

MPU6000读取过程

```
rc.sensors中
if mpu6000 -X -R 4 start
then
Firmware/src/drivers/mpu6000/ mpu6000.cpp 中
extern "C" { __EXPORT int mpu6000_main(int argc, char *argv[]); }

int mpu6000_main(int argc, char *argv[])
{
    .....
    if (!strcmp(verb, "start")) {
        mpu6000::start(external_bus, rotation, accel_range, device_type);
    }
    .....
}

void
start(bool external_bus, enum Rotation rotation, int range, int device_type)
{
    //路径
    int fd;
    MPU6000 **g_dev_ptr = external_bus ? &g_dev_ext : &g_dev_int;
    const char *path_accel = external_bus ? MPU_DEVICE_PATH_ACCEL_EXT : MPU_DEVICE_PATH_ACCEL;
    const char *path_gyro = external_bus ? MPU_DEVICE_PATH_GYRO_EXT : MPU_DEVICE_PATH_GYRO;

    if (*g_dev_ptr != nullptr)
        /* if already started, the still command succeeded */
    {
        errx(0, "already started");
    }

    /* create the driver */
    if (external_bus) {
#ifdef PX4_SPI_BUS_EXT
# if defined(PX4_SPIDEV_EXT_ICM)
        spi_dev_e cs = (spi_dev_e)(device_type == 6000 ? PX4_SPIDEV_EXT_MPU : PX4_SPIDEV_EXT_ICM);
# else
        spi_dev_e cs = (spi_dev_e) PX4_SPIDEV_EXT_MPU;
# endif
        *g_dev_ptr = new MPU6000(PX4_SPI_BUS_EXT, path_accel, path_gyro, cs, rotation, device_type);
# else
        errx(0, "External SPI not available");
# endif

    } else {
#ifdef PX4_SPIDEV_ICM
        spi_dev_e cs = (spi_dev_e)(device_type == 6000 ? PX4_SPIDEV_MPU : PX4_SPIDEV_ICM);
# else
```

```

        spi_dev_e cs = (spi_dev_e) PX4_SPIDEV_MPU;
#endif

        *g_dev_ptr = new MPU6000(PX4_SPI_BUS_SENSORS, path_accel, path_gyro, cs, rotation, device_type);
    }

    if (*g_dev_ptr == nullptr) {
        goto fail;
    }

    //注册设备名
    if (OK != (*g_dev_ptr)->init()) {
        goto fail;
    }

    //打开设备
    /* set the poll rate to default, starts automatic data collection */
    fd = open(path_accel, O_RDONLY);

    if (fd < 0) {
        goto fail;
    }

    //这里操作传感器
    if (ioctl(fd, SENSORIOCPOLLRATE, SENSOR_POLLRATE_DEFAULT) < 0) {
        goto fail;
    }

    if (ioctl(fd, ACCELIOSRANGE, range) < 0) {
        goto fail;
    }

    close(fd);

    exit(0);
fail:
    if (*g_dev_ptr != nullptr) {
        delete(*g_dev_ptr);
        *g_dev_ptr = nullptr;
    }
    errx(1, "no device on this bus");
}

跳到 init()中
int
MPU6000::init()
{
    int ret;

    /* do SPI init (and probe) first */
    ret = SPI::init();

    /* if probe/setup failed, bail now */
    if (ret != OK) {

```

```

        DEVICE_DEBUG("SPI setup failed");
        return ret;
    }

    /* allocate basic report buffers */
    _accel_reports = new ringbuffer::RingBuffer(2, sizeof(accel_report));

    if (_accel_reports == nullptr) {
        goto out;
    }

    _gyro_reports = new ringbuffer::RingBuffer(2, sizeof(gyro_report));

    if (_gyro_reports == nullptr) {
        goto out;
    }

    if (reset() != OK) {
        goto out;
    }

    /* Initialize offsets and scales */
    _accel_scale.x_offset = 0;
    _accel_scale.x_scale = 1.0f;
    _accel_scale.y_offset = 0;
    _accel_scale.y_scale = 1.0f;
    _accel_scale.z_offset = 0;
    _accel_scale.z_scale = 1.0f;

    _gyro_scale.x_offset = 0;
    _gyro_scale.x_scale = 1.0f;
    _gyro_scale.y_offset = 0;
    _gyro_scale.y_scale = 1.0f;
    _gyro_scale.z_offset = 0;
    _gyro_scale.z_scale = 1.0f;

    /* do CDev init for the gyro device node, keep it optional */
    ret = _gyro->init();

    /* if probe/setup failed, bail now */
    if (ret != OK) {
        DEVICE_DEBUG("gyro init failed");
        return ret;
    }

    _accel_class_instance = register_class_devname(ACCEL_BASE_DEVICE_PATH);

    measure();

```

```

/* advertise sensor topic, measure manually to initialize valid report */
struct accel_report arp;
_accel_reports->get(&arp);

/* measurement will have generated a report, publish */
_accel_topic = orb_advertise_multi(ORB_ID(sensor_accel), &arp,
                                   &_accel_orb_class_instance, (is_external()) ? ORB_PRIO_MAX : ORB_PRIO_HIGH);

if (_accel_topic == nullptr) {
    warnx("ADVERT FAIL");
}

```

```

/* advertise sensor topic, measure manually to initialize valid report */
struct gyro_report grp;
_gyro_reports->get(&grp);

_gyro->_gyro_topic = orb_advertise_multi(ORB_ID(sensor_gyro), &grp,
                                          &_gyro->_gyro_orb_class_instance, (is_external()) ? ORB_PRIO_MAX : ORB_PRIO_HIGH);

if (_gyro->_gyro_topic == nullptr) {
    warnx("ADVERT FAIL");
}

```

out:

```
return ret;
```

```
}
```

与 init()并列的 ioctl()中

int

```
MPU6000::ioctl(struct file *filp, int cmd, unsigned long arg)
```

```
{
```

```
switch (cmd) {
```

```
case SENSORIOCRESET:
```

```
return reset();
```

```
case SENSORIOCSPOLLRATE: {
```

```
switch (arg) {
```

```
/* switching to manual polling */
```

```
case SENSOR_POLLRATE_MANUAL:
```

```
stop();
```

```
_call_interval = 0;
```

```
return OK;
```

```
/* external signalling not supported */
```

```
case SENSOR_POLLRATE_EXTERNAL:
```

```

/* zero would be bad */
case 0:
    return -EINVAL;

/* set default/max polling rate */
case SENSOR_POLLRATE_MAX:
    return ioctl(filp, SENSORIOCSPOLLRATE, 1000);

case SENSOR_POLLRATE_DEFAULT:
    return ioctl(filp, SENSORIOCSPOLLRATE, MPU6000_ACCEL_DEFAULT_RATE);

/* adjust to a legal polling interval in Hz */
/* 设置 measure 的采样周期 */
default: {
    /* do we need to start internal polling? */
    bool want_start = (_call_interval == 0);

    /* convert hz to hrt interval via microseconds */
    unsigned ticks = 1000000 / arg;

    /* check against maximum sane rate */
    if (ticks < 1000) {
        return -EINVAL;
    }

    // adjust filters
    float cutoff_freq_hz = _accel_filter_x.get_cutoff_freq();
    float sample_rate = 1.0e6f / ticks;
    _set_dlpf_filter(cutoff_freq_hz);
    _accel_filter_x.set_cutoff_frequency(sample_rate, cutoff_freq_hz);
    _accel_filter_y.set_cutoff_frequency(sample_rate, cutoff_freq_hz);
    _accel_filter_z.set_cutoff_frequency(sample_rate, cutoff_freq_hz);

    float cutoff_freq_hz_gyro = _gyro_filter_x.get_cutoff_freq();
    _set_dlpf_filter(cutoff_freq_hz_gyro);
    _gyro_filter_x.set_cutoff_frequency(sample_rate, cutoff_freq_hz_gyro);
    _gyro_filter_y.set_cutoff_frequency(sample_rate, cutoff_freq_hz_gyro);
    _gyro_filter_z.set_cutoff_frequency(sample_rate, cutoff_freq_hz_gyro);

    /* update interval for next measurement */
    /* XXX this is a bit shady, but no other way to adjust... */
    _call_interval = ticks;

    /*
     * set call interval faster then the sample time. We
     * then detect when we have duplicate samples and reject
     * them. This prevents aliasing due to a beat between the
     * stm32 clock and the mpu6000 clock

```

```

        */
        _call.period = _call_interval - MPU6000_TIMER_REDUCTION;

        /* if we need to start the poll state machine, do it */
        /* 这里进行 measure */
        if (want_start) {
            start();
        }

        return OK;
    }
}

case SENSORIOCGPOLLRATE:
    if (_call_interval == 0) {
        return SENSOR_POLLRATE_MANUAL;
    }

    return 1000000 / _call_interval;

case SENSORIOCSQUEUEDEPTH: {
    /* lower bound is mandatory, upper bound is a sanity check */
    if ((arg < 1) || (arg > 100)) {
        return -EINVAL;
    }

    irqstate_t flags = px4_enter_critical_section();

    if (!_accel_reports->resize(arg)) {
        px4_leave_critical_section(flags);
        return -ENOMEM;
    }

    px4_leave_critical_section(flags);

    return OK;
}

case SENSORIOCGQUEUEDEPTH:
    return _accel_reports->size();

case ACCELIOCGSAMPLERATE:
    return _sample_rate;

case ACCELIOCSSAMPLERATE:
    _set_sample_rate(arg);
    return OK;

```

```

case ACCELIOCGLOWPASS:
    return _accel_filter_x.get_cutoff_freq();

case ACCELIOCLOWPASS:
    // set hardware filtering
    _set_dlpf_filter(arg);
    // set software filtering
    _accel_filter_x.set_cutoff_frequency(1.0e6f / _call_interval, arg);
    _accel_filter_y.set_cutoff_frequency(1.0e6f / _call_interval, arg);
    _accel_filter_z.set_cutoff_frequency(1.0e6f / _call_interval, arg);
    return OK;

case ACCELIOCSSCALE: {
    /* copy scale, but only if off by a few percent */
    struct accel_calibration_s *s = (struct accel_calibration_s *) arg;
    float sum = s->x_scale + s->y_scale + s->z_scale;

    if (sum > 2.0f && sum < 4.0f) {
        memcpy(&_accel_scale, s, sizeof(_accel_scale));
        return OK;
    } else {
        return -EINVAL;
    }
}

case ACCELIOCGSCALE:
    /* copy scale out */
    memcpy((struct accel_calibration_s *) arg, &_accel_scale, sizeof(_accel_scale));
    return OK;

case ACCELIOCSRANGE:
    return set_accel_range(arg);

case ACCELIOCGRANGE:
    return (unsigned long)((_accel_range_m_s2) / MPU6000_ONE_G + 0.5f);

case ACCELIOCSELFTEST:
    return accel_self_test();

default:
    /* give it to the superclass */
    return SPI::ioctl(filp, cmd, arg);
}

```

```

}
进入 start()

```

```

void
MPU6000::start()
{

```

```

/* make sure we are stopped first */
stop();

/* discard any stale data in the buffers */
_accel_reports->flush();
_gyro_reports->flush();

/* start polling at the specified rate */
hrt_call_every(&_call,
               1000,
               _call_interval - MPU6000_TIMER_REDUCTION,
               (hrt_callout)&MPU6000::measure_trampoline, this);
}

```

进入 `measure_trampoline()`

void

`MPU6000::measure_trampoline(void *arg)`

```

{
    MPU6000 *dev = reinterpret_cast<MPU6000 *>(arg);

    /* make another measurement */
    dev->measure();
}

```

该函数实现周期延时 100ms 后以 `_call_interval` 这个周期调用 `MPU6000::measure_trampoline` 这个函数。而

`measure_trampoline` 又调用 `measure`。从而实现一直读取传感器的数值。即我们可以通过调用 **`MPU6000::ioctl`**

来调用 **`measure`** 函数。

进入 `measure()`

`IMU380::measure()`

```

{
    ...
    //读取传感器的值
    /*
     * Fetch the full set of measurements from the MPU6000 in one pass.
     */
    mpu_report.cmd = DIR_READ | MPUREG_INT_STATUS;

    // sensor transfer at high clock speed
    //时钟频率
    set_frequency(MPU6000_HIGH_BUS_SPEED);
    //读传感器
    if (OK != transfer((uint8_t *)&mpu_report, ((uint8_t *)&mpu_report), sizeof(mpu_report))) {
        return;
    }

    check_registers(); ..
    //将加速度计的值转化为加速度单位 m/s^2.
    _accel_range_scale = 1.0f / 4000.0f;
}

```



```

float x_in_new = ((xraw_f * _accel_range_scale) - _accel_scale.x_offset) * _accel_scale.x_scale;
float y_in_new = ((yraw_f * _accel_range_scale) - _accel_scale.y_offset) * _accel_scale.y_scale;
float z_in_new = ((zraw_f * _accel_range_scale) - _accel_scale.z_offset) * _accel_scale.z_scale;
...
//滤波
arb.x = _accel_filter_x.apply(x_in_new);
arb.y = _accel_filter_y.apply(y_in_new);
arb.z = _accel_filter_z.apply(z_in_new);

//积分
math::Vector<3> aval(x_in_new, y_in_new, z_in_new);
math::Vector<3> aval_integrated;

bool accel_notify = _accel_int.put(arb.timestamp, aval, aval_integrated, arb.integral_dt);
arb.x_integral = aval_integrated(0);
arb.y_integral = aval_integrated(1);
arb.z_integral = aval_integrated(2);
...
//此处陀螺仪类似
...

//最后通过消息发送出去
if (accel_notify && !(_pub_blocked)) {
    /* publish it */
    orb_publish(ORB_ID(sensor_accel), _accel_topic, &arb);
}

if (gyro_notify && !(_pub_blocked)) {
    /* publish it */
    orb_publish(ORB_ID(sensor_gyro), _gyro->_gyro_topic, &grb);
}
}

```

有以下几点得知道在哪设置：

读传感器的时钟频率: `set_frequency(MPU6000_HIGH_BUS_SPEED);`

传感器采样周期: `ioctl()`中的 `default` 里面的 `_call_interval` 值

用逻辑分析仪分析



SPI 驱动结构

接着 MPU6000 读取过程

进入 if (OK != transfer((uint8_t *)&mpu_report, ((uint8_t *)&mpu_report), sizeof(mpu_report)))这里的 transfer()
int

SPI::transfer(uint8_t *send, uint8_t *recv, unsigned len)

```
{
    int result;

    if ((send == nullptr) && (recv == nullptr)) {
        return -EINVAL;
    }

    LockMode mode = up_interrupt_context() ? LOCK_NONE : locking_mode;

    /* lock the bus as required */
    switch (mode) {
    default:
    case LOCK_PREEMPTION: {
        irqstate_t state = px4_enter_critical_section();
        result = _transfer(send, recv, len); //读取
        px4_leave_critical_section(state);
    }
    break;

    case LOCK_THREADS:
        SPI_LOCK(_dev, true);
        result = _transfer(send, recv, len); //读取
        SPI_LOCK(_dev, false);
        break;

    case LOCK_NONE:
        result = _transfer(send, recv, len); //读取
        break;
    }

    return result;
}
```

进入 _transfer(send, recv, len)

int

SPI::_transfer(uint8_t *send, uint8_t *recv, unsigned len)

```
{
    SPI_SETFREQUENCY(_dev, _frequency);
    SPI_SETMODE(_dev, _mode);
    SPI_SETBITS(_dev, 8);
    SPI_SELECT(_dev, _device, true);

    /* do the transfer */
```

```

    SPI_EXCHANGE(_dev, send, recv, len);

    /* and clean up */
    SPI_SELECT(_dev, _device, false);

    return OK;
}

```

这个里面的函数随便挑一个进入都会跟到

Firmware/build_px4fmu-v2_default/px4fmu-v2/Nuttx/nuttx/include/nuttx/spi.h 里面来

比如进入 SPI_SETFREQUENCY(_dev, _frequency);

跟到了#define SPI_SETFREQUENCY(d,f) ((d)->ops->setfrequency(d,f))

再进入((d)->ops->setfrequency(d,f))

跟到了 uint32_t (*setfrequency)(FAR struct spi_dev_s *dev, uint32_t frequency);

```

struct spi_ops_s
{
#ifdef CONFIG_SPI_OWNBUS
    int      (*lock)(FAR struct spi_dev_s *dev, bool lock);
#endif
    void      (*select)(FAR struct spi_dev_s *dev, enum spi_dev_e devid,
                        bool selected);
    uint32_t (*setfrequency)(FAR struct spi_dev_s *dev, uint32_t frequency);
    void      (*setmode)(FAR struct spi_dev_s *dev, enum spi_mode_e mode);
    void      (*setbits)(FAR struct spi_dev_s *dev, int nbits);
    uint8_t   (*status)(FAR struct spi_dev_s *dev, enum spi_dev_e devid);
#ifdef CONFIG_SPI_CMDDATA
    int      (*cmddata)(FAR struct spi_dev_s *dev, enum spi_dev_e devid, bool cmd);
#endif
    uint16_t (*send)(FAR struct spi_dev_s *dev, uint16_t wd);
#ifdef CONFIG_SPI_EXCHANGE
    void      (*exchange)(FAR struct spi_dev_s *dev, FAR const void *txbuffer,
                          FAR void *rxbuffer, size_t nwords);
#else
    void      (*sndblock)(FAR struct spi_dev_s *dev, FAR const void *buffer,
                          size_t nwords);
    void      (*recvblock)(FAR struct spi_dev_s *dev, FAR void *buffer,
                           size_t nwords);
#endif
    int      (*registercallback)(FAR struct spi_dev_s *dev, spi_mediachange_t callback,
                                void *arg);
};

```

```

struct spi_dev_s
{
    const struct spi_ops_s *ops;
};

```

build_px4fmu-v2_default 这个目录下都是编译时自动拷贝系统文件生成的，其实

build_px4fmu-v2_default/px4fmu-v2/Nuttx/nuttx/include/nuttx/spi.h 来自于 Firmware/Nuttx/nuttx/include/nuttx/spi.h
 这是 nuttx 中关于 spi 的抽象，定义了 spi 操作的所有结构体和 API。是对 spi 对象最关键的抽象结构体。

可以看出，spi_ops_s 是一个函数的结构体，当调用这些函数时将会调用最底层 stm32_spi.c 中定义的函数来完成操作。如调用 exchange 函数将调用 stm32_spi.c 中的 spi_exchange 函数。

当把 spi_ops_s 结构体设置完后，就可以调用最底层 stm32_spi.c，设置 stm32f4 的寄存器了

由此，关于 SPI 的读取数据的整个框架就非常清晰了！

SPI 有 pix 的抽象层，nuttX 的抽象层指针结构体，stm32 的驱动层

接下来是初始化的框架

px4fmu-v2 中用到了三条的 SPI，一条 SPI1，一条 SPI2 和一条 SPI4，其中 SPI1 用作传感器，SPI2 用作 RAM，

SPI4 作为外部引出。具体定义在 Firmware/src/driver/boards/px4fmu-v2/board_config.h 中有

```
#define PX4_SPI_BUS_SENSORS 1
#define PX4_SPI_BUS_RAMTRON 2
#define PX4_SPI_BUS_EXT 4
#define PX4_SPI_BUS_BARO PX4_SPI_BUS_SENSORS
```

在 Firmware/src/drivers/mpu6000/mpu6000.cpp 中

```
void
```

```
start(bool external_bus, enum Rotation rotation, int range, int device_type)
```

```
{
```

```
.....
```

```
}
```

可以看到 mpu6000 用的是外部 SPI

```
start(bool external_bus, enum Rotation rotation, int range, int device_type)
```

```
{
```

```
.....
```

```
*g_dev_ptr = new MPU6000(PX4_SPI_BUS_SENSORS, path_accel, path_gyro, cs, rotation, device_type);
```

```
.....
```

```
if (OK != (*g_dev_ptr)->init())
```

```
.....
```

```
}
```

start 函数就是调用 new MPU6000(PX4_SPI_BUS_SENSORS, path_accel, path_gyro, cs, rotation, device_type); 的构造函数，而我们在构造函数中发现有

```
MPU6000::MPU6000(int bus, const char *path_accel, const char *path_gyro, spi_dev_e device, enum Rotation rotation, int device_type) :
```

```
    SPI("MPU6000", path_accel, bus, device, SPIDEV_MODE3, MPU6000_LOW_BUS_SPEED),
```

```
.....
```

也就是调用 MPU6000 构造函数的时候也会调用 SPI 的构造函数。

接着进入 (*g_dev_ptr)->init()

```
Int MPU6000::init()
```

```
{
```

```
    int ret;
```

```

    /* do SPI init (and probe) first */
    ret = SPI::init();
    .....
}
再进入 SPI::init()
int SPI::init()
{
    .....
    if (_dev == nullptr) {
        _dev = px4_spibus_initialize(_bus);
    }
    .....
    /* do base class init, which will create the device node, etc. */
    ret = CDev::init();
    .....
}

```

该函数 `_dev = px4_spibus_initialize(_bus);` 在 **src/platforms/px4_micro_hal.h** 中定义的

```

# define px4_spibus_initialize(port_1based)    up_spiinitialize(port_1based)
up_spiinitialize() 在 Firmware/Nuttx/nuttx/include/nuttx/spi.h 中声明，函数实现在
Firmware/Nuttx/nuttx/arch/arm/src/stm32/stm32_spi.c
FAR struct spi_dev_s *up_spiinitialize(int port)
{
    FAR struct stm32_spidev_s *priv = NULL;

    irqstate_t flags = irqsave();

#ifdef CONFIG_STM32_SPI1
    if (port == 1)
    {
        /* Select SPI1 */

        priv = &g_spildev;

        /* Only configure if the port is not already configured */

        if ((spi_getreg(priv, STM32_SPI_CR1_OFFSET) & SPI_CR1_SPE) == 0)
        {
            /* Configure SPI1 pins: SCK, MISO, and MOSI */

            stm32_configgpio(GPIO_SPI1_SCK);
            stm32_configgpio(GPIO_SPI1_MISO);
            stm32_configgpio(GPIO_SPI1_MOSI);

            /* Set up default configuration: Master, 8-bit, etc. */

            spi_portinitialize(priv);
        }
    }
}

```

```

else
#endif
#ifdef CONFIG_STM32_SPI2
    if (port == 2)
    {
        /* Select SPI2 */

        priv = &g_spi2dev;

        /* Only configure if the port is not already configured */

        if ((spi_getreg(priv, STM32_SPI_CR1_OFFSET) & SPI_CR1_SPE) == 0)
        {
            /* Configure SPI2 pins: SCK, MISO, and MOSI */

            stm32_configgpio(GPIO_SPI2_SCK);
            stm32_configgpio(GPIO_SPI2_MISO);
            stm32_configgpio(GPIO_SPI2_MOSI);

            /* Set up default configuration: Master, 8-bit, etc. */

            spi_portinitialize(priv);
        }
    }
else
#endif
#ifdef CONFIG_STM32_SPI3
    if (port == 3)
    {
        /* Select SPI3 */

        priv = &g_spi3dev;

        /* Only configure if the port is not already configured */

        if ((spi_getreg(priv, STM32_SPI_CR1_OFFSET) & SPI_CR1_SPE) == 0)
        {
            /* Configure SPI3 pins: SCK, MISO, and MOSI */

            stm32_configgpio(GPIO_SPI3_SCK);
            stm32_configgpio(GPIO_SPI3_MISO);
            stm32_configgpio(GPIO_SPI3_MOSI);

            /* Set up default configuration: Master, 8-bit, etc. */

            spi_portinitialize(priv);
        }
    }
else

```

```

#endif
#ifdef CONFIG_STM32_SPI4
    if (port == 4)
    {
        /* Select SPI4 */

        priv = &g_spi4dev;

        /* Only configure if the port is not already configured */

        if ((spi_getreg(priv, STM32_SPI_CR1_OFFSET) & SPI_CR1_SPE) == 0)
        {
            /* Configure SPI4 pins: SCK, MISO, and MOSI */

            stm32_configgpio(GPIO_SPI4_SCK);
            stm32_configgpio(GPIO_SPI4_MISO);
            stm32_configgpio(GPIO_SPI4_MOSI);

            /* Set up default configuration: Master, 8-bit, etc. */

            spi_portinitialize(priv);
        }
    }
else
#endif
#ifdef CONFIG_STM32_SPI5
    if (port == 5)
    {
        /* Select SPI5 */

        priv = &g_spi5dev;

        /* Only configure if the port is not already configured */

        if ((spi_getreg(priv, STM32_SPI_CR1_OFFSET) & SPI_CR1_SPE) == 0)
        {
            /* Configure SPI5 pins: SCK, MISO, and MOSI */

            stm32_configgpio(GPIO_SPI5_SCK);
            stm32_configgpio(GPIO_SPI5_MISO);
            stm32_configgpio(GPIO_SPI5_MOSI);

            /* Set up default configuration: Master, 8-bit, etc. */

            spi_portinitialize(priv);
        }
    }
else
#endif

```

```

#ifdef CONFIG_STM32_SPI6
    if (port == 6)
    {
        /* Select SPI6 */

        priv = &g_spi6dev;

        /* Only configure if the port is not already configured */

        if ((spi_getreg(priv, STM32_SPI_CR1_OFFSET) & SPI_CR1_SPE) == 0)
        {
            /* Configure SPI6 pins: SCK, MISO, and MOSI */

            stm32_configgpio(GPIO_SPI6_SCK);
            stm32_configgpio(GPIO_SPI6_MISO);
            stm32_configgpio(GPIO_SPI6_MOSI);

            /* Set up default configuration: Master, 8-bit, etc. */

            spi_portinitialize(priv);
        }
    }
    else
#endif
    {
        spidbg("ERROR: Unsupported SPI port: %d\n", port);
        return NULL;
    }

    irqrestore(flags);
    return (FAR struct spi_dev_s *)priv;
}

```

其次，对于需要注册到 cdev 中的 spi 来说：