# UDC Driver

## aotg\_init()

module\_init(aotg\_init);

static int \_\_init aotg\_init(void)

{

mutex\_init(&aotg\_onoff\_mutex);

td\_cache = kmem\_cache\_create("aotg\_hcd", sizeof(struct aotg\_td), 0, SLAB\_HWCACHE\_ALIGN|SLAB\_PANIC, NULL);

**platform\_driver\_register**(&aotg\_hcd\_driver);

create\_acts\_hcd\_proc();

**platform\_driver\_register**(&aotg\_udc\_driver);

create\_acts\_udc\_proc();

**aotg\_udc\_add**();

start\_mon\_wq = create\_singlethread\_workqueue("aotg\_start\_mon\_wq");

INIT\_DELAYED\_WORK(&start\_mon\_wker, **start\_mon**);

queue\_delayed\_work(start\_mon\_wq, &start\_mon\_wker, msecs\_to\_jiffies(10000));

return 0;

}

### struct platform\_driver aotg\_hcd\_driver //aotg hcd driver

struct platform\_driver aotg\_hcd\_driver = {

**.probe = aotg\_probe,**

.remove = aotg\_remove,

.shutdown = aotg\_hcd\_shutdown,

.driver = {

.owner = THIS\_MODULE,

.name = platform\_drv\_name, // "aotg\_hcd"

.of\_match\_table = **aotg\_of\_match**,

#ifdef CONFIG\_PM

.pm = DEV\_PM\_OPS,

#endif

},

};

### struct platform\_driver aotg\_udc\_driver //aotg udc driver

struct platform\_driver aotg\_udc\_driver = {

.driver = {

.name = "aotg\_udc",

.owner = THIS\_MODULE,

.of\_match\_table = **aotg\_of\_match**,

},

**.probe = aotg\_probe,**

.remove = aotg\_remove,

};

### struct of\_device\_id aotg\_of\_match

struct of\_device\_id **aotg\_of\_match**[] = {

{.compatible = "actions,s700-usb2.0-0"}, //S700

{.compatible = "actions,s700-usb2.0-1"}, //S700

{.compatible = "actions,s900-usb2.0-0"}, //S900

{.compatible = "actions,s900-usb2.0-1"}, //S900

{},

};

### actions,s700-usb2.0-0 && actions,s700-usb2.0-1

usb2h0: usb@e01d0000 {

compatible = "actions,s700-usb2.0-0";

reg = <0 0xe01d0000 0 0x1000>;

interrupts = <GIC\_SPI 24 IRQ\_TYPE\_LEVEL\_HIGH>;

power-domains = <&powergate POWER\_DOMAIN\_USB2H0>;

clocks = <&clock CLK\_USB2H0\_PLLEN>, <&clock CLK\_USB2H0\_PHY>, <&clock CLK\_USB2H0\_CCE>;

clock-names = "usbh0\_pllen", "usbh0\_phy", "usbh0\_cce";

resets = <&reset RESET\_USBH0>;

reset-names = "usb2h0";

};

usb2h1: usb@e01d8000 {

compatible = "actions,s700-usb2.0-1";

reg = <0 0xe01d8000 0 0x1000>;

interrupts = <GIC\_SPI 61 IRQ\_TYPE\_LEVEL\_HIGH>;

power-domains = <&powergate POWER\_DOMAIN\_USB2H1>;

clocks = <&clock CLK\_USB2H1\_PLLEN>, <&clock CLK\_USB2H1\_PHY>, <&clock CLK\_USB2H1\_CCE>;

clock-names = "usbh1\_pllen", "usbh1\_phy", "usbh1\_cce";

resets = <&reset RESET\_USBH1>;

reset-names = "usb2h1";

};

## aotg\_probe()

struct aotg\_plat\_data aotg\_data[2];

int aotg\_probe(struct platform\_device \*pdev)

{

aotg\_hcd\_get\_dts(pdev); **//pdev->id= [0: "actions,s700-usb2.0-0", 1:"actions,s700-usb2.0-1"]**

struct resource \*res\_mem = platform\_get\_resource(pdev, IORESOURCE\_MEM, 0);

request\_mem\_region(res\_mem->start, res\_mem->end - res\_mem->start + 1, dev\_name(&pdev->dev));

aotg\_data[pdev->id].base = devm\_ioremap(&pdev->dev, res\_mem->start, res\_mem->end - res\_mem->start + 1);

aotg\_plat\_data\_fill(&pdev->dev, pdev->id); //aotg\_data[pdev->id].usbecs = [S700: 0xE024c094, 0xE0228094]

aotg\_data[pdev->id].rsrc\_start = res\_mem->start;

aotg\_data[pdev->id].rsrc\_len = res\_mem->end - res\_mem->start + 1;

pdev->dev.dma\_mask = &aotg\_dmamask;

pdev->dev.coherent\_dma\_mask = DMA\_BIT\_MASK(32);

aotg\_data[pdev->id].**irq = platform\_get\_irq(pdev, 0);**

aotg\_data[pdev->id].dev = &pdev->dev;

device\_init\_wakeup(&pdev->dev, true);

if (pdev->id) aotg\_data[pdev->id].clk\_usbh\_pllen = devm\_clk\_get(&pdev->dev, "usbh1\_pllen");

else aotg\_data[pdev->id].clk\_usbh\_pllen = devm\_clk\_get(&pdev->dev, "usbh0\_pllen");

if (pdev->id) aotg\_data[pdev->id].clk\_usbh\_phy = devm\_clk\_get(&pdev->dev, "usbh1\_phy");

else aotg\_data[pdev->id].clk\_usbh\_phy = devm\_clk\_get(&pdev->dev, "usbh0\_phy");

if (ic\_type[pdev->id] == S900) {

if (pdev->id) aotg\_data[pdev->id].clk\_usbh\_cce = devm\_clk\_get(&pdev->dev, "usbh1\_cce");

else aotg\_data[pdev->id].clk\_usbh\_cce = devm\_clk\_get(&pdev->dev, "usbh0\_cce");

}

}

# Webcam

## webcam\_init()

module\_init(webcam\_init);

static int \_\_init webcam\_init(void)

{

return **usb\_composite\_probe**(&webcam\_driver);

}

### struct usb\_composite\_driver

struct usb\_composite\_driver {

const char \*name; // "g\_webcam"

const struct usb\_device\_descriptor \*dev;

struct usb\_gadget\_strings \*\*strings;

enum usb\_device\_speed max\_speed; // USB\_SPEED\_SUPER

unsigned needs\_serial:1;

…

struct usb\_gadget\_driver gadget\_driver; //由usb\_composite\_probe()初始化

};

### struct usb\_composite\_driver webcam\_driver

static \_\_refdata struct usb\_composite\_driver webcam\_driver = {

.name = "g\_webcam",

.dev = &webcam\_device\_descriptor,

.strings = webcam\_device\_strings,

.max\_speed = USB\_SPEED\_SUPER,

.bind = webcam\_bind,

.unbind = webcam\_unbind,

};

### struct usb\_device\_descriptor webcam\_device\_descriptor

#define WEBCAM\_VENDOR\_ID 0x1d6b /\* Linux Foundation \*/

#define WEBCAM\_PRODUCT\_ID 0x0102 /\* Webcam A/V gadget \*/

#define WEBCAM\_DEVICE\_BCD 0x0010 /\* 0.10 \*/

static struct usb\_device\_descriptor webcam\_device\_descriptor = {

.bLength = USB\_DT\_DEVICE\_SIZE, //18

.bDescriptorType = USB\_DT\_DEVICE, //0x01

.bcdUSB = cpu\_to\_le16(0x0200),

.bDeviceClass = USB\_CLASS\_MISC, //0xef

.bDeviceSubClass = 0x02,

.bDeviceProtocol = 0x01,

.bMaxPacketSize0 = 0, /\* dynamic \*/

.idVendor = cpu\_to\_le16(WEBCAM\_VENDOR\_ID),

.idProduct = cpu\_to\_le16(WEBCAM\_PRODUCT\_ID),

.bcdDevice = cpu\_to\_le16(WEBCAM\_DEVICE\_BCD),

.iManufacturer = 0, //"Linux Foundation"

.iProduct = 0, //"Webcam gadget"

.iSerialNumber = 0, /\* dynamic \*/

.bNumConfigurations = 0, /\* dynamic \*/

};

### struct usb\_gadget\_strings \*webcam\_device\_strings

static struct usb\_gadget\_strings \*webcam\_device\_strings[] = {

&webcam\_stringtab,

NULL,

};

static struct usb\_gadget\_strings webcam\_stringtab = {

.language = 0x0409, /\* en-us \*/

.strings = webcam\_strings,

};

static struct usb\_string webcam\_strings[] = {

[USB\_GADGET\_MANUFACTURER\_IDX].s = webcam\_vendor\_label,

[USB\_GADGET\_PRODUCT\_IDX].s = webcam\_product\_label,

[USB\_GADGET\_SERIAL\_IDX].s = "",

[STRING\_DESCRIPTION\_IDX].s = webcam\_config\_label,

{ }

};

static char webcam\_vendor\_label[] = "Linux Foundation";

static char webcam\_product\_label[] = "Webcam gadget";

static char webcam\_config\_label[] = "Video";

## usb\_composite\_probe()

int usb\_composite\_probe(struct usb\_composite\_driver \*driver)

{

struct usb\_gadget\_driver \*gadget\_driver;

driver->gadget\_driver = composite\_driver\_template;

gadget\_driver = &driver->gadget\_driver;

gadget\_driver->function = (char \*) driver->name; //"g\_webcam"

gadget\_driver->driver.name = driver->name; //"g\_webcam"

gadget\_driver->max\_speed = driver->max\_speed;

return **usb\_gadget\_probe\_driver**(gadget\_driver);

}

### struct usb\_gadget\_driver

struct usb\_gadget\_driver {

char \*function; //"g\_webcam"

enum usb\_device\_speed max\_speed;

…

struct device\_driver driver; // driver.name = "g\_webcam"

};

### struct usb\_gadget\_driver composite\_driver\_template

static const struct usb\_gadget\_driver composite\_driver\_template = {

.bind = composite\_bind,

.unbind = composite\_unbind,

.setup = composite\_setup,

.disconnect = composite\_disconnect,

.suspend = composite\_suspend,

.resume = composite\_resume,

.driver = {.owner = THIS\_MODULE,},

};

## usb\_gadget\_probe\_driver()

int usb\_gadget\_probe\_driver(struct usb\_gadget\_driver \*driver)

{

struct usb\_udc \*udc = NULL;

list\_for\_each\_entry(udc, &udc\_list, list)

if (!udc->driver) goto found;

return -ENODEV;

found:

return **udc\_bind\_to\_driver**(udc, driver);

}

## udc\_bind\_to\_driver()

static int udc\_bind\_to\_driver(struct usb\_udc \*udc, struct usb\_gadget\_driver \*driver)

{

udc->driver = driver;

udc->dev.driver = &driver->driver;

udc->gadget->dev.driver = &driver->driver;

ret = **driver->bind**(udc->gadget, driver); // composite\_bind()

ret = usb\_gadget\_udc\_start(udc->gadget, driver);

**kobject\_uevent**(&udc->dev.kobj, **KOBJ\_CHANGE**);

return 0;

}

### struct usb\_udc

udc->gadget->dev.driver = udc->dev.driver = udc->driver = webcam\_driver. gadget\_driver

webcam\_driver. gadget\_driver由usb\_composite\_probe()初始化

struct usb\_udc {

struct usb\_gadget\_driver \*driver; //由usb\_composite\_probe()初始化

struct usb\_gadget \*gadget;

struct device dev;

struct list\_head list;

};

# End