Android 15 Camera
自动检测使用指南

版本号: 1.1
发布日期: 2025.06.17

版本历史

版本号	日期	制/修订人	内容描述
1.0	2024.12.25	AWA2153	创建初始版本
1.1	2025.06.17	AWA1259	修改一处描述疑议

目　录

1 概述

1.1 编写目的

介绍 Camera sensor list 的数据结构，流程，API 接口。

1.2 适用范围

适用 Android Camera 开发。

1.3 相关人员

Camera 模块开发人员，驱动维护人员，客户支持人员。

2 模块介绍

2.1 术语、定义、缩略语

- Probe：驱动注册函数。
- CCI：Camera control interface，其实质是 twi 协议的子集的实现。
- CSI：CMOS Sensor Interface，camera sensor 的接口。
- V4L2：Video for linux 2，Linux 内核中关于视频设备的内核驱动框架。
- V4l2_subdev：V4L2 子设备，在 VIN 框架中 Camera 属于此类设备。

2.2 概念阐述

Camera Detector 主要负责探测当前机器上特定总线 (csi0 或者 csi1) 上的 camera ID，并由此来挂载正确的驱动。客户经常需要一个方案板上使用不同的 camera，同时又要求只使用一个固件来适配，由于这个需求经常被提到，因此需要一个合理有效的流程来做到 camera 的自动匹配。

另外由于每个 camera sensor 都相当于一颗 ic，控制这些 sensor 行为的命令都不一致，所需供电也有差别，甚至几乎每款 sensor 的上电和掉电时序都有不同要求，因此不可能使用同一个 power_on 和 power_off 来使需要检测的 camera 都能进入一个正确的可供 twi 读写的状态，这样就需要在检测每款 sensor 前，都需要执行了正确的上电时序 (power_on) 以及掉电时序 (power_off)，然后才可以读 sensor ID，并以此来检测当前是否加载了正确的 sensor 驱动。

目前 Camera Detector 的设计基于 V4L2 设备以及 V4l2_subdev 的挂载以及卸载流程之上，即在 V4L2 挂载 V4l2_subdev 之后做检测，如果检测失败则执行卸载流程，如果检测成功，则流程继续，这样做可以充分利用已经有的 sensor 驱动接口，例如每个 sensor 都有自己 power_on 和 power_off 操作，也有自己的 sensor_detect 操作。

3 模块流程设计

3.1 Camera Detector 流程图

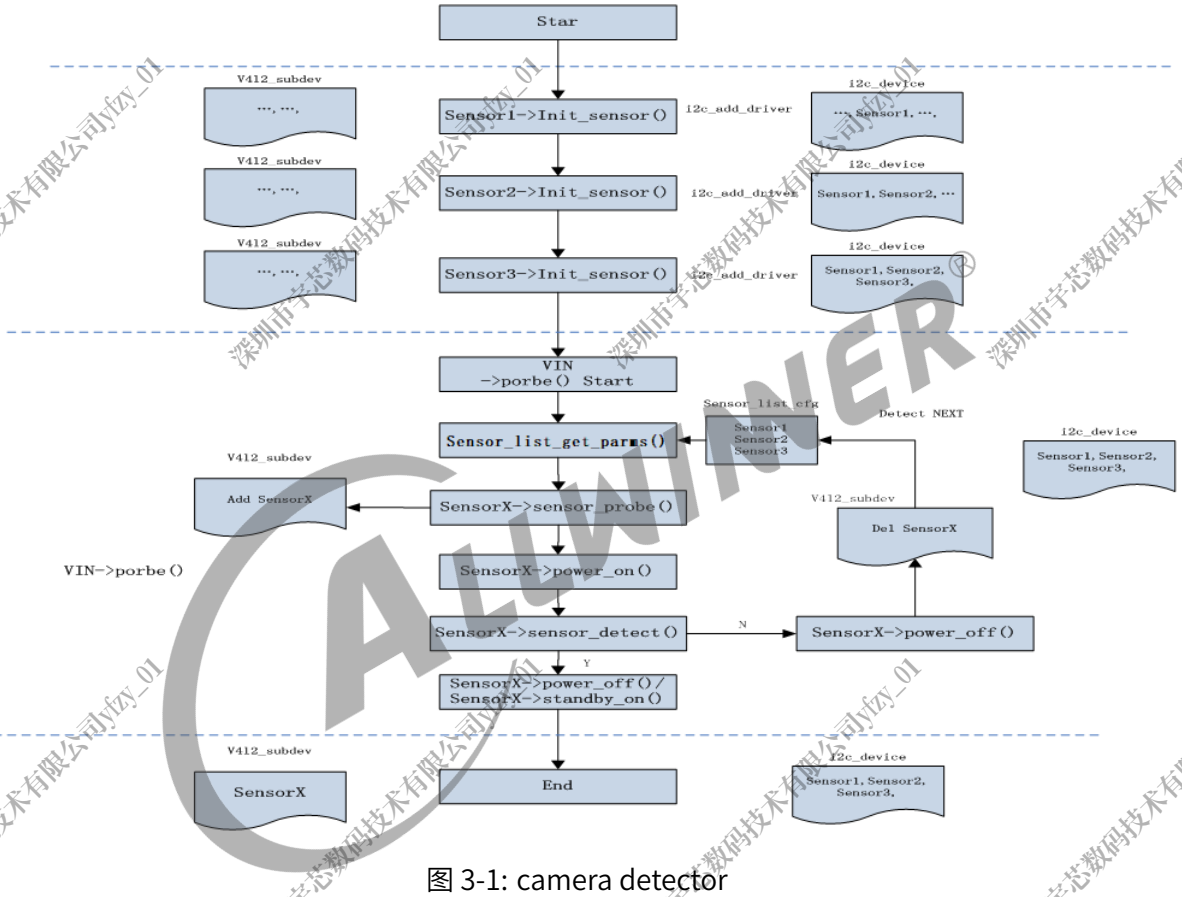

图 3-1: camera detector

3.2 Camera Detector 流程解析

Camera 自动检测主要分如下几步完成：

1. 系统启动加载各个有待检测的 camera sensor 驱动，此时会调用 twi_add_driver 函数，将该 sensor 驱动添加到 twi 设备列表中。如图 1 所示，在 insmod 时，每个驱动的 init_sensor 函数会被执行，则其主要作用就是将当前驱动添加到 twi driver 下，为之后挂载 subdev 做准备。
2. 当 insmod vin_v4l2.ko 时，会调用 vin 的 probe 函数，此时驱动调用 sensor_list_get_parms 获取 board.dts 中 sensor_list 节点的配置，其中包括，standby 类型，sensor 类型（YUV,RAW）以及所需的 hflip 设置和 vflip 设置，另外还包括 vcm 马达的配置。

3. 获得一个 sensor 的配置，然后执行 sensor_probe，此时会将该 sensor 作为 v4l2_subdev 挂载到 v4l2 设备上，这样 v4l2 设备便可以访问 v4l2_subdev 中的操作函数集。
4. 给 sensor 上电，执行 sensor->power_on() 操作，然后执行 sensor->detect() 操作，如果检测成功则执行 sensor->power_off() 或者 sensor->standby_on()（根据 sensor 设定的模式），并跳转第 5 步；如果检测失败则返回第 3 步继续执行。
5. 检测结束。

4 数据结构设计

4.1 Sensor config 数据结构

```
struct sensor_instance {
  char cam_name[TWI_NAME_SIZE];     /* sensor name */
  int cam_addr;            /* sensor twi addr */
  int cam_type;            /* sensor type */
  int is_isp_used;         /* 0:not use isp 1:use isp  */
  int is_bayer_raw;        /* 0:yuv 1:bayer raw rgb */
  int vflip;           /* flip in vertical direction 0:disable 1:enable */
  int hflip;           /* flip in horizontal direction 0:disable 1:enable */
  int act_addr;            /* actuator twi addr */
  char act_name[TWI_NAME_SIZE];     /* actuator name */
  char isp_cfg_name[TWI_NAME_SIZE];   /* isp config name */
};

struct sensor_list {
  int use_sensor_list;           /* use sensor_list function flag */
  int used;              /* use rear sensor or front sensor */
  int csi_sel;               /* parser channel ID, must be configured and valid */
  int device_sel;              /* device id of the sensor ：0：rear sensor 1：front sensor;(can continue to increase)*/
  int mclk_id;             /* ID of the mclk used by the sensor */
  int sensor_bus_sel;          /* ID of the twi used by the sensor */
  int sensor_bus_type;           /* communication mode used by the sensor 0: twi 1:cci 2:spi */
  int act_bus_sel;           /* ID of the twi used by the actuator */
  int act_bus_type;            /* communication mode used by the actuator 0: twi 1:cci 2:spi */
  int act_separate;            /* parameter is no longer used */
  int power_set;           /* no need to pay attention to the use of fpga */
  int detect_num;              /* number of adaptive sensors required */
  char sensor_pos[32];         /* sensor position : "rear" or "front" */
  int valid_idx;             /* valid sensor flag 0: valid -1:no valid */
  struct vin_power power[ENUM_MAX_REGU];    /* no need to pay attention to the use of fpga */
  struct gpio_config gpio[MAX_GPIO_NUM];     /* save gpio config */
  struct sensor_instance inst[MAX_DETECT_NUM];   /* save sensor_instance */
};
```

4.2 V4l2_subdev_ops 数据结构

```
static const struct v4l2_ctrl_ops sensor_ctrl_ops = {
  .g_volatile_ctrl = sensor_g_ctrl,   /* value Get a new value for this control*/
  .s_ctrl = sensor_s_ctrl,      /* Actually set the new control */
};

static const struct v4l2_subdev_core_ops sensor_core_ops = {
  .reset = sensor_reset,      /* generic reset command. The argument selects which subsystems to reset. */
  .init = sensor_init,      /* initialize the sensor registers to some sort of reasonable default values */
```

```
    .s_power = sensor_power,    /* puts subdevice in power saving mode (on == 0) or normal operation mode (on == 1). */
    .ioctl = sensor_ioctl,      /* called at the end of ioctl() syscall handler at the V4L2 core. */
};

static const struct v4l2_subdev_video_ops sensor_video_ops = {
    .g_mbus_config = sensor_g_mbus_config,      /* get the media bus configuration of a remote sub-device. */
};

static const struct v4l2_subdev_pad_ops sensor_pad_ops = {
    .enum_mbus_code = sensor_enum_code,      /* callback for VIDIOC_SUBDEV_ENUM_MBUS_CODE() ioctl handler code
        . */
    .enum_frame_size = sensor_enum_frame_size,    /* callback for VIDIOC_SUBDEV_ENUM_FRAME_SIZE() ioctl handler
        code. */
    .get_fmt = sensor_get_fmt,          /* callback for VIDIOC_SUBDEV_G_FMT() ioctl handler code. */
    .set_fmt = sensor_set_fmt,          /* callback for VIDIOC_SUBDEV_S_FMT() ioctl handler code. */
};
static const struct v4l2_subdev_ops sensor_ops = {
    .core = &sensor_core_ops,    /* pointer to &struct v4l2_subdev_core_ops. Can be %NULL */
    .video = &sensor_video_ops,    /* pointer to &struct v4l2_subdev_video_ops. Can be %NULL */
    .pad = &sensor_pad_ops,      /* pointer to &struct v4l2_subdev_pad_ops. Can be %NULL */
};
```

5 接口设计

5.1 内部接口设计

5.1.1 Sensor 驱动相关接口

```
static __init int init_sensor(void);
static int sensor_probe(struct twi_client *client, const struct twi_device_id *id);
static int sensor_power(struct v4l2_subdev *sd, int on);
static int sensor_detect(struct v4l2_subdev *sd);
```

5.1.2 VIN 驱动相关接口

```
int parse_modules_from_device_tree(struct vin_md *vind);
int sensor_list_get_parms(struct sensor_list *sensors, char *pos);
struct v4l2_subdev *v4l2_twi_new_subdev(struct v4l2_device *v4l2_dev,
    struct twi_adapter *adapter, const char *client_type,
    u8 addr, const unsigned short *probe_addrs);
struct v4l2_subdev *v4l2_twi_new_subdev_board(struct v4l2_device *v4l2_dev,
    struct twi_adapter *adapter, struct twi_board_info *info,
    const unsigned short *probe_addrs);
int __must_check v4l2_device_register_subdev(struct v4l2_device *v4l2_dev,
            struct v4l2_subdev *sd);
```

6 出错处理

如果当前 sensor 检测失败，会注销掉相关 sensor 在 twi 或者 subdev 上所占用的资源。

7 使用方法

方案支持自动检测 camera 的功能，该功能支持在同一个方案上采用不同的模组组合。如果需要使用该方案，需要在 board.dts 中作出相应的配置：

1. 设置相应 csi 上的相关选项如：vinc0_sensor_list = 1 表示后置摄像头开启自适应功能，vinc1_sensor_list = 1 表示前置摄像头开启自适应功能；

```
vinc0:vinc@2009000 {
        vinc0_csi_sel = <0>;
        vinc0_mipi_sel = <0>;
        vinc0_isp_sel = <0>;
        vinc0_isp_tx_ch = <0>;
        vinc0_tdm_rx_sel = <0xff>;
        vinc0_rear_sensor_sel = <0>;
        vinc0_front_sensor_sel = <1>;
        vinc0_sensor_list = <1>;
        status = "okay";
};
vinc1:vinc@2009200 {
        vinc1_csi_sel = <0>;
        vinc1_mipi_sel = <0>;
        vinc1_isp_sel = <0>;
        vinc1_isp_tx_ch = <0>;
        vinc1_tdm_rx_sel = <0xff>;
        vinc1_rear_sensor_sel = <0>;
        vinc1_front_sensor_sel = <1>;
        vinc1_sensor_list = <0>;
        status = "okay";
};
```

图 7-1: board.dts 配置

2. 明确定义出前后摄像头，例如 sensor0_pos = “rear” ，sensor1_pos = “front”；

3. 内核配置中使能 sensor list。

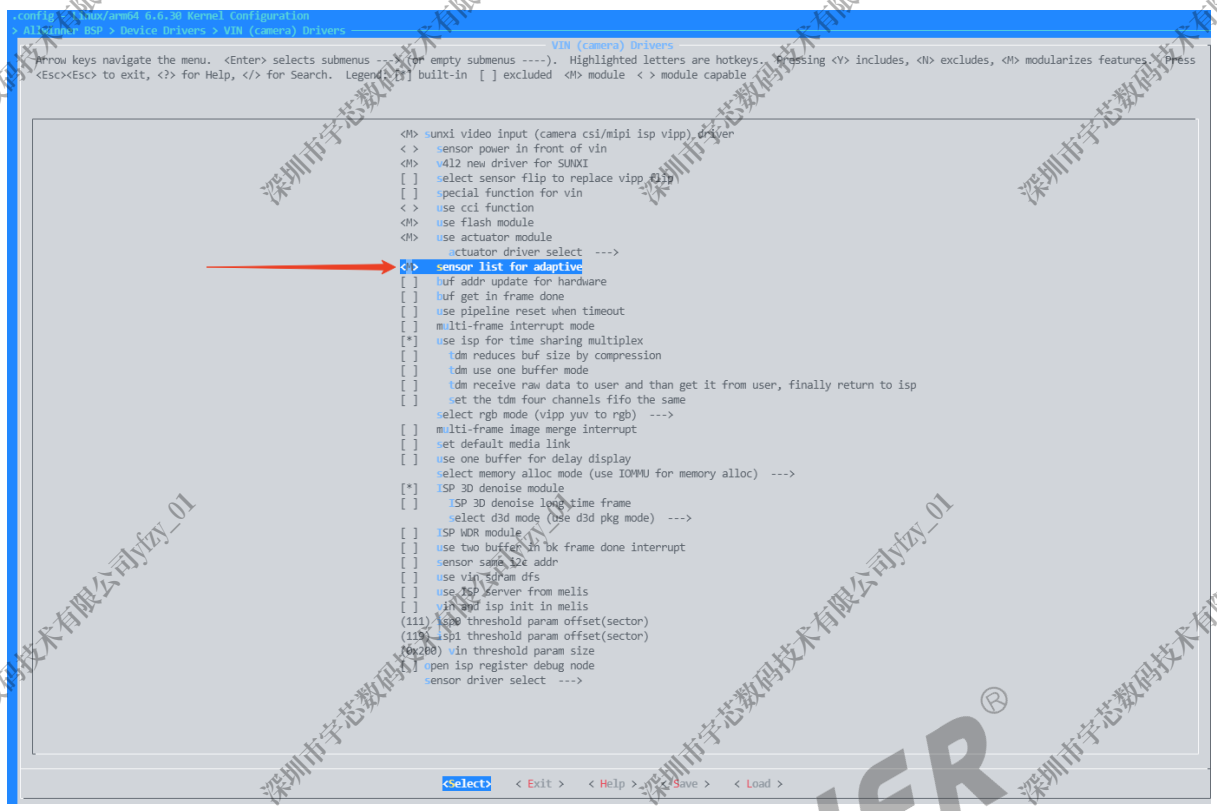

图 7-2: 内核配置配置

7.1 sensor list 配置说明

如果完成了前面的步骤，则驱动就会去试图读取 board.dts 文件定义的 sensor 信息。

sensor list 的配置在 board.dts，sensor_list 节点：

节点配置说明如下：

```
sensor_listX:sensor_list@200b820 {
  sensorX0_mname = "";      /* sensor name of the first sensor  */
  sensorX0_twi_addr = <>;     /* twi addr of the first sensor */
  sensorX0_type = <>;     /* sensor type of the first sensor */
    sensorX0_hflip =  <>;      /* flip in horizontal direction of the first sensor */
    sensorX0_vflip = <>;      /* flip in vertical direction of the first sensor */
    sensorX0_act_used = <>;      /* act used or not of the first sensor 0: not use; 1: use*/
    sensorX0_act_name = "";      /* act name of the first sensor */
    sensorX0_act_twi_addr = <>;   /* act twi addr of the first sensor */
    sensorX1_mname = "";      /* sensor name of the second sensor  */
    sensorX1_twi_addr = <>;      /* twi addr of the second sensor */
    sensorX1_type = <>;      /* sensor type of the second sensor */
    sensorX1_hflip =  <>;      /* flip in horizontal direction of the second sensor */
    sensorX1_vflip = <>;      /* flip in vertical direction of the second sensor */
    sensorX1_act_used = <>;      /* act used or not of the second sensor 0: not use; 1: use*/
    sensorX1_act_name = "";      /* act name of the second sensor */
    sensorX1_act_twi_addr = <>;   /* act twi addr of the second sensor */
    sensorX2_mname = "";      /* sensor name of the third sensor  */
    sensorX2_twi_addr = <>;      /* twi addr of the third sensor */
```

```
    sensorX2_type = <>;      /* sensor type of the third sensor */
    sensorX2_hflip =  <>;      /* flip in horizontal direction of the third sensor */
    sensorX2_vflip = <>;      /* flip in vertical direction of the third sensor */
    sensorX2_act_used = <>;      /* act used or not of the third sensor 0: not use; 1: use*/
    sensorX2_act_name = "";      /* act name of the third sensor */
    sensorX2_act_twi_addr = <>;   /* act twi addr of the third sensor */
    status= "";         /* open or close sensor_list device */
};
```

注意：

1. sensor_listX 中使用的 sensor 模组的供电配置使用 snesorX 节点中供电配置。
2. 以上说明适用于前后摄，每个位置最多可以配置 3 个 sensor 自适应，这 3 个（或者更少）里面必须包含 dts 定义的对应位置的 sensor。
3. 每个 sensor_list 节点的 twi sel 在对应的 sensor 节点中配置，列如：sensor_list0 节点中所用的三个 sensor 所使用的 twi id 都是 sensor0 节点中 sensor0_twi_cci_id 属性的值。

如果前后摄需要自适应同一个 sensor（如 gc5025+gc2385 模组和 gc2385+gc030a 模组）的情况，那么前后摄的 sensor name 不能一样。下面以需要自适应 gc5025+gc2385 模组和 gc2385+gc030a 模组这个两个模组为例，配置参考如下：

```
sensor_list0:sensor_list@200b820 {
   sensor00_mname = "gc5025_mipi";
   sensor00_twi_addr = <0x6e>;
   sensor00_type = <1>;
   sensor00_hflip =  <0>;
   sensor00_vflip = <0>;
   sensor00_act_used = <1>;
   sensor00_act_name = "dw9714_act";
   sensor00_act_twi_addr = <0x18>;
   sensor01_mname = "gc2385_mipi";
   sensor01_twi_addr = <0x6c>;
   sensor01_type = <1>;
   sensor01_hflip =  <0>;
   sensor01_vflip = <0>;
   sensor01_act_used = <0>;
   sensor01_act_name = "";
   sensor01_act_twi_addr = <>;
   sensor02_mname = "gc5035_mipi";
   sensor02_twi_addr = <0x6a>;
   sensor02_type = <1>;
   sensor02_hflip =  <0>;
   sensor02_vflip = <0>;
   sensor02_act_used = <0>;
    sensor02_act_name = "";
   sensor02_act_twi_addr = <>;
   status= "okay";
};

sensor_list0:sensor_list@200b820 {
   sensor00_mname = "gc2385_mipi_2";
   sensor00_twi_addr = <0x69>;
   sensor00_type = <1>;
     sensor00_hflip =  <0>;
     sensor00_vflip = <0>;
     sensor00_act_used = <0>;
```

```
    sensor00_act_name = "";
    sensor00_act_twi_addr = <>;
    sensor01_mname = "gc030a_mipi";
    sensor01_twi_addr = <0x42>;
    sensor01_type = <1>;
    sensor01_hflip =  <0>;
    sensor01_vflip = <0>;
    sensor01_act_used = <0>;
    sensor01_act_name = "";
    sensor01_act_twi_addr = <>;
    sensor02_mname = "c2590_mipi";
    sensor02_twi_addr = <0x6c>;
    sensor02_type = <1>;
    sensor02_hflip =  <0>;
    sensor02_vflip = <0>;
    sensor02_act_used = <0>;
    sensor02_act_name = "";
    sensor02_act_twi_addr = <>;
    status= "okay";
};
```

7.2 camera.cfg 命名方式

在 camera.cfg 有对闪光灯，聚焦，变焦功能等配置, 如需自动化识别 sensor，需要将命名为 camera_[sensor0_name]_[sensor1_name].cfg, 如 camera_gc5025_mipi_gc2385_mipi.cfg, HAL 会自动名字并加载。

7.3 sensor_name 命名方式

mipi sensor 的 sensor_name 为:sensor 型号 _mipi, 例如：gc5025 模组的的 sensor_name 为 gc5025_mipi；并口 sensor 的 sensor_name 为 sensor 型号，例如：ar0238 模组的 sensor_name 为 ar0238。开启 sensor_list 模式时，前后摄需要自适应同一个 sensor（如 gc5025+gc2385 模组和 gc2385+gc030a 模组）的情况，那么前后摄的 sensor name 不能一样，一般后摄的 sensor_name 写为 sensor 型号 _mipi，前摄的 sensor_name 写为 sensor 型号 _mipi_2，例如 gc2385 的 sensor_name 如下:

```
#define SENSOR_NUM   0x2
#define SENSOR_NAME   "gc2385_mipi"
#define SENSOR_NAME_2   "gc2385_mipi_2"
```

著作权声明

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。