一、背景

最近调试硬件的imx290，因为一直没真正去了解过sensor的原理，只是拿别人弄好的驱动随便修改修改，点亮sensor，但是这次sensor确点不亮了，于是研究了相关的电路图和原理性的东西；知识都很基础，sensor也很简单~

二、硬件相关知识

1.基础的sensor290知识

请先阅读基本的《IMX290LQR-C_(E)Data_Sheet_E15510C59.pdf》

首先要确定sensor板基础连接方式：

1.三路电源是否正常？1.2V 1.8V 2.9V 确定这几个电源是否正常，如果不正常就没有下文了.

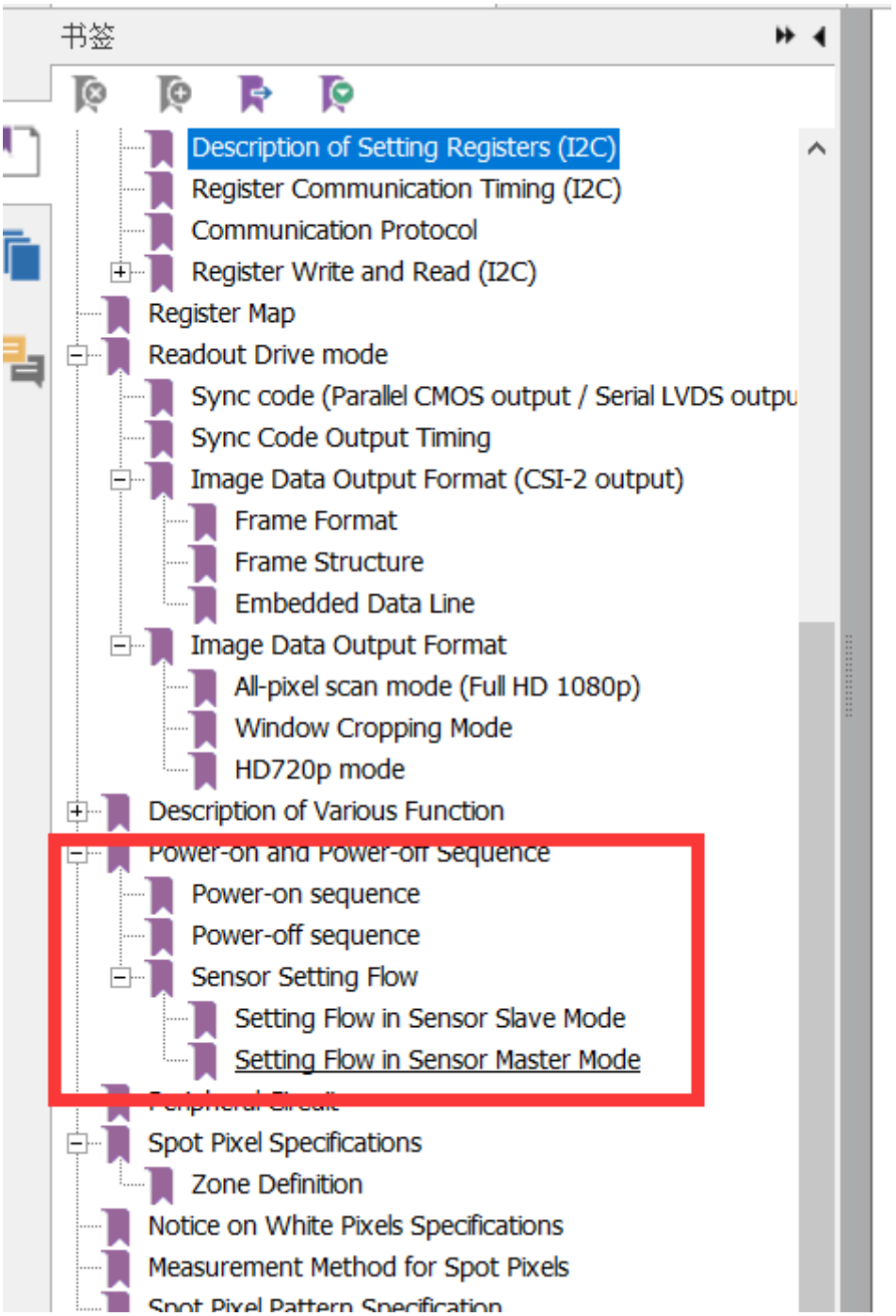
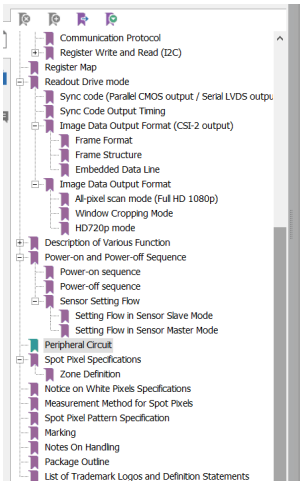
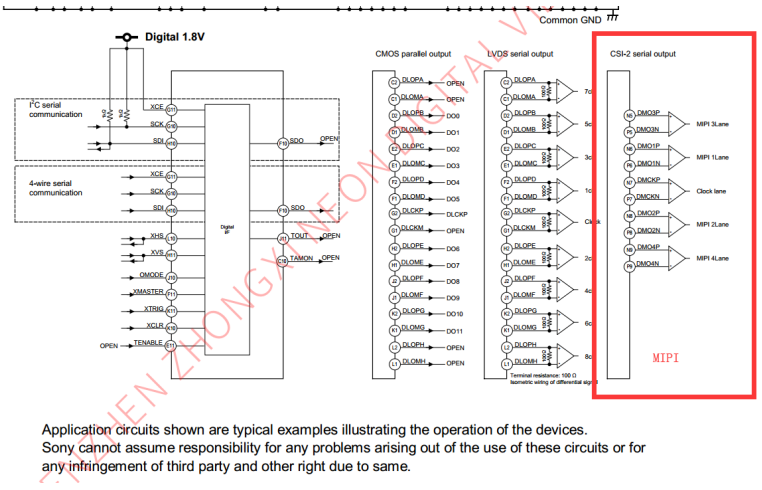


图1 电源相关描述

2.确定硬件连接相关，sensor CSI-1（串行mipi）和lvds的接线方式不一样，但是3519上的口一般都是复用的.规格书描述如下：

图2 PIN定义相关和电源相关描述





Application circuits shown are typical examples illustrating the operation of the devices.
Sony cannot assume responsibility for any problems arising out of the use of these circuits or for any infringement of third party and other right due to same.

图3 mipi lvds硬件接线图

由于对于硬件了解不到，所以拿着驱动各种试，发现i2c spi都能通信，但是一直收不到数据，最后发现特么用的配置驱动，电路根本都没接mipi的数据线~~电路图参考如下（图4 电路图连线图）同时还要注意OMODE pin的电平，这个是硬件配置的；

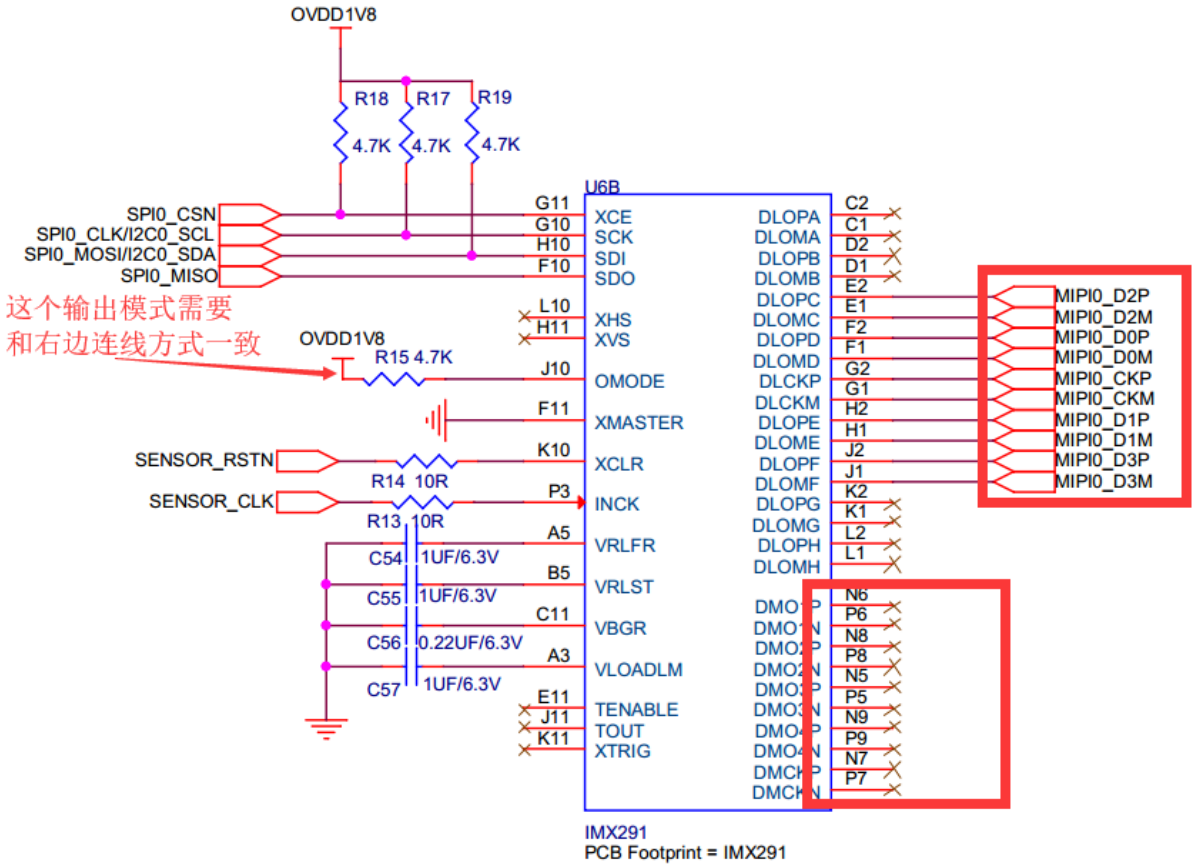


图4 电路图连线图

3. SPI还是I2C通信确认，这个需要确认主板和Sensor两边的走线；我遇到的情况是sensor拉了SPI的线路，但是主板SPI缺没连接上；同时还需要确认SPI是用几线通信方式（4线SPI还是3线SPI）

4.关于硬件的总结大概都这么多，发现以前遇到很多问题都没仔细研究个硬件相关的知识；了解软件和硬件相关知识后，可以真的软件和硬件相关做很多的优化比如下面：

A.排线问题，如果确定Sensor的通信方式，比如MIPI i2c，排线就可以减少很多.不但Sensor 还有主板的走线都会简化很多，而且可以对数据线用大量的地包围，效果会很好；
B.主板走线问题，少拉一组线，可能8层板就变为6层板子；（eg：做硬件过程中，发现同样的38*38 板子layout，别人6层可以搞定，我们却要8层，还感觉很密集,这部分其实需要软件和硬件共同来考虑和调试的,大部分硬件的人不怎么懂软件，软件的人不怎么懂硬件）

三、软件相关

主要调试海思平台的sensor，因为sensor是否点亮只会和三个方面有关，硬件(上面已经描述)、配置、数据接口

3.1 I2C 和SPI相关

I2C|SPI 只是作为sensor基本配置的通道，hisi大部分都会选择硬件的I2C|SPI 接口，因为驱动都是app层面的，基本只需要复用相关寄存器即可；

SPI通信都是ChipID+Addr+Data的方式，具体就参考图5 IMX290 SPI寄存器，

Setting Registers Using Serial Communication

This sensor can write and read the setting values of the various registers shown in the Register Map by 4-wire serial communication and I^2C communication. See the Register Map for the addresses and setting values to be set. Because the two communication systems are judged at the first communication, once they are judged, the communication cannot be switched until sensor reset. The pin for 4-wire serial communication and I^2C communication is shared, so the external pin XCE must be fixed to power supply side when using I^2C communication.

Description of Setting Registers (4-wire)

The serial data input order is LSB-first transfer. The table below shows the various data types and descriptions.

Serial Data Transfer Order

Chip ID	Start address	Data	Data	Data	...
(8 bit)	(8 bit)	(8 bit)	(8 bit)	(8 bit)	(8 bit)

Type and Description

Type	Description
Chip ID	02h: Write to the Chip ID = 02h register 03h: Write to the Chip ID = 03h register 04h: Write to the Chip ID = 04h register 05h: Write to the Chip ID = 05h register 06h: Write to the Chip ID = 06h register 82h: Read from the Chip ID = 02h register 83h: Read from the Chip ID = 03h register 84h: Read from the Chip ID = 04h register 85h: Read from the Chip ID = 05h register 86h: Read from the Chip ID = 06h register
Address	Designate the address according to the Register Map. When using a communication method that designates continuous addresses, the address is automatically incremented from the previously transmitted address.
Data	Input the setting values according to the Register Map.

图5 IMX290 SPI寄存器

I2C通信，都是3000的地址开始，基本和SPI一一对应关系：（基本I2C SPI 互换相关驱动直接修改这部分就完事了,因为HISI基本都有某些平台把驱动点亮了）

0x3000对应02H

0x3100对应03H

0x3200对应04H

0x3300对应05H

0x3400对应06H

(1) Registers corresponding t Chip ID = 02h in Write mode. (Read: Chip ID = 82h)

Address		bit	Register name	Description	Default value after reset		Reflection timing
1-wire	I^2C				By register	By address	
00h	3000h	0	STANDBY	Standby 0: Operating 1: Standby	1h	01h	Immediately
		1		Fixed to "0h"	0h		–
		2		Fixed to "0h"	0h		–
		3		Fixed to "0h"	0h		–
		4		Fixed to "0h"	0h		–
		5		Fixed to "0h"	0h		–
		6		Fixed to "0h"	0h		–
		7		Fixed to "0h"	0h		–
01h	3001h	0	REGHOLD	Register hold (Function not to update V reflection register) 0: Invalid 1: Valid	0h	00h	Immediately
		1		Fixed to "0h"	0h		–
		2		Fixed to "0h"	0h		–
		3		Fixed to "0h"	0h		–
		4		Fixed to "0h"	0h		–
		5		Fixed to "0h"	0h		–
		6		Fixed to "0h"	0h		–
		7		Fixed to "0h"	0h		–
02h	3002h	0	XMSTA	Setting of master mode operation 0: Master mode operation start 1: Master mode operation stop	1h	01h	Immediately
		1		Fixed to "0h"	0h		–
		2		Fixed to "0h"	0h		–
		3		Fixed to "0h"	0h		–
		4		Fixed to "0h"	0h		–
		5		Fixed to "0h"	0h		–

VANCHIP DIGITAL VIDEO CO., Ltd

图6 IMX290 SPI和I2C 参考

```
include <unistd.
include "hi_comm
include "hi_sns_
ifdef HI_GPIO_I2
include "gpioi
else
include "hi_i2
endif
imx290_i2c_addr
imx290_addr_byte
imx290_data_byte
g_fd
g_astimx290
g_aunImx290BusIn
IMX290_SENSOR_10
IMX290_SENSOR_10
IMX290_SENSOR_10
imx290_i2c_init
```

```
00653:  #endif
00654:      imx290_write_register (IspDev,0x3000, 0x01); /* standby */
00655:      imx290_write_register (IspDev,0x3002, 0x01); /* XTMSTA */
00656:
00657:      imx290_write_register (IspDev,0x3005, 0x00);
00658:      imx290_write_register (IspDev,0x3007, 0x00);
00659:      imx290_write_register (IspDev,0x3009, 0x01);
00660:      imx290_write_register (IspDev,0x300a, 0x3c);
00661:      imx290_write_register (IspDev,0x300c, 0x11);
00662:      imx290_write_register (IspDev,0x300f, 0x00);
00663:      imx290_write_register (IspDev,0x3010, 0x21);
00664:      imx290_write_register (IspDev,0x3012, 0x64);
00665:      imx290_write_register (IspDev,0x3016, 0x09);
00666:      imx290_write_register (IspDev,0x3018, 0xC4); /* VMAX */
00667:      imx290_write_register (IspDev,0x3019, 0x04); /* VMAX */
00668:      imx290_write_register (IspDev,0x301c, 0xEC); /* HMAX */
00669:      imx290_write_register (IspDev,0x301d, 0x07); /* HMAX */
00670:
```

图7 IMX290 I2C代码参考相关

3.2 sync code相关

以前每次适配相关的combo_dev_attr_t 里面属性，我都只能看别人写好的驱动试试，一些东西修改后也不知道什么问题.仔细研究学习后总结如下：

sync code 简单理解是frame的协议头而已，参考结构体，只要把Sensor的数据按顺序填入即可

```
/* each vc has 4 params, sync_code[i]:
   sync_mode is SYNC_MODE_SOL: SOF, EOF, SOL, EOL
```

sync_mode is SYNC_MODE_SAV: invalid sav, invalid eav, valid sav, valid eav */

```
typedef struct
{
    img_size_t          img_size;               /* oringnal sensor input image size */
    wdr_mode_e          wdr_mode;               /* WDR mode */
    lvds_sync_mode_e    sync_mode;              /* sync mode: SOL, SAV */
    raw_data_type_e     raw_data_type;          /* raw data type: 8/10/12/14 bit */
    lvds_bit_endian     data_endian;            /* data endian: little/big */
    lvds_bit_endian     sync_code_endian;       /* sync code endian: little/big */
    short               lane_id[LVDS_LANE_NUM]; /* lane id: -1 - disable */

    /* each vc has 4 params, sync_code[i]:
       sync_mode is SYNC_MODE_SOL: SOF, EOF, SOL, EOL
       sync_mode is SYNC_MODE_SAV: invalid sav, invalid eav, valid sav, valid eav */
    unsigned short      sync_code[LVDS_LANE_NUM][WDR_VC_NUM][SYNC_CODE_NUM];
} lvds_dev_attr_t;

typedef struct
{
    raw_data_type_e     raw_data_type;          /* raw data type: 8/10/12/14 bit */
    short               lane_id[MIPI_LANE_NUM]; /* lane_id: -1 - disable */
} mipi_dev_attr_t;

typedef struct
{
    input_mode_t        input_mode;             /* input mode: MIPI/LVDS/SUBLVDS/HISPI/DC */

    union
    {
        mipi_dev_attr_t     mipi_attr;          /* for MIPI configuration */
        lvds_dev_attr_t     lvds_attr;          /* for LVDS/SUBLVDS/HISPI configuration */
    };
} combo_dev_attr_t;
```

图8 sync code 结构体定义

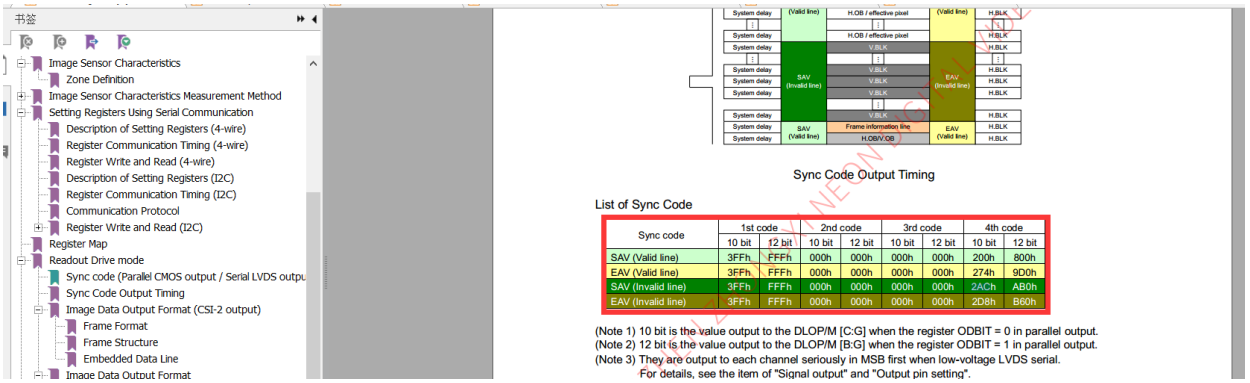


图9 IMX290 sync code 定义

```
combo_dev_attr_t LVDS_4lane_SENSOR_IMX290_10BIT_NORMAL_ATTR =
{
    /* input mode */
    .input_mode = INPUT_MODE_LVDS,
    {
        .lvds_attr = {
        .img_size = {1920, 1080},
        HI_WDR_MODE_NONE,
        LVDS_SYNC_MODE_SAV,
        RAW_DATA_10BIT,
        LVDS_ENDIAN_BIG,
        LVDS_ENDIAN_BIG,
        .lane_id = {0, 1, 2, 3, -1, -1, -1, -1},
        .sync_code = {
                {{0x2AC, 0x2D8, 0x200, 0x274},
                {0x2AC, 0x2D8, 0x200, 0x274},
                {0x2AC, 0x2D8, 0x200, 0x274},
                {0x2AC, 0x2D8, 0x200, 0x274}},

                {{0x2AC, 0x2D8, 0x200, 0x274},
                {0x2AC, 0x2D8, 0x200, 0x274},
                {0x2AC, 0x2D8, 0x200, 0x274},
                {0x2AC, 0x2D8, 0x200, 0x274}},

                {{0x2AC, 0x2D8, 0x200, 0x274},
                {0x2AC, 0x2D8, 0x200, 0x274},
                {0x2AC, 0x2D8, 0x200, 0x274},
                {0x2AC, 0x2D8, 0x200, 0x274}},

                {{0x2AC, 0x2D8, 0x200, 0x274},
                {0x2AC, 0x2D8, 0x200, 0x274},
                {0x2AC, 0x2D8, 0x200, 0x274},
                {0x2AC, 0x2D8, 0x200, 0x274}},
```

图10 IMX290 代码中的sync code 用法

代码里面sync code 和规格书里面的sync code 一一对应即可；非常简单，因为整个sensor驱动的框架都是固定的；

3.3 WDR 与非WDR 相关；

个人理解，imx290 这种wdr 都用的hisi ISP 内部调节，用2to1 3to1的方式，采用长短曝光整合成一帧图像，达到WDR的目的，大致流程时候，比如2to1 30frame/s 输出，就把Sensor 配置为60 frame/s 输出，isp 开启2to1模式，sensor 驱动曝光参数回调函数里调过注册的寄存器设置长曝光时间和短曝光时间；最后ISP合成为30frame/s 的图像给hisi mpp；（只是猜测，我也没仔细研究isp内部流程）具体这么实现的可以参考hisi的pdf文档，isp sensor适配相关~

3.4 IMX290 291 307等sensor区分

sensor 有很多同类型 costdown版本，基本都是pin2pin的，电路一样直接可以替换的，有时候难免没法直接区分相关sensor，但是可以通过寄存器来区分，具体区分办法为下面的：

■ Register data readout

For recognize IMX307 or IMX327 readout from register below.

Chip ID ：03h
Address ：DCh
(In 4-wire communication, Chip ID is 83h when reading the register value.)

Product	Data (Binary)
IMX307LQR	xxxx x10x
IMX327LQR/LQR1	xxxx x11x
IMX290LQR	xxxx x000
IMX291LQR	xxxx x001

**Do not read out the other registers that communication prohibited*

图11 IMX290 291 307等sensor寄存器差异