

PL25A1 USB 2.0 Host-to-Host Bridge Controller

SDK Integration Guide

1 Introduction

Prolific PL25A1 SDK directly utilized libusb to provide the interface of device control and data transfer to the upper software, so the application developer can control PL25A1 IC easy through the SDK. libusb is a very popular and powerful USB library running in user space, and it's able to cross many popular operating systems. This SDK provides the cross-platform sample codes which can be run on Windows, Linux, and MAC OS X. It means that the customers or end users can easily develop their USB host-to-host applications on all popular platforms.

This document will illustrate the following:

1. The directory tree structure of PL25A1 SDK
2. Building sample code processes for Windows, Linux, and MAC OS X
3. Run built sample program

2 Directory tree

There are four directories under SDK, and each of them is listed below:

libusb – It includes whole libusb v1.0.21 source from the library developer.

MSVC – It includes Visual Studio solution and projects. The built libraries and executable files will be generated and stored under this directory.

Source – It includes all sample codes which cross Windows, Linux, and MAC OS X operating systems. The three sample programs are as follows:

- listdev: To list PL25A1 device and show its information
- transmit: To transmit data from sender to receiver
- benchmark: To do performance benchmark for the sender side

XCode – It includes MAC XCode workspace and projects.

3 Build code

The following sections show the building processes for Windows, Linux, and MAC OS

X individually.

a. Windows

To build codes and developing applications for Windows, Microsoft Visual Studio and SDK must be installed. VS 2015 is selected to demo the operating procedures because of its popularity. After necessary software and components installation, the user can build the sample code step by step:

- (1) Open Visual Studio solution file “PL25A1_Libusb_SDK\MSVC\MSVC.sln” and four projects will be listed in the left side. Refer to Figure 1 - MSVC solution.

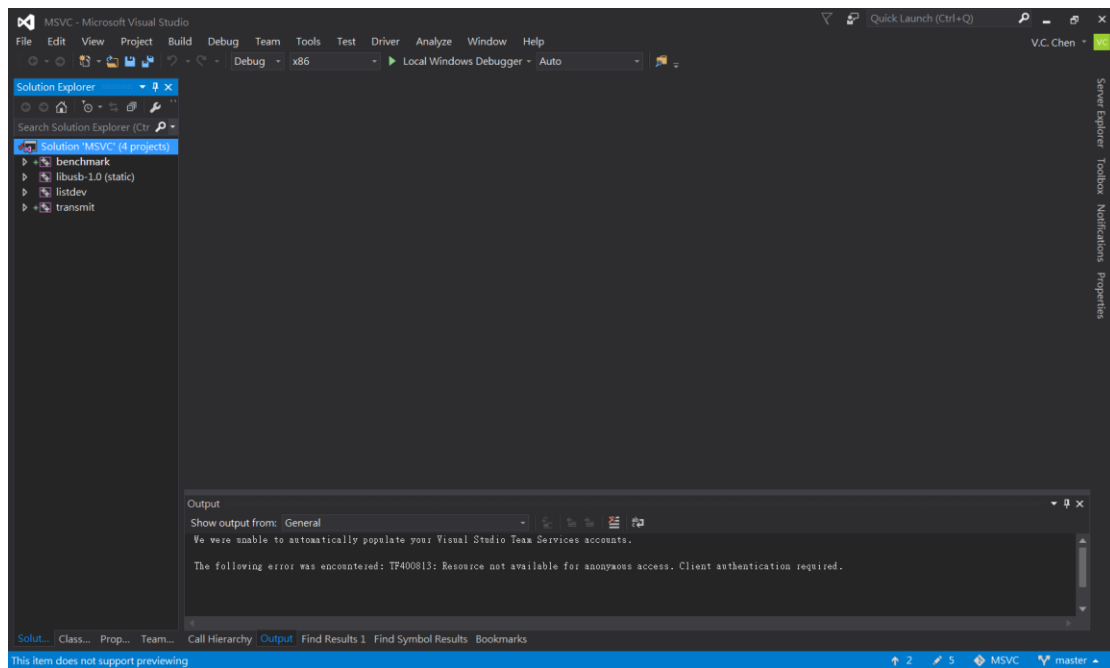


Figure 1 - MSVC solution

- (2) Select Project 'benchmark' and right-click it, and the menu shows. Refer to Figure 2 – Configure Project Properties.

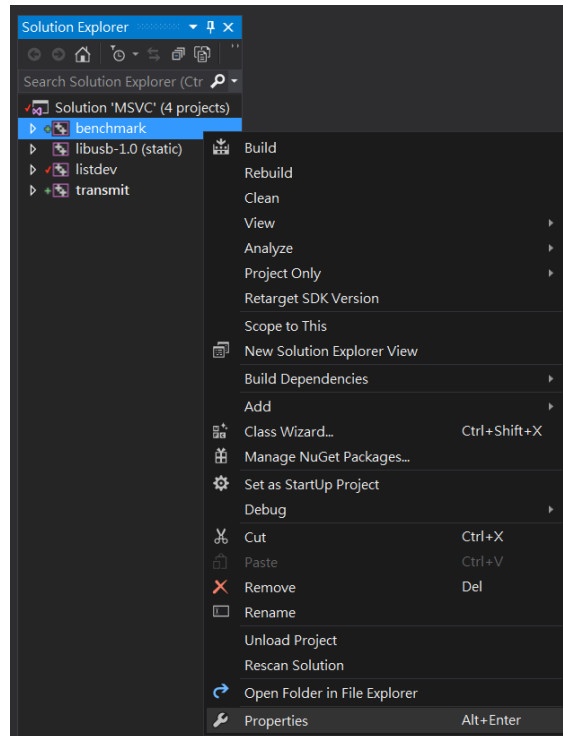


Figure 2 – Configure Project Properties

- (3) Left-click "Properties" and Property Pages show. Refer to Figure 3 – Configuration Properties - General.

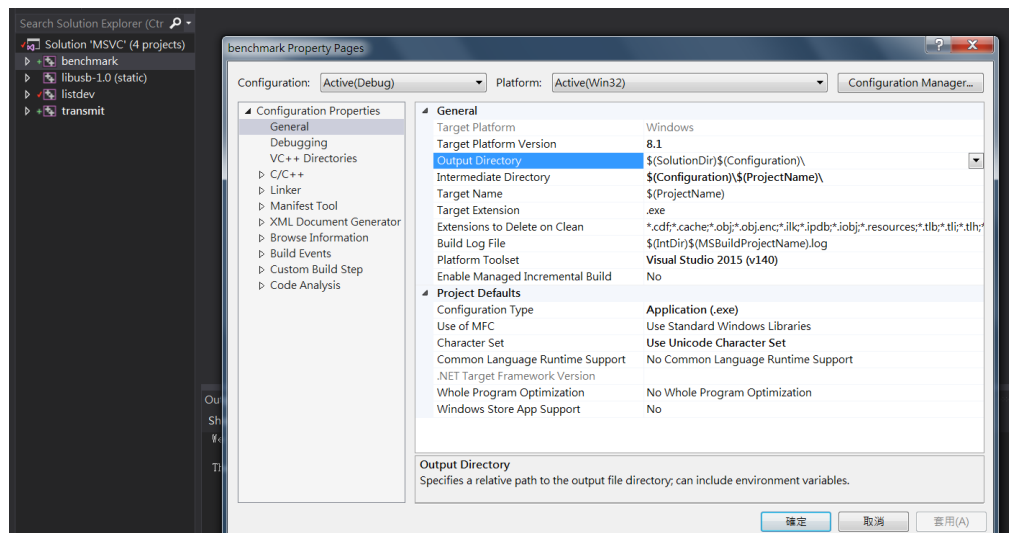


Figure 3 – Configuration Properties - General

- (4) Select and configure "Output Directory" and "Intermediate Directory" as follows:

benchmark

Output Directory: \$(SolutionDir)\$(Configuration)\

Intermediate Directory: \$(Configuration)\\$(ProjectName)\

- (5) Repeat Step (2) to (4) for Project libusb-1.0 (static), listdev, and transmit, and the configurations are as follows:

libusb-1.0 (static)

Output Directory: \$(SolutionDir)\$(Configuration)\lib\

Intermediate Directory: \$(SolutionDir)\$(Configuration)\lib\libusb-1.0\

listdev

Output Directory: \$(SolutionDir)\$(Configuration)\

Intermediate Directory: \$(Configuration)\\$(ProjectName)\

transmit

Output Directory: \$(SolutionDir)\$(Configuration)\

Intermediate Directory: \$(Configuration)\\$(ProjectName)\

- (6) Select Solution 'MSVC' (4 projects) and right-click it, and the menu shows.
- (7) Left-click "Build Solution" to build whole solution which contains libusb and three sample programs. Refer to Figure 4 – Build Solution and Figure 5 – Show output from: Build.

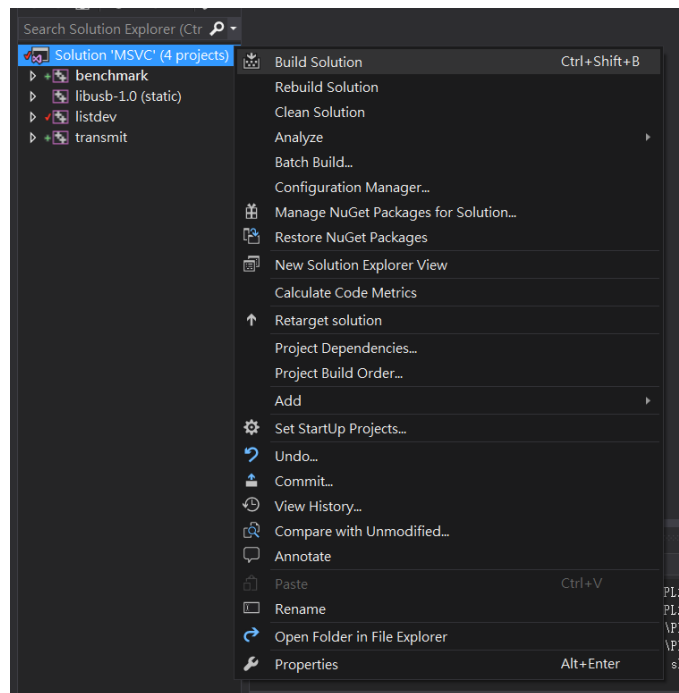


Figure 4 – Build Solution

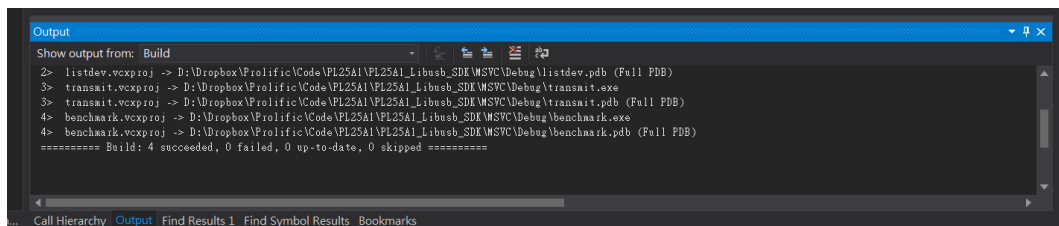


Figure 5 – Show output from: Build

Following shows how to debug specific sample code:

- (1) Select the specific project to debug (listdev in this illustration), and right-click it. Refer to Figure 6 – Open project menu.

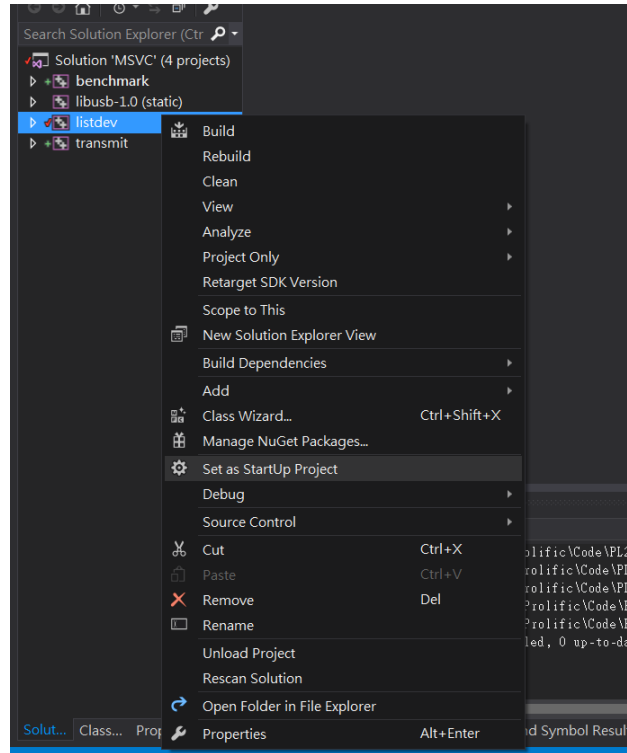


Figure 6 – Open project menu

- (2) Select “Set as StartUp Project”, and left-click it.
- (3) Select “Debug”, and then select and left-click “Start Debugging” to start debugging. Refer to Figure 7 – Start Debugging.

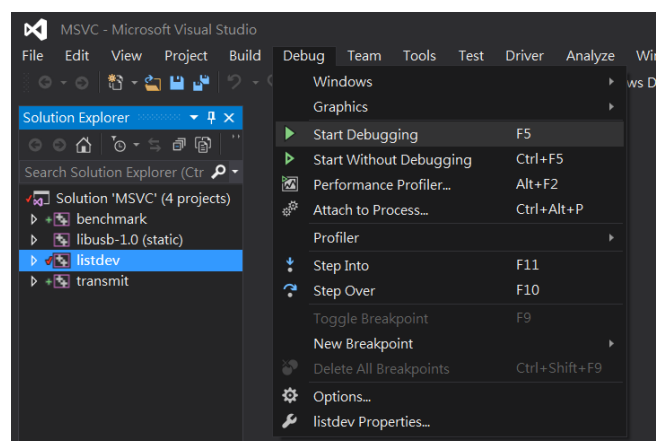


Figure 7 – Start Debugging

b. Linux

GCC must be installed to build codes and developing applications for Linux. Ubuntu is a very popular Linux distribution, so it is selected to demo the operating procedures. To build the sample codes, libusb source and package must be configured, built, and installed first.

Build and install libusb

Assume that PL25A1 Source is stored in ~/ , and follow the steps for libusb installation:

- (1) Change directory to the directory 'libusb'

```
~$ cd PL25A1_Libusb_SDK/libusb
```

- (2) Run the command to configure, make , and install

```
~/PL25A1_Libusb_SDK/libusb $ ./configure && make && sudo make install
```

After that, the built libraries will be installed to /usr/local/lib.

Build sample codes

To build sample codes, do the below:

- (1) Change directory to the directory 'Source'

```
~$ cd PL25A1_Libusb_SDK/Source
```

- (2) Just type 'make'

```
~$ make
```

listdev, transmit, and benchmark programs will be generated in the same directory.

c. MAC OS X

Xcode must be installed to build codes and developing applications for MAC OS X.

- (1) Open PL25A1 Xcode workspace file from

PL25A1_Libusb_SDK/Xcode/Xcode.xcworkspace. The opened workspace looks like Figure 8 – Xcode workspace.

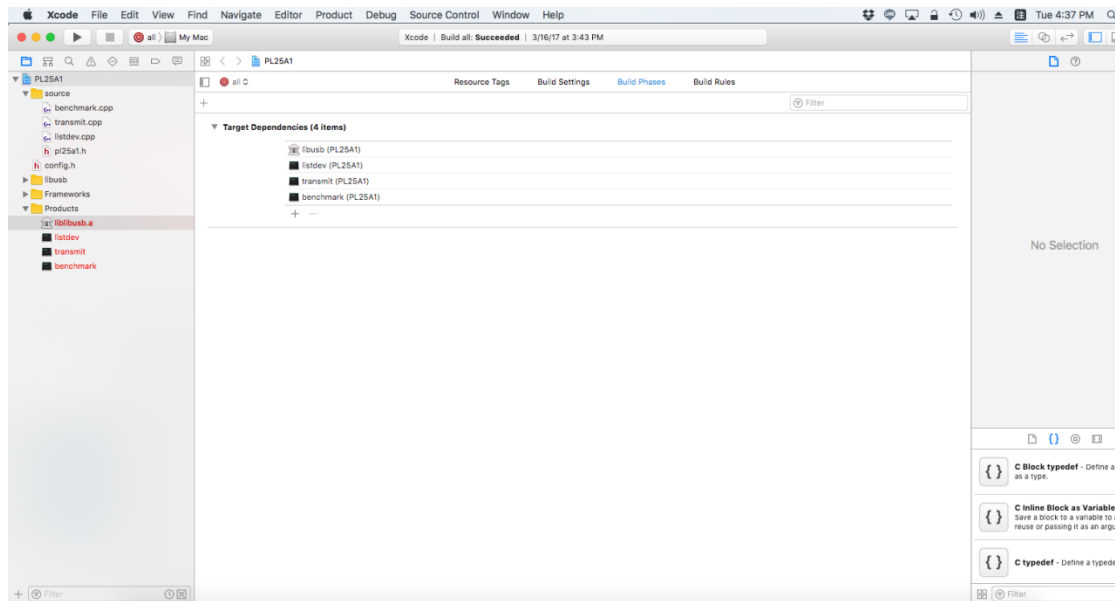


Figure 8 – Xcode workspace

- (2) Ensure that all targets are selected to build as Figure 9 – Target to build and Figure 10 – Select ‘all’ to build.

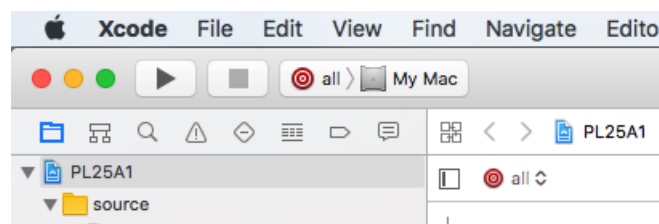


Figure 9 – Target to build

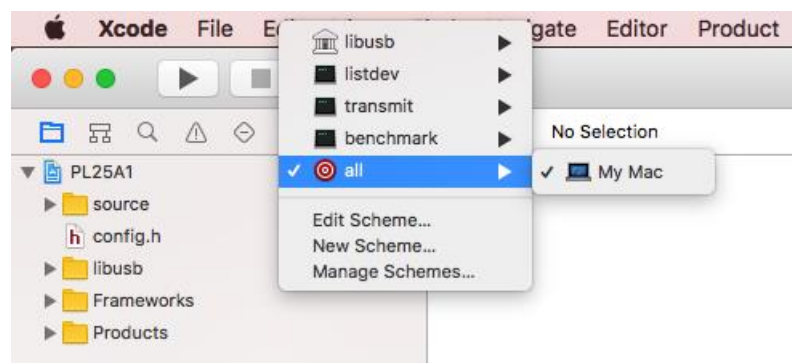


Figure 10 – Select ‘all’ to build

- (3) Select ‘Product’ in the menu, and then select and click “Build” to start building process. Refer to Figure 11 – Build the codes.

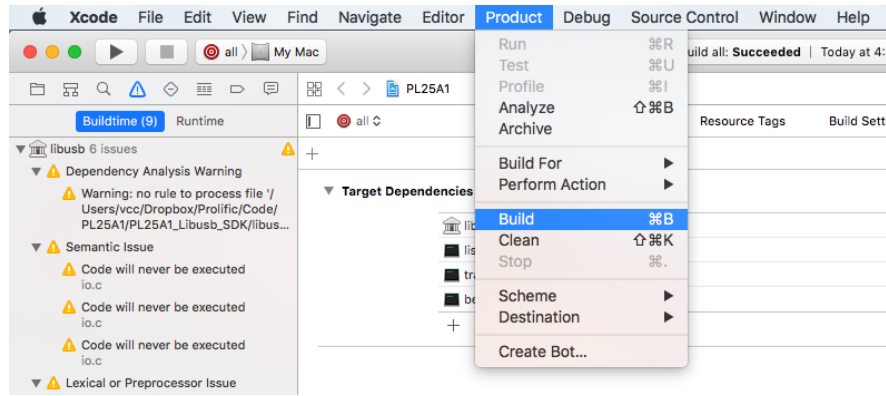


Figure 11 – Build the codes

4 Run sample codes

Hardware equipment must be setup before running the sample programs. There are two computers and one PL25A1 cable in this test environment. One computer is called PC_A running Windows, and the other is called PC_B running Ubuntu. One side of PL25A1 cable is connected to one USB port of PC_A, and the other is connected to one USB port of PC_B. After the setup, environment should look like as Figure 12 – Hardware environment setup.

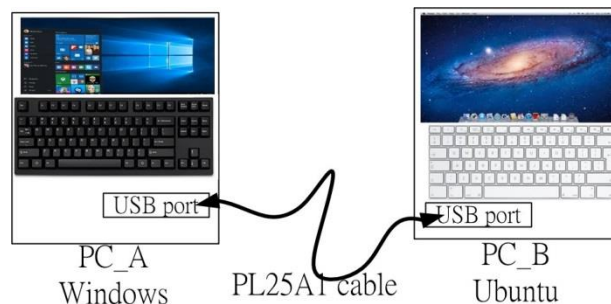


Figure 12 – Hardware environment setup

When hardware is ready, the next is to start executing the programs. Running them in Windows, MAC OS X, and Linux are very similar, but there are two differences between them:

- (1) Running in DOS command prompt of Windows or running in Terminal of Ubuntu/MAC OS X.
- (2) The 'sudo' must be used for each program in Ubuntu and MAC OS X.

The following shows the procedures for the test environment:

Run listdev in Windows

- (1) Open DOS command prompt in Windows on PC_A. Look like as Figure 13 – DOS

command prompt.

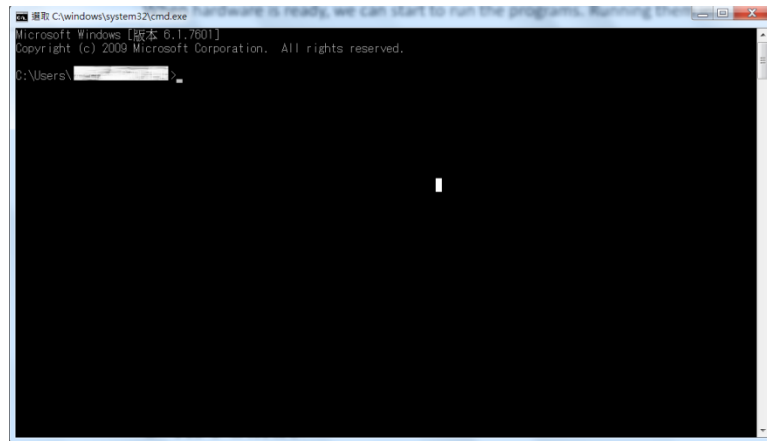


Figure 13 – DOS command prompt

- (2) Change directory to “PL25A1_Libusb_SDK\MSVC\Debug”
- (3) To run listdev program is very easy, just type ‘listdev’. Information of the found PL25A1 device will be shown as Figure 14 – Run listdev in Windows.

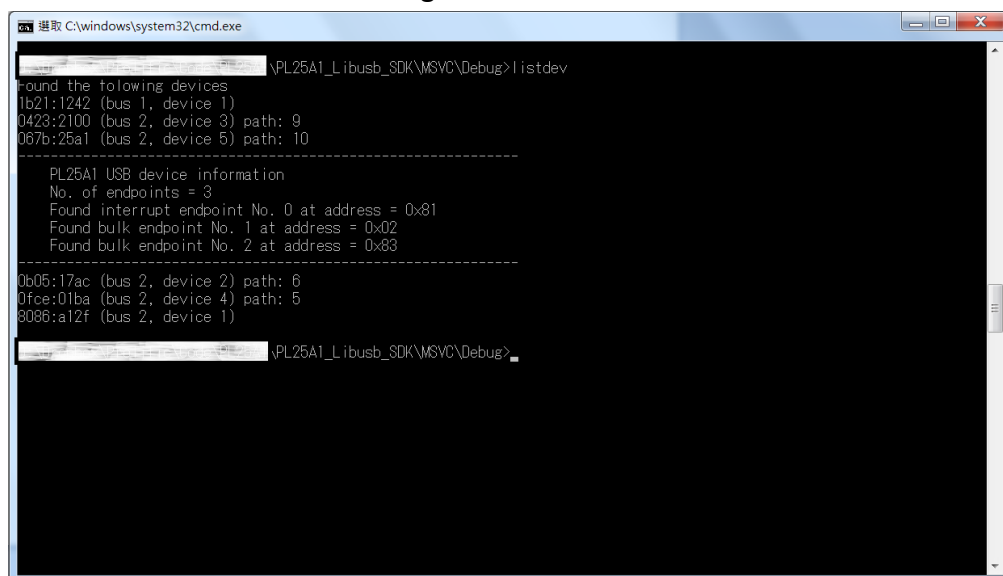


Figure 14 – Run listdev in Windows

Run listdev in Ubuntu

- (1) Open Terminal in Ubuntu on PC_B. Look like as Figure 15 – Terminal in Ubuntu.

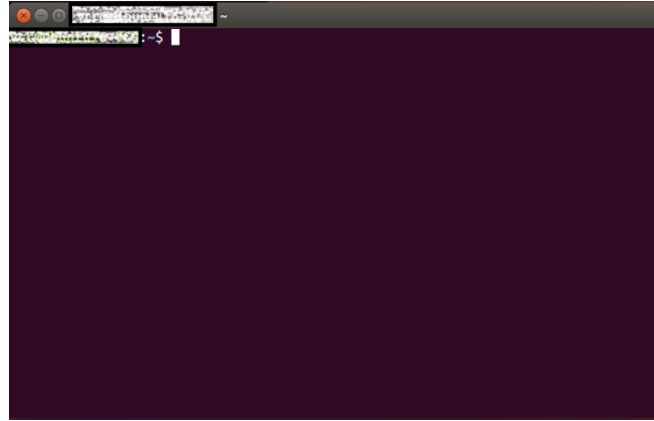


Figure 15 – Terminal in Ubuntu

- (2) Change directory to “PL25A1_Libusb_SDK/Source”
- (3) To run listdev program, just type ‘sudo ./listdev’ in Terminal. Information of the found PL25A1 device will be shown as Figure 16 – Run listdev in Ubuntu 16.04

```

-rw-rw-r-- 1 vcc vcc 3201  14 16:14 pl25a1.h
-rw-rw-r-- 1 vcc vcc 2711  16 20:16 Readme.txt
-rwxrwxr-x 1 vcc vcc 14352  19 20:36 transmit
-rw-rw-r-- 1 vcc vcc 12436  14 20:16 transmit.cpp
~/PL25A1_Libusb_SDK/Source$ sudo ./listdev
[sudo] password for user:
Found the following devices
093a:2521 (bus 2, device 3) path: 1.8
8087:0024 (bus 2, device 2) path: 1
1d6b:0002 (bus 2, device 1)
8087:0024 (bus 1, device 2) path: 1
1d6b:0002 (bus 1, device 1)
1d6b:0003 (bus 4, device 1)
067b:25a1 (bus 3, device 2) path: 3
-----
PL25A1 USB device information
No. of endpoints = 3
Found interrupt endpoint No. 0 at address = 0x81
Found bulk endpoint No. 1 at address = 0x02
Found bulk endpoint No. 2 at address = 0x83
-----
1d6b:0002 (bus 3, device 1)
$

```

Figure 16 – Run listdev in Ubuntu 16.04

Run transmit in Windows and Ubuntu

Warning!!! Before proceeding, refer to “Application Note: Build and Install Modified Kernel Driver for Ubuntu” to replace with modified plusb.ko file. The certificate files (signing_key.x509 and signing_key.pem) and modified plusb drivers (plusb.c) are stored under the directory /Source. This pre-installation is very important for Linux distribution. Don’t run transmit or benchmark before this. If the user has run the program, detach both sides of PL25A1 cable and then re-attach them to two computers. This action will reset hardware to normal status.

The procedure is a little complicated for running transmit program. In the demo, one transmit program will run as receiver in Windows and the other will run as sender in Ubuntu. The steps are as below:

- (1) Open DOS command prompt in Windows on PC_A. Look like as Figure 13 – DOS

command prompt.

- (2) Change directory to “PL25A1_Libusb_SDK\MSVC\Debug”
- (3) To run transmit program as receiver here, type ‘transmit recv’. After that, the receiver will wait the sender transmit the data. The output shows as Figure 17 – Run transmit as receiver.

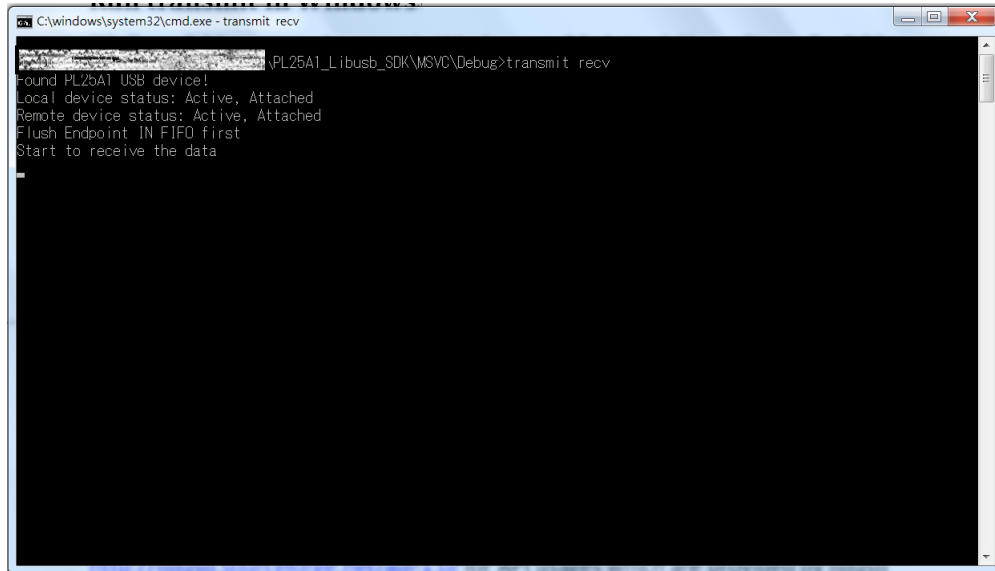


Figure 17 – Run transmit as receiver

- (4) Open Terminal in Ubuntu on PC_B. Look like as Figure 15 – Terminal in Ubuntu.
- (5) Change directory to “PL25A1_Libusb_SDK/Source”
- (6) To run transmit program as sender here, type “sudo ./transmit send”. It looks like as Figure 18 - Run transmit as sender.

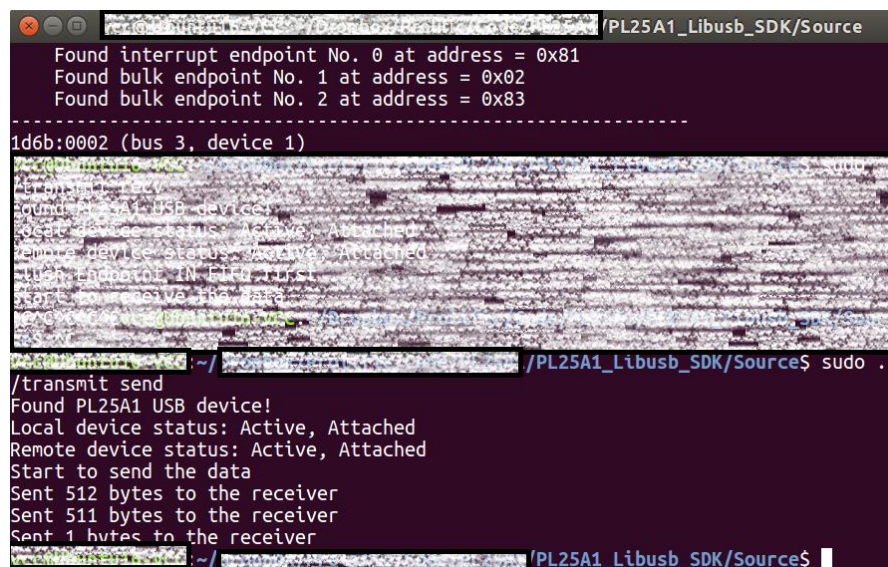
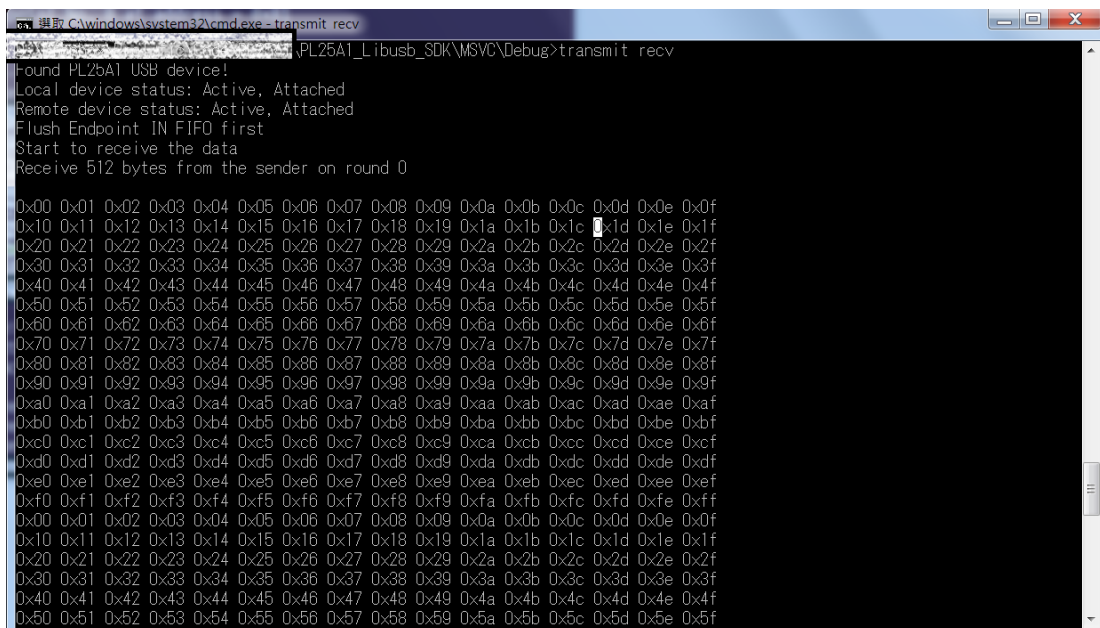


Figure 18 - Run transmit as sender

- (7) In receiver side, DOS command prompt shows the output of received data. It looks like Figure 19 – Output of receiver side.



```

cmd - 运行 C:\windows\system32\cmd.exe - transmit recv
PL25A1_Libusb_SDK\MSVC\Debug>transmit recv
Found PL25A1 USB device!
Local device status: Active, Attached
Remote device status: Active, Attached
Flush Endpoint IN FIFO first
Start to receive the data
Receive 512 bytes from the sender on round 0

0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f
0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f
0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x2a 0x2b 0x2c 0x2d 0x2e 0x2f
0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f
0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f
0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57 0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f
0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x6a 0x6b 0x6c 0x6d 0x6e 0x6f
0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77 0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f
0x80 0x81 0x82 0x83 0x84 0x85 0x86 0x87 0x88 0x89 0x8a 0x8b 0x8c 0x8d 0x8e 0x8f
0x90 0x91 0x92 0x93 0x94 0x95 0x96 0x97 0x98 0x99 0x9a 0x9b 0x9c 0x9d 0x9e 0x9f
0xa0 0xa1 0xa2 0xa3 0xa4 0xa5 0xa6 0xa7 0xa8 0xa9 0xaa 0xab 0xac 0xad 0xae 0xaf
0xb0 0xb1 0xb2 0xb3 0xb4 0xb5 0xb6 0xb7 0xb8 0xb9 0xba 0xbb 0xbc 0xbd 0xbe 0xbf
0xc0 0xc1 0xc2 0xc3 0xc4 0xc5 0xc6 0xc7 0xc8 0xc9 0xca 0xcb 0xcc 0xcd 0xce 0xcf
0xd0 0xd1 0xd2 0xd3 0xd4 0xd5 0xd6 0xd7 0xd8 0xd9 0xda 0xdb 0xdc 0xdd 0xde 0xdf
0xe0 0xe1 0xe2 0xe3 0xe4 0xe5 0xe6 0xe7 0xe8 0xe9 0xea 0xeb 0xec 0xed 0xee 0xef
0xf0 0xf1 0xf2 0xf3 0xf4 0xf5 0xf6 0xf7 0xf8 0xf9 0xfa 0xfb 0xfc 0xfd 0xfe 0xff
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f
0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f
0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x2a 0x2b 0x2c 0x2d 0x2e 0x2f
0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f
0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f
0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57 0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f

```

Figure 19 – Output of receiver side

Run benchmark in Windows and Ubuntu

The procedure is almost same as “Run transmit in Windows and Ubuntu” except replacing ‘transmit’ with ‘benchmark’ in DOS command prompt/Terminal.

5 API Guide

The SDK is based on the famous libusb directly, so the user can refer to <http://libusb.sourceforge.net/api-1.0/> for API usages which are provided by libusb organization.