Cameras in embedded systems: Device tree and ACPI view

Sakari Ailus - Intel
A typical embedded system with a camera

- Image Signal Processor
- Raw camera sensor
- Lens voice coil
Raw sensors

- Raw sensors have little processing logic in the sensor itself
  - Analogue and digital gain but not much more

This is how white looks like! -->
Image signal processors

- Process the image for viewing

After ISP processing white looks like this --->
Video4Linux and Media controller

- Video4Linux (V4L2) is the Linux API for capturing images
  - Video capture cards
  - USB webcams
  - Cameras in embedded devices
- Media controller is a control interface for complex media devices
  - Image pipeline discovery and configuration
  - Device discovery
Example of a media graph
Probing

• Each driver is probed separately

• How to tell drivers they all are part of the same media device?
Media device setup

**Media device driver**

1. `media_device_init()`
2. `v4l2_device_register()`
3. `video_register_device()`
4. `v4l2_device_register_subdev(sensor)`
5. `v4l2_device_register_subdev(isp)`
6. `v4l2_register_subdev_nodes()`
7. `media_device_register()`

**Sensor driver**
V4L2 async
V4L2 async

- The V4L2 async framework facilitates sub-device registration
- V4L2 sub-device device node creation and media device registration postponed after probe
- To do its job, the V4L2 async framework makes use of firmware provided information
V4L2 async example (ISP)

- prob(dev)
- parse local endpoints
- of_graph_get_remote_port_parent(endpoint)
- v4l2_async_register_notifier(notifier)
- Look for sub-devices matching the notifier list
- No match found
- Add ISP notifier
V4L2 async (sensor)

- probe(dev)
- v4l2_of_parse_endpoint()
- v4l2_async_register_subdev(sd)
- Look for a notifier matching the sub-device
- Found it!
- notifier->bound()
- notifier->complete()
Device tree
Device tree

- System hardware description in a human readable format
- Originates from Sparc / Open Firmware
- Primarily used on embedded systems
  - ARM
  - Also PowerPC, Sparc and x86
- Tree structure
  - Nodes
  - Properties
- Source code compiled into binary before use
Device Tree standard and bindings

- Device Tree specification maintained by devicetree.org
  - Syntax and some semantics
- Bindings define the interface between the firmware and the software
  - Bindings are Operating System specific
  - Linux Device tree binding documentation part of the Linux kernel source
  - FreeBSD developers appear to be converging towards using Linux DT bindings
Device tree graphs

- **phandle** properties can be used to refer to other nodes in the tree
- Port is an interface in a device (as in an IP block)
- Endpoint describes one end of a connection to a port [7]
Sensor node

си2с2 {
    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)>;
        port {
            smia_1_1: endpoint {
                link-frequencies = /bits/ 64 <199200000 210000000 499200000>;
                clock-lanes = <0>;
                data-lanes = <1 2>;
                remote-endpoint = <&csi2a_ep>;
            }
        }
    }
};

source: arch/arm/boot/dts/omap3-n9.dts
ISP node board specific part

&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 = <1 3>;
                crc = <1>;
                lane-polarities = <1 1 1>;
            }
        }
    }
};

source: arch/arm/boot/dts/omap3-n9.dts
OF graph API

- Parse port and endpoint nodes under device nodes
- Enumerate over endpoints
- Obtain remote endpoint
  - Based on the phandle value
ACPI

- Advanced Configuration and Power Interface
- Operating system independent
- Origins in x86 and PC
  - Increasingly used in embedded systems
- Device discovery and enumeration
- **Power management**
- ACPI methods
  - Runnable code
  - ACPI virtual machine
ACPI

- ACPI specifications developed by UEFI Forum
  - Roughly one specification per year

- What do you do if you need to add a new kind of a device?
  - A new ACPI specification?
ACPI Device Specific Data

- _DSD object type part of ACPI 5.1 and later
  - Key-value pairs (property extension) and
  - Tree structures (hierarchical data extension)

- Together property and data extension could be used to implement very similar functionality to Devicetree

- _DSD property registry [6]
  - Light-weight approach for registering _DSD properties
fwnode property API

- Access properties independently of underlying firmware implementation
  - Device Tree
  - ACPI
- Makes use of ACPI _DSD property extension [2]
Future work
Fwnode graph API

- Functionally the same as the OF graph API
  - But is firmware independent
- Device tree implementation is used on Device tree
- Makes use of the _DSD hierarchical data extension [3] on ACPI
- Implementation at RFC level [4]
V4L2 fwnode API

- "V4L2 ACPI support"
  - Embedded systems with I²C components
  - Requires both fwnode graph API and V4L2 fwnode API
- Same functionality as V4L2 OF API
- V4L2 fwnode and V4L2 OF fully interoperable
  - Sub-device driver using V4L2 fwnode works with a media device driver using V4L2 OF
  - and vice versa!
- RFC implementation available [5]
Flash

- LED flash devices supported
- But the kernel has no knowledge which sensor they're related to
  - This is rather important if there are multiple cameras in the system, such as most mobile phones nowadays
- Standardise phandle property for this?
Camera module

- Currently there's no "camera module" concept in the kernel (nor DT or ACPI)
- Camera module construction is important for the user space
  - Which sensor and lens are related?
  - What kind of lens is there?
  - What's the voice coil spring constant?
  - Is there an infra red filter? What kind of filter is it?
  - What's the aperture size?
Camera module power on and power off sequences

- Regulators, clocks and/or GPIOs may be shared between module components

- Power on and power off sequences device component specific
  - Which order and when each resource may be enabled?
  - E.g. regulator and clock are enabled, then after 10 ms the reset GPIO can be lifted and the device is ready for use

- Requirements of both lens and sensor must be considered for module power-up sequence
Questions?
References

[7] Documentation/devicetree/bindings/graph.txt
Firmware logistics

**Device tree**

- Linux kernel
- dts
- SoC vendor
- System vendor
- OS binaries
- Flash memory
- Device tree

**ACPI**

- SoC vendor
- BIOS vendor
- System vendor
- BIOS in flash memory
- Runtime ACPI tables
- OS binaries
- Selective table replacement
- (initrd)
- Motherboard
- Website
- Support
- Append kernel image
ACPI camera example
ACPI camera example

Scope (_SB.PCl0.12C2)
{
  Device (CAM0)
  {
    Name (_DSD, Package () {
      /* device specific data */
      Package () {
        Package () { "compatible", Package () { "nokia,smia" } },
        Package () { "lanes", 4 },
        Package () { "clock-frequency", 24000000 },
      },
      /* data extension */
      Package () {
        Package () { "ports", "PRTS" },
      }
    })
  }
}
ACPI camera example

Name (PRTS, Package() {
        /* data extension */
        Package () {
                Package () { "port@0", "PRT0" },
        }
    })
Name (PRT0, Package() {
        /* device specific data */
        Package () {
                Package () { "port", 0 },
        },
        /* data extension */
        Package () {
                Package () { "endpoint@0", "EP0" },
        }
    })
ACPI camera example

Name (EP0, Package() {
    /* device specific data */
    Package () {
        Package () { "endpoint", 0 },
        Package () { "clock-lanes", 0 },
        Package () { "data-lanes", Package () { 1, 2, 3, 4 } },
        Package () { "link-frequencies",
            Package () { 209600000, 342000000, 451200000 } },
        Package () { "remote-endpoint", Package() { \_SB.PCI0.ISP, 0, 0, 0 } },
    }
})
}
ACPI ISP example

Scope (_SB.PCl0)
{
  Device (ISP)
  {
    Name (_DSD, Package () {
      /* data extension */
      Package () {
        Package () { "ports", "PRTS" },
      }
    })
  };
  Name (PRTS, Package() {
    /* data extension */
    Package () {
      Package () { "port@4", "PRT4" },
    }
  })
}
ACPI ISP example

Name (PRT4, Package() {
    /* device specific data */
    Package () {
        Package () { "port", 4 }, /* CSI-2 port number */
    }, /* data extension */
    Package () {
        Package () { "endpoint@0", "EP0" },
    }
})
Name (EP0, Package() {
    /* device specific data */
    Package () {
        Package () { "endpoint", 0 },
        Package () { "clock-lanes", 0 },
        Package () { "data-lanes", Package () { 1, 2, 3, 4 } },
        Package () { "remote-endpoint", Package () { \_SB.PCI0.I2C2.CAM0, 0, 0, 0 } },
    }
})