Rockchip-USB-FFS-Test-Demo

发布版本：1.2

作者邮箱：wulf@rock-chips.com

日期：2019-11-11

文档密级：内部资料

概述

本文档提供 Rockchip 平台 USB FFS Test Demo 的使用方法。

产品版本

芯片名称	内核版本
RK3399、RK3368、RK3366、RK3328、RK3288、RK312X、RK3188、RK30XX、RK3308、RK3326、PX30	Linux-4.4、Linux-4.19

读者对象

本文档（本指南）主要适用于以下工程师：

软件工程师

技术支持工程师

修订记录

日期	版本	作者	修改说明
2018-07-02	V1.0	吴良峰	初始版本
2019-01-09	V1.1	吴良峰	使用 markdownlint 修订格式
2019-11-11	V1.2	吴良峰	修改文档名称，支持Linux-4.19

测试 Demo 源码

1. Simple-Demo: kernel/tools/usb/ffs-aio-example/simple
2. Multibuf-Demo: kernel/tools/usb/ffs-aio-example/multibuff

Note:

- The two test demo showing usage of Asynchronous I/O API of FunctionFS.
- "Simple-Demo" is a simple example of bidirectional data; "Multibuf-Demo" shows multi-buffer data transfer, which may to be used in high performance applications.
- Both examples contains userspace applications for device and for host.
- It needs libaio library on the device, and libusb library on host.
- Only support USB2.0

Toolchain 下载地址（ARCH=arm64）

`ssh://wulf@10.10.10.29:29418/rk/prebuilts/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu`

Note: "wulf"请修改为自己的 Gerrit 用户名

Libaio 下载地址

https://pagure.io/libaio.git

Libaio 库的编译

进入 libaio/src 目录下，修改 Makefile 的 “CC”和“AR”

```
diff --git a/src/Makefile b/src/Makefile
index eadb336..9d3f19b 100644
--- a/src/Makefile
+++ b/src/Makefile
@@ -1,3 +1,5 @@
+CC = $(CROSS_COMPILE)gcc
+AR = $(CROSS_COMPILE)ar
 prefix=/usr
 includedir=$(prefix)/include
 libdir=$(prefix)/lib
```

然后，执行 make 命令

```
make ARCH=arm64  CROSS_COMPILE=../..toolchain/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-
```

生成静态库：libaio.a

生成动态库：libaio.so.1.0.1

建议使用静态库 libaio.a 来编译 FFS 测试 Demo

测试 Demo 的编译

Device_app 的编译

1. 将 libaio/src/libaio.h 拷贝到 kernel/tools/include/tools/.

2. 将静态库 libaio.a 分别拷贝到 kernel/tools/usb/ffs-aio-example/multibuff/device_app/. 和 kernel/tools/usb/ffs-aio-example/simple/device_app/.

3. 修改 aio_multibuff.c 和 aio_simple.c 的头文件

```
diff --git a/tools/usb/ffs-aio-example/multibuff/device_app/aio_multibuff.c
b/tools/usb/ffs-aio-example/multibuff/device_app/aio_multibuff.c
index aaca1f4..e0bf98c 100644
--- a/tools/usb/ffs-aio-example/multibuff/device_app/aio_multibuff.c
+++ b/tools/usb/ffs-aio-example/multibuff/device_app/aio_multibuff.c
@@ -42,7 +42,7 @@
 #include <stdbool.h>
 #include <sys/eventfd.h>

-#include "libaio.h"
+#include <tools/libaio.h>
 #define IOCB_FLAG_RESFD         (1 << 0)

 #include <linux/usb/functionfs.h>
diff --git a/tools/usb/ffs-aio-example/simple/device_app/aio_simple.c
b/tools/usb/ffs-aio-example/simple/device_app/aio_simple.c
index 1f44a29..3dab7f1 100644
--- a/tools/usb/ffs-aio-example/simple/device_app/aio_simple.c
+++ b/tools/usb/ffs-aio-example/simple/device_app/aio_simple.c
@@ -42,7 +42,7 @@
 #include <stdbool.h>
 #include <sys/eventfd.h>

-#include "libaio.h"
+#include <tools/libaio.h>
 #define IOCB_FLAG_RESFD         (1 << 0)

 #include <linux/usb/functionfs.h>
```

4. 增加 Makefile 文件（指定在当前目录下，查找静态库 libaio.a 文件）

```
kernel/tools/usb/ffs-aio-example/simple/device_app/Makefile

# Makefile for USB tools
CC = $(CROSS_COMPILE)gcc
AIO_LIBS = -L. -laio
WARNINGS = -Wall -Wextra
CFLAGS = $(WARNINGS) -static -I../../../../include
LDFLAGS = $(AIO_LIBS)

all: aio_simple
%: %.c
        $(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)

clean:
        $(RM) aio_simple
```

```
kernel/tools/usb/ffs-aio-example/multibuff/device_app/Makefile

# Makefile for USB tools
CC = $(CROSS_COMPILE)gcc
```

```
AIO_LIBS = -L. -laio
WARNINGS = -Wall -Wextra
CFLAGS = $(WARNINGS) -static -I../../../../include
LDFLAGS = $(AIO_LIBS)

all: aio_multibuff
%: %.c
	$(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)

clean:
	$(RM) aio_multibuff
```

5. 执行 make 命令

```
make ARCH=arm64 CROSS_COMPILE=../../../../../../toolchain/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-
```

在 ffs-aio-example/simple/device_app 和 ffs-aio-example/multibuff/device_app 目录下，分别执行上述的 make 命令，编译成功后，得到 ARM64 平台的可执行文件 “aio_simple” 和 “aio_multibuff”。

Host_app 的编译

Host_app 可以运行于 PC Ubuntu，编译时不需要对源码做任何改动，只要在 kernel/tools/usb/ffs-aio-example/simple/host_app 和 kernel/tools/usb/ffs-aio-example/multibuff/host_app 目录下执行 make 命令即可，得到可执行文件“test”。

测试方法

1. 将编译 Demo Device-app 得到的可执行文件 “aio_simple” 和 “aio_multibuff” 拷贝到测试平台的 /data/. 路径下，

   并设置可执行的权限。

2. **断开测试平台 USB 与 PC 的连接。**

3. **配置 Configfs 和 Function FS Gadget**

   1.1 通用的配置方法

   如果是使用 RK Android 平台，配置方法请参考“1.2 基于 RK3399 Android 挖掘机平台的配置方法”。

```
#usb init参考android 脚本 init.rk30board.usb.rc和init.usb.configfs.rc

#Manual / Command line instructions :
#Mount ConfigFS and create Gadget
mount -t configfs none /config
mkdir /config/usb_gadget/g1

#Set default Vendor and Product IDs and so on for now
echo 0x1d6b > /config/usb_gadget/g1/idVendor
echo 0x0105 > /config/usb_gadget/g1/idProduct
echo 0x0310 > /config/usb_gadget/g1/bcdDevice
echo 0x0200 > /config/usb_gadget/g1/bcdUSB

#Create English strings and add random deviceID
mkdir /config/usb_gadget/g1/strings/0x409
```

```
echo 0123459876 > /config/usb_gadget/g1/strings/0x409/serialnumber

#Update following if you want to
echo "rockchip" > /config/usb_gadget/g1/strings/0x409/manufacturer
echo "rkusbtest" >  /config/usb_gadget/g1/strings/0x409/product

#Create gadget configuration
mkdir /config/usb_gadget/g1/configs/b.1
mkdir /config/usb_gadget/g1/configs/b.1/strings/0x409
echo "test" > /config/usb_gadget/g1/configs/b.1/strings/0x409/configuration
echo 500 > /config/usb_gadget/g1/configs/b.1/MaxPower

#Set os_desc and link it to the gadget configuration
echo 0x1 > /config/usb_gadget/g1/os_desc/b_vendor_code
echo "MSFT100" > /config/usb_gadget/g1/os_desc/qw_sign
ln -s /config/usb_gadget/g1/configs/b.1 /config/usb_gadget/g1/os_desc/b.1

#Create test FunctionFS function
#And link it to the gadget configuration
mkdir /config/usb_gadget/g1/functions/ffs.test
rm /config/usb_gadget/g1/configs/b.1/f1
ln -s /config/usb_gadget/g1/functions/ffs.test
/config/usb_gadget/g1/configs/b.1/f1

#Create ffs test and mount it, then  /dev/usb-ffs/test/ep0 will be created
mkdir -p /dev/usb-ffs/test
mount -o rmode=0770,fmode=0660,uid=1024,gid=1024 -t functionfs test
/dev/usb-ffs/test
```

1.2 基于 RK3399 Android 挖掘机平台的配置方法

如果是基于 RK3399 Android 挖掘机平台进行测试，由于 Android 的 usb init 文件已经创建的 Configfs，并完成了部分 Configfs 的配置工作，所以只需要再执行如下的配置步骤：

```
#usb init参考android 脚本 init.rk30board.usb.rc和init.usb.configfs.rc

#Manual / Command line instructions :

#Set default Vendor and Product IDs and so on for now
echo 0x1d6b > /config/usb_gadget/g1/idVendor
echo 0x0105 > /config/usb_gadget/g1/idProduct

#Set gadget configuration
echo "test" > /config/usb_gadget/g1/configs/b.1/strings/0x409/configuration

#Create test FunctionFS function
#And link it to the gadget configuration
mkdir /config/usb_gadget/g1/functions/ffs.test
rm /config/usb_gadget/g1/configs/b.1/f1
ln -s /config/usb_gadget/g1/functions/ffs.test
/config/usb_gadget/g1/configs/b.1/f1

#Create ffs test and mount it, then  /dev/usb-ffs/test/ep0 will be created
mkdir -p /dev/usb-ffs/test
mount -o rmode=0770,fmode=0660,uid=1024,gid=1024 -t functionfs test
/dev/usb-ffs/test
```

4. 执行测试平台的可执行文件 “aio_simple” 或 “aio_multibuff”

   `./aio_simple /dev/usb-ffs/test &`

   `./aio_multibuff /dev/usb-ffs/test &`

   如果执行成功，可以在 /dev/usb-ffs/test 目录下，查看到 ep0/ep1/ep2 三个设备端点。

5. 使能 USB 控制器

   `echo  fe800000.dwc3 >/config/usb_gadget/g1/UDC`

6. 连接 USB 到 PC ubuntu 的 USB 接口，然后执行 lsusb，查看是否有 USB 设备 “1d6b:0105 Linux Foundation FunctionFS Gadget”，如果存在，则表明 USB FFS Gadget 枚举成功。

7. 在 PC ubuntu 上，执行 host 端的测试 app“test”，则会通过 libusb 主动搜索 ID 为 “1d6b:0105” 的 USB 设备，并进行 USB 传输测试。

测试 Demo USB 3.0 的支持

Kernel tools 源码提供的 USB FFS 测试 Demo 最高只能支持 USB 2.0，不能支持 USB 3.0，如果要支持 USB 3.0 ，需要更新如下的补丁，测试方法与 USB 2.0 一样。

```
diff --git a/tools/usb/ffs-aio-example/multibuff/device_app/aio_multibuff.c
b/tools/usb/ffs-aio-example/multibuff/device_app/aio_multibuff.c
index aaca1f4..e0bf98c 100644
--- a/tools/usb/ffs-aio-example/multibuff/device_app/aio_multibuff.c
+++ b/tools/usb/ffs-aio-example/multibuff/device_app/aio_multibuff.c
@@ -57,16 +57,30 @@ static const struct {
        struct usb_functionfs_descs_head_v2 header;
        __le32 fs_count;
        __le32 hs_count;
+       __le32 ss_count;
+       __le32 os_count;
        struct {
                struct usb_interface_descriptor intf;
                struct usb_endpoint_descriptor_no_audio bulk_sink;
                struct usb_endpoint_descriptor_no_audio bulk_source;
        } __attribute__ ((__packed__)) fs_descs, hs_descs;
+       struct {
+               struct usb_interface_descriptor intf;
+               struct usb_endpoint_descriptor_no_audio sink;
+               struct usb_ss_ep_comp_descriptor sink_comp;
+               struct usb_endpoint_descriptor_no_audio source;
+               struct usb_ss_ep_comp_descriptor source_comp;
+       } __attribute__ ((__packed__)) ss_descs;
+       struct usb_os_desc_header os_header;
+       struct usb_ext_compat_desc os_desc;
+
 } __attribute__ ((__packed__)) descriptors = {
        .header = {
                .magic = htole32(FUNCTIONFS_DESCRIPTORS_MAGIC_V2),
                .flags = htole32(FUNCTIONFS_HAS_FS_DESC |
-                                FUNCTIONFS_HAS_HS_DESC),
+                                FUNCTIONFS_HAS_HS_DESC |
+                                FUNCTIONFS_HAS_SS_DESC |
+                                FUNCTIONFS_HAS_MS_OS_DESC),
                .length = htole32(sizeof(descriptors)),
        },
        .fs_count = htole32(3),
```

```
@@ -115,6 +129,57 @@ static const struct {
                        .wMaxPacketSize = htole16(512),
                },
        },
+       .ss_count = htole32(5),
+       .ss_descs = {
+               .intf = {
+                       .bLength = sizeof(descriptors.ss_descs.intf),
+                       .bDescriptorType = USB_DT_INTERFACE,
+                       .bInterfaceNumber = 0,
+                       .bNumEndpoints = 2,
+                       .bInterfaceClass = USB_CLASS_VENDOR_SPEC,
+                       .iInterface = 1,
+               },
+               .sink = {
+                       .bLength = sizeof(descriptors.ss_descs.sink),
+                       .bDescriptorType = USB_DT_ENDPOINT,
+                       .bEndpointAddress = 1 | USB_DIR_IN,
+                       .bmAttributes = USB_ENDPOINT_XFER_BULK,
+                       .wMaxPacketSize = htole16(1024),
+               },
+               .sink_comp = {
+                       .bLength = sizeof(descriptors.ss_descs.sink_comp),
+                       .bDescriptorType = USB_DT_SS_ENDPOINT_COMP,
+                       .bMaxBurst = 4,
+               },
+               .source = {
+                       .bLength = sizeof(descriptors.ss_descs.source),
+                       .bDescriptorType = USB_DT_ENDPOINT,
+                       .bEndpointAddress = 2 | USB_DIR_OUT,
+                       .bmAttributes = USB_ENDPOINT_XFER_BULK,
+                       .wMaxPacketSize = htole16(1024),
+               },
+               .source_comp = {
+                       .bLength = sizeof(descriptors.ss_descs.source_comp),
+                       .bDescriptorType = USB_DT_SS_ENDPOINT_COMP,
+                       .bMaxBurst = 4,
+               },
+       },
+       .os_count = htole32(1),
+       .os_header = {
+               .interface = htole32(1),
+               .dwLength = htole32(sizeof(descriptors.os_header) +
sizeof(descriptors.os_desc)),
+               .bcdVersion = htole32(1),
+               .wIndex = htole32(4),
+               .bCount = htole32(1),
+               .Reserved = htole32(0),
+       },
+       .os_desc = {
+               .bFirstInterfaceNumber = 0,
+               .Reserved1 = htole32(1),
+               .CompatibleID = {0},
+               .SubCompatibleID = {0},
+               .Reserved2 = {0},
+       },
 };
```

```
 #define STR_INTERFACE "AIO Test"
diff --git a/tools/usb/ffs-aio-example/simple/device_app/aio_simple.c
b/tools/usb/ffs-aio-example/simple/device_app/aio_simple.c
index 1f44a29..3dab7f1 100644
--- a/tools/usb/ffs-aio-example/simple/device_app/aio_simple.c
+++ b/tools/usb/ffs-aio-example/simple/device_app/aio_simple.c
@@ -55,16 +55,30 @@ static const struct {
        struct usb_functionfs_descs_head_v2 header;
        __le32 fs_count;
        __le32 hs_count;
+       __le32 ss_count;
+       __le32 os_count;
        struct {
                struct usb_interface_descriptor intf;
                struct usb_endpoint_descriptor_no_audio bulk_sink;
                struct usb_endpoint_descriptor_no_audio bulk_source;
        } __attribute__ ((__packed__)) fs_descs, hs_descs;
+       struct {
+               struct usb_interface_descriptor intf;
+               struct usb_endpoint_descriptor_no_audio sink;
+               struct usb_ss_ep_comp_descriptor sink_comp;
+               struct usb_endpoint_descriptor_no_audio source;
+               struct usb_ss_ep_comp_descriptor source_comp;
+       } __attribute__ ((__packed__)) ss_descs;
+       struct usb_os_desc_header os_header;
+       struct usb_ext_compat_desc os_desc;
+
 } __attribute__ ((__packed__)) descriptors = {
        .header = {
                .magic = htole32(FUNCTIONFS_DESCRIPTORS_MAGIC_V2),
                .flags = htole32(FUNCTIONFS_HAS_FS_DESC |
-                                FUNCTIONFS_HAS_HS_DESC),
+                                FUNCTIONFS_HAS_HS_DESC |
+                                FUNCTIONFS_HAS_SS_DESC |
+                                FUNCTIONFS_HAS_MS_OS_DESC),
                .length = htole32(sizeof(descriptors)),
        },
        .fs_count = htole32(3),
@@ -113,6 +127,57 @@ static const struct {
                        .wMaxPacketSize = htole16(512),
                },
        },
+       .ss_count = htole32(5),
+       .ss_descs = {
+               .intf = {
+                       .bLength = sizeof(descriptors.ss_descs.intf),
+                       .bDescriptorType = USB_DT_INTERFACE,
+                       .bInterfaceNumber = 0,
+                       .bNumEndpoints = 2,
+                       .bInterfaceClass = USB_CLASS_VENDOR_SPEC,
+                       .iInterface = 1,
+               },
+               .sink = {
+                       .bLength = sizeof(descriptors.ss_descs.sink),
+                       .bDescriptorType = USB_DT_ENDPOINT,
+                       .bEndpointAddress = 1 | USB_DIR_IN,
+                       .bmAttributes = USB_ENDPOINT_XFER_BULK,
+                       .wMaxPacketSize = htole16(1024),
```

```
+               },
+               .sink_comp = {
+                       .bLength = sizeof(descriptors.ss_descs.sink_comp),
+                       .bDescriptorType = USB_DT_SS_ENDPOINT_COMP,
+                       .bMaxBurst = 4,
+               },
+               .source = {
+                       .bLength = sizeof(descriptors.ss_descs.source),
+                       .bDescriptorType = USB_DT_ENDPOINT,
+                       .bEndpointAddress = 2 | USB_DIR_OUT,
+                       .bmAttributes = USB_ENDPOINT_XFER_BULK,
+                       .wMaxPacketSize = htole16(1024),
+               },
+               .source_comp = {
+                       .bLength = sizeof(descriptors.ss_descs.source_comp),
+                       .bDescriptorType = USB_DT_SS_ENDPOINT_COMP,
+                       .bMaxBurst = 4,
+               },
+       },
+       .os_count = htole32(1),
+       .os_header = {
+               .interface = htole32(1),
+               .dwLength = htole32(sizeof(descriptors.os_header) +
sizeof(descriptors.os_desc)),
+               .bcdVersion = htole32(1),
+               .wIndex = htole32(4),
+               .bCount = htole32(1),
+               .Reserved = htole32(0),
+       },
+       .os_desc = {
+               .bFirstInterfaceNumber = 0,
+               .Reserved1 = htole32(1),
+               .CompatibleID = {0},
+               .SubCompatibleID = {0},
+               .Reserved2 = {0},
+       },
 };

 #define STR_INTERFACE "AIO Test"
```