



USB Pipe Application Notes


Issue **00B02**

Date **2019-01-15**

Copyright © HiSilicon (Shanghai) Technologies Co., Ltd. 2019. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of HiSilicon (Shanghai) Technologies Co., Ltd.

Trademarks and Permissions

 , **HISILICON** , and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

HiSilicon (Shanghai) Technologies Co., Ltd.

Address: New R&D Center, 49 Wuhe Road, Bantian,
Longgang District,
Shenzhen 518129 P. R. China

Website: <http://www.hisilicon.com>

Email: support@hisilicon.com



About This Document

Purpose

This document introduces the USB pipe as a transmission solution between the AI coprocessor and the main control, and describes the involved APIs and operation procedures.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3516C	V500
Hi3516D	V300
Hi3516A	V300



Intended Audience

This document is intended for:




- Technical support engineers
- Software development developers

Symbol Conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
	Indicates an imminently hazardous situation which, if not avoided, will result in death or serious injury.
	Indicates a potentially hazardous situation which, if not avoided, could result in death or serious injury.



Symbol	Description
	Indicates a potentially hazardous situation which, if not avoided, may result in minor or moderate injury.
	Indicates a potentially hazardous situation which, if not avoided, could result in equipment damage, data loss, performance deterioration, or unanticipated results. NOTICE is used to address practices not related to personal injury.
	Calls attention to important information, best practices and tips. NOTE is used to address information not related to personal injury, equipment damage, and environment deterioration.

Change History

Changes between document issues are cumulative. The latest document issue contains all the changes made in earlier issues.

Issue 00B02 (2019-01-15)

This issue is the second draft release, which incorporates the following changes:

Section 1.2 is added, and chapter 4 is modified.

Issue 00B01 (2018-11-20)

This issue is the first draft release.



Contents

About This Document.....	i
Purpose.....	i
Related Versions.....	i
Intended Audience.....	i
Symbol Conventions	i
Change History.....	ii
1 Introduction.....	1
1.1 Overview	1
1.2 Supported Chip Models and Kernel Versions.....	1
2 Host End.....	2
2.1 User-Mode Interface	2
2.2 Operating Procedure.....	2
3 Device End.....	4
3.1 User-Mode Interface	4
3.2 Operating Procedure.....	4
4 Sample Instructions	7



1 Introduction

1.1 Overview

The USB pipe can be used as a transmission solution between the AI coprocessor and the main control (MC) through a USB port. It aims to implement the following:

- Transmit the data of the MC to the AI coprocessor through a USB port. Then the AI coprocessor processes the data and returns the result. A low transmission processing latency and high transmission performance must be ensured.
- The MCs of traditional devices such as IP cameras (IPC), network video recorders (NVR), and digital video recorder (DVR) have limited CPU and bus resources due to the restrictions of the original design specifications. Therefore, the resource consumption on the MC must be reduced. For example, you can reduce memory copy operations and CPU usage.

The USB pipe demo solution consists of four parts: USB pipe gadget driver, USB pipe class driver, host sample app, and device sample app. The solution has the following features:

- It adopts the USB bulk transfer mode. There are two bulk transfer channels—one is inbound, while the other is outbound.
- Currently, it supports only single-thread transmission. This transmission model is more suitable for hosts to send a large amount of data while receive only a small amount of data.
- The host sends data based on the buffer of consecutive physical addresses, implementing transmission with zero copy and saving CPU and bandwidth overheads.

Summary: The USB pipe solution is a data traffic model designed for the AI coprocessor. Especially for scenarios where a large amount of data is transmitted and a small amount of data is received, the solution is optimized and can deal with limited host CPU and bandwidth resources. Therefore, this solution is more suitable for the AI coprocessor.

1.2 Supported Chip Models and Kernel Versions

- The device end supports Hi3516C V500 and Hi3516D V300 and Linux 4.9.
- The host end supports Linux 3.10, Linux 3.18, and Linux 4.9, without requirements on the chip model.



2 Host End

2.1 User-Mode Interface

The user mode has only one input and output control (IOCTL) interface. An app sends or receives USB data through the IOCTL interface.

The commands are designed as follows:

- #define IOCTL_OPEN_USBPIPE 0x1 /* Opens the USB pipe. */
- #define IOCTL_CLOSE_USBPIPE 0x2 /* Closes the USB pipe. */
- #define IOCTL_STREAM_ON 0x3 /* Reserved. */
- #define IOCTL_STREAM_OFF 0x4 /* Reserved. */
- #define IOCTL_QUEUE_PHY_BUFF 0x5 /* Sends data. */
- #define IOCTL_DEQUEUE_RESULT 0x6 /* Receives data. */

Parameters:

```
struct usbpipe_ioctl_arg {  
    unsigned long addr; /* It is a physical address for the  
IOCTL_QUEUE_PHY_BUFF command, while a virtual address for the  
IOCTL_DEQUEUE_RESULT command. */  
    unsigned int len; /* It is the length of the sent data for the  
IOCTL_QUEUE_PHY_BUFF command, while returns the length of the read data  
for the IOCTL_DEQUEUE_RESULT command. */  
};
```

2.2 Operating Procedure

Step 1 Compile and generate **usbpipe.ko**.

Select a USB pipe from **menuconfig**.

The location of the USB pipe is as follows:

```
Device Drivers --->  
  [*] USB support ---->
```



```
<M> USB Pipe driver
```

```
< > NXP ISP 1760/1761 support
**> USB port drivers ***
< > USB Serial Converter support ----
*** USB Miscellaneous drivers ***
< > EMI 6|2m USB Audio interface support
< > EMI 2|6 USB Audio interface support
< > ADU devices from Ontrak Control Systems
< > USB 7-Segment LED Display
< > USB Diamond Rio500 support
< > USB Lego Infrared Tower support
< > USB LCD driver support
< > Cypress CY7C63xxx USB driver support
< > Cypress USB thermometer driver support
< > Siemens ID USB Mouse Fingerprint sensor support
< > Elan PCMCIA CardBus Adapter USB Client
< > Apple Cinema Display support
< > USB LD driver
< > PlayStation 2 Trance Vibrator driver support
< > IO Warrior driver support
< > USB testing driver
< > USB EHSET Test Fixture driver
< > iSight firmware loading support
< > USB YUREX driver support
< > Functions for loading firmware on EZUSB chips
< > USB3503 HSIC to USB20 Driver
< > USB4604 HSIC to USB20 Driver
< > USB Link Layer Test driver
<*> USB Pipe driver
<*> USB Physical Layer drivers --->
<*> USB Gadget Support --->
< > USB ULPI PHY interface support
```

Step 2 Run **insmod usbpipe.ko**. (location: **drivers/usb/misc/usbpipe.ko**)

Step 3 If the peer end is configured correctly and the USB connection is normal, the **/dev/usbpipe0** device node is generated.

Step 4 Run **usbpipe-sample** to control the **/dev/usbpipe0** device node to implement the host transmission function.



NOTE

You can run the **debugfs** command (after running **mount debugfs**, use **cat /sys/kernel/debug/usbpipe/queues**) to view the debugging information about TX and RX data packets.

----End



3 Device End

3.1 User-Mode Interface

After the device driver is installed and the device connection is ready, the `/dev/usbpipe0` device node is generated. The user program controls the data transmission of the USB pipe by operating the generated device node. The following three interfaces are implemented:

- **Poll:** Checks whether data can be read and written.
- **Read:** Reads data.
- **Write:** Writes data.

3.2 Operating Procedure

Step 1 Modify dtsi, and set the USB port to device mode.

```
#define USB_HOST 0
#if USB_HOST
    xhci_0@0x100e0000 {
        compatible = "generic-xhci";
        reg = <0x100e0000 0x10000>;
        interrupts = <0 27 4>;
        usb2-lpm-disable;
    };
#else
    hidwc3_0@0x100e0000 {
        compatible = "snps,dwc3";
        reg = <0x100e0000 0x10000>;
        interrupts = <0 27 4>;
        interrupt-names = "peripheral";
        maximum-speed = "high-speed";
        dr_mode = "peripheral";
    };
};
```

Step 2 Compile and generate `libcomposite.ko` and `g_usbpipe.ko`.

Select a USB pipe from `menuconfig`.

The location of the USB pipe is as follows:



```
Device Drivers --->
[*] USB support --->
    <*> DesignWare USB3 DRD Core Support
        DWC3 Mode Selection (Gadget only mode) --->
    <*> USB Gadget Support --->
        <M> USBPIPE Gadget

--- USB Gadget Support
[ ] Debugging messages (DEVELOPMENT)
[ ] Debugging information files (DEVELOPMENT)
[ ] Debugging information files in debugfs (DEVELOPMENT)
(2) Maximum VBUS Power usage (2-500 mA)
(2) Number of storage pipeline buffers
USB Peripheral Controller --->
< > USB functions configurable through configs
<M> USB Gadget Drivers
    < > Gadget Zero (DEVELOPMENT)
    < > Ethernet Gadget (with CDC Ethernet support)
    < > Network Control Model (NCM) support
    < > Gadget Filesystem
    < > Function Filesystem
    < > Mass Storage Gadget
    < > Serial Gadget (with CDC ACM and CDC OBEX support)
    < > Printer Gadget
    < > CDC Composite Device (Ethernet and ACM)
    < > CDC Composite Device (ACM and mass storage)
    < > Multifunction Composite Gadget
    < > HID Gadget
    <M> USBPIPE Gadget
    < > EHCI Debug Device Gadget
    < > USB Webcam Gadget
```

Step 3 `insmod libcomposite.ko` (location: `drivers/usb/gadget/libcomposite.ko`)

Step 4 Run `insmod g_usbpipe.ko` (location: `drivers/usb/gadget/legacy/g_usbpipe.ko`)

Step 5 If the peer end is configured correctly and the USB connection is normal, the `/dev/usbpipe0` device node is generated.

Step 6 Run `usbpipe-sample` to control the `/dev/usbpipe0` device node to implement the host transmission function.



NOTE

You can run the `debugfs` command (after running `mount debugfs`, use `cat /sys/kernel/debug/usbpipe/queues`) to view the detailed information about TX and RX queues of the driver.

----End



NOTICE

- To simply the design:
 - (1) USB plugging and unplugging are not supported.
 - (2) The .ko driver cannot be dynamically unloaded.
 - Only one channel can be read or written.
 - Currently, the **device-sample** of the IOCTL at the host end supports only the close command. If the host sample exits abnormally, the **device-sample** will receive an error code.
-



4 Sample Instructions

Perform the following steps:

Step 1 The host devices are connected through USB ports (board-level connection in XVR scenarios).

Step 2 Prepare the host driver and sample.

- Obtain the USB pipe class driver code and compile it to `.ko` or directly obtain the `.ko` binary file.
- Compile the sample: **`arm-hixxxx-linux-gcc -Wall -Wextra -g -o usbpipe-sample usbpipe-sample.c -lpthread`** (`arm-hixxxx-linux-gcc` is the compiler that is used).
- Run **`insmod usbpipe.ko`** (the dependent USB host driver needs to be installed first) or **`modprobe usbpipe.ko`**.

Step 3 Prepare the driver and sample of the device end.

- Obtain the USB pipe gadget driver code and compile it to **`g_usbpipe.ko`**.
- Compile the sample: **`arm-hixxxx-linux-gcc usbpipe-gadget.c -g -o usbpipe-gadget`** (`arm-hixxxx-linux-gcc` is the compiler that is used).
- Run **`insmod g_usbpipe.ko`** (the dependent USB device driver needs to be installed first) or **`modprobe g_usbpipe.ko`**.

Step 4 Execute the sample program.

- On the device end, run **`./usbpipe-gadget`**.
- On the host end, run **`./usbpipe-sample`**.

The following information is displayed on the host end, indicating that the test is normal:



```
usbpipe-sample: info: Speed is 38.5649 MB/s
usbpipe-sample: info: Speed is 38.4109 MB/s
usbpipe-sample: info: RTT information:
usbpipe-sample: info: rounds: 256
usbpipe-sample: info: cur_rtt:26.119 ms
usbpipe-sample: info: avg_rtt:26.146 ms
usbpipe-sample: info: Speed is 38.3421 MB/s
usbpipe-sample: info: Speed is 38.31 MB/s
usbpipe-sample: info: RTT information:
usbpipe-sample: info: rounds: 512
usbpipe-sample: info: cur_rtt:26.087 ms
usbpipe-sample: info: avg_rtt:26.154 ms
usbpipe-sample: info: Speed is 38.3028 MB/s
usbpipe-sample: info: Speed is 38.2933 MB/s
```

NOTICE

In the `data_thread` function of the `usbpipe-sample.c` file, the address parameter `data.addr` needs to be set according to the memory allocation. The code also contains related comments. The `usbpipe-sample` and `usbpipe-gadget` codes are for reference only, which can be modified as required.

----End