in ccsgit/driver/sp_virtual_uart.c

```
/*
 * Called when the driver is loaded (insmod).
 */
static int __init sp_virtual_uart_driver_init(void)
{
    int error;

    /* Register the platform device with the kernel */
    error = platform_device_register(&sp_virtual_uart_device);
    if (error)
        return error;

    // Get a handle for the IPC interface
    port_handle = ipc_attach(IPC0_DEVICE, IPC_PORT_SP_VIRUAL_UART, recv_callback, NULL);
    if (port_handle == NULL)
        printk("%s: ipc_attach failed\n", __func__);

    ipc_send( port_handle, 0, (void *)1, 0 );  // bit 0, turn on RX (SP:R4 -> AP) pipe
    /*
     * Register the platform driver with the kernel.  This will cause the
     * platform driver's probe function to be invoked.
     */
    return platform_driver_register(&sp_virtual_uart_driver);
}
```

在 `init` function中create platform device and create platform driver,这很少见。在enable device tree supoort的情况下，driver只负责 `driver register` 。

这里device与driver的match是通过各自structure中的.name field来实现的。

```c
/*
 * Defines a platform driver structure that can be passed to
 * platform_driver_register() to associate a dtiver with a named device.
 */
static struct platform_device sp_virtual_uart_device  = {
    .name = MY_DRIVER_NAME,
    .id   = 0,
    .num_resources = 0,
    .resource = 0,
};

/*
 * Defines a platform driver structure that can be passed to
 * platform_driver_register() to associate a driver with a named device.
 */
static struct platform_driver sp_virtual_uart_driver = {
    .probe  = driver_probe,
    .remove = driver_remove,
    .driver = {
        .name   = MY_DRIVER_NAME,
        .owner  = THIS_MODULE,
    },
};
```

具体match的逻辑见drivers/base/platform.c

```c
/**
 * platform_match - bind platform device to platform driver.
 * @dev: device.
 * @drv: driver.
 *
 * Platform device IDs are assumed to be encoded like this:
 * "<name><instance>", where <name> is a short description of the type of
 * device, like "pci" or "floppy", and <instance> is the enumerated
 * instance of the device, like '0' or '42'.  Driver IDs are simply
 * "<name>".  So, extract the <name> from the platform_device structure,
 * and compare it against the name of the driver. Return whether they match
 * or not.
 */
static int platform_match(struct device *dev, struct device_driver *drv)
{
    struct platform_device *pdev = to_platform_device(dev);
    struct platform_driver *pdrv = to_platform_driver(drv);

    /* When driver_override is set, only bind to the matching driver */
    if (pdev->driver_override)
        return !strcmp(pdev->driver_override, drv->name);

    /* Attempt an OF style match first */
    if (of_driver_match_device(dev, drv))
        return 1;

    /* Then try ACPI style match */
    if (acpi_driver_match_device(dev, drv))
        return 1;

    /* Then try to match against the id table */
    if (pdrv->id_table)
        return platform_match_id(pdrv->id_table, pdev) != NULL;

    /* fall-back to driver name match */
    return (strcmp(pdev->name, drv->name) == 0);
}
```

这里sp_virtual_uart driver会命中的是下面的逻辑

```c
    /* fall-back to driver name match */
    return (strcmp(pdev->name, drv->name) == 0);
```