



# R818 Android 10 Input 开发指南

版本号: 1.0  
发布日期: 2020.06.04

## 版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.06.04		Input 开发使用说明

## 目 录

<b>1 前言</b>	<b>1</b>
1.1 编写目的	1
1.2 适用范围	1
1.3 相关人员	1
<b>2 驱动模块的使用</b>	<b>2</b>
2.1 驱动模块适用范围	2
2.2 驱动模块配置信息	2
2.3 使用步骤	4
2.3.1 Gsensor 使用步骤	4
2.3.2 CTP 使用步骤	6
2.3.3 lsensor 使用步骤	7
2.3.4 gyroscope 使用步骤	8
2.4 如何添加一款新的设备	8
2.4.1 增加设备驱动	9
2.4.1.1 驱动中 I2C 地址的设计	9
2.4.2 驱动中 detect 函数的设计	9
2.4.3 Sensor Hal 层增加	11
<b>3 其他注意事项</b>	<b>13</b>

# 1 前言

## 1.1 编写目的

为达到能快速使用的目的。文档对 CTP 与 sensor 的加载原理，使用方法步骤，如何添加一个新的模组等做了详细的讲解。

## 1.2 适用范围

介绍本模块设计适用 R818 平台。

## 1.3 相关人员

相关的开发与维护人员应该仔细阅读本文档。

## 2 驱动模块的使用

### 2.1 驱动模块适用范围

该模块适用于挂载在 I2C 总线上的设备的检测和加载。

### 2.2 驱动模块配置信息

配置文件在：longan/device/config/chips/r818/configs/xxx/board.dts，R818 已将部分 sensor 和 TP 的相关配置从 sys\_config.fex 搬到 board.dts 中，请相关模块负责人留意。

#### 1)Gsensor 配置

为加快驱动的加载速度，目前采用直接注册 i2c 设备的方式加载，因此 gsensor 的配置是配置到对应的 twi 节点下，如：

```
twi: twi@0x05002400{
    clock-frequency = <200000>;
    pinctrl-0 = <&twi_pins_a>;
    pinctrl-1 = <&twi_pins_b>;
    status = "okay";
    gsensor {
        compatible = "allwinner,sc7a20";           // 对应驱动中的 .compatible
        reg = <0x19>;                               // 标示gsensor的i2c从机地址，必填
        device_type = "gsensor";                   // 标示使用gsensor
        status = "okay";                           // 标示使能gsensor
        gsensor_twi_id = <0x1>;
        gsensor_twi_addr = <0x19>;
        gsensor_int1 = <&pio PH 11 6 1 0xffffffff 0xffffffff>;
        gsensor_supply = <&reg_dcdc1>;              // 标示使用哪路电
        gsensor_vcc_io_val = <3300>;                // 标示gsensor的供电电压
    };
};
```

#### 2)CTP 配置

为加快驱动的加载速度，目前采用直接注册 i2c 设备的方式加载，因此 ctp 的配置是配置到对应的 twi 节点下，如：

```
twi0: twi@0x05002000{
    clock-frequency = <400000>;
    pinctrl-0 = <&twi0_pins_a>;
    pinctrl-1 = <&twi0_pins_b>;
    status = "okay";
```

```

ctp {
    compatible = "allwinner,gs1X680";           // 对应驱动中的 .compatible
    reg = <0x40>;                               // 标示CTP的i2c从机地址，必填
    device_type = "ctp";                       // 标示使用CTP
    status = "okay";                           // 标示使能CTP
    ctp_name = "gs1X680_3676_1280x800";        // 标示CTP的fw信息，目前gs1X680，
    gs1X680new和gt9xx的驱动需要用到此项
    ctp_twi_id = <0x0>;
    ctp_twi_addr = <0x40>;
    ctp_screen_max_x = <0x320>;                // 标示CTP的x轴范围
    ctp_screen_max_y = <0x500>;                // 标示CTP的y轴范围
    ctp_revert_x_flag = <1>;                    // 标示CTP的x轴是否翻转
    ctp_revert_y_flag = <1>;                    // 标示CTP的y轴是否翻转

    ctp_exchange_x_y_flag = <0x1>;             // 标示CTP的x轴、y轴互换
    ctp_int_port = <&pio PH 9 6 0xffffffff 0xffffffff 0>; // 标示CTP的中断口配置
    ctp_wakeup = <&pio PH 10 1 0xffffffff 0xffffffff 1>; // 标示CTP的唤醒口配置
    ctp-supply = <&reg_ldoio0>;                 // 标示CTP使用哪路电源
    ctp_power_ldo_vol = <3300>;                 // 标示CTP的供电电压
};
};
};

```

### 3) Lsensor 配置

为加快驱动的加载速度，目前采用直接注册 i2c 设备的方式加载，因此 light sensor 的配置是配置到对应的 twi 节点下，如：

```

twi1: twi@0x05002400{
    clock-frequency = <200000>;
    pinctrl-0 = <&twi1_pins_a>;
    pinctrl-1 = <&twi1_pins_b>;
    status = "okay";
    lightsensor {
        compatible = "allwinner,stk3x1x";       // 对应驱动中的 .compatible
        reg = <0x48>;                           // 标示lightsensor的i2c从机地址，必
        填
        device_type = "lightsensor";            // 标示使用lightsensor
        status = "okay";                        // 标示使能lightsensor
        ls_twi_id = <0x1>;
        ls_twi_addr = <0x48>;
        ls_int = <&pio PH 4 6 1 0xffffffff 0xffffffff>; // 标示lightsensor的中断
        管脚
        lightsensor-supply = <&reg_dcdc1>;        // 标示lightsensor使用哪路电源
    };
};

```

### 4) gyroscope 配置

gyroscope 的配置仍在 longan/device/config/chips/R818/configs/xxx/sys\_config.fex 中进行配置。

```

;-----
;gyroscope
;-----
[gy_para]
gy_used          = 0
gy_twi_id        = 2

```

```
gy_twi_addr      = 0x6a
gy_int1          = port:PA10<6><1><default><default>
gy_int2          =
```

## 2.3 使用步骤

### 2.3.1 Gsensor 使用步骤

#### 1) 支持的模组

bma250, bma250e, bma223 (共用驱动 bma250.ko) ; mma8452 (使用 mma8452.ko) ; mma7660 (使用 mma7660.ko) ; mm8652, mma8653 (共用驱动 mma865x.ko) ; sc7a20(使用 sc7a20.ko); sc7a30(使用 sc7a30.ko); sc7660(使用 sc7660.ko); mxc6225 (使用 mxc622x.ko) 。

驱动源文件目录：longan/kernel/linux-4.9/drivers/input/sensor.

#### 2) 驱动的配置

为快速的检测到设备，可以将对应的驱动配置到内核或者编译成 ko 并加载。

如果使用 ko 加载的方式，则将在编译过程中拷贝到 android 指定的目录中备用。

编译后的目录：out/target/product/xxx/vendor/modules，省略的部分为 lunch 时选择的配置文件夹名称。

机器中对应的目录为：/vendor/modules

#### 3) board.dts 文件的修改

为对应的模组添加配置，配置电源，gpio，中断，i2c 地址等，参见 2.2 节。

#### 4) 驱动的加载

如果使用 ko 加载的方式，则在 android/device/softwinner/xxx-xxx/init.device.rc 中添加驱动加载的模块，为了快速的检测到设备，此语句应该放置在模块加载的最前面，如下所示：

```
on boot
.....

#inmod tp module

#inmod tp module
insmod /vendor/modules/gslX680new.ko
chown system system /sys/devices/platform/soc/twi0/i2c-0/0-0040/input/input4/
runtime_suspend
#insmod gsensor module
insmod /vendor/modules/bma250.ko
```

## 5) 方向的调整

使用模组前，请确认 `gsensor.cfg` 文件中是否已经存在模组的方向配置项，若没有，请添加模组的方向配置项。

`gsensor.cfg` 文件主要用于存放各个模组的方向值，只需要将已经调试好的方向值写到该文件中即可。该文件存放在 `android/device/softwinner/xxxxxx-xxx/config`，每个设备都有五项值，如下所示：

表 2-1: gsensor 含义对照表

字段	含义
<code>gsensor_name</code>	Gsensor 名称，必须与驱动中设备名相同
<code>gsensor_direct_x</code>	Gsensor x 轴的方向，当定义成 <code>true</code> 时，x 轴取正值，当定义为 <code>false</code> 时，x 轴取负值
<code>gsensor_direct_y</code>	Gsensor y 轴的方向，当定义成 <code>true</code> 时，y 轴取正值，当定义为 <code>false</code> 时，y 轴取负值
<code>gsensor_direct_z</code>	Gsensor z 轴的方向，当定义成 <code>true</code> 时，z 轴取正值，当定义为 <code>false</code> 时，z 轴取负值
<code>gsensor_xy_revert</code>	XY 轴对调，当设为 <code>TRUE</code> 时，x 轴变为原来 y 轴

该文件存放在机器的 `system/usr` 目录下，当发现方向不正确时，按照如下步骤进行调试。

Gsensor 方向调试说明：

假定机器的长轴为 X 轴，短轴为 Y 轴，垂直方向为 Z 轴。

首先调试 Z 轴：

第一步观察现象：

旋转机器，发现当只有垂直  $90^\circ$  时或者是在旋转后需要抖动一下，方向才会发生变化，则说明 Z 轴反了。若当机器大概  $45^\circ$  拿着的时候也可以旋转，说明 Z 轴方向正确。无需修改 Z 轴方向。

第二步修改 Z 轴为正确方向：

此时需要找到当前使用模组的方向向量（根据模组的名称）。如果此时该方向 Z 轴向量（`gsensor_direct_z`）的值为 `false`，则需要修改为 `true`；当为 `true`，则需要修改为 `false`。通过 `adb shell` 将修改后的 `gsensor.cfg` 文件 push 到 `system/usr` 下，重启机器，按第一步观察现象。

其次查看 X，Y 轴是否互换：

第一步观察现象：

首先假定长轴为 X 轴，短轴为 Y 轴，以 X 轴为底边将机器立起来。查看机器的 X，Y 方向是否正好互换，若此时机器的 X，Y 方向正好互换，在说明需要将 X，Y 方向交换。若此时 X，Y 方向没有反置，则进入 X，Y 方向的调试。



## 第二步交换 X, Y 方向

当需要 X, Y 方向交换时, 此时需要找到当前使用模组的方向向量 (根据模组的名称)。如果此时该 X, Y 轴互换向量 (`gsensor_xy_revert`) 的值为 `false`, 则需要修改为 `true`, 当为 `true`, 则需要修改为 `false`。通过 `adb shell` 将修改后的 `gsnesor.cfg` 文件 `push` 到 `system/usr` 下, 重启机器, 按第一步观察现象。

再次调试 X, Y 轴方向:

第一步观察现象:

首先假定长轴为 X 轴, 短轴为 Y 轴, 以 X 轴为底边将机器立起来, 查看机器的方向是否正确, 如果正确, 说明长轴配置正确, 如果方向正好相反, 说明长轴配置错误。将机器旋转到短轴, 查看机器方向是否正确, 如果正确, 说明短轴配置正确, 如果方向正好相反, 说明短轴配置错误。

第二步修改 X, Y 轴方向:

当需要修改 X, Y 轴方向时, 当只有长轴方向相反或者是只有短轴方向相反时, 则只修改方向不正确的一个轴, 当两个方向都相反时, 则同时修改 X 与 Y 轴方向向量。找到当前使用模组的方向向量 (根据模组的名称)。

若长轴方向相反, 如果此时该方向 X 轴向量 (`gsnesor_direct_x`) 的值为 `false`, 则需要修改为 `true`, 当为 `true`, 则需要修改为 `false`。

若短轴方向相反, 如果此时该方向 Y 轴向量 (`gsnesor_direct_y`) 的值为 `false`, 则需要修改为 `true`, 当为 `true`, 则需要修改为 `false`。

通过 `adb shell` 将修改后的 `gsnesor.cfg` 文件 `push` 到 `system/usr` 下, 重启机器, 按第一步观察现象。若发现还是反向 X 轴或者 Y 轴的方向仍然相反, 则说明 X 轴为短轴, Y 轴为长轴。此时:

若长轴方向相反, 如果此时该方向 Y 轴向量 (`gsnesor_direct_y`) 的值为 `false`, 则需要修改为 `true`, 当为 `true`, 则需要修改为 `false`。

若短轴方向相反, 如果此时该方向 X 轴向量 (`gsnesor_direct_x`) 的值为 `false`, 则需要修改为 `true`, 当为 `true`, 则需要修改为 `false`。

## 2.3.2 CTP 使用步骤

### 1) 支持的模组

FT 系列: `ft5202`, `ft5204`, `ft5316`, `ft5x06`(共用一个驱动, `ft5x_ts.ko`); 源码目录: `longan/kernel/linux-4.9/drivers/input/touchscreen/ftxxxx`。

Goodix 系列: `gt813`, `gt827`, `gt828`(`gt82x.ko`), 源码目录: `longan/kernel/linux-4.9/drivers/input/touchscreen/gt82x`。

gt9xx 系列: 以 `gt9xx` 系列中带 `flash` 的触摸 IC 可以在此驱动中添加支持, 源码目录: `longan/kernel/linux-4.9/drivers/input/touchscreen/gt9xx`。

longan/kernel/linux-4.9/drivers/input/touchscreen/gt9xxnew。

gsl 系列：gsl1680,gsl2681,gsl1688,gsl1680e(gslX680.ko), 驱动源文件目录：longan/kernel/linux-4.9/drivers/input/touchscreen/gslx680new;

## 2) 驱动的拷贝

编译过程中使用的命令：extract-bsp，将把驱动拷贝到指定的目录中备用。

编译后的目录：out/target/product/xxx/vendor/modules，省略的部分为 lunch 时选择选择的配置文件夹名称。

机器中对应的目录为：/vendor/modules

## 3) board.dts 文件的修改

为对应的模组添加配置，配置电源，gpio，中断，i2c 地址等，参见 2.2 节。

## 4) 驱动的加载

在 android/device/softwinner/peony-xxx/init.xxx.rc 中添加驱动加载的模块，为了快速的检测到设备，此命令应该放置在模块加载的最前面，如下所示：

```
on boot
.....

#inmod tp module

#inmod tp module
insmod /vendor/modules/gslX680new.ko
chown system system /sys/devices/platform/soc/twi0/i2c-0/0-0040/input/input4/
runtime_suspend
#insmod gsensor module
insmod /vendor/modules/bma250.ko
```

## 2.3.3 lsensor 使用步骤

### 1) 支持的模组

501als 和 jsa212 对应的驱动 ltr\_501.ko 和 jsa1212.ko，源码目录 longan/kernel/linux-4.9/drivers/iio/light/。

### 2) 驱动的拷贝

编译过程中使用的命令：extract-bsp，将把驱动拷贝到指定的目录中备用。

编译后的目录：out/target/product/xxx/vendormodules，省略的部分为 lunch 时选择选择的配置文件夹名称。

机器中对应的目录为：/vendor/modules

### 3) board.dts 文件的修改

为对应的模组添加配置，配置电源，gpio，中断，i2c 地址等，参见 2.2 节。

### 4) 驱动的加载

在 android/device/softwinner/peony-xxx/init.xxx.rc 中添加驱动加载的模块，为了快速的检测到设备，此命令应该放置在模块加载的最前面，如下所示：

```
on boot
.....

#insmod gsensor module

#insmod gsensor module
insmod /vendor/modules/jsa1212.ko
```

## 2.3.4 gyroscope 使用步骤

**1) 支持的模组** l3gd20 对应的驱动 l3gd20\_gyr.ko，源码目录 longan/kernel/linux-4.9/drivers/iio/gyro。

**2) 驱动的拷贝**编译过程中使用的命令：extract-bsp，将把驱动拷贝到指定的目录中备用。编译后的目录：out/target/product/xxx/vendor/modules，省略的部分为 lunch 时选择选择的配置文件夹名称。机器中对应的目录为：/vendor/modules

**3) sys\_config.fex 文件的修改**为对应的模组添加配置，配置电源，gpio，中断，i2c 地址等，参见 2.2 节。

**4) 驱动的加载**在 android/device/softwinner/peony-xxx/init.xxx.rc 中添加驱动加载的模块，为了快速的检测到设备，此命令应该放置在模块加载的最前面，如下所示：

```
on boot
.....

#insmod gsensor module

#insmod gsensor module
insmod /vendor/modules/l3gd20_gyr.ko
```

## 2.4 如何添加一款新的设备

目前添加一款新的设备有两种方式，一种是常规的直接注册到 i2c 中，一种是通过自动检测添加，由于第一种是比较常规的做法，在此不做说明。

自动检测中不支持的模组加入到自动检测模块中，请按照如下的步骤进行。

## 2.4.1 增加设备驱动

驱动设计时，设备的驱动必须编译为模块的形式。且 i2c 地址应该固定写于驱动中。对于有 chip id 的寄存器，应在 detect 函数中读取 chip id 值，进行相应的判断，对于没有 chip id 值的，则应进行 i2c 的测试，避免加载错误的驱动的情况发生。以下针对 i2c 地址以及 detect 的方法进行详细的说明。

### 2.4.1.1 驱动中 I2C 地址的设计

将设备可能出现的 I2C 地址都写入 normal\_i2c 数组中，进行检测时，只有正确的地址才能配对成功，若设备只有唯一确定的一个 I2C 地址，则只写一个即可。数组最后必须以 I2C\_CLIENT\_END 结尾，标识检测地址结束。

#### 1) 设备有多个 I2C 地址

有多个 I2C 地址的设备，以 Gsensor 驱动 bma250 为例进行说明，如下所示：

```
static const unsigned short normal_i2c[] = {0x18,0x19,0x38,0x08, I2C_CLIENT_END};
```

在驱动的 i2c\_driver 中添加设备地址表，如下所示：

```
static struct i2c_driver bma250_driver = {  
    .....  
    .address_list = normal_i2c,  
};
```

#### 2) 设备有一个唯一的 I2C 地址

设备有一个唯一的 I2C 地址，以 ctp 驱动 gslX680 为例子进行说明，如下所示：

```
static const unsigned short normal_i2c[] = {0x40, I2C_CLIENT_END};
```

在驱动的 i2c\_driver 中添加设备地址表，如下所示：

```
static struct i2c_driver gsl_ts_driver= {  
    .....  
    .address_list = normal_i2c,  
};
```

## 2.4.2 驱动中 detect 函数的设计

ctp 设备对应的为 ctp\_detect ,gsensor 函数设备对应的为 gsensor detect 函数。detect 函数的作用是进行硬件的检测，将检测通过的设备的名称复制到 info 结构体中，完成设备的匹配，

将设备挂接到 i2c 总线上。detect 函数为 i2c\_driver 中的接口，因此需要在注册 I2C 设备之前添加该函数，否则将可以造成驱动加载失败。

ctp 驱动中，detect 函数的添加，如下所示：

```
static int __init ft5x_ts_init(void)
{
    .....
    ft5x_ts_driver.detect = ctp_detect;
    ret = i2c_add_driver(&ft5x_ts_driver);
    .....
};
```

Gsensor 驱动中，detect 函数的添加，如下所示：

```
static int __init bma250_init(void)
{
    .....
    bma250_driver.detect = gsensor_detect;
    ret = i2c_add_driver(&bma250_driver);
    .....
};
```

1) 有 chip id 值时，读取 chip id 值，以 bma250 为例子：

```
static int gsensor_detect(struct i2c_client *client, struct i2c_board_info *info)
{
    struct i2c_adapter *adapter = client->adapter;
    int ret;

    dprintk(DEBUG_INIT, "%s enter \n", __func__);

    if (!i2c_check_functionality(adapter, I2C_FUNC_SMBUS_BYTE_DATA))
        return -ENODEV;

    if (twi_id == adapter->nr) {
        for(i2c_num=0;i2c_num<(sizeof(i2c_address)/sizeof(i2c_address[0]));i2c_num++) {
            client->addr = i2c_address[i2c_num];
            pr_info("%s:addr= 0x%x,i2c_num:%d\n",__func__,client->addr,i2c_num);
            ret = i2c_smbus_read_byte_data(client,BMA250_CHIP_ID_REG);
            pr_info("Read ID value is :%d",ret);
            if ((ret &0x00FF) == BMA250_CHIP_ID) {
                pr_info("Bosch Sensortec Device detected!\n" );
                strncpy(info->type, SENSOR_NAME, I2C_NAME_SIZE);
                return 0;
            }
            pr_info("%s:Bosch Sensortec Device not found, \
maybe the other gsensor equipment! \n",__func__);
            return -ENODEV;
        }
    } else {
        return -ENODEV;
    }
}
```

2) 当没有 chip id 值时，进行 i2c 的通信，以 gslX680 为例：

```
static int ctp_detect(struct i2c_client *client, struct i2c_board_info *info)
{
    struct i2c_adapter *adapter = client->adapter;
    int ret;

    if (!i2c_check_functionality(adapter, I2C_FUNC_SMBUS_BYTE_DATA))
        return -ENODEV;

    if(twi_id == adapter->nr){
        dprintk(DEBUG_INIT,"%s: addr= %x\n",__func__,client->addr);
        ret = ctp_i2c_test(client);
        if(!ret){
            printk("%s:I2C connection might be something wrong \n",__func__);
            return -ENODEV;
        }else{
            strncpy(info->type, CTP_NAME, I2C_NAME_SIZE);
            return 0;
        }
    }else{
        return -ENODEV;
    }
}
```

### 2.4.3 Sensor Hal 层增加

如果添加的设备为 sensor，在相应的 sensor hal 层也需要添加该设备的支持。

主要需要添加描述设备的 sensor\_t 结构体相关信息即可。

从 Android 10 起，Sensor Hal 使用 2.0 版本，R818 的 sensor hal 层目录：android/hardware/aw/sensors/aw\_sensors/2.0。

为了实现兼容，hal 层中将 sensor\_t 结构体封装为 sensor\_extend\_t 结构体，只需要在 sensorDetect.cpp 和 sensors.h 文件中增加相关内容即可。

sensors.h 中添加一个宏定义，该宏定义为收到的数据转换为 1G 时的常量。

sensorDetect.cpp，sensor 的 sensor\_extend\_t 结构体变量为 gsensorlist，可以将添加的设备的相关信息放置到该变量的任意位置即可。为了可以快速的匹配，使用的设备应该尽量的放在最前面，不使用的设备放后面。

sensor\_extend\_t 相关变量的说明如下表所示。

表 2-2: sensor\_extend\_t 变量说明表

名称	说明
Sensors::name	Sensors 结构体中的 name 为设备注册的信息，即 getevent 命令时看到的设备名称



名称	说明
Sensors::lsg	将收到的数据转换为 1G 时的常量
sList	用于描述一个传感器的结构体

## 3 其他注意事项

### 1) 关注设备特性

使用该模块时，一定要弄清楚设备的特性，关注设备的以下特性：

□. I2C 地址是否唯一。弄清楚设备的 I2C 可以有多少个，必须将可能的地址都列入扫描列表中。

□. Chip id 值是否唯一。在扫描列表中，必须列出设备的可能的所有的 chip id 值。

□. 设备在上电之后何时才可以进行操作。根据设备的特性，设备在上电之后，需要等待多少时间之后才能进行操作，确认 I2C 操作的可行性。

□. 特殊特性。如必须通过 i2c 总线进行第二次读取后，才能读取到正确的数值，目前该模块中 retry 次数为四次，即当需要重试的次数超过四次时，请做相应的修改。

### 2) 同类设备，I2C 地址相同时

同类设备中，出现 I2C 地址冲突时，如果不能凭借设备的特有特性（chip id 值等）进行区分，特别是没有 chip id 值的设备，在使用自动检测时，一定要将不使用的相同地址的设备剔除掉，可以在 sys\_config.fex 文件的扫描列表中，在想要剔除的设备名称后写 0 即可。

当设备中地址冲突的两个设备，一个有 chip id 一个没有 chip id，可以不用剔除。

同类设备，i2c 地址相同时，需要剔除的条件是：两个设备都没有 chip id 值（或者设备的特有属性时），需要将方案中不使用的设备剔除掉，否则将会按照顺序进行检测，先检测到的设备会被加载到系统中，可能造成错误。

### 3) 关于 chip id 值

chip id 值，使用时，请确认设备的 chip id 值与列表中所给的是否一致，如果不一致，请将设备的 chip id 值增加到功能模块的列表中。



## 著作权声明

版权所有 © 2022 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

## 商标声明



（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

## 免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。