深圳市灵明光子科技有限公司
Shenzhen Adaps Photonics Technology Co.,Ltd

ADS6401 dToF **小面阵模组**

Linux 驱动开发指南

Ads6401 dToF Sensor

Linux Driver Guide

For **Flood module**

V3.0.0

Version	Date	Notes
V1.0	2024/09/20	Initial Release
V2.0.0	2025/01/17	Update to rk3568 SoC Linux platform
V3.0.0	2025/08/27	Update to rk3568 SoC Linux driver v3.2.13

目录

1. 简介

本文档提供灵明光子 ADS 6401 dToF sensor 以及相关模组参考驱动开发方案。ADS6401 参考设计基于 SoC Linux 平台 v4l2 驱动框架。参考 SoC 平台为 Rockchip RK3568。

驱动中增加了许多的设备属性，在打开 uart/adb 终端后，通过一些简单的命令即可查询到驱动的版本号、dump 寄存器，甚至执行一些单元测试。

驱动中增加 debugfs 的文件/sys/kernel/debug/adaps/dbg_ctrl，使用简单的 echo 命令可以开关基于功能模块的 log 输出，及打开 test pattern 输出等额外的功能支持。

请注意，Ads6401 芯片目前有两个大类的模组：**散点模组** 和 **小面阵模组**，两种模组共用一套驱动源码，以编译开关来选择。为了让用户更好的理解，本文档侧重介绍小面阵模组的驱动组成与实现，有关散点模组的驱动介绍，请参考《ADS6401_Linux_Guide_4_Spot_module.pdf》。

下面先简单介绍一下两种模组的组成。

- 散点模组使用OPN7020作为vcsel driver芯片，内置64K bytes的eeprom用于存储标定数据。
- 小面阵模组使用PhotonIC 5015作为vcsel driver芯片，内置32K bytes的eeprom用于存储标定数据，同时内置一个MCU用于控制vcsel driver芯片，温度采集和vop电压控制等。

2. Ads6401 小面阵模组构成

Ads6401 小面阵模组驱动包括以下几个 i2c 设备的驱动

设备功能	I2c 从地址	寄存器地址范围	备注
Ads6401 dToF sensor	0x5E/0x4A	0x00 – 0xFF	模组内置
标定数据 eeprom	0x50		模组内置
内置 MCU （控制 vcsel driver 等）	0x60	0x0000 – 0x7FFF VCSEL driver 寄存器 0x8000 – 0x807F MCU 功能控制	MCU 实现了一个 I2C 从设备，受外部 SoC 的控制
dvcc 电压控制 （TPS62864）	0x45	0x01 – 0x05	rk3568 转接板上（非模组内置）

注：Ads6401 dToF sensor 的确切 i2c 地址，跟硬件电路有关

除了 0x5E/0x4A 的 dToF sensor 主设备，其他几个 i2c 设备使用 devm_i2c_new_dummy_device()来注册，附在主设备上，这样易于管理和控制。

dToF 的工作需要有一个特殊的负压(Vop 电压，范围-20V~-30V)，而且 Vop 电压要根据芯片内部温度的变化而及时调整（在 ads6401.c 驱动中在开始出图后会启动一个 timer 来定时读取温度并调整 vop 电压），这些电压是一般的 SoC 芯片不提供的，所以需要新增电路来提供这些电压，并能通过软件进行调整控制。Vop 电压调节功能不属于 ads6401 模组内部（小面阵模组内部实现了根据温度自动控制 Vop 电压调整的功能，但是精度可能不及使用 SoC 来控制的好，客户可尝试打开该功能并验证是否可符合产品要求），是我司参考方案的外部转接板上的元件，在 rk3568 + 新的转接板上，Vop 由 rk3568 的 PWM 输出来控制转接

板电路来提供，客户可根据您的产品需要来参考实现或自己设计实现。

3. dts 设备树里的配置修改

增加一个 i2c 设备，并将它连接到 d-phy 设备节点上，同时修改 d-phy 的下一级输出节点到 raw data 输出，而不是 isp 模块，详见参考驱动源码包。

```
&i2c4 {
    status = "okay";
    clock-frequency = <1000000>;

    ads6401: ads6401@5e {
        reg = <0x5e>;
        status = "okay";
        compatible = "adaps,ads6401";
        pwms = <&pwm14 0 100000 0>,
               <&pwm2 0 100000 PWM_POLARITY_INVERTED>;
        pwm-names = "pwm_vbat_or_pvdd", "pwm_vop";
        clocks = <&cru CLK_CIF_OUT>;
        clock-names = "xvclk";
        power-domains = <&power RK3568_PD_VI>;
        pinctrl-names = "default";
        pinctrl-0 = <&cif_clk>;
        interrupt-parent = <&gpio4>;
        interrupts = <RK_PB5 IRQ_TYPE_EDGE_FALLING>,
                     <RK_PB4 IRQ_TYPE_EDGE_FALLING>;
        reset-gpios = <&gpio3 RK_PB6 GPIO_ACTIVE_LOW>;
        iovcc_en-gpios = <&gpio0 RK_PA0 GPIO_ACTIVE_HIGH>;
        dvcc_en-gpios = <&gpio3 RK_PB5 GPIO_ACTIVE_HIGH>;
        fsync_irq-gpios = <&gpio4 RK_PB5 GPIO_ACTIVE_HIGH>;
        drverr_irq-gpios = <&gpio4 RK_PB4 GPIO_ACTIVE_HIGH>;
        rockchip,camera-module-index = <0>;
        rockchip,camera-module-facing = "back";
        rockchip,camera-module-name = "ADS-6401-TOF";
        rockchip,camera-module-lens-name = "Unknown";
        port {
            ads6401_out: endpoint {
                remote-endpoint = <&mipi_in_ucam0>;
                data-lanes = <1 2>;
            };
        };
    };
```

PWM for vop/vbat

24M xclk输出给ads6401

frame_sync,
vcsel driver error
中断

D:\...kernel\rockchip_original\arch\arm64\boot\dts\rockchip\rk3568-evb1-ddr4-v10.dtsi

```
&csi2_dphy0 {
    status = "okay";

    ports {
        #address-cells = <1>;
        #size-cells = <0>;
        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;

            mipi_in_ucam0: endpoint@1 {
                reg = <1>;
                remote-endpoint = <&ucam_out0>;
                data-lanes = <1 2 3 4>;
            };
            mipi_in_ucam1: endpoint@2 {
                reg = <2>;
                remote-endpoint = <&gc8034_out>;
                data-lanes = <1 2 3 4>;
            };
            mipi_in_ucam2: endpoint@3 {
                reg = <3>;
                remote-endpoint = <&ov5695_out>;
                data-lanes = <1 2>;
            };
        };
        port@1 {
            reg = <1>;
            #address-cells = <1>;
            #size-cells = <0>;

            csidphy_out: endpoint@0 {
                reg = <0>;
                remote-endpoint = <&isp0_in>;
            };
        };
    };
};
```

* D:\Documents\Internal\swift\ads6401_Linux_Driver_document\ads6401_Linux_driver\kernel\adaps_modifi

```
&csi2_dphy1 {
    status = "okay";

    ports {
        #address-cells = <1>;
        #size-cells = <0>;
        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;

            mipi_in_ucam0: endpoint@1 {
                reg = <1>;
                remote-endpoint = <&ads6401_out>;
                data-lanes = <1 2>;
            };
            mipi_in_ucam1: endpoint@2 {
                reg = <2>;
                remote-endpoint = <&gc8034_out>;
                data-lanes = <1 2 3 4>;
            };
            mipi_in_ucam2: endpoint@3 {
                reg = <3>;
                remote-endpoint = <&ov5695_out>;
                data-lanes = <1 2>;
            };
        };
        port@1 {
            reg = <1>;
            #address-cells = <1>;
            #size-cells = <0>;

            csidphy_out: endpoint@0 {
                reg = <0>;
                remote-endpoint = <&mipi_csi2_input>;
            };
        };
    };
};
```

ads6401只用两个data lane, 因此rk3568 mipi rx的d-dphy使用split模式

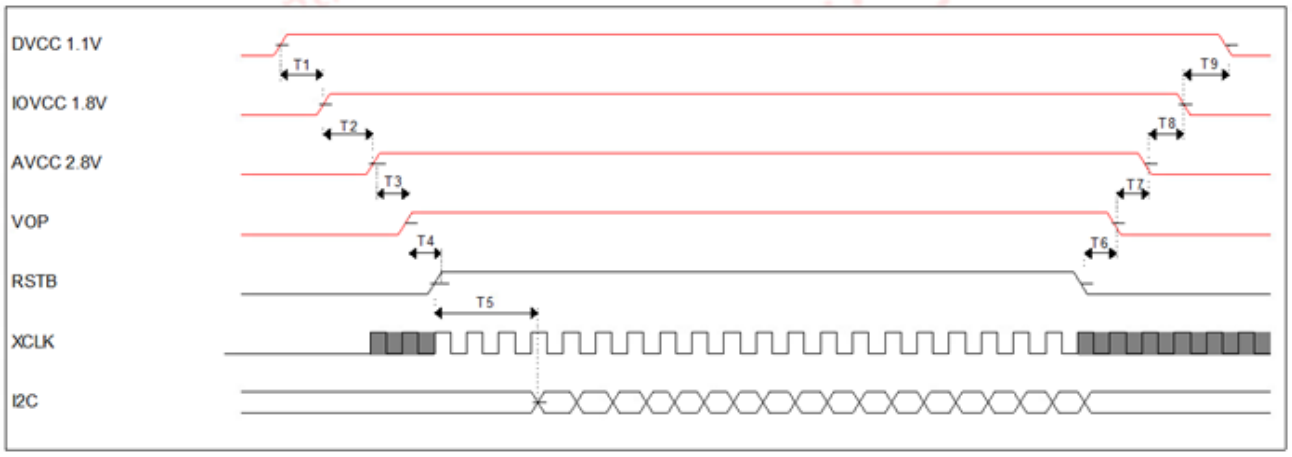
ads6401使用两个data lane

ads6401不需要rk3568的isp模块处理，修改成raw输出

```
/*add by Adaps to support vicap start*/
&mipi_csi2 {
    status = "okay";
    ports {
        #address-cells = <1>;
        #size-cells = <0>;
        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;

            mipi_csi2_input: endpoint@1 {
                reg = <1>;
                remote-endpoint = <&csidphy_out>;
                data-lanes = <1 2 3 4>;
            };
        };
        port@1 {
            reg = <1>;
            #address-cells = <1>;
            #size-cells = <0>;

            mipi_csi2_output: endpoint@0 {
                reg = <0>;
                remote-endpoint = <&cif_mipi_in>;
                data-lanes = <1 2 3 4>;
            };
        };
    };
};

&rkcif_mipi_lvds {
    status = "okay";

    port {
        cif_mipi_in: endpoint {
            remote-endpoint = <&mipi_csi2_output>;
            data-lanes = <1 2 3 4>;
        };
```

4. Ads6401 芯片上下电时序

Ads6401 芯片上电时序遵循先低压后高压的原则，下电时则正好相反，先高压后低压的原则。Linux v4l2 框架下 sensor 的上电和下电通常在 struct v4l2_subdev_core_ops 结构体的 s_power()回调函数里来实现，对于我们的参考驱动 ads6401.c，对应 sensor_s_power(), sensor_set_power_on(), sensor_set_power_off()三个函数。具体的上下电控制，根据 SoC 板的硬件设计来定，比如使用 Linux regulator 来供电，或者通过 gpio 口来控制开关硬件电路开关某路电压，或者通过硬件设计的 RC 延时上电来实现都可以。



- T1, T2, T3, T4 >= 100 us
- T5 >= 500 us
- T6, T7, T8, T9 > 0 us
- XCLK 为 24M（默认）或 25M

建议在 DVCC 和 IOVCC 上电后再开启 XCLK 24M 时钟输入给 dToF sensor

5. ADS6401 小面阵模组寄存器配置

在 Linux/Android 下，camera 类设备的配置，通常分成几段来配置：

- 全局初始化配置（共同部分的配置）
- work mode 相关部分配置(对应 rgb camera 的各种 resolution 配置)

- stream_on 部分配置
- stream_off 部分配置

1. 全局初始化配置

   参考 ads6401.c 中 mcuctrl_init_regs，sensor_init_regs 及 vcseldriver_init_regs

2. 写 ROI SRAM 寄存器(0xF7, 0xFB)

   具体数值从模组的 eeprom 里(swift_flood_module_eeprom_data_t 数据结构的 sramData[]部分)读出来，每个寄存器最大可写 2048 bytes. 一般情况下，0xF7 和 0xFB 写入的内容是相同的，但都需要写。有的平台可以一次性往一个寄存器地址写 2K bytes 数据，比如 Rockchip 和 Hisilicon，但是有的平台上(比如高通)，不能一次写 2K 这么大，就要分多次写，总的要求是要确保正确写入了，也就是写入 ROI SRAM 寄存器(0xf7, 0xfb)的数据和读出来的进行对比要完全一致。

3. 根据需要，配置合适的工作模式设置

   PCM 灰度模式(2560x32)配置，参考 ads6401.c 的 sensor_pcm_regs

   FHR 模式(4104x32)配置，参考 ads6401.c 的 sensor_fhr_regs

4. 配置开始出图（streamOn）寄存器

   参考 ads6401.c 的 sensor_streamon_regs

5. 当需要停止出图时，配置停止出图（streamOff）寄存器

   参考 ads6401.c 的 sensor_streamoff_regs

a) dToF sensor 全局初始化配置

```
#define REG_NULL                          0xFF
#define REG16_NULL                        0xFFFF

struct setting_rvd {
    UINT8  reg;
    UINT8  val;
    UINT32  delayUs;
};

static const struct setting_rvd sensor_init_regs[] = {
    {0XA8, 0X30, 0},
    {0XD4, 0X32, 0},
    {0XD5, 0XFA, 0},
#if (MIPI_SPEED == MIPISPEED_500M_BPS)
       {0XE6, 0X00, 0},
       {0XE7, 0X7D, 0},
       {0XE9, 0X62, 0},
       {0X7E, 0X0F, 0},
       {0X81, 0X80, 0},
       {0XA5, 0X11, 0},
       {0X88, 0X05, 0},
       {0X8E, 0X05, 0},
#elif (MIPI_SPEED == MIPISPEED_720M_BPS)
       {0XE6, 0X00, 0},
       {0XE7, 0X78, 0},
       {0XE9, 0X42, 0},
       {0X7E, 0X0C, 0},
       {0X81, 0X00, 0},
       {0XA5, 0X11, 0},
       {0X88, 0X07, 0},
       {0X8E, 0X07, 0},
#elif (MIPI_SPEED == MIPISPEED_1G_BPS)
       {0XE6, 0X00, 0},
       {0XE7, 0X54, 0},
       {0XE9, 0X43, 0},
       {0X7E, 0X0B, 0},
       {0X81, 0X80, 0},
       {0XA5, 0X13, 0},
       {0X88, 0X09, 0},
       {0X8E, 0X09, 0},
#elif (MIPI_SPEED == MIPISPEED_1G2_BPS)
       {0XE6, 0X00, 0},
```

```
        {0XE7, 0X64, 0},
        {0XE9, 0X41, 0},
        {0X7E, 0X00, 0},
        {0X81, 0X20, 0},
        {0XA5, 0X11, 0},
        {0X88, 0X0B, 0},
        {0X8E, 0X0B, 0},
#elif (MIPI_SPEED == MIPISPEED_1G5_BPS)
        {0XE6, 0X00, 0},
        {0XE7, 0X7D, 0},
        {0XE9, 0X41, 0},
        {0X7E, 0X00, 0},
        {0X81, 0X20, 0},
        {0XA5, 0X11, 0},
        {0X88, 0X0D, 0},
        {0X8E, 0X0D, 0},
#endif
    {0XB0, 0X21, 0},
    {0XB1, 0X21, 0},
    {0XAC, 0X01, 0},
    {0XD3, 0X41, 0},
    {0XC6, 0X3F, 0},
    {0XAE, 0X40, 0},
    {0XA6, 0X0B, 0},  // 30 fps for 250M sys clock
    {0XD9, 0X1A, 0},
    {0XDA, 0X60, 0},
#if defined(NON_CONTINUOUS_MIPI_CLK)
        {0X04, 0x04, 0},
        {0X05, 0X18, 0},     //TCLW
        {0X06, 0X18, 0},     //TCLT
        {0X07, 0X78, 0},     //TDLW
#else
        {0X04, 0x0C, 0},
#endif

#if (MIPI_SPEED == MIPISPEED_500M_BPS)
        {0XA9, 0X20/SYS_CLK_FACTOR, 0},
#elif (MIPI_SPEED == MIPISPEED_720M_BPS)
        {0XA9, 0X15/SYS_CLK_FACTOR, 0},
#elif (MIPI_SPEED == MIPISPEED_1G_BPS)
        {0XA9, 0X10/SYS_CLK_FACTOR, 0},
#elif (MIPI_SPEED == MIPISPEED_1G2_BPS)
        {0XA9, 0X0, 0},
#elif (MIPI_SPEED == MIPISPEED_1G5_BPS)
```

```
        {0XA9, 0X0, 0},
#endif
    {0X08, 0XFA, 0},
    {0X0C, 0X05, 0},
    {0XBA, 0X18, 0},
    {0XAA, 0X01, 0},
    {0XDF, 0X00, 0},

    {REG_NULL, 0x00, 0},
};
```

b) dToF sensor FHR 深度模式配置

```
static struct setting_rvd sensor_fhr_regs[] = {
    {0XA4, 0X00, 0},   //bit0: 1 as master, 0 as slave
    {0XB6, 0X26, 0},
    {0XBC, (BIT(7) | VCSEL_LASER_PERIOD), 0},// VCSEL_LASER_PERIOD=96/127
    {0XCB, 0X04, 0},
    {0X18, 0X08, 0},
    {0X19, 0X10, 0},
    {0XAF, 0X00, 0},
    {0XBF, 0X01, 0},
    {0XC0, FINE_EXPOSURE_TIME, 0},  // fine exposure time, default 0x3C
    {0XC1, 0X01, 0},
    {0XBD, 0X66, 0},
    {0XBE, 0X66, 0},
    {0XC3, 0X06, 0},
    {0XC4, 0X08, 0},
    {0XC5, 0X06, 0},
    {0XB0, 0X21, 0},
    {0XB1, 0X83, 0},
    {0XB5, 0X0F, 0},
    {0XBB, 0X06, 0},
    {0XB7, 0XF9, 0},
    {0XB8, 0X43, 0},
    {0XB9, 0X08, 0},
    {0XAA, 0X01, 0},
    {0XD9, 0X1F, 0},
    {0XC5, 0X10, 0},
    {0XB2, 0X01, 0},
    {0XB3, 0X32, 0},
    {0XB4, 0X16, 0},
    {REG_NULL, 0x00, 0}
};
```

c) dToF sensor PCM 灰度模式配置

灰度模式下，ADS6401(Rx 芯片)只以 master 模式工作

```
static struct setting_rvd sensor_pcm_regs[] = {
    {0XA4, 0X01, 0},
    {0XB6, 0X01, 0},
    {0XBC, 0XE0, 0},
    {0XCB, 0X04, 0},
    {0X18, 0X00, 0},
    {0X19, 0X0A, 0},
    {0XC3, 0X0B, 0},
    {0XC4, 0X08, 0},
    {0XC2, 0X20, 0}, // PCM expore time change to 2560us.
    {0XAF, 0X20, 0},
    {0XC5, 0X10, 0},
    {0XB7, 0Xf9, 0},
    {0XB2, 0X01, 0},
    {0XB3, 0X32, 0},
    {0XB4, 0X16, 0},

    {REG_NULL, 0x00, 0}
};
```

d) mcuctrl & VCSEL driver 初始化配置

```
struct setting_r16vd {
    UINT16  reg;
    UINT8   val;
    UINT32  delayUs;
};

static const struct setting_r16vd mcuctrl_init_regs[] = {
    {0x8031, 0x01, 0},
#if defined(VOP_ADJUST_BY_BUILT_IN_MCU)
    {0x8032, 0x01, 0},
#else
    {0x8032, 0x00, 0},
#endif
    {0x8033, 0x00, 0},
    {0x8034, 0x00, 0},
    {0x8035, 0x00, 0},
```

```
    {0x8023, 0x01, 0},
#if defined(VOP_ADJUST_BY_BUILT_IN_MCU)
    {0x8024, 0x01, 0},
#else
    {0x8024, 0x00, 0},
#endif
    {0x8000, 0x12, 0},
    {0x8001, 0xc0, 0},
    {0x8002, 0x07, 0},
    {0x8003, 0x80, 0},
    {REG16_NULL, 0x00, 0},
};

static const struct setting_r16vd vcseldriver_init_regs[] = { // for PHX3D 5015
    //Reset chip
    {0x0081, 0x06, 0},
    {0x0081, 0x04, 0},
    {0x0081, 0x00, 0},
    {0x0081, 0x01, 0},
    {0x0081, 0x03, 0},
    {0x0081, 0x07, 500},

#ifdef VCSEL_ERR_DETECT_ENABLE
    //Mask all the errors
    {0x00A2, 0xc7, 0},
    {0x00A3, 0xbF, 0},
    {0x00A4, 0x10, 0},
#else
    {0x00A2, 0xff, 0},
    {0x00A3, 0xfF, 0},
    {0x00A4, 0x19, 0},
#endif
    //Emission settings
    {0x0084, 0x00, 0},
    {0x0082, 0x24, 0},
    //PD_RES=100kΩ
    {0x0020, 0x0B, 0},
    {0x0088, 0x02, 0},
    {0x0089, 0x03, 0},

    //FWHM Code
    //You can change 0x83~0x84 to change the FWHM value
    {0x0083, 0xC8, 0},
```

```
    {0x0084, 0x00, 0},
    {0x001D, 0xE3, 0},
    {0x008E, 0x65, 0},
    {0x0093, 0xFF, 0},
    {0x0094, 0x01, 0},
    {0x00F2, 0xFF, 0},
    {0x00F3, 0x01, 0},
    {0x0096, 0x50, 0},
    {0x0097, 0xE0, 0},
    {0x0098, 0x01, 0},
    {0x00A0, 0x0A, 0},
    {0x00E3, 0x66, 0},
    {0x00E3, 0x64, 0},
    {0x00E3, 0x24, 0},
    {0x00E5, 0x30, 0},
#ifdef VCSEL_ERR_DETECT_ENABLE
    {0x00EE, 0xcb, 0}, // change vcsel temperature limit to 65 degree
    {0x00EF, 0x04, 0}, // change vcsel temperature limit to 65 degree
#else
    {0x00EE, 0xFF, 0},
    {0x00EF, 0x05, 0},
#endif
    {0x00F0, 0xFF, 0},
    {0x00F9, 0x96, 0},
    {0x00FA, 0x07, 0},
    {0x00FB, 0x96, 0},
    {0x00FD, 0x02, 0},
    {0x00E6, 0x80, 0},
    {0x0087, 0x33, 0},
    {0x0085, 0xFF, 0},
    {0x0087, 0x33, 0},
    {0x0086, 0xFF, 0},

    //Boost settings
    //You can change 0x8A~0x8D to change the LDVCC voltage
    {0x008A, 0x84, 0},
    {0x008B, 0x03, 0},
    {0x008C, 0x84, 0},
    {0x008D, 0x03, 0},
    {0x0043, 0x0C, 0},
    {0x0062, 0x08, 0},
    {0x0082, 0x20, 0},
    {0x0044, 0x09, 0},
    {0x0044, 0x03, 0},
```

```
    {0x0082, 0xA4, 10},

    {0x0080, 0x01, 0},
    {0x0042, 0xFF, 0},
    {0x003E, 0x05, 0},
    {REG16_NULL, 0x00, 0},
};
```

e) dToF sensor 起流(stream_On)配置

```
static const struct setting_rvd sensor_streamon_regs[] = {
    {0xAB, 0x01, 0},
    {REG_NULL, 0x00, 0},
};

static const struct setting_rvd stream_on_with_mipi_reset_first[] = {
    {0xAE, 0x10, 0},
    {0xAB, 0x01, 0},
    {REG_NULL, 0x00, 0},
};

static const struct setting_rvd stream_on_first_and_then_reset_mipi[] = {
    {0xAB, 0x01, 0},
    {0xAE, 0x10, 0},
    {REG_NULL, 0x00, 0},
};
```

f) dToF sensor 停流(stream_Off)配置

```
static const struct setting_rvd sensor_streamoff_regs[] = {
    {0xAB, 0x00, 0},
    {REG_NULL, 0x00, 0},
};
```

g) dToF sensor ROI SRAM 寄存器配置

Ads6401 芯片包括 4 个 Calib ROI SRAM 地址，一共 2K bytes 的实际空间，SRAM 位宽均为 16bit，用于存放 8bit Row 地址和 8bit Column 地址。

寄存器地址	实际有效物理地址	备注
0xFE	0x300~0x3FF	16bit位宽，实际存放240个SPOT address
0xFD	0x200~0x2FF	16bit位宽，实际存放240个SPOT address
0xFC	0x100~0x1FF	16bit位宽，实际存放240个SPOT address
0xFB	0x000~0x0FF	16bit位宽，实际存放240个SPOT address

如上图所示，Calib SRAM zone 的实际有效数据是 480 bytes，但是为了操作方便起见，在 EEPROM 存储 Calib SRAM 数据时，通常每个 SRAM zone 会占用 512 bytes(末尾的 32 bytes 填 0x00 或 0xFF)，这样 4 个寄存器的地址空间都连成一片了，如果平台支持，最快的写 Calib SRAM 的方式是一次性往 0xFB 寄存器写入 512x4=2048 bytes。

但是，不同的 SoC 平台，使用 i2c/cci/spi 一次性往同一个寄存器地址写入的长度可能是不同的。我们曾经见过的特殊情况，是高通 SM8250/SM8450 上使用 cci (应该是一种 cost down 的 i2c 接口，内部的 dma buffer 可能比较小)接口一次性最多只能写入 450 bytes，都不够一个 Calib SRAM zone 的 512 bytes，无奈之下，只有每次写 256 bytes，这时就需要同时操控 CSRU_SRAM_OFFSET_REG(0xC7)寄存器来配合使用。

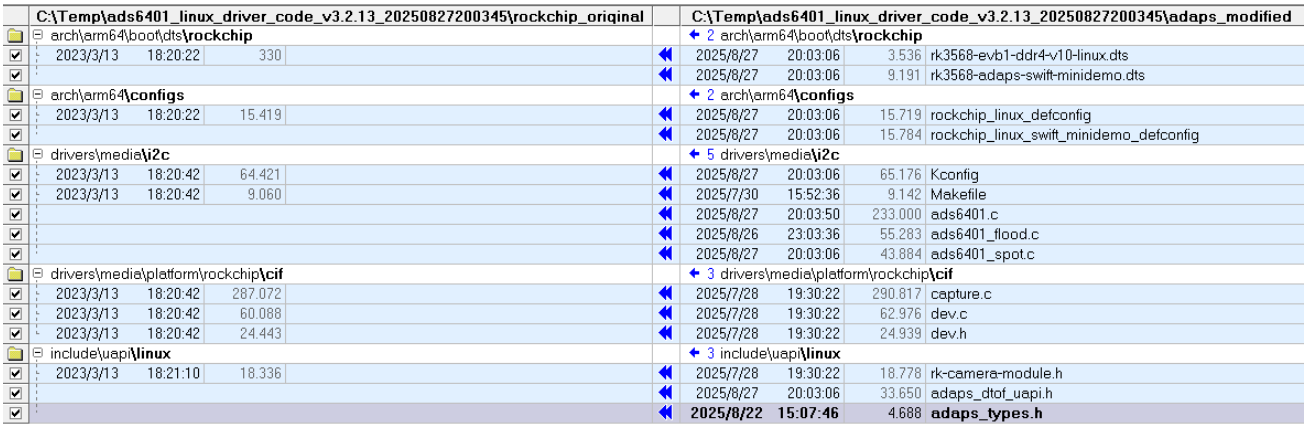
而如果平台比较友好，比如我们适配过的 rockchip 和海思的某些 SoC，就可以一次性写 2K bytes，这样就简单好多。

在 porting 到一个新的 SoC 平台时，我们可能不清楚该平台的 i2c 一次性能写多长，这种情况下，可以先尝试一次性全写入，同时打开 ADAPS_DBG_ROISRAM_RB_4_VERIFY 运行时 debug 开关，通过在写完后读回来进行对比（一定要使用 memcmp 函数进行比较，如果只是 dump 出来肉眼检

查，不一定可靠），看看是否完全一致来做出选择。

6. ADS6401 Linux 驱动源码介绍

ADS6401 参考设计基于 Rockchip rk3568 SoC Linux（kernel 版本 5.10）平台和 v4l2 camera 驱动框架。



C:\Temp\ads6401_linux_driver_code_v3.2.13_20250827200345\rockchip_original				C:\Temp\ads6401_linux_driver_code_v3.2.13_20250827200345\adaps_modified			
arch\arm64\boot\dts**rockchip**				← 2 arch\arm64\boot\dts**rockchip**			
2023/3/13	18:20:22	330		2025/8/27	20:03:06	3.536	rk3568-evb1-ddr4-v10-linux.dts
				2025/8/27	20:03:06	9.191	rk3568-adaps-swift-minidemo.dts
arch\arm64**configs**				← 2 arch\arm64**configs**			
2023/3/13	18:20:22	15.419		2025/8/27	20:03:06	15.719	rockchip_linux_defconfig
				2025/8/27	20:03:06	15.784	rockchip_linux_swift_minidemo_defconfig
drivers\media**i2c**				← 5 drivers\media**i2c**			
2023/3/13	18:20:42	64.421		2025/8/27	20:03:06	65.176	Kconfig
2023/3/13	18:20:42	9.060		2025/7/30	15:52:36	9.142	Makefile
				2025/8/27	20:03:50	233.000	ads6401.c
				2025/8/26	23:03:36	55.283	ads6401_flood.c
				2025/8/27	20:03:06	43.884	ads6401_spot.c
drivers\media\platform\rockchip**cif**				← 3 drivers\media\platform\rockchip**cif**			
2023/3/13	18:20:42	287.072		2025/7/28	19:30:22	290.817	capture.c
2023/3/13	18:20:42	60.088		2025/7/28	19:30:22	62.976	dev.c
2023/3/13	18:20:42	24.443		2025/7/28	19:30:22	24.939	dev.h
include\uapi**linux**				← 3 include\uapi**linux**			
2023/3/13	18:21:10	18.336		2025/7/28	19:30:22	18.778	rk-camera-module.h
				2025/8/27	20:03:06	33.650	adaps_dtof_uapi.h
				2025/8/22	**15:07:46**	4.688	**adaps_types.h**

如上截图是 Linux kernel 改到的对比，修改了一些文件，新增了 3 个.c 文件。在 drivers/media/i2c 目录下，新增了两个头文件 adaps_types.h 和 adaps_dtof_uapi.h 位于 kernel/include/uapi/linux 目录下。它们提供一些我司 dToF sensor 特需的数据类型、结构体、ioctl 命令的定义等。

在 ads6401.c 文件里，根据 SWIFT_MODULE_TYPE 的定义分两种情况来配置一些编译功能开关，实现对驱动功能的配置，如下截图所示。

```
#define SWIFT_MODULE_TYPE ..... ADS6401_MODULE_SPOT  //ADS6401_MODULE_FLOOD

#if defined(CONFIG_SWIFT_MINI_DEMO_BOX)
    #define MINI_DEMO_BOX
#endif

#if (ADS6401_MODULE_FLOOD != SWIFT_MODULE_TYPE)
    #define ADS6401_MODULE_TYPE_NAME ..... "Spot"

    #define VCSEL_LASER_PERIOD ..... 96  //127 ..... //<=127, laser repetition frequency period, unit is ns
    #define ENABLE_BIG_FOV_MODULE

    //for rk3568 + new adapter board: 2 PWM output from rk3568 for pvdd and vop voltage adjustment.
    #define ENABLE_SOC_PWM_4_PVDD_VOLTAGE
    #define ENABLE_SOC_PWM_4_VOP_VOLTAGE
    #define ENABLE_VCSEL_DRV_ERROR_IRQ

    #define TTY_DRIVERNAME_4_UART ..... "ttyFIQ"
    #define CALIB_DATA_READY_IN_EEPROM_CHIP
    #define OPN7020_VCSEL_DRIVER_ENABLE
    #define MIPI_SPEED ..... MIPISPEED_1G_BPS
    #define NON_CONTINUOUS_MIPI_CLK

    #define MIPI_RESET_DEFAULT_CFG ..... STREAM_ON_WITH_MIPI_IP_RESET_FIRST
    #define ROISRAM_ANCHOR_PREPROCCESS_ENABLE
    #define ENABLE_RUNTIME_REGISTER_UPDATE
    //#define ENABLE_SYSFREQ_ADJUST_4_LOW_POWER ..... //only apply to 1G mipi speed now.
    //#define FRAMERATE_TEST_4_60FPS

    #define SENSOR_XCLK_FROM_SOC
    #define DEVICE_CONCURRENCY_OPEN_CHECK
    #define ENABLE_SENSOR_FSYNC_IRQ
    #define ENABLE_SENSOR_ROI_SWITCH_BY_FSYNC_IRQ

    #define IGNORE_PROBE_FAILURE

#else
    //#error "ADS6401_MODULE_FLOOD is selected!"
    #define ADS6401_MODULE_TYPE_NAME ..... "Flood"

    //#define VOP_ADJUST_BY_MCU
    #define ENABLE_SOC_PWM_4_VOP_VOLTAGE
    #define TTY_DRIVERNAME_4_UART ..... "ttyFIQ"
    #define CALIB_DATA_READY_IN_EEPROM_CHIP
    #define ENABLE_VCSEL_4_PCM_MODE_CFG ..... true  // enable vcsel for pcm mode of flood module
    #define VCSEL_LASER_PERIOD ..... 96 ..... //<=127, laser repetition frequency period, unit is ns

    #define MIPI_SPEED ..... MIPISPEED_1G_BPS
    //#define USE_DIV1_FOR_1G_MIPISPEED
    #define NON_CONTINUOUS_MIPI_CLK

    #define MIPI_RESET_DEFAULT_CFG ..... STREAM_ON_WITH_MIPI_IP_RESET_FIRST
```

同时，根据 SWIFT_MODULE_TYPE 的定义，ads6401.c 中将#include ads6401_spot.c 或 ads6401_flood.c 文件，这两个文件实现了两种模组有差异的部分功能。

另外，实现了两个方便调试的功能：

第一， 在 debugfs 文件系统下，新增了 /sys/kernel/debug/adaps/dbg_ctrl 文件，它用来控制一些 ads6401 驱动的行为，用户可在终端通过 cat /sys/kernel/debug/adaps_dbg_ctrl 查看当前值，也可以使用 echo 0xXXXXXXXX > /sys/kernel/debug/adaps/dbg_ctrl 来改变驱动的一些行为，比如当该变量修改为 0x2 时，在试图起流(stream_on)时，会在 kernel log 里 dump 各个功能部件的寄存器值（可使用 dmesg 来查看），而当

改变量修改为 0x8 是，在出图时，ads6401 dToF sensor 将输出 test pattern 数据，以便软件工程师 porting 到一个新的平台时，尽快确认 ads6401 是否已基本正常工作，以及 mipi 通路是否已打通和正确？当然还有一些其他的功能，这里就不一一列举了，用户可参考源码包中 ads6401.c 的如下枚举定义：

```
enum adaps_dbg_type_t {
    ADAPS_DBG_TRACE_REGS_READ          = BIT(0),
    ADAPS_DBG_DUMP_REGS                = BIT(1),
    ADAPS_DBG_TRACE_REGS_WRITE         = BIT(2),
    ADAPS_DBG_TESTPATTERN_ENABLE       = BIT(3),

    ADAPS_DBG_POWER_CTRL               = BIT(4),
    ADAPS_DBG_DEBUG_INFO_ENABLE        = BIT(5),
    ADAPS_DBG_DUMP_SRAM_REG            = BIT(6),
    ADAPS_DBG_PM_RUNTIME               = BIT(7),

    ADAPS_DBG_VOLTAGE_UPDATE           = BIT(8),
    ADAPS_DBG_DISABLE_VCSEL_DRVIER     = BIT(9),
    ADAPS_DBG_ROISRAM_RB_4_VERIFY      = BIT(10),
    ADAPS_DBG_IOCTL_CMD                = BIT(11),

    ADAPS_DBG_V4L2_CALLBACK            = BIT(12),
    ADAPS_DBG_MUTEX_LOCK               = BIT(13),
    ADAPS_DBG_DISABLE_ROI_SWITCH       = BIT(14),
    ADAPS_DBG_DISABLE_ERR_IRQ_HANDLE   = BIT(15),
```

第二， 为 ads6401 驱动添加如下设备属性，这些也将对驱动程序的调试有非常大的帮助，比如通过 cat /path/to/register 可以实时 dump 模组的各个功能模块的寄存器，而 echo write XX YY > /path/to/register 可以将 ads6401 的地址为 0xXX 的寄存器修改为 0xYY。而 cat /path/to/info 可以查看驱动的版本、编译日期时间等不会变的信息，cat /path/to/status 可以查看当前一些状态，是否上电状态，是否正在出图，出图了多少时间，当前的芯片内部温度是多少，期望的 vop 电压是多少等等。vop_pwm_test、vbat_pwm_test 和 pvdd_pwm_test 可用来使用命令将该功能的 PWM 进行手动配置，比如将 vop PWM 输出配置为千分之 500，或者配置目标输出电压为-25.60 V，然后我们就可以用万用表看看实际输出的 Vop 电压是否符合设计预期，从而判断软件算法是否正确，硬件电路的误差有多少等等。

```
static struct attribute *my_drv_attrs[] = {
    &dev_attr_mipi_performance_dbg.attr,
    &dev_attr_register.attr,
    &dev_attr_info.attr,
    &dev_attr_status.attr,
    &dev_attr_config.attr,
    &dev_attr_i2c_address.attr,
    &dev_attr_frequency.attr,
    &dev_attr_force_sensor_role.attr,
    &dev_attr_vop_adjust_interval.attr,
    &dev_attr_reset_mipi_4_streamon.attr,
    &dev_attr_manual_reset_mipi.attr,
    &dev_attr_manual_powerctl.attr,
    &dev_attr_reset_gpio.attr,
#if defined(ENABLE_SOC_PWM_4_VOP_VOLTAGE)
    &dev_attr_vop_pwm_test.attr,
#endif

#if (ADS6401_MODDULE_FLOOD == SWIFT_MODULE_TYPE)
    &dev_attr_vbat_pwm_test.attr,
    &dev_attr_force_enable_vcsel_4_pcm_mode.attr,
#else
    &dev_attr_pvdd_pwm_test.attr,
#endif
    NULL,
};

static const struct attribute_group my_drv_attrs_group = {
    .attrs = my_drv_attrs,
};
```

```
root@rk356x:~# cd /userdata/
root@rk356x:/userdata# find /sys -name register
/sys/devices/platform/fe5d0000.i2c/i2c-4/4-005e/register
root@rk356x:/userdata#
root@rk356x:/userdata# ln -s /sys/devices/platform/fe5d0000.i2c/i2c-4/4-005e ads6401
```

```
root@rk356x:/userdata/ads6401# cat status
probe success:                  Yes
dbg_ctrl:                       0x2

Sensor Chip id:                 0x6401
Sensor Role:                    Master
raw level of reset_gpio:        High
raw level of iovcc_en_gpio:     High
raw level of dvcc_en_gpio:      High
raw level of drverr_irq_gpio:   High
vcsel_drv_err_irq:              123 (0)
Power_on:                       Yes (3)
Streaming:                      Yes (0:01:46)
Vcsel driver initilized:        Yes
current temperature:            60.25 degree
current expected Vop:           -25.76 V
current pvdd:                   7.60 V
Curr workmode:                  PTM-FHR
Curr measurement type:          Full-distance
Curr enviroment type:           Indoor
Curr power mode:                Div3
mipi data lanes:                2
Reset MIPI mode for stream_on: No reset
```

```
root@rk356x:/userdata/ads6401# cat info
Adaps ads6401 dToF sensor driver
Version:                        2.1.0_LM20250114A
Build Time:                     Jan 14 2025,14:38:57
I2C Bus Num:                    4
I2C bus frequency:              1000000Hz
I2C address for ads6401:        0x5e
Current TTY:                    pts1
VBD in efuse:                   2362
anchor_x:                       1
anchor_y:                       0
tdc_delay_major:                0x9
tdc_delay_minor:                0xb
```

```
root@rk356x:/userdata/ads6401# cat vop_pwm_test
period_ns: 100000 ns
duty_ns: 85300ns
status: Enabled
```

7. 附录

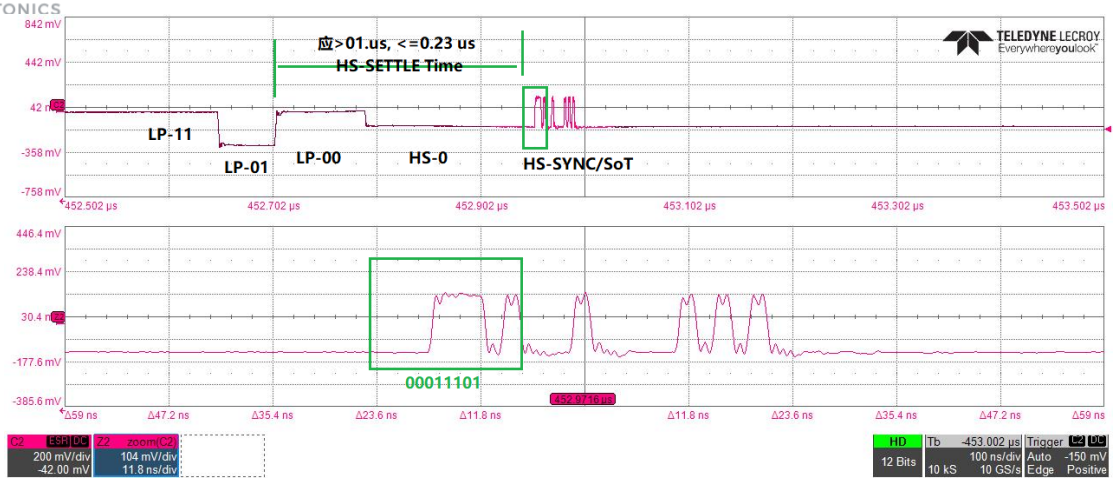
a) 起流前需要复位 MIPI 模块问题说明

ADS6401 dToF sensor 芯片已适配高通 db410、sm8250、sm8450、sm8550 SoC 平台，rockchip rk3588 平台以及海思的某个 SoC 平台。有的平台必须要在 stream_on 之前复位一下 dToF 芯片 MIPI 模块，有的平台却不需要。

```
enum {
    STREAM_ON_WITHOUT_MIPI_IP_RESET         = 0, // 不需要复位芯片内 mipi 模块
    STREAM_ON_WITH_MIPI_IP_RESET_FIRST       = 1, // 先复位芯片内 mipi 模块，再起流
    STREAM_ON_FIRST_AND_THEN_RESET_MIPI_IP  = 2  // 先起流，再复位芯片内 mipi 模块
};
```

原因：

当 ADS6401 MIPI 使用连续时钟时，某些 SoC 的 MIPI RX IP 可能需要监测到 MIPI clock 信号有从 LP 到 HS 转换的过程（如下图所示上半部），才会产生中断，而另外一些 SoC 的 MIPI RX IP 不监测 MIPI clock 信号，而只监测 data lane 就能产生中断。因此，如果在适配到一个新的平台时，可以先尝试配置为不需要复位芯片内 mipi 模块（`STREAM_ON_WITHOUT_MIPI_IP_RESET`），这样是最好的。



b) Test Patten 使能及其输出 RAW data 格式说明

在 porting 到一个新的 SoC 平台初期，我们可以通过使能 swift 芯片的 test patten（打开运行时动态控制开关的 ADAPS_DBG_TESTPATTERN_ENABLE 位，使寄存器 0xDF 设置为 0x08），通过检查接收到的 raw data 是否正确来确认 mipi 通路是否配置正确了？

在 test pattern 使能后：

1.PCM 模式的 raw data 为全 0x00；

2，PHR 模式的起初 2 个包是不同的，后面的 30 个包的有效数据是不变的（在寄存器配置固定的情况下）

请留意，camera 的 mipi 输出 buffer 大小有 stride 的概念，即每行占用的字节数，有的平台可能每行的大小要求 16 的整数倍。在高通 SM8450 平台上，PHR/PCM/FHR 的帧宽都是跟 swift 的本身设计一致的，即分别是 1032/2560/4104，但是在 rk3568/rk3588 上，PHR 的帧宽都是跟 swift 的本身设计不一致，抓到的数据大小为 1280x32，根据我曾经转换成 C 语言数组形式与高通 android12 上抓到的数据对比，发现应该是每行前面 1032 byte 为有效数据，

1033-1280 byte 为冗余数据，前面的有效数据是一致的。PCM 模式的帧宽 2560 能被 32 整除，抓出来的数据也正好是 2560x32。

以下 PCM 模式 raw data 文件的 md5 值及 raw data.zip 文件供参考。

```
David.chen@SZ-PC002 MINGW64 /c/temp/rk3588/rk3588_pcmphr_testpattern_0803a/PCM
$ md5sum *
030a4f48dc8db0956add25994004e5ca *frame000_20230803_094233.raw
030a4f48dc8db0956add25994004e5ca *frame001_20230803_094233.raw
030a4f48dc8db0956add25994004e5ca *frame002_20230803_094233.raw
030a4f48dc8db0956add25994004e5ca *frame003_20230803_094234.raw
030a4f48dc8db0956add25994004e5ca *frame004_20230803_094234.raw
030a4f48dc8db0956add25994004e5ca *frame005_20230803_094235.raw
030a4f48dc8db0956add25994004e5ca *frame006_20230803_094235.raw
030a4f48dc8db0956add25994004e5ca *frame007_20230803_094236.raw
```

rk3588_pcmphr_testpattern_0803a.zip

c) Ads6401 作为 slave，对于 vsync 信号的规格说明

vsync 的的极性由寄存器 0xA4 配置，有效宽度在 0xA7 寄存器配置，而频率则等于帧率.

0xA4 寄存器的 bit3 控制 vsync 信号的极性，该位为 0 时低电平有效，为 1 时高电平有效。注意从低往高数，为 bit0--bit7.

0xA7 寄存器用于 VSYNC 信号宽度配置，以 1us（1MHz）为单位，有效范围从 0x01 – 0xFF，分别代表 vsync 信号的宽度为 1us – 255us.

Swift 工作在 Slave 模式时，帧率由外部 master 控制，spadis 的曝光时间+masking+MIPI Tx 时间不得大于外部 master 的帧率对应的时间。

d) SoC 端 RX 模块 mipi 配置建议

在 Linux 系统中，V4L2_CID_LINK_FREQ 和 V4L2_CID_PIXEL_RATE 是与视频捕获和处理相关的两个控制 ID，它们都属于 Video4Linux（V4L2）框架。

V4L2_CID_LINK_FREQ 用于表示摄像头捕获设备的接口或链路的频率。这通常指摄像头的图像传感器与图像处理器之间的数据传输频率。这个参数通常以赫兹（Hz）为单位，表示每秒钟传输数据的次数。

V4L2_CID_PIXEL_RATE 用于指定图像传感器每秒输出的像素数量，通常以像素/秒（pixels/second）为单位。

Ads6401 的 mipi 频率通常配置为 1GHz，所以链接频率 SENSOR_LINK_FREQ 设置为 500MHz，而像素率 SENSOR_PIXEL_RATE 根据以下公式计算：

SENSOR_PIXEL_RATE = ((SENSOR_LINK_FREQ / SENSOR_BITS_PER_SAMPLE) * 2 * SENSOR_LANES)

其中 SENSOR_BITS_PER_SAMPLE 为 8，而 SENSOR_LANES 为 2，因此计算出来的像素率为 250M 像素/秒.

e）关于 chipid（0xA2, 0xA3）寄存器值的说明

早期的 swift 芯片，chipid 值的 otp 写为 0x41, 0x01，后来更改为 0x64, 0x01（更符合商业型号 ads6401），具体以芯片读出值为准。