LONTIUM SEMICONDUCTOR CORPORATION

LT8911EXB Linux Driver Software configuration and debugging guide

目 录

一 Introduction

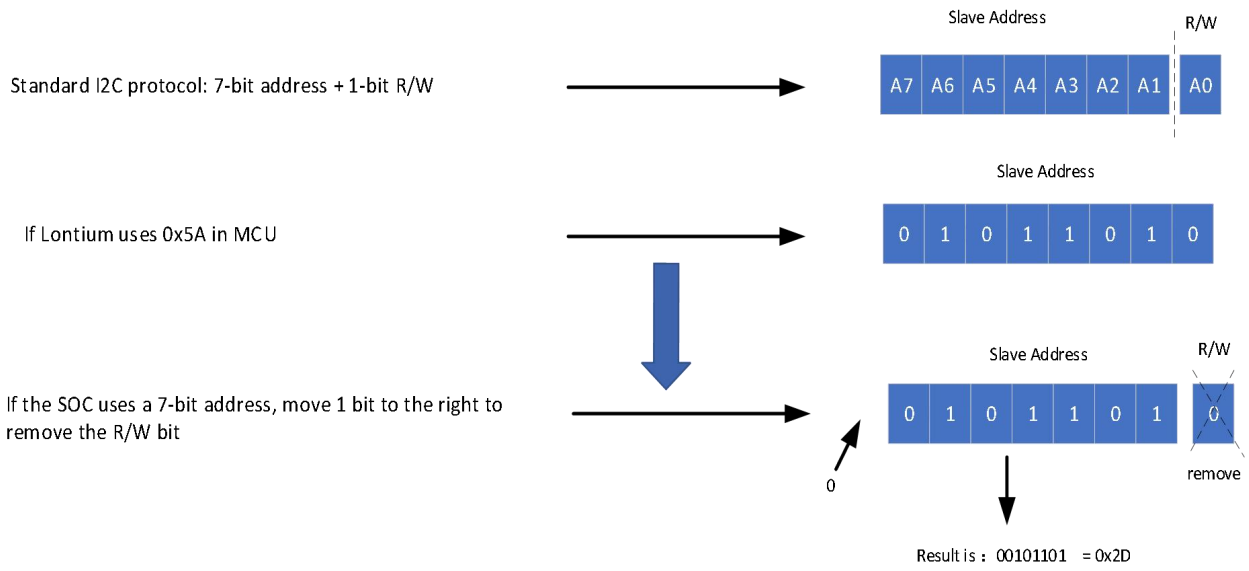
1. The LT8911EXB does not have a built-in MCU, and requires an external MCU or SOC to read and write registers to the chip through the I2C to work normally.
2. The chip register address is 16 bits (the actual read and write operation is based on 8-bit address, the read and write timing can refer to 8-bit address), in which the high 8 bits are the Bank address, and the low 8 bits are the offset address in the Bank.

   For example, chip id register 0x8102, 0x81 is the bank address, and 0x02 is the offset address in the bank. Bank must be specified before register read and write. Register read and write in the same Bank need not be specified repeatedly. Cross-bank operation must first select the corresponding Bank address. As shown in the following figure, HDMI_WriteI2C_Byte(0xff,0x81) specifies that the bank is 0x81, and HDMI_ReadI2C_Byte(0x02) is the register that reads the address 0x02 in 0x81bank.

```
HDMI_WriteI2C_Byte(0xff,0x81);//register bank
HDMI_ReadI2C_Byte(0x02)
```

3. The chip can output internal test images independently of the input signal.
4. The LT8911EXB's I2C address is 8-bit 0x52 and 0x5A used on the MCU, and can be selected by the address pin setting, refer to the reference schematic for details.

   However, when using a 7-bit address on an SOC, the address given by Lontium needs to be shifted one bit to the right, as shown below:



二 Driver File Introduction

① Driver file layout:

```
ysy@ubuntu:~/workspace/imx6ull_pro/Imx6ull_source_code/Linux-4.9.88/drivers/video/LT8911EXB$ tree -L 1
.
├── doc
├── include
├── Kconfig
├── LontiumDrv.c
├── lt8911.c
├── LT8911EXB_Main.c
├── Makefile
└── OcmI2cMaster.c

2 directories, 6 files
ysy@ubuntu:~/workspace/imx6ull_pro/Imx6ull_source_code/Linux-4.9.88/drivers/video/LT8911EXB$
```

The main documents are described as follows:

- LontiumDrv.c: Linux driver entry file. view the driver logic from this file
- LT8911EXB_Main.c: The chip configuration entry file is where the chip configuration logic begins.
- include: Store all header files of the driver.
- Makefile: The configuration file is used to build the driver.
- Kconfig : The configuration file is used to build the driver.
- doc: Directory for storing the introduction document
- Other files: Provides functions that are called during register configuration.

② LontiumDrv.c source code introduction:

```
232 static int chip_resume(struct device *dev)
233 {
234
235     //power on
236     gpiod_set_value(lt8911exb->power_gpio,1);
237     msleep(5);
238     atomic_set(&thread_should_stop, 0);
239     kthread_obj = kthread_run(LT8911EXB_Main, NULL, "LT8911EXB_kthread");
240         printk(KERN_INFO "lt8911exb Resume");
241     return 0;
242 }
243
244
245 static const struct dev_pm_ops chip_pm_ops = {
246     .suspend = chip_suspend,
247     .resume =  chip_resume,
248 };
249
250
251
252
253 static const struct i2c_device_id chip_ids[] = {
254         {"lt8911exb", 0},
255         { }
256 };
257 MODULE_DEVICE_TABLE(i2c, chip_ids);
258
259
260 static const struct of_device_id chip_id_table[] = {
261         {.compatible = "lontium,lt8911exb"},
262         { }
263 };
264 MODULE_DEVICE_TABLE(of, chip_id_table);
265
266
267 static struct i2c_driver chip_driver = {
268         .driver = {
269                 .owner = THIS_MODULE,
270                 .name = "lt8911exb",
271                 .pm = &chip_pm_ops,
272                 .of_match_table = chip_id_table,
273
274         },
275         .probe    = chip_probe,
276         .remove   = chip_remove,
277         .id_table = chip_ids,
278 };
279
```

```
248 &i2c1 {
249     clock-frequency = <100000>;
250     pinctrl-names = "default";
251     pinctrl-0 = <&pinctrl_i2c1>;
252     status = "okay";
253     lt9611uxd:lt9611uxd@41{
254             compatible = "lontium,lt9611uxd";
255             reg = <0x41>;
256             power-gpios =  <&gpio4 21 GPIO_ACTIVE_HIGH>;
257             reset-gpios =  <&gpio4 22 GPIO_ACTIVE_LOW>;
258             interrupt-gpios =  <&gpio4 23 GPIO_ACTIVE_HIGH>;
259     };
260
261     lt8911exb:lt8911exb@29{
262             compatible = "lontium,lt8911exb";
263             reg = <0x29>;
264             power-gpios =  <&gpio4 21 GPIO_ACTIVE_HIGH>;
265             reset-gpios =  <&gpio4 22 GPIO_ACTIVE_LOW>;
266     };
267
268
269 };
```

- "lontium,lt8911exb": Key characters that match the device tree.
- chip_probe : Once the driver matches the device tree when loaded, chip_probe is executed.
- chip_pm_ops: Register the sleep function -chip_suspend and wake function -chip_resume, if

the platform's power cannot use driver control, here you can use the following to control the reset pin:

```
static int chip_suspend(struct device *dev)
{
    atomic_set(&thread_should_stop, 1);
    kthread_stop(kthread_obj);
    msleep(5);
    gpiod_set_value(lt8911exb->reset_gpio,0);        //reset pin Pull low - Low power consumption
    printk(KERN_INFO "LT8911EXB Suspend");
    return 0;
}
static int chip_resume(struct device *dev)
{
    gpiod_set_value(lt8911exb->reset_gpio,1);         //reset pin pull high - Returns to normal
    msleep(5);
    atomic_set(&thread_should_stop, 0);
    kthread_obj = kthread_run(LT8911EXB_Main, NULL, "LT8911EXB_kthread");    //reinitialize LT8911EXB
    printk(KERN_INFO "LT8911EXB Resume");
    return 0;
}
```

③ chip_probe source code introduction:

```
160 static int chip_probe(struct i2c_client *client, const struct i2c_device_id *id)
161 {
162         int ret;
163
164         chip_dev_init();
165
166         lt8911exb = devm_kzalloc(&client->dev, sizeof(*lt8911exb), GFP_KERNEL);
167         if (lt8911exb == NULL)
168                 return -ENOMEM;
169
170         lt8911exb->trans_i2c = client;
171
172         lt8911exb->dev = &client->dev;
173
174         lt8911exb->chip_regmap = devm_regmap_init_i2c(client, &chip_regmap_config);
175         if (IS_ERR(lt8911exb->chip_regmap)) {
176         dev_err(&client->dev, "Failed to initialize regmap\n");
177         return PTR_ERR(lt8911exb->chip_regmap);
178     }
179
180         ret = chip_parse_dts(lt8911exb->dev);
181         if (ret<0){
182                 dev_err(&client->dev, "Failed to parse device tree\n");
183                 return ret;
184         }
185
186         i2c_set_clientdata(client, lt8911exb);
187
188         kthread_obj = kthread_run(LT8911EXB_Main, NULL, "LT8911EXB_kthread");
189         if (IS_ERR(kthread_obj)) {
190                 kthread_obj = NULL;
191         chip_dev_exit();
192         dev_err(&client->dev, "Failed to create kernel thread\n");
193         return PTR_ERR(kthread_obj);
194     }
195
196         return 0;
197
198 }
```

- ret = chip_parse_dts(lt8911exb->dev): Parses the GPIO resources configured in the device tree.
- kthread_obj = kthread_run(LT8911EXB_Main, NULL, "LT8911EXB_kthread"): Create a thread, call LT8911EXB_Main function, LT8911EXB_Main function will initialize the chip.

三 Add To The Kernel Method

① Take the driver/video directory added to the kernel as an example. Create the LT8911EXB folder in the video directory:

```
ysy@ubuntu:~/workspace/imx6ull_pro/Imx6ull_source_code/Linux-4.9.88/drivers/video$ ls
backlight   console          display_timing.o  hdmi.c  Kconfig  LT6911UXC  LT8911EXB  LT9211D    LT9611UXD  of_display_timing.c  of_videomode.c  vgastate.c   videomode.o
built-in.o  display_timing.c  fbdev             hdmi.o  logo     LT6911UXE  LT9211C    LT9611UXC  Makefile   of_display_timing.o  of_videomode.o  videomode.c
ysy@ubuntu:~/workspace/imx6ull_pro/Imx6ull_source_code/Linux-4.9.88/drivers/video$
ysy@ubuntu:~/workspace/imx6ull_pro/Imx6ull_source_code/Linux-4.9.88/drivers/video$
```

Copy the provided driver source file into the LT8911EXB directory and configure the driver/video/Makefile and driver/video/Kconfig files that already exist in the video directory:

- driver/video/Makefile:

```
obj-y                           += LT9211C/
obj-y                           += LT9211D/
obj-y                           += LT6911UXC/
obj-y                           += LT6911UXE/
obj-y                           += LT9611UXD/
obj-y                           += LT8911EXB/

obj-$(CONFIG_VIDEOMODE_HELPERS) += display_timing.o videomode.o
```

- driver/video/Kconfig:

```
endmenu

source "drivers/video/LT9611UXC/Kconfig"

source "drivers/video/LT9211C/Kconfig"

source "drivers/video/LT9211D/Kconfig"

source "drivers/video/LT6911UXC/Kconfig"

source "drivers/video/LT6911UXE/Kconfig"

source "drivers/video/LT9611UXD/Kconfig"

source "drivers/video/LT8911EXB/Kconfig"
```

② Go to the LT8911EXB directory and add the driver/video/LT8911EXB/Makefile and driver/video/LT8911EXB/

Kconfig, the source code for these two files is provided, and the configuration content is as follows:

- driver/video/LT8911EXB/Makefile:

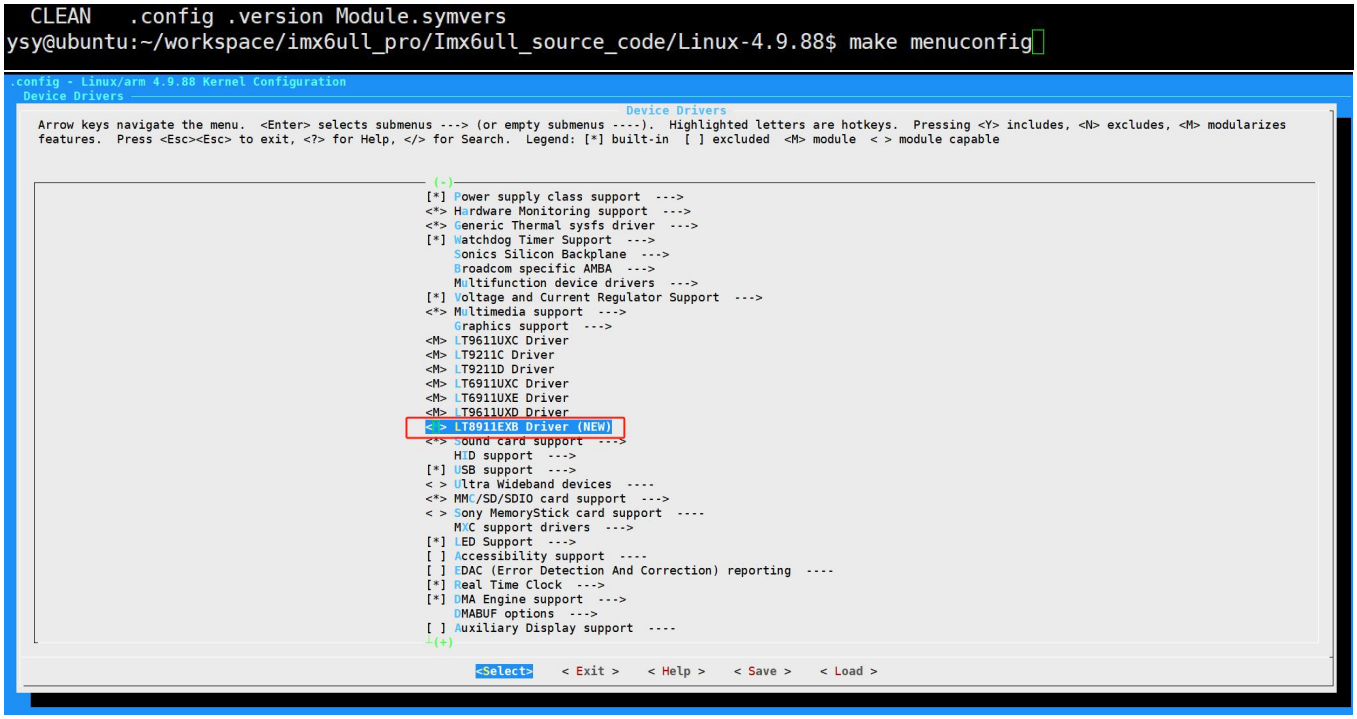
```
lt8911exb-y := LontiumDrv.o  LT8911EXB_Main.o  OcmI2cMaster.o lt8911.o

obj-m += lt8911exb.o
```

- driver/video/LT8911EXB/Kconfig:

```
config LT8911EXB
        tristate "LT8911EXB Driver"
        default m
        help
          Control LT8911EXB Chip
          Select m:build as a separate module;
          Select y:Build into the kernel
          Select n:Not build
```

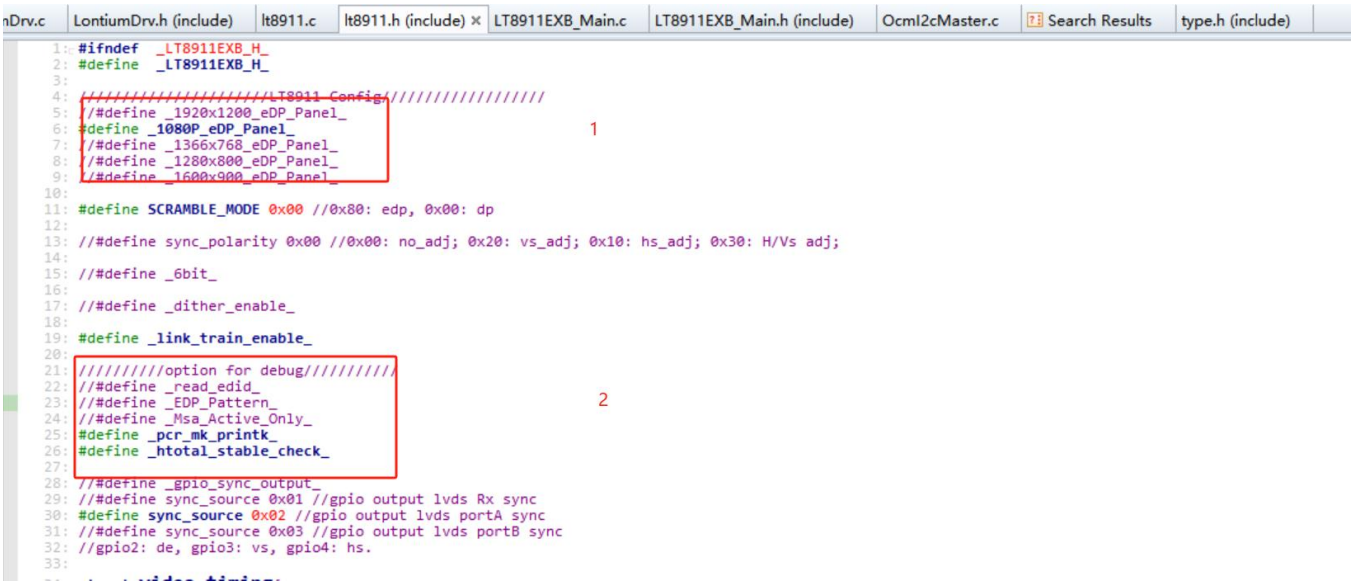
③ Execute the **make menuconfig** command in the kernel directory, and then configure the added driver to participate in compilation at compile time.

- At the red line below, select M to compile into a separate ko, which requires manual insmod.
- Select y to be automatically loaded when the kernel starts.
- Select n not to participate in compilation.
- Click Save and Exit to add the configuration to the configuration file .config, and then build.

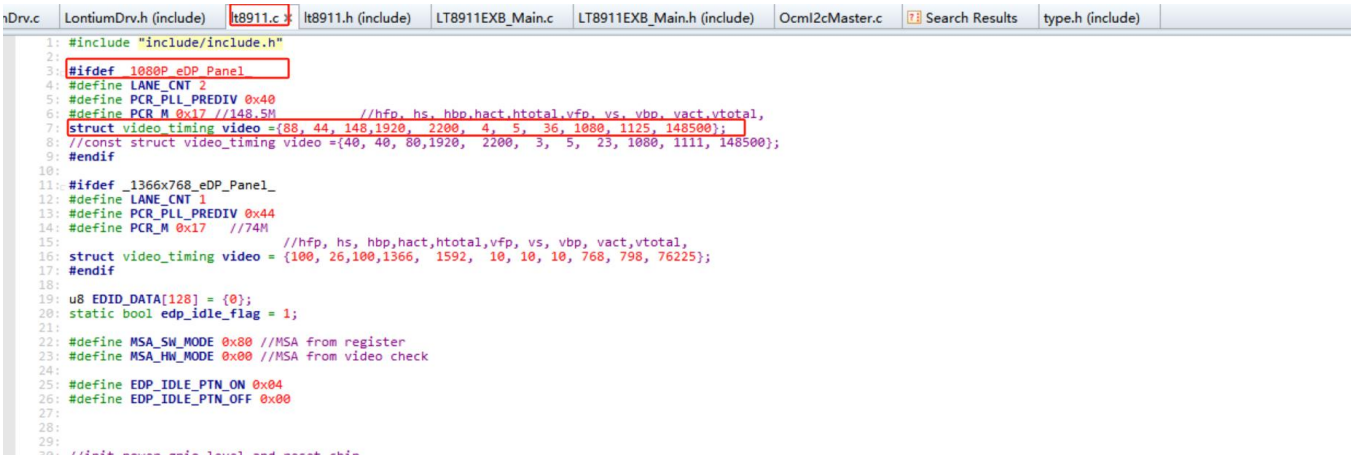


[NOTE]: Can also set CONFIG_LT8911EXB to a fixed value in xxxx_defconfig for your platform, so that you don't need make menuconfig.

四 LT8911EXB Software Configuration



1. Configure the output resolution as required,such as #define _1080P_eDP_Panel_ , output 1080p.
2. #define _EDP_Pattern_: output pattern.



1. _1080P_eDP_Panel_ is used here.
2. struct video_timing video = {}: Fill in specific timing information.