# H616 USB 模块说明书

2.5

2019.12.30

# 文档履历

| 版本号 | 日期 | 制/修订人 | 内容描述 |
|---|---|---|---|
| 1.0 | 2016.12.03 | AWA0001 | |
| 2.0 | 2018.12.21 | AWA1458 | 修正一些参数，增加只支持 device 模式的配置说明 |
| 2.1 | 2019.01.29 | AWA1430 | 增加一些 USB 说明 |
| 2.2 | 2019.03.01 | AWA1430 | 增加 menuconfig 配置说明 |
| 2.3 | 2019.09.12 | AWA1430 | 针对 H3 配置做修改 |
| 2.4 | 2019.11.14 | AWA1430 | 针对 AW1823 配置做修改 |
| 2.5 | 2019.12.30 | AWA1430 | 修改‘USB 系统概念简介’章节的标点符号使用 |

# 目录

# 1. 概述

## 1.1 编写目的

介绍 USB 模块配置方法，以及使用注意事项。

## 1.2 相关人员

sdk 维护人员、USB 驱动开发者、系统配置人。

# 2. 模块功能介绍

## 2.1 相关术语介绍

| 术语 | 解释说明 |
| --- | --- |
| HCD | 主机控制器驱动 |
| URB | USB Request Block 用于组织每一次的 USB 设备驱动的数据传输请求 |
| EP | Endpoint 端点 |
| Mass Storage | I2C Core 将所有 I2C 控制器成为 I2C 适配器，可以理解成控制器的软件名称 |
| ADB | Android Debug Bridge，Android 调试桥接器 |
| MTP | Media Transfer Protocol，媒体传输协议 |
| PTP | picture transfer protoco，图片传输协议 |

## 2.2 USB 系统概念简介

由于 USB 系统由两部组成，所以 usb 驱动也由相应两部分组成。

主机侧驱动主要由应用层驱动，用于管理 hub 和 hcd 的 usb core 层驱动，主机控制器驱动这三部分组成，应用层驱动主要是通过 usb 的基本数据通信结构 urb 对 usb 设备进行操作，这类驱动是针对某一个 usb 接口写的，所以也叫作 usb 接口驱动，通常在主机端驱动人员要写的就是这类驱动。usb core 驱动里包含一个守护进程，通常进程是休眠的，当 usb port 上产生变化，usb root hub 监控到其端口上有设备插入或拔出，它就会去唤醒守护进程，然后运行 usb 设备的枚举，进而运行后面操作。主机控制器驱动里包含两部分，一部分是和 usb 接口标准相关的驱动，一部分是和控制器本身相关的，其中和接口标准相关的驱动占主要部分。

在 usb 设备侧，驱动由三层组成：Upper Layers，gadget drivers, device controller drivers。upper layers 属于应用层，通过 gadget drivers 来使用和控制 device controllers；gadget drivers 使用 gadget API，实现与具体硬件无关；device controller drivers 提供 gadget API，实现 gadget ops 和 endpoint ops。

不管是在主机侧还是设备侧，一个 usb device 它主要可由配置、接口、端口三部分表示。在主机侧它们分别由 usb_host_config、usb_inferface、usb_host_endpoint 表示。而设备侧由 usb_configuration、usb_function 表示。像我们现在的智能手机它可以用来拍照也可以用来当 U 盘，这两种就属于不同的配置。一个 usb 设备它可能包含多个配置，在使用某些功能前必需选择好相应的配置。一个配置由多个接口组成，一个接口表示一个功能，一个接口里可以包含多个接口设置。每个接口设置里包含与实

现某一功能相关的端口，配置、接口和端口结构示意如下图所示。

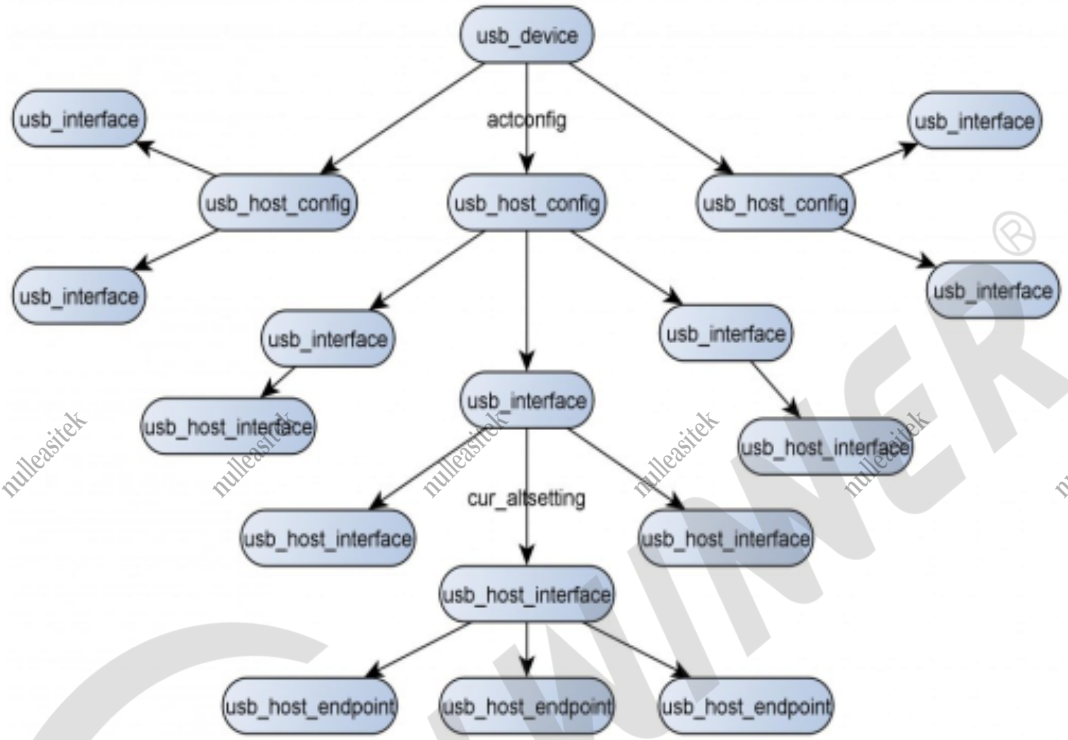


图 1: 接口和端口结构示意图

当一个 usb 设备插入到 host 端时，主机端的 root hub 通过轮训监控或中断方式来触发 usb 枚举。当 usb 设备通过枚举后，通过 usb 总线找到与其配置驱动，然后 usb 设备才能正常工作，这个枚举过程由主机和设备共同完成。对于没有运行操作系统的 usb 设备，生产产家已经把设备侧枚举所需过程固化在设备里，驱动工程师可不用去关心设备枚举，他们只需要完成 usb 接口驱动程序即可。但对于有运行操作系统的 usb 设备就完全不一样了，驱动工程师不仅要实现 usb 设备功能驱动，还得实现与主机枚举过程相对应的驱动。如主机在对 usb 设备进行复位操作后，主机会发送 usb 请求去设置 usb 设备的地址，相应的 usb 设备侧必须实现设备地址机制，并返回 0 长度 usb 请求，做为收到数据 ACK。当主机要获取 usb 设备各种描述符或设置各种配置时，usb 设备侧也要实现相应操作。

## 2.3 USB 驱动程序框架

Linux 内核提供了完整的 USB 驱动程序框架。USB 总线采用树形结构，在一条总线上只能有唯一的主机设备。Linux 内核从主机和设备两个角度观察 USB 总线结构。下图是 Linux 内核从主机和设备两个角度观察 USB 总线结构的示意图。

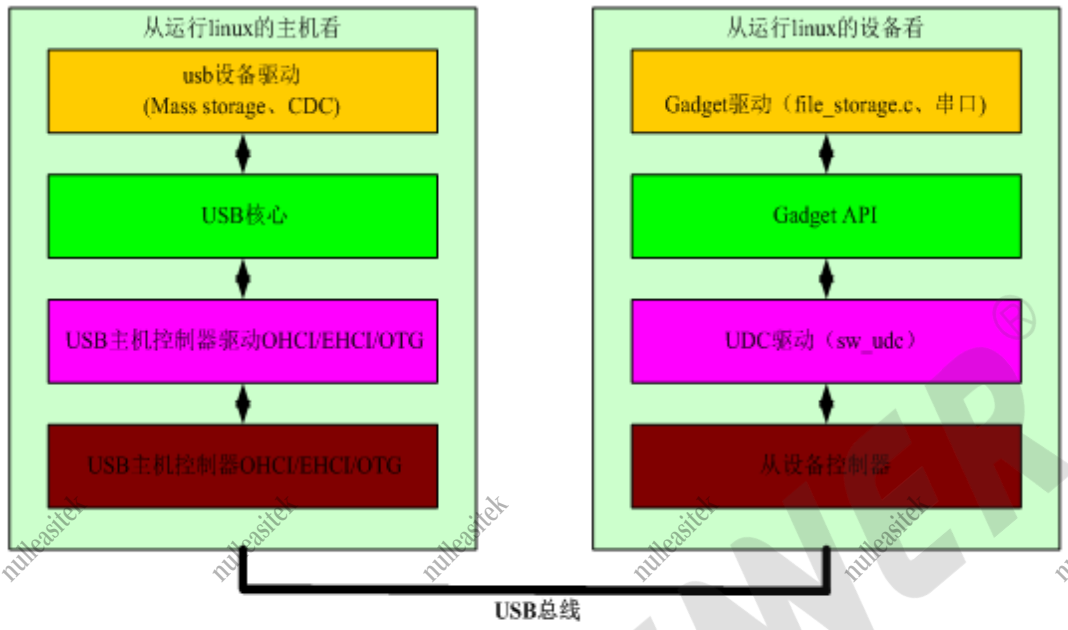


图 2: usb 驱动总体结构

USB 子系统主要任务包括：

a. 注册和管理设备驱动；

b. USB 设备寻找驱动，并初始化和配置设备；

c. 内核中表现设备的树形结构；

d. 与设备交互。

# 3. 模块配置介绍

## 3.1 menuconfig 配置

在命令行中进入内核根目录，执行 make ARCH=arm menuconfig（64 位平台为 make ARCH=arm64 menuconfig）进入配置主界面，并按以下步骤操作：

首先，选择 Device Drivers 选项进入下一级配置，如下图所示：

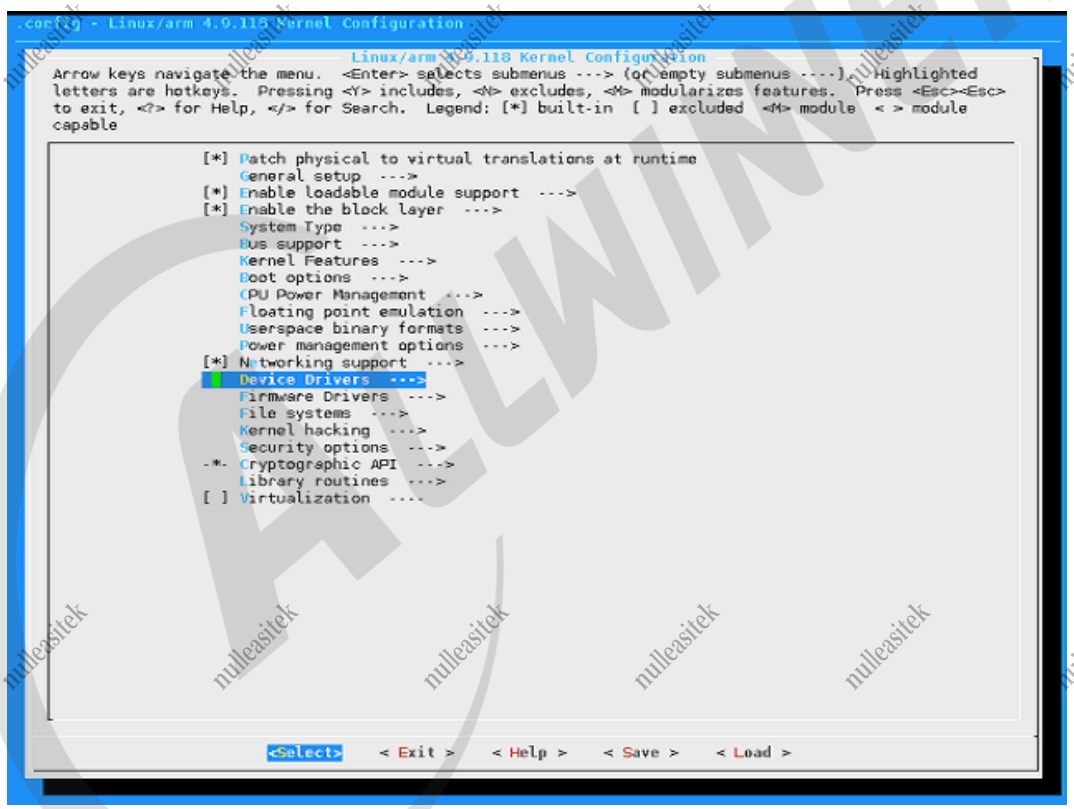


图 3: Device Drivers 选项配置

然后，选择 USB support 选项，进入下一级配置，如下图所示：

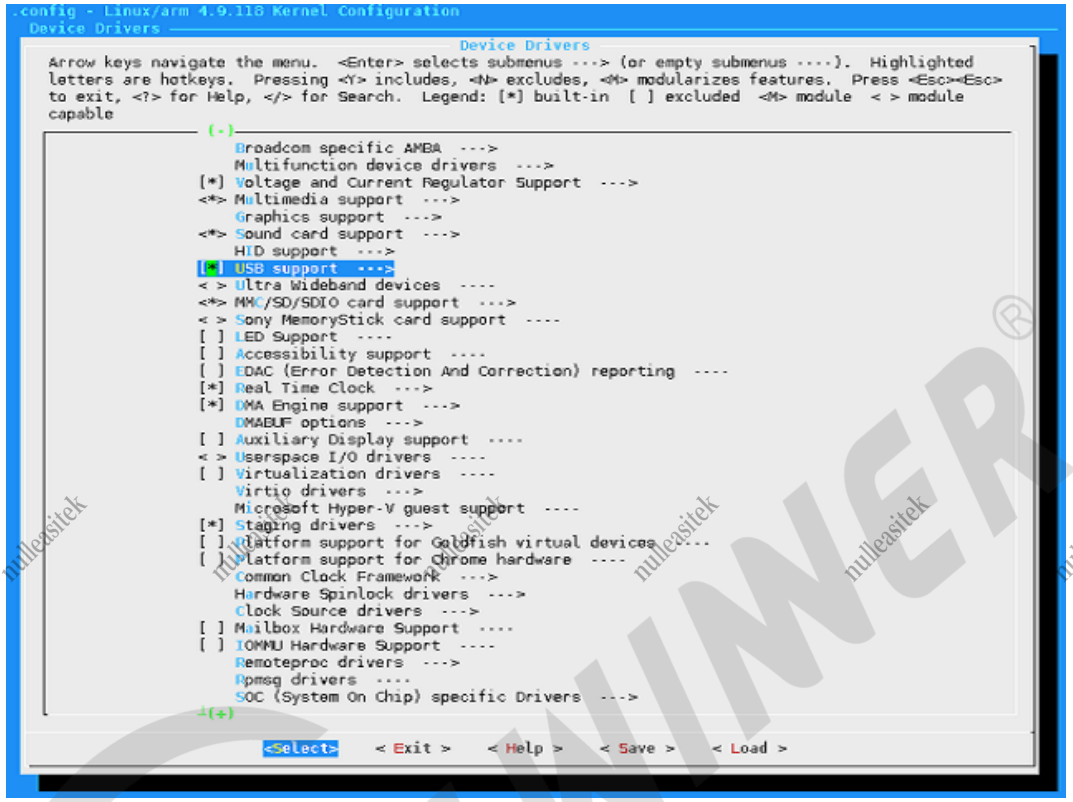


图 4: USB support 选项配置

打开如下两图的选项，如下图所示：

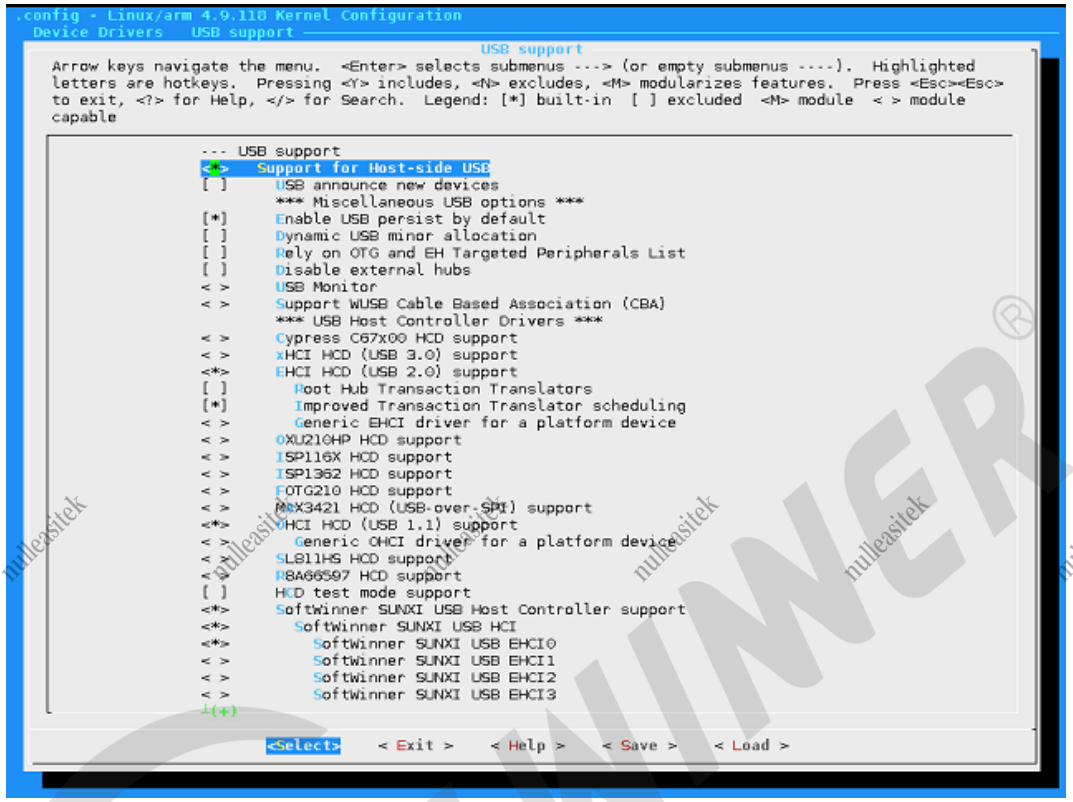


图 5: USB support 详细配置 1

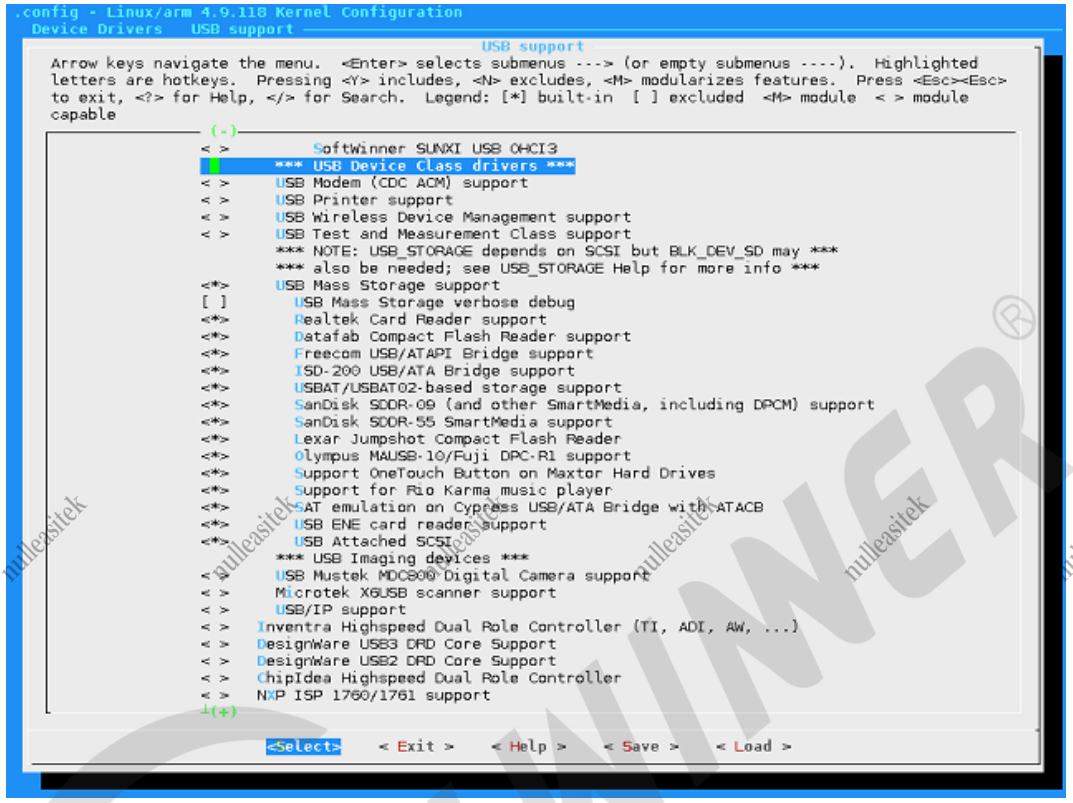


图 6: USB support 详细配置 2

然后，选择 USB Gadget Support，进入下一级配置，如下图所示:

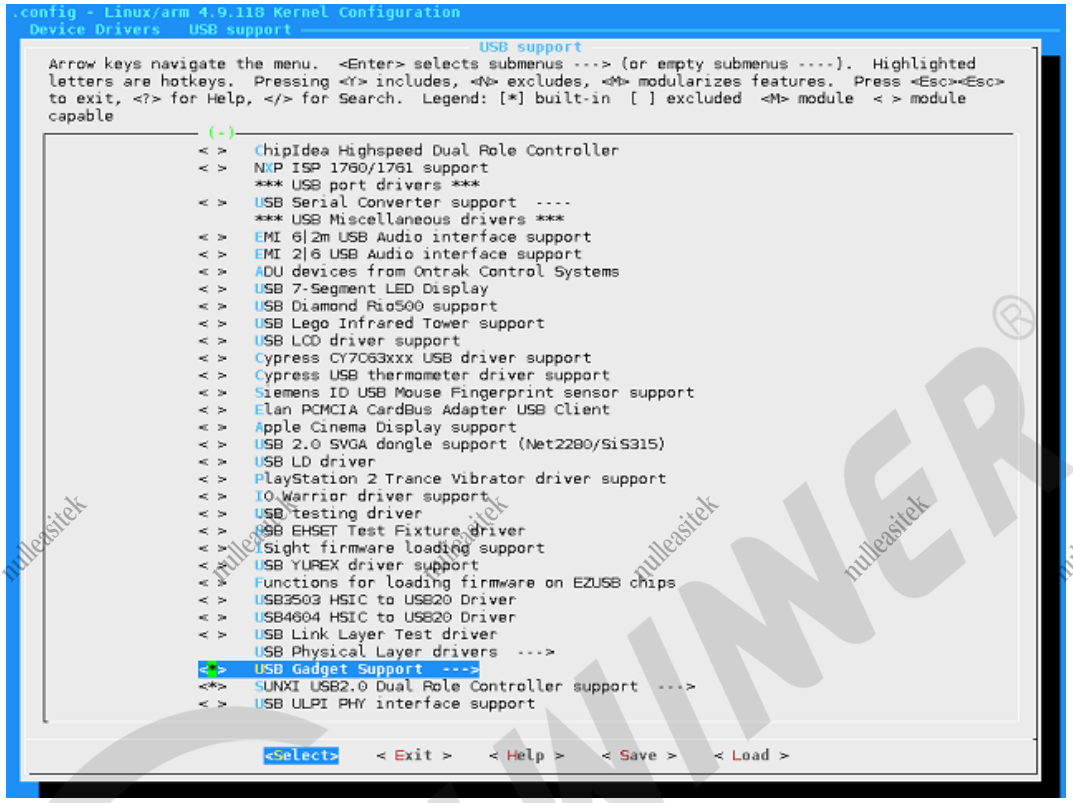


图 7: USB Gadget Suppor 选项配置

打开下图的选项，如下图所示：

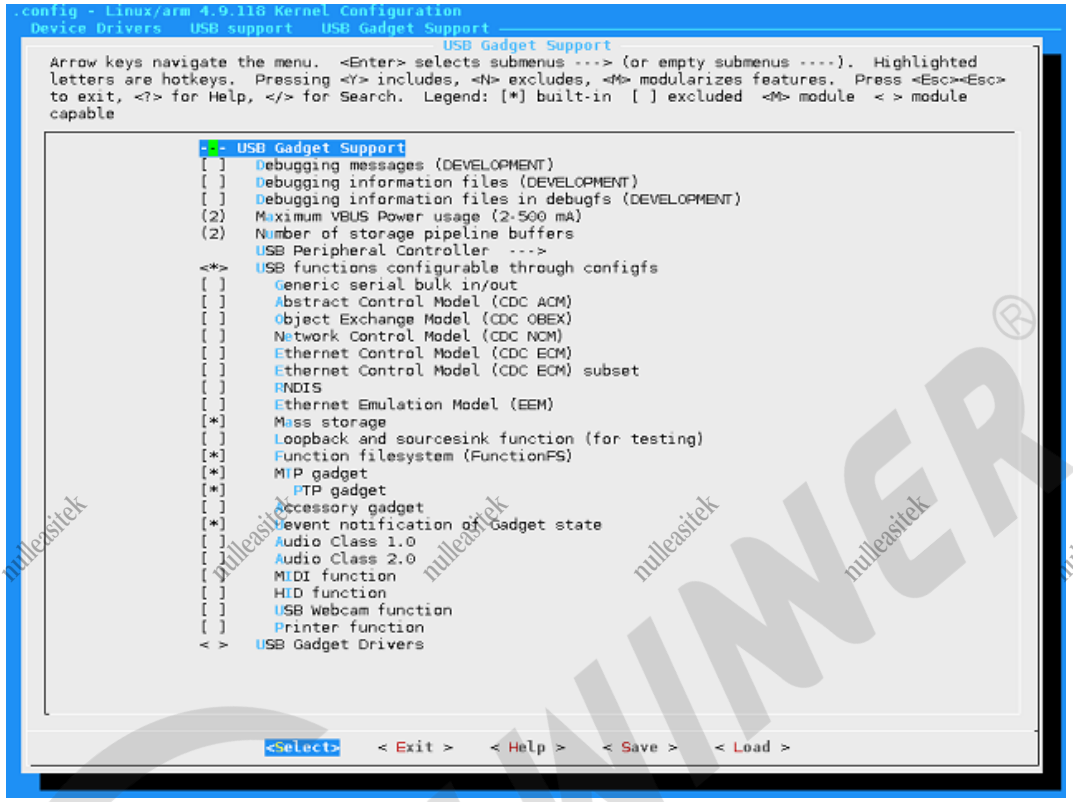


图 8: USB Gadget Suppor 详细配置

然后，返回上一级，即 USB support，进入 SUNXI USB2.0 Dual Role controller support，并打开下图选项，如下图所示：

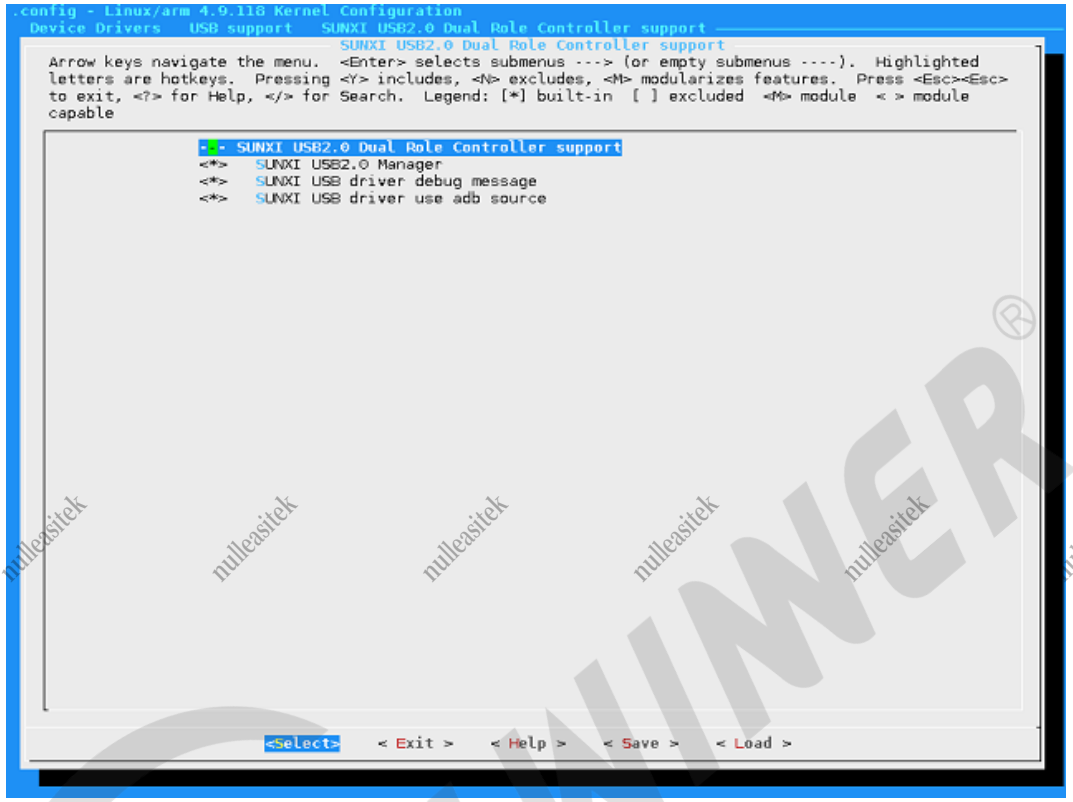


图 9: SUNXI USB2.0 Dual Role controller support 详细配置

# 3.2 board.dts 配置

## 3.2.1 usbc0

因为 usb0 otg device 控制器与标准 hci 复用为 usb0 的接口，所以 usbc0 为 hci 和 device 的总体。

```
/* ---------------------------------
 *[usbc0]：控制器0的配置。
 *usb_used：USB使能标志。置1，表示系统中USB模块可用,置0,则表示系统USB禁用。
 *usb_port_type：USB端口的使用情况。 0：device only;1：host only;2：OTG
 *usb_detect_type：USB端口的检查方式。0：不做检测;1：vbus/id检查;2：id/dpdm检查
 *usb_id_gpio：USB ID pin脚配置。具体请参考gpio配置说明。
 *usb_det_vbus_gpio：USB DET_VBUS pin脚配置。具体请参考gpio配置说明。
 *usb_drv_vbus_gpio：USB DRY_VBUS有两种配置。(1)gpio方式;(2)"axp_ctrl",表示axp 提供接口。
 *usb_det_vbus_gpio: "axp_ctrl",表示axp 提供
```

```
    *----------------------------------
    *----------------------------------
    *--- USB0控制标志
    *----------------------------------
    */
  usbc0:usbc0@0 {
    device_type = "usbc0";
    usb_port_type = <0x0>
    usb_detect_type = <0x0>
    usb_detect_mode = <0x0>
    usb_id_gpio;
    usb_det_vbus_gpio;
    usb_drv_vbus_gpio;
    usb_host_init_state = <0x0>
    usb_regulator_io = "nocare"
    usb_wakeup_suspend = <0x0>
    usb_luns = <0x3>
    usb_serial_unique = <0x0>
    usb_serial_number = "20080411"
    status = "okay"
  }
```

1. usbc_used USB 使能标志。置 1，表示系统中 USB 模块可用, 置 0，则表示系统 USB 禁用
2. usb_port_type USB 端口的使用类型；0：device only；1：host only；2：OTG。
3. usb_detect_type USB 端口的检查方式。0：不做检测；1：vbus/id 检查；2：通过 id/dpdm 检测。
4. usb_id_gpio USB ID pin 脚配置。
5. usb_det_vbus_gpio USB DET_VBUS pin 脚配置。
6. usb_drv_vbus_gpio USB DRY_VBUS pin 脚配置。
7. usb_host_init_state 表示系统起后是否初始 host 控制器驱动；0：不初始化，1：初始化，如 usb_port_type 为 device 或者 otg 时，应设为 0。
8. usb_regualtor_io usb 控制器的供电域。
9. usb_regulator_vol usb 控制器的供电电压。
10. usb_wakeup_suspend 表示是否支持 usb 远程唤醒，1 为支持，0 为不支持。

# 4. 动态调试节点

不管 usb0 的 otg 配置怎么样，应用和串口都可以通过操作 usb 的角色管理节点，进行动态切换，方便调试。切换角色：

```
device：cat sys/devices//platform/soc/usbc/usb_device
host： cat sys/devices/platform/soc/usbc/usb_host
```

另外可以通过 otg_role 查看当前的状态。

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

# 5. usb 配置举例

## 5.1 otg

如存在方案中存在 id pin 时，可以把 otg 配置为 otg 功能，软件能通过对 id pin 的自动检测切换 otg 角色。当接上 otg 线时，自动切换到 host，当接 usb 线到 pc 时，自动切换到 device. 配置如下：

```
usbc0:usbc0@0 {
    usb_port_type = <0x2>
    usb_detect_type = <0x1>
    usb_id_gpio = port:PH12<0><1><default><default>
    usb_det_vbus_gpio = "axp_ctrl"
    usb_drv_vbus_gpio = "axp_ctrl"
    usb_host_init_state = <0x0>
    usb_regulator_io = "nocare"
    usb_regulator_vol = <0x0>
    usb_wakeup_suspend = <0x0>
    status = "okay"
}
```

## 5.2 host

该配置直接将 usb0 配为 host，使用时不具备 otg 与 device 功能，配置如下：

```
usbc0:usbc0@0 {
    usb_port_type = <0x1>
    usb_detect_type = <0x1>
    usb_id_gpio = port:PH12<0><1><default><default>
    usb_det_vbus_gpio = "axp_ctrl"
    usb_drv_vbus_gpio = "axp_ctrl"
    usb_host_init_state = <0x1>
    usb_regulator_io = "nocare"
    usb_regulator_vol = <0x0>
    usb_wakeup_suspend = <0x0>
    status = "okay"
}
```

## 5.3 device

该配置直接将 usb0 配为 device，使用时不具备 otg 与 host 功能，配置如下：配置如下：

```
usbc0:usbc0@0 {
   usb_port_type = <0x0>
   usb_detect_type = <0x1>
   usb_id_gpio = port:PH12<0><1><default><default>
   usb_det_vbus_gpio = "axp_ctrl"
   usb_drv_vbus_gpio = "axp_ctrl"
   usb_host_init_state = <0x0>
   usb_regulator_io = "nocare"
   usb_regulator_vol = <0x0>
   usb_wakeup_suspend = <0x0>
   status = "okay"
}
```

## 5.4 其他 usb 口配置

其他 usb 口（非 usb0），配置项与 usb0 有区别，各配置项如下：

```
/* --------------------------------
 *--- USB1控制标志
 *--------------------------------
usbc1:usbc1@0 {
   usb_drv_vbus_gpio = port:PI1<0><1><default><default>
   usb_host_init_state = <0x1> //是否初始host控制器驱动,0:不初始化，1：初始化
   usb_regulator_io = "nocare"
   usb_wakeup_suspend = <0x0>
   status = "okay"
}
```

# 6. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner. The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This document neither states nor implies warranty of any kind, including fitness for any particular application.tates nor implies warranty of any kind, including fitness for any particular application.