Table of Contents

Open Multimedia Applications Platform
(OMAP)

OMAP 3

The 3rd generation OMAP, the OMAP 3 **is broken into 3 distinct groups: the OMAP34x, the OMAP35x, and the OMAP36x.** OMAP34x and OMAP36x are distributed directly to large handset (such as cell phone) manufacturers. OMAP35x is a variant of OMAP34x intended for catalog distribution channels. The OMAP36x is a 45 nm version of the 65 nm OMAP34x with higher clock speed

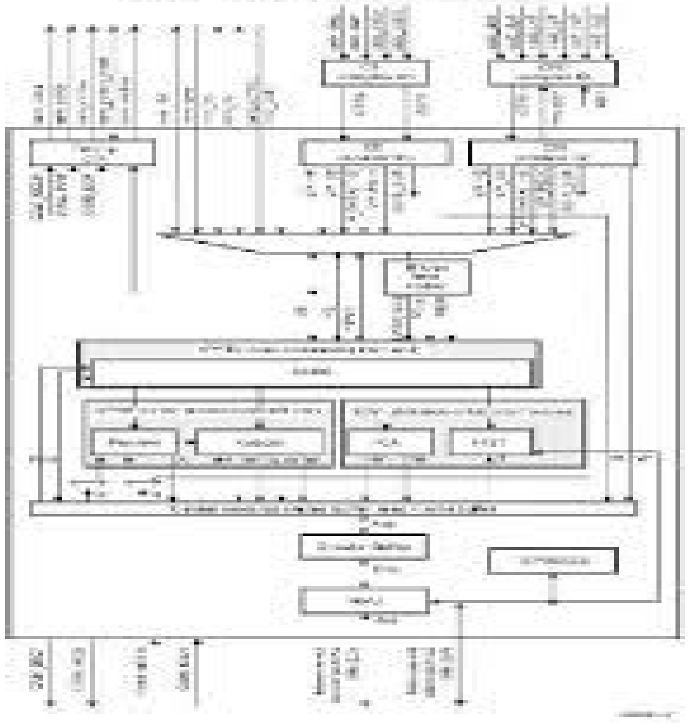
Camera ISP Overview

The camera ISP is a key component for imaging and video applications such as video preview, video record, and still-image capture with or without digital zooming.

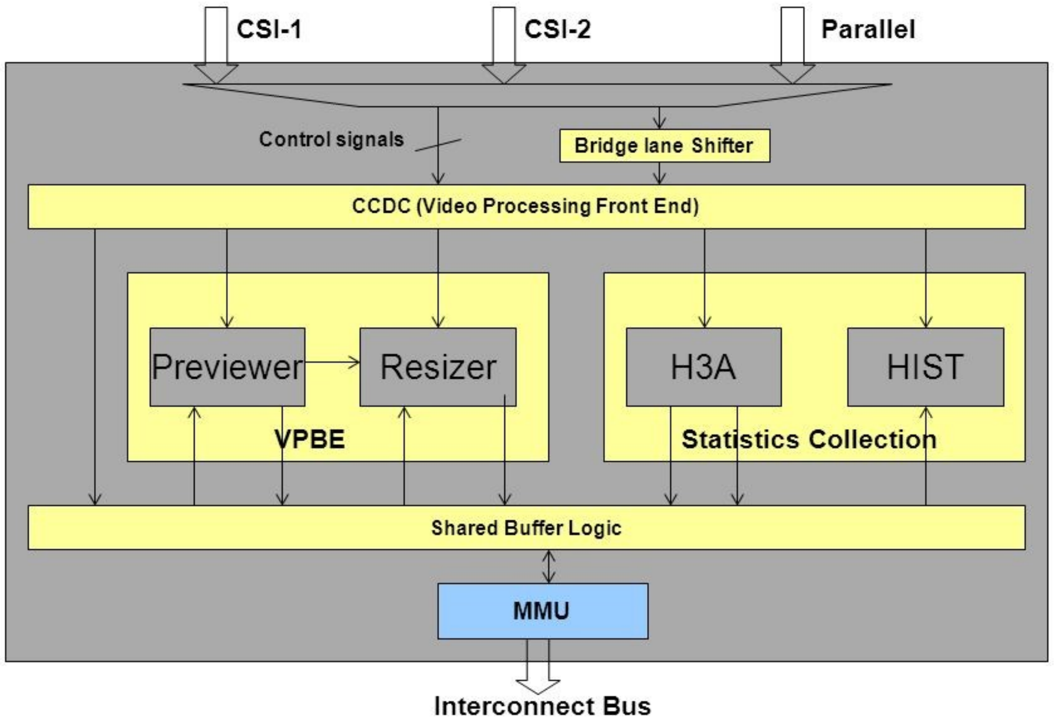
The camera ISP provides the system interface and the processing capability to connect RAW image- sensor modules to the device.

The camera I**SP supports one parallel interface (CPI) and two serial interfaces: MIPI® CSI1 and one MIPI CSI2 interface**, that can be active simultaneously. However, only one of these interfaces can use the video-processing hardware, when it is not used by the parallel interface. Because of a pin-muxing limitation, the parallel interface and CSI1 (CSIb) cannot be active at the same time.

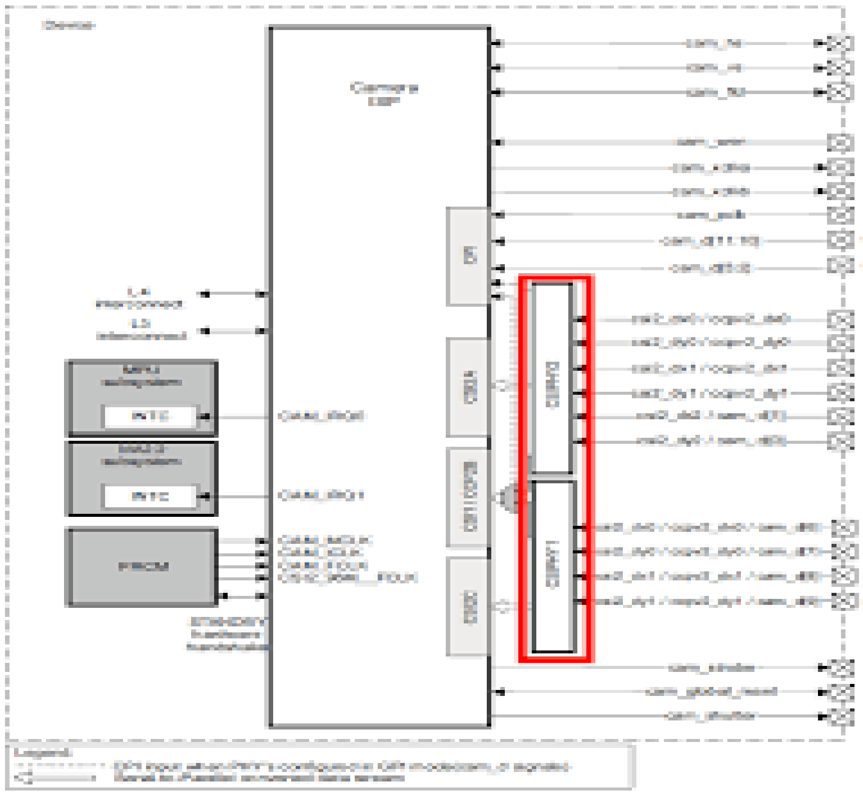
Figure 12-35. Camera ISP Block Diagram



TI OMAP3 ISP: Block Diagram



Courtesy of Texas Instruments



MIPI CSI2 serial interface:

The camera ISP supports one MIPI CSI2 serial interface (CSIa) **with 2 data lanes. MIPI CSI2 enables data transfer at up to 1.6G bps. It is based on the MIPI CSI2 Specification 1.0.**

– Transfer pixels and data received by the CSI2 DSI_PHY RX to the system memory or to the image pipe line
– Uses unidirectional data link
– Supports two data-configurable links, in addition to the clock signaling.
– **Maximum data rate of up to 800M bps per data lane**
– Data merger for two-data-lane configuration
– Error detection and correction by the protocol engine
– DMA engine integrated with dedicated FIFO
– 1-D and 2-D addressing mode (rotation is not supported by the 2D mode)

– Ping-pong mechanism for double buffering
– Burst support
– JPEG support for unknown length transfer
– **RGB, RAW, and YUV formats supported**
– Storage in progressive mode for interlaced stream (using line numbering)
– Conversion of the RGB formats

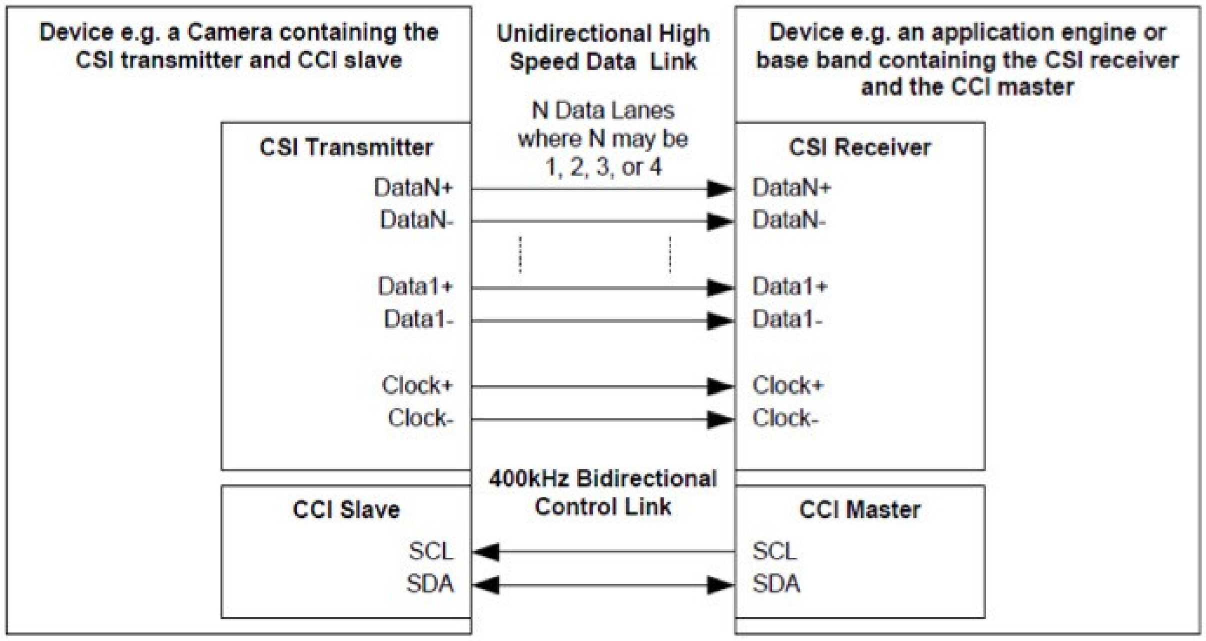
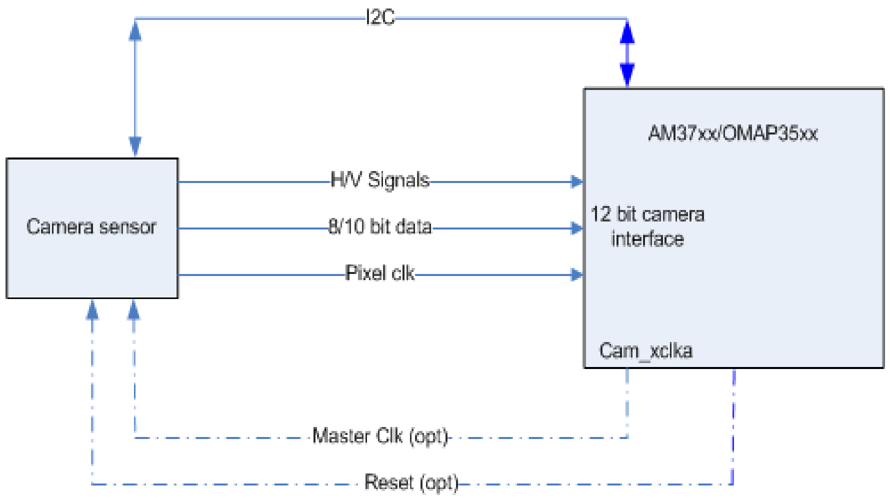

Device e.g. a Camera containing the CSI transmitter and CCI slave
Unidirectional High Speed Data Link
Device e.g. an application engine or base band containing the CSI receiver and the CCI master
N Data Lanes where N may be 1, 2, 3, or 4
CSI Transmitter
DataN+
DataN-
Data1+
Data1-
Clock+
Clock-
CSI Receiver
DataN+
DataN-
Data1+
Data1-
Clock+
Clock-
400kHz Bidirectional Control Link
CCI Slave
SCL
SDA
CCI Master
SCL
SDA

Figure 1 CSI-2 and CCI Transmitter and Receiver Interface

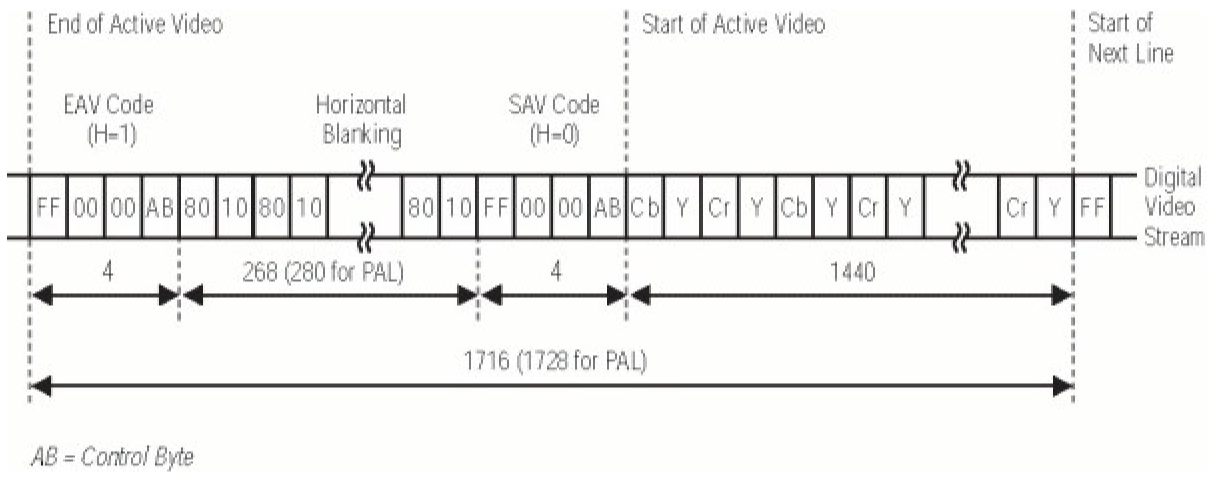
Parallel interface:

The camera parallel interface (CPI) supports two modes:

– SYNC mode: In this mode, the image-sensor module provides horizontal and vertical synchronization signals to the parallel interface, along with the pixel clock. This mode works with 8-, 10-, 11-, and 12-bit data (above 10-bit RAW data, the processing pipe cannot be used; data must be transferred to memory). SYNC mode supports progressive and interlaced image-sensor modules.



– **ITU mode:** In this mode, the image-sensor module provides an ITU-R BT 656-compatible data stream. **The horizontal and vertical synchronization signals are not provided to the interface. Instead, the data stream embeds start-of-active (SAV) and end-of-active video (EAV) synchronization code**. This mode works in 8- and 10-bit configurations.

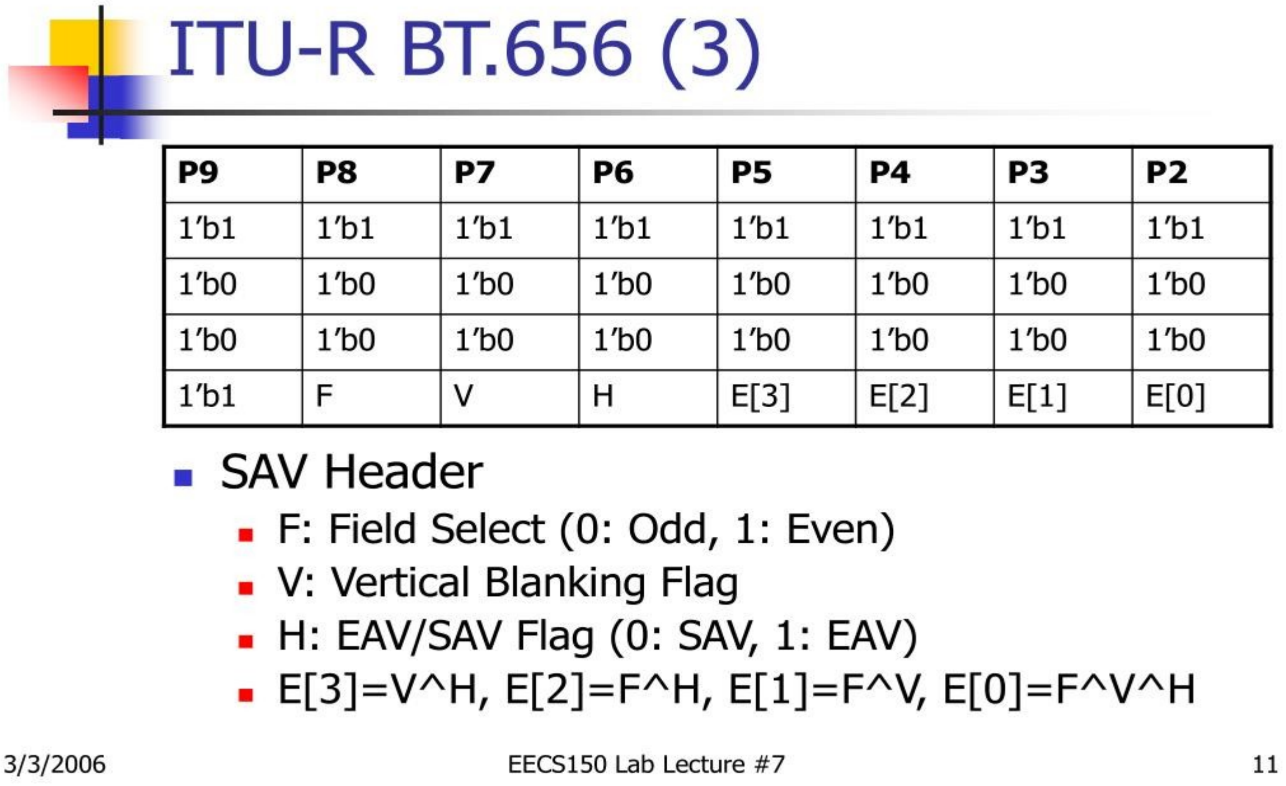


START OF ACTIVE FRAME (SAV)
END OF ACTIVE FRAME (EAV)

1 Byte	**2 Byte**	**3 Byte**	**4 Byte**
0xFF	0X00	0X00	**VARY**

4 Byte

Bit 9	1
Bit 8	F
Bit 7	V (V=0 START OF LINE) (V=1 END OF LINE)
Bit 6	H (H = 0 SAV) (H=1 EAV) FRAME
Bit 5	E[3]
Bit 4	E[2]
Bit 3	E[1]
Bit 2	E[0]
Bit 1	0
Bit 0	0



ITU-R BT.656 (3)

P9	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**
1’b1	1’b1	1’b1	1’b1	1’b1	1’b1	1’b1	1’b1
1’b0	1’b0	1’b0	1’b0	1’b0	1’b0	1’b0	1’b0
1’b0	1’b0	1’b0	1’b0	1’b0	1’b0	1’b0	1’b0
1’b1	F	V	H	E[3]	E[2]	E[1]	E[0]

- SAV Header
  - F: Field Select (0: Odd, 1: Even)
  - V: Vertical Blanking Flag
  - H: EAV/SAV Flag (0: SAV, 1: EAV)
  - E[3]=V^H, E[2]=F^H, E[1]=F^V, E[0]=F^V^H

3/3/2006 EECS150 Lab Lecture #7 11

Video processing:

The video-processing hardware removes the need for expensive camera modules to perform processing functions. The hardware pipeline contains **two parts: front end and back end.**

Video processing front end (VPFE):

CCDC:

Performs signal-processing operations on RAW image input data. The output data can go directly to memory for software processing, or to the video-processing back end for further processing. The video-processing front end is supported by the CCDC module.

Signal-processing operations include:
- Optical clamping
- Optical black clamp
- Black-level compensation
- Look-up table (LUT) based faulty pixel correction
- 2D lens-shading compensation
- Data formatter
- Output formatter

Image-Signal Processing (CCDC)

Digital clamp

Digital clamp is enabled only if optical black clamping is disabled: CCDC_CLAMP[31] CLAMPEN = 0. The digital clamp DC value to be subtracted from the raw image data.

Optical Black Clamping

Optical black clamping is enabled by setting CCDC_CLAMP[31] CLAMPEN to 1. When enabled, an average of black-pixel samples is computed over a window. If the height of the window is 2N, the average value multiplied by a programmable gain factor is subtracted from the raw image data for the following 2N lines. Every 2N lines, a new average value is computed. For the first 2N lines, 0 is subtracted.

Black Compensation

Black compensation applies an offset to the raw image data. The offset is applied according to the phase and color for each phase

Faulty-Pixel Correction

Faulty-pixel correction is enabled by setting the CCDC_FPC[15] FPCEN bit to 1. Before activating faulty- pixel correction, set the number of faulty pixels to be corrected in a frame with the CCDC_FPC[14:0] FPNUM bit field, set the faulty-pixel LUT in memory, and set the CCDC_FPC_ADDR register to the LUT address. The address should be aligned to a 64-bit byte boundary; the 6 LSBs are ignored. Reading the register always shows the 6 LSBs as 0.

If the CCDC module cannot fetch the required faulty-pixel entry in time, an error is set in the CCDC_FPC[16] FPERR bit. After the bit is set, no more faulty pixels are corrected in the frame. The bit is Automatically cleared on the end of the frame and the feature re enabled for the following frame.

Video processing back end (VPBE):

Performs signal-processing operations on RAW image input
data. Outputs YCbCr 4:2:2 data.

• Preview module: Signal-processing operations include:
• A-law decompression: transforms non-linear 8-bit data to 10-bit linear data. The CCDC module can perform A-law compression

- Noise reduction and faulty pixel correction
- Dark frame capture and subtraction
- Horizontal median filter
- Programmable filter: 3x3 kernel of the same color
- Couplet faulty pixel correction

• Digital gain
• White balance
• Programmable color filter array (CFA) interpolation: 5x5 kernel
• Black adjustment
• Programmable color correction (RGB to RGB)
• Programmable gamma correction: 1024 entries for each color

1. Programmable color conversion (RGB to YCbCr 4:4:4)

2. Color subsampling (YCbCr 4:4:4 to YCbCr 4:2:2)
3. Luminance enhancement (non-linear), chrominance suppression and offset
4. The preview module can also work from memory to memory.

Resizer module:

Performs on-the-fly up sampling (up to x4) and down sampling (down to x0.25) of YCbCr 4:2:2 data by applying high-quality horizontal and vertical filters. The horizontal and vertical resizer ratios are independent. Applicable ratios are 256/N, with N ranging from 64 to 1024. This feature enables digital zooming (upsampling) and video preview (downsampling).

The resizer module can also work from memory to memory. Higher or lower ratios can be obtained by combining on-the-fly resizing followed by memory-to-memory resizing.

Statistic collection modules (SCM):

The host CPU uses statistics to adjust various parameters for processing image data.

3A metrics:

Collects on-the-fly RAW image data metrics, which are required to perform the control loops for **auto white balance (AWB), auto exposure (AE), and autofocus (AF)**. The MPU subsystem typically uses data metrics to adjust various parameters for processing image data.

Histogram:

Performs on-the-fly pixel binning of RAW image, based on color value ranges and regions. Supports up to 4 regions and up to 256 bins per color. The MPU subsystem typically uses the histogram with 3A metrics to adjust various parameters for processing image data. The histogram module can also work from memory to memory.

Supporting /Common Controllers

Central-resource shared buffer logic (SBL):

Buffers and schedules memory accesses requested by camera ISP modules

Circular buffer:

Prevents storage of full image frames in memory when data must be post processed and/or preprocessed by software

Memory management unit (MMU):

Manages virtual-to-physical address translation for external addresses and solves the memory-fragmentation issue. Enables the camera driver to dynamically allocate and deallocate memory; the MMU handles memory fragmentation.

Clock generator:

Generates two independent clocks that can be used by two external image sensors

Timing control:

– Generation of two clocks that can be used by the external image sensors
– Generation of signals for strobe flash, mechanical shutter, and global reset. Support for red-eye removal.

OMAP3 ISP DRIVER

Introduction

This file documents the Texas Instruments OMAP 3 Image Signal Processor (ISP) driver located under **drivers/media/platform/omap3isp**. The original driver was written by Texas Instruments but since that it has been rewritten (twice) at Nokia.

The driver has been successfully used on the following versions of OMAP 3:

- 3430
- 3530
- 3630

/* ISP: OMAP 34xx ES 1.0 */ #define ISP_REVISION_1_0	0x10
/* ISP2: OMAP 34xx ES 2.0, 2.1 and 3.0 */ #define ISP_REVISION_2_0	0x20
/* ISP2P: OMAP 36xx */ #define ISP_REVISION_15_0	0xF0

The driver implements V4L2, Media controller and v4l2_subdev interfaces. Sensor, lens and flash drivers using the v4l2_subdev interface in the kernel are supported.

Split to subdevs

The OMAP 3 ISP is split into V4L2 subdevs, each of the blocks inside the ISP having one subdev to represent it. Each of the subdevs provide a V4L2 subdev interface to userspace.

OMAP3 ISP CCP2	ispccp2.c, ispccp2.h	Parallel Interface - (SYNC , ITU)
OMAP3 ISP CSI2a	ispcsi2.c, ispcsi2.h ispcsiphy.c, ispcsiphy.h	MIPI CSI 1/ CSI 2
OMAP3 ISP	ispccdc.c, ispccdc.h	VFED - CCDC

CCDC		
OMAP3 ISP preview	isppreview.c, isppreview.h	VBED - PREVIEW
OMAP3 ISP resizer	ispresizer.c, ispresizer.h	VBED – RE SIZER
OMAP3 ISP AEWB	isph3a_aewb.c,	3A
OMAP3 ISP AF	isph3a_af.c ,isph3a.h	3A
OMAP3 ISP histogram	isphist.c,isphist.h	Histogram
Host Controller – Start and Helper files	isp.c,isp.h ispvideo.c,ispvideo.h , ispstat.c,ispstat.h	

Each possible link in the ISP is modelled by a link in the Media controller interface

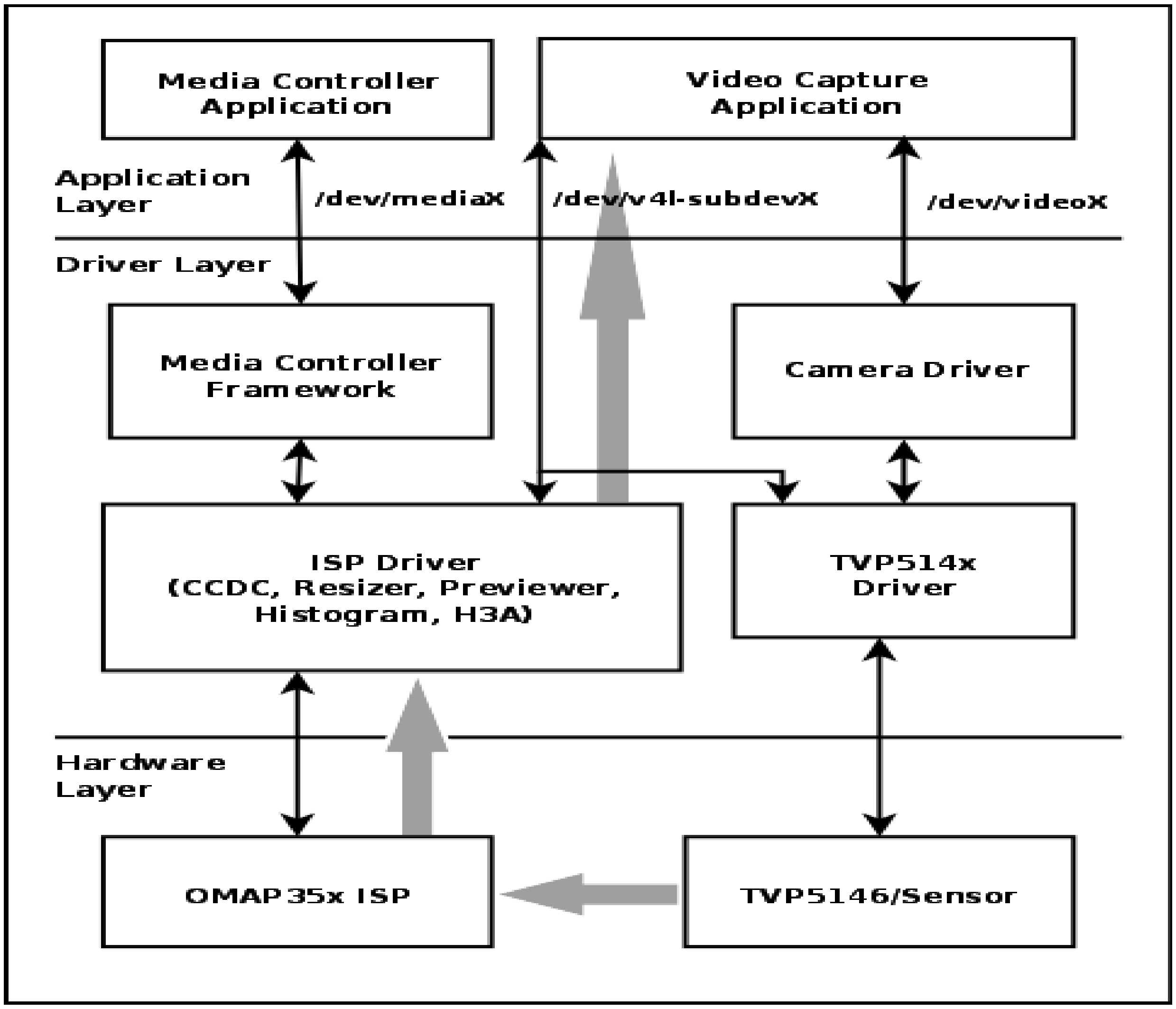
Module Name	**Base Address (hex)**	**Size**
ISP	0x480B C000	256 bytes
ISP_CBUFF	0x480B C100	256 bytes
ISP_CSI1B	0x480B C400	512 bytes
ISP_CCDC	0x480B C600	512 bytes
ISP_HIST	0x480B CA00	512 bytes
ISP_H3A	0x480B CC00	512 bytes
ISP_PREVIEW	0x480B CE00	512 bytes
ISP_RESIZER	0x480B D000	512 bytes
ISP_SBL	0x480B D200	512 bytes
ISP_CSI2A	0x480B D800	1024 bytes
CSI2PHY_SCP	0x480B D970	512 bytes

Controlling the OMAP 3 ISP

In general, the settings given to the OMAP 3 ISP take effect at the beginning of the following frame. This is done when the module becomes idle during the vertical blanking period on the sensor. In memory-to-memory operation the pipe is run one frame at a time. Applying the settings is done between the frames.

All the blocks in the ISP, excluding the CSI-2 and possibly the CCP2 receiver, insist on receiving complete frames. Sensors must thus never send the ISP partial frames.

Driver Flow



Step 1: omap3 isp platform driver

```
static const struct of_device_id omap3isp_of_table[] = {
	{ .compatible = "ti,omap3-isp" },
	{ },
};
MODULE_DEVICE_TABLE(of, omap3isp_of_table);

static struct platform_driver omap3isp_driver = {
	.probe = isp_probe,
	.remove = isp_remove,
	.id_table = omap3isp_id_table,
	.driver = {
		.name = "omap3isp",
		.pm	= &omap3isp_pm_ops,
		.of_match_table = omap3isp_of_table,
	},
};

module_platform_driver(omap3isp_driver);

MODULE_AUTHOR("Nokia Corporation");
MODULE_DESCRIPTION("TI OMAP3 ISP driver");
MODULE_LICENSE("GPL");
MODULE_VERSION(ISP_VIDEO_DRIVER_VERSION);
```

Step 2: isp_probe

```
static int isp_probe(struct platform_device *pdev)
{
	struct isp_device *isp;
	struct resource *mem;
	int ret;
	int i, m;
	isp = devm_kzalloc(&pdev->dev, sizeof(*isp), GFP_KERNEL);
	if (!isp) {
		dev_err(&pdev->dev, "could not allocate memory\n");
		return -ENOMEM;
	}
	 ------------------------------------------------------------
	------------------------------------------------------------
	/* Entities */
	ret = isp_initialize_modules(isp);
	if (ret < 0)
		goto error_iommu;

	ret = isp_register_entities(isp);
	if (ret < 0)
		goto error_modules;

	ret = isp_create_links(isp);
	if (ret < 0)
		goto error_register_entities;

	isp->notifier.ops = &isp_subdev_notifier_ops;

	ret = v4l2_async_notifier_register(&isp->v4l2_dev, &isp-
>notifier);
	if (ret)
		goto error_register_entities;

	isp_core_init(isp, 1);
	omap3isp_put(isp);

	return 0;
}
```

step 3: isp_initialize_modules

```
static int isp_initialize_modules(struct isp_device *isp)
{
	int ret;

	ret = omap3isp_csiphy_init(isp);
```

```
if (ret < 0) {
        dev_err(isp->dev, "CSI PHY initialization failed\n");
        return ret;
}

ret = omap3isp_csi2_init(isp);
if (ret < 0) {
        dev_err(isp->dev, "CSI2 initialization failed\n");
        goto error_csi2;
}

ret = omap3isp_ccp2_init(isp);
if (ret < 0) {
        if (ret != -EPROBE_DEFER)
                dev_err(isp->dev, "CCP2 initialization failed\n");
        goto error_ccp2;
}

ret = omap3isp_ccdc_init(isp);
if (ret < 0) {
        dev_err(isp->dev, "CCDC initialization failed\n");
        goto error_ccdc;
}

ret = omap3isp_preview_init(isp);
if (ret < 0) {
        dev_err(isp->dev, "Preview initialization failed\n");
        goto error_preview;
}
ret = omap3isp_resizer_init(isp);
if (ret < 0) {
        dev_err(isp->dev, "Resizer initialization failed\n");
        goto error_resizer;
}

ret = omap3isp_hist_init(isp);
if (ret < 0) {
        dev_err(isp->dev, "Histogram initialization failed\n");
        goto error_hist;
}

ret = omap3isp_h3a_aewb_init(isp);
if (ret < 0) {
        dev_err(isp->dev, "H3A AEWB initialization failed\n");
```

```
                goto error_h3a_aewb;
        }

        ret = omap3isp_h3a_af_init(isp);
        if (ret < 0) {
                dev_err(isp->dev, "H3A AF initialization failed\n");
                goto error_h3a_af;
        }

        return 0;
}
```

step 4: isp_register_entities

```
static int isp_register_entities(struct isp_device *isp)
{
        int ret;

        isp->media_dev.dev = isp->dev;
        strlcpy(isp->media_dev.model, "TI OMAP3 ISP",
                sizeof(isp->media_dev.model));
        isp->media_dev.hw_revision = isp->revision;
        isp->media_dev.ops = &isp_media_ops;
        media_device_init(&isp->media_dev);

        isp->v4l2_dev.mdev = &isp->media_dev;

        ret = v4l2_device_register(isp->dev, &isp->v4l2_dev);
        if (ret < 0) {
                dev_err(isp->dev, "%s: V4L2 device registration failed (%d)\n",
                        __func__, ret);
                goto done;
        }

        /* Register internal entities */
        ret = omap3isp_ccp2_register_entities(&isp->isp_ccp2, &isp-
>v4l2_dev);
        if (ret < 0)
                goto done;

        ret = omap3isp_csi2_register_entities(&isp->isp_csi2a, &isp-
>v4l2_dev);
```

```
        if (ret < 0)
                goto done;

        ret = omap3isp_ccdc_register_entities(&isp->isp_ccdc, &isp-
>v4l2_dev);
        if (ret < 0)
                goto done;

        ret = omap3isp_preview_register_entities(&isp->isp_prev,
                                              &isp->v4l2_dev);
        if (ret < 0)
                goto done;

        ret = omap3isp_resizer_register_entities(&isp->isp_res, &isp-
>v4l2_dev);
        if (ret < 0)
                goto done;

        ret = omap3isp_stat_register_entities(&isp->isp_aewb, &isp-
>v4l2_dev);
        if (ret < 0)
                goto done;

        ret = omap3isp_stat_register_entities(&isp->isp_af, &isp-
>v4l2_dev);
        if (ret < 0)
                goto done;

        ret = omap3isp_stat_register_entities(&isp->isp_hist, &isp-
>v4l2_dev);
        if (ret < 0)
                goto done;

done:
        if (ret < 0)
                isp_unregister_entities(isp);

        return ret;
}
```

step 5: isp_create_links

```
static int isp_create_links(struct isp_device *isp)
{
        int ret;

        /* Create links between entities and video nodes. */
         ret = media_create_pad_link(
                        &isp->isp_csi2a.subdev.entity, CSI2_PAD_SOURCE,
                        &isp->isp_csi2a.video_out.video.entity, 0, 0);
        if (ret < 0)
                return ret;

        ret = media_create_pad_link(
                        &isp->isp_ccp2.video_in.video.entity, 0,
                        &isp->isp_ccp2.subdev.entity, CCP2_PAD_SINK, 0);
        if (ret < 0)
                return ret;

        ret = media_create_pad_link(
                        &isp->isp_ccdc.subdev.entity, CCDC_PAD_SOURCE_OF,
                        &isp->isp_ccdc.video_out.video.entity, 0, 0);
        if (ret < 0)
                return ret;

         ret = media_create_pad_link(
                        &isp->isp_prev.video_in.video.entity, 0,
                        &isp->isp_prev.subdev.entity, PREV_PAD_SINK, 0);
        if (ret < 0)
                return ret;

         ret = media_create_pad_link(
                        &isp->isp_prev.subdev.entity, PREV_PAD_SOURCE,
                        &isp->isp_prev.video_out.video.entity, 0, 0);
        if (ret < 0)
                return ret;

         ret = media_create_pad_link(
                        &isp->isp_res.video_in.video.entity, 0,
                        &isp->isp_res.subdev.entity, RESZ_PAD_SINK, 0);
        if (ret < 0)
                return ret;
        ret = media_create_pad_link(
                        &isp->isp_res.subdev.entity, RESZ_PAD_SOURCE,
                        &isp->isp_res.video_out.video.entity, 0, 0);
```

```
if (ret < 0)
        return ret;

/* Create links between entities. */
 ret = media_create_pad_link(
                &isp->isp_csi2a.subdev.entity, CSI2_PAD_SOURCE,
                &isp->isp_ccdc.subdev.entity, CCDC_PAD_SINK, 0);
if (ret < 0)
        return ret;

 ret = media_create_pad_link(
                &isp->isp_ccp2.subdev.entity, CCP2_PAD_SOURCE,
                &isp->isp_ccdc.subdev.entity, CCDC_PAD_SINK, 0);
if (ret < 0)
        return ret;

ret = media_create_pad_link(
                &isp->isp_ccdc.subdev.entity, CCDC_PAD_SOURCE_VP,
                &isp->isp_prev.subdev.entity, PREV_PAD_SINK, 0);
if (ret < 0)
        return ret;

ret = media_create_pad_link(
                &isp->isp_ccdc.subdev.entity, CCDC_PAD_SOURCE_OF,
                &isp->isp_res.subdev.entity, RESZ_PAD_SINK, 0);
if (ret < 0)
        return ret;

ret = media_create_pad_link(
                &isp->isp_prev.subdev.entity, PREV_PAD_SOURCE,
                &isp->isp_res.subdev.entity, RESZ_PAD_SINK, 0);
if (ret < 0)
        return ret;

ret = media_create_pad_link(
                &isp->isp_ccdc.subdev.entity, CCDC_PAD_SOURCE_VP,
                &isp->isp_aewb.subdev.entity, 0,
                MEDIA_LNK_FL_ENABLED | MEDIA_LNK_FL_IMMUTABLE);
if (ret < 0)
        return ret;

ret = media_create_pad_link(
                &isp->isp_ccdc.subdev.entity, CCDC_PAD_SOURCE_VP,
                &isp->isp_af.subdev.entity, 0,
                MEDIA_LNK_FL_ENABLED | MEDIA_LNK_FL_IMMUTABLE);
        if (ret < 0)
                return ret;

        ret = media_create_pad_link(
                &isp->isp_ccdc.subdev.entity, CCDC_PAD_SOURCE_VP,
                &isp->isp_hist.subdev.entity, 0,
                MEDIA_LNK_FL_ENABLED | MEDIA_LNK_FL_IMMUTABLE);
        if (ret < 0)
                return ret;

        return 0;
}
```

step 6: v4l2_async_notifier_register

```
/**
 * struct v4l2_async_notifier_operations - Asynchronous V4L2 notifier
operations
 * @bound:      a subdevice driver has successfully probed one of the
subdevices
 * @complete:   All subdevices have been probed successfully. The
complete
 *              callback is only executed for the root notifier.
 * @unbind:     a subdevice is leaving
 */

struct v4l2_async_notifier_operations {
        int (*bound)(struct v4l2_async_notifier *notifier,
                     struct v4l2_subdev *subdev,
                     struct v4l2_async_subdev *asd);
        int (*complete)(struct v4l2_async_notifier *notifier);
        void (*unbind)(struct v4l2_async_notifier *notifier,
                       struct v4l2_subdev *subdev,
                       struct v4l2_async_subdev *asd);
};


static const struct v4l2_async_notifier_operations isp_subdev_notifier_ops =
{
        .complete = isp_subdev_notifier_complete,
};


static int isp_subdev_notifier_complete(struct v4l2_async_notifier
*async)
{
        struct isp_device *isp = container_of(async, struct isp_device,
                                              notifier);
        struct v4l2_device *v4l2_dev = &isp->v4l2_dev;
        struct v4l2_subdev *sd;
        int ret;

        ret = media_entity_enum_init(&isp->crashed, &isp->media_dev);
        if (ret)
                return ret;

        list_for_each_entry(sd, &v4l2_dev->subdevs, list) {
                if (sd->notifier != &isp->notifier)
                        continue;

                ret = isp_link_entity(isp, &sd->entity,
                                      v4l2_subdev_to_bus_cfg(sd)->interface);
                if (ret < 0)
                        return ret;
        }

        ret = v4l2_device_register_subdev_nodes(&isp->v4l2_dev);
        if (ret < 0)
                return ret;

        return media_device_register(&isp->media_dev);
}
```

step 7: struct isp_device , struct isp_video

```
struct isp_device {
        struct v4l2_device v4l2_dev;
        struct v4l2_async_notifier notifier;
        struct media_device media_dev;
        struct device *dev;
        u32 revision;
         ---------------------------------------------
         ---------------------------------------------
       /* ISP modules */
        struct ispstat isp_af;
        struct ispstat isp_aewb;
        struct ispstat isp_hist;
        struct isp_res_device isp_res;
        struct isp_prev_device isp_prev;
        struct isp_ccdc_device isp_ccdc;
        struct isp_csi2_device isp_csi2a;
        struct isp_csi2_device isp_csi2c;
        struct isp_ccp2_device isp_ccp2;
        struct isp_csiphy isp_csiphy1;
        struct isp_csiphy isp_csiphy2;

        unsigned int sbl_resources;
        unsigned int subclk_resources;

    /* Video buffers queue */
        struct vb2_queue *queue;
        struct mutex queue_lock;        /* protects the queue */
        spinlock_t irqlock;             /* protects dmaqueue */
        struct list_head dmaqueue;
        enum isp_video_dmaqueue_flags dmaqueue_flags;


};



struct isp_video_operations {
        int(*queue)(struct isp_video *video, struct isp_buffer *buffer);
};

struct isp_video {
        struct video_device video;
        enum v4l2_buf_type type;
        struct media_pad pad;

        struct mutex mutex;             /* format and crop settings */
        atomic_t active;
```

```
	struct isp_device *isp;

	unsigned int capture_mem;
	unsigned int bpl_alignment;	/* alignment value */
	unsigned int bpl_zero_padding;	/* whether the alignment is optional */
	unsigned int bpl_max;		/* maximum bytes per line value */
	unsigned int bpl_value;		/* bytes per line value */
	unsigned int bpl_padding;	/* padding at end of line */

	/* Pipeline state */
	struct isp_pipeline pipe;
	struct mutex stream_lock;	/* pipeline and stream states */
	bool error;

	 /* Video buffers queue */
	 struct vb2_queue *queue;
	struct mutex queue_lock;	/* protects the queue */
	spinlock_t irqlock;		/* protects dmaqueue */
	struct list_head dmaqueue;
	enum isp_video_dmaqueue_flags dmaqueue_flags;

	const struct isp_video_operations *ops;
};
```

step 8: isp_initialize_modules: omap3isp_csiphy_init

```
/*
 * omap3isp_csiphy_init - Initialize the CSI PHY frontends
 */
int omap3isp_csiphy_init(struct isp_device *isp)
{
	struct isp_csiphy *phy1 = &isp->isp_csiphy1;
	struct isp_csiphy *phy2 = &isp->isp_csiphy2;

	phy2->isp = isp;
	phy2->csi2 = &isp->isp_csi2a;
	phy2->num_data_lanes = ISP_CSIPHY2_NUM_DATA_LANES;
	phy2->cfg_regs = OMAP3_ISP_IOMEM_CSI2A_REGS1;
	phy2->phy_regs = OMAP3_ISP_IOMEM_CSIPHY2;
	mutex_init(&phy2->mutex);

	phy1->isp = isp;
	mutex_init(&phy1->mutex);

	if (isp->revision == ISP_REVISION_15_0) {
		phy1->csi2 = &isp->isp_csi2c;
		phy1->num_data_lanes = ISP_CSIPHY1_NUM_DATA_LANES;
		phy1->cfg_regs = OMAP3_ISP_IOMEM_CSI2C_REGS1;
		phy1->phy_regs = OMAP3_ISP_IOMEM_CSIPHY1;
	}

	return 0;
}
```

step 9: isp_initialize_modules: omap3isp_csi2_init

```
/*
 * omap3isp_csi2_init - Routine for module driver init
 */
int omap3isp_csi2_init(struct isp_device *isp)
{
	struct isp_csi2_device *csi2a = &isp->isp_csi2a;
	struct isp_csi2_device *csi2c = &isp->isp_csi2c;
	int ret;

	csi2a->isp = isp;
	csi2a->available = 1;
	csi2a->regs1 = OMAP3_ISP_IOMEM_CSI2A_REGS1;
	csi2a->regs2 = OMAP3_ISP_IOMEM_CSI2A_REGS2;
	csi2a->phy = &isp->isp_csiphy2;
	csi2a->state = ISP_PIPELINE_STREAM_STOPPED;
	init_waitqueue_head(&csi2a->wait);

	ret = csi2_init_entities(csi2a);
	if (ret < 0)
		return ret;

	if (isp->revision == ISP_REVISION_15_0) {
		csi2c->isp = isp;
		csi2c->available = 1;
		csi2c->regs1 = OMAP3_ISP_IOMEM_CSI2C_REGS1;
		csi2c->regs2 = OMAP3_ISP_IOMEM_CSI2C_REGS2;
		csi2c->phy = &isp->isp_csiphy1;
		csi2c->state = ISP_PIPELINE_STREAM_STOPPED;
		init_waitqueue_head(&csi2c->wait);
	}

	return 0;
}
```

step 10: i**sp_initialize_modules: omap3isp_csi2_init:
csi2_init_entities**

```
static int csi2_init_entities(struct isp_csi2_device *csi2)
{
	struct v4l2_subdev *sd = &csi2->subdev;
	struct media_pad *pads = csi2->pads;
	struct media_entity *me = &sd->entity;
	int ret;

	v4l2_subdev_init(sd, &csi2_ops);
	sd->internal_ops = &csi2_internal_ops;
	strlcpy(sd->name, "OMAP3 ISP CSI2a", sizeof(sd->name));

	sd->grp_id = 1 << 16;	/* group ID for isp subdevs */
	 v4l2_set_subdevdata(sd, csi2);
	 sd->flags |= V4L2_SUBDEV_FL_HAS_DEVNODE;

	pads[CSI2_PAD_SOURCE].flags = MEDIA_PAD_FL_SOURCE;
	pads[CSI2_PAD_SINK].flags = MEDIA_PAD_FL_SINK
				    | MEDIA_PAD_FL_MUST_CONNECT;


	 me->ops = &csi2_media_ops;
	ret = media_entity_pads_init(me, CSI2_PADS_NUM, pads);
	if (ret < 0)
		return ret;

	csi2_init_formats(sd, NULL);

	/* Video device node */
	csi2->video_out.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
	csi2->video_out.ops = &csi2_ispvideo_ops;
	csi2->video_out.bpl_alignment = 32;
	csi2->video_out.bpl_zero_padding = 1;
	csi2->video_out.bpl_max = 0x1ffe0;
```

```
        csi2->video_out.isp = csi2->isp;
        csi2->video_out.capture_mem = PAGE_ALIGN(4096 * 4096) * 3;

        ret = omap3isp_video_init(&csi2->video_out, "CSI2a");
        if (ret < 0)
                goto error_video;

        return 0;

error_video:
        media_entity_cleanup(&csi2->subdev.entity);
        return ret;
}
```

step 11: isp_initialize_modules: omap3isp_csi2_init: csi2_init_entities: omap3isp_video_init

```
int omap3isp_video_init(struct isp_video *video, const char *name)
{
        const char *direction;
        int ret;

        switch (video->type) {
        case V4L2_BUF_TYPE_VIDEO_CAPTURE:
                direction = "output";
                video->pad.flags = MEDIA_PAD_FL_SINK
                                 | MEDIA_PAD_FL_MUST_CONNECT;
                break;
        case V4L2_BUF_TYPE_VIDEO_OUTPUT:
                direction = "input";
                video->pad.flags = MEDIA_PAD_FL_SOURCE
                                 | MEDIA_PAD_FL_MUST_CONNECT;
                video->video.vfl_dir = VFL_DIR_TX;
                break;

        default:
                return -EINVAL;
        }
        ret = media_entity_pads_init(&video->video.entity, 1, &video->pad);
        if (ret < 0)
                return ret;
        mutex_init(&video->mutex);
        atomic_set(&video->active, 0);

        spin_lock_init(&video->pipe.lock);
        mutex_init(&video->stream_lock);
        mutex_init(&video->queue_lock);
        spin_lock_init(&video->irqlock);

        /* Initialize the video device. */
        if (video->ops == NULL)
                video->ops = &isp_video_dummy_ops;

        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;
        video->pipe.stream_state = ISP_PIPELINE_STREAM_STOPPED;

        video_set_drvdata(&video->video, video);

        return 0;
}
```

step 12: isp_initialize_modules: omap3isp_ccdc_init

```
int omap3isp_ccdc_init(struct isp_device *isp)
{
        struct isp_ccdc_device *ccdc = &isp->isp_ccdc;
        int ret;

        spin_lock_init(&ccdc->lock);
        init_waitqueue_head(&ccdc->wait);
        mutex_init(&ccdc->ioctl_lock);

        ccdc->stopping = CCDC_STOP_NOT_REQUESTED;

        INIT_WORK(&ccdc->lsc.table_work, ccdc_lsc_free_table_work);
        ccdc->lsc.state = LSC_STATE_STOPPED;
        INIT_LIST_HEAD(&ccdc->lsc.free_queue);
```

```
	spin_lock_init(&ccdc->lsc.req_lock);

	ccdc->clamp.oblen = 0;
	ccdc->clamp.dcsubval = 0;

	ccdc->update = OMAP3ISP_CCDC_BLCLAMP;
	ccdc_apply_controls(ccdc);

	ret = ccdc_init_entities(ccdc);
	if (ret < 0) {
		mutex_destroy(&ccdc->ioctl_lock);
		return ret;
	}

	return 0;
}
```

**step 13: isp_initialize_modules: omap3isp_ccdc_init:
ccdc_init_entities**

```
static int ccdc_init_entities(struct isp_ccdc_device *ccdc)
{
	struct v4l2_subdev *sd = &ccdc->subdev;
	struct media_pad *pads = ccdc->pads;
	struct media_entity *me = &sd->entity;
	int ret;

	ccdc->input = CCDC_INPUT_NONE;

	v4l2_subdev_init(sd, &ccdc_v4l2_ops);
	sd->internal_ops = &ccdc_v4l2_internal_ops;
	strlcpy(sd->name, "OMAP3 ISP CCDC", sizeof(sd->name));
	sd->grp_id = 1 << 16;	/* group ID for isp subdevs */
	v4l2_set_subdevdata(sd, ccdc);
	sd->flags |= V4L2_SUBDEV_FL_HAS_EVENTS |
V4L2_SUBDEV_FL_HAS_DEVNODE;

	pads[CCDC_PAD_SINK].flags = MEDIA_PAD_FL_SINK
				    | MEDIA_PAD_FL_MUST_CONNECT;
	pads[CCDC_PAD_SOURCE_VP].flags = MEDIA_PAD_FL_SOURCE;
	pads[CCDC_PAD_SOURCE_OF].flags = MEDIA_PAD_FL_SOURCE;

	me->ops = &ccdc_media_ops;
	ret = media_entity_pads_init(me, CCDC_PADS_NUM, pads);
	if (ret < 0)
		return ret;

	ccdc_init_formats(sd, NULL);

	ccdc->video_out.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
	ccdc->video_out.ops = &ccdc_video_ops;
	ccdc->video_out.isp = to_isp_device(ccdc);
	ccdc->video_out.capture_mem = PAGE_ALIGN(4096 * 4096) * 3;
	ccdc->video_out.bpl_alignment = 32;

	ret = omap3isp_video_init(&ccdc->video_out, "CCDC");
	if (ret < 0)
		goto error;

	return 0;

error:
	media_entity_cleanup(me);
	return ret;
}
```

step 14: isp_register_entities:omap3isp_csi2_register_entities

```
int omap3isp_csi2_register_entities(struct isp_csi2_device *csi2,
				    struct v4l2_device *vdev)
{
	int ret;

	/* Register the subdev and video nodes. */
	ret = v4l2_device_register_subdev(vdev, &csi2->subdev);
	if (ret < 0)
		goto error;

	ret = omap3isp_video_register(&csi2->video_out, vdev);
	if (ret < 0)
		goto error;

	return 0;

error:
        omap3isp_csi2_unregister_entities(csi2);
        return ret;
}
```

step 15: isp_register_entities:omap3isp_csi2_register_entities:

omap3isp_video_register

```
int omap3isp_video_register(struct isp_video *video, struct v4l2_device
*vdev)
{
        int ret;

        video->video.v4l2_dev = vdev;

        ret = video_register_device(&video->video, VFL_TYPE_GRABBER, -1);
        if (ret < 0)
                dev_err(video->isp->dev,
                        "%s: could not register video device (%d)\n",
                        __func__, ret);

        return ret;
}
```

step 16: isp_register_entities:omap3isp_ccdc_register_entities

```
int omap3isp_ccdc_register_entities(struct isp_ccdc_device *ccdc,
        struct v4l2_device *vdev)
{
        int ret;

        /* Register the subdev and video node. */
        ret = v4l2_device_register_subdev(vdev, &ccdc->subdev);
        if (ret < 0)
                goto error;

        ret = omap3isp_video_register(&ccdc->video_out, vdev);
        if (ret < 0)
                goto error;
        return 0;

error:
        omap3isp_ccdc_unregister_entities(ccdc);
        return ret;
}
```

sub dev : Driver Skeleton

Step 1: ispcsi2

Init / start function	int omap3isp_csi2_init(struct isp_device *isp) ==>csi2_init_entities
Exit / clean function	void omap3isp_csi2_cleanup(struct isp_device *isp)

subdev operations	static const struct v4l2_subdev_ops csi2_ops = {
	        .video = &csi2_video_ops,
	        .pad = &csi2_pad_ops,
	};

subdev internal operations	static const struct v4l2_subdev_internal_ops csi2_internal_ops = {
	        .open = csi2_init_formats,
	};

isp_video_operations → ops (isp_video: custom)	static const struct isp_video_operations csi2_ispvideo_ops = {
	        .queue = csi2_queue,
	};

omap3isp_video_init	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;**

	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,
	};
	static const struct v4l2_ioctl_ops
	isp_video_ioctl_ops = {
	.vidioc_querycap = isp_video_querycap,
	.vidioc_g_fmt_vid_cap = isp_video_get_format,
	.vidioc_s_fmt_vid_cap = isp_video_set_format,
	.vidioc_try_fmt_vid_cap = isp_video_try_format,
	.vidioc_g_fmt_vid_out = isp_video_get_format,
	.vidioc_s_fmt_vid_out = isp_video_set_format,
	.vidioc_try_fmt_vid_out = isp_video_try_format,
	.vidioc_g_selection = isp_video_get_selection,
	.vidioc_s_selection = isp_video_set_selection,
	.vidioc_g_parm = isp_video_get_param,
	.vidioc_s_parm = isp_video_set_param,
	.vidioc_reqbufs = isp_video_reqbufs,
	.vidioc_querybuf = isp_video_querybuf,
	.vidioc_qbuf = isp_video_qbuf,
	.vidioc_dqbuf = isp_video_dqbuf,
	.vidioc_streamon = isp_video_streamon,
	.vidioc_streamoff = isp_video_streamoff,
	.vidioc_enum_input = isp_video_enum_input,
	.vidioc_g_input = isp_video_g_input,
	.vidioc_s_input = isp_video_s_input,
	};
omap3isp_csi2_isr - CSI2 interrupt handling.	omap3isp_csi2_isr

static const unsigned int csi2_input_fmts[] = {
MEDIA_BUS_FMT_SGRBG10_1X10,
MEDIA_BUS_FMT_SGRBG10_DPCM8_1X8,
MEDIA_BUS_FMT_SRGGB10_1X10,
MEDIA_BUS_FMT_SRGGB10_DPCM8_1X8,
MEDIA_BUS_FMT_SBGGR10_1X10,
MEDIA_BUS_FMT_SBGGR10_DPCM8_1X8,

MEDIA_BUS_FMT_SGBRG10_1X10,
MEDIA_BUS_FMT_SGBRG10_DPCM8_1X8,
MEDIA_BUS_FMT_YUYV8_2X8,
};

Step 2: ispccdc

Init / start function	omap3isp_ccdc_init ==>ccdc_init_entities
Exit / clean function	void omap3isp_ccdc_cleanup(struct isp_device *isp)

subdev operations	**static const struct v4l2_subdev_ops ccdc_v4l2_ops** = {
	        .core = &ccdc_v4l2_core_ops,
	        .video = &ccdc_v4l2_video_ops,
	        .pad = &ccdc_v4l2_pad_ops,
	};

subdev internal operations	**static const struct v4l2_subdev_internal_ops ccdc_v4l2_internal_ops** = {
	        .open = ccdc_init_formats,
	};

isp_video_operations → ops (isp_video: custom)	static const struct isp_video_operations ccdc_video_ops = {
	        .queue = ccdc_video_queue,
	};

omap3isp_video_init	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;**

	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,
	};

	static const struct v4l2_ioctl_ops
	isp_video_ioctl_ops = {
	  .vidioc_querycap = isp_video_querycap,
	  .vidioc_g_fmt_vid_cap = isp_video_get_format,
	  .vidioc_s_fmt_vid_cap = isp_video_set_format,
	   .vidioc_try_fmt_vid_cap = isp_video_try_format,
	   .vidioc_g_fmt_vid_out = isp_video_get_format,
	   .vidioc_s_fmt_vid_out = isp_video_set_format,
	   .vidioc_try_fmt_vid_out = isp_video_try_format,
	    .vidioc_g_selection = isp_video_get_selection,
	     .vidioc_s_selection = isp_video_set_selection,
	        .vidioc_g_parm = isp_video_get_param,
	        .vidioc_s_parm = isp_video_set_param,
	        .vidioc_reqbufs = isp_video_reqbufs,
	        .vidioc_querybuf = isp_video_querybuf,
	        .vidioc_qbuf = isp_video_qbuf,
	        .vidioc_dqbuf = isp_video_dqbuf,
	        .vidioc_streamon = isp_video_streamon,
	        .vidioc_streamoff = isp_video_streamoff,
	       .vidioc_enum_input = isp_video_enum_input,
	        .vidioc_g_input = isp_video_g_input,
	        .vidioc_s_input = isp_video_s_input,
	};

omap3isp_ccdc_isr	omap3isp_ccdc_isr

static const unsigned int ccdc_fmts[] = {
MEDIA_BUS_FMT_Y8_1X8,
MEDIA_BUS_FMT_Y10_1X10,
MEDIA_BUS_FMT_Y12_1X12,

MEDIA_BUS_FMT_SGRBG8_1X8,
MEDIA_BUS_FMT_SRGGB8_1X8,
MEDIA_BUS_FMT_SBGGR8_1X8,
MEDIA_BUS_FMT_SGBRG8_1X8,
MEDIA_BUS_FMT_SGRBG10_1X10,
MEDIA_BUS_FMT_SRGGB10_1X10,
MEDIA_BUS_FMT_SBGGR10_1X10,
MEDIA_BUS_FMT_SGBRG10_1X10,
MEDIA_BUS_FMT_SGRBG12_1X12,
MEDIA_BUS_FMT_SRGGB12_1X12,
MEDIA_BUS_FMT_SBGGR12_1X12,
MEDIA_BUS_FMT_SGBRG12_1X12,
MEDIA_BUS_FMT_YUYV8_2X8,
MEDIA_BUS_FMT_UYVY8_2X8,
};

Step 3: isppreview

Init / start function	omap3isp_preview_init==>preview_init_entities
Exit / clean function	omap3isp_preview_cleanup

subdev operations	**static const struct v4l2_subdev_ops preview_v4l2_ops** = {
	.core = &preview_v4l2_core_ops,
	.video = &preview_v4l2_video_ops,
	.pad = &preview_v4l2_pad_ops,
	};

subdev internal operations	**static const struct v4l2_subdev_internal_ops preview_v4l2_internal_ops** = {
	.open = preview_init_formats,
	};

isp_video_operations → ops (isp_video: custom)	**static const struct isp_video_operations preview_video_ops** = {
	.queue = preview_video_queue,
	};

omap3isp_video_init	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;**

	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,
	};

	static const struct v4l2_ioctl_ops
	isp_video_ioctl_ops = {
	.vidioc_querycap = isp_video_querycap,
	.vidioc_g_fmt_vid_cap = isp_video_get_format,
	.vidioc_s_fmt_vid_cap = isp_video_set_format,
	.vidioc_try_fmt_vid_cap = isp_video_try_format,
	.vidioc_g_fmt_vid_out = isp_video_get_format,
	.vidioc_s_fmt_vid_out = isp_video_set_format,
	.vidioc_try_fmt_vid_out = isp_video_try_format,
	.vidioc_g_selection = isp_video_get_selection,
	.vidioc_s_selection = isp_video_set_selection,
	.vidioc_g_parm = isp_video_get_param,
	.vidioc_s_parm = isp_video_set_param,
	.vidioc_reqbufs = isp_video_reqbufs,
	.vidioc_querybuf = isp_video_querybuf,
	.vidioc_qbuf = isp_video_qbuf,
	.vidioc_dqbuf = isp_video_dqbuf,
	.vidioc_streamon = isp_video_streamon,
	.vidioc_streamoff = isp_video_streamoff,
	.vidioc_enum_input = isp_video_enum_input,
	.vidioc_g_input = isp_video_g_input,
	.vidioc_s_input = isp_video_s_input,
	};

omap3isp_preview_isr	omap3isp_preview_isr

/* previewer format descriptions */
static const unsigned int preview_input_fmts[] = {
MEDIA_BUS_FMT_Y8_1X8,
MEDIA_BUS_FMT_SGRBG8_1X8,
MEDIA_BUS_FMT_SRGGB8_1X8,
MEDIA_BUS_FMT_SBGGR8_1X8,
MEDIA_BUS_FMT_SGBRG8_1X8,
MEDIA_BUS_FMT_Y10_1X10,
MEDIA_BUS_FMT_SGRBG10_1X10,
MEDIA_BUS_FMT_SRGGB10_1X10,
MEDIA_BUS_FMT_SBGGR10_1X10,
MEDIA_BUS_FMT_SGBRG10_1X10,
};

static const unsigned int preview_output_fmts[] = {
MEDIA_BUS_FMT_UYVY8_1X16,
MEDIA_BUS_FMT_YUYV8_1X16,
};

Step 4: ispresizer

Init / start function	omap3isp_resizer_init==>resizer_init_entities
Exit / clean function	omap3isp_preview_cleanup
subdev operations	**static const struct v4l2_subdev_ops resizer_v4l2_ops** = { .video = &resizer_v4l2_video_ops, .pad = &resizer_v4l2_pad_ops,

	};

subdev internal operations	**static const struct v4l2_subdev_internal_ops resizer_v4l2_internal_ops** = {
	        .open = resizer_init_formats,
	};

isp_video_operations → ops (isp_video: custom)	**static const struct isp_video_operations resizer_video_ops** = {
	        .queue = resizer_video_queue,
	}

omap3isp_video_init	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;**

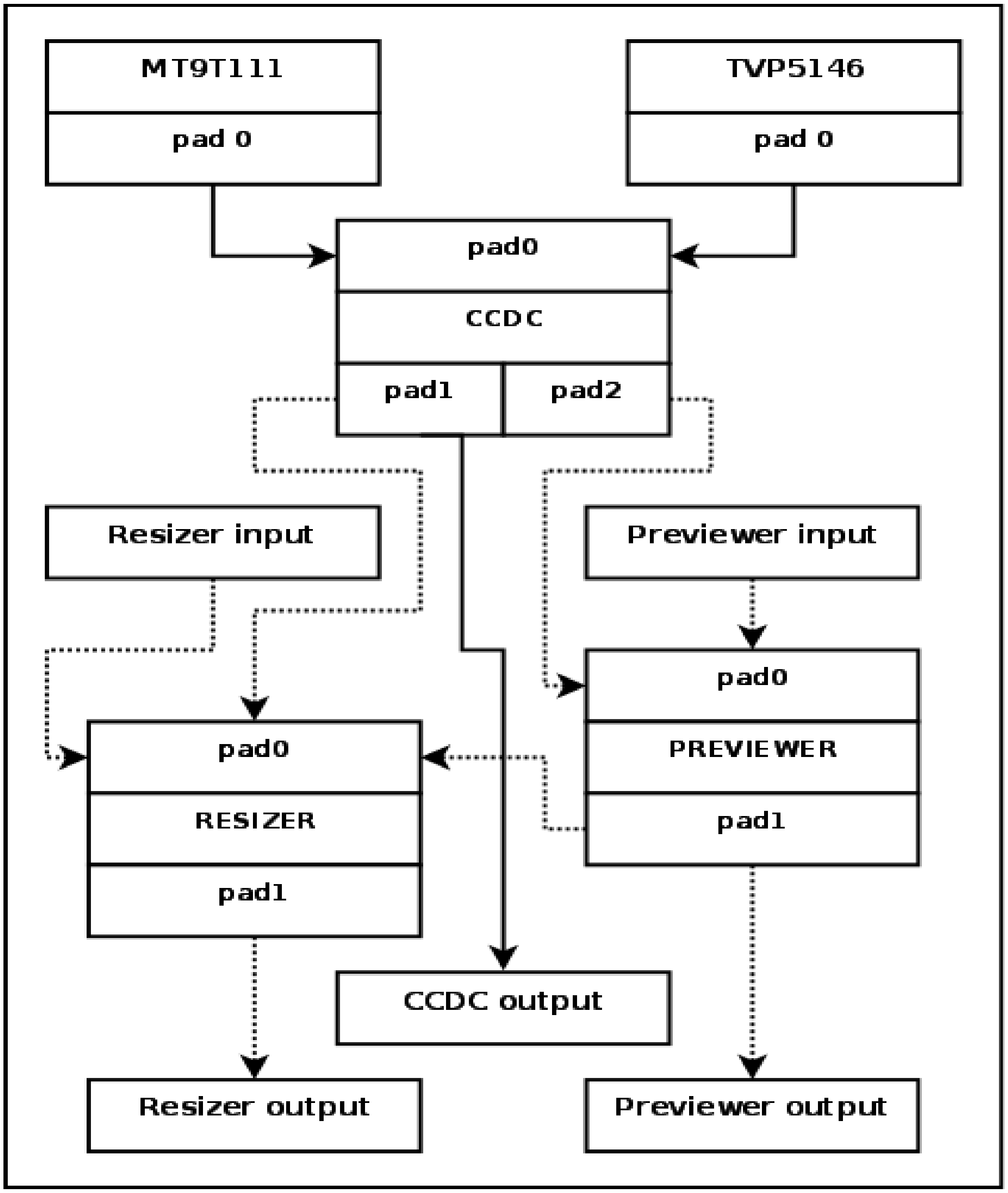
	static const struct v4l2_file_operations isp_video_fops = {
	        .owner = THIS_MODULE,
	        .unlocked_ioctl = video_ioctl2,
	        .open = isp_video_open,
	        .release = isp_video_release,
	        .poll = isp_video_poll,
	        .mmap = isp_video_mmap,
	};

	static const struct v4l2_ioctl_ops isp_video_ioctl_ops = {
	  .vidioc_querycap = isp_video_querycap,
	  .vidioc_g_fmt_vid_cap = isp_video_get_format,
	  .vidioc_s_fmt_vid_cap = isp_video_set_format,
	   .vidioc_try_fmt_vid_cap = isp_video_try_format,
	   .vidioc_g_fmt_vid_out = isp_video_get_format,
	   .vidioc_s_fmt_vid_out = isp_video_set_format,
	   .vidioc_try_fmt_vid_out = isp_video_try_format,
	    .vidioc_g_selection = isp_video_get_selection,
	     .vidioc_s_selection = isp_video_set_selection,
	        .vidioc_g_parm = isp_video_get_param,
	        .vidioc_s_parm = isp_video_set_param,

	.vidioc_reqbufs = isp_video_reqbufs,
	.vidioc_querybuf = isp_video_querybuf,
	.vidioc_qbuf = isp_video_qbuf,
	.vidioc_dqbuf = isp_video_dqbuf,
	.vidioc_streamon = isp_video_streamon,
	.vidioc_streamoff = isp_video_streamoff,
	.vidioc_enum_input = isp_video_enum_input,
	.vidioc_g_input = isp_video_g_input,
	.vidioc_s_input = isp_video_s_input,
	};
omap3isp_resizer_isr	omap3isp_resizer_isr

static const unsigned int resizer_formats[] = {
MEDIA_BUS_FMT_UYVY8_1X16,
MEDIA_BUS_FMT_YUYV8_1X16,
};

Media pads : isp_create_links



Step 1: csi2

```
#define CSI2_PAD_SINK            0
#define CSI2_PAD_SOURCE          1
#define CSI2_PADS_NUM            2
```

```
        pads[CSI2_PAD_SOURCE].flags = MEDIA_PAD_FL_SOURCE;
        pads[CSI2_PAD_SINK].flags = MEDIA_PAD_FL_SINK
                                    | MEDIA_PAD_FL_MUST_CONNECT;

        me->ops = &csi2_media_ops;
        ret = media_entity_pads_init(me, CSI2_PADS_NUM, pads);
        if (ret < 0)
                return ret;
```

Step 2: ccp2

```
#define CCP2_PAD_SINK                   0
#define CCP2_PAD_SOURCE                 1
#define CCP2_PADS_NUM                   2

        pads[CCP2_PAD_SINK].flags = MEDIA_PAD_FL_SINK
                                    | MEDIA_PAD_FL_MUST_CONNECT;
        pads[CCP2_PAD_SOURCE].flags = MEDIA_PAD_FL_SOURCE;

        me->ops = &ccp2_media_ops;
        ret = media_entity_pads_init(me, CCP2_PADS_NUM, pads);
        if (ret < 0)
                return ret;
```

step 3: ccdc

```
#define CCDC_PAD_SINK                   0
#define CCDC_PAD_SOURCE_OF              1
#define CCDC_PAD_SOURCE_VP              2
#define CCDC_PADS_NUM                   3

       pads[CCDC_PAD_SINK].flags = MEDIA_PAD_FL_SINK
                                    | MEDIA_PAD_FL_MUST_CONNECT;
        pads[CCDC_PAD_SOURCE_VP].flags = MEDIA_PAD_FL_SOURCE;
        pads[CCDC_PAD_SOURCE_OF].flags = MEDIA_PAD_FL_SOURCE;
        me->ops = &ccdc_media_ops;
        ret = media_entity_pads_init(me, CCDC_PADS_NUM, pads);
        if (ret < 0)
                return ret;
```

step 4: resizer

```
#define RESZ_PAD_SINK                   0
#define RESZ_PAD_SOURCE                 1
#define RESZ_PADS_NUM                   2

        pads[RESZ_PAD_SINK].flags = MEDIA_PAD_FL_SINK
                                    | MEDIA_PAD_FL_MUST_CONNECT;
        pads[RESZ_PAD_SOURCE].flags = MEDIA_PAD_FL_SOURCE;

        me->ops = &resizer_media_ops;
        ret = media_entity_pads_init(me, RESZ_PADS_NUM, pads);
        if (ret < 0)
                return ret;
```

step 4: Create links between entities.

```
     /* Create links between entities. */
        ret = media_create_pad_link(
                        &isp->isp_csi2a.subdev.entity, CSI2_PAD_SOURCE,
                        &isp->isp_ccdc.subdev.entity, CCDC_PAD_SINK, 0);
        if (ret < 0)
                return ret;

        ret = media_create_pad_link(
                        &isp->isp_ccp2.subdev.entity, CCP2_PAD_SOURCE,
                        &isp->isp_ccdc.subdev.entity, CCDC_PAD_SINK, 0);
        if (ret < 0)
                return ret;

         ret = media_create_pad_link(
                        &isp->isp_ccdc.subdev.entity, CCDC_PAD_SOURCE_VP,
                        &isp->isp_prev.subdev.entity, PREV_PAD_SINK, 0);
        if (ret < 0)
                return ret;

        ret = media_create_pad_link(
                        &isp->isp_ccdc.subdev.entity, CCDC_PAD_SOURCE_OF,
                        &isp->isp_res.subdev.entity, RESZ_PAD_SINK, 0);
        if (ret < 0)
                return ret;


         ret = media_create_pad_link(
                        &isp->isp_prev.subdev.entity, PREV_PAD_SOURCE,
                        &isp->isp_res.subdev.entity, RESZ_PAD_SINK, 0);

         ret = media_create_pad_link(
                        &isp->isp_ccdc.subdev.entity, CCDC_PAD_SOURCE_VP,
                        &isp->isp_aewb.subdev.entity, 0,
                        MEDIA_LNK_FL_ENABLED | MEDIA_LNK_FL_IMMUTABLE);
        if (ret < 0)
                return ret;

        ret = media_create_pad_link(
                        &isp->isp_ccdc.subdev.entity, CCDC_PAD_SOURCE_VP,
                        &isp->isp_af.subdev.entity, 0,
                        MEDIA_LNK_FL_ENABLED | MEDIA_LNK_FL_IMMUTABLE);
        if (ret < 0)
                return ret;

        ret = media_create_pad_link(
                        &isp->isp_ccdc.subdev.entity, CCDC_PAD_SOURCE_VP,
                        &isp->isp_hist.subdev.entity, 0,
                        MEDIA_LNK_FL_ENABLED | MEDIA_LNK_FL_IMMUTABLE);
        if (ret < 0)
                return ret;
```

OMAP3 ISP: ISR

Step 1: ISP IRQ: isp.c (probe)

drivers/media/platform/omap3isp/isp.c

```
	/* Interrupt */
	ret = platform_get_irq(pdev, 0);
	if (ret <= 0) {
		dev_err(isp->dev, "No IRQ resource\n");
		ret = -ENODEV;
		goto error_iommu;
	}
	isp->irq_num = ret;

	if (devm_request_irq(isp->dev, isp->irq_num, isp_isr,
IRQF_SHARED,    "OMAP3 ISP", isp)) {
		dev_err(isp->dev, "Unable to request IRQ\n");
		ret = -EINVAL;
		goto error_iommu;
	}
```

Step 2: ISP ISR HANDLER : isp.c (probe)

```
static irqreturn_t isp_isr(int irq, void *_isp)
{
	static const u32 ccdc_events = IRQ0STATUS_CCDC_LSC_PREF_ERR_IRQ
|
					IRQ0STATUS_CCDC_LSC_DONE_IRQ |
					IRQ0STATUS_CCDC_VD0_IRQ |
					IRQ0STATUS_CCDC_VD1_IRQ |
					IRQ0STATUS_HS_VS_IRQ;
	struct isp_device *isp = _isp;
	u32 irqstatus;

	irqstatus = isp_reg_readl(isp, OMAP3_ISP_IOMEM_MAIN,
ISP_IRQ0STATUS);
	isp_reg_writel(isp, irqstatus, OMAP3_ISP_IOMEM_MAIN,
ISP_IRQ0STATUS);

	isp_isr_sbl(isp);

	if (irqstatus & IRQ0STATUS_CSIA_IRQ)
		omap3isp_csi2_isr(&isp->isp_csi2a);
```

```
        if (irqstatus & IRQ0STATUS_CSIB_IRQ)
                omap3isp_ccp2_isr(&isp->isp_ccp2);

        if (irqstatus & IRQ0STATUS_CCDC_VD0_IRQ) {
                if (isp->isp_ccdc.output & CCDC_OUTPUT_PREVIEW)
                        omap3isp_preview_isr_frame_sync(&isp->isp_prev);
                if (isp->isp_ccdc.output & CCDC_OUTPUT_RESIZER)
                        omap3isp_resizer_isr_frame_sync(&isp->isp_res);
                omap3isp_stat_isr_frame_sync(&isp->isp_aewb);
                omap3isp_stat_isr_frame_sync(&isp->isp_af);
                omap3isp_stat_isr_frame_sync(&isp->isp_hist);
        }

        if (irqstatus & ccdc_events)
                omap3isp_ccdc_isr(&isp->isp_ccdc, irqstatus & ccdc_events);

                if (irqstatus & IRQ0STATUS_PRV_DONE_IRQ) {
                if (isp->isp_prev.output & PREVIEW_OUTPUT_RESIZER)
                        omap3isp_resizer_isr_frame_sync(&isp->isp_res);
                omap3isp_preview_isr(&isp->isp_prev);
        }

        if (irqstatus & IRQ0STATUS_RSZ_DONE_IRQ)
                omap3isp_resizer_isr(&isp->isp_res);

        if (irqstatus & IRQ0STATUS_H3A_AWB_DONE_IRQ)
                omap3isp_stat_isr(&isp->isp_aewb);

        if (irqstatus & IRQ0STATUS_H3A_AF_DONE_IRQ)
                omap3isp_stat_isr(&isp->isp_af);

        if (irqstatus & IRQ0STATUS_HIST_DONE_IRQ)
                omap3isp_stat_isr(&isp->isp_hist);

        omap3isp_flush(isp);

#if defined(DEBUG) && defined(ISP_ISR_DEBUG)
        isp_isr_dbg(isp, irqstatus);
#endif

        return IRQ_HANDLED;
}
```

OMAP3 ISP: DEVICE TREE

Step 1: OMAP3 ISP : OMAP34XX : omap3-n900.dts

```
  isp: isp@480bc000 {
                compatible = "ti,omap3-isp";
                reg = <0x480bc000 0x12fc
                      0x480bd800 0x017c>;
                interrupts = <24>;
                iommus = <&mmu_isp>;
                syscon = <&scm_conf 0x6c>;
                ti,phy-type = <OMAP3ISP_PHY_TYPE_COMPLEX_IO>;
                #clock-cells = <1>;
                ports {
                        #address-cells = <1>;
                        #size-cells = <0>;
                };
 };

&isp {
        vdds_csib-supply = <&vaux2>;

        pinctrl-names = "default";
        pinctrl-0 = <&camera_pins>;

        ports {
                port@1 {
                        reg = <1>;

                        csi_isp: endpoint {
                                remote-endpoint = <&csi_cam1>;
                                bus-type = <3>; /* CCP2 */
                                clock-lanes = <1>;
                                data-lanes = <0>;
                                lane-polarity = <0 0>;
                                /* Select strobe = <1> for back camera, <0> for front
                                     camera */
                                strobe = <1>;
                        };
                };
        };
};


&i2c3 {
        pinctrl-names = "default";
        pinctrl-0 = <&i2c3_pins>;

        clock-frequency = <400000>;

    cam1: camera@3e {
                compatible = "toshiba,et8ek8";
                reg = <0x3e>;
                vana-supply = <&vaux4>;
                clocks = <&isp 0>;
                clock-names = "extclk";
                clock-frequency = <9600000>;
                reset-gpio = <&gpio4 6 GPIO_ACTIVE_HIGH>; /* 102 */
                lens-focus = <&ad5820>;

                port {
                    csi_cam1: endpoint {
                        bus-type = <3>; /* CCP2 */
                        strobe = <1>;
                        clock-inv = <0>;
                        crc = <1>;

                        remote-endpoint = <&csi_isp>;
                    };
                };
        };
};
```

Step 2: OMAP3 ISP : OMAP36XX : omap3-n950.dts

```
  isp: isp@480bc000 {
                compatible = "ti,omap3-isp";
                reg = <0x480bc000 0x12fc
                       0x480bd800 0x0600>;
                interrupts = <24>;
                iommus = <&mmu_isp>;
                syscon = <&scm_conf 0x2f0>;
                ti,phy-type = <OMAP3ISP_PHY_TYPE_CSIPHY>;
                #clock-cells = <1>;
                ports {
                        #address-cells = <1>;
                        #size-cells = <0>;
                };
```

```
 };

&isp {
        vdd-csiphy1-supply = <&vaux2>;
        vdd-csiphy2-supply = <&vaux2>;
        ports {
                port@2 {
                        reg = <2>;
                        csi2a_ep: endpoint {
                                remote-endpoint = <&smia_1_1>;
                                clock-lanes = <2>;
                                data-lanes = <3 1>;
                                crc = <1>;
                                lane-polarities = <1 1 1>;
                        };
                };
        };
};

&i2c2 {
        smia_1: camera@10 {
                compatible = "nokia,smia";
                reg = <0x10>;
                /* No reset gpio */
                vana-supply = <&vaux3>;
                clocks = <&isp 0>;
                clock-frequency = <9600000>;
                nokia,nvm-size = <(16 * 64)>;
                flash-leds = <&as3645a_flash &as3645a_indicator>;
                port {
                        smia_1_1: endpoint {
                                link-frequencies = /bits/ 64 <210000000 333600000
398400000>;
                                clock-lanes = <0>;
                                data-lanes = <1 2>;
                                remote-endpoint = <&csi2a_ep>;
                        };
                };
        };
};
```

Step 3: isp.c (probe)

drivers/media/platform/omap3isp/isp.c

```
	ret = v4l2_async_notifier_parse_fwnode_endpoints(
		&pdev->dev, &isp->notifier, sizeof(struct isp_async_subdev),
		isp_fwnode_parse);
	if (ret < 0)
		goto error;


	isp->notifier.ops = &isp_subdev_notifier_ops;

	ret = v4l2_async_notifier_register(&isp->v4l2_dev, &isp->notifier);
	if (ret)
		goto error_register_entities;
```

Step 4: isp_fwnode_parse

```
enum isp_of_phy {
	ISP_OF_PHY_PARALLEL = 0,
	ISP_OF_PHY_CSIPHY1,
	ISP_OF_PHY_CSIPHY2,
};

static int isp_fwnode_parse(struct device *dev,
			    struct v4l2_fwnode_endpoint *vep,
			    struct v4l2_async_subdev *asd)
{
	struct isp_async_subdev *isd =
		container_of(asd, struct isp_async_subdev, asd);
	struct isp_bus_cfg *buscfg = &isd->bus;
	bool csi1 = false;
	unsigned int i;

	dev_dbg(dev, "parsing endpoint %pOF, interface %u\n",
		to_of_node(vep->base.local_fwnode), vep->base.port);

	switch (vep->base.port) {
	case ISP_OF_PHY_PARALLEL:
		buscfg->interface = ISP_INTERFACE_PARALLEL;
		buscfg->bus.parallel.data_lane_shift =
			vep->bus.parallel.data_shift;
		buscfg->bus.parallel.clk_pol =
			!!(vep->bus.parallel.flags
				& V4L2_MBUS_PCLK_SAMPLE_FALLING);
		buscfg->bus.parallel.hs_pol =
			!!(vep->bus.parallel.flags &
V4L2_MBUS_VSYNC_ACTIVE_LOW);
		buscfg->bus.parallel.vs_pol =
			!!(vep->bus.parallel.flags &
V4L2_MBUS_HSYNC_ACTIVE_LOW);
		buscfg->bus.parallel.fld_pol =
			!!(vep->bus.parallel.flags & V4L2_MBUS_FIELD_EVEN_LOW);
		buscfg->bus.parallel.data_pol =
			!!(vep->bus.parallel.flags &
V4L2_MBUS_DATA_ACTIVE_LOW);
		buscfg->bus.parallel.bt656 = vep->bus_type ==
V4L2_MBUS_BT656;
		break;

	case ISP_OF_PHY_CSIPHY1:
	case ISP_OF_PHY_CSIPHY2:
		switch (vep->bus_type) {
		case V4L2_MBUS_CCP2:
		case V4L2_MBUS_CSI1:
			dev_dbg(dev, "CSI-1/CCP-2 configuration\n");
			csi1 = true;
			break;
		case V4L2_MBUS_CSI2:
			dev_dbg(dev, "CSI-2 configuration\n");
			csi1 = false;
			break;
		default:
			dev_err(dev, "unsupported bus type %u\n",
				vep->bus_type);
			return -EINVAL;
		}

		switch (vep->base.port) {
		case ISP_OF_PHY_CSIPHY1:
			if (csi1)
				buscfg->interface = ISP_INTERFACE_CCP2B_PHY1;
			else
				buscfg->interface = ISP_INTERFACE_CSI2C_PHY1;
			break;
		case ISP_OF_PHY_CSIPHY2:
			if (csi1)
				buscfg->interface = ISP_INTERFACE_CCP2B_PHY2;
		else
			buscfg->interface = ISP_INTERFACE_CSI2A_PHY2;
		break;
	}
	if (csi1) {
		buscfg->bus.ccp2.lanecfg.clk.pos =
			vep->bus.mipi_csi1.clock_lane;
		buscfg->bus.ccp2.lanecfg.clk.pol =
			vep->bus.mipi_csi1.lane_polarity[0];
		dev_dbg(dev, "clock lane polarity %u, pos %u\n",
			buscfg->bus.ccp2.lanecfg.clk.pol,
			buscfg->bus.ccp2.lanecfg.clk.pos);

		buscfg->bus.ccp2.lanecfg.data[0].pos =
			vep->bus.mipi_csi1.data_lane;
		buscfg->bus.ccp2.lanecfg.data[0].pol =
			vep->bus.mipi_csi1.lane_polarity[1];

		dev_dbg(dev, "data lane polarity %u, pos %u\n",
			buscfg->bus.ccp2.lanecfg.data[0].pol,
			buscfg->bus.ccp2.lanecfg.data[0].pos);

		buscfg->bus.ccp2.strobe_clk_pol =
			vep->bus.mipi_csi1.clock_inv;
		buscfg->bus.ccp2.phy_layer = vep->bus.mipi_csi1.strobe;
		buscfg->bus.ccp2.ccp2_mode =
			vep->bus_type == V4L2_MBUS_CCP2;
		buscfg->bus.ccp2.vp_clk_pol = 1;

		buscfg->bus.ccp2.crc = 1;
	} else {
		buscfg->bus.csi2.lanecfg.clk.pos =
			vep->bus.mipi_csi2.clock_lane;
		buscfg->bus.csi2.lanecfg.clk.pol =
			vep->bus.mipi_csi2.lane_polarities[0];
		dev_dbg(dev, "clock lane polarity %u, pos %u\n",
			buscfg->bus.csi2.lanecfg.clk.pol,
			buscfg->bus.csi2.lanecfg.clk.pos);

		buscfg->bus.csi2.num_data_lanes =
			vep->bus.mipi_csi2.num_data_lanes;

		for (i = 0; i < buscfg->bus.csi2.num_data_lanes; i++) {
			buscfg->bus.csi2.lanecfg.data[i].pos =
                                vep->bus.mipi_csi2.data_lanes[i];
                        buscfg->bus.csi2.lanecfg.data[i].pol =
                                vep->bus.mipi_csi2.lane_polarities[i + 1];
                        dev_dbg(dev,
                                "data lane %u polarity %u, pos %u\n", i,
                                buscfg->bus.csi2.lanecfg.data[i].pol,
                                buscfg->bus.csi2.lanecfg.data[i].pos);
                }
                /*
                 * FIXME: now we assume the CRC is always there.
                 * Implement a way to obtain this information from the
                 * sensor. Frame descriptors, perhaps?
                 */
                buscfg->bus.csi2.crc = 1;
        }
        break;

default:
        dev_warn(dev, "%pOF: invalid interface %u\n",
                 to_of_node(vep->base.local_fwnode), vep->base.port);
        return -EINVAL;
}

return 0;
}
```

Step 5: isp_subdev_notifier_ops

static int isp_subdev_notifier_complete(struct v4l2_async_notifier *async)

```
{
        struct isp_device *isp = container_of(async, struct isp_device,
                                              notifier);
        struct v4l2_device *v4l2_dev = &isp->v4l2_dev;
        struct v4l2_subdev *sd;
        int ret;

        ret = media_entity_enum_init(&isp->crashed, &isp->media_dev);
        if (ret)
                return ret;

        list_for_each_entry(sd, &v4l2_dev->subdevs, list) {
                if (sd->notifier != &isp->notifier)
```

```
                        continue;

                ret = isp_link_entity(isp, &sd->entity,
                                           v4l2_subdev_to_bus_cfg(sd)->interface);
                if (ret < 0)
                        return ret;
        }

        ret = v4l2_device_register_subdev_nodes(&isp->v4l2_dev);
        if (ret < 0)
                return ret;

        return media_device_register(&isp->media_dev);
}

static const struct v4l2_async_notifier_operations
isp_subdev_notifier_ops = {
        .complete = isp_subdev_notifier_complete,
};
```