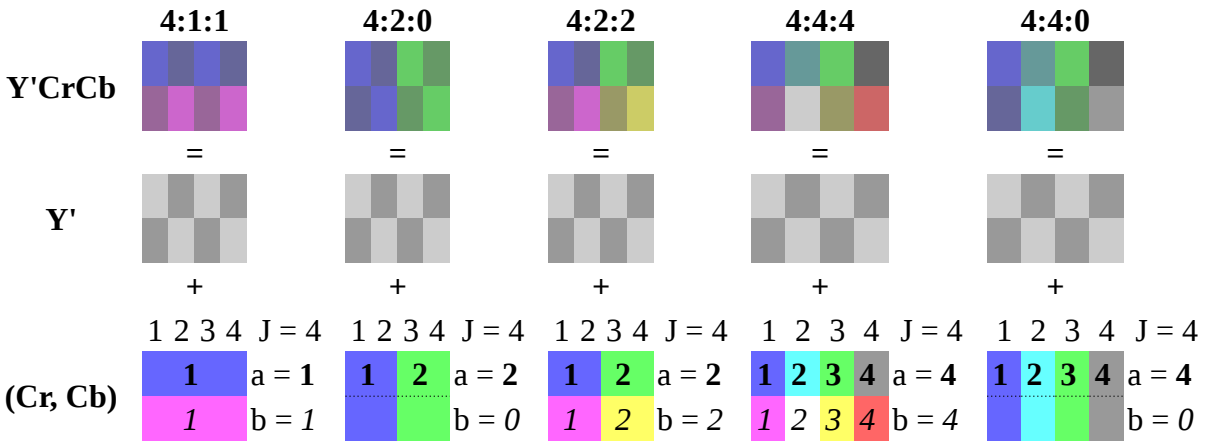
YUV Color Format

1. Source YUV buffer is a **packed or planar buffer**. Packed means that the YUV bits are grouped together, planar means the Y, U and V buffers are separated in 3 different memory area.

2. YUV channel size; the single Y,U,V channel could be of **8-bit, 10-bit, 12-bit**, etc

The sub-sampling scheme is commonly expressed as a three part ratio *J*:*a*:*b* (e.g. 4:2:2) or four parts if alpha channel is present (e.g. 4:2:2:4), that describe the number of luminance and chrominance samples in a conceptual region that is *J* pixels wide, and 2 pixels high. The parts are (in their respective order):

- *J*: horizontal sampling reference (width of the conceptual region). Usually, 4.
- *a*: number of chrominance samples (Cr, Cb) in the first row of *J* pixels.
- *b*: number of changes of chrominance samples (Cr, Cb) between first and second row of *J* pixels.



Pixel Format Example: (YUV)(FOUR CC)

```
/* Luminance+Chrominance formats */ /*packed, 8 Bit*/
#define V4L2_PIX_FMT_YUYV    v4l2_fourcc('Y', 'U', 'Y', 'V') /* 16 YUV 4:2:2     */
#define V4L2_PIX_FMT_YYUV    v4l2_fourcc('Y', 'Y', 'U', 'V') /* 16 YUV 4:2:2     */
#define V4L2_PIX_FMT_YVYU    v4l2_fourcc('Y', 'V', 'Y', 'U') /* 16 YVU 4:2:2     */
#define V4L2_PIX_FMT_UYVY    v4l2_fourcc('U', 'Y', 'V', 'Y') /* 16 YUV 4:2:2     */
#define V4L2_PIX_FMT_VYUY    v4l2_fourcc('V', 'Y', 'U', 'Y') /* 16 YUV 4:2:2     */

/* two planes -- one Y, one Cr + Cb interleaved  */ /*planar 8 Bit*/
#define V4L2_PIX_FMT_NV12    v4l2_fourcc('N', 'V', '1', '2') /* 12 Y/CbCr 4:2:0  */
#define V4L2_PIX_FMT_NV16    v4l2_fourcc('N', 'V', '1', '6') /* 16 Y/CbCr 4:2:2  */
#define V4L2_PIX_FMT_NV24    v4l2_fourcc('N', 'V', '2', '4') /* 24 Y/CbCr 4:4:4  */
```

Pixel format Example in gst-launch- Capture a frame:

Ubuntu UVC Camera:

```
gst-launch-1.0 v4l2src device=/dev/video0 ! "video/x-raw, format=(string)UYVY,
width=(int)2304, height=(int)1296".
```

Xilinx Platform:

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! video/x-raw,
format=NV12,width=3840,height=2160,framerate=60/1
```

I.MX6 Platform:

```
gst-launch-1.0 imxv4l2videosrc device=/dev/video0 imx-capture-mode=5 fps-n=30
focus-mode=6
```

imx-capture-mode for ov5640 example resolutions:

```
ov5640_mode_VGA_640_480 = 0,
ov5640_mode_QVGA_320_240 = 1,
ov5640_mode_NTSC_720_480 = 2,
ov5640_mode_PAL_720_576 = 3,
ov5640_mode_720P_1280_720 = 4,
ov5640_mode_1080P_1920_1080 = 5
```

Pixel Format – Exercise (Identify: Packed/Planer , Sub sampling, Pixel Bits, width, height, Image Size)

S.NO	FOURCC	PIXEL FORMAT	ANALYSIS
1	YUYV	V4L2_PIX_FMT_YUYV	Step 1: Packed Step 2: NILL Step 3: Y,U,V : Each 8 Bit Step 4: Sub-Sampling: 4:2:2 Step 5: 2 Pixel = 2 (Y) + 1 (U) +1(V) = 4*8=32 1 Pixel = 32/2=16 bits Step 6: Frame size(640x480) = 640*480*16/8 Bytes = 614400 Bytes
2	NV12	V4L2_PIX_FMT_NV12	Step 1: Planer Step 2: Y is one plane ,UV Packed in another Plane Step 3: Y,U,V : Each 8 Bit Step 4: Sub-Sampling: 4:2:0 Step 5: 4 Pixel = 4 (Y) + 1 (U) +1(V) = 6*8=48 1 Pixel = 48/4=12 bits Step 6: Frame size(640x480) = 640*480*12/8

			Bytes = 460800 Bytes
3	NV24	V4L2_PIX_FMT_NV24	Step 1: Planer Step 2: Y is one plane ,UV Packed in another Plane Step 3: Y,U,V : Each 8 Bit Step 4: Sub-Sampling: 4:4:4 Step 5: 1 Pixel = 1 (Y) + 1 (U) +1 (V) = 3*8=24 1 Pixel = 24 bits Step 6: Frame size(640x480) = 640*480*24/8 Bytes = 921600 Bytes
4	NV16	V4L2_PIX_FMT_NV16	Step 1: Planer Step 2: Y is one plane ,UV Packed in another Plane Step 3: Y,U,V : Each 8 Bit Step 4: Sub-Sampling: 4:2:2 Step 5: 2 Pixel = 2 (Y) + 1 (U) +1(V) = 4*8=32 1 Pixel = 32/2=16 bits Step 6: Frame size(640x480) = 640*480*2 Bytes = 614400 Bytes
5	Y41P	V4L2_PIX_FMT_Y41P	Step 1: Packed Step 2: NILL Step 3: Y,U,V : Each 8 Bit Step 4: Sub-Sampling: 4:1:1 Step 5: 4 Pixel = 4 (Y) + 1 (U) +1(V) = 6*8=48 1 Pixel = 48/4=12 bits Step 6: Frame size(640x480) = 640*480*12/8 Bytes = 460800 Bytes

V4L2 Application

IOCTLS

S.NO	IOCTL CMD	DESCRIPTION	ARGUMENT
1	VIDIOC_QUERYCAP	Query device capabilities	struct v4l2_capability { __u8 driver[16]; __u8 card[32]; __u8 bus_info[32]; __u32 version; **__u32 capabilities;** __u32 device_caps; __u32 reserved[3];

			}; /* Values for 'capabilities' field */ #define V**4L2_CAP_VIDEO_CAPTURE** 0x00000001 /* Is a video capture device */

2	VIDIOC_G_FMT,
	VIDIOC_S_FMT,
	VIDIOC_TRY_FMT
		Get or set the data format, try a format	struct v4l2_format {
			__u32 type;
			union {
			struct v4l2_pix_format pix; /* V4L2_BUF_TYPE_VIDEO_CAPTURE */
			struct v4l2_pix_format_mplane pix_mp; /* V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE */
			struct v4l2_window win; /* V4L2_BUF_TYPE_VIDEO_OVERLAY */
			struct v4l2_vbi_format vbi; /* V4L2_BUF_TYPE_VBI_CAPTURE */
			struct v4l2_sliced_vbi_format sliced; /* V4L2_BUF_TYPE_SLICED_VBI_CAPTURE */
			struct v4l2_sdr_format sdr; /* V4L2_BUF_TYPE_SDR_CAPTURE */
			struct v4l2_meta_format meta; /* V4L2_BUF_TYPE_META_CAPTURE */
			__u8 raw_data[200]; /* user-defined */
			} fmt;
			};
			
			struct v4l2_pix_format {
			__u32 width;
			__u32 height;
			__u32 pixelformat;
			__u32 field; /* enum v4l2_field */
			__u32 bytesperline; /* for padding, zero if unused */
			__u32 sizeimage;
			__u32 colorspace; /* enum v4l2_colorspace */
			__u32 priv; /* private data, depends on pixelformat */
			__u32 flags; /* format flags (V4L2_PIX_FMT_FLAG_*) */
			union {
			/* enum v4l2_ycbcr_encoding */
			__u32 ycbcr_enc;
			/* enum v4l2_hsv_encoding */

			__u32 hsv_enc; }; __u32 quantization; /* enum v4l2_quantization */ __u32 xfer_func; /* enum v4l2_xfer_func */ };

3	VIDIOC_REQBUFS	Initiate Memory Mapping	struct v4l2_requestbuffers {
			__u32 count;
			__u32 type; /* enum v4l2_buf_type */
			__u32 memory; /* enum v4l2_memory */
			__u32 capabilities;
			__u32 reserved[1];
			};

4	VIDIOC_QUERYBUF	Query the status of a buffer	struct v4l2_buffer {
			__u32 index;
			__u32 type;
			__u32 bytesused;
			__u32 flags;
			__u32 field;
			struct timeval timestamp;
			struct v4l2_timecode timecode;
			__u32 sequence;
			/* memory location */
			__u32 memory;
			union {
			__u32 offset;
			unsigned long userptr;
			struct v4l2_plane *planes;
			__s32 fd;
			} m;
			__u32 length;
			__u32 reserved2;
			union {
			__s32 request_fd;
			__u32 reserved;
			};
			};

5	VIDIOC_QBUF	Exchange a	struct v4l2_buffer {

	VIDIOC_DQBUF	buffer with the driver	__u32 index;
			__u32 type;
			__u32 bytesused;
			__u32 flags;
			__u32 field;
			struct timeval timestamp;
			struct v4l2_timecode timecode;
			__u32 sequence;
			
			/* memory location */
			__u32 memory;
			union {
			__u32 offset;
			unsigned long userptr;
			struct v4l2_plane *planes;
			__s32 fd;
			} m;
			__u32 length;
			__u32 reserved2;
			union {
			__s32 request_fd;
			__u32 reserved;
			};
			};

6	VIDIOC_STREAMON VIDIOC_STREAMOFF	Start or stop streaming	enum v4l2_buf_type { **V4L2_BUF_TYPE_VIDEO_CAPTURE** = 1, V4L2_BUF_TYPE_VIDEO_OUTPUT = 2, V4L2_BUF_TYPE_VIDEO_OVERLAY = 3, V4L2_BUF_TYPE_VBI_CAPTURE = 4, V4L2_BUF_TYPE_VBI_OUTPUT = 5, V4L2_BUF_TYPE_SLICED_VBI_CAPTURE = 6, V4L2_BUF_TYPE_SLICED_VBI_OUTPUT = 7, V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY = 8, V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE = 9, V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE = 10, V4L2_BUF_TYPE_SDR_CAPTURE = 11, V4L2_BUF_TYPE_SDR_OUTPUT = 12, V4L2_BUF_TYPE_META_CAPTURE = 13, V4L2_BUF_TYPE_META_OUTPUT

			= 14, /* Deprecated, do not use */ V4L2_BUF_TYPE_PRIVATE = 0x80, };

Capture (V4L2)Application

Step 1:

#define VIDEO_DEVICE_NODE "/dev/video0"
int fd=-1;

int open_device()
{
	fd=open(**VIDEO_DEVICE_NODE**,O_RDWR);
	if(fd < 0){
		printf("device open failed\n");
		return -1;
	}
}

Step 2:

int close_device()
{
	if(fd>0){
		close(fd);
	}
}

Step 3:

int device_capabilities()
{
	struct v4l2_capability cap;
	int ret;
	ret=ioctl(fd,VIDIOC_QUERYCAP,&cap);

	if(cap.**capabilities** & **V4L2_CAP_VIDEO_CAPTURE**){
		printf("Device Support the video capture mode\n");
		return 0;

```
	}else
		return -1;
}
```

Step 4:

```
int get_set_format()
{
	int ret;
	struct v4l2_format fmt;
	fmt.fmt.pix.field=V4L2_FIELD_NONE;
	fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

	ret=ioctl(fd,VIDIOC_G_FMT, &fmt);
	if(ret < 0)
		printf("Unable to get device get format\n");
	else {
		printf("width=%d\n",fmt.fmt.pix.width);
		printf("height=%d\n",fmt.fmt.pix.height);
		printf("bytesperline=%d\n",fmt.fmt.pix.bytesperline);
		printf("image size=%d\n",fmt.fmt.pix.sizeimage);
		printf("colorspace=%d\n",fmt.fmt.pix.colorspace);
		sizeimage=fmt.fmt.pix.sizeimage;
	}
}
```

Step 5:

```
int request_buffers()
{
	struct v4l2_requestbuffers buffers;
	int ret;
	buffers.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
	buffers.count=3;
	buffers.memory= V4L2_MEMORY_MMAP;
	ret=ioctl(fd, VIDIOC_REQBUFS, &buffers);
	if(ret < 0){
		printf("request buffers failed\n");
	}
}
```

Step 6:

```
int query_buffers()
{
```

```
        struct v4l2_buffer buffer;
        int i,ret;

        for(i=0;i<3;i++) {
                buffer.index=i;
                buffer.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
                buffer.memory= V4L2_MEMORY_MMAP;
                ret=ioctl(fd,VIDIOC_QUERYBUF,&buffer);
                if(ret < 0) {
                        printf("querybuf failed at index=%d\n",i);
                }
                user_ptrs[i]=mmap(0,sizeimage, PROT_READ , MAP_SHARED ,fd,0);

                ret=ioctl(fd,VIDIOC_QBUF,&buffer);
                if(ret < 0) {
                        printf("QBUF after querybuf failed\n");
                }
        }
}
```

Step 7:

```
int stream_on()
{
        int ret;
        int type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
        ret = ioctl(fd,VIDIOC_STREAMON,&type);
        if(ret < 0) {
                printf("stream on failed\n");
        }
}
```

Step 8:

```
int stream_off()
{
        int ret;
        int type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
        ret = ioctl(fd, VIDIOC_STREAMOFF, &type);
        if(ret < 0){
                printf("stream off failed\n");
        }
}
```

Step 9:

```
int process_image()
{
        struct v4l2_buffer buffer;
        buffer.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
       buffer.memory= V4L2_MEMORY_MMAP;
        int n=100;
        int ret;

        while(n > 1) {
                ret=ioctl(fd,VIDIOC_DQBUF,&buffer);
                if(ret < 0) {
                        printf("DQBUF failed\n");
                }
                n--;

                ret=ioctl(fd,VIDIOC_QBUF,&buffer);
                if(ret < 0) {
                        printf("QBUF failed\n");
                }

        }
}
```

Step 10:

```
int clear_buffers()
{
        struct v4l2_requestbuffers buffers;
      int ret,i;

      for(i=0;i<3;i++) {
                munmap(user_ptrs[i],sizeimage);
        }
      buffers.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
      buffers.count=0;
      buffers.memory= V4L2_MEMORY_MMAP;
      ret=ioctl(fd, VIDIOC_REQBUFS, &buffers);
      if(ret < 0){
            printf("clearing : request buffers failed\n");
      }
}
```

Step 11:

```
int main()
{
	int ret;
	ret = open_device();
	if(ret < 0) {
		printf("Device open failed\n");
		return -1;
	}
	ret = device_capabilities();
	if(ret < 0) {
		printf("Device does not support video capture\n");
	}
	get_set_format();
	request_buffers();
	query_buffers();
	stream_on();
	process_image();
	stream_off();
	clear_buffers();
	close_device();
}
```

YAVTA Application:

clone:
https://github.com/fastr/yavta

Compile & Test on Ubuntu USB Camera:

./yavta -c10 -fYUYV -n3 -s640x480 –
file=/home/alex/Desktop/interview_2019/complex_subsystem/video-camera/application/capture

Usage: ./yavta [options] device
Supported options:

-c, --capture[=nframes]	**Capture frames**
-d, --delay	Delay (in ms) before requeuing buffers
-f, --format format	**Set the video format**
-F, --file[=prefix]	**Read/write frames from/to disk**
-h, --help	Show this help screen
-i, --input input	Select the video input
-l, --list-controls	List available controls
-n, --nbufs n	**Set the number of video buffers**

-p, --pause	Pause before starting the video stream
-q, --quality n	MJPEG quality (0-100)
-r, --get-control ctrl	Get control 'ctrl'
-s, --size WxH	**Set the frame size**

v4l2-Application-Code Flow

Step 1: Application → v4l2-dev.c (drivers/media/v4l2-core) → driver(drivers/media/platform/omap3isp/ispdevice.c)

Application	**Drivers/media/v4l2-core/v4l2-dev.c**	**Implementation**

YAVTA	**static const struct file_operations v4l2_fops** = {
	.owner = THIS_MODULE,
	.read = v4l2_read,
	.write = v4l2_write,
	.open = v4l2_open,
	.get_unmapped_area = v4l2_get_unmapped_area,
		v4l2_open:
		struct video_device *vdev;
		if (vdev->fops->open) {
		if (video_is_registered(vdev))
		ret = **vdev->fops->open**(filp);
		else
		ret = -ENODEV;
		}

	.mmap = v4l2_mmap,
	.unlocked_ioctl = v4l2_ioctl,
	#ifdef CONFIG_COMPAT
	.compat_ioctl = v4l2_compat_ioctl32,
	#endif
	.release = v4l2_release,
	.poll = v4l2_poll,
	.llseek = no_llseek,
	};
	
	/* **Part 3: Initialize the character device */**
	vdev->cdev = cdev_alloc();
	if (vdev->cdev == NULL) {
	ret = -ENOMEM;
	goto cleanup;
	}
	vdev->cdev->ops = &**v4l2_fops;**
	vdev->cdev->owner = owner;
	ret = cdev_add(vdev->cdev, MKDEV(VIDEO_MAJOR, vdev->minor), 1);
	if (ret < 0) {
	pr_err("%s: cdev_add failed\n", __func__);
	kfree(vdev->cdev);
	vdev->cdev = NULL;
	goto cleanup;
	}
		v4l2_release:
		struct video_device *vdev = video_devdata(filp);
		int ret = 0;
		
		if (vdev->fops->release)
		ret = v**dev->fops->release**(filp);
		
		v4l2_mmap:
		if (video_is_registered(vdev))
		ret = **vdev->fops->mmap**(filp, vm);
		
		v4l2_ioctl:
		
		struct video_device *vdev = video_devdata(filp);
		
		if (vdev->fops->unlocked_ioctl) {
		if (video_is_registered(vdev))
		
		ret = **vdev->fops->unlocked_ioctl**(filp, cmd, arg);
		
		}

Step 2: (ispvideo.c) drivers/media/platform/omap3isp

struct video_device *video;

static const struct v4l2_file_operations isp_video_fops = {
.owner = THIS_MODULE,
.unlocked_ioctl = video_ioctl2,
.open = isp_video_open,

```
        .release = isp_video_release,
        .poll = isp_video_poll,
        .mmap = isp_video_mmap,
};


 video->video.fops = &isp_video_fops;
snprintf(video->video.name, sizeof(video->video.name),
         "OMAP3 ISP %s %s", name, direction);
video->video.vfl_type = VFL_TYPE_GRABBER;
video->video.release = video_device_release_empty;
video->video.ioctl_ops = &isp_video_ioctl_ops;
```

step 3: .unlocked_ioctl = video_ioctl2,

(drivers/media/v4l2-core/v4l2-ioctl.c)

```
long video_ioctl2(struct file *file,
            unsigned int cmd, unsigned long arg)
{
        return video_usercopy(file, cmd, arg, __video_do_ioctl);
}
EXPORT_SYMBOL(video_ioctl2);
```

```
static long __video_do_ioctl(struct file *file, unsigned int cmd, void
*arg)
{
        struct video_device *vfd = video_devdata(file);
        struct mutex *lock; /* ioctl serialization mutex */
        const struct v4l2_ioctl_ops *ops = vfd->ioctl_ops;
        bool write_only = false;
        struct v4l2_ioctl_info default_info;
        const struct v4l2_ioctl_info *info;
        void *fh = file->private_data;
        struct v4l2_fh *vfh = NULL;
        int dev_debug = vfd->dev_debug;
        long ret = -ENOTTY;

        if (ops == NULL) {
                pr_warn("%s: has no ioctl_ops.\n",
                                video_device_node_name(vfd));
                return ret;
        }
```

```
if (test_bit(V4L2_FL_USES_V4L2_FH, &vfd->flags))
        vfh = file->private_data;

lock = v4l2_ioctl_get_lock(vfd, vfh, cmd, arg);

if (lock && mutex_lock_interruptible(lock))
        return -ERESTARTSYS;

if (!video_is_registered(vfd)) {
        ret = -ENODEV;
        goto unlock;
}

if (v4l2_is_known_ioctl(cmd)) {
        info = &v4l2_ioctls[_IOC_NR(cmd)];

        if (!test_bit(_IOC_NR(cmd), vfd->valid_ioctls) &&
           !((info->flags & INFO_FL_CTRL) && vfh && vfh->ctrl_handler))
                goto done;

        if (vfh && (info->flags & INFO_FL_PRIO)) {
                ret = v4l2_prio_check(vfd->prio, vfh->prio);
                if (ret)
                        goto done;
        }
} else {
        default_info.ioctl = cmd;
        default_info.flags = 0;
        default_info.debug = v4l_print_default;
        info = &default_info;
}

write_only = _IOC_DIR(cmd) == _IOC_WRITE;
if (info != &default_info) {
        ret = info->func(ops, file, fh, arg);
} else if (!ops->vidioc_default) {
        ret = -ENOTTY;
} else {
        ret = ops->vidioc_default(file, fh,
                vfh ? v4l2_prio_check(vfd->prio, vfh->prio) >= 0 : 0,
                cmd, arg);
}
```

```
done:
        if (dev_debug & (V4L2_DEV_DEBUG_IOCTL |
V4L2_DEV_DEBUG_IOCTL_ARG)) {
                if (!(dev_debug & V4L2_DEV_DEBUG_STREAMING) &&
                    (cmd == VIDIOC_QBUF || cmd == VIDIOC_DQBUF))
                        goto unlock;

                v4l_printk_ioctl(video_device_node_name(vfd), cmd);
                if (ret < 0)
                        pr_cont(": error %ld", ret);
                if (!(dev_debug & V4L2_DEV_DEBUG_IOCTL_ARG))
                        pr_cont("\n");
                else if (_IOC_DIR(cmd) == _IOC_NONE)
                        info->debug(arg, write_only);
                else {
                        pr_cont(": ");
                        info->debug(arg, write_only);
                }
        }

unlock:
        if (lock)
                mutex_unlock(lock);
        return ret;
}
```

step 4: v4l2_ioctls - struct v4l2_ioctl_info

```
struct v4l2_ioctl_info {
        unsigned int ioctl;
        u32 flags;
        const char * const name;
        int (*func)(const struct v4l2_ioctl_ops *ops, struct file *file,
                        void *fh, void *p);
        void (*debug)(const void *arg, bool write_only);
};

#define IOCTL_INFO(_ioctl, _func, _debug, _flags)      \
        [_IOC_NR(_ioctl)] = {                          \
                .ioctl = _ioctl,                       \
                .flags = _flags,                       \
                .name = #_ioctl,                       \
                .func = _func,                          \
```

```
                .debug = _debug,                                \
        }

static struct v4l2_ioctl_info v4l2_ioctls[] = {
        IOCTL_INFO(VIDIOC_QUERYCAP, v4l_querycap, v4l_print_querycap, 0),
        IOCTL_INFO(VIDIOC_ENUM_FMT, v4l_enum_fmt, v4l_print_fmtdesc,
INFO_FL_CLEAR(v4l2_fmtdesc, type)),
        IOCTL_INFO(VIDIOC_G_FMT, v4l_g_fmt, v4l_print_format, 0),
        IOCTL_INFO(VIDIOC_S_FMT, v4l_s_fmt, v4l_print_format,
INFO_FL_PRIO),
        IOCTL_INFO(VIDIOC_REQBUFS, v4l_reqbufs, v4l_print_requestbuffers,
INFO_FL_PRIO | INFO_FL_QUEUE),
        IOCTL_INFO(VIDIOC_QUERYBUF, v4l_querybuf, v4l_print_buffer,
INFO_FL_QUEUE | INFO_FL_CLEAR(v4l2_buffer, length)),
        IOCTL_INFO(VIDIOC_G_FBUF, v4l_stub_g_fbuf, v4l_print_framebuffer,
0),
        IOCTL_INFO(VIDIOC_S_FBUF, v4l_stub_s_fbuf, v4l_print_framebuffer,
INFO_FL_PRIO),
        IOCTL_INFO(VIDIOC_OVERLAY, v4l_overlay, v4l_print_u32,
INFO_FL_PRIO),
        IOCTL_INFO(VIDIOC_QBUF, v4l_qbuf, v4l_print_buffer,
INFO_FL_QUEUE),
        IOCTL_INFO(VIDIOC_EXPBUF, v4l_stub_expbuf, v4l_print_exportbuffer,
INFO_FL_QUEUE | INFO_FL_CLEAR(v4l2_exportbuffer, flags)),
        IOCTL_INFO(VIDIOC_DQBUF, v4l_dqbuf, v4l_print_buffer,
INFO_FL_QUEUE),
        IOCTL_INFO(VIDIOC_STREAMON, v4l_streamon, v4l_print_buftype,
INFO_FL_PRIO | INFO_FL_QUEUE),
        IOCTL_INFO(VIDIOC_STREAMOFF, v4l_streamoff, v4l_print_buftype,
INFO_FL_PRIO | INFO_FL_QUEUE),
        IOCTL_INFO(VIDIOC_G_PARM, v4l_g_parm, v4l_print_streamparm,
INFO_FL_CLEAR(v4l2_streamparm, type)),
        IOCTL_INFO(VIDIOC_S_PARM, v4l_s_parm, v4l_print_streamparm,
INFO_FL_PRIO),
        IOCTL_INFO(VIDIOC_G_STD, v4l_stub_g_std, v4l_print_std, 0),
        IOCTL_INFO(VIDIOC_S_STD, v4l_s_std, v4l_print_std, INFO_FL_PRIO),
        IOCTL_INFO(VIDIOC_ENUMSTD, v4l_enumstd, v4l_print_standard,
INFO_FL_CLEAR(v4l2_standard, index)),
        IOCTL_INFO(VIDIOC_ENUMINPUT, v4l_enuminput, v4l_print_enuminput,
INFO_FL_CLEAR(v4l2_input, index)),
        IOCTL_INFO(VIDIOC_G_CTRL, v4l_g_ctrl, v4l_print_control,
INFO_FL_CTRL | INFO_FL_CLEAR(v4l2_control, id)),
        IOCTL_INFO(VIDIOC_S_CTRL, v4l_s_ctrl, v4l_print_control,
INFO_FL_PRIO | INFO_FL_CTRL),
```

```
	IOCTL_INFO(VIDIOC_G_TUNER, v4l_g_tuner, v4l_print_tuner,
INFO_FL_CLEAR(v4l2_tuner, index)),
	IOCTL_INFO(VIDIOC_S_TUNER, v4l_s_tuner, v4l_print_tuner,
INFO_FL_PRIO),
	IOCTL_INFO(VIDIOC_G_AUDIO, v4l_stub_g_audio, v4l_print_audio, 0),
	IOCTL_INFO(VIDIOC_S_AUDIO, v4l_stub_s_audio, v4l_print_audio,
INFO_FL_PRIO),
	IOCTL_INFO(VIDIOC_QUERYCTRL, v4l_queryctrl, v4l_print_queryctrl,
INFO_FL_CTRL | INFO_FL_CLEAR(v4l2_queryctrl, id)),
	IOCTL_INFO(VIDIOC_QUERYMENU, v4l_querymenu,
v4l_print_querymenu, INFO_FL_CTRL | INFO_FL_CLEAR(v4l2_querymenu,
index)),
	IOCTL_INFO(VIDIOC_G_INPUT, v4l_stub_g_input, v4l_print_u32, 0),
	IOCTL_INFO(VIDIOC_S_INPUT, v4l_s_input, v4l_print_u32,
INFO_FL_PRIO),
	IOCTL_INFO(VIDIOC_G_EDID, v4l_stub_g_edid, v4l_print_edid,
INFO_FL_ALWAYS_COPY),
	IOCTL_INFO(VIDIOC_S_EDID, v4l_stub_s_edid, v4l_print_edid,
INFO_FL_PRIO | INFO_FL_ALWAYS_COPY),
	IOCTL_INFO(VIDIOC_G_OUTPUT, v4l_stub_g_output, v4l_print_u32, 0),
	IOCTL_INFO(VIDIOC_S_OUTPUT, v4l_s_output, v4l_print_u32,
INFO_FL_PRIO),
	IOCTL_INFO(VIDIOC_ENUMOUTPUT, v4l_enumoutput,
v4l_print_enumoutput, INFO_FL_CLEAR(v4l2_output, index)),
	IOCTL_INFO(VIDIOC_G_AUDOUT, v4l_stub_g_audout,
v4l_print_audioout, 0),
	IOCTL_INFO(VIDIOC_S_AUDOUT, v4l_stub_s_audout,
v4l_print_audioout, INFO_FL_PRIO),
	IOCTL_INFO(VIDIOC_G_MODULATOR, v4l_g_modulator,
v4l_print_modulator, INFO_FL_CLEAR(v4l2_modulator, index)),
	IOCTL_INFO(VIDIOC_S_MODULATOR, v4l_s_modulator,
v4l_print_modulator, INFO_FL_PRIO),
	IOCTL_INFO(VIDIOC_G_FREQUENCY, v4l_g_frequency,
v4l_print_frequency, INFO_FL_CLEAR(v4l2_frequency, tuner)),
	IOCTL_INFO(VIDIOC_S_FREQUENCY, v4l_s_frequency,
v4l_print_frequency, INFO_FL_PRIO),
	IOCTL_INFO(VIDIOC_CROPCAP, v4l_cropcap, v4l_print_cropcap,
INFO_FL_CLEAR(v4l2_cropcap, type)),
	IOCTL_INFO(VIDIOC_G_CROP, v4l_g_crop, v4l_print_crop,
INFO_FL_CLEAR(v4l2_crop, type)),
	IOCTL_INFO(VIDIOC_S_CROP, v4l_s_crop, v4l_print_crop,
INFO_FL_PRIO),
	IOCTL_INFO(VIDIOC_G_SELECTION, v4l_g_selection,
v4l_print_selection, INFO_FL_CLEAR(v4l2_selection, r)),
```

```
	IOCTL_INFO(VIDIOC_S_SELECTION, v4l_s_selection, v4l_print_selection,
INFO_FL_PRIO | INFO_FL_CLEAR(v4l2_selection, r)),
	IOCTL_INFO(VIDIOC_G_JPEGCOMP, v4l_stub_g_jpegcomp,
v4l_print_jpegcompression, 0),
	IOCTL_INFO(VIDIOC_S_JPEGCOMP, v4l_stub_s_jpegcomp,
v4l_print_jpegcompression, INFO_FL_PRIO),
	IOCTL_INFO(VIDIOC_QUERYSTD, v4l_querystd, v4l_print_std, 0),
	IOCTL_INFO(VIDIOC_TRY_FMT, v4l_try_fmt, v4l_print_format, 0),
	IOCTL_INFO(VIDIOC_ENUMAUDIO, v4l_stub_enumaudio,
v4l_print_audio, INFO_FL_CLEAR(v4l2_audio, index)),
	IOCTL_INFO(VIDIOC_ENUMAUDOUT, v4l_stub_enumaudout,
v4l_print_audioout, INFO_FL_CLEAR(v4l2_audioout, index)),
	IOCTL_INFO(VIDIOC_G_PRIORITY, v4l_g_priority, v4l_print_u32, 0),
	IOCTL_INFO(VIDIOC_S_PRIORITY, v4l_s_priority, v4l_print_u32,
INFO_FL_PRIO),
	IOCTL_INFO(VIDIOC_G_SLICED_VBI_CAP, v4l_g_sliced_vbi_cap,
v4l_print_sliced_vbi_cap, INFO_FL_CLEAR(v4l2_sliced_vbi_cap, type)),
	IOCTL_INFO(VIDIOC_LOG_STATUS, v4l_log_status, v4l_print_newline,
0),
	IOCTL_INFO(VIDIOC_G_EXT_CTRLS, v4l_g_ext_ctrls,
v4l_print_ext_controls, INFO_FL_CTRL),
	IOCTL_INFO(VIDIOC_S_EXT_CTRLS, v4l_s_ext_ctrls,
v4l_print_ext_controls, INFO_FL_PRIO | INFO_FL_CTRL),
	IOCTL_INFO(VIDIOC_TRY_EXT_CTRLS, v4l_try_ext_ctrls,
v4l_print_ext_controls, INFO_FL_CTRL),
	IOCTL_INFO(VIDIOC_ENUM_FRAMESIZES, v4l_stub_enum_framesizes,
v4l_print_frmsizeenum, INFO_FL_CLEAR(v4l2_frmsizeenum, pixel_format)),
	IOCTL_INFO(VIDIOC_ENUM_FRAMEINTERVALS,
v4l_stub_enum_frameintervals, v4l_print_frmivalenum,
INFO_FL_CLEAR(v4l2_frmivalenum, height)),
	IOCTL_INFO(VIDIOC_G_ENC_INDEX, v4l_stub_g_enc_index,
v4l_print_enc_idx, 0),
	IOCTL_INFO(VIDIOC_ENCODER_CMD, v4l_stub_encoder_cmd,
v4l_print_encoder_cmd, INFO_FL_PRIO | INFO_FL_CLEAR(v4l2_encoder_cmd,
flags)),
	IOCTL_INFO(VIDIOC_TRY_ENCODER_CMD, v4l_stub_try_encoder_cmd,
v4l_print_encoder_cmd, INFO_FL_CLEAR(v4l2_encoder_cmd, flags)),
	IOCTL_INFO(VIDIOC_DECODER_CMD, v4l_stub_decoder_cmd,
v4l_print_decoder_cmd, INFO_FL_PRIO),
	IOCTL_INFO(VIDIOC_TRY_DECODER_CMD, v4l_stub_try_decoder_cmd,
v4l_print_decoder_cmd, 0),
	IOCTL_INFO(VIDIOC_DBG_S_REGISTER, v4l_dbg_s_register,
v4l_print_dbg_register, 0),
	IOCTL_INFO(VIDIOC_DBG_G_REGISTER, v4l_dbg_g_register,
```

```
v4l_print_dbg_register, 0),
        IOCTL_INFO(VIDIOC_S_HW_FREQ_SEEK, v4l_s_hw_freq_seek,
v4l_print_hw_freq_seek, INFO_FL_PRIO),
        IOCTL_INFO(VIDIOC_S_DV_TIMINGS, v4l_stub_s_dv_timings,
v4l_print_dv_timings, INFO_FL_PRIO | INFO_FL_CLEAR(v4l2_dv_timings,
bt.flags)),
        IOCTL_INFO(VIDIOC_G_DV_TIMINGS, v4l_stub_g_dv_timings,
v4l_print_dv_timings, 0),
        IOCTL_INFO(VIDIOC_DQEVENT, v4l_dqevent, v4l_print_event, 0),
        IOCTL_INFO(VIDIOC_SUBSCRIBE_EVENT, v4l_subscribe_event,
v4l_print_event_subscription, 0),
        IOCTL_INFO(VIDIOC_UNSUBSCRIBE_EVENT, v4l_unsubscribe_event,
v4l_print_event_subscription, 0),
        IOCTL_INFO(VIDIOC_CREATE_BUFS, v4l_create_bufs,
v4l_print_create_buffers, INFO_FL_PRIO | INFO_FL_QUEUE),
        IOCTL_INFO(VIDIOC_PREPARE_BUF, v4l_prepare_buf, v4l_print_buffer,
INFO_FL_QUEUE),
        IOCTL_INFO(VIDIOC_ENUM_DV_TIMINGS, v4l_stub_enum_dv_timings,
v4l_print_enum_dv_timings, INFO_FL_CLEAR(v4l2_enum_dv_timings, pad)),
        IOCTL_INFO(VIDIOC_QUERY_DV_TIMINGS, v4l_stub_query_dv_timings,
v4l_print_dv_timings, INFO_FL_ALWAYS_COPY),
        IOCTL_INFO(VIDIOC_DV_TIMINGS_CAP, v4l_stub_dv_timings_cap,
v4l_print_dv_timings_cap, INFO_FL_CLEAR(v4l2_dv_timings_cap, pad)),
        IOCTL_INFO(VIDIOC_ENUM_FREQ_BANDS, v4l_enum_freq_bands,
v4l_print_freq_band, 0),
        IOCTL_INFO(VIDIOC_DBG_G_CHIP_INFO, v4l_dbg_g_chip_info,
v4l_print_dbg_chip_info, INFO_FL_CLEAR(v4l2_dbg_chip_info, match)),
        IOCTL_INFO(VIDIOC_QUERY_EXT_CTRL, v4l_query_ext_ctrl,
v4l_print_query_ext_ctrl, INFO_FL_CTRL |
INFO_FL_CLEAR(v4l2_query_ext_ctrl, id)),
};
#define V4L2_IOCTLS ARRAY_SIZE(v4l2_ioctls)
```

step 4: struct v4l2_ioctl_info - func

```
static int v4l_querycap(const struct v4l2_ioctl_ops *ops,
                        struct file *file, void *fh, void *arg)
{
        struct v4l2_capability *cap = (struct v4l2_capability *)arg;
```

```
	struct video_device *vfd = video_devdata(file);
	int ret;

	cap->version = LINUX_VERSION_CODE;
	cap->device_caps = vfd->device_caps;
	cap->capabilities = vfd->device_caps | V4L2_CAP_DEVICE_CAPS;

	ret = ops->vidioc_querycap(file, fh, cap);

	cap->capabilities |= V4L2_CAP_EXT_PIX_FORMAT;
	/*
	 * Drivers MUST fill in device_caps, so check for this and
	 * warn if it was forgotten.
	 */
	WARN(!(cap->capabilities & V4L2_CAP_DEVICE_CAPS) ||
		!cap->device_caps, "Bad caps for driver %s, %x %x",
		cap->driver, cap->capabilities, cap->device_caps);
	cap->device_caps |= V4L2_CAP_EXT_PIX_FORMAT;

	return ret;
}


static int v4l_s_input(const struct v4l2_ioctl_ops *ops,
				struct file *file, void *fh, void *arg)
{
	struct video_device *vfd = video_devdata(file);
	int ret;

	ret = v4l_enable_media_source(vfd);
	if (ret)
		return ret;
	return ops->vidioc_s_input(file, fh, *(unsigned int *)arg);
}

static int v4l_streamon(const struct v4l2_ioctl_ops *ops,
				struct file *file, void *fh, void *arg)
{
	return ops->vidioc_streamon(file, fh, *(unsigned int *)arg);
}


static int v4l_streamoff(const struct v4l2_ioctl_ops *ops,
				struct file *file, void *fh, void *arg)
```

```
{
        return ops->vidioc_streamoff(file, fh, *(unsigned int *)arg);
}
```

Linux : Video Device:

S.NO		
1	Linux Kernel version	4.19.0 [https://github.com/Xilinx/linux-xlnx]
2	V4l2 core PATH	drivers/media/v4l2-core/
3	V4L2 Core Files	videodev-objs := v4l2-dev.o v4l2-ioctl.o v4l2-device.o v4l2-fh.o v4l2-event.o v4l2-ctrls.o v4l2-subdev.o v4l2-clk.o v4l2-async.o

S.NO	Structure	Allocation & Register
1	**struct video_device**	video_device_alloc, video_device_release video_register_device, video_unregister_device **struct v4l2_file_operations** **struct v4l2_ioctl_ops**
2	**struct v4l2_device**	**v4l2_device_register, v4l2_device_unregister, v4l2_async_notifier_register** **struct v4l2_async_notifier_operations**
3	**struct v4l2_subdev**	**v4l2_subdev_init, v4l2_async_register_subdev v4l2_async_unregister_subdev** **struct v4l2_subdev_ops** **struct v4l2_subdev_internal_ops**

```
struct video_device {
#if defined(CONFIG_MEDIA_CONTROLLER);
  struct media_entity entity;
  struct media_intf_devnode *intf_devnode;
  struct media_pipeline pipe;
#endif;
  const struct v4l2_file_operations *fops;
  u32 device_caps;
  struct device dev;
  struct cdev *cdev;
  struct v4l2_device *v4l2_dev;
  struct device *dev_parent;
```

```
  struct v4l2_ctrl_handler *ctrl_handler;
  struct vb2_queue *queue;
  struct v4l2_prio_state *prio;
  char name[32];
  enum vfl_devnode_type vfl_type;
  enum vfl_devnode_direction vfl_dir;
  int minor;
  u16 num;
  unsigned long flags;
  int index;
  spinlock_t fh_lock;
  struct list_head        fh_list;
  int dev_debug;
  v4l2_std_id tvnorms;
  void (*release)(struct video_device *vdev);
  const struct v4l2_ioctl_ops *ioctl_ops;
  unsigned long valid_ioctls[BITS_TO_LONGS(BASE_VIDIOC_PRIVATE)];
  struct mutex *lock;
};
```

Linux : Media Device

S.NO	**Structure**	**Allocation & Register**
1	media-objs := media-device.o media-devnode.o media-entity.o	obj-$(CONFIG_MEDIA_SUPPORT) += media.o
2	struct media_entity *entity, struct media_pad *pads	media_entity_pads_init, media_entity_cleanup
3	struct media_entity *entity, struct media_pipeline *pipe	media_pipeline_start
4	struct media_graph *graph, struct media_device *mdev	media_graph_walk_init,media_graph_walk_start,media_graph_walk_next
5	struct media_device *mdev, struct media_entity *entity	media_device_register_entity, media_device_unregister_entity

Media Device

A Media device is the umbrella device under which multiple sub-entities called Media entities can be accessed, modified and worked upon. The Media Device is exposed to the user in form of a device file, which can be opened to enumerate, set and get the parameters of each of the media entities. For example, in DM365 implementation the entire VPFE capture device with its IPIPE, IPIPEIF, CCDC etc is exposed as a Media Device - /dev/media0. If there were a display driver, it would be exposed as a Media device too.

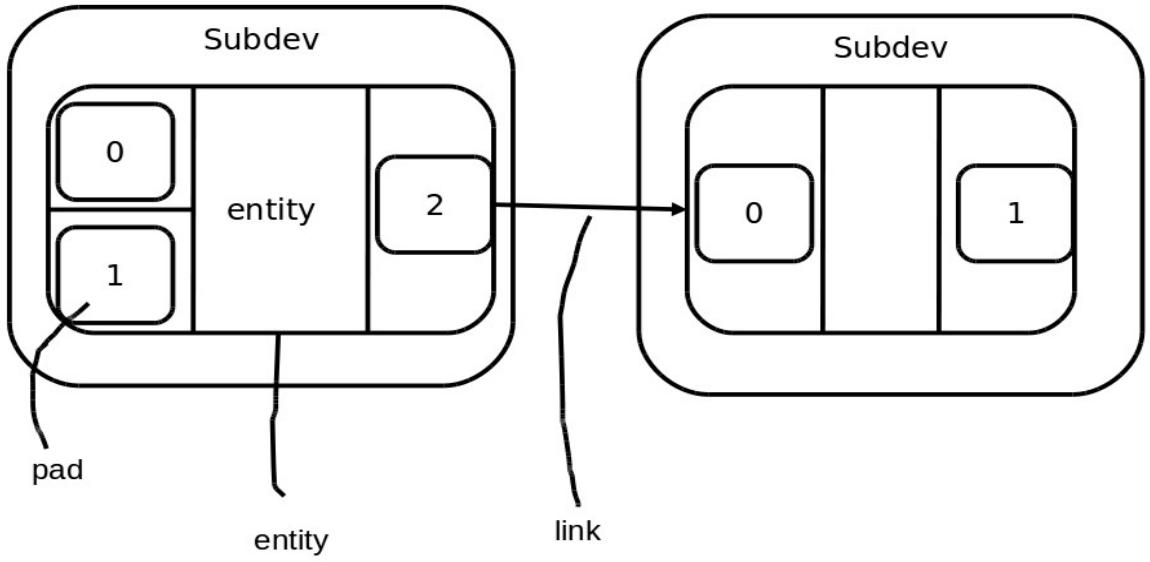
Media Entity

A Media entity is a sub-block of a a particular media device which usually does a particular function,

can be though about as an connect-able but self-contained block which might have a register set f its own for setting the parameters and can be programed in an independent way. This could be a sub-IP, or a helping device on the same board which offloads a particular function like RAW Capture, YUV Capture, filter etc. On DM365 all the sensors, Video Decoders are media entities and the core itself has been modeled with CCDC, Previewer, Resizer, H3A, AEW as entities. These could be enumerated in the standard V4L2 way using the Media device as the enumerating device. Each of these entities has one or more input and output pads, and is connect-able to another entity through a 'link', between the pads.

Sub-devices

Conceptually similar to a media Entity, a subdevice is viewed as sub-block of a **V4L2 Video device which is independently configurable through its own set of file operations. The file operations are exposed through V4L2 -like IOCTLS particular to subdevs.Each sub-device is exposed to the user level through device files starting with "/dev/v4l-subdev-*"**. User applications need to configure V4L2 related settings like format, crop,, size parameters through these device handles for each of the sub-devices to make the work in tandem. Structurally, there is almost an one-to-one correspondence between a Media Entity and a sub-device.



Pads

"Pads" are input and output connect-able points of a Media Entity. Depending on the number of connections the entity can have the pads are pre-fixed in the driver. Typically, a device like a sensor or a video decoder would have only an output pad since it only feeds video into the system, and a

/dev/video pad would be modeled as an input pad since it is the end of the stream. The other entities like Resizer, previewer would have typically an input and an output pad and sometimes more depending on the capability.

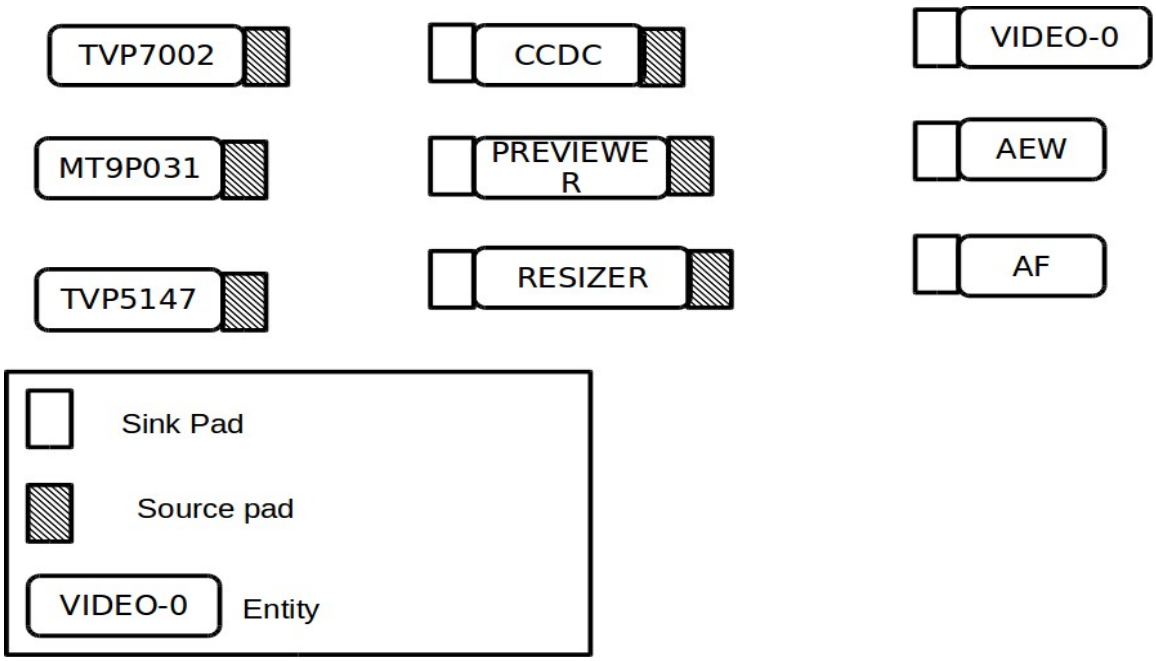
Link

A link is a 'connection' between pads of different entities. **These links can be set, get and enumerated through the Media Device. The application, for proper working of a driver is responsible for setting up of the links properly so that the driver understands the source and destination of the video data**.

Entity Graph

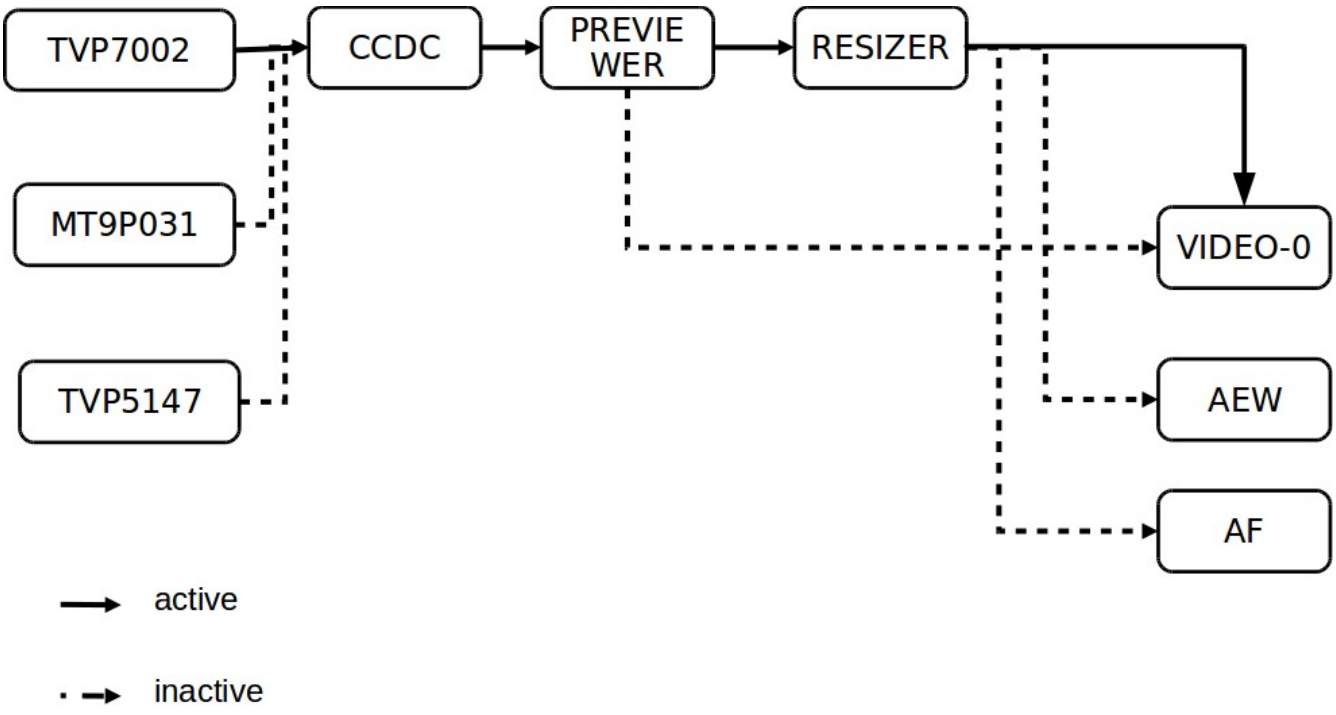
An entity graph is the complete setup of **different entities, pads, and the links**. For proper working of the software, an entity graph should be properly setup, and the driver, before it can start streaming will validate for the proper graph and the appropriate settings on the sub-device to get to know the intent of the application. Choice of if the video input is RAW BAYER or YUV video is determined by the appropriate setup of the entity graph.

The DM365 supports the following fixed entity and pad configuration.



Continuous Mode Operation

Continuous Mode refers to the configuration where the input data stream is regulated by the standard capture format like NTSC/720p etc and where the data is input from an external source t be stored int DDR. Here the input data is “streamed” to the DDR and thereby the exchange of buffers between drivers and applications happen on a continuous basis regulated by the interrupts generated at every field or frame. So, the sink is invariably a video node whereas the source is an external sensor or decoder or an inbuilt ADC. The streaming mode supported here is the standard V4L2 mode of streaming.



v4l2 subdev : media entity : init

static const struct media_entity_operations **xsdirxss_media_ops** = {

.link_validate = v4l2_subdev_link_validate

};

/* Initialize V4L2 subdevice and media entity */

pad.flags = **MEDIA_PAD_FL_SOURCE;**

subdev->entity.ops = &**xsdirxss_media_ops**;

ret = **media_entity_pads_init**(&subdev->entity, 1, &pad);

video device: media device: init

```
media_dev.dev = xdev->dev;
strlcpy(media_dev.model, "Xilinx Video Composite Device",
        sizeof(media_dev.model));
media_dev.hw_revision = 0;

media_device_init(media_dev);

xdev->v4l2_dev.mdev = media_dev;
ret = v4l2_device_register(xdev->dev, &xdev->v4l2_dev);
```

media device: application

open media device

```
media_fd = open("/dev/media0", O_RDWR);
```

enumerate media-entities

```
struct media_entity_desc entity[ENTITY_COUNT];

        /* enumerate media-entities */
        printf("enumerating media entities\n");
        index = 0;
        do {
                memset(&entity[index], 0, sizeof(struct media_entity_desc));
                entity[index].id = index | MEDIA_ENT_ID_FLAG_NEXT;

                ret = ioctl(media_fd, MEDIA_IOC_ENUM_ENTITIES, &entity[index]);
                if (ret < 0) {
                        if (errno == EINVAL)
                                break;
                }else {
                        if (!strcmp(entity[index].name, E_VIDEO_CCDC_OUT_NAME)) {
                                E_VIDEO =  entity[index].id;
                        }
                        else if (!strcmp(entity[index].name, E_MT9P031_NAME)) {
                                E_MT9P031 =  entity[index].id;
                        }
                        else if (!strcmp(entity[index].name, E_CCDC_NAME)) {
                                E_CCDC =  entity[index].id;
                        }
                        printf("[%x]:%s\n", entity[index].id, entity[index].name);
                }
```

```
        index++;
    } while (ret == 0 && index < ENTITY_COUNT);
    entities_count = index;
    printf("total number of entities:%x\n", entities_count);
```

enumerate links for each entity

```
    printf("5.enumerating links/pads for entities\n");

    links.pads = malloc(sizeof( struct media_pad_desc) * entity[index].pads);
    links.links = malloc(sizeof(struct media_link_desc) * entity[index].links);

    for(index = 0; index < entities_count; index++) {

        links.entity = entity[index].id;

        ret = ioctl(media_fd, MEDIA_IOC_ENUM_LINKS, &links);
        if (ret < 0) {
            if (errno == EINVAL)
                break;
        }else{
            /* display pads info first */
            if(entity[index].pads)
                printf("pads for entity %x=", entity[index].id);

            for(i = 0;i< entity[index].pads; i++)
            {
                printf("(%x, %s) ", links.pads->index,(links.pads->flags &
MEDIA_PAD_FL_INPUT)?"INPUT":"OUTPUT");
                links.pads++;
            }

            printf("\n");

            /* display links now */
            for(i = 0;i< entity[index].links; i++)
            {
                printf("[%x:%x]---------->[%x:%x]",links.links-
>source.entity,
                    links.links->source.index,links.links-
>sink.entity,links.links->sink.index);
                    if(links.links->flags & MEDIA_LNK_FL_ENABLED)
                        printf("\tACTIVE\n");
                    else
                        printf("\tINACTIVE \n");
```

```
                links.links++;
            }

            printf("\n");
        }
    }
```

enable 'mt9p031-->ccdc' link

```
    printf("6. ENABLEing link [tvp7002]----------->[ccdc]\n");
    memset(&link, 0, sizeof(link));

    link.flags |=  MEDIA_LNK_FL_ENABLED;
    link.source.entity = E_MT9P031;
    link.source.index = P_MT9P031;
    link.source.flags = MEDIA_PAD_FL_OUTPUT;

    link.sink.entity = E_CCDC;
    link.sink.index = P_CCDC_SINK;
    link.sink.flags = MEDIA_PAD_FL_INPUT;

    ret = ioctl(media_fd, MEDIA_IOC_SETUP_LINK, &link);
    if(ret) {
        printf("failed to enable link between tvp7002 and ccdc\n");
        goto cleanup;
    } else
        printf("[tvp7002]----------->[ccdc]\tENABLED\n");
```

enable 'ccdc->memory' link

```
    printf("7. ENABLEing link [ccdc]----------->[video_node]\n");
    memset(&link, 0, sizeof(link));

    link.flags |=  MEDIA_LNK_FL_ENABLED;
    link.source.entity = E_CCDC;
    link.source.index = P_CCDC_SOURCE;
    link.source.flags = MEDIA_PAD_FL_OUTPUT;

    link.sink.entity = E_VIDEO;
    link.sink.index = P_VIDEO;
    link.sink.flags = MEDIA_PAD_FL_INPUT;

    ret = ioctl(media_fd, MEDIA_IOC_SETUP_LINK, &link);
    if(ret) {
```

```
        printf("failed to enable link between ccdc and video node\n");
        goto cleanup;
    } else
        printf("[ccdc]---------->[video_node]\t ENABLED\n");

    printf("*********************************************\n");
```

Setting up ISP pipeline

Enumerate Media Entities and print the Media Device topology (*/dev/media0* by default):
media-ctl -p

This section describes how to translate valid ISP pipelines into links between Media Entities using ***media-ctl*** utility.

Following example demonstrates setting ISP pipeline for MT9T031 sensor to **Sensor->CCDC->Preview->Memory** and configuring entity pads formats.
Set up ISP pipeline:

media-ctl -r -l '"mt9t001 3-005d":0->"OMAP3 ISP CCDC":0[1], "OMAP3 ISP CCDC":2->"OMAP3 ISP preview":0[1], "OMAP3 ISP preview":1->"OMAP3 ISP preview output":0[1]'

-r	Reset all links to inactive
-l	Set up links by comma-separated list of links descriptors
"mt9t001 3-005d":0->"OMAP3 ISP CCDC":0[1]	Link output pad number 0 of Camera sensor to CCDC input pad number 1 and set this link active
"OMAP3 ISP CCDC":2->"OMAP3 ISP preview":0[1]	Link output pad number 2 of CCDC to Preview input pad number 0 and set this link active
"OMAP3 ISP preview":1->"OMAP3 ISP preview output":0[1]	Link output pad number 1 of Preview to Preview Output input pad number 0 and set this link active

Configure pads formats:

```
media-ctl -f '"mt9t001 3-005d":0 [SGRBG10 2048x1536 (32,20)/2048x1536], "OMAP3 ISP CCDC":2 [SGRBG10 2048x1536],  "OMAP3 ISP preview":1 [UYVY 2048x1536]'
```

-f	Set up pads formats by comma-separated list of formats descriptors
"mt9t001 3-005d":0 [SGRBG10 2048x1536 (32,20)/2048x1536]	Set up Camera sensor pad number 0 format to RAW Bayer 10bit image with resolution 2048x1536. Set maximum allowed sensor window width by specifying crop rectangle.

"OMAP3 ISP CCDC":2 [SGRBG10 2048x1536]	Set up CCDC pad number 2 format to RAW Bayer 10bit image with resolution 2048x1536.
"OMAP3 ISP preview":1 [UYVY 2048x1536]	Set up Preview pad number 1 format to YUV4:2:2 image with resolution 2048x1536.

Linux: v4l2 subdev

struct v4l2_subdev

Many drivers need to communicate with sub-devices. These devices can do all sort of tasks, but most commonly they handle audio and/or video muxing, encoding or decoding. For webcams common sub-devices are sensors and camera controllers.

Usually these are I2C devices, but not necessarily. In order to provide the driver with a consistent interface to these sub-devices the v4l2_subdev struct (v4l2-subdev.h) was created.

Each sub-device driver must have a v4l2_subdev struct. This struct can be stand-alone for simple sub-devices or it might be embedded in a larger struct if more state information needs to be stored. Usually there is a low-level device struct (e.g. i2c_client) that contains the device data as setup by the kernel. It is recommended to store that pointer in the private data of v4l2_subdev using v4l2_set_subdevdata(). That makes it easy to go from a v4l2_subdev to the actual low-level bus-specific device data.

You also need a way to go from the low-level struct to v4l2_subdev. For the common i2c_client struct the i2c_set_clientdata() call is used to store a v4l2_subdev pointer, for other busses you may have to use other methods.

Bridges might also need to store per-subdev private data, such as a pointer to bridge-specific per-subdev private data. The v4l2_subdev structure provides host private data for that purpose that can be accessed with v4l2_get_subdev_hostdata() and v4l2_set_subdev_hostdata().

From the bridge driver perspective you load the sub-device module and somehow obtain the v4l2_subdev pointer. For i2c devices this is easy: you call i2c_get_clientdata(). For other busses something similar needs to be done. Helper functions exists for sub-devices on an I2C bus that do most of this tricky work for you.

Each v4l2_subdev contains function pointers that sub-device drivers can

implement (or leave NULL if it is not applicable). Since sub-devices can do so many different things and you do not want to end up with a huge ops struct of which only a handful of ops are commonly implemented, the function pointers are sorted according to category and each category has its own ops struct.

The top-level ops struct contains pointers to the category ops structs, which may be NULL if the subdev driver does not support anything from that category.

It looks like this:

```
struct v4l2_subdev_core_ops {
        int (*log_status)(struct v4l2_subdev *sd);
        int (*init)(struct v4l2_subdev *sd, u32 val);
        ...
};

struct v4l2_subdev_tuner_ops {
        ...
};

struct v4l2_subdev_audio_ops {
        ...
};

struct v4l2_subdev_video_ops {
        ...
};

struct v4l2_subdev_pad_ops {
        ...
};
```

struct v4l2_subdev_ops

```
{
        const struct v4l2_subdev_core_ops  *core;
        const struct v4l2_subdev_tuner_ops *tuner;
        const struct v4l2_subdev_audio_ops *audio;
        const struct v4l2_subdev_video_ops *video;
        const struct v4l2_subdev_pad_ops *video;
};
```

The core ops are common to all subdevs, the other categories are implemented depending on the sub-device. E.g. a video device is unlikely to support the audio ops and vice versa.

This setup limits the number of function pointers while still making it easy to add new ops and categories.

A sub-device driver initializes the v4l2_subdev struct using:

```
	v4l2_subdev_init(sd, &ops);
```

struct v4l2_subdev_internal_ops

```
{
	int (*registered)(struct v4l2_subdev *sd);
	void (*unregistered)(struct v4l2_subdev *sd);
	int (*open)(struct v4l2_subdev *sd, struct v4l2_subdev_fh *fh);
	int (*close)(struct v4l2_subdev *sd, struct v4l2_subdev_fh *fh);
};
```

```
	/* Initialize V4L2 subdevice and media entity */
	subdev = &xsdirxss->subdev;
	v4l2_subdev_init(subdev, &xsdirxss_ops);

	subdev->dev = &pdev->dev;
	subdev->internal_ops = &xsdirxss_internal_ops;
	strscpy(subdev->name, dev_name(&pdev->dev), sizeof(subdev->name));

	subdev->flags |= V4L2_SUBDEV_FL_HAS_EVENTS |
V4L2_SUBDEV_FL_HAS_DEVNODE;

	   ret = v4l2_async_register_subdev(subdev);
	if (ret < 0) {
		dev_err(&pdev->dev, "failed to register subdev\n");
		goto error;
	}
```