Storage User Manual

This document describes how to run storage application on Ameba

Table of Contents

1 Introduction

Storage is a key feature of embedded system. Ameba provides flexible method of storage management. In this document, three kinds of application scenarios will be mentioned.

1. SD Card Read/Write
2. USB MSC
3. Flash Read/Write

-SD Card Read/Write: Ameba works as a SD host device write data to and read data back from SD card via Fatfs APIs.

-USB MSC: Ameba works as a USB mass storage device and use SD card as its physical memory medium. It can be recognized by any USB host device such as windows machine.

-Flash read/Write: Ameba provides on board flash memory component which can be used as storage device. The data can be written to/ read back from flash memory via Fatfs API.

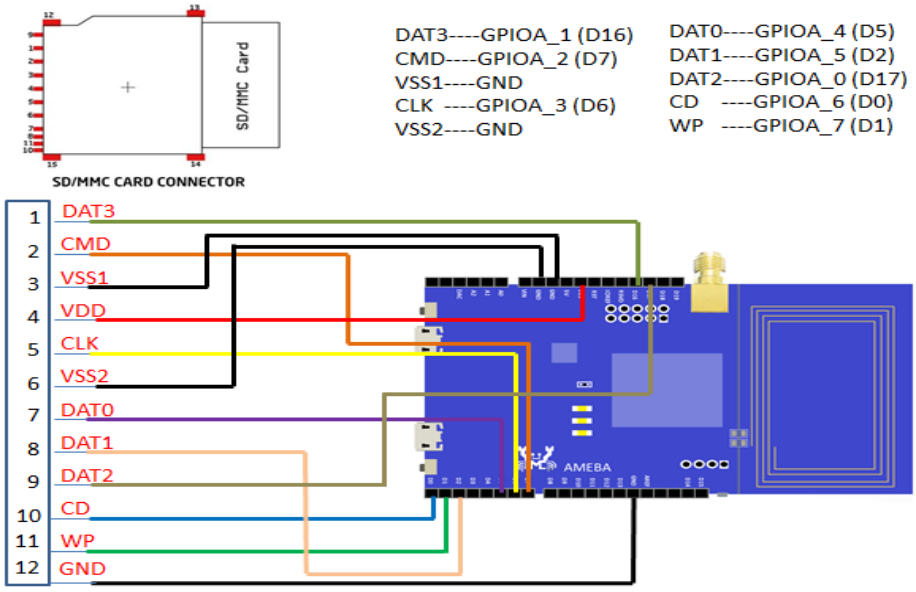
2 SD Card Read/Write

Ameba can access to SD card because it has integrated SD host controller which has a 4 bits parallel SD/MMC interface.

In this application, embedded file system (FatFs) will run on Ameba to manage the SD memory card, and an example is provided in SDK showing how it works.

2.1 Hardware setup

Connect your Ameba DEV board with SD/MMC card connector before moving on, you can refer to the follow picture. After then, plug in a compatible SD card to the card connector.



2.2 How to run the example

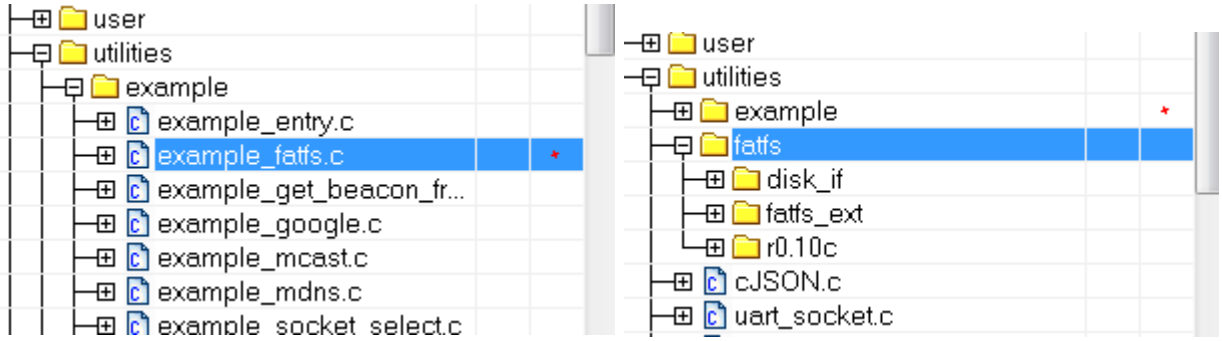
Step 1, config FATFS example with SD memory card in platform_opts.h as follows.

```
/* For FATFS example*/
#define CONFIG_EXAMPLE_FATFS                1
#if CONFIG_EXAMPLE_FATFS
#define CONFIG_FATFS_EN 1
#if CONFIG_FATFS_EN
// fatfs version
#define FATFS_R_10C
// fatfs disk interface
#define FATFS_DISK_USB   0
#define FATFS_DISK_SD    1
#endif
#endif
```

Step 2, rebuild the project and download firmware to DEV board.

2.3 Example component

Open SDK by IAR, the FATFS example is under **utilites->example->example_fatfs.c**



Fatfs relevant source is under **utilites->fatfs**. There are three sub-folders in FatFs source tree, **disk_if** contains disk interface that called by file system layer, **fatfs_ext** provides FatFs extension functions while **r0.10c** include the standard source file of FatFs provided by official.

SD low level driver is released as lib_sdcard.a in director of **component\soc\realtek\8195a\misc\bsp\lib\common\IAR\lib_sdcard.a** and added to IAR project under folder **lib.**

2.4 Behavior description

Simply, test message will be wrote to the connected SD memory card and then read back to verify SD card write and read function.

Both the write stage and read stage will show in the console with the written and read length of this test message as well as its content.

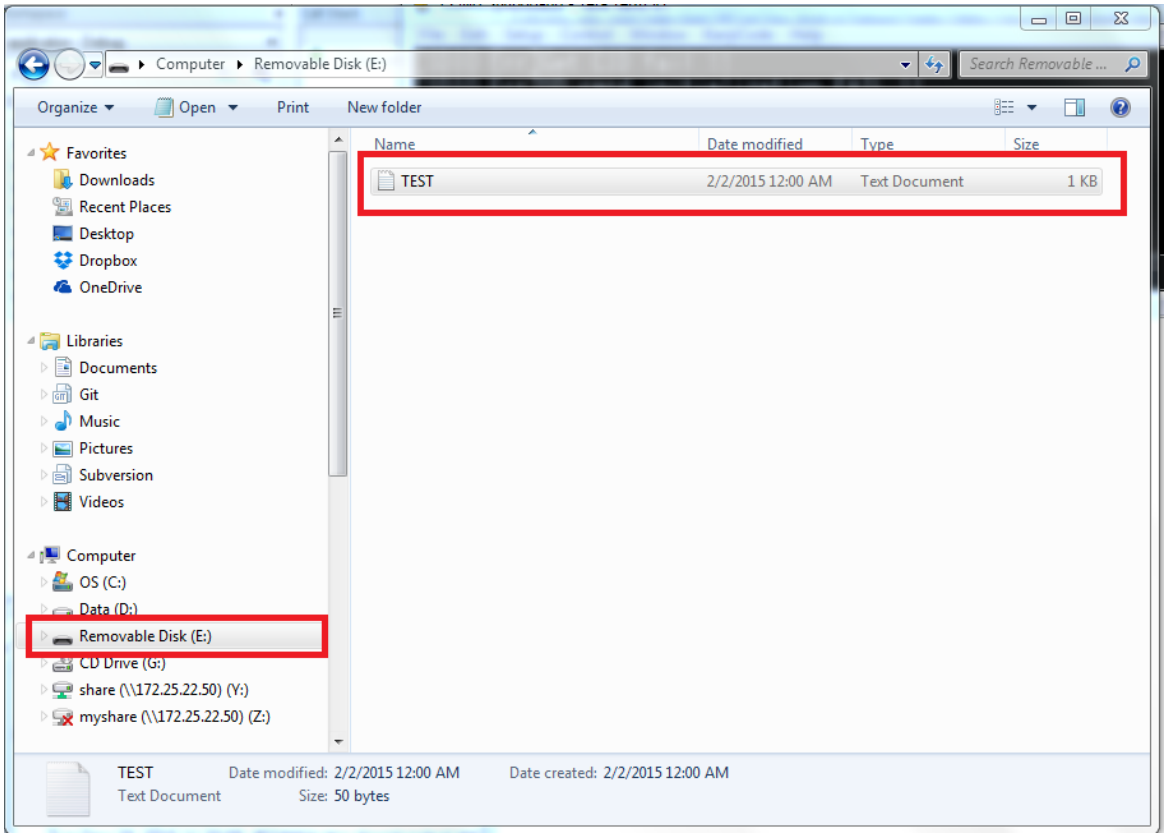
The test message "*Ameba memory card test information: Hello world.*" will be stored in SD card with a file name "TEST.TXT". User can plug in the SD card to Windows/Linux machine to read the file.

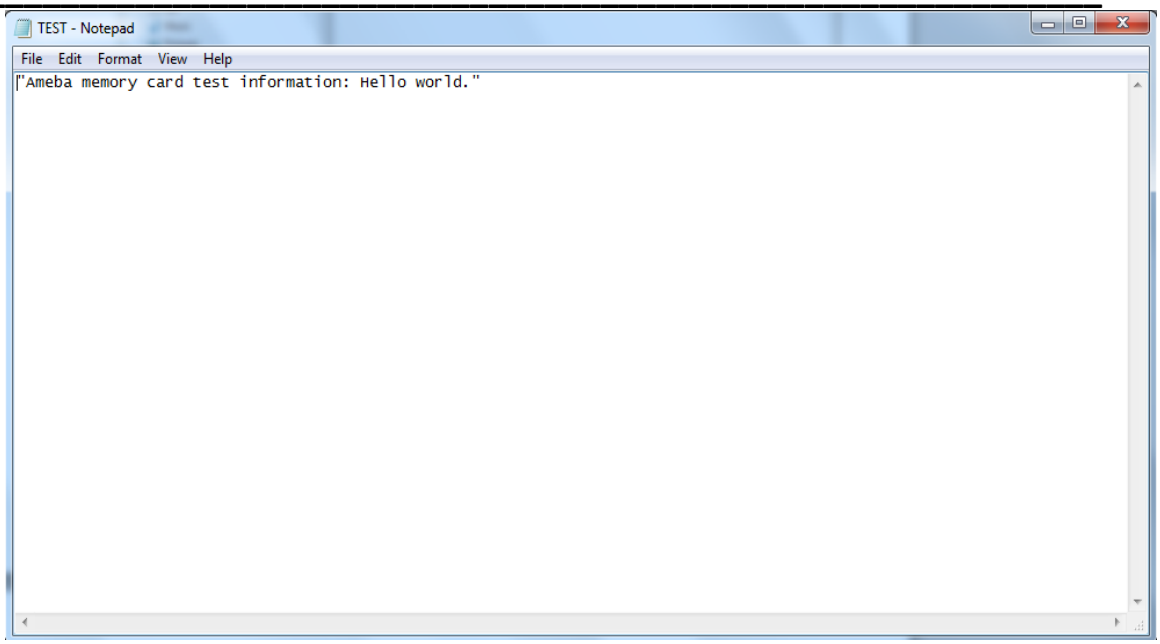
```
Init Disk driver.
Register disk driver to Fatfs.
FatFS Write/Read test begin......

Test file name:TEST.TXT

Write 50 bytes.
Write content:
"Ameba memory card test information: Hello world."

Read 50 bytes.
Read content:
"Ameba memory card test information: Hello world."
```





2.5 Trouble Shooting

If the FATFS example do not work well on Ameba platform, please check the following.

1. SD card conenctor compitable problem.
2. SD card compitable problem. Please refer to chapter 6 for SD compitability test result.
3. Please make sure the Ameba development board supports SD crad, only RTL8195AM support SD Host function.

2.6 File system

A file system is used to control how data is stored and retrieved. Currently, Ameba supports the most common embedded filesystem-FatFs.

FatFs is a generic FAT file system module for small embedded systems. The FatFs module is written in compliance with ANSI C (C89) and completely separated from the disk I/O layer and it supports FAT16 and FAT32 filesystem.

FatFs application interface:

- File Access
  - f_open - Open/Create a file
  - f_close - Close an open file
  - f_read - Read data
  - f_write - Write data
  - f_lseek - Move read/write pointer, Expand size

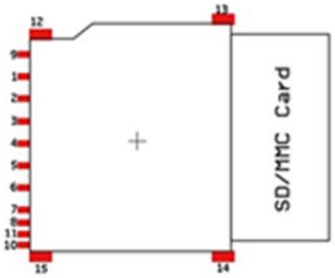
- f_truncate - Truncate size
- f_sync - Flush cached data
- f_forward - Forward data to the stream
- f_gets - Read a string
- f_putc - Write a character
- f_puts - Write a string
- f_printf - Write a formatted string

- Directory Access
  - f_opendir - Open a directory
  - f_closedir - Close an open directory
  - f_readdir - Read an item
- File/Directory Management
  - f_stat - Check existance of a file or sub-directory
  - f_unlink - Remove a file or sub-directory
  - f_rename - Rename or move a file or sub-directory
  - f_chmod - Change attribute of a file or sub-directory
  - f_utime - Change timestamp of a file or sub-directory
  - f_mkdir - Create a sub-directory
  - f_chdir - Change current directory
  - f_chdrive - Change current drive
  - f_getcwd - Retrieve the current directory and drive
- Volume Management
  - f_mount - Register/Unregister a work area of a volume
  - f_mkfs - Create an FAT volume on the logical drive
  - f_fdisk - Create logical drives on the physical drive
  - f_getfree - Get total size and free size on the volume
  - f_getlabel - Get volume label
  - f_setlabel - Set volume label

More details about Fatfs, please click http://elm-chan.org/fsw/ff/00index_e.html.

2.7 Micro SD card pin connection

For cases makers use micro SD card without SD sniffer, CD and WP pins not supported, what need to do is just connect GPIOA_6(D0) to ground, and leave GPIOA_7(D1) unconnected.



SD/MMC CARD CONNECTOR

DAT3----GPIOA_1 (D16)
CMD----GPIOA_2 (D7)
VSS1----GND
CLK ----GPIOA_3 (D6)
VSS2----GND

DAT0----GPIOA_4 (D5)
DAT1----GPIOA_5 (D2)
DAT2----GPIOA_0 (D17)
GND ---GPIOA_6 (D0)
Unconnected ---GPIOA_7 (D1)

3 USB MSC

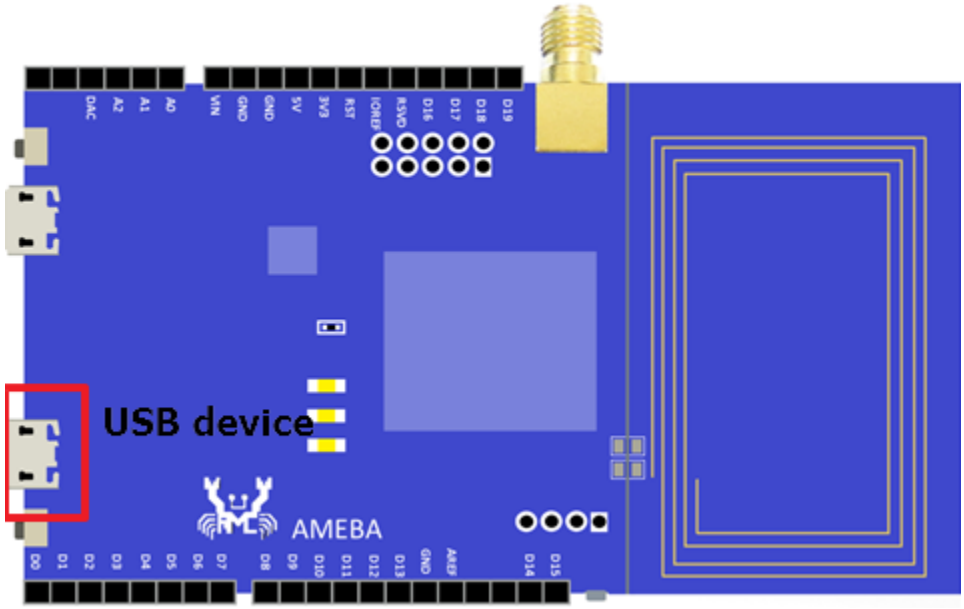
With USB device v2.0 interface, Ameba can be designed to a USB mass storage device class (USB MSC).

In this application, Ameba boots up as USB mass storage and uses SD card as its physical memory medium, USB host end (eg,. windows machine) can recognize Ameba and write data to and read data from SD card via USB interface.

3.1 Hardware setup

In order to run this application successfully, the hardware setup should be confirm before moving on.

1. Connect Ameba to USB host end with a micro USB cable. Ameba DEV board has designed a micro USB connector on board.



2. Connect SD card to Ameba DEV board refer to chapter 2.1.

3.2 How to run the example

Step 1, config USB MSC example in platform_opts.h

```
#define CONFIG_EXAMPLE_USB_MASS_STORAGE        1
```

Step 2, rebuild the project and download firmware to DEV board.

3.3 Example component

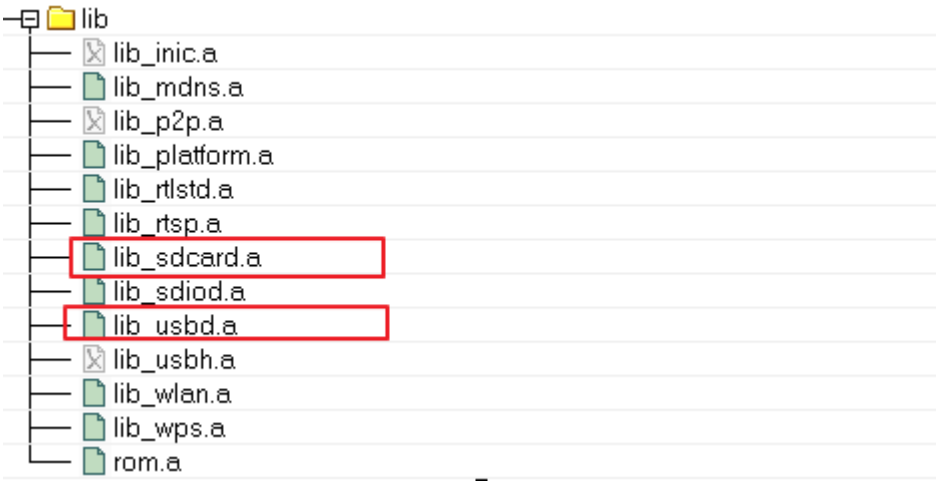
Open SDK by IAR, the USB MSC example is under **utilites->example->example_usb_msc.c**

This application is powered by two libs, one is **lib_usbd.a** which contains USB device core driver and USB MSC driver, the other is **lib_sdcard.a** providing SD card accessing functions.

Lib_usb.a: **component\soc\realtek\8195a\misc\bsp\lib\common\IAR\lib_usbd.a**

Lib_sdcard.a: **component\soc\realtek\8195a\misc\bsp\lib\common\IAR\lib_sdcard.a**

Both of the two libs has been added to IAR project under folder **lib** by default**.**



3.4 Behavior description

On the serial console, USB MSC loading log will be printed as the picture, make sure there is no error reported. After the MSC driver is successfully loaded, USB host end will recognize Ameba as a USB mass storage device. Now user can operate the USB mass storage device by adding or deleting file on the memory medium.

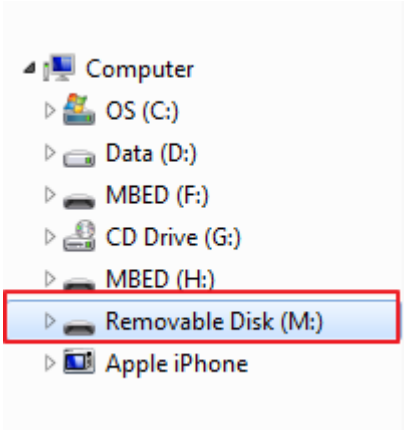
```
Boot from Flash:YES
===== Enter Image 1 ====
SDR Controller Init

load NEW fw 0
Flash Image2:Addr 0xb000, Len 271200, Load to SRAM 0x10006000
Image3 length: 0x14ac4, Image3 Addr: 0x30000000
Img2 Sign: RTKWin, InfaStart @ 0x10006079
===== Enter Image 2 ====
#interface 1 is initialized
interface 0 is initialized

Initializing WIFI ...
Start LOG SERVICE MODE

#
WIFI initialized

init_thread(52), Available heap 0x9c58
USB INIT DONE, Available heap [0xafe0]
USB MSC driver load done, Available heap [0x17a0]
```



4 Flash Read/Write

Ameba is provided with flash memory component on board and the data can be stored even when the power is off.

In this application, the example is given where the data is read from/written to flash memory through the embedded file system FATFS API.

4.1 Hardware setup

Follow the below instruction to setup Ameba to run the example.

1. Connect board to PC with micro-USB cable
2. Run the example as shown in chapter 4.2

4.2 How to run the example

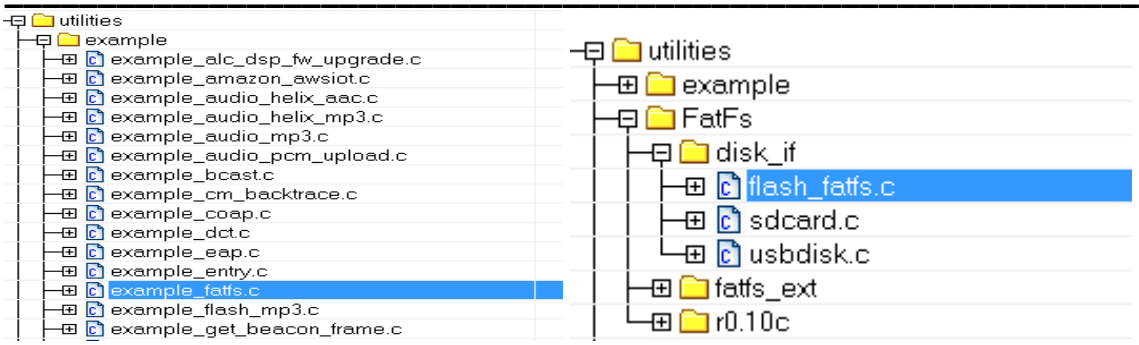
Step 1: Config FATFS example with Flash memory in platform_opts.h as follows.

```
/* For FATFS example*/
#define CONFIG_EXAMPLE_FATFS            1
#if CONFIG_EXAMPLE_FATFS
#define CONFIG_FATFS_EN 1
#if CONFIG_FATFS_EN
// fatfs version
#define FATFS_R_10C
// fatfs disk interface
#define FATFS_DISK_USB  0
#define FATFS_DISK_SD   0
#define FATFS_DISK_FLASH  1
#endif
#endif
```

Step 2: Rebuild the project and download firmware to DEV board.

4.3 Example component

Open SDK by IAR, the FATFS example is under **utilites->example->example_fatfs.c**



Fatfs relevant source is under **utilites->fatfs**.

There are three sub-folders in FatFs source tree,

disk_if contains disk interface called by file system layer,

fatfs_ext provides FatFs extension functions and

r0.10c include the standard source file of FatFs.

4.4 Behavior description

On the serial console, the given text message "*Ameba memory card test information: Hello world*." will be written to the Ameba flash memory with the given file name "TEST.TXT" in the example.

Through the console, the user can know the number of bytes written to and read from flash memory.

```
# Init Disk driver.
Register disk driver to Fatfs.
FatFS Write/Read test begin......

Test file name:TEST.TXT

Write 50 bytes.
Write content:
"Ameba memory card test information: Hello world."

Read 50 bytes.
Read content:
"Ameba memory card test information: Hello world."
```

4.5 File system

A file system is used to control how data is stored and retrieved. Currently, Ameba supports the most common embedded filesystem-FatFs.

Refer chapter 2.6 for more details.

5 File Management

The directory access FatFs API's (refer chapter 2.6) are used for easier file management in storage devices.

All files inside SD card and flash memory can be listed and showed in serial console along with the path of the file as shown below while executing the example.

```
# Init Disk driver.
Register disk driver to Fatfs.
FatFS Write/Read test begin......

Test file name:TEST.TXT

Write 50 bytes.
Write content:
"Ameba memory card test information: Hello world."

Read 50 bytes.
Read content:
"Ameba memory card test information: Hello world."

WIFI initialized

List all files), Available heap 0x8ff8
      AudioFlash.mp3              0://AudioFlash.mp3
            TEST.TXT              0://TEST.TXT
```

6 SD card compatibility test

The following list is SD card that has passed compatibility test. The SD card that is not listed in the following list is not guaranteed.

Vendor	Model	Test result
SanDisk	SanDisk Extreme microSD U3 16GB	Pass
	SanDisk Extreme Pro microSDHC UHS-I 16GB	pass
	SanDisk Ultra microSDXC UHS-I Class10 64GB	pass
PNY	PNY microSDHC UHS-1 Class10 16GB	Pass
	PNY MicroSDHC UHS-1 Class10 32GB	pass
Apacer 宇瞻	Apacer MicroSDHC UHS-I Class10 16GB	Pass
	Apacer MicroSDHC UHS-I Class10 32GB	pass
廣穎	Silicon Power Micro SDHC Class4 16GB	Pass
	Superior MicroSDHC UHS-I Class10 32GB	pass
Samsung	Ultra-fast microSDHC 16GB	Pass
	Ultra-fast microSDHC UHS-1 Class10 32GB	Pass
	microSDXC Class10 UHS-I EVO 128GB	Pass
TOPMORE	microSDHC UHS-1 Class10 32GB	Pass
	microSDHC Class6 16GB	Pass
KINGMAX	MicroSDHC PRO UHS-I Class10 16GB 80MB/s	Pass
	MicroSDHC Class10 UHS-1 PRO 32GB	Pass
創見	microSDHC Class10 32GB	Pass
	microSDXC Class10 UHS-I 300x 16GB	Pass
威剛	Premier microSDHC U1 Class10 32GB	Pass
	Premier microSDHC U1	Pass

	Class10 16GB	
TOSHIBA	EXCERIA 16GB Micro SDHC Card R48MB	Pass
	EXCERIA 32GB UHS-I U3 MICRO SDHC	Pass