Soc-Camera Subsystem

Terminology

The following terms are used in this document:
- camera / camera device / camera sensor - a video-camera sensor chip, capable of connecting to a variety of systems and interfaces, typically uses i2c for control and configuration, and a parallel or a serial bus for data.
- camera host - an interface, to which a camera is connected. Typically a specialised interface, present on many SoCs, e.g., PXA27x and PXA3xx, SuperH, AVR32, i.MX27, i.MX31.
- camera host bus - a connection between a camera host and a camera. Can be parallel or serial, consists of data and control lines, e.g., clock, vertical and horizontal synchronization signals.

Purpose of the soc-camera subsystem

The soc-camera subsystem provides a unified API between camera host drivers and camera sensor drivers. It implements a V4L2 interface to the user, currently only the mmap method is supported.

This subsystem has been written to connect drivers for System-on-Chip (SoC) video capture interfaces with drivers for CMOS camera sensor chips to enable the reuse of sensor drivers with various hosts. The subsystem has been designed to support multiple camera host interfaces and multiple cameras per interface, although most applications have only one camera sensor.

Existing drivers

As of 2.6.27-rc4 there are two host drivers in the mainline: pxa_camera.c for PXA27x SoCs and sh_mobile_ceu_camera.c for SuperH SoCs, and four sensor drivers: mt9m001.c, mt9m111.c, mt9v022.c and a generic soc_camera_platform.c driver. This list is not supposed to be updated, look for more examples in your tree.

Camera host API

A host camera driver is registered using the

```
soc_camera_host_register(struct soc_camera_host *);
```

function. The host object can be initialized as follows:

```
static struct soc_camera_host pxa_soc_camera_host = {
        .drv_name       = PXA_CAM_DRV_NAME,
        .ops            = &pxa_soc_camera_host_ops,
};
```

All camera host methods are passed in a struct soc_camera_host_ops:

```
static struct soc_camera_host_ops pxa_soc_camera_host_ops = {
        .owner          = THIS_MODULE,
```

```
        .add            = pxa_camera_add_device,
        .remove         = pxa_camera_remove_device,
        .suspend        = pxa_camera_suspend,
        .resume         = pxa_camera_resume,
        .set_fmt_cap    = pxa_camera_set_fmt_cap,
        .try_fmt_cap    = pxa_camera_try_fmt_cap,
        .init_videobuf  = pxa_camera_init_videobuf,
        .reqbufs        = pxa_camera_reqbufs,
        .poll           = pxa_camera_poll,
        .querycap       = pxa_camera_querycap,
        .try_bus_param  = pxa_camera_try_bus_param,
        .set_bus_param  = pxa_camera_set_bus_param,
};
```

.add and .remove methods are called when a sensor is attached to or detached from the host, apart from performing host-internal tasks they shall also call sensor driver's .init and .release methods respectively. .suspend and .resume methods implement host's power-management functionality and its their responsibility to call respective sensor's methods. .try_bus_param and .set_bus_param are used to negotiate physical connection parameters between the host and the sensor. .init_videobuf is called by soc-camera core when a video-device is opened, further video-buffer management is implemented completely
by the specific camera host driver. The rest of the methods are called from respective V4L2 operations.

Camera API

Sensor drivers can use struct soc_camera_link, typically provided by the platform, and used to specify to which camera host bus the sensor is connected, and arbitrarily provide platform .power and .reset methods for the camera. soc_camera_device_register() and soc_camera_device_unregister() functions are used to add a sensor driver to or remove one from the system. The registration function takes a pointer to struct soc_camera_device as the only parameter. This struct can be initialized as follows:

```
        /* link to driver operations */
        icd->ops        = &mt9m001_ops;
        /* link to the underlying physical (e.g., i2c) device */
        icd->control    = &client->dev;
        /* window geometry */
        icd->x_min      = 20;
        icd->y_min      = 12;
        icd->x_current  = 20;
        icd->y_current  = 12;
        icd->width_min  = 48;
        icd->width_max  = 1280;
        icd->height_min = 32;
        icd->height_max = 1024;
        icd->y_skip_top = 1;
        /* camera bus ID, typically obtained from platform data */
        icd->iface      = icl->bus_id;
```

struct soc_camera_ops provides .probe and .remove methods, which are called by the soc-camera core, when a camera is matched against or removed from a camera

host bus, .init, .release, .suspend, and .resume are called from the camera host driver as discussed above. Other members of this struct provide respective V4L2 functionality.

struct soc_camera_device also links to an array of struct
soc_camera_data_format,
listing pixel formats, supported by the camera.

VIDIOC_S_CROP and VIDIOC_S_FMT behaviour

Above user ioctls modify image geometry as follows:

VIDIOC_S_CROP: sets location and sizes of the sensor window. Unit is one sensor pixel. Changing sensor window sizes preserves any scaling factors, therefore user window sizes change as well.

VIDIOC_S_FMT: sets user window. Should preserve previously set sensor window as much as possible by modifying scaling factors. If the sensor window cannot be preserved precisely, it may be changed too.

In soc-camera there are two locations, where scaling and cropping can taks place: in the camera driver and in the host driver. User ioctls are first passed to the host driver, which then generally passes them down to the camera driver. It is more efficient to perform scaling and cropping in the camera driver to save camera bus bandwidth and maximise the framerate. However, if the camera driver failed to set the required parameters with sufficient precision, the host driver may decide to also use its own scaling and cropping to fulfill the user's request.

Camera drivers are interfaced to the soc-camera core and to host drivers over the v4l2-subdev API, which is completely functional, it doesn't pass any data. Therefore all camera drivers shall reply to .g_fmt() requests with their current output geometry. This is necessary to correctly configure the camera bus. .s_fmt() and .try_fmt() have to be implemented too. Sensor window and scaling factors have to be maintained by camera drivers internally. According to the V4L2 API all capture drivers must support the VIDIOC_CROPCAP ioctl, hence we rely on camera drivers implementing .cropcap(). If the camera driver does not support cropping, it may choose to not implement .s_crop(), but to enable cropping support by the camera host driver at least the .g_crop method must be implemented.

User window geometry is kept in .user_width and .user_height fields in struct soc_camera_device and used by the soc-camera core and host drivers. The core updates these fields upon successful completion of a .s_fmt() call, but if these fields change elsewhere, e.g., during .s_crop() processing, the host driver is responsible for updating them.

--
Author: Guennadi Liakhovetski <g.liakhovetski@gmx.de>