StarFive 赛昉科技

JH7110 USB Developing Guide

VisionFive 2

Version: 1.1

Date: 2023/8/28

Doc ID: JH7110-DGEN-012

Legal Statements

Important legal notice before reading this documentation.

PROPRIETARY NOTICE

Information in this document is provided "as is," with all faults. Contents may be periodically updated or revised due to product development. Shanghai StarFive Technology Co., Ltd. (hereinafter "StarFive") reserves the right to make changes without further notice to any products herein.

StarFive expressly disclaims all warranties, representations, and conditions of any kind, whether express or implied, including, but not limited to, the implied warranties or conditions of merchantability, fitness for a particular purpose, and non-infringement.

StarFive does not assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation indirect, incidental, special, exemplary, or consequential damages.

Contact Us

Address: Room 502, Building 2, No. 61 Shengxia Rd., China (Shanghai) Pilot Free Trade Zone, Shanghai, 201203, China

Website: http://www.starfivetech.com

Email:

- Sales: sales@starfivetech.com
- Support: support@starfivetech.com

Contents

List of Tables

List of Figures

Preface

About this guide and technical support information.

About this document

This document mainly provides the SDK developers with the programing basics and debugging know-how for the USB of the StarFive next generation SoC platform - JH7110.

Audience

This document mainly serves the USB relevant driver developers. If you are developing other modules, place a request to your sales or support consultant for our complete documentation set on JH7110.

Revision History

Table 0-1 Revision History

Version	Released	Revision
1.1	2023/8/28	Added USB Overcurrent Debug *(on page 22)*.
1.0	2022/11/10	First official release.

Notes and notices

The following notes and notices might appear in this guide:

- **Tip:**
  Suggests how to apply the information in a topic or step.
- **Note:**
  Explains a special case or expands on an important point.
- **Important:**
  Points out critical information concerning a topic or step.

- **CAUTION:**
  Indicates that an action or step can cause loss of data, security problems, or performance issues.

- **Warning:**
  Indicates that an action or step can result in physical harm or cause damage to hardware.

1. Introduction

Universal Serial Bus (USB) was promoted by Intel, Microsoft and other manufacturers in 1995, to solve the contradiction between the increasing variety of computer peripherals and the limited motherboard slots and ports. It has the advantages of high speed data transmission rate, easy expansion, support for plug and play and support for hot plug. The standard has been widely used in SoC platforms.

1.1. Function Introduction

The JH7110 SoC Platform supports the following features and specifications on the USB.

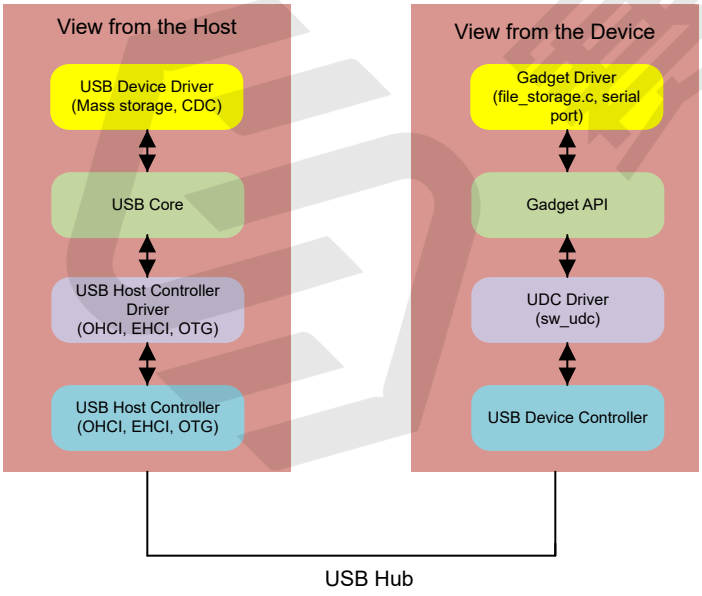
- As the master device, connect mass storage devices, USB mouse and other USB devices.
- As the slave device, perform ADB debugs and mass storage devices.

1.2. Driver Framework

Linux kernel provides a complete set of USB driver programs. The USB framework uses a tree structure and only allows to have one host device on a bus.

The following diagram shows the USB framework from the host or slave's point of view.

Figure 1-1 Driver Framework



The main tasks of the USB subsystem include:

- Register and manage device drivers
- Locate drivers for USB devices, then initialize and configure the device

- Display device tree in the kernel
- Interact with devices

1.3. Device Tree Overview

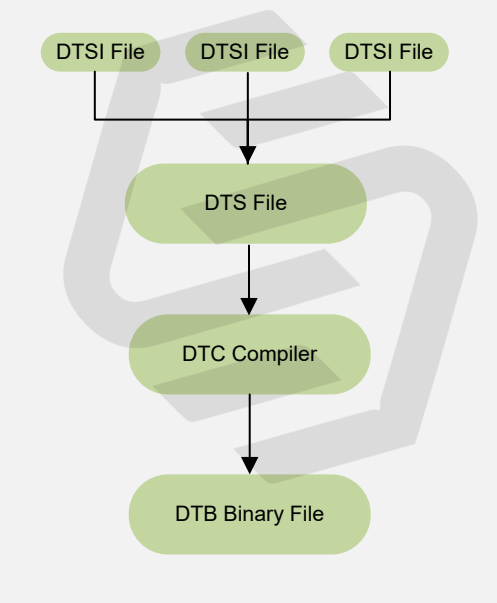
Since Linux 3.x, device tree is introduced as a data structure and language to describe hardware configuration. It is a system-readable description of hardware settings so that the operating system doesn't have to hard code details of the machine.

A device tree is primarily represented in the following forms.

- *Device Tree Compiler (DTC)*: The tool used to compile device tree into system-readable binaries.
- *Device Tree Source (DTS)*: The human-readable device tree description file. You can locate the target parameters and modify hardware configuration in this file.
- *Device Tree Source Information (DTSI)*: The human-readable header file which you can include in device tree description. You can locate the target parameters and modify hardware configuration in this file.
- *Device Tree Blob (DTB)*: The system-readable device tree binary blob files which is burned in system for execution.

The following diagram shows the relationship (workflow) of the above forms.

Figure 1-2 Device Tree Workflow



1.4. Device Tree

A DTS/DTSI file is used to store all the device tree configuration.

The device tree of USB is stored in the following path:

```
linux-5.15/arch/riscv/boot/dts/starfive/
```

The following is an example of the USB configuration in the file `jh7110-common.dtsi`.

```
&usbdrd30 {
        pinctrl-names = "default";
        pinctrl-0 = <&usb_pins>;
        dr_mode = "host"; /*host or peripheral*/
        status = "disabled";
 };
```

The following is an example of the USB configuration in the file `jh7110.dtsi`.

```
usbdrd30: usbdrd{
compatible = "starfive,jh7110-cdns3";
         reg = <0x0 0x10210000 0x0 0x1000>,
                   <0x0 0x10200000 0x0 0x1000>;
         clocks = <&clkgen JH7110_USB_125M>,
                      <&clkgen JH7110_USB0_CLK_APP_125>,
                      <&clkgen JH7110_USB0_CLK_LPM>,
                       <&clkgen JH7110_USB0_CLK_STB>,
                       <&clkgen JH7110_USB0_CLK_USB_APB>,
                       <&clkgen JH7110_USB0_CLK_AXI>,
                       <&clkgen JH7110_USB0_CLK_UTMI_APB>,
                       <&clkgen JH7110_PCIE0_CLK_APB>;
           clock-names = "125m","app","lpm","stb","apb","axi","utmi", "phy";
           resets = <&rstgen RSTN_U0_CDN_USB_PWRUP>,
                        <&rstgen RSTN_U0_CDN_USB_APB>,
                        <&rstgen RSTN_U0_CDN_USB_AXI>,
                        <&rstgen RSTN_U0_CDN_USB_UTMI_APB>,
                         <&rstgen RSTN_U0_PLDA_PCIE_APB>;
             reset-names = "pwrup","apb","axi","utmi", "phy";
             starfive,stg-syscon = <&stg_syscon 0x4 0xc4 0x148 0x1f4>;
             starfive,sys-syscon = <&sys_syscon 0x18>;
             status = "disabled";
             #address-cells = <2>;
             #size-cells = <2>;
             #interrupt-cells = <1>;
             ranges;
             usbdrd_cdns3: usb@10100000 {
                   compatible = "cdns,usb3";
                   reg = <0x0 0x10100000 0x0 0x10000>,
                              <0x0 0x10110000 0x0 0x10000>,
                              <0x0 0x10120000 0x0 0x10000>;
                    reg-names = "otg", "xhci", "dev";
                    interrupts = <100>, <108>, <110>;
                    interrupt-names = "host", "peripheral", "otg";
                    phy-names = "cdns3,usb3-phy", "cnds3,usb2-phy";
```

```
                    maximum-speed = "super-speed";
            };
};
```

The following list provides an example of the parameters contained in the code block above.

- **reg**: The basic address and the max offset of the module.
- **clocks**: The clocks used by the module. Usually, this parameter is filled with the system clock source or the clock of the module.
- **resets**: The reset items used by the module.
- **starfive,stg-syscon**: The register values for the StarFive STG SYSCON control registers.
- **starfive,sys-syscon**: The register values for the StarFive SYS SYSCON control registers.
- **status**: The status of the module.
    - **okay**: The module is enabled.
    - **disabled**: The module is disabled.
- **other**: Other parameters for USB registration and association.

2. Configuration

2.1. VisionFive 2 Board Level Configuration

The board level DTS file stores all identification information for each board.

The board level DTS files are stored in the following path:

```
linux-5.15/arch/riscv/boot/dts/starfive/jh7110-visionfive-v2.dts
linux-5.15/arch/riscv/boot/dts/starfive/jh7110-visionfive-v2.dtsi
```

The following code block provides an example of the USB configuration in the board level DTS file.

```
&usbdrd30 {
 pinctrl-names = "default";
 pinctrl-0 = <&usb_pins>;
 dr_mode = "host"; /*host or peripheral*/
 status = "disabled";
};
```

In the above code block, the parameter **dr_mode** represents the default mode of the USB port, and the following options are available.

- **peripheral**: Device mode as the slave device.
- **host**: Host mode as the master device.

Note:

If you are using the USB standby mode, note the following:

- Your IC should be able to support remote wakeup.
- Follow the *JH7110 Hardware Design Reference* strictly when designing your device.
- Add an additional parameter named "**wakeup-source**" in the above code block and enable the USB standby function.

2.2. VisionFive 2 Pin Control Configuration

The pin control configuration for USB host is stored in the following file:

```
linux-5.15/arch/riscv/boot/dts/starfive/jh7110-visionfive-v2.dtsi
```

The following code block provides an example of the content details.

```
usb_pins: usb-pins {
        drive-vbus-pin {
                sf,pins = <PAD_GPIO25>;
                sf,pinmux = <PAD_GPIO25_FUNC_SEL 0>;
                sf,pin-ioconfig = <IO(GPIO_IE(1))>;
                sf,pin-gpio-dout = <GPO_USB0_DRIVE_VBUS_IO>;
                sf,pin-gpio-doen = <OEN_LOW>;
        };
};
```

2.3. Kernel Menu Configuration

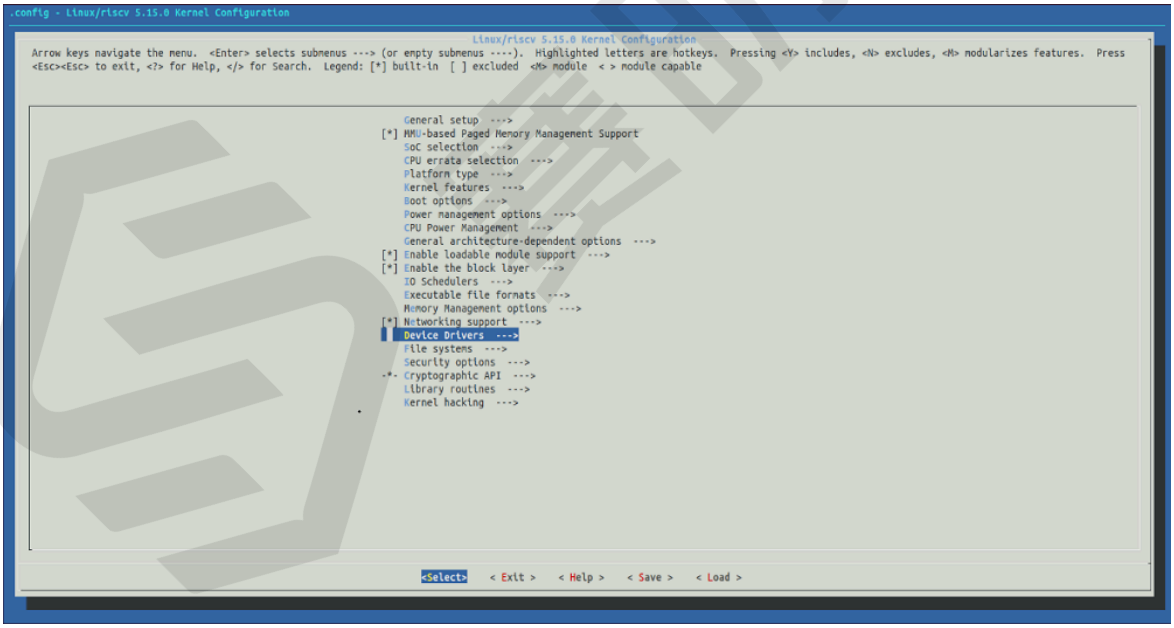
Follow the steps below to enable the kernel configuration for USB.

1. Under the root directory of `freelight-u-sdk`, type the following command to enter the kernel menu configuration GUI.

   ```
   make linux-menuconfig
   ```

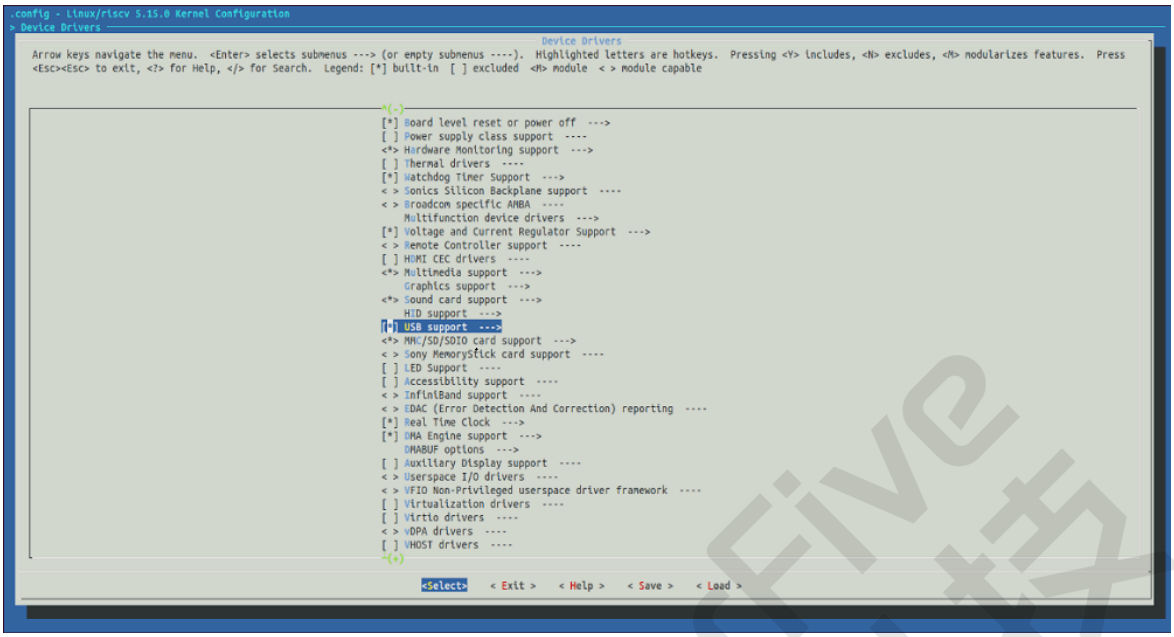
2. Enter the **Device Drivers** menu.

   Figure 2-1 Device Drivers

3. Enter the **USB Support** menu.

Figure 2-2 USB Support



4. In the **USB Support** menu, select **support for host-side USB**, **PCI based USB host interface** and **xHCI HCD USB 3.0 support** options.

Figure 2-3 Host-Side USB and PCI based USB

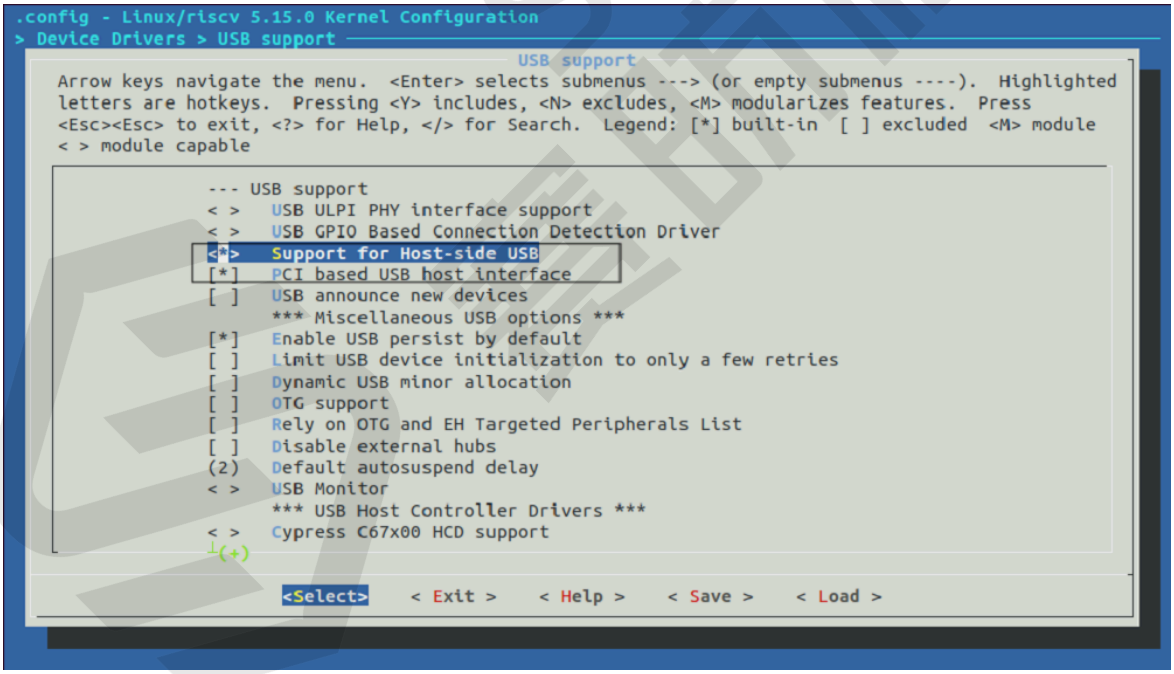
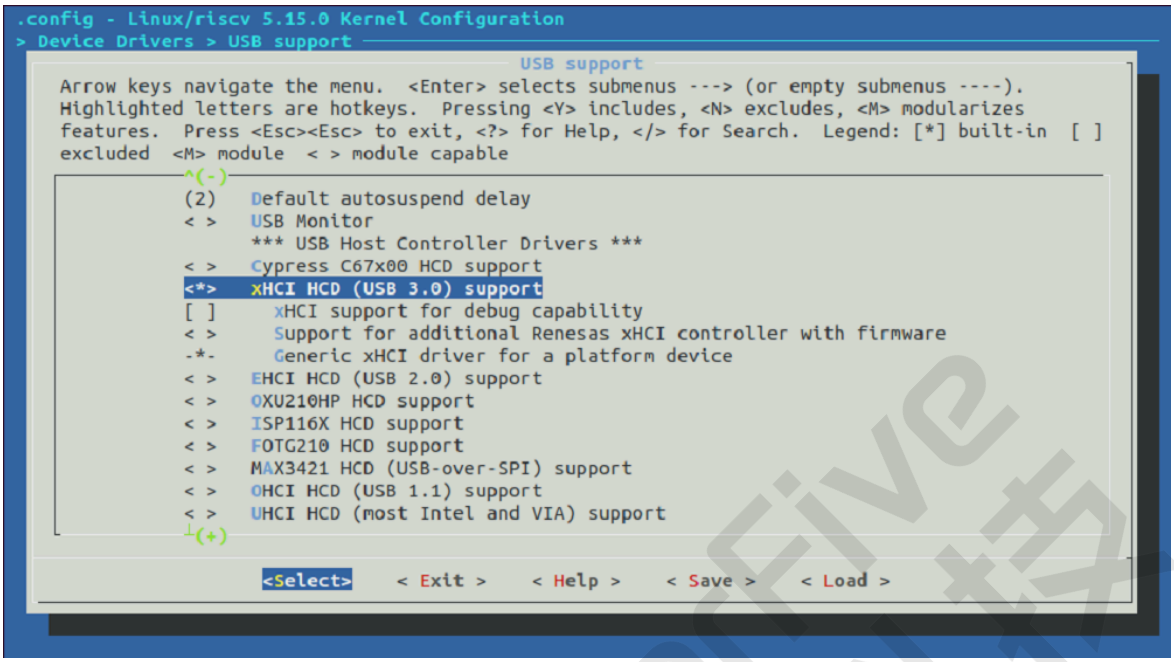


Figure 2-4 xHCI HCD USB 3.0 support

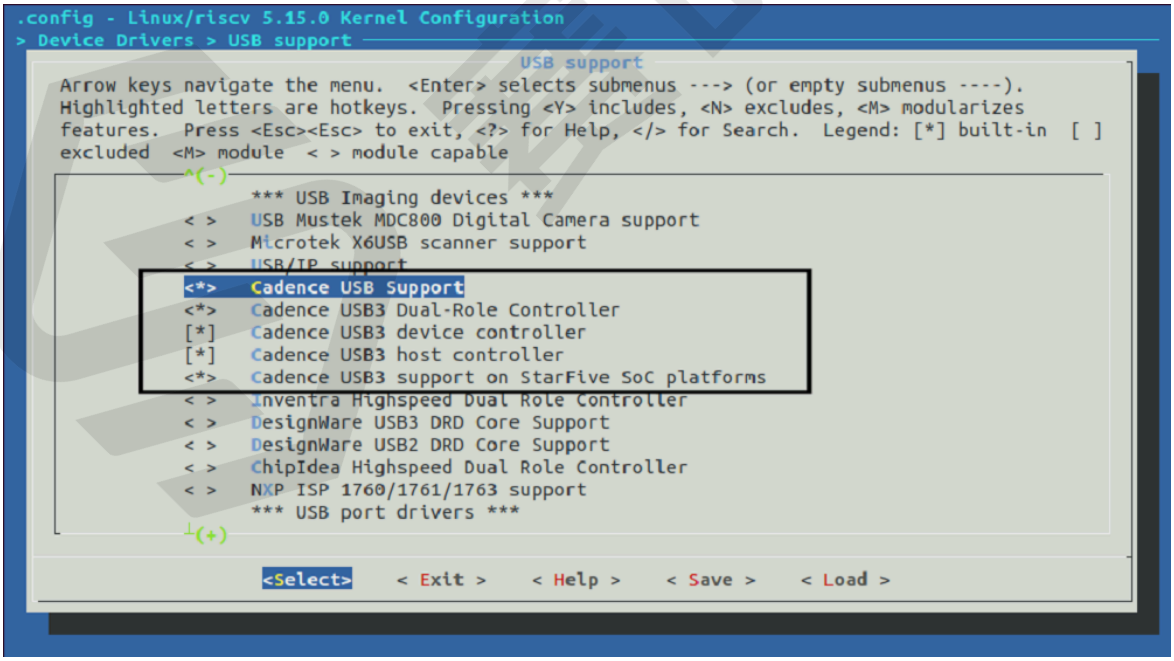


Note:

JH7110 supports one USB port. One of the PCIe2.0 lanes can be shared by USB3.0. So when enabling USB 3.0, you should disable the reused PCIe port.

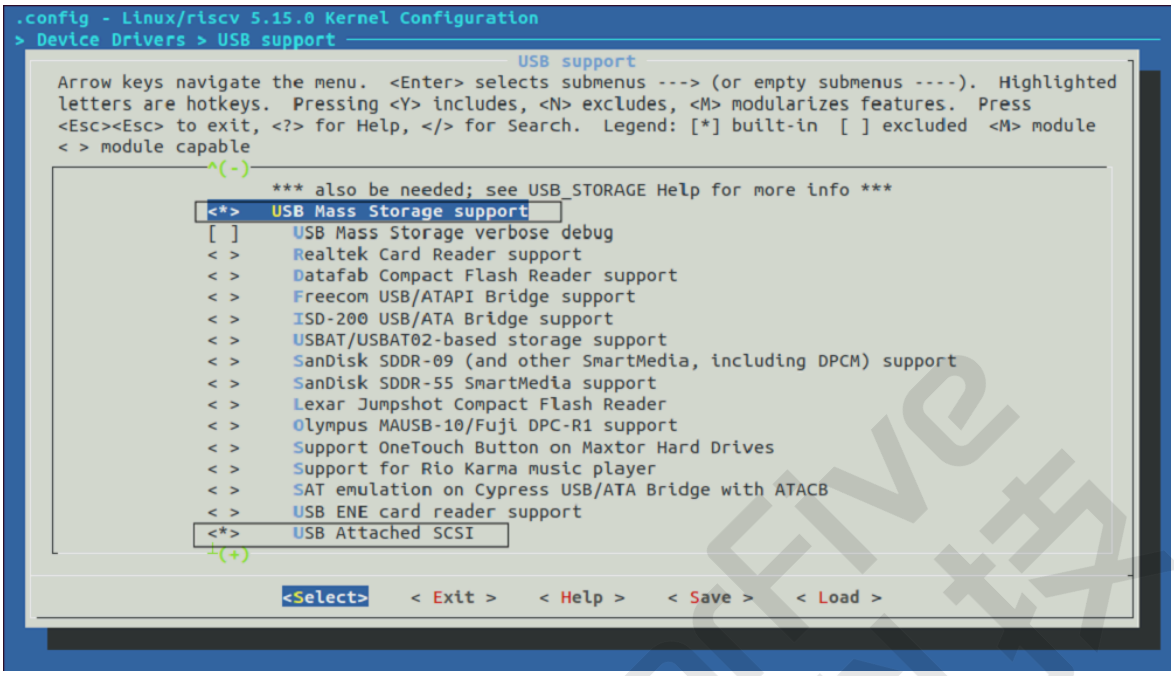
5. In the **USB Support** menu, select all the options starting with **Cadence USB**.

Figure 2-5 Cadence USB Support Options



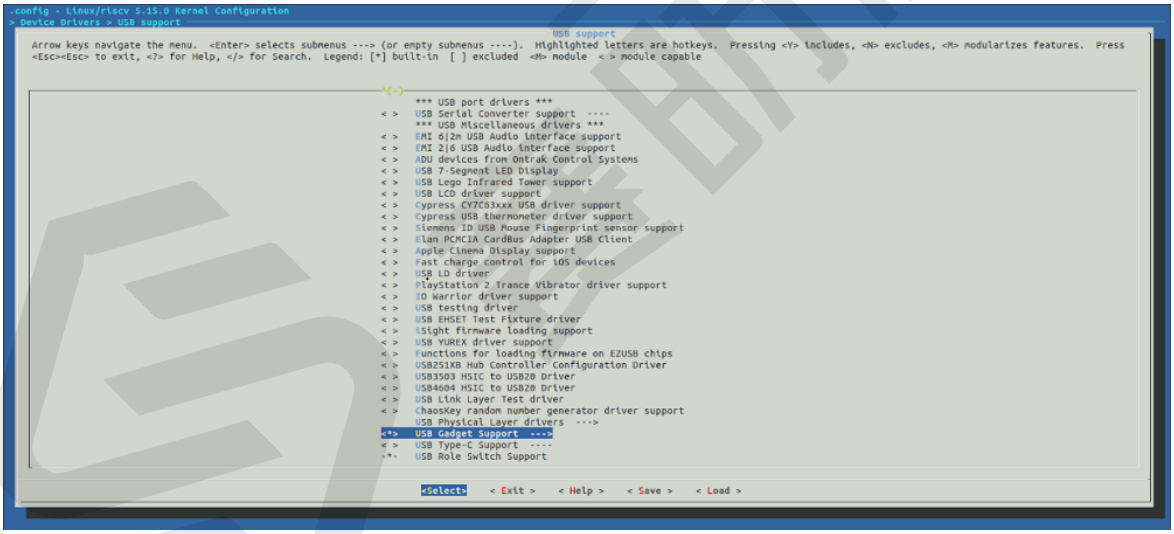
6. To enable the function of USB attached disks, in the **USB Support** menu, select the **USB Mass Storage support** and the **USB Attached SCSI** options.

Figure 2-6 USB Mass Storage Support and USB Attached SCSI



7. To enable the function of USB device, select the **USB Gadget Support** option and enter the menu.

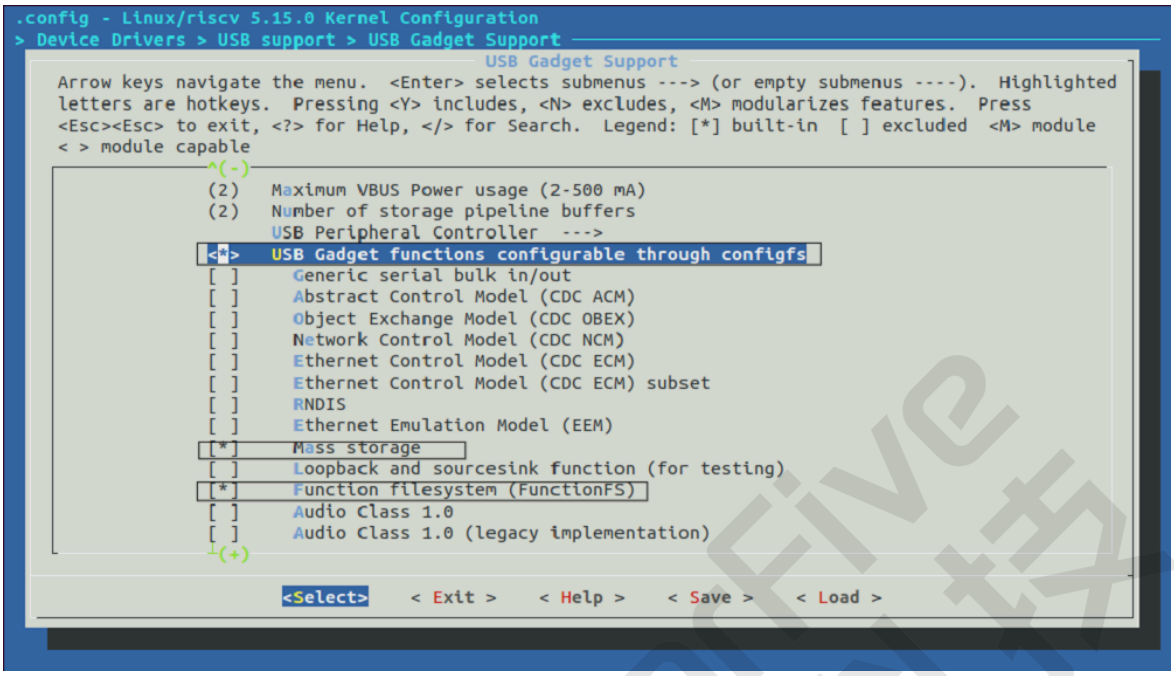
Figure 2-7 USB Gadget Support



8. In the following screen, open your target function.

   For example, to enable the functions of configfs, functionfs and mass storage, select the **USB gadget functions configurable through configfs**, **Mass storage** and **Function filesystem (FunctionFS)**.

Figure 2-8 USB Gadget Functions



9. Save your change before you exit the kernel configuration dialog.

2.4. Source Code Structure

The source code of USB driver is in the `drivers/usb` directory.

The following code blocks show the source code of the USB driver.

USB Host

The following list shows the source code of the USB driver in host mode.

- For USB on the core level:

```
drivers/usb/core
├── Makefile
├── usb.c
├── hub.c
├── hcd.c
├── urb.c
├── message.c
├── driver.c
├── config.c
├── file.c
├── buffer.c
├── sysfs.c
├── endpoint.c
├── devio.c
├── notify.c
```

```
├── generic.c
├── quirks.c
├── devices.c
├── phy.c
├── port.c
├── of.c
├── hcd-pci.c
├── usb-acpi.c
├── Kconfig
```

- For Xhci-host:

```
drivers/usb/host/
├── Makefile
├── Kconfig
├── xhci.c
├── xhci-plat.c
├── xhci-mem.c
├── xhci-ext-caps.c
├── xhci-ring.c
├── xhci-hub.c
├── xhci-dbg.c
├── xhci-trace.c
├── xhci-pci.c
├── xhci-debugfs.c
```

- For CDNS3 on the core and role level:

```
drivers/usb/cdns3/
├── Makefile
├── core.c
├── drd.c
├── Kconfig
```

- For CDNS3 on the platform level:

```
drivers/usb/cdns3/
├── cdns3-plat.c
├── cdns3-starfive.c
```

- For CDN3 host:

```
drivers/usb/cdn3/
├── host.c
```

USB Device

The following list shows the source code of the USB driver in device mode.

- For USB gadget driver:

```
drivers/usb/gadget/
├── usbstring.c
├── config.c
├── epautoconf.c
├── composite.c
├── functions.c
├── configfs.c
├── u_f.c
├── Makefile
├── Kconfig
```

- For USB device function (mass storage and functionfs):

```
drivers/usb/gadget/function/
├── f_mass_storage.c
├── storage_common.c
├── f_fs.c
├── Makefile
├── Kconfig
```

- For USB UDC driver:

```
drivers/usb/gadget/udc/
├── core.c
├── trace.c
├── Makefile
├── Kconfig
```

- For CDN3 USB device driver:
  - For Linux 5.15:

```
drivers/usb/cdn3/
├── cdns3-gadget.c
├── cdns3-ep0.c
```

  - For Linux 5.10:

```
drivers/usb/cdn3/
├── gadget.c
├── ep0.c
```

2.5. Gadget Configuration

Gadgets refer to USB devices with a USB device controller embedded. Based on their configuration, gadgets can be connected to PC, mass storage devices, and UAC devices.

Linux has an original gadget framework, and it can be generally configured in the following procedure:

1. Enable **USB Gadget Support** following the procedure in Kernel Menu Configuration *(on page 13)*.
2. Use the `configfs` framework to build the gadget function.
   a. Use the following commands to build gadgets and write gadget PID, VID and sequence number.

```
mkdir /sys/kernel/config/usb_gadget/g1
echo "VID" > /sys/kernel/config/usb_gadget/g1/idVendor
echo "PID" > /sys/kernel/config/usb_gadget/g1/idProduct
mkdir /sys/kernel/config/usb_gadget/g1/strings/0x409
echo "manufacturer"
 > /sys/kernel/config/usb_gadget/g1/strings/0x409/manufacturer
echo "product"
 > /sys/kernel/config/usb_gadget/g1/strings/0x409/product
```

   b. Use the following commands to build gadget relevant configuration.

```
mkdir /sys/kernel/config/usb_gadget/g1/configs/c.1
echo 0xc0
 > /sys/kernel/config/usb_gadget/g1/configs/c.1/bmAttributes
echo 500 > /sys/kernel/config/usb_gadget/g1/configs/c.1/MaxPower
mkdir /sys/kernel/config/usb_gadget/g1/configs/c.1/strings/0x409
```

   c. Use the following commands to build gadget functions.

```
mkdir /sys/kernel/config/usb_gadget/g1/functions/<name>.<instance
 name>
```

   The above code block contains the following parameters.

   - **name**: Name of the function.
   - **instance name**: Input any string as the instance name.

   d. Use the following commands to build connections between a function and its configuration.

```
ln -s /sys/kernel/config/usb_gadget/g1/functions/<name>.<instance
 name> /sys/kernel/config/
usb_gadget/g1/configs/c.1
```

   e. Use the following command to enable a gadget.

```
echo <udc name> > UDC
```

3. Debug

3.1. USB Debug Nodes

Use the following command to check the current role of USB.

```
cat /sys/devices/platform/soc/10210000.usbdrd/of_node/dr_mode
```

3.2. USB Debugfs

To Get Information of Hubs and Devices

You can use USB debugfs to get the information of the USB hubs and devices.

The following code block provides an example.

```
mount -t debugfs none /sys/kernel/debug
cd /sys/kernel/debug/usb
cat devices
```

To Get Information of Root Hub

You can use USB debugfs to get information of the USB root hub.

The following code block provides an example for USB2.0.

```
T:  Bus=01 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=480  MxCh= 1
B:  Alloc=  0/800 us ( 0%), #Int=  0, #Iso=  0
D:  Ver= 2.00 Cls=09(hub  ) Sub=00 Prot=01 MxPS=64 #Cfgs=  1
P:  Vendor=1d6b ProdID=0002 Rev= 5.15
S:  Manufacturer=Linux 5.15.79-dirty xhci-hcd
S:  Product=xHCI Host Controller
S:  SerialNumber=xhci-hcd.1.auto
C:* #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=  0mA
I:* If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   4 Ivl=256ms
```

The following code block provides an example for USB3.0.

```
T:  Bus=02 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=5000 MxCh= 1
B:  Alloc=  0/800 us ( 0%), #Int=  0, #Iso=  0
D:  Ver= 3.00 Cls=09(hub  ) Sub=00 Prot=03 MxPS= 9 #Cfgs=  1
P:  Vendor=1d6b ProdID=0003 Rev= 5.15
S:  Manufacturer=Linux 5.15.79-dirty xhci-hcd
```

```
S:  Product=xHCI Host Controller
S:  SerialNumber=xhci-hcd.1.auto
C:* #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=  0mA
I:* If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   4 Ivl=256ms
```

To Get Information of Attached Devices

You can use USB debugfs to get information of the USB attached devices.

The following code block provides an example of getting information of an attached Kingston SSD mass storage device.

```
T:  Bus=02 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  5 Spd=5000 MxCh= 0
D:  Ver= 3.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 9 #Cfgs=  1
P:  Vendor=174c ProdID=55aa Rev= 1.00
S:  Manufacturer=asmedia
S:  Product=ASM1153E
S:  SerialNumber=000000000000000000
C:* #Ifs= 1 Cfg#= 1 Atr=c0 MxPwr=  0mA
I:  If#= 0 Alt= 0 #EPs= 2 Cls=08(stor.) Sub=06 Prot=50 Driver=uas
E:  Ad=81(I) Atr=02(Bulk) MxPS=1024 Ivl=0ms
E:  Ad=02(O) Atr=02(Bulk) MxPS=1024 Ivl=0ms
I:* If#= 0 Alt= 1 #EPs= 4 Cls=08(stor.) Sub=06 Prot=62 Driver=uas
E:  Ad=81(I) Atr=02(Bulk) MxPS=1024 Ivl=0ms
E:  Ad=02(O) Atr=02(Bulk) MxPS=1024 Ivl=0ms
E:  Ad=83(I) Atr=02(Bulk) MxPS=1024 Ivl=0ms
E:  Ad=04(O) Atr=02(Bulk) MxPS=1024 Ivl=0ms
```

3.3. USB Overcurrent Debug

USB overcurrent indicates that the current limit of the USB port has been exceeded.

Many electronic devices have a rated current, once the current has been exceeded the device may be burned out. So these devices are equipped with current protection modules. When it exceeds the set current, the device powers off automatically to protect the device, which is called overcurrent protection.

For example, if the USB port of the development board is used to power the device and there is a high current surge, the development board will protect the USB port itself, causing the USB port to be unavailable.

On JH7110, the USB host has an OC signal to detect the USB signal. If the signal is not connected, JH7110 will judge that the USB port is in an overcurrent state, causing it to be unusable.

The following is the workaround to avoid this problem:

1. Enable USB host mode by configuring the node of the USB controller in the DTS file of the kernel, and remove overcurrent pinctrl. The following is an example:

   The original node:

```
usb_pins: usb-pins {
 drive-vbus-pin {
   starfive,pins = <PAD_GPIO33>;
   starfive,pinmux = <PAD_GPIO33_FUNC_SEL 0>;
   starfive,pin-ioconfig = <IO(GPIO_IE(1))>;
   starfive,pin-gpio-dout = <GPO_HIGH>;
   starfive,pin-gpio-doen = <OEN_LOW>;
 };

 overcurrent-pin {
   starfive,pins = <PAD_GPIO34>;
   starfive,pinmux = <PAD_GPIO34_FUNC_SEL 0>;
   starfive,pin-ioconfig = <IO(GPIO_IE(1))>;
   starfive,pin-gpio-din = <GPI_USB0_OVERCURRENT_N_IO>;
   starfive,pin-gpio-doen = <OEN_HIGH>;
 };
};

&usbdrd30 {
 pinctrl-names = "default";
 dr_mode = "host"; /*host or peripheral*/
 status = "disabled";
};
```

   Modified node:

```
usb_pins: usb-pins {
 drive-vbus-pin {
   starfive,pins = <PAD_GPIO33>;
   starfive,pinmux = <PAD_GPIO33_FUNC_SEL 0>;
   starfive,pin-ioconfig = <IO(GPIO_IE(1))>;
   starfive,pin-gpio-dout = <GPO_HIGH>;
   starfive,pin-gpio-doen = <OEN_LOW>;
 };
};

&usbdrd30 {
 pinctrl-names = "default";
 dr_mode = "host"; /*host or peripheral*/
 status = "okay";
};
```

2. Set up USB overcurrent signal by configuring `sys iomux` register.

```
value = readl(priv->base + JH7110_SYS_GPI); // priv->base:0x13040000;
 JH7110_SYS_GPI: 0x80
value &= ~(0x7f << 16);
value |= BIT(16);
writel(value, priv->base + JH7110_SYS_GPI);
```

Note:

To avoid timing issues, please modify the above settings during the U-Boot phase.

4. FAQ

4.1. USB Host Debug

Follow the procedure below to debug USB host mode.

1. Replace the current USB device with other USB devices, to ensure the USB device itself is not a faulty device.
2. Replace the current USB cable with other USB cables, to ensure the USB cable itself is not faulty.
3. Try the same connection on other PC hosts.
4. If your device has multiple USB ports, try out on other ports.
5. Connect the USB-Hub device with an independent power supply, and then connect a USB device to the USB-Hub to confirm if the host function works properly.
6. Confirm if the host driver is loaded successfully.

   Use the following command to verify.

   ```
   cat /sys/devices/platform/soc/10210000.usbdrd/of_node/dr_mode
   host
   ```

   Note:
   If the folder `10210000.usbdrd` does not exist, it means that the host driver is not installed.

7. Make sure you already have the latest code or patch.
8. Under the same condition, compare the registers of the failed board against the ones which are working properly.

4.2. USB Device Debug

Follow the procedure below to debug USB device mode.

1. Replace the current PC USB port with other USB port, to ensure the USB port itself is not a faulty device.
2. Replace the current USB cable with other USB cables, to ensure the USB cable itself is not faulty.
3. Try the same connection on other PC.

4. If your device has multiple USB devices ports, try out on other ports.
5. Confirm if the host driver is loaded successfully.

   Use the following command to verify.

   ```
   cat /sys/devices/platform/soc/10210000.usbdrd/of_node/dr_mode
   peripheral
   ```

   Note:
   If the folder `10210000.usbdrd` does not exist, it means that the host driver is not installed.

6. Make sure you already have the latest code or patch.
7. Under the same condition, compare the registers of the failed board against the ones which are working properly.

5. Gadget Configuration Examples

5.1. Gadget as Mass Storage

Use the commands below if you want to configure the gadget as a mass storage device.

```
dd if=/dev/zero of=/dev/a.bin bs=1M count=100
mount -t configfs none /sys/kernel/config
mkdir /sys/kernel/config/usb_gadget/g1
echo "0x1d6b " > /sys/kernel/config/usb_gadget/g1/idVendor
echo "0x0001" > /sys/kernel/config/usb_gadget/g1/idProduct
mkdir /sys/kernel/config/usb_gadget/g1/strings/0x409
mkdir /sys/kernel/config/usb_gadget/g1/functions/mass_storage.usb0
echo /dev/a.bin
 > /sys/kernel/config/usb_gadget/g1/functions/mass_storage.usb0/lun.0/file
mkdir /sys/kernel/config/usb_gadget/g1/configs/c.1
echo 0xc0 > /sys/kernel/config/usb_gadget/g1/configs/c.1/bmAttributes
echo 500 > /sys/kernel/config/usb_gadget/g1/configs/c.1/MaxPower
mkdir /sys/kernel/config/usb_gadget/g1/configs/c.1/strings/0x409
ln
 -s /sys/kernel/config/usb_gadget/g1/functions/mass_storage.usb0/ /
sys/kernel/config/
usb_gadget/g1/configs/c.1/
ls /sys/class/udc/ | xargs echo > /sys/kernel/config/usb_gadget/g1/UDC
```

5.2. Gadget as ADB

Use the commands below if you want to configure the gadget as an ADB.

```
mount -t configfs none /sys/kernel/config
mkdir /sys/kernel/config/usb_gadget/g1
echo "0x1d6b" > /sys/kernel/config/usb_gadget/g1/idVendor
echo "0x0002" > /sys/kernel/config/usb_gadget/g1/idProduct
mkdir /sys/kernel/config/usb_gadget/g1/strings/0x409
echo "20181111"
 > /sys/kernel/config/usb_gadget/g1/strings/0x409/serialnumber
echo "starfive"
 > /sys/kernel/config/usb_gadget/g1/strings/0x409/manufacturer
mkdir /sys/kernel/config/usb_gadget/g1/functions/ffs.adb
mkdir /sys/kernel/config/usb_gadget/g1/configs/c.1
echo 0xc0 > /sys/kernel/config/usb_gadget/g1/configs/c.1/bmAttributes
echo 500 > /sys/kernel/config/usb_gadget/g1/configs/c.1/MaxPower
mkdir /sys/kernel/config/usb_gadget/g1/configs/c.1/strings/0x409
```

```
ln
 -s /sys/kernel/config/usb_gadget/g1/functions/ffs.adb/ /
sys/kernel/config/usb_gadget/g1/
configs/c.1/ffs.adb
mkdir /dev/usb-ffs
mkdir /dev/usb-ffs/adb
mount -o uid=2000,gid=2000 -t functionfs adb /dev/usb-ffs/adb/
ls /sys/class/udc/ | xargs echo > /sys/kernel/config/usb_gadget/g1/UDC
```

5.3. Gadget as Mass Storage and ADB

Use the commands below if you want to configure the gadget as both a Mass Storage and an ADB.

```
mount -t configfs none /sys/kernel/config
mkdir /sys/kernel/config/usb_gadget/g1
echo "0x1d6b" > /sys/kernel/config/usb_gadget/g1/idVendor
echo "0x0003" > /sys/kernel/config/usb_gadget/g1/idProduct
mkdir /sys/kernel/config/usb_gadget/g1/strings/0x409
echo "20181111"
 > /sys/kernel/config/usb_gadget/g1/strings/0x409/serialnumber
echo "stafive" > /sys/kernel/config/usb_gadget/g1/strings/0x409/manufacturer
mkdir /sys/kernel/config/usb_gadget/g1/functions/ffs.adb
mkdir /sys/kernel/config/usb_gadget/g1/functions/mass_storage.usb0
echo ${BLOCK_PATH}
 > /sys/kernel/config/usb_gadget/g1/functions/mass_storage.usb0/lun.0/
file
mkdir /sys/kernel/config/usb_gadget/g1/configs/c.1
echo 0xc0 > /sys/kernel/config/usb_gadget/g1/configs/c.1/bmAttributes
echo 500 > /sys/kernel/config/usb_gadget/g1/configs/c.1/MaxPower
mkdir /sys/kernel/config/usb_gadget/g1/configs/c.1/strings/0x409
ln
 -s /sys/kernel/config/usb_gadget/g1/functions/ffs.adb/ /
sys/kernel/config/usb_gadget/g1/
configs/c.1/ffs.adb
ln
 -s /sys/kernel/config/usb_gadget/g1/functions/mass_storage.usb0/ /
sys/kernel/config/
usb_gadget/g1/configs/c.1/mass_storage.usb0
mkdir /dev/usb-ffs
mkdir /dev/usb-ffs/adb
mount -o uid=2000,gid=2000 -t functionfs adb /dev/usb-ffs/adb/
ls /sys/class/udc/ | xargs echo > /sys/kernel/config/usb_gadget/g1/UDC
```