sign up log in tour help stack overflow careers

How to write a simple device driver in Linux?

I need to write an SPI Linux device driver for omap4 from scratch. I know some basics of writing device drivers. But, I don't know how to start writing platform specific device driver from scratch. Can somebody please help me.

I've written some basic char drivers and I thought writing SPI device driver would be similar to it. But, char driver has a structure file_operations which contains the functions implemented in the driver.

```
struct file_operations Fops = {
    .read = device_read,
    .write = device_write,
    .ioctl = device_ioctl,
    .open = device_open,
    .release = device_release,  /* a.k.a. close */
};
```

Now I am going through spi-omap2-mcspi.c code as reference to get an idea to start writing my own driver for spi.

But I don't see functions such as open, read, write etc. Don't know from where the program starts.

Kindly provide me with some insight into writing platform drivers.

c linux linux-kernel device-driver spi

edited Jan 8 at 8:38 asked Mar 25 '14 at 11:07
Sagar Jain
731 3 23

Just a question: Why do you want to rewrite the SPI driver? I have used the OMAP4 SPI driver before and didn't had any problems with it. – Nils Pipenbrinck Aug 13 '14 at 9:45

1 @NilsPipenbrinck: The main purpose of writing the driver is to learn. – Sagar Jain Aug 13 '14 at 10:02

3 Answers

First start by writing a generic kernel module. There are multiple places to look up for information but I found this link to be very useful. After you have gone through all examples specified there you can start writing your own Linux Driver Module.

Please note, that you will not get away with just copy-pasting the example code and hope it will work, no. Kernel API can sometimes change and examples will not work. Examples provided there should be looked at as a guide how to do something. Depending on the kernel version you are using you have to modify the example in order to work.

Consider using TI platform provided functions as much as you can, because that can really do a lot work for you, like requesting and enabling needed clocks, buses and power supplies. If I recall correctly you can use the functions to acquire memory mapped address ranges for direct access to registers. I have to mention that I have bad experience with TI provided functions because they do not properly release/clean-up all acquired resources, so for some resources I had to call other kernel services to release them during module unload.

Edit 1:

I'm not entirely familiar with Linux SPI implementation but I would start by looking at omap2_mcspi_probe() function in drivers/spi/spi-omap2-mcspi.c file. As you can see there, it registers it's methods to Linux master SPI driver using this API: Linux/include/linux/spi/spi.h. In contrast to char driver the main functions here are *_transfer() functions. Look up at the struct descriptions in spi.h file for further details. Also, have a look at this alternative device driver API, too.

edited Mar 27 '14 at 21:30

answered Mar 25 '14 at 19:23
Nenad Radulovic
176 1 4

I assume your OMAP4 linux uses one of `arch/arm/boot/dts/{omap4.dtsi,am33xx.dtsi}` device-tree, thus it compiles `drivers/spi/spi-omap2-mcspi.c` (if you don't know about device-tree, read this). Then:

- the SPI master driver is done,
- it (most probably) registers with Linux SPI core framework `drivers/spi/spi.c` ,
- it (probably) works fine on your OMAP4.

You actually don't need to care about the *master driver* to write your *slave device driver*. How do I know `spi-omap2-mcspi.c` is a master driver? It calls `spi_register_master()` .

SPI master, SPI slave ?

Please refer to `Documentation/spi/spi_summary` . The doc refers to *Controller driver* (master) and *Protocol drivers* (slave). From your description, I understand you want to write a *Protocol/Device driver*.

SPI protocol ?

To understand that, you need your slave device datasheet, it shall tell you:

- the **SPI mode** understood by your device,
- the **protocol** it expects on the bus.

Contrary to i2c, SPI does not define a protocol or handshake, SPI chips manufacturers have to define their own. So check the datasheet.

SPI mode

From `include/linux/spi/spi.h` :

```
 * @mode: The spi mode defines how data is clocked out and in.
 *  This may be changed by the device's driver.
 *  The "active low" default for chipselect mode can be overridden
 *  (by specifying SPI_CS_HIGH) as can the "MSB first" default for
 *  each word in a transfer (by specifying SPI_LSB_FIRST).
```

Again, check your SPI device datasheet.

An example SPI device driver?

To give you a relevant example, I need to know your SPI device type. You would understand that a **SPI flash device driver** is different from a **SPI FPGA device driver**. Unfortunately there are not so many SPI device drivers out there. To find them:

```
$ cd linux
$ git grep "spi_new_device\|spi_add_device"
```

answered Jun 19 '14 at 3:25
m-ric
1,486 1 14 30

I don't know if I understood your question correctly. As m-ric pointed out, there are master drivers and slave drivers.

Usually master drivers are more hardware bound, I mean, they usually manipulate IO registers or do some memory mapped IO.

For some architectures already supported by linux kernel (like omap3 and omap4) master drivers are already implemented (McSPI).

So I assume you want to USE those SPI facilities of omap4 to implement a slave device driver (your protocol, to communicate with your external device through SPI).

I've written the following example for BeagleBoard-xM (omap3). The full code is at https://github.com/rslemos/itrigue/blob/master/alsadriver/itrigue.c (worth a view, but have more initialisation code, for ALSA, GPIO, module parameters). I've tried to set apart code that deals with SPI (maybe I forgot something, but anyway you should get the idea):

```
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/spi/spi.h>

/* MODULE PARAMETERS */
static uint spi_bus = 4;
static uint spi_cs = 0;
static uint spi_speed_hz = 1500000;
static uint spi_bits_per_word = 16;

/* THIS IS WHERE YOUR DEVICE IS CREATED; THROUGH THIS YOU INTERACT WITH YOUR EXTERNAL
DEVICE */
static struct spi_device *spi_device;


/* SETUP SPI */

static inline __init int spi_init(void) {
    struct spi_board_info spi_device_info = {
        .modalias = "module name",
        .max_speed_hz = spi_speed_hz,
        .bus_num = spi_bus,
        .chip_select = spi_cs,
        .mode = 0,
    };

    struct spi_master *master;

    int ret;

    // get the master device, given SPI the bus number
    master = spi_busnum_to_master( spi_device_info.bus_num );
    if( !master )
        return -ENODEV;

    // create a new slave device, given the master and device info
    spi_device = spi_new_device( master, &spi_device_info );
    if( !spi_device )
        return -ENODEV;

    spi_device->bits_per_word = spi_bits_per_word;

    ret = spi_setup( spi_device );
    if( ret )
        spi_unregister_device( spi_device );

    return ret;
}

static inline void spi_exit(void) {
    spi_unregister_device( spi_device );
}
```

To write data to your device:

```
spi_write( spi_device, &write_data, sizeof write_data );
```

The above code is independent of implementation, that is, it could use McSPI, bit-banged GPIO, or any other implementation of an SPI master device. This interface is described in `linux/spi/spi.h`

To make it work in BeagleBoard-XM I had to add the following to the kernel command line:

```
omap_mux=mcbsp1_clkr.mcspi4_clk=0x0000,mcbsp1_dx.mcspi4_simo=0x0000,mcbsp1_dr.mcspi4_somi=0x
```

So that an McSPI master device is created for omap3 McSPI4 hardware facility.

Hope that helps.

edited Jun 24 '14 at 1:32

answered Jun 23 '14 at 19:19
rslemos
747 4 10

I want to write a master driver – Sagar Jain Aug 9 '14 at 14:41