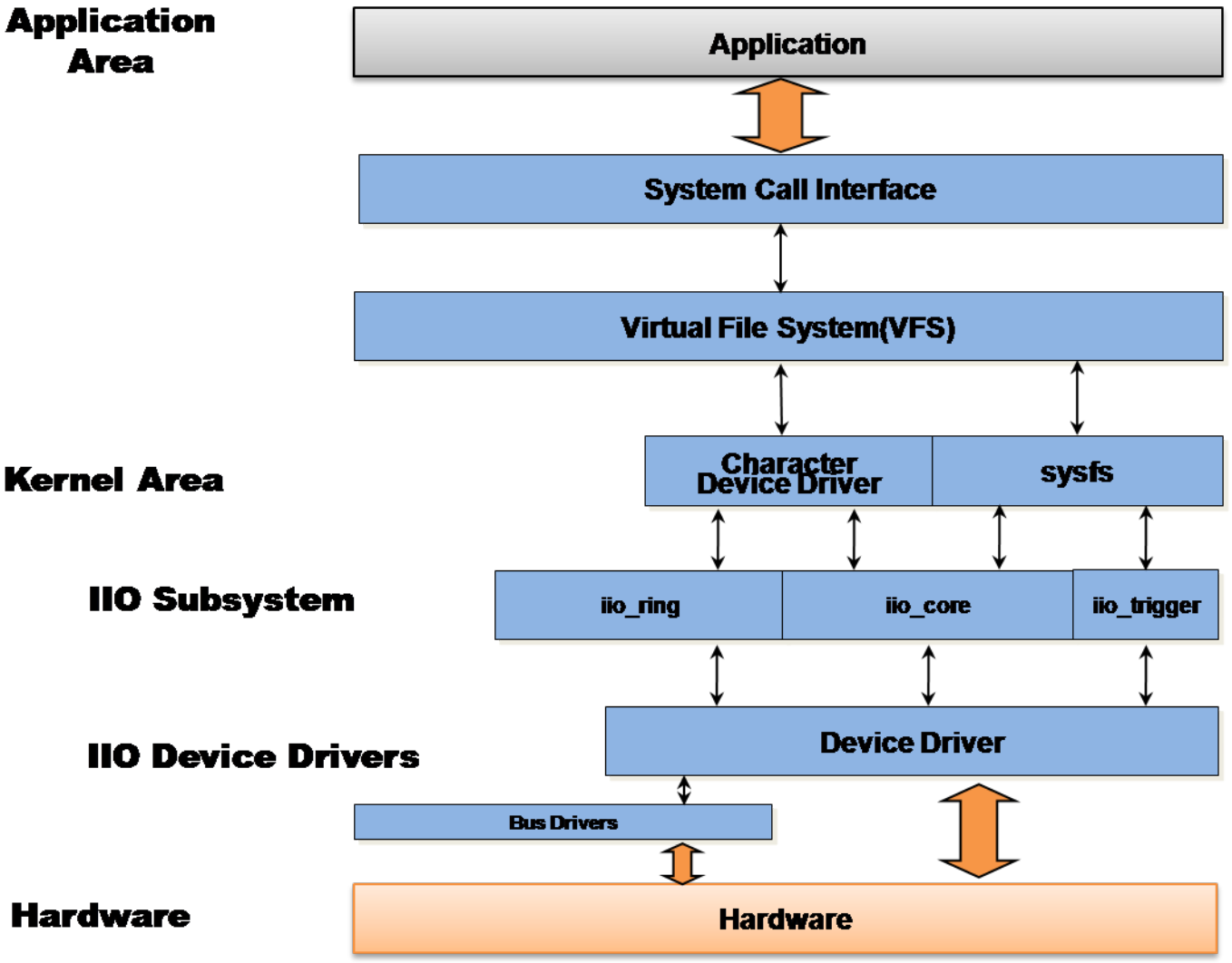
IIO是Industrial I/O subsystem（工业IO子系统）。可以用于很多的输入，大多数是工业用途，和Input方式有所区别。
下面是IIO支持的设备，其中imu/inv_mpu 文件夹下，就是invensense的传感器驱动。

```
iio
├── accel
├── adc
├── addac
├── cdc
├── dac
├── dds
├── Documentation
│   └── dac
├── gyro
├── impedance-analyzer
├── imu
│   └── inv_mpu
│       └── inv_test
├── light
├── magnetometer
├── meter
├── pressure
├── resolver
└── trigger
```



Rk3288没打补丁之前的版本，使用了Input输入输出，还没有研究，现在根据Invensense和H8使用的IIO输入输出方式总结一下rk3288补丁的驱动逻辑。

在Linux 3.0以上版本的内核中，使用了DTS（设备树）的模式对设备进行管理。
在驱动代码 `inv_mpu/inv_mpu_core.c` 中：
使用了 of_device_id 来与DTS的结点匹配

```
static const struct of_device_id inv_mpu_of_match[] = {
    { .compatible = "invensense,itg3500", },
    { .compatible = "invensense,mpu3050", },
    { .compatible = "invensense,mpu6050", },
    { .compatible = "invensense,mpu9150", },
    { .compatible = "invensense,mpu6500", },
    { .compatible = "invensense,mpu9250", },
    { .compatible = "invensense,mpu6xxx", },
    { .compatible = "invensense,mpu9350", },
    { .compatible = "invensense,mpu6515", },
    {}
};
```

并且使用了 `i2c_driver` 来和驱动的处理函数进行绑定：

```
static struct i2c_driver inv_mpu_driver = {
```

```
    .class = I2C_CLASS_HWMON,
    .probe      =   inv_mpu_probe,
    .remove     =   inv_mpu_remove,
    .shutdown   =   inv_mpu_shutdown,
    .id_table   =   inv_mpu_id,
    .driver = {
        .owner  =   THIS_MODULE,
        .name   =   "inv-mpu-iio",
        .pm     =       INV_MPU_PMOPS,
        .of_match_table = of_match_ptr(inv_mpu_of_match),
    },
    .address_list = normal_i2c,
};
```

然后再添加驱动：

```
static int __init inv_mpu_init(void)
{
    int result = i2c_add_driver(&inv_mpu_driver);
    if (result) {
        pr_err("failed\n");
        return result;
    }
    return 0;
}
```

驱动的添加不是重点，略过，直接看 inv_mpu_probe 这个函数，这个函数相当于一次初始化，获取设备结点，初始化缓存ring等等。
其中有函数 of_inv_parse_platform_data 函数可以读取设备结点：

```
static int of_inv_parse_platform_data(struct i2c_client *client,
                      struct mpu_platform_data *pdata)
{
    int ret;
    int length = 0,size = 0;
    struct property *prop;
    u32 orientation[9];
    int orig_x,orig_y,orig_z;
    int i;
    struct device_node *np = client->dev.of_node;
    unsigned long irq_flags;
    int irq_pin;
    int gpio_pin;
    int debug;

    gpio_pin = of_get_named_gpio_flags(np, "irq-gpio", 0,(enum of_gpio_flags
*)&irq_flags);
    gpio_request(gpio_pin, "mpu6500");
    irq_pin = gpio_to_irq(gpio_pin);
    client->irq = irq_pin;
    i2c_set_clientdata(client, &mpu_data);
```

```
        ret = of_property_read_u32(np,"mpu-int_config",&mpu_data.int_config);
        if(ret!=0){
                dev_err(&client->dev, "get mpu-int_config error\n");
                return -EIO;
        }

        ret = of_property_read_u32(np,"mpu-
level_shifter",&mpu_data.level_shifter);
        if(ret!=0){
                dev_err(&client->dev, "get mpu-level_shifter error\n");
                return -EIO;
        }

        prop = of_find_property(np, "mpu-orientation", &length);
        if (!prop){
                dev_err(&client->dev, "get mpu-orientation length error\n");
                return -EINVAL;
        }

        size = length / sizeof(u32);

        if((size > 0)&&(size <10)){
                ret = of_property_read_u32_array(np, "mpu-orientation",
                                        orientation,
                                         size);

                if(ret<0){
                        dev_err(&client->dev, "get mpu-orientation data
error\n");
                        return -EINVAL;
                }
        }
        else{
                printk(" use default orientation\n");
        }

        for(i=0;i<9;i++)
                mpu_data.orientation[i]= orientation[i];

        ret = of_property_read_u32(np,"orientation-x",&orig_x);
        if(ret!=0){
                dev_err(&client->dev, "get orientation-x error\n");
                return -EIO;
        }

        if(orig_x>0){
                for(i=0;i<3;i++)
                        if(mpu_data.orientation[i])
                                mpu_data.orientation[i]=-1;
        }

        ret = of_property_read_u32(np,"orientation-y",&orig_y);
        if(ret!=0){
                dev_err(&client->dev, "get orientation-y error\n");
```

```
                return -EIO;
        }

        if(orig_y>0){
                for(i=3;i<6;i++)
                        if(mpu_data.orientation[i])
                                mpu_data.orientation[i]=-1;
        }


        ret = of_property_read_u32(np,"orientation-z",&orig_z);
        if(ret!=0){
                dev_err(&client->dev, "get orientation-z error\n");
                return -EIO;
        }

        if(orig_z>0){
                for(i=6;i<9;i++)
                        if(mpu_data.orientation[i])
                                mpu_data.orientation[i]=-1;
        }

        ret = of_property_read_u32(np,"mpu-debug",&debug);
        if(ret!=0){
                dev_err(&client->dev, "get mpu-debug error\n");
                return -EINVAL;
        }

        if(debug){

printk("int_config=%d,level_shifter=%d,client.addr=%x,client.irq=%x\n",mpu_dat
a.int_config, \
                        mpu_data.level_shifter,client->addr,client->irq);

                for(i=0;i<size;i++)
                        printk("%d ",mpu_data.orientation[i]);
                printk("\n");

        }

    return 0;
}
```

DTS设备文件中的结点定义如下，可以对应着上面的函数来看一下,，其中的参数是一一对应的，以键值对的方式：

```
mpu6500@68 {
                compatible = "invensense,mpu6500";
                reg = <0x68>;
                mpu-int_config = <0x10>;
                mpu-level_shifter = <0>;
                mpu-orientation = <0 1 0 1 0 0 0 0 1>;
                orientation-x= <1>;
```

```
                orientation-y= <1>;
                orientation-z= <1>;
                irq-gpio = <&gpio8 GPIO_A2 IRQ_TYPE_LEVEL_LOW>;
                mpu-debug = <1>;
                status = "okay";
        };
sensor@0d {
          compatible = "ak8963";
          reg = <0x0d>;
          type = <SENSOR_TYPE_COMPASS>;
          irq-gpio = <&gpio8 GPIO_A1 IRQ_TYPE_EDGE_RISING>;
          irq_enable = <1>;
          poll_delay_ms = <30>;
          layout = <5>;
          };
```

IIO子系统也同样是用虚拟文件的方式上报数据到user space的，这个文件的内容就是从ring（环形缓冲区）所上报的。
于是，我们看一下ring的初始化函数 `inv_mpu_configure_ring` ，想来这个函数一定注册了某种机制进行数据的上报：

```
int inv_mpu_configure_ring(struct iio_dev *indio_dev)
{
    int ret;
    struct inv_mpu_iio_s *st = iio_priv(indio_dev);
    struct iio_buffer *ring;

    ring = iio_kfifo_allocate(indio_dev);
    if (!ring)
        return -ENOMEM;
    indio_dev->buffer = ring;
    /* setup ring buffer */
    ring->scan_timestamp = true;
    indio_dev->setup_ops = &inv_mpu_ring_setup_ops;
    /*scan count double count timestamp. should subtract 1. but
    number of channels still includes timestamp*/
    if (INV_MPU3050 == st->chip_type)
        ret = request_threaded_irq(st->client->irq, inv_irq_handler,
            inv_read_fifo_mpu3050,
            IRQF_TRIGGER_RISING | IRQF_SHARED, "inv_irq", st);
    else
        ret = request_threaded_irq(st->client->irq, inv_irq_handler,
            inv_read_fifo,
            IRQF_TRIGGER_RISING | IRQF_SHARED, "inv_irq", st);
    if (ret)
        goto error_iio_sw_rb_free;

    indio_dev->modes |= INDIO_BUFFER_TRIGGERED;
    return 0;
error_iio_sw_rb_free:
    iio_kfifo_free(indio_dev->buffer);
    return ret;
}
```

看这句语句：

```
ret = request_threaded_irq(st->client->irq, inv_irq_handler,
            inv_read_fifo,
            IRQF_TRIGGER_RISING | IRQF_SHARED, "inv_irq", st);
```

request_threaded_irq 是Linux 2.6之后提供的一个申请中断的函数，其中第一个参数是终端号，第二个参数是硬中断处理函数（可以通过返回 IRQ_WAKE_THREADED唤醒中断线程，也可返回IRQ_HANDLE不执行中断线程），第三个是中断线程。

直接来看中断线程 inv_read_fifo ，从名字上可以看出应该是从FIFO缓存读取数据：

```
/**
 *  inv_read_fifo() - Transfer data from FIFO to ring buffer.
 */
irqreturn_t inv_read_fifo(int irq, void *dev_id)
{

    struct inv_mpu_iio_s *st = (struct inv_mpu_iio_s *)dev_id;
    struct iio_dev *indio_dev = iio_priv_to_dev(st);
    size_t bytes_per_datum;
    int result;
    u8 data[BYTES_FOR_DMP + QUATERNION_BYTES];
    u16 fifo_count;
    u32 copied;
    s64 timestamp;
    struct inv_reg_map_s *reg;
    s64 buf[8];
    s8 *tmp;

    int64_t tm_i2c_sum = 0;
    int64_t tm = 0;
    int64_t tm_begin, tm_end;

//  unsigned long flags;
    tm_begin = get_time_ns();

//  local_irq_save(flags);
//  preempt_disable();

    mutex_lock(&indio_dev->mlock);

    if (!(iio_buffer_enabled(indio_dev)))
        goto end_session;

    reg = &st->reg;
    if (!(st->chip_config.accl_fifo_enable |
        st->chip_config.gyro_fifo_enable |
        st->chip_config.dmp_on |
        st->chip_config.compass_fifo_enable |
```

```
    st->mot_int.mot_on))
    goto end_session;
if (st->mot_int.mot_on)
    inv_process_motion(st);
if (st->chip_config.dmp_on && st->chip_config.smd_enable) {
    /* dmp interrupt status */
    result = inv_i2c_read(st, REG_DMP_INT_STATUS, 1, data);
    if (!result)
        if (data[0] & SMD_INT_ON) {
            sysfs_notify(&indio_dev->dev.kobj, NULL,
                    "event_smd");
            st->chip_config.smd_enable = 0;
        }
}
if (st->chip_config.lpa_mode) {
    result = inv_i2c_read(st, reg->raw_accl,
                  BYTES_PER_SENSOR, data);
    if (result)
        goto end_session;
    inv_report_gyro_accl_compass(indio_dev, data,
                     get_time_ns());
    goto end_session;
}
bytes_per_datum = get_bytes_per_datum(st);
fifo_count = 0;
if (bytes_per_datum != 0) {
    tm = get_time_ns();
    result = inv_i2c_read(st, reg->fifo_count_h,
            FIFO_COUNT_BYTE, data);
    tm_i2c_sum += get_time_ns()-tm;
    if (result)
        goto end_session;
    fifo_count = be16_to_cpup((__be16 *)(&data[0]));
    if (fifo_count == 0)
        goto flush_fifo;
    if (fifo_count < bytes_per_datum)
        goto end_session;
    /* fifo count can't be odd number */
    if (fifo_count & 1)
        goto flush_fifo;
    if (fifo_count >  FIFO_THRESHOLD)
        goto flush_fifo;
    /* timestamp mismatch. */
    if (kfifo_len(&st->timestamps) <
        fifo_count / bytes_per_datum)
        goto flush_fifo;
    if (kfifo_len(&st->timestamps) >
        fifo_count / bytes_per_datum + TIME_STAMP_TOR) {
        if (st->chip_config.dmp_on) {
            result = kfifo_to_user(&st->timestamps,
            &timestamp, sizeof(timestamp), &copied);
            if (result)
                goto flush_fifo;
        } else {
            goto flush_fifo;
```

```
                }
            }
        } else {
            result = kfifo_to_user(&st->timestamps,
                &timestamp, sizeof(timestamp), &copied);
            if (result)
                goto flush_fifo;
        }
        tmp = (s8 *)buf;
        while ((bytes_per_datum != 0) && (fifo_count >= bytes_per_datum)) {
            tm = get_time_ns();
            result = inv_i2c_read(st, reg->fifo_r_w, bytes_per_datum,
                data);
            tm_i2c_sum += get_time_ns()-tm;
            if (result)
                goto flush_fifo;

            result = kfifo_to_user(&st->timestamps,
                &timestamp, sizeof(timestamp), &copied);
            if (result)
                goto flush_fifo;
            inv_report_gyro_accl_compass(indio_dev, data, timestamp);
            fifo_count -= bytes_per_datum;
        }
        if (bytes_per_datum == 0 && st->chip_config.compass_fifo_enable)
            inv_report_gyro_accl_compass(indio_dev, data, timestamp);

    end_session:
        mutex_unlock(&indio_dev->mlock);
    //  local_irq_restore(flags);
    //  preempt_enable();

        tm_end = get_time_ns();
    //  if (tm_end-tm_begin > 700000)
    //      pr_info("%s: [%lld] %lld %lld %lld\n", __func__, timestamp, tm_begin-
    timestamp, tm_i2c_sum, tm_end-tm_begin);

        return IRQ_HANDLED;

    flush_fifo:
        /* Flush HW and SW FIFOs. */
        inv_reset_fifo(indio_dev);
        inv_clear_kfifo(st);
        mutex_unlock(&indio_dev->mlock);

        return IRQ_HANDLED;
    }
```

注释说明了，这是从FIFO缓存读数据到iio的ring缓冲区

```
    result = inv_i2c_read(st, reg->fifo_r_w, bytes_per_datum,data);
```

这句话就是关键语句，reg->fifo_r_w就是FIFO寄存器的地址（这个寄存器用处从FIFO缓冲区读取输出，FIFO的大小有512字节，按照各种标志位存有原始数据），bytes_per_datum是大小，data就是读出来的数据。

之后用了 `inv_report_gyro_accl_compass(indio_dev, data, timestamp);` 对数据进行上报。

```
static int inv_report_gyro_accl_compass(struct iio_dev *indio_dev,
                    u8 *data, s64 t)
{
    struct inv_mpu_iio_s *st = iio_priv(indio_dev);
    short g[THREE_AXIS], a[THREE_AXIS], c[THREE_AXIS];
    int q[4];
    int result, ind;
    u32 word;
    u8 d[8], compass_divider;
    u8 buf[64];
    u64 *tmp;
    int source, i;
    struct inv_chip_config_s *conf;

    conf = &st->chip_config;
    ind = 0;

    if (conf->quaternion_on & conf->dmp_on) {
        for (i = 0; i < ARRAY_SIZE(q); i++) {
            q[i] = be32_to_cpup((__be32 *)(&data[ind + i * 4]));
            st->raw_quaternion[i] = q[i];
            memcpy(&buf[ind + i * sizeof(q[i])], &q[i],
                        sizeof(q[i]));
        }
        ind += QUATERNION_BYTES;
    }

    if (conf->accl_fifo_enable) {
        for (i = 0; i < ARRAY_SIZE(a); i++) {
            a[i] = be16_to_cpup((__be16 *)(&data[ind + i * 2]));
            memcpy(&buf[ind + i * sizeof(a[i])], &a[i],
                        sizeof(a[i]));
        }
        ind += BYTES_PER_SENSOR;
    }

    if (conf->gyro_fifo_enable) {
        for (i = 0; i < ARRAY_SIZE(g); i++) {
            g[i] = be16_to_cpup((__be16 *)(&data[ind + i * 2]));
            memcpy(&buf[ind + i * sizeof(g[i])], &g[i],
                        sizeof(g[i]));
        }
        ind += BYTES_PER_SENSOR;
    }

    if (conf->dmp_on && (conf->tap_on || conf->display_orient_on)) {
        word = (u32)(be32_to_cpup((u32 *)&data[ind]));
```

```
        source = ((word >> 16) & 0xff);
        if (source) {
            st->tap_data = (DMP_MASK_TAP & (word & 0xff));
            st->display_orient_data =
            ((DMP_MASK_DIS_ORIEN & (word & 0xff)) >>
              DMP_DIS_ORIEN_SHIFT);
        }

        /* report tap information */
        if (source & INT_SRC_TAP)
            sysfs_notify(&indio_dev->dev.kobj, NULL, "event_tap");
        /* report orientation information */
        if (source & INT_SRC_DISPLAY_ORIENT)
            sysfs_notify(&indio_dev->dev.kobj, NULL,
                     "event_display_orientation");
    }
    /*divider and counter is used to decrease the speed of read in
        high frequency sample rate*/
    if (conf->compass_fifo_enable) {
#if 1
        c[0] = g_akm_mag[0];
        c[1] = g_akm_mag[1];
        c[2] = g_akm_mag[2];
#else
        c[0] = 0;
        c[1] = 0;
        c[2] = 0;
        if (conf->dmp_on)
            compass_divider = st->compass_dmp_divider;
        else
            compass_divider = st->compass_divider;

        if (compass_divider <= st->compass_counter) {
            /*read from external sensor data register */
            result = inv_i2c_read(st, REG_EXT_SENS_DATA_00,
                          NUM_BYTES_COMPASS_SLAVE, d);
            /* d[7] is status 2 register */
            /*for AKM8975, bit 2 and 3 should be all be zero*/
            /* for AMK8963, bit 3 should be zero*/
            if ((DATA_AKM_DRDY == d[0]) &&
                (0 == (d[7] & DATA_AKM_STAT_MASK)) &&
                (!result)) {
                u8 *sens;
                sens = st->chip_info.compass_sens;
                c[0] = (short)((d[2] << 8) | d[1]);
                c[1] = (short)((d[4] << 8) | d[3]);
                c[2] = (short)((d[6] << 8) | d[5]);
                c[0] = (short)(((int)c[0] *
                           (sens[0] + 128)) >> 8);
                c[1] = (short)(((int)c[1] *
                           (sens[1] + 128)) >> 8);
                c[2] = (short)(((int)c[2] *
                           (sens[2] + 128)) >> 8);
                c[0] ^= (short)t;
                c[1] ^= (short)t;
```

```
                c[2] ^= (short)t;
                st->raw_compass[0] = c[0];
                st->raw_compass[1] = c[1];
                st->raw_compass[2] = c[2];
            }
            st->compass_counter = 0;
        } else if (compass_divider != 0) {
            //st->compass_counter++;
        }
#endif

        if (!conf->normal_compass_measure) {
            c[0] = 0;
            c[1] = 0;
            c[2] = 0;
            conf->normal_compass_measure = 1;
        }
        for (i = 0; i < 3; i++)
            memcpy(&buf[ind + i * sizeof(c[i])], &c[i],
                        sizeof(c[i]));

        ind += BYTES_PER_SENSOR;
    }
    tmp = (u64 *)buf;
    tmp[DIV_ROUND_UP(ind, 8)] = t;

    if (ind > 0) {
//      new_tm(t, &test_delay[DELAY_STAT_REPORT], "MPU KRNL report");

        int64_t tm_cur = get_time_ns();
        int64_t tm_delta = tm_cur - t;
        if (tm_min==0 && tm_max==0)
            tm_min = tm_max = tm_delta;
        else if (tm_delta < tm_min)
            tm_min = tm_delta;
        else if (tm_delta > tm_max)
            tm_max = tm_delta;
        tm_sum += tm_delta;
        tm_count++;

        if (unlikely((tm_cur-tm_last_print) >= 1000000000)) {
//          pr_info("MPU6050 report size: %d rate: %lld\n", ind, tm_count);
//          pr_info("MPU KRNL report: [%lld] %lld,%d,%lld\n", t, tm_min,
(u32)tm_sum/(u32)tm_count, tm_max);
            tm_last_print = tm_cur;
            tm_min = tm_max = tm_count = tm_sum = 0;
        }

#ifdef INV_KERNEL_3_10
            iio_push_to_buffers(indio_dev, buf);
#else
        iio_push_to_buffer(indio_dev->buffer, buf, t);
#endif
//      pr_info("MPU KERL report: tm=%lld, %lld, %lld\n", t, tm, tm-t);
    }
```

```
        return 0;
    }
```

这个函数根据MPU芯片的标志位，对数据进行了解析。最后使用 `iio_push_to_buffers(indio_dev, buf);` 把数据送至IIO设备的缓存，IIO子系统会把该数据通过虚拟文件系统进行上报。

小结：驱动层没有对原始的数据进行改动，直接进行上报，rk288对数据的改动应该在HAL进行。