

# Rockchip-USB-FFS-Test-Demo

发布版本：1.1

作者邮箱：[wulf@rock-chips.com](mailto:wulf@rock-chips.com)

日期：2019-01-09

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	Linux4.4

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

## Rockchip-USB-FFS-Test-Demo

- 测试Demo源码
- Toolchain下载地址（ARCH=arm64）
- Libaio下载地址
- Libaio库的编译
- 测试Demo的编译
  - Device\_app的编译
  - Host\_app的编译
- 测试方法
- 测试Demo USB 3.0 的支持

## 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”

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

然后，执行 make 命令

```
1 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

```
1 diff --git a/tools/usb/ffs-aio-example/multibuff/device_app/aio_multibuff.c
2 b/tools/usb/ffs-aio-example/multibuff/device_app/aio_multibuff.c
3 index aaca1f4..e0bf98c 100644
4 --- a/tools/usb/ffs-aio-example/multibuff/device_app/aio_multibuff.c
5 +++ b/tools/usb/ffs-aio-example/multibuff/device_app/aio_multibuff.c
6 @@ -42,7 +42,7 @@
7  #include <stdbool.h>
8  #include <sys/eventfd.h>
9
10 -#include "libaio.h"
11 +#include <tools/libaio.h>
12  #define IOCB_FLAG_RESFD          (1 << 0)
13
14  #include <linux/usb/functionfs.h>
15 diff --git a/tools/usb/ffs-aio-example/simple/device_app/aio_simple.c
16 b/tools/usb/ffs-aio-example/simple/device_app/aio_simple.c
17 index 1f44a29..3dab7f1 100644
18 --- a/tools/usb/ffs-aio-example/simple/device_app/aio_simple.c
19 +++ b/tools/usb/ffs-aio-example/simple/device_app/aio_simple.c
20 @@ -42,7 +42,7 @@
21  #include <stdbool.h>
22  #include <sys/eventfd.h>
23
24 -#include "libaio.h"
25 +#include <tools/libaio.h>
26  #define IOCB_FLAG_RESFD          (1 << 0)
27
28  #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）

```
1 kernel/tools/usb/ffs-aio-example/simple/device_app/Makefile
2
3 # Makefile for USB tools
4 CC = $(CROSS_COMPILE)gcc
5 AIO_LIBS = -L. -laio
6 WARNINGS = -Wall -Wextra
7 CFLAGS = $(WARNINGS) -static -I../../../include
8 LDFLAGS = $(AIO_LIBS)
9
10 all: aio_simple
```

```

11  %: %.c
12      $(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)
13
14  clean:
15      $(RM) aio_simple

```

```

1  kernel/tools/usb/ffs-aio-example/multibuff/device_app/Makefile
2
3  # Makefile for USB tools
4  CC = $(CROSS_COMPILE)gcc
5  AIO_LIBS = -L. -laio
6  WARNINGS = -Wall -Wextra
7  CFLAGS = $(WARNINGS) -static -I../..../include
8  LDFLAGS = $(AIO_LIBS)
9
10 all: aio_multibuff
11 %: %.c
12     $(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)
13
14 clean:
15     $(RM) aio_multibuff

```

## 5. 执行 make 命令

```

1  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”。

```

1  #usb init参考android 脚本 init.rk30board.usb.rc和init.usb.configfs.rc
2

```

```

3 #Manual / Command line instructions :
4 #Mount ConfigFS and create Gadget
5 mount -t configfs none /config
6 mkdir /config/usb_gadget/g1
7
8 #Set default Vendor and Product IDs and so on for now
9 echo 0x1d6b > /config/usb_gadget/g1/idVendor
10 echo 0x0105 > /config/usb_gadget/g1/idProduct
11 echo 0x0310 > /config/usb_gadget/g1/bcdDevice
12 echo 0x0200 > /config/usb_gadget/g1/bcdUSB
13
14 #Create English strings and add random deviceID
15 mkdir /config/usb_gadget/g1/strings/0x409
16 echo 0123459876 > /config/usb_gadget/g1/strings/0x409/serialnumber
17
18 #Update following if you want to
19 echo "rockchip" > /config/usb_gadget/g1/strings/0x409/manufacturer
20 echo "rkusbtest" > /config/usb_gadget/g1/strings/0x409/product
21
22 #Create gadget configuration
23 mkdir /config/usb_gadget/g1/configs/b.1
24 mkdir /config/usb_gadget/g1/configs/b.1/strings/0x409
25 echo "test" > /config/usb_gadget/g1/configs/b.1/strings/0x409/configuration
26 echo 500 > /config/usb_gadget/g1/configs/b.1/MaxPower
27
28 #Set os_desc and link it to the gadget configuration
29 echo 0x1 > /config/usb_gadget/g1/os_desc/b_vendor_code
30 echo "MSFT100" > /config/usb_gadget/g1/os_desc/qw_sign
31 ln -s /config/usb_gadget/g1/configs/b.1 /config/usb_gadget/g1/os_desc/b.1
32
33 #Create test FunctionFS function
34 #And link it to the gadget configuration
35 mkdir /config/usb_gadget/g1/functions/ffs.test
36 rm /config/usb_gadget/g1/configs/b.1/f1
37 ln -s /config/usb_gadget/g1/functions/ffs.test
   /config/usb_gadget/g1/configs/b.1/f1
38
39 #Create ffs test and mount it, then /dev/usb-ffs/test/ep0 will be created
40 mkdir -p /dev/usb-ffs/test
41 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：

```

1 #usb init参考android 脚本 init.rk30board.usb.rc和init.usb.configfs.rc
2
3 #Manual / Command line instructions :
4
5 #Set default Vendor and Product IDs and so on for now
6 echo 0x1d6b > /config/usb_gadget/g1/idVendor

```

```

7 echo 0x0105 > /config/usb_gadget/g1/idProduct
8
9 #Set gadget configuration
10 echo "test" > /config/usb_gadget/g1/configs/b.1/strings/0x409/configuration
11
12 #Create test FunctionFS function
13 #And link it to the gadget configuration
14 mkdir /config/usb_gadget/g1/functions/ffs.test
15 rm /config/usb_gadget/g1/configs/b.1/f1
16 ln -s /config/usb_gadget/g1/functions/ffs.test
    /config/usb_gadget/g1/configs/b.1/f1
17
18 #Create ffs test and mount it, then /dev/usb-ffs/test/ep0 will be created
19 mkdir -p /dev/usb-ffs/test
20 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。

```

1 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
2 index aaca1f4..e0bf98c 100644
3 --- a/tools/usb/ffs-aio-example/multibuff/device_app/aio_multibuff.c
4 +++ b/tools/usb/ffs-aio-example/multibuff/device_app/aio_multibuff.c
5 @@ -57,16 +57,30 @@ static const struct {
6         struct usb_functionfs_descs_head_v2 header;
7         __le32 fs_count;
8         __le32 hs_count;
9 +     __le32 ss_count;
10 +     __le32 os_count;
11     struct {
12         struct usb_interface_descriptor intf;
13         struct usb_endpoint_descriptor_no_audio bulk_sink;
14         struct usb_endpoint_descriptor_no_audio bulk_source;

```

```

15     } __attribute__((packed)) fs_descs, hs_descs;
16 +     struct {
17 +         struct usb_interface_descriptor intf;
18 +         struct usb_endpoint_descriptor_no_audio sink;
19 +         struct usb_ss_ep_comp_descriptor sink_comp;
20 +         struct usb_endpoint_descriptor_no_audio source;
21 +         struct usb_ss_ep_comp_descriptor source_comp;
22 +     } __attribute__((packed)) ss_descs;
23 +     struct usb_os_desc_header os_header;
24 +     struct usb_ext_compat_desc os_desc;
25 +
26 } __attribute__((packed)) descriptors = {
27     .header = {
28         .magic = htole32(FUNCTIONFS_DESCRIPTOR_MAGIC_V2),
29         .flags = htole32(FUNCTIONFS_HAS_FS_DESC |
30 -             FUNCTIONFS_HAS_HS_DESC),
31 +             FUNCTIONFS_HAS_HS_DESC |
32 +             FUNCTIONFS_HAS_SS_DESC |
33 +             FUNCTIONFS_HAS_MS_OS_DESC),
34         .length = htole32(sizeof(descriptors)),
35     },
36     .fs_count = htole32(3),
37 @@ -115,6 +129,57 @@ static const struct {
38         .wMaxPacketSize = htole16(512),
39     },
40 },
41 + .ss_count = htole32(5),
42 + .ss_descs = {
43 +     .intf = {
44 +         .bLength = sizeof(descriptors.ss_descs.intf),
45 +         .bDescriptorType = USB_DT_INTERFACE,
46 +         .bInterfaceNumber = 0,
47 +         .bNumEndpoints = 2,
48 +         .bInterfaceClass = USB_CLASS_VENDOR_SPEC,
49 +         .iInterface = 1,
50 +     },
51 +     .sink = {
52 +         .bLength = sizeof(descriptors.ss_descs.sink),
53 +         .bDescriptorType = USB_DT_ENDPOINT,
54 +         .bEndpointAddress = 1 | USB_DIR_IN,
55 +         .bmAttributes = USB_ENDPOINT_XFER_BULK,
56 +         .wMaxPacketSize = htole16(1024),
57 +     },
58 +     .sink_comp = {
59 +         .bLength = sizeof(descriptors.ss_descs.sink_comp),
60 +         .bDescriptorType = USB_DT_SS_ENDPOINT_COMP,
61 +         .bMaxBurst = 4,
62 +     },
63 +     .source = {
64 +         .bLength = sizeof(descriptors.ss_descs.source),
65 +         .bDescriptorType = USB_DT_ENDPOINT,
66 +         .bEndpointAddress = 2 | USB_DIR_OUT,
67 +         .bmAttributes = USB_ENDPOINT_XFER_BULK,

```

```

68 +             .wMaxPacketSize = htole16(1024),
69 +         },
70 +         .source_comp = {
71 +             .bLength = sizeof(descriptors.ss_descs.source_comp),
72 +             .bDescriptorType = USB_DT_SS_ENDPOINT_COMP,
73 +             .bMaxBurst = 4,
74 +         },
75 +     },
76 +     .os_count = htole32(1),
77 +     .os_header = {
78 +         .interface = htole32(1),
79 +         .dwLength = htole32(sizeof(descriptors.os_header) +
sizeof(descriptors.os_desc)),
80 +         .bcdVersion = htole32(1),
81 +         .wIndex = htole32(4),
82 +         .bCount = htole32(1),
83 +         .Reserved = htole32(0),
84 +     },
85 +     .os_desc = {
86 +         .bFirstInterfaceNumber = 0,
87 +         .Reserved1 = htole32(1),
88 +         .CompatibleID = {0},
89 +         .SubCompatibleID = {0},
90 +         .Reserved2 = {0},
91 +     },
92 + };
93
94 #define STR_INTERFACE "AIO Test"
95 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
96 index 1f44a29..3dab7f1 100644
97 --- a/tools/usb/ffs-aio-example/simple/device_app/aio_simple.c
98 +++ b/tools/usb/ffs-aio-example/simple/device_app/aio_simple.c
99 @@ -55,16 +55,30 @@ static const struct {
100     struct usb_functionfs_descs_head_v2 header;
101     __le32 fs_count;
102     __le32 hs_count;
103 +    __le32 ss_count;
104 +    __le32 os_count;
105     struct {
106         struct usb_interface_descriptor intf;
107         struct usb_endpoint_descriptor_no_audio bulk_sink;
108         struct usb_endpoint_descriptor_no_audio bulk_source;
109     } __attribute__((packed)) fs_descs, hs_descs;
110 +    struct {
111 +        struct usb_interface_descriptor intf;
112 +        struct usb_endpoint_descriptor_no_audio sink;
113 +        struct usb_ss_ep_comp_descriptor sink_comp;
114 +        struct usb_endpoint_descriptor_no_audio source;
115 +        struct usb_ss_ep_comp_descriptor source_comp;
116 +    } __attribute__((packed)) ss_descs;
117 +    struct usb_os_desc_header os_header;
118 +    struct usb_ext_compat_desc os_desc;

```



```

119 +
120 } __attribute__((packed)) descriptors = {
121     .header = {
122         .magic = htole32(FUNCTIONFS_DESCRIPTOR_MAGIC_V2),
123         .flags = htole32(FUNCTIONFS_HAS_FS_DESC |
124 -             FUNCTIONFS_HAS_HS_DESC),
125 +             FUNCTIONFS_HAS_HS_DESC |
126 +             FUNCTIONFS_HAS_SS_DESC |
127 +             FUNCTIONFS_HAS_MS_OS_DESC),
128         .length = htole32(sizeof(descriptors)),
129     },
130     .fs_count = htole32(3),
131 @@ -113,6 +127,57 @@ static const struct {
132         .wMaxPacketSize = htole16(512),
133     },
134 },
135 + .ss_count = htole32(5),
136 + .ss_descs = {
137 +     .intf = {
138 +         .bLength = sizeof(descriptors.ss_descs.intf),
139 +         .bDescriptorType = USB_DT_INTERFACE,
140 +         .bInterfaceNumber = 0,
141 +         .bNumEndpoints = 2,
142 +         .bInterfaceClass = USB_CLASS_VENDOR_SPEC,
143 +         .iInterface = 1,
144 +     },
145 +     .sink = {
146 +         .bLength = sizeof(descriptors.ss_descs.sink),
147 +         .bDescriptorType = USB_DT_ENDPOINT,
148 +         .bEndpointAddress = 1 | USB_DIR_IN,
149 +         .bmAttributes = USB_ENDPOINT_XFER_BULK,
150 +         .wMaxPacketSize = htole16(1024),
151 +     },
152 +     .sink_comp = {
153 +         .bLength = sizeof(descriptors.ss_descs.sink_comp),
154 +         .bDescriptorType = USB_DT_SS_ENDPOINT_COMP,
155 +         .bMaxBurst = 4,
156 +     },
157 +     .source = {
158 +         .bLength = sizeof(descriptors.ss_descs.source),
159 +         .bDescriptorType = USB_DT_ENDPOINT,
160 +         .bEndpointAddress = 2 | USB_DIR_OUT,
161 +         .bmAttributes = USB_ENDPOINT_XFER_BULK,
162 +         .wMaxPacketSize = htole16(1024),
163 +     },
164 +     .source_comp = {
165 +         .bLength = sizeof(descriptors.ss_descs.source_comp),
166 +         .bDescriptorType = USB_DT_SS_ENDPOINT_COMP,
167 +         .bMaxBurst = 4,
168 +     },
169 + },
170 + .os_count = htole32(1),
171 + .os_header = {

```

```
172 +         .interface = htobe32(1),
173 +         .dwLength = htobe32(sizeof(descriptors.os_header) +
    sizeof(descriptors.os_desc)),
174 +         .bcdVersion = htobe32(1),
175 +         .wIndex = htobe32(4),
176 +         .bCount = htobe32(1),
177 +         .Reserved = htobe32(0),
178 +     },
179 +     .os_desc = {
180 +         .bFirstInterfaceNumber = 0,
181 +         .Reserved1 = htobe32(1),
182 +         .CompatibleID = {0},
183 +         .SubCompatibleID = {0},
184 +         .Reserved2 = {0},
185 +     },
186 };
187
188 #define STR_INTERFACE "AIO Test"
```