



Tina Linux USB 开发指南

**版本号: 1.4
发布日期: 2021.05.14**

版本历史

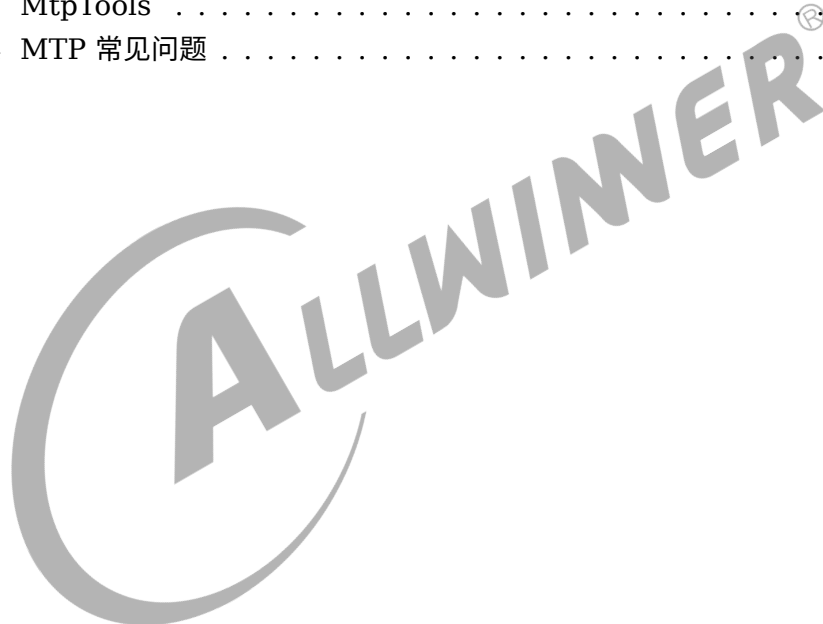
版本号	日期	制/修订人	内容描述
1.0	2020.08.10	AWA0985	新建初始版本
1.1	2020.09.21	AWA0985	新增 adb,mtp 相关章节
1.2	2020.12.25	AWA0985	新增 USB 功能裁剪说明
1.3	2021.04.10	AWA0985	增加 linux-5.4 相关说明
1.4	2021.05.14	AWA0985	增加 printer 功能配置说明



目 录

1 概述	1
1.1 编写目的	1
1.2 适用范围	1
1.3 相关人员	1
2 模块介绍	2
2.1 相关术语介绍	2
2.2 模块配置	2
2.2.1 Device Tree 配置 (Linux-4.9 内核)	2
2.2.2 Device Tree 配置 (Linux-5.4 内核)	4
2.2.3 内核配置	6
2.3 代码结构	7
3 常用功能配置	9
3.1 配置 OTG 功能	9
3.2 USB Gadget 功能配置	10
3.2.1 ADB 功能配置	10
3.2.2 MTP 功能配置	11
3.2.3 Mass Storage 功能配置	11
3.2.4 RNDIS 功能配置	12
3.2.5 UAC1 功能配置	13
3.2.6 HID 功能配置	14
3.2.7 printer 功能配置	14
3.3 修改 UDC Endpoint 配置	15
3.4 USB Host 功能配置	20
3.4.1 U 盘功能	20
3.4.2 USB 声卡	20
3.4.3 USB 网卡	21
3.4.4 USB HID 设备	21
3.4.5 USB 摄像头	21
3.4.6 USB 打印机	21
3.5 支持 USB Standby	21
3.6 设置 USB 序列号	22
3.7 测试 USB 速率	22
3.8 USB 功能裁剪	23
3.8.1 删除多余的 usb gadget 功能	23
3.8.2 仅用到 usb0 的 OTG 功能	23
3.8.3 仅使用到 adb 功能	24
4 常见调试方法	25
4.1 调试节点	25
4.2 测试 USB 眼图	25

5 USB 相关应用工具	27
5.1 setusbconfig 脚本	27
5.2 ADB	27
5.2.1 ADB 功能说明	28
5.2.1.1 adb shell 功能	28
5.2.1.2 adb push/pull 功能	28
5.2.1.3 adb 网络连接	28
5.2.1.4 adb forward 功能	28
5.2.1.5 adb auth 功能	29
5.2.2 ADB 常见问题	29
5.3 MTP	30
5.3.1 MTP 服务	31
5.3.2 MTP 相关配置	32
5.3.2.1 menuconfig 选中 MtpDaemon 软件包	32
5.3.2.2 配置挂载目录	32
5.3.3 MtpTools	32
5.3.4 MTP 常见问题	33



1 概述

1.1 编写目的

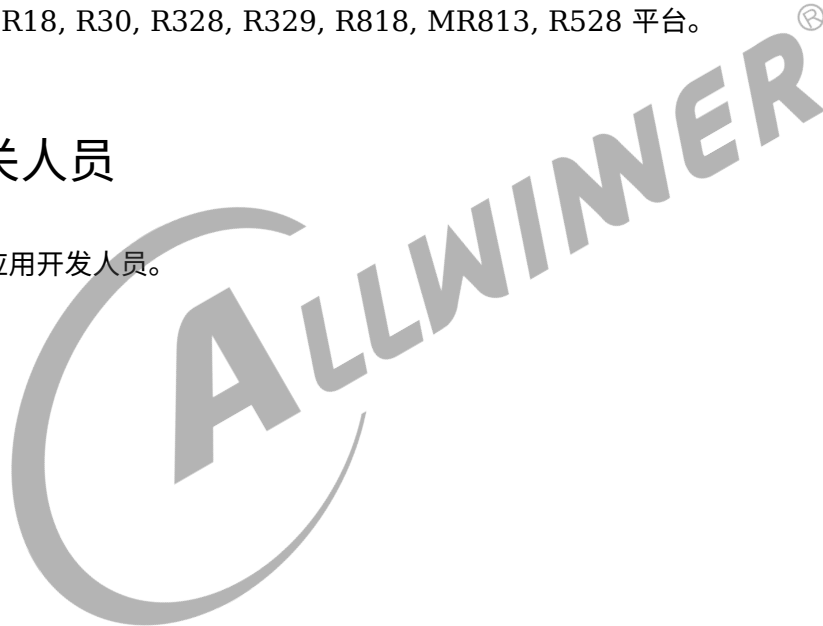
介绍全志 USB 模块的使用方法。

1.2 适用范围

该文档适用于 R18, R30, R328, R329, R818, MR813, R528 平台。

1.3 相关人员

USB 驱动和应用开发人员。



2 模块介绍

USB 有主机功能和从设备功能。做主机时，能连接 U 盘、USB 鼠标等 USB 设备；做从设备时，具有 ADB 调试等从设备功能。

芯片硬件上支持 USB device/Host(OHCI/EHCI)

2.1 相关术语介绍

术语	解释
USB	Universal Serial Bus, 通用串行总线
OTG	On-The-Go
ADB	Android Debug Bridge, Android 调试桥
HCD	Host Controller Driver, 主机控制器驱动
UDC	USB Device Controller, USB 设备控制器
HCI	Host Controller Interface, 主机控制器接口
EHCI	Enhanced Host Controller Interface, 增强型主机控制器接口
OHCI	Open Host Controller Interface, 开放式主机控制器接口

2.2 模块配置

2.2.1 Device Tree 配置 (Linux-4.9 内核)

USB0 配置:

```
usb0:usb0@0 {
    device_type = "usb0";
    usb_port_type = <0x2>;
    usb_detect_type = <0x1>;
    usb_id_gpio = <&pio PB 3 0 0 0xffffffff 0xffffffff>;
    usb_det_vbus_gpio = <&pio PB 3 0 0 0xffffffff 0xffffffff>;
    usb_regulator_io = "nocare";
    usb_wakeup_suspend = <0>;
    usb_serial_unique = <0>;
    usb_serial_number = "20080411";
    rndis_wceis = <1>;
    status = "okay";
}
```

};

参数	功能
usb_port_type	USB 模式, 0:device; 1:host; 2:OTG;
usb_detect_type	USB 检测类型, 0:none; 1:vbus/id detect; 2:id/dpdm; 一般只用 1
usb_detect_mode	USB ID 脚检测模式, 0: 线程轮询; 1: 中断检测; 默认为 0
usb_id_gpio	USB ID 脚配置
usb_det_vbus_gpio	USB VBUS 检测引脚配置; 一般使用对应 gpio 即可; 注意, 如果方案不需要检测 VBUS, 那么这里请填写 USB ID 脚, 如果方案由 PMU 检测 VBUS, 那么这里请填写 “axp_ctrl”
det_vbus_supply	与 usb_det_vbus_gpio 类似, 用于检测 VBUS 是否存在。它会指向 usb_power_supply 节点。
usb_regulator_io	IO 的供电, 如果没有则 “nocare”
usb_wakeup_suspend	是否支持 USB 唤醒; 如果需要支持进入深度休眠后的 USB 唤醒, 则置为 1
usb_serial_unique	是否使用唯一序列号 (由芯片 chipid 得到)
usb_serial_number	如果 usb_serial_unique 为 0, 那么使用这里的配置的值作为序列号
status	是否启用该模块。“okay”: 启用; “disabled”: 关闭

det_vbus_supply 存在的作用:

1. 如果 usb_det_vbus_gpio 设置的是 axp_ctrl, 那么这里用于获取 power_supply 来检测 vbus 是否存在;
2. 卸载 device 驱动时, 需要获取 power_supply, 然后设置限流值;

有些时候, 还需要对udc进行配置:

```
udc:udc-controller@0x05100000 {
    det_vbus_supply = <&usb_power_supply>;
};
```

这里det_vbus_supply同样指向usb_power_supply节点, 这里的目的是让udc获取到usb power, 并设置限流值。udc驱动中会检测到USB接入的是PC, 然后软件设置USB限流值。

如果PMU支持并开启了BC1.2协议(识别USB的类型并且硬件上自动设置限流值), 那么这里就不需要配置。

无论是 USB0 还是 USB1, EHCI/OHCI 的配置都是类似的, 具体 EHCI/OHCI 配置如下:

```
ehci0:ehci0-controller@0x04101000 {
    drvvbus-supply = <&reg_drivevbus>;
};

ohci0:ohci0-controller@0x04101400 {
    drvvbus-supply = <&reg_drivevbus>;
};
```

这里 reg_drivevbus 主要配置对应的 regulator 作为 OTG_5V 供电。常见的配置有：

```

使用PMU自带的DRVVBUS引脚使能对应的5V输出：
regulator0: regulators@0 {
    reg_drivevbus: drivevbus {
        regulator-name = "axp2585-drivevbus";
    };
};

使用GPIO去控制5V输出：
reg_usb0_vbus: usb0-vbus {
    compatible = "regulator-fixed";
    gpio = <&pio PH 12 1 2 0 0>;
    regulator-name = "usb0-vbus";
    regulator-min-microvolt = <5000000>;
    regulator-max-microvolt = <5000000>;
    enable-active-high;
};

```

USB1 配置：

```

usbc1:usbc1@0 {
    device_type = "usbc1";
    usb_regulator_io = "nocare";
    usb_wakeup_suspend = <0>;
    status = "okay";
};

```

配置与 USB0 类似，但少了 UDC 相关部分：

参数	功能
usb_regulator_io	IO 的供电，如果没有则 “nocare”
usb_wakeup_suspend	是否支持 USB 唤醒；如果需要支持进入深度休眠后的 USB 唤醒，则置为 1
status	是否启用该模块。“okay”：启用；“disabled”：关闭

2.2.2 Device Tree 配置 (Linux-5.4 内核)

5.4 内核下，USB 的 DTS 配置有些许不同，主要是 gpio 的配置存在差异。

USB0 配置：

```

usbc0:usbc0@0 {
    device_type = "usbc0";
    usb_port_type = <0x2>;
    usb_detect_type = <0x1>;
    usb_id_gpio = <&pio PD 21 GPIO_ACTIVE_HIGH>;
    enable-active-high;
    usb_det_vbus_gpio = <&pio PD 20 GPIO_ACTIVE_HIGH>;
    usb_wakeup_suspend = <0>;
    usb_serial_unique = <0>;
};

```

```
usb_serial_number = "20080411";
rndis_wceis = <1>;
status = "okay";
};
```

参数	功能
usb_port_type	USB 模式, 0:device; 1:host; 2:OTG;
usb_detect_type	USB 检测类型, 0:none; 1:vbus/id detect; 2:id/dpdm; 一般只用 1
usb_detect_mode	USB ID 脚检测模式, 0: 线程轮询; 1: 中断检测; 默认为 0
usb_id_gpio	USB ID 脚配置
usb_det_vbus_gpio	USB VBUS 检测引脚配置; 一般使用对应 gpio 即可; 注意, 如果方案不需要检测 VBUS, 那么这里请填写 USB ID 脚, 如果方案由 PMU 检测 VBUS, 那么这里请填写 "axp_ctrl"
det_vbus_supply	与 usb_det_vbus_gpio 类似, 用于检测 VBUS 是否存在。它会指向 usb_power_supply 节点。
usb_wakeup_suspend	是否支持 USB 唤醒; 如果需要支持进入深度休眠后的 USB 唤醒, 则置为 1
usb_serial_unique	是否使用唯一序列号 (由芯片 chipid 得到)
usb_serial_number	如果 usb_serial_unique 为 0, 那么使用这里的配置的值作为序列号
status	是否启用该模块。"okay": 启用; "disabled": 关闭

有些时候, 还需要对udc进行配置:

```
udc:udc-controller@0x05100000 {
    det_vbus_supply = <&usb_power_supply>;
};
```

这里det_vbus_supply同样指向usb_power_supply节点, 这里的目的是让udc获取到usb power, 并设置限流值。udc驱动中会检测到USB接入的是PC, 然后软件设置USB限流值。

如果PMU支持并开启了BC1.2协议(识别USB的类型并且硬件上自动设置限流值), 那么这里就不需要配置。

无论是 USB0 还是 USB1, EHCI/OHCI 的配置都是类似的, 具体 EHCI/OHCI 配置如下:

```
ehci0:ehci0-controller@0x04101000 {
    drvvbus-supply = <&reg_drivevbus>;
};

ohci0:ohci0-controller@0x04101400 {
    drvvbus-supply = <&reg_drivevbus>;
};
```

这里 reg_drivevbus 主要配置对应的 regulator 作为 OTG_5V 供电。常见的配置有:

```
使用PMU自带的DRVVBUS引脚使能对应的5V输出:
regulator0: regulators@0 {
    reg_drivevbus: drivevbus {
        regulator-name = "axp2585-drivevbus";
```

```

    };
};

使用GPIO去控制5V输出:
reg_usb0_vbus: usb0-vbus {
    compatible = "regulator-fixed";
    regulator-name = "usb0-vbus";
    regulator-min-microvolt = <5000000>;
    regulator-max-microvolt = <5000000>;
    regulator-enable-ramp-delay = <1000>;
    gpio = <&pio PD 19 GPIO_ACTIVE_HIGH>;
    enable-active-high;
};
    
```

USB1 配置:

```

usbcl:usbcl@0 {
    device_type = "usbcl";
    usb_regulator_io = "nocare";
    usb_wakeup_suspend = <0>;
    status = "okay";
};
    
```

配置与 USB0 类似，但少了 UDC 相关部分:

参数	功能
usb_regulator_io	IO 的供电，如果没有则“nocare”
usb_wakeup_suspend	是否支持 USB 唤醒；如果需要支持进入深度休眠后的 USB 唤醒，则置为 1
status	是否启用该模块。“okay”：启用；“disabled”：关闭

2.2.3 内核配置

USB device 相关配置:

```

Device Drivers --->
[*] USB support --->
    <*> USB Gadget Support --->
        USB Peripheral Controller --->
            <*> SoftWinner SUNXI USB Peripheral Controller
    
```

USB Host 相关配置:

```

Device Drivers --->
[*] USB support --->
    <*> Support for Host-side USB
        <*> EHCI HCD (USB 2.0) support
        <*> OHCI HCD (USB 1.1) support
        <*> SoftWinner SUNXI USB Host Controller support
        <*> SoftWinner SUNXI USB HCI
        <*> SoftWinner SUNXI USB EHCI0
    
```

```
<*>      SoftWinner SUNXI USB EHCI1
<*>      SoftWinner SUNXI USB OHCI0
<*>      SoftWinner SUNXI USB OHCI1
```

USB 角色管理相关配置:

```
Device Drivers --->
[*] USB support --->
  <*>  SUNXI USB2.0 Dual Role Controller support --->
  <*>  SUNXI USB2.0 Manager
```

其他内核配置，例如 gadget 驱动配置，请参考[常用功能配置](#)章节

2.3 代码结构

USB 驱动的源代码位于内核 drivers/usb 目录下，如下是 sunxi 平台相关源码:

USB Host:

```
drivers/usb/host/
├─ ehci_sunxi.c
├─ ohci_sunxi.c
├─ sunxi_hci.c
└─ sunxi_hci.h
```

UDC 和 Manager:

```
drivers/usb/sunxi_usb/
├─ include
│  └─ sunxi_hcd.h
│  └─ sunxi_sys_reg.h
│  └─ sunxi_udc.h
│  └─ sunxi_usb_board.h
│  └─ sunxi_usb_bsp.h
│  └─ sunxi_usb_config.h
│  └─ sunxi_usb_debug.h
│  └─ sunxi_usb_typedef.h
├─ Kconfig
├─ Makefile
├─ manager
│  └─ usbc0_platform.c
│  └─ usbc_platform.h
│  └─ usb_hcd_servers.c
│  └─ usb_hcd_servers.h
│  └─ usb_hw_scan.c
│  └─ usb_hw_scan.h
│  └─ usb_manager.c
│  └─ usb_manager.h
│  └─ usb_msg_center.c
│  └─ usb_msg_center.h
├─ misc
│  └─ sunxi_usb_debug.c
└─ udc
```

```
| | | sunxi_udc_board.c  
| | | sunxi_udc_board.h  
| | | sunxi_udc.c  
| | | sunxi_udc_config.h  
| | | sunxi_udc_debug.c  
| | | sunxi_udc_debug.h  
| | | sunxi_udc_dma.c  
| | | sunxi_udc_dma.h  
| | |  
| | | usbc  
| | | | | usbc.c  
| | | | | usbc_dev.c  
| | | | | usbc_i.h  
| | | | | usbc_phy.c
```



3 常用功能配置

3.1 配置 OTG 功能

OTG 功能下，需要根据 USB ID 脚去进行 Device, Host 模式的切换；如果需要支持 NULL 模式 (既不加载 Device 也不加载 Host 驱动), 那么还需要 VBUS 状态检测引脚。

涉及到的主要改动点：

```
usb_port_type配置为2,即OTG模式。
usb_id_gpio配置对应的USB ID引脚。
usb_det_vbus_gpio, 需要根据实际情况进行配置:
1. 如果需要检测VBUS状态, 则按下面情况进行配置:
  a. 由gpio引脚进行检测。直接配置对应的gpio即可。
  b. 由我司PMU检测。填写"axp_ctrl"
2. 如果不需要检测VBUS状态(缺少NULL模式)
   那么直接填写USB ID的gpio配置(也就是VBUS与ID状态一致)。
```

otg 功能相关的调试节点，每个芯片平台的路径可能会不一样，具体可以通过 `find -name otg_role` 确认，下面默认以 `/sys/devices/platform/soc/usbc0/otg_role` 这个路径举例。

验证方法：

```
拔插OTG线，并通过下面节点查询当前USB角色：
cat /sys/devices/platform/soc/usbc0/otg_role
```

调试方法：

```
确认当前角色：
cat /sys/devices/platform/soc/usbc0/otg_role
```

```
开启角色切换调试：
echo 1 > /sys/devices/platform/soc/usbc0/hw_scan_debug
会有下面类似打印：
Id=3,role=2

role表示当前角色， 0:NULL; 1:HOST; 2:DEVICE

Id值，主要看低两位，bit0表示ID，bit1表示VBUS，
例如3表示ID脚为高，VBUS脚也为高，所以当前应该为Device模式
```

Id 值的所有情况：

Id 值	ID	VBUS	模式
0x00	0	0	Host
0x01	1	0	Null
0x02	0	1	Host
0x03	1	1	Device

- ID 脚一般情况下为高电平，只有接入 OTG 线时会拉低；
- VBUS 为 1 表示 micro USB 口有接入外部电源；
- 一般不会出现 Id 值为 0x02 的情况，这表示接入 OTG 线，但是还检测到 VBUS；
- 如果没有 VBUS 检测，Id 值就只有 0x00 和 0x03 两种情况，也就是说要么加载 device 驱动，要么加载 host 驱动；这会带来一些影响：usb 相关时钟一直被打开，导致有一定功耗，以及硬件射频。

3.2 USB Gadget 功能配置

USB Gadget 支持众多功能，它们的配置方法比较类似：

- 内核配置选中对应功能。
- 私有 gadget 功能配置。通过 configs 创建对应的 gadget 功能，并设置私有属性节点

其中，针对部分 gadget 功能，我们在 `setusbconfig` 脚本中封装了 gadget 功能配置，可供参考，具体请看 [setusbconfig](#) 章节。

3.2.1 ADB 功能配置

内核相关配置：

```
Device Drivers --->
[*] USB support --->
  <*> USB Gadget Support --->
    <*> USB functions configurable through configs
      [*] Function filesystem (FunctionFS)
```

进入系统后输入 `setusbconfig adb` 即可切换并配置为 adb 模式，涉及到的 configs 相关配置：

```
设置idVendor, idProduct:
echo 0x18d1 > /sys/kernel/config/usb_gadget/g1/idVendor
echo 0xD002 > /sys/kernel/config/usb_gadget/g1/idProduct

创建adb功能:
mkdir /sys/kernel/config/usb_gadget/g1/functions/ffs.adb
```

```
建立config软链接：
ln -s /sys/kernel/config/usb_gadget/g1/functions/ffs.adb/ /sys/kernel/config/usb_gadget/g1/
  configs/c.1/ffs.adb

挂载functionfs：
mount -o uid=2000,gid=2000 -t functionfs adb /dev/usb-ffs/adb/

设置对应的UDC控制器名称：
udc_controller=`ls /sys/class/udc`
echo $udc_controller > /sys/kernel/config/usb_gadget/g1/UDC
```

使用方法：需要开机启动 adbd 服务, 具体请参考[ADB 功能](#)章节

3.2.2 MTP 功能配置

内核相关配置：

```
Device Drivers --->
[*] USB support --->
  <*> USB Gadget Support --->
    <*> USB functions configurable through configfs
      [*] MTP gadget
```

进入系统后输入 `setusbconfig mtp` 即可切换并配置为 `mtp` 模式, 涉及到的 `configfs` 相关配置：

```
设置idVendor, idProduct:
echo 0x045E > /sys/kernel/config/usb_gadget/g1/idVendor
echo 0x00C9 > /sys/kernel/config/usb_gadget/g1/idProduct

创建mtp功能:
mkdir /sys/kernel/config/usb_gadget/g1/functions/mtp.gs0

建立config软链接:
ln -s /sys/kernel/config/usb_gadget/g1/functions/mtp.gs0/ /sys/kernel/config/usb_gadget/g1/
  configs/c.1/mtp.gs0

设置对应的UDC控制器名称:
udc_controller=`ls /sys/class/udc`
echo $udc_controller > /sys/kernel/config/usb_gadget/g1/UDC
```

使用方法：需要开机启动 MtpDaemon 服务, 具体请参考[MTP 功能](#)章节

3.2.3 Mass Storage 功能配置

内核相关配置：

```
Device Drivers --->
[*] USB support --->
  <*> USB Gadget Support --->
    <*> USB functions configurable through configfs
```

```
[*] Mass storage
```

进入系统后输入 `setusbconfig mass_storage` 即可切换并配置为 `mass_storage` 模式，涉及到的 `configs` 相关配置：

```
设置idVendor, idProduct:
echo 0x1f3a > /sys/kernel/config/usb_gadget/g1/idVendor
echo 0x1000 > /sys/kernel/config/usb_gadget/g1/idProduct

创建mass storage功能:
mkdir /sys/kernel/config/usb_gadget/g1/functions/mass_storage.usb0

建立config软链接:
ln -s /sys/kernel/config/usb_gadget/g1/functions/mass_storage.usb0/ /sys/kernel/config/
usb_gadget/g1/configs/c.1/f1

设置对应的UDC控制器名称:
udc_controller=`ls /sys/class/udc`
echo $udc_controller > /sys/kernel/config/usb_gadget/g1/UDC
```

使用方法：

```
把要挂载的块设备写入file节点中:
echo /dev/by-name/boot-res > /sys/kernel/config/usb_gadget/g1/functions/mass_storage.usb0/
lun.0/file

卸载:
echo "" > /sys/kernel/config/usb_gadget/g1/functions/mass_storage.usb0/lun.0/file
```

3.2.4 RNDIS 功能配置

内核相关配置：

```
Device Drivers --->
[*] USB support --->
  <*> USB Gadget Support --->
    <*> USB functions configurable through configs
      [*] RNDIS
```

进入系统后输入 `setusbconfig rndis` 即可切换并配置为 `rndis` 模式，涉及到的 `configs` 相关配置：

```
设置idVendor, idProduct:
echo 0x0525 > /sys/kernel/config/usb_gadget/g1/idVendor
echo 0xa4a2 > /sys/kernel/config/usb_gadget/g1/idProduct

创建rndis功能:
mkdir /sys/kernel/config/usb_gadget/g1/functions/rndis.usb0

建立config软链接:
ln -s /sys/kernel/config/usb_gadget/g1/functions/rndis.usb0/ /sys/kernel/config/usb_gadget/
g1/configs/c.1/rndis.usb0
```

```
设置对应的UDC控制器名称:  
udc_controller=`ls /sys/class/udc`  
echo $udc_controller > /sys/kernel/config/usb_gadget/g1/UDC
```

使用方法:

通过ifconfig命令确认是否新增了网卡设备。

3.2.5 UAC1 功能配置

内核相关配置:

```
Device Drivers --->  
[*] USB support --->  
  <*> USB Gadget Support --->  
    <*> USB functions configurable through configs  
      [*] Audio Class 1.0
```

configs 相关配置:

```
设置idVendor, idProduct:  
echo 0x1d61 > /sys/kernel/config/usb_gadget/g1/idVendor  
echo 0x0101 > /sys/kernel/config/usb_gadget/g1/idProduct  
  
创建uac1功能:  
mkdir /sys/kernel/config/usb_gadget/g1/functions/uac1.usb0  
  
设置UAC1私有属性节点:  
设置通道数掩码,例如支持两通道:  
echo 0x3 > /sys/kernel/config/usb_gadget/g1/functions/uac1.usb0/c_chmask  
echo 0x3 > /sys/kernel/config/usb_gadget/g1/functions/uac1.usb0/p_chmask  
  
注意我们默认代码UDC的endpoint中仅有一个ISO端点,所以只支持单项的同步传输,  
默认会将p_chmask设置为0,也就是关闭playback通路  
  
设置采样率:  
echo 16000 > /sys/kernel/config/usb_gadget/g1/functions/uac1.usb0/p_srate  
echo 48000 > /sys/kernel/config/usb_gadget/g1/functions/uac1.usb0/c_srate  
  
建立config软链接:  
ln -s /sys/kernel/config/usb_gadget/g1/functions/uac1.usb0/ /sys/kernel/config/usb_gadget/  
g1/configs/c.1/uac1.usb0  
  
设置对应的UDC控制器名称:  
udc_controller=`ls /sys/class/udc`  
echo $udc_controller > /sys/kernel/config/usb_gadget/g1/UDC
```

使用方法:

```
默认配置下,仅使能了c_chmask,即PC端往小机端送音频数据,小机端播放出来  
PC端对新增的USB声卡进行播歌  
然后小机端执行下面命令:  
arecord -Dhw:UAC1Gadget -f S16_LE -r 48000 -c 2 |aplay  
表示从虚拟声卡(拿到PC端送来的数据)录音,然后直接播放出来
```

3.2.6 HID 功能配置

内核相关配置:

```
Device Drivers --->
[*] USB support --->
  <*> USB Gadget Support --->
    <*> USB functions configurable through configfs
      [*] HID function
```

进入系统后输入 `setusbconfig hid` 即可切换并配置为 hid 模式, 涉及到的 configfs 相关配置:

```
设置idVendor, idProduct:
echo 0x0525 > /sys/kernel/config/usb_gadget/g1/idVendor
echo 0xa4ac > /sys/kernel/config/usb_gadget/g1/idProduct

创建HID功能:
mkdir /sys/kernel/config/usb_gadget/g1/functions/hid.usb0

设置HID相关节点(根据实际HID功能进行配置,下面是模拟键盘的例子):
echo 1 > /sys/kernel/config/usb_gadget/g1/functions/hid.usb0/protocol
echo 8 > /sys/kernel/config/usb_gadget/g1/functions/hid.usb0/report_length
echo -ne \\x05\\x01\\x09\\x06\\xa1\\x01\\x05\\x07\\x19\\xe0\\x29\\xe7\\x15\\x00\\x25\\x01\\x75\\x01\\x95\\x08\\x81\\x02\\x95\\x01\\x75\\x08\\x81\\x03\\x95\\x05\\x75\\x01\\x05\\x08\\x19\\x01\\x29\\x05\\x91\\x02\\x95\\x01\\x75\\x03\\x91\\x03\\x95\\x06\\x75\\x08\\x15\\x00\\x25\\x65\\x05\\x07\\x19\\x00\\x29\\x65\\x81\\x00\\xc0 > /sys/kernel/config/usb_gadget/g1/functions/hid.usb0/report_desc

建立config软链接:
ln -s /sys/kernel/config/usb_gadget/g1/functions/hid.usb0/ /sys/kernel/config/usb_gadget/g1/configs/c.1/hid.usb0

设置对应的UDC控制器名称:
udc_controller=`ls /sys/class/udc`
echo $udc_controller > /sys/kernel/config/usb_gadget/g1/UDC
```

使用方法:

```
设置gadget hid功能后, 会生成/dev/hidgX节点,
然后可以参考linux-4.9/Documentation/usb/gadget_hid.txt里面的demo去进行测试。

测试命令:
./hid_gadget_test /dev/hidg0 keyboard

然后按下对应按键, 观察PC端是否接受到对应消息。
```

3.2.7 printer 功能配置

内核相关配置:

```
Device Drivers --->
[*] USB support --->
  <*> USB Gadget Support --->
```

```
<*> USB functions configurable through configs
[*] Printer function
```

进入系统后输入 `setusbconfig printer` 即可切换并配置为 `printer` 模式, 涉及到的 `configs` 相关配置:

```
设置idVendor, idProduct:
echo 0x0525 > /sys/kernel/config/usb_gadget/g1/idVendor
echo 0xa4a8 > /sys/kernel/config/usb_gadget/g1/idProduct

创建printer功能:
mkdir /sys/kernel/config/usb_gadget/g1/functions/printer.usb0

设置printer相关节点(根据实际情况进行设置, 下面是例子):
echo "MFG:linux;MDL:g_printer;CLS:PRINTER;SN:1;" > /sys/kernel/config/usb_gadget/g1/
  functions/printer.usb0/pnp_string
echo 10 > /sys/kernel/config/usb_gadget/g1/functions/printer.usb0/q_len

建立config软链接:
ln -s /sys/kernel/config/usb_gadget/g1/functions/printer.usb0/ /sys/kernel/config/
  usb_gadget/g1/configs/c.1/printer.usb0

设置对应的UDC控制器名称:
udc_controller=`ls /sys/class/udc`
echo $udc_controller > /sys/kernel/config/usb_gadget/g1/UDC
```

使用方法:

```
设置gadget printer功能后, 会生成/dev/g_printer0节点,
然后可以参考linux-4.9/Documentation/usb/gadget_printer.txt里面的demo去进行测试。
```

3.3 修改 UDC Endpoint 配置

UDC 硬件上一般支持 4 组 (极个别芯片可支持 5 组) 可配置的 endpoint, 默认代码下我们配置了两组 BULK ep, 一个 ISO ep, 一个 INT ep。

根据实际使用的 gadget 功能, 要求的 ep 传输类型也不一样, 下表简单列举的部分 gadget 功能要求的 ep 类型、个数:

gadget 功能	ep 类型	个数
hid	INT	2, IN/OUT
accessory	BULK	2, IN/OUT
loopback	BULK	2, IN/OUT
mass_storage	BULK	2, IN/OUT
mtp	BULK	2, IN/OUT
	INT	1, IN
uac1	ISOC	2, IN/OUT
rndis	INT	1, IN

gadget 功能	ep 类型	个数
	BULK	2, IN/OUT
serial	BULK	2, IN/OUT
printer	BULK	2, IN/OUT

个别功能根据实际配置，要求的ep可能也有变化，例如uac1：
如果有2个ISOC ep, 那么可以同时支持录音、播放；
如果只有一个ISOC ep, 可以支持录音或者播放。

下面介绍如何修改代码支持不同类型的 ep, 改动点只有两个文件:

1. 修改 drivers/usb/sunxi_usb/include/sunxi_udc.h

进行修改前，请注意：

我们芯片的udc分两种：
a. 4K FIFO, 4组ep
b. 8K FIFO, 5组ep
请根据spec或者sunxi_udc.h中的宏去区分，大部分芯片都是第一种，4K FIFO, 4组ep。

修改 ep_fifo 配置：

```
默认配置：
static const struct sw_udc_fifo ep_fifo[] = {
    {ep0name,      0,    512, 0},
    {ep1in_bulk_name, 512, 512, 0},
    {ep1out_bulk_name, 1024, 512, 0},
    {ep2in_int_name, 1536, 512, 0},
    {ep2out_int_name, 2048, 512, 0},
    {ep3_iso_name, 2560, 512, 0},
    {ep4in_int_name, 3072, 512, 0},
    {ep4out_int_name, 3584, 512, 0},
};
static const int ep_fifo_in[] = {0, 1, 3, 5, 6, 7};
static const int ep_fifo_out[] = {0, 2, 4, 5, 7, 8};
```

ep_fifo中，第一列表示ep名称；第二列表示FIFO起始地址；第三列表示ep的长度；第四列参数设置为0即可。

ep_fifo_in的配置, 这里会配置所有输入方向的ep(ep0 + 4组ep)的序号, 序号指的是上面ep_fifo数组的index。

例如ep_fifo_in数组中：

0表示ep0对应ep_fifo[0], 即ep0;
1对应ep_fifo[1], 即ep1in-bulk;
3表示ep_fifo[3], 即ep2in-bulk;
5表示ep_fifo[5], 即ep3-iso;
6表示ep_fifo[6], 即ep4-int;
7在这里没有作用, 只有某些芯片支持5组ep的才会用到这个index;

ep_fifo_out的配置方法与ep_fifo_out类似。

2. 修改 drivers/usb/sunxi_usb/udc/sunxi_udc.c

主要修改 static struct sunxi_udc sunxi_udc 这个结构体里面的 ep 数组。

数组需要根据 sunxi_udc.h 中的 ep_fifo 一一对应, 修改每个 ep 时需要注意:

- num 字段, 表示第几组 ep;
- name 字段, 对应的 ep 名称;
- maxpacket 字段, FIFO 大小, 一般为 512;
- caps, 指定传输类型以及方向;
- bEndpointAddress, 传输方向以及 ep 序号;
- bmAttributes, 传输类型;

下面一些修改例子:

1. 支持 ISO IN/OUT

```
drivers/usb/sunxi_usb/include/sunxi_udc.h改动:

#define SW_UDC_EP_ISO_FIFO_SIZE          512
static const char ep3in_iso_name[]      = "ep3in-iso";
static const char ep3out_iso_name[]     = "ep3out-iso";

static const struct sw_udc_fifo ep_fifo[] = {
    {ep0name,      0,    512,  0},
    {ep1in_bulk_name, 512, 512,  0},
    {ep1out_bulk_name, 1024, 512,  0},
    {ep2in_bulk_name, 1536, 512,  0},
    {ep2out_bulk_name, 2048, 512,  0},
    {ep3in_iso_name, 2560, 512,  0},
    {ep3out_iso_name, 3072, 512,  0},
    {ep4_int_name, 3584, 512,  0},
};

static const int ep_fifo_in[] = {0, 1, 3, 5, 7, 7};
static const int ep_fifo_out[] = {0, 2, 4, 6, 7, 8};

drivers/usb/sunxi_usb/udc/sunxi_udc.c改动:
主要改动了ep[5],ep[6]
    .ep[5] = {
        .num          = 3,
        .ep = {
            .name      = ep3in_iso_name,
            .ops        = &sunxi_udc_ep_ops,
            .maxpacket = SW_UDC_EP_ISO_FIFO_SIZE,
            .maxpacket_limit = SW_UDC_EP_ISO_FIFO_SIZE,
            .caps      = USB_EP_CAPS(USB_EP_CAPS_TYPE_ISO,
                USB_EP_CAPS_DIR_IN),
        },
        .dev          = &sunxi_udc,
        .bEndpointAddress = (USB_DIR_IN | 3),
        .bmAttributes = USB_ENDPOINT_XFER_ISOC,
    },

    .ep[6] = {
        .num          = 3,
        .ep = {
            .name      = ep3out_iso_name,
            .ops        = &sunxi_udc_ep_ops,
```

```

        .maxpacket = SW_UDC_EP_ISO_FIFO_SIZE,
        .maxpacket_limit = SW_UDC_EP_ISO_FIFO_SIZE,
        .caps = USB_EP_CAPS(USB_EP_CAPS_TYPE_ISO,
            USB_EP_CAPS_DIR_OUT),
    },
    .dev = &sunxi_udc,
    .bEndpointAddress = (USB_DIR_OUT | 3),
    .bmAttributes = USB_ENDPOINT_XFER_ISOC,
},

```

2. 支持两个 INT 传输，方向均支持

默认代码支持了一个INT传输，所以增加一个即可，最简单的就是将ISO端点改为INT：

```

drivers/usb/sunxi_usb/include/sunxi_udc.h改动:
static const char ep3_int_name[] = "ep3-int";
static const struct sw_udc_fifo ep_fifo[] = {
    {ep0name, 0, 512, 0},
    {ep1in_bulk_name, 512, 512, 0},
    {ep1out_bulk_name, 1024, 512, 0},
    {ep2in_bulk_name, 1536, 512, 0},
    {ep2out_bulk_name, 2048, 512, 0},
    {ep3_int_name, 2560, 1024, 0},
    {ep4_int_name, 3584, 512, 0},
};

drivers/usb/sunxi_usb/udc/sunxi_udc.c改动:
主要改动了ep[5]
    .ep[5] = {
        .num = 3,
        .ep = {
            .name = ep3_int_name,
            .ops = &sunxi_udc_ep_ops,
            .maxpacket = SW_UDC_EP_FIFO_SIZE,
            .maxpacket_limit = SW_UDC_EP_FIFO_SIZE,
            .caps = USB_EP_CAPS(USB_EP_CAPS_TYPE_INT,
                USB_EP_CAPS_DIR_ALL),
        },
        .dev = &sunxi_udc,
        .bEndpointAddress = 3,
        .bmAttributes = USB_ENDPOINT_XFER_INT,
    },
},

```

3. 支持 4 组 BULK IN/OUT

每个 ep 都至少占用 512 字节的 FIFO, 如果支持 4 组 BULK, 并且 IN/OUT 都需要, 那么共需要 $8 \times 512 = 4K$ 的 FIFO, 另外 ep0 至少需要 64 字节的 FIFO。而我们 UDC 的 FIFO 实际大小是 $4K + 64\text{byte}$, 是可以满足该需求的, 配置方法如下所示:

默认代码下支持了两组BULK端点，现在需要多增加两组BULK。

```

drivers/usb/sunxi_usb/include/sunxi_udc.h改动:
static const char ep3in_bulk_name[] = "ep3in-bulk";
static const char ep3out_bulk_name[] = "ep3out-bulk";
static const char ep4in_bulk_name[] = "ep4in-bulk";
static const char ep4out_bulk_name[] = "ep4out-bulk";

```

```
static const struct sw_udc_fifo ep_fifo[] = {
    {ep0name,      0,      64,  0},
    {ep1in_bulk_name, 64,    512, 0},
    {ep1out_bulk_name, 512+64, 512, 0},
    {ep2in_bulk_name, 1024+64, 512, 0},
    {ep2out_bulk_name, 1536+64, 512, 0},
    {ep3in_bulk_name, 2048+64, 512, 0},
    {ep3out_bulk_name, 2560+64, 512, 0},
    {ep4in_bulk_name, 3072+64, 512, 0},
    {ep4out_bulk_name, 3584+64, 512, 0},
};

static const int ep_fifo_in[] = {0, 1, 3, 5, 7, 7};
static const int ep_fifo_out[] = {0, 2, 4, 6, 8, 8};
```

drivers/usb/sunxi_usb/udc/sunxi_udc.c改动:

主要改动了ep[5],ep[6],ep[7],ep[8]

```
.ep[5] = {
    .num                = 3,
    .ep = {
        .name           = ep3in_bulk_name,
        .ops            = &sunxi_udc_ep_ops,
        .maxpacket      = SW_UDC_EP_FIFO_SIZE,
        .maxpacket_limit = SW_UDC_EP_FIFO_SIZE,
        .caps           = USB_EP_CAPS(USB_EP_CAPS_TYPE_BULK,
            USB_EP_CAPS_DIR_IN),
    },
    .dev                = &sunxi_udc,
    .bEndpointAddress  = (USB_DIR_IN | 3),
    .bmAttributes      = USB_ENDPOINT_XFER_BULK,
},

.ep[6] = {
    .num                = 3,
    .ep = {
        .name           = ep3out_bulk_name,
        .ops            = &sunxi_udc_ep_ops,
        .maxpacket      = SW_UDC_EP_FIFO_SIZE,
        .maxpacket_limit = SW_UDC_EP_FIFO_SIZE,
        .caps           = USB_EP_CAPS(USB_EP_CAPS_TYPE_BULK,
            USB_EP_CAPS_DIR_OUT),
    },
    .dev                = &sunxi_udc,
    .bEndpointAddress  = (USB_DIR_OUT | 3),
    .bmAttributes      = USB_ENDPOINT_XFER_BULK,
},

.ep[7] = {
    .num                = 4,
    .ep = {
        .name           = ep4in_bulk_name,
        .ops            = &sunxi_udc_ep_ops,
        .maxpacket      = SW_UDC_EP_FIFO_SIZE,
        .maxpacket_limit = SW_UDC_EP_FIFO_SIZE,
        .caps           = USB_EP_CAPS(USB_EP_CAPS_TYPE_BULK,
            USB_EP_CAPS_DIR_IN),
    },
    .dev                = &sunxi_udc,
    .bEndpointAddress  = (USB_DIR_IN | 4),
```

```

    .bmAttributes      = USB_ENDPOINT_XFER_BULK,
},
.ep[8] = {
    .num              = 4,
    .ep = {
        .name         = ep4out_bulk_name,
        .ops          = &sunxi_udc_ep_ops,
        .maxpacket    = SW_UDC_EP_FIFO_SIZE,
        .maxpacket_limit = SW_UDC_EP_FIFO_SIZE,
        .caps         = USB_EP_CAPS(USB_EP_CAPS_TYPE_BULK,
                                   USB_EP_CAPS_DIR_OUT),
    },
    .dev              = &sunxi_udc,
    .bEndpointAddress = (USB_DIR_OUT | 4),
    .bmAttributes     = USB_ENDPOINT_XFER_BULK,
},
},

```

3.4 USB Host 功能配置

3.4.1 U 盘功能

```

Device Drivers --->
  SCSI device support --->
    <*> SCSI device support
      <*> SCSI disk support
  [*] USB support --->
    <*>   USB Mass Storage support
    <*>   Datafab Compact Flash Reader support
    <*>   Freecom USB/ATAPI Bridge support
    <*>   ISD-200 USB/ATA Bridge support
    <*>   USBAT/USBAT02-based storage support
    <*>   SanDisk SDDR-09 (and other SmartMedia, including DPCM) support
    <*>   SanDisk SDDR-55 SmartMedia support
    <*>   Lexar Jumpshot Compact Flash Reader
    <*>   Olympus MAUSB-10/Fuji DPC-R1 support
    <*>   Support OneTouch Button on Maxtor Hard Drives
    <*>   Support for Rio Karma music player
    <*>   SAT emulation on Cypress USB/ATA Bridge with ATACB
    <*>   USB Attached SCSI

```

3.4.2 USB 声卡

```

Device Drivers --->
  <*> Sound card support --->
    [*] USB sound devices --->
      <*> USB Audio/MIDI driver

```

3.4.3 USB 网卡

```
Device Drivers --->
[*] Network device support --->
    <*> USB Network Adapters --->
在这选择对应的驱动
例如是rndis驱动:
    <*> Multi-purpose USB Networking Framework
    <*> Host for RNDIS and ActiveSync devices
```

3.4.4 USB HID 设备

```
Device Drivers --->
HID support --->
    USB HID support --->
    <*> USB HID transport layer
```

3.4.5 USB 摄像头

```
Device Drivers --->
<*> Multimedia support --->
    [*] Media USB Adapters --->
        <*> USB Video Class (UVC)
        [*] UVC input events device support
```

3.4.6 USB 打印机

```
Device Drivers --->
SCSI device support --->
    <*> SCSI device support
    <*> SCSI disk support
[*] USB support --->
    <*> USB Printer support
```

接入 usb 打印机设备后，会生成对应设备节点：/dev/usb/lp0

3.5 支持 USB Standby

部分芯片可以支持 USB Standby，例如外接 USB 鼠标键盘，在点击鼠标时，可以唤醒系统。

软件上注意配置 `usb_wakeup_suspend` 为 1 即可。

硬件上注意 vdd-usb 的电在休眠时不能关。如果用的是 USB0, 那么注意 OTG-5V 的电也不能掉, 否则 USB 外设不能正常工作。

3.6 设置 USB 序列号

很多时候 adb devices 查看设备时, 它的序列号都是 20080411, 它的值就是有 USB 序列号决定的。

如果要求每个设备都有自己唯一的序列号, 那么可以配置芯片的 chipid 转换得到序列号, 并设置到 USB 驱动中, 修改步骤如下所示:

1. 修改 uboot 配置, 增加 CONFIG_SUNXI_SERIAL 配置

增加该配置后, uboot阶段会读取chipid并转换得到serial number。
并将该serial number设置到环境变量snum中, 然后启动内核时获取该变量, 并增加到内核cmdline中: "androidboot.serialno="

2. usb 配置使用唯一序列号

修改dts, 将usb0的usb_serial_unique改为1
usb相关驱动中会读取cmdline中"androidboot.serialno"字段获取序列号
后续gadget驱动中均会配置使用该序列号
注意, 进入系统后不要再修改/sys/kernel/config/usb_gadget/g1/strings/0x409/serialnumber该节点。

验证方法:

可以通过adb devices查看序列号

3.7 测试 USB 速率

可以通过挂载移动硬盘, 然后进行测试读写性能。

写性能:

```
确认移动硬盘的挂载目录, 如/mnt/exUDISK/, 然后执行下面命令:  
time dd if=/dev/zero of=/mnt/exUDISK/test.bin bs=1MB count=1024  
1024+0 records in  
1024+0 records out  
real 0m 36.63s  
user 0m 0.00s  
sys 0m 3.41s  
所以写性能为:1024/36.63=27.95Mb/s
```

读性能:

```
time dd if=/mnt/exUDISK/test.bin of=/dev/null bs=1MB count=1024
1024+0 records in
1024+0 records out
real    0m 28.99s
user    0m 0.01s
sys     0m 2.38s
所以读性能为:1024/28.99=35.33Mb/s
```

注意，不同的 U 盘、移动硬盘，受存储介质、文件系统格式影响，速率可能会存在较大差别，建议使用 fat32 文件系统的移动硬盘进行测试；

3.8 USB 功能裁剪

一般我们默认 SDK 都会默认开启 USB0 的 OTG 功能，USB1 的 Host 功能。而不同客户产品对 USB 功能的需求不一样，可以根据下面几个例子进行功能裁剪：

3.8.1 删除多余的 usb gadget 功能

实际产品仅用到有限种类的 gadget 功能，可以通过 make kernel_menuconfig 进入相关配置项：

```
Device Drivers --->
[*] USB support --->
  <*> USB Gadget Support --->
    <*> USB functions configurable through configs //必须要选configs功能
    [*] Mass storage //USB Masss Storage功能，可
        以理解设备模拟为U盘
    [*] Function filesystem (FunctionFS) //ADB就是通过该ffs功能实现
    [*] MTP gadget
    [*] HID function
    [*] Printer function
```

实际产品可以根据情况仅选用自己需要用到的 gadget 功能。

3.8.2 仅用到 usb0 的 OTG 功能

可以删除 USB1 相关的配置

```
Device Drivers --->
[*] USB support --->
  < > SoftWinner SUNXI USB EHCI1
  < > SoftWinner SUNXI USB OHCI1
```

把 EHCI1,OHCI1 的配置去掉。

3.8.3 仅使用到 adb 功能

adb 功能需要用到 USB0 的 device 功能，因此可以把 USB1 以及 Host 相关配置都删除。

```
Device Drivers --->
[*] USB support --->
  <*> Support for Host-side USB
  <*> USB Gadget Support --->
    <*> USB functions configurable through configs //必须要选configfs功能
    [*] Function filesystem (FunctionFS) //ADB就是通过该ffs功能实现
  <*> SUNXI USB2.0 Dual Role Controller support --->
```

“USB Support”配置页中仅保持上面选项，其他例如“HCI”，“USB Mass Storage”相关配置都去掉。



4 常见调试方法

4.1 调试节点

查看 USB0 当前 Role:

```
cat /sys/devices/platform/soc/usbc0/otg_role
```

手动切换到 Host 模式:

```
cat /sys/devices/platform/soc/usbc0/usb_host
```

手动切换到 Device 模式:

```
cat /sys/devices/platform/soc/usbc0/usb_device
```

注意，手动切换模式后，OTG 自动切换角色的功能就会失效；

USB1 调试节点:

卸载主机驱动

```
echo 0 > /sys/devices/platform/soc/5200000.ehci1-controller/ehci_enable  
echo 0 > /sys/devices/platform/soc/5200000.ohci1-controller/ohci_enable
```

加载主机驱动

```
echo 1 > /sys/devices/platform/soc/5200000.ehci1-controller/ehci_enable  
echo 1 > /sys/devices/platform/soc/5200000.ohci1-controller/ohci_enable
```

4.2 测试 USB 眼图

USB0 OTG Device 眼图测试:

手动切换Device模式:

```
cat /sys/devices/platform/soc/usbc0/usb_device
```

每个芯片平台的节点路径会有些差异，请通过find命令确认:

```
find /sys -name otg_ed_test
```

眼图测试命令:

```
echo test_pack > /sys/devices/platform/soc/5100000.udc-controller/otg_ed_test
```

USB0 OTG Host 眼图测试:

手动切换Host模式:

```
cat /sys/devices/platform/soc/usbc0/usb_host
```

每个芯片平台的节点路径会有些差异, 请通过find命令确认(注意选择带有ehci0的路径):

```
find /sys -name ed_test
```

眼图测试命令:

```
echo test_pack > /sys/devices/platform/soc/5101000.ehci0-controller/ed_test
```

USB1 Host 眼图测试:

每个芯片平台的节点路径会有些差异, 请通过find命令确认(注意选择带有ehci1的路径):

```
find /sys -name ed_test
```

眼图测试命令:

```
echo test_pack > /sys/devices/platform/soc/4201000.ehci1-controller/ed_test
```



5 USB 相关应用工具

5.1 setusbconfig 脚本

setusbconfig 脚本位于:tina/package/base-files 目录下:

- setusbconfig, 3.4, 3.10 内核等旧平台使用。
- setusbconfig-configfs, 4.4, 4.9 及以上内核较新平台使用。

一般我们使用的都是 setusbconfig-configfs, 它里面主要封装了一些 USB gadget 功能的配置方法, 以便于切换 gadget 功能。

```
切换到adb功能:
setusbconfig adb

切换到mtp功能:
setusbconfig mtp
```

5.2 ADB

ADB 功能是从 Android 移植过来的, 设备端会运行 adbd 服务, 而 Host 端 (一般为 PC) 通过 adb 工具进行调试, 如 adb shell, adb push/pull 等。

Tina SDK 中支持 adbd 服务, 请 make menuconfig 选中对应软件包:

```
Allwinner --->
<*> adb..... adb for Tina Linux
```

选中该软件包后, 开机默认运行 adbd 服务, 并且设置 USB gadget 为 ADB 功能。

5.2.1 ADB 功能说明

5.2.1.1 adb shell 功能

PC端执行adb shell,可进入控制台。

PC端执行adb shell + command, 例如adb shell ls /可以直接将结果输出到终端。

5.2.1.2 adb push/pull 功能

推送文件到小机端：

```
adb push test.bin /mnt/UDISK/
```

从小机端拉取文件：

```
adb pull /mnt/UDISK/test.bin .
```

5.2.1.3 adb 网络连接

修改 adbd 启动脚本：

```
tina/package/utils/adb/adbd-configfs.init
```

取消下面注释

```
#ADB_TRANSPORT_PORT=5555
```

保证小机端与 PC 在同一个局域网，并且可以 ping 通。

PC端执行adb connect 192.168.1.101建立连接。

然后就可以开始执行adb shell等命令了。

5.2.1.4 adb forward 功能

adb forward 功能用于数据转发，执行该命令后，应用可以通过 socket 通信，实现 PC 与设备之间的数据交互。

PC端执行:adb forward tcp:11111 tcp:22222

tcp:11111表示PC端监听的端口。

tcp:22222表示设备端的端口。

PC端对11111端口的数据读写，都会转发到设备端22222端口

所以设备端的一些应用可以通过22222端口与PC端进行数据交互。

5.2.1.5 adb auth 功能

该功能主要用于权限管理，只有经过公私钥认证后才能进行 adb 通信。

使用前需要 make menuconfig 选中对应功能：

```
Allwinner --->
  <*> adb..... adb for Tina Linux
  <*> adbd_auth_service..... adb auth service for Tina Linux
```

默认开机就会使能该功能，执行 adb 命令时会进行公私钥校验。

公钥的安装：

```
默认允许安装所有公钥，可以通过修改tina/package/utils/adb/auth/aw_adb_auth_service.c
取消下面注释
/*aw_adbd_install_pubkey_detector(publickey_detector); */
然后在publickey_detector函数中判断是否安装公钥，返回false则不安装

公钥的安装路径：/mnt/UDISK/adb_keys
```

如果/mnt/UDISK/adb_keys 下面没有对应的公钥，并且不允许安装新公钥，这时候 adb 功能就不能使用了。

注意事项：1.Windows 下面公私钥的路径：

```
%USERPROFILE%\android\adbkey // 私钥
%USERPROFILE%\android\adbkey.pub // 公钥
```

2.Ubuntu 下面公私钥的路径：

```
用户目录/.android/adbkey // 私钥
用户目录/.android/adbkey.pub // 公钥
```

5.2.2 ADB 常见问题

- 注意是否启动了 adbd 服务

```
ps | grep adbd 确认是否存在adbd进程，如果没有请确认：
```

1. 是否选中adb软件包。
2. adbd启动脚本是否执行了。

可通过命令手动运行adbd服务：

```
/etc/init.d/adbd start
```

- 如果是 busybox 启动方式，需要注意挂载 devpts

```
adb shell的过程中，需要创建虚拟终端，依赖于devpts
mount -t devpts /dev/pts /dev/pts
```

- adb shell 不能使用，但是 push/pull 功能正常

```
adb shell会用到虚拟终端，依赖部分内核配置：
CONFIG_VT_CONSOLE
CONFIG_UNIX98_PTYS
CONFIG_DEVPTS_MULTIPLE_INSTANCES
CONFIG_CONSOLE_TRANSLATIONS
```

5.3 MTP

MTP，全称是 Media Transfer Protocol（媒体传输协议）。它是微软的一个为计算机和便携式设备之间传输图像、音乐等所定制的协议。

按照 MTP 定义的规则，可以使得电脑能够获取小机端的目录信息，并且可以进行增、删、修改、查看等操作。因此，这就要求电脑端以及小机端都需要有一个服务或者程序去发送、解析、回应数据包。

MTP 的应用分为两种角色，一个是作为 Initiator，另一个作为 Responder。

- Initiator —— 在 MTP 中所有的请求都有 Initiator 发起。例如，PC 请求获取小机设备的文件数据。
- Responder —— 它会处理 Initiator 的请求；除此之外，Responder 也会发送 Event 事件。

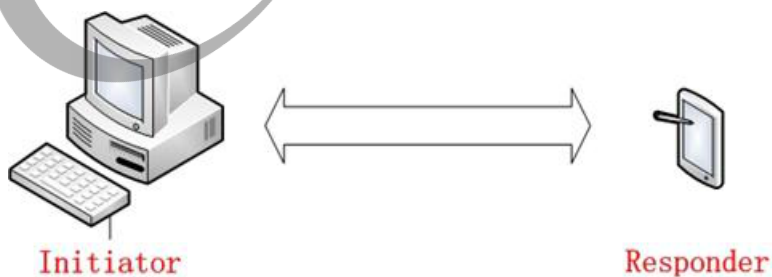


图 5-1: MTP 数据传输

一般 Windows PC 都默认支持 MTP。ubuntu 等 PC 也有一些第三方包可以支持，例如 libmtp, mtpfs 等。而小机端上则需要一个应用去处理 PC 发送过来的请求（实际上就是按照 MTP 生成的数据包），并做出回应。

Tina 参考 Android MTP 系统服务，实现了自己的 MTP 服务。

5.3.1 MTP 服务

MtpDaemon 软件设计流程图：

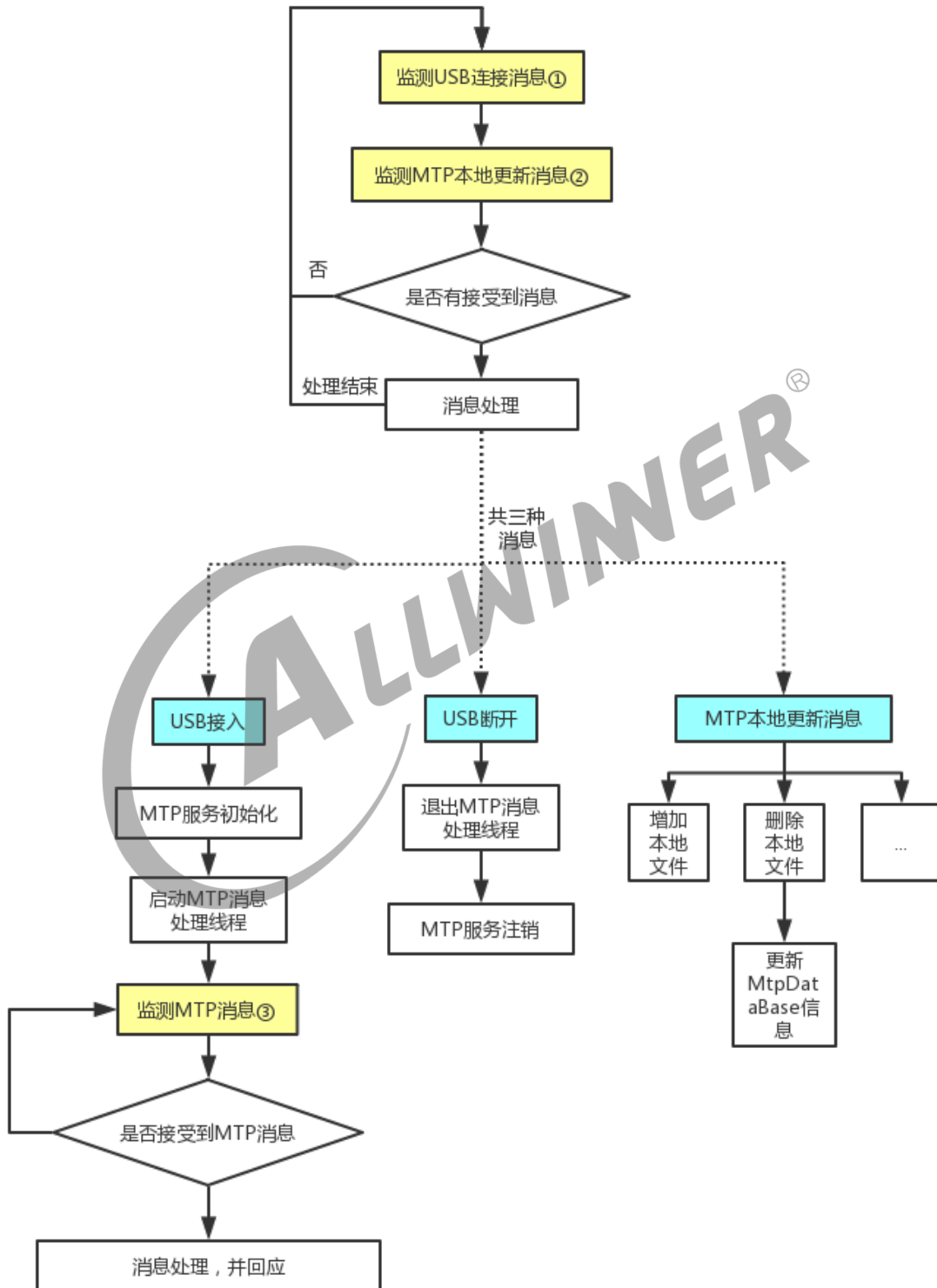


图 5-2: MtpDaemon 软件流程

- 1) 监测 usb uevent 事件
- 2) 监测 fifo 看是否接受到消息，该消息主要用于更新小机端 MtpDataBase(用于保存文件目录信息的对象)，详细请查阅MtpTools 章节。
- 3) 监测 MTP 消息，该消息是从/dev/mtp_usb 节点获取

5.3.2 MTP 相关配置

5.3.2.1 menuconfig 选中 MtpDaemon 软件包

```
Allwinner --->
<*> MtpDaemon..... Tina MTP Daemon
```

5.3.2.2 配置挂载目录

默认配置下，只会把小机端/mnt/UDISK 挂载到 PC 上，并且挂载的盘符名称为“Tina 存储设备”。如果想修改或者新增目录，可以修改下面路径的源码：

```
tina/package/allwinner/mtp/src/main.c

static mtp_storage_t gStorageArray[] = {
    {NULL, 65537, "/mnt/UDISK", "Tina存储设备", 0, 0},
    //{NULL, 65539, "/boot", "boot", 0, 0},
    //{NULL, 65540, "/boot-res", "boot-res", 0, 0},
};
```

可以在该数组上添加需要的目录及相关信息，例如

```
{NULL, 65539, "/boot", "boot", 0, 0},
```

第一个参数固定填写NULL即可，MtpDaemon会自动对这个成员赋值。

第二个参数表示Storage ID，对每个要挂载的目录都有唯一的ID，例如上面设置的65537为/mnt/UDISK的storage ID，这个值是一个32bit的整型数，建议从65537累加上去即可。

第三个参数表示要挂载到PC的目录，是一个绝对路径。

第四个参数表示保留空间(暂时无用)。

第五个参数表示该单个文件大小的最大值(暂时无用)。

5.3.3 MtpTools

MtpDaemon 作为 responder 处理并回应 PC 发送过来的 MTP 消息，因此在 PC 上可以对小机端的目录、文件进行操作。但为了让小机端的修改能够同步到 PC 上，Tina 提供了 MtpTools。

MtpTools 主要通过 fifo 发送消息给 MtpDaemon，然后 MtpDaemon 根据该消息去更新 Mtp-DataBase，即更新用于描述目录文件信息的对象，此时 PC 上只需要刷新下即可同步得到小机端的修改。

下面介绍下 MtpTools 的几个命令：

```
MtpTools usage:
MtpTools [function] [path]
-f, --function      function option, contains add,remove,update,cut,copy
-t, --type          object type, FILE or DIR
-s, --spath         object src path
-d, --dpath         object dest path, specify dest path of object which will be cut or
                    copy
-h, --help          show help

example:
MtpTools -f add -t FILE -s /mnt/UDISK/test
```

参数	功能说明
-f	指定要执行的动作，包含 add-新增, remove-移除, update-更新, cut-剪切, copy-复制
-t	指定要操作对象的类型，文件 or 目录
-s	表示操作对象的源路径
-d	表示操作对象的目标路径

举例说明：

新增一个名为 test 的文件：

```
MtpTools -f add -t FILE -s /mnt/UDISK/test
```

实际上有一种比较简单的方法，就是直接更新目录信息，例如：

```
MtpTools -f update -t DIR -s /mnt/UDISK
```

这样不管该目录下进行了任何修改，全都会更新一遍。（他会遍历更新所有目录、文件信息，相对耗时多一些）。

5.3.4 MTP 常见问题

- 开机后 MTP 功能没法正常使用

1. ps | grep Mtp 确认 MtpDaemon 是否运行。
2. setusbconfig mtp 切换到 MTP 功能。
3. 确认 Windows 端 USB 驱动是否正常识别。
如果没有识别到，可以按下面步骤手动加载驱动：
选择“浏览计算机以查找驱动程序软件” ->
“从计算机的设备驱动程序列表中选择” ->
“便携设备”
“标准 MTP 设备” ->
“MTP USB 设备”
4. 修改盘符名称。

默认是hostname(即TinaLinux)，可以在src/MtpServer.c中定义MODEL_NAME宏去设定其他盘符名称。

ALLWINER®



著作权声明

版权所有 © 2021 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

、 **全志科技** （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。