

## How to use USB driver

\*\*\*\*\*

### 1. Introduction

\*\*\*\*\*

ADSP21535 has one USB Device Controller (UDC) which supports four types of USB transfers (Control, Bulk, Interrupt, ISO).

The USB driver makes it possible for the applications in user mode to access the UDC. The USB driver processes USB enumeration automatically and provides the interface for USB to transfer information to the application.

With the USB driver, the user application will be able to communicate with USB Host (PC) via three types of transfer: Bulk, Interrupt, ISO. The Control transfer will be processed by USB driver internally, no related interface will be visible to user application.

\*\*\*\*\*

### 2. System interface

\*\*\*\*\*

The USB driver register itself is a standard character device driver.

The Major number of USB driver is 221. The device name is /dev/adi\_usb

So the standard file operations can be used to access the USB driver:

1) open()  
open("/dev/adi\_usb",...) will return the fd of USB driver  
The USB driver can be accessed by multi-process at the same time.

2) close()

3) read()  
USB driver does \*NOT\* support this operation

4) write()  
USB driver does \*NOT\* support this operation

5) ioctl()  
Following ioctl commands are supported

CMD\_USB\_BULK\_INTR\_READ:

Read from USB BULK pipe.

The argument points to structure of 'usb\_bulk\_intr\_ioctl' show as below.

```
typedef struct
{
    unsigned char * buf;
    int len;
    int flag;
} usb_bulk_intr_ioctl;
```

The 'buf' is the buffer to receive data.  
The 'len' is the count of data to receive.

The 'flag' must be 1 means read from BULK pipe

The number of received data will be returned

`CMD_USB_BULK_INTR_WRITE:`

Write to USB BULK/INTERRUPT pipe.

The argument points to structure of `usb_bulk_intr_ioctl`.

The 'buf' points to the buffer to send.

The 'len' is the count of data to send.

The 'flag' is 1 means write to BULK pipe

The 'flag' is 2 means write to INTERRUPT pipe

The number of data sent out will be returned

`CMD_USB_ISO_READ:`

Read from USB ISO pipe.

The argument points to structure of 'usb\_iso\_data' show as below.

```
typedef struct
```

```
{  
    unsigned short len;  
    char * data_buf;  
} usb_iso_frame;
```

```
typedef struct
```

```
{  
    unsigned long frame_num;  
    usb_iso_frame frame[1];  
} usb_iso_data;
```

The 'frame\_num' is the number of frames to receive.

The 'frame' points to each ISO frames.

The 'frame\_num' will be set with the number of frames actually received when return. The 'len' of each ISO frame will also be set to the length of the frame data. The data of each ISO frame will be copied to buffer pointed by 'data\_buf' of the frame.

`CMD_USB_ISO_WRITE:`

Write USB ISO pipe.

The argument points to structure of 'usb\_iso\_data'.

The 'frame\_num' is the number of frames to be transmitted.

The 'frame' points to each ISO frames.

The 'frame\_num' shall be set with the number of frames to send. The 'len' of each ISO frame shall also be set to the length of the frame data. The 'data\_buf' of each ISO frame shall point to the data buffer of the frame.

The 'frame\_num' will be set with the number of frames actually transmitted when return.

NOTE:

A set of user space APIs are available to simplify programming.

\*\*\*\*\*

3. User API for USB driver

\*\*\*\*\*

1) `usb_pipe_open()`

Syntax:

```
usb_pipe_handle usb_pipe_open (int pipe_type)
```

Description:

Open one of the pipes of the UDC. One pipe consists of two endpoints that have the same type (Bulk, Interrupt or Isochronous)

The specific pipe will NOT be accessible until it has been opened.

NOTE:

When Interrupt pipe type is specified, the returned pipe handle can NOT be used to call `usb_pipe_read()`. Since only USB Interrupt IN (data to USB Host) is supported by USB kernel driver and USB Host.

Parameters:

<code>pipe_type</code>	The type of the pipe to be opened
1:	Bulk pipe
2:	Interrupt pipe
3:	Isochronous pipe

Return:

The handle of the opened endpoint will be returned if succeed, otherwise negative error number will returned.

2) `usb_pipe_close()`

Syntax:

```
void usb_pipe_close (usb_pipe_handle pipe)
```

Description:

Close the pipe that had been opened

Parameters:

`pipe` The handle returned by `usb_pipe_open()`

3) `usb_pipe_write()`

Syntax:

```
int usb_pipe_write (usb_pipe_handle pipe, char * buf, int len)
```

Description:

Send data to USB Host through the specific pipe. The data will be stored into internal buffer first, and will be written into the FIFO of the specific endpoint of UDC whenever the room of FIFO is available. Eventually, the data will be sent to USB Host.

Parameters:

<code>pipe</code>	The handle returned by <code>usb_pipe_open()</code>
<code>buf</code>	The pointer of the buffer for data to be sent
<code>len</code>	The length of data

Return:

The bytes of the data sent out will be returned if success, otherwise negative error number will be returned

4) `usb_pipe_read()`

Syntax:

```
int usb_pipe_read (usb_pipe_handle pipe, char * buf, int len)
```

Description:

Get data sent by USB Host via specific USB endpoint. Whenever the data from USB Host are received, they will be stored into internal buffer first. Then the data will be returned to the caller whenever this function is called.

Parameters:

`pipe` The handle returned by `usb_pipe_open()`  
`buf` The pointer of the buffer for data to be sent  
`len` The length of data

Return:

The bytes of the data sent out will be returned if success, otherwise negative error number will be returned

5) `usb_pipe_iso_write()`

Syntax:

```
int usb_pipe_iso_write (usb_pipe_handle pipe, usb_iso_data * iso_data)
```

Description:

Send Isochronous data to USB Host.

Parameters:

`pipe` The handle returned by `usb_pipe_open()`  
`iso_data` ISO frames descriptor

Return:

The bytes of the data sent out will be returned if success, otherwise negative error number will be returned

6) `usb_pipe_iso_read()`

Syntax:

```
int usb_pipe_iso_read (usb_pipe_handle pipe, usb_iso_data * iso_data)
```

Description:

Get Isochronous data sent by USB Host.

Parameters:

`pipe` The handle returned by `usb_pipe_open()`  
`iso_data` ISO frames descriptor

Return:

The bytes of the data sent out will be returned if success, otherwise

negative error number will be returned

```
*****
4. Usage of USB driver
*****
```

- ```
1) Read/Write from/to USB BULK/INTERRUPT pipe
a) Call usb_pipe_open() to open pipe
b) Call usb_pipe_read()/write() to read/write
c) Call usb_pipe_close() to close pipe
```

- ```
2) Read from USB ISO pipe
a) Call usb_pipe_open() to open ISO pipe
```

- ```
b) Allocate 'usb_iso_data' structure
```

```
iso_data = (usb_iso_data *) malloc(sizeof(usb_iso_data) +
                                   (frame_num - 1) * sizeof(usb_iso_frame));
```

- ```
c) Set the 'frame_num' of usb_iso_data structure with number frames
you want to read.
```

```
iso_data->frame_num = frame_num;
```

- ```
d) Setup buffer for each frame
```

```
for (i = 0; i < frame_num; i++)
    iso_data->frame[i].data_buf = malloc(...);
```

```
NOTE: Sufficient memory shall be allocated for each frame.
Buffer of each frame shall be able to hold at least 8-byte data.
```

- ```
e) Call usb_pipe_iso_read() to read ISO frames
```

- ```
f) Check 'iso_data->frame_num' to determine the number received frames.
```

- ```
g) Check each frame's length and data
```

```
for (i = 0; i < iso_data->frame_num; i++)
{
    len = iso_data->frame[i].len;
    memcpy(buf, iso_data->frame[i].data_buf, len);
    .....
}
```

- ```
h) Call usb_pipe_close() to close the ISO pipe
```

- ```
3) Write to USB ISO pipe
```

- ```
a) Call usb_pipe_open() to open ISO pipe
```

- ```
b) Allocate 'usb_iso_data' structure
```

```
iso_data = (usb_iso_data *) malloc(sizeof(usb_iso_data) +
                                   (frame_num - 1) * sizeof(usb_iso_frame));
```

- ```
c) Set the 'frame_num' of usb_iso_data structure with number frames
you want to write.
```

```
iso_data->frame_num = frame_num;
```

```
d) Setup buffer and length for each frame
for (i = 0; i < frame_num; i++)
{
    iso_data->frame[i].data_buf = malloc(len);
    memcpy(iso_data->frame[i].data_buf, ...);
    iso_data->frame[i].len = len;
}
```

NOTE: Length of each frame shall NOT exceed 8-byte data.

e) Call `usb_pipe_iso_write()` to write ISO frames

f) Check `'iso_data->frame_num'` to determine the number of frames transmitted.

g) Call `usb_pipe_close()` to close the ISO pipe

For detail information for usage for USB drive, pls refer to `usb_test.c`

```
*****
```

5. NOTE

```
*****
```

- 1) Read from Interrupt pipe is NOT supported.
- 2) The length of each ISO frames sent to USB Host must be same as the maximum packet size of ISO IN endpoint. It is set to 8 bytes now.
- 3) Don't try to write more than 63 bytes to the Bulk OUT endpoint from USB Host(PC) each time. The size of Bulk OUT DMA buffer is only 63-bytes. Sending 64-byte packet will be NAKed forever!  
The root cause of this issue is a UDC hardware bug. The bug will prevent us from getting a correct data length for the Bulk OUT transfer unless we make the size of DMA buffer less than max packet size (64-bytes).