全 志 科 技
Allwinner Technology

A20 平台 CTP 与 SENSOR 自动检测使用文档

V1.0

2013-06-17

Revision History

Version	Date	Changes compared to previous issue
V1.0	2013-06-17	初始版本

目录

1. 前言

1.1 编写目的

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

1.2 适用范围

适用于 A20 相对应平台。

1.3 相关人员

使用自动加载功能以及相关的开发与维护人员应该仔细阅读本文档。

1.4 文档介绍

本文档第二章主要说明自动检测模块的功能以及实现原理，需要注意的事项；第三章主要说明自动检测功能的使用步骤；第四章说明如果添加一个没有支持的设备到自动检测中来。

1.5 说明

本文主要说明 A20 的../drivers/input/sw_device.c，并不断进行更新补充中，由于文档不断补充，代码也不断更新，有些地方可能和实际代码中有细微差别，请注意。

2. 模块介绍

2.1 模块功能介绍

自动检测模块主要针对使用 I2C 总线的设备，根据 i2c 地址的扫描情况，查看 chip id 值（设备特有属性）进而可以识别当前使用的模组。将检测到的模组写入 device.info 文件中，在应用中进行驱动的加载。

该模块的使用，主要针对当更换同类设备时，不需要进行新的配置以及固件的制作。提高了设备的兼容性以及易操作性。

2.2 模块使用注意事项

（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，在想要剔除的设备的名称后写 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。

（4）设备检测

设备检测将按照 sw_device.c 中的 sw_device_info 结构体定义的数组进行顺序的检测，该结构体中的顺序与 sysconfig.fex 中 XXX_detect_list 列表中的顺序一致，当检测到没有 chip ID 的设备通信成功时，将查找具有相同地址的可以读取 chip id 的设备，如果 chip id 值匹配，则说明该设备为存在 chip id 值，若找不到 chip id 值匹配的设备，则说明该设备为没有 chip id 值。因此当设备列表中存在两个以上的设备地址冲突且其中两个或以上都没有 chip id 值时，只能留可能使用的一个没有 chip id 值的设备，不然将可能造成检测

错误的情况。

2.3 模块源码结构介绍

自动检测源码目录：\lichee\linux-3.x\drivers\input\sw_device.c

2.4 支持自动检测设备介绍

（1）Gsensor 设备

Gsensor 设备支持的模组如下所示。

表 1 Gsensor 支持列表

支持的模组	Chip ID 寄存器	Chip ID 值	I2C 地址	KO 文件名称
bma250	0x00	3	0x18 , 0x19	bma250.ko
bma250e			0x18 , 0x19	
bma222			0x08	
bma150		2	0x38	
mma8452	0x0d	0x2a	SA0 = 0: 0x1c	mma8452.ko
			SA0 = 1: 0x1d	
mma7660	无	无	0x4c	mma7660.ko
mma8652	0x0d	0x4A	0x1d	mma865x.ko
mma8653		0x5A		
stk8312	0x12	0x28	0x3d	stk831x.ko
stk8313		0x3c	0x22	
afa750	0x37	0x3d , 0x3c	0x3d	afa750
lis3de_acc	0x0f	0x33	0x28 , 0x29	lis3de_acc.ko
lis3dh_acc	0x0f	0x33	0x18 , 0x19	lis3dh_acc.ko
kxtik-1004	0x0f	0x05	0x0f	kxtik.ko
kxtj9-1005		0x08		
dmard10	无	无	0x18	Dmard10.ko
dmard06	0x0f	0x06	0x1c	dmard06.ko
mxc6225	无	无	0x15	mxc622x
fxos8700	0x0d	0xc7	0x1c , 0x1d 0x1e , 0x1f	fxos8700.ko
lsm303d	0x0f	0x49	0x1e,0x1d	lsm303d.ko

（2）CTP 设备

自动检测中支持的 CTP 设备列表如下所示。

表 2 ctp 支持列表

支持的模组		Chip ID 寄存器	Chip ID 值	I2C 地址	KO 文件名称
FT 系列	ft5306	0xa3	0x55	0x38	ft5x_ts.ko

	ft5406				
	ft5606		0x08		
	ft5506				
GT 系列	gt811	0x715	0x11	0x5d	gt811.ko
	gt813	0xf7d	0x13	0x5d	gt82x.ko
	gt827		0x27		
	gt828		0x28		
	gt9xx	0x8140	0x39	0x14, 0x5d	gt9xx_ts.ko
GLS 系列	gsl1680	无	无	0x40	gslX680.ko
	gsl2680				
	gsl3680				
ZET	Zet622x	无	无	0x76	zet622x.ko

2.5 模块配置介绍

2.5.1 sys_config.fex 配置

配置文件的位置：\lichee\tools\pack\chips\sun7i\configs\android\wing-XXX 目录下。

(1) Gsensor 配置

sys_config.fex 文件中关系到自动 gsensor 自动检测功能的为 “gsensor_para”以及“gsensor_list_para”两块参数。如下所示：

```
;------------------------------------------------------------------------------
--
; G sensor configuration
; gs_twi_id --- TWI ID for controlling Gsensor (0: TWI0, 1: TWI1, 2: TWI2)
;------------------------------------------------------------------------------
--
[gsensor_para]
gsensor_used    = 1 //标示使用 gsensor
gsensor_twi_id   = 1 //使用哪组 i2c
gsensor_twi_addr =
gsensor_int1    =
gsensor_int2        =
```

如果gsensor_used 设置为0 则将退出gsensor的自动检测。

```
;-------------------------------------------------------------------------
--
; G sensor automatic detection configuration
;gsensor_detect_used    ---  Whether  startup  automatic  inspection  function.
1:used,0:unused
;Module name postposition 1 said detection, 0 means no detection.
;-------------------------------------------------------------------------
--
[gsensor_list_para]
gsensor_det_used        = 1 //设置为 1 时，启动自动检测，设置为 0 时，退出自动检测。
bma250              = 1 //设置为 1，该模块支持的 I2C 地址添加到扫描列表
mma8452             = 1
mma7660             = 1
mma865x             = 1
afa750              = 1
lis3de_acc          = 1
lis3dh_acc          = 1
Kxtik               = 1
dmard10             = 0 //设置为 0，该模块支持的 I2C 地址从扫描列表中剔除
dmard06             = 1
mxc622x             = 1
fxos8700                = 1
lsm303d             = 1
```

当gsensor_det_used 设置为1时，启用自动检测，将设置为0时，退出自动检测。模块的名称后面写 1表示添加到自动检测扫描列表，写0表示剔除自动检测扫描列表。

gsensor_list_para列表中的名称顺序必须与 sw_device.c中sensors的名称顺序一一对应。

(2) CTP 配置

sys_config.fex 文件中关系到自动 ctp 自动检测功能的为 “ctp_para”以及“ctp_list_para”两块参数。如下所示：

```
[ctp_para]
ctp_used            = 1  //标示使用 CTP
ctp_twi_id      = 2  //使用哪组 i2c
```

```
........
;-------------------------------------------------------------------------------
--
```

如果ctp_used 设置为0 则将退出gsensor的自动检测。

```
;-------------------------------------------------------------------------------
--
; CTP automatic detection configuration
;ctp_detect_used      ---   Whether   startup   automatic   inspection   function.
1:used,0:unused
;Module name postposition 1 said detection, 0 means no detection.
;-------------------------------------------------------------------------------
--
[ctp_list_para]
ctp_det_used              = 1 //设置为 1 时，启动自动检测，设置为 0 时，退出自动检测。
ft5x_ts                   = 1 //设置为 1，该模块支持的 I2C 地址添加到扫描列表
gt82x                = 0 //设置为 0，该模块支持的 I2C 地址从扫描列表中剔除
gslX680              = 1
gt9xx_ts             = 1
gt811                = 1
zet622x              =1
```

ctp_list_para列表中的名称顺序必须与 sw_device.c中ctps的名称顺序一一对应。

当ctp_det_used 设置为1时，启用自动检测，将设置为0时，退出自动检测。模块的名称后面写 1表示添加到自动检测扫描列表，写0表示剔除自动检测扫描列表。

2.5.2 menuconfig 的配置

在编译服务器上，目录为\lichee\linux-3.3 上，输入命令：

```
make ARCH=arm menuconfig
```

如下所示：

```
/lichee/linux-3.3$ make ARCH=arm menuconfig
```

进入目录 Device Drivers\Input device support\目录下可以看到 device 模块是编译

为模块、编译进内核、不编译。device 驱动默认编译为模块。

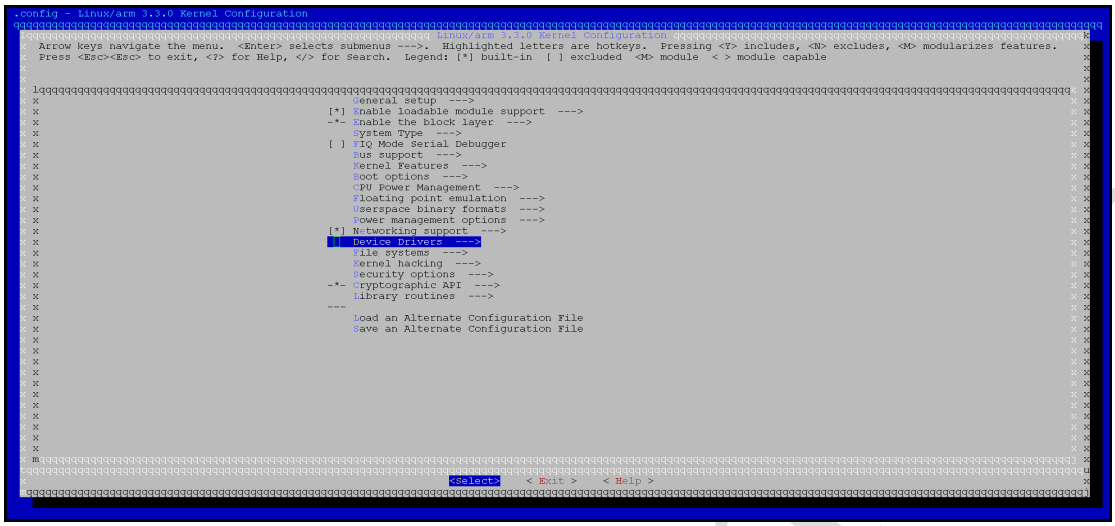

图 1　Device Drivers 选项配置

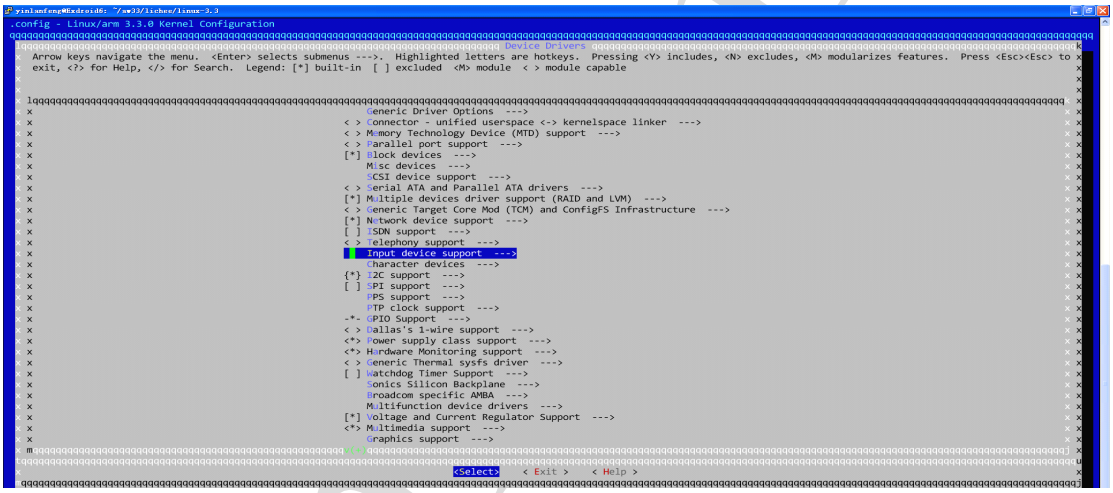

图 2　Input device support 选项配置

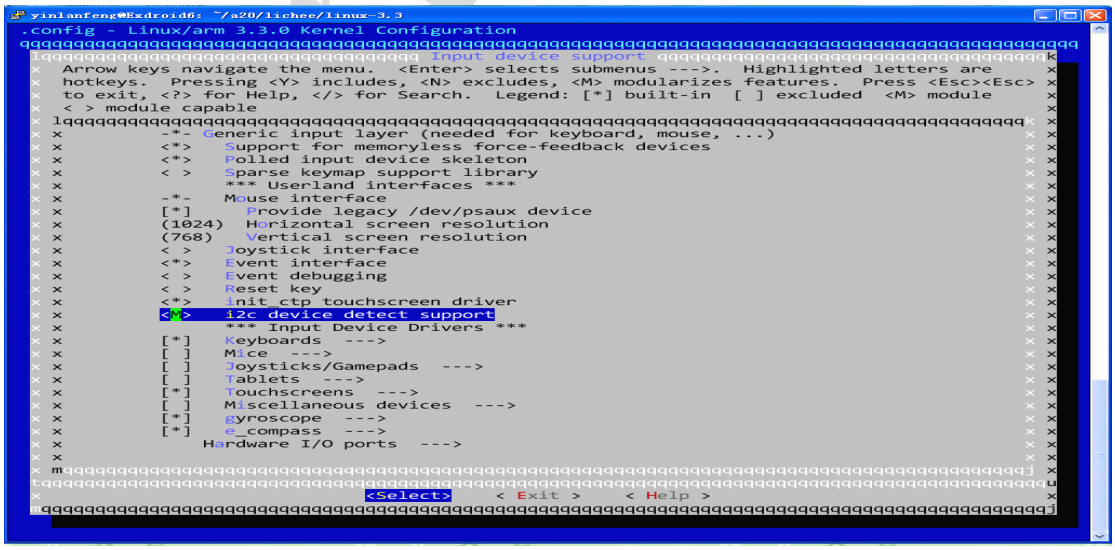

图 3 device 配置为模块

2.6 device.info 文件

Device.info 文件的作用是记录下当前使用的设备，自动检测功能模块将读取其中的设备驱动名称，进行检测，如果检测失败将会去扫描支持列表，查找使用的设备,也就是只有当更换设备时才会去扫描设备支持列表。否则将只检测使用的设备。上层应用中根据其中的驱动名称进行相应设备驱动的加载。

使用时，请不要随意修改文件的格式。文件的格式如下：

```
;Behind the equals sign said detected equipment corresponding to the name of the driver
;Note: don't change the file format!
gsensor_module_name = "mma7660"
ctp_module_name = "ft5x_ts"
```

文件使用关键字 =“驱动名称” 的方式进行命名。注意不要随便修改文件中的关键字,引号中的驱动名称可以根据方案中使用的设备进行相应的修改，注意，驱动的名称是用双引号引起来的。

若检测不到该文件，将自动生成该文件，且从扫描列表中找到当前使用的设备。

Device.info 文件存在 data 目录下，即：/data/device/info

3. 自动检测功能使用步骤

3.1 Gsensor 使用步骤

（1）支持的模组

bma250，bma250e，bma222，bma150（共用驱动 bma250.ko）；mma8452；mma7660；mm8652，mma8653（共用驱动 mma865x.ko）；afa750；lis3de_acc;lis3dh_acc;dmard06；dmard10；kxtik-1004、kxtj9-1005（共用驱动 kxtik.ko）；mxc6225;fxos8700;lsm303d。

驱动源文件目录：mma7660，mma8452，mma865x，afa750，kxtik 驱动源文件存放在\lichee\linux-3.0\drivers\hwmon 中；bma250，lis3de_acc,lis3dh_acc 驱动源文件存放在目录： \lichee\linux-3.0\drivers\gsensor 。

其中 fxos8700 与 lsm303d 为二合一设备，为磁力计与重力感应，目前该设备只支持检测功能，使用时，还需要根据具体的版型进行相关的调试工作，驱动源文件存放在：

\lichee\linux-3.0\drivers\input/e_compass

（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\...\system\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：/system/vendor/modules

（3）sys_config.fex 文件的修改

若 sys_config.fex 文件中不存在 gsensor_list_para 配置项时，请添加该配置项，且将不使用的存在冲突的设备设置为 0，即将其剔除出扫描列表。如将 dmard10 剔除掉。

```
[gsensor_list_para]
gsensor_det_used        = 1
bma250                  = 1
mma8452                 = 1
mma7660                 = 1
mma865x                 = 1
afa750                  = 1
lis3de_acc              = 1
lis3dh_acc              = 1
kxtik                   = 1
dmard10                 = 0///剔除该设备
dmard06                 = 1
mxc622x                 = 1
fxos8700                = 1
lsm303d                 = 1
```

（4）驱动的加载

使用自动检测功能时，只需要加载 sw_device.ko 即可。

在 android4.X\device\softwinner\wing-XXX\init.XXX.rc 中添加驱动加载的模块，为了快速的检测到设备，此语句应该放置在模块加载的最前面，如下所示：

```
…… ……
on boot
 #insmod device driver
        insmod /system /vendor/modules/sw _device.ko

#insmod mali driver
        insmod /system /vendor/modules/ump.ko
        insmod /system /vendor/modules/mali.ko

 #insmod video driver
        insmod /system /vendor/modules/cedarx.ko

 #csi module
        insmod /system /vendor/modules/videobuf-core.ko
        insmod /system /vendor/modules/videobuf-dma-contig.ko
…… ……
```

如果 CTP 使用而 GSENSOR 不想使用时，只需要修改 sys_config.fex 中 的“gsensor_list_para”主键下的 “gsensor_det_used”子键值，将其设置为 0 即可，而不需要将加载语句删除掉。

（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\android4.X\device\softwinner\wing-XXX，每个设备都有五项值，如下所示：

gsensor_name	Gsensor 名称，必须与驱动中设备名相同
gsensor_direct_x	Gsensor x 轴的方向，当定义成 true 时，x 轴取正值，当定义为 false 时，x 轴取负值
gsensor_direct_y	Gsensor y 轴的方向，当定义成 true 时，y 轴取正值，当定义为 false 时，y 轴取负值
gsensor_direct_z	Gsensor z 轴的方向，当定义成 true 时，z 轴取正值，当定义为 false 时，z 轴取负值

gsensor_xy_revert	XY 轴对调，当设为 TRUE 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° 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° 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（gsnesor_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 gsnesor.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。

3.2 CTP 使用步骤

（1）自动检测支持的模组

FT 系列：ft5306，ft5406，ft5606，ft5506(共用一个驱动，ft5x_ts.ko)；

Goodix 系列：gt813,gt827,gt828(gt82x.ko)；gt811(gt811.ko),gt9 系列(gt9xx_ts.ko)

GSL 系列：gsl1680,gsl2680,gsl3680(gslX680.ko

zet 系列：zet622x(zet622x.ko)

驱动源文件目录：\lichee\linux-3.x\drivers\input\sw_touchscreen

（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\...\system\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：/system/vendor/modules

（3）sys_config.fex 文件的修改

若 sys_config.fex 文件中不存在 ctp_list_para 配置项时，请添加该配置项，且将不使用的存在冲突的设备设置为 0，即将其剔除出扫描列表。已支持的设备不存在冲突情况，但是为了减少检测时间，也可以将不使用的模组剔除出扫描列表。将 gt82x 系列剔除出扫描列表，如下所示。

```
[ctp_list_para]
ctp_det_used     = 1
ft5x_ts          = 1
gt82x            =0//剔除该设备
gslX680          = 1
```

（4）驱动的加载

使用自动检测功能时，只需要加载 sw_device.ko 即可。

在 android4.X\device\softwinner\wing-XXX\init.XXX.rc 中添加驱动加载的模块，为了快速的检测到设备，此语句应该放置在模块加载的最前面，如下所示：

```
…… ……
on boot
 #insmod device driver
        insmod /system/vendor/modules/sw_device.ko

#insmod mali driver
        insmod /system/vendor/modules/ump.ko
        insmod /system/vendor/modules/mali.ko

 #insmod video driver
        insmod /system/vendor/modules/cedarx.ko

 #csi module
        insmod /system/vendor/modules/videobuf-core.ko
        insmod /system/vendor/modules/videobuf-dma-contig.ko
…… ……
```

如果 Gsensor 使用而 CTP 不想使用时，只需要修改 sys_config.fex 中 的“ctp_list_para”主键下的 “ctp_det_used”子键值，将其设置为 0 即可，而不需要将加载语句删除掉。

（5）IDC 文件

使用 adb shell getevent 命令，或者设备的名称为“gslX680”，“gt82x”，“ft5x_ts”，“sunxi-ts”，“gt818_ts”，“gt811_ts”，“gt9xx”，“sw-ts”时，使用的 idc 名字均为 tp.idc。idc 文件放置的目录为:system/usr/idc，则在配置文件为 wing_xx.mk 拷贝语句如下所示：

```
PRODUCT_COPY_FILES += \
device/softwinner/wing-xxx/sw-keyboard.kl:system/usr/keylayout/sw-keyboard.kl \
    device/softwinner/wing-xxx/tp.idc:system/usr/idc/tp.idc \
```

4. 如何添加一款新的设备

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。

4.1. 增加设备驱动

详细的移植说明：Ctp 驱动的移植可以参照 ctp 使用文档，gsensor 驱动的移植可以参照 Gsensor 驱动开发说明文档。

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。

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[2] = {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,
};
```

4.1.2 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 gsl_ts_init(void)
{
…… ……
        gsl_ts_driver.detect = ctp_detect;
        ret = i2c_add_driver(&gsl_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(&gsl_ts_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, i = 0 ,retry = 2;

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

        if (twi_id == adapter->nr) {
              while(retry--) {

              ret = i2c_smbus_read_byte_data(client,BMA250_CHIP_ID_REG);

              dprintk(DEBUG_INIT, "%s:addr = 0x%x, Read ID value is:%d\n",
                    __func__, client->addr, ret);

              while((chip_id_value[i++]) && (i < 5)){
                    dprintk(DEBUG_INIT, "chip:%d\n", chip_id_value[i - 1]);
                    if((ret & 0x00FF) == chip_id_value[i - 1]){
                          strlcpy(info->type, SENSOR_NAME, I2C_NAME_SIZE);
                          return 0;
                    }
              }
              msleep(1);
        }

                    dprintk(DEBUG_INIT, "%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 为例子，如下：

```
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{
             strlcpy(info->type, CTP_NAME, I2C_NAME_SIZE);
                return 0;
             }

        }else{
                return -ENODEV;
        }
}
```

4.2 sw_device.c 中的修改

sw_device.c 中的修改主要为增加设备的描述信息，将相关的描述信息增加到相应的 sw_device_info 中的变量中即可。

结构体 sw_device_info 用于存放设备的相关信息，该结构体的相关变量如下列表所示：

名称	用途
name	存放设备的驱动名称，如“bma250”，“gt82x”。
i2c_address	存放同一驱动下面支持的所有 i2c 地址。
chip_id_reg	存放设备的 chip id 寄存器地址
id_value	存放设备的 chip id 值
same_flag	写 1 表示有两个或者以上的设备地址相同，且其中一个以上没有 chip id

注意：当有两个或者以上的设备地址相同，且存在两个设备没有 chip id 值时，将按照顺序检测，返回第一个检测到的设备，可能造成错误，因此当存在两个设备没有 chip id 时，为了确保无误，请将不使用的设备剔除掉。

CTP 设备的 sw_device_info 结构体变量为 ctps，如果增加的设备为 ctp 设备，只需要按照 sw_device_info 中的格式，添加相关的信息放置到 ctps 变量中即可。

Gsensor 设备的 sw_device_info 结构体变量为 gsensors，如果增加的设备为 gsensor 设备，只需要按照 sw_device_info 中的格式，添加相关的信息放置到 gsensors 变量中即可。

以下针对 sw_device_info 中相关的设置进行相关的说明。

（1）进行 chip id 值检测

ctp 中需要增加 gt811 这个设备到自动检测中，编译之后的模块驱动名称为:“gt811”, I2C 地址可以设置为 0x5d；chip id 寄存器为 0x715；chip id 值为 0x11。

描述信息如下所示：

```
/* ctp info*/
static struct sw_device_info ctps[] = {
    { "ft5x_ts", {   0x38 },  0x03, {0x55,0x08,0x02,0x06,0x03}, 0},
    {  "gt82x", {   0x5d },  0xf7d, {0x13,0x27,0x28       }, 0},
    { "gslX680", {   0x40 },  0x00, {0x00              }, 0},
    {"gt9xx_ts", {0x14, 0x5d}, 0x8140, {0x39             }, 0},
    {  "gt811", {   0x5d },  0x715, {0x11              }, 0},
    { "zet622x", {   0x76 },  0x00, {0x00              }, 0},
};
```

（2）不进行 chip id 值检测

当设备没有 chip id 值时，可以不进行 chip id 的检测，此时 chip_id_reg 以及 id_value 都必须设置为 0x00，否者将可能造成错误。这些设备都只检测 i2c 通信而不进行 chip id 值的判断，当所支持的设备中，没有产生地址冲突的情况时，也可以将 chip_id_reg 以及 id_value 设置为 0x00 而不检测 chip id。

zet622x ctp 设备没有 chip id 如下所示：

```
/* ctp info*/
static struct sw_device_info ctps[] = {
    { "ft5x_ts", {   0x38 },  0x03, {0x55,0x08,0x02,0x06,0x03}, 0},
    {  "gt82x", {   0x5d },  0xf7d, {0x13,0x27,0x28       }, 0},
    { "gslX680", {   0x40 },  0x00, {0x00              }, 0},
    {"gt9xx_ts", {0x14, 0x5d}, 0x8140, {0x39             }, 0},
    {  "gt811", {   0x5d },  0x715, {0x11              }, 0},
    { "zet622x", {   0x76 },  0x00, {0x00              }, 0},
};
```

（3）same_flag

same_flag 为当支持的设备中有两个或者以上的设备地址相同，当有一个以上设备是没有 chip id 时，需要写 1 进行标识。当有两个设备以上地址相同且没有 chip id，则将按照顺序进行检测，且返回检测到的第一个设备。为避免错误，当两个以上设备地址相同且没有 chip id 值时，请将方案中不需要的设备剔除掉。剔除的方法为 sys_config.fex 中相应的模组名称后写 0。

4.3 sys_config.fex 的配置

当增加一个设备时，除了在 sw_device.c 中添加 sw_device_info 结构体设备的配置信息外，sys_config.fex 文件中也需要在设备的列表中增加该设备，增加检测的可选性。

ctp 中的设备列表为“[ctp_list_para]”，gsensor 中的设备列表为

“[gsensor_list_pata]”。如在 sw_device.c 中的 ctps 增加了“gt9xx”，在在“ctp_list_para”中需要增加相应的配置项，否则将默认添加到扫描列表中。

Xxx_list_para 中设备的名称需要跟 sw_device_info 中设备的名称顺序对应，否则将无法正确解析，默认将所有的都添加到扫描列表中。

如下 ctp 中增加 zet622x 设备：

```
[ctp_list_para]
ctp_det_used        = 1
ft5x_ts             = 1
gt82x               = 1
gslX680             = 1
gt9xx_ts            = 1
gt811               = 1
zet622x             =1
```

4.4 GSENSOR HAL 层增加

如果添加的设备为 Gsensor，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。

A20 的 sensor hal 层目录：

android4.x\device\softwinner\common\hardware\libsensors

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 文件中增加该结构体的相关内容即可。

Gsensor 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置在最前面，不使用的设备放后面。

sensor_extend_t 相关变量的说明如下表所示。

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
	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

4.5 TP idc 文件的添加

由于 android4.0 之后需要使用 idc 文件来识别是否为触摸屏，为了实现兼容性，可以将有可能使用的 tp 的 idc 文件都添加到固件中，另一种方法是所有的 tp 使用一个文件，但是在框架层需要做一些修改，当读取到的驱动的名字为需要实现兼容性的设备时，则匹配固定的文件。当不希望实现这样的固定匹配时，可以将其删除掉。

目前使用的方法为，tp 均使用 tp.idc 进行相应的匹配。

修改的位置：

Android ： 头文件：android4.X\frameworks\base\include\androidfw\InputDevice.h

源文件：\android4.X\frameworks\base\libs\androidfw\InputDevice.cpp

增加设备兼容性的模组时，只需要修改 InputDevice.h 文件即可。

如增加了“gt9XX”，结构体如下：

```
struct CtpName {
        inline CtpName():
            number (8),
            default_name (String8("ctp_")),
                ctp_name ({String8("gslX680"),String8("gt82x"),String8("ft5x_ts"),String8("sunxi-ts"),String8("gt818_ts"),
                String8("gt811_ts"),String8("sw_-ts"),String8("gt9xx")}){
                }
        int number;
        String8 default_name;
        String8 ctp_name[8];

};
```

增加设备时，需要做的事情：

1）增加模组的设备，即 number 数值加一，number 由 7 变为 8。

2）增加模组的名称，即增加 ctp_name 中的设备的名称，名称为注册的 input_dev 结构体名称或者使用 getevent 命令查看到的名称。

3）增加 ctp_name 的数值，应与 number 数值一致。