Cameras, Devicetree and ACPI: A Device Driver Perspective

Sakari Ailus
2022-09-15
Cameras

- **USB webcams**
  - Found in many laptops as well as pluggable USB devices

- **Complex cameras**
  - Typically consisting of
    - Camera module
    - CSI-2 (or parallel) bus receiver
    - Image signal processor (ISP)
  - Virtually all mobile phones
  - Increasingly found in laptops, too
USB cameras

• Uvcvideo driver implements V4L2 interface applications can use directly

• That’s all you need!
Complex cameras
Processing images in complex cameras

- Raw data received in system memory
- Image data processed in ISP (Image Signal Processor)
- Camera control, including control algorithms in user space
- libcamera
Bus view, with camera related information

- CSI-2 bus also requires connected lanes, PHY type etc.
- Parallel buses have configuration as well
Camera support in Devicetree and ACPI firmware
Device tree

- System hardware description
- Originates from Sparc / Open Firmware
- Used on a variety on architectures
  - ARM{,64}, MIPS, PowerPC, Sparc and some x86 systems
- Tree structure
  - Nodes
  - Properties
- Source code compiled into binary (DTB) before use
Devicetree camera support

• First firmware type to support complex cameras
  − Since 2012

• Bindings defined in dt-schema and kernel
  − Documentation/devicetree/bindings/media/video-interface-devices.yaml
  − Documentation/devicetree/bindings/media/video-interfaces.yaml
  − Documentation/devicetree/bindings/leds/common.yaml
  − Additionally device specific bindings
ACPI

• Advanced Configuration and Power Interface
• Operating system independent
• Virtually every PC nowadays
  - Also found in other architectures such as ARM64
• Device discovery and enumeration
• Power management
• AML code run in ACPI virtual machine
ACPI _DSD

- ACPI supports Devicetree-like nodes and properties via _DSD objects
- String references to other nodes
ACPI camera support – Linux specific

• Graph and LED definitions amended by
  – Documentation/firmware-guide/acpi/dsd/graph.rst
  – Documentation/firmware-guide/acpi/dsd/leds.rst

• Otherwise uses DT definitions

• Used in many x86 Chromebooks

• _DSD only

• Support added in 2017
ACPI camera support – IPU3 specific

- IPU3 specific binary data structures in a few custom ACPI objects
- Found in Skylake and Kaby lake laptops shipped with Windows
- Enables cameras in many laptops
- Some information not available from firmware
- Power management in drivers
- Support added in 2019
ACPI _CRS

- Used to describe various things such as serial buses
- CSI-2 connection resource descriptor
- Partially describes a CSI-2 bus
- Added in ACPI 6.4
ACPI camera support – MIPI
DisCo for Imaging

• DisCo – Series of MIPI specifications defining ACPI firmware interfaces for MIPI hardware standards

• ACPI _CRS and _DSD

• Vendor and operating system independent

• Expected to be released later this year
Parsing camera related information from firmware
Driver developer’s problem

```c
if (is_of_node(dev_fwnode(dev)))
    ret = parse_of();
else if (is_acpi_node(dev_fwnode(dev)))
    ret = parse_acpi();
else
    ret = -ENODEV;

```
Parsing camera definitions on ACPI

• Non-Linux specific definitions for ACPI rely at least partially on binary data structures
  – Little or no similarity with Devicetree

• Parser determined based on ACPI objects as well as _DSD data nodes and properties
V4L2 fwnode framework

- Helps drivers in parsing information from firmware
- Origins in V4L2 OF framework
- Uses fwnode property API to parse Devicetree nodes and properties
v4l2_fwnode_endpoint_alloc_parse()

- Main function to access Devicetree graph endpoint specific information for sub-devices
- Parses Devicetree compliant data structure of nodes and properties
- v4l2_fwnode_endpoint_free()
ACPI camera definition parsing – Linux specific

- Fwnode property API ACPI backend
- Graph parsing code for ACPI _DSD nodes and properties
- A few additional checks in graph parsing code in drivers/acpi/properties.c for camera definitions
Software nodes

- Software provide kernel-initialised nodes and properties
- Typically created by other drivers
- Fwnode property API software node backend
- Maybe added to existing fwnodes
ACPI camera definition parsing – IPU3 specific

- No similarity with Devicetree definitions
- Parsing code located in
  - drivers/media/pci/intel/ipu3/cio2-bridge.c
  - drivers/platform/x86/intel/int3472/
- New nodes instantiated and properties added based on parsed binary data structure
- Regulators and clocks created
ACPI camera definition parsing – MIPI DisCo for Imaging

• Combination of ACPI _CRS binary data structure and _DSD nodes and properties

• Software nodes and properties to be created from information in binary data structures

• Yet again the result will be Devicetree-like view to V4L2 fwnode framework!
Summary of differences in complex camera firmware parsing

**Devicetree**

**ACPI – Linux**
- Parse custom binary data structure
- Create software nodes and properties
- Create regulators and clocks

**ACPI – IPU3**
- Parse ACPI _CRS data structure and _DSD properties
- Create software nodes and properties

**ACPI – DisCo**

**Devicetree compliant**
Driver probe example

```c
int driver_probe(struct i2c_client *client)
{
    struct fwnode_handle *fwnode = dev_fwnode(dev);
    struct fwnode_handle *endpoint;
    struct v4l2_fwnode_endpoint vep = { .bus_type = V4L2_MBUS_MIPI_DPHY };,
    int ret;

    endpoint =
        fwnode_graph_get_endpoint_by_id(fwnode, 0 /* port */, 0 /* endpoint */, FWNODE_GRAPH_ENDPOINT_NEXT);

    ret = v4l2_fwnode_endpoint_alloc_parse(endpoint, &vep);
    fwnode_handle_put(endpoint);
    if (ret)
        return ret;

    /* configuration stored in vep */
    v4l2_fwnode_endpoint_free(&vep);

    return ret;
}
```
struct v4l2_fwnode_endpoint {
    struct fwnode_endpoint base;
    enum v4l2_mbus_type bus_type;
    struct {
        struct v4l2_mbus_config_parallel parallel;
        struct v4l2_mbus_config_mipi_csi1 mipi_csi1;
        struct v4l2_mbus_config_mipi_csi2 mipi_csi2;
    } bus;
    u64 *link_frequencies;
    unsigned int nr_of_link_frequencies;
};
struct v4l2_mbus_config_mipi_csi2 {
    unsigned int flags;
    unsigned char data_lanes[V4L2_MBUS_CSI2_MAX_DATA_LANES];
    unsigned char clock_lane;
    unsigned char num_data_lanes;
    bool lane_polarities[1 + V4L2_MBUS_CSI2_MAX_DATA_LANES];
};
Future work
Camera module

• Camera module typically consists of
  - Camera sensor
  - Lens and lens VCM
  - IR filter

• Camera control algorithms need detailed information of the properties of the camera module

• Camera module not visible in firmware currently
Camera module, continued

• Tuning parameters
  – Lens shading table (unless in sensor EEPROM)
  – Other sensor parameters
  – VCM current limits

• Camera module name in DT/ACPI?

• Information on the camera module itself outside kernel
Power management in ACPI and Devicetree based systems

- ACPI handles power management via _PSx methods and power resources (_PRx)
- In Devicetree systems the driver is responsible for its power on and off sequences
  - Regulators, GPIOs and clocks available to the driver
- Runtime PM supported if CONFIG_PM option enabled
I²C device power state in driver probe and remove

- I²C devices are powered on for driver probe ACPI based systems
- Off in Devicetree based systems
  - No other option as the driver acquires resources for powering on and off the device during its probe function
- This causes significant complications in writing drivers
- In principle the problem touches all I²C drivers that support both Devicetree and ACPI
  - A large number of such drivers are camera sensor drivers
I²C device power state in driver probe and remove (continued)

- Further complicated by supporting low power state probe
  - Privacy LED wired to camera sensor power supply
  - ACPI only for now
  - Devicetree support?
- All combinations of
  - CONFIG_PM enabled or disabled
  - Runtime PM may be disabled through sysfs when CONFIG_PM is enabled
  - DT vs. ACPI vs. ACPI but in low power state
- On some ACPI systems you may need regulators, clocks or GPIOs
Example of power management in driver probe function

```c
int driver_probe(struct i2c_client *client)
{
    int ret, full_power = acpi_dev_state_d0(&client->dev));

    if (full_power) {
        ret = power_on_device(&client->dev);
        if (ret)
            return -ENODEV;
        ret = identify_device(&client->dev);
        if (ret)
            goto err_power_off;
        pm_runtime_set_active(&client->dev);
    }

    pm_runtime_enable(&client->dev);
    pm_runtime_idle(&client->dev);
    return 0;

err_power_off:
    if (!pm_runtime_status_suspended(&client->dev))
        power_off_device(&client->dev);
    return ret;
}
```

Copyright © Intel Corporation 2022
Thank you!