USB Flash Disk and SD Card File Management Control Chip CH376

Datasheet
Version: 1A
http://wch.cn

1. Overview

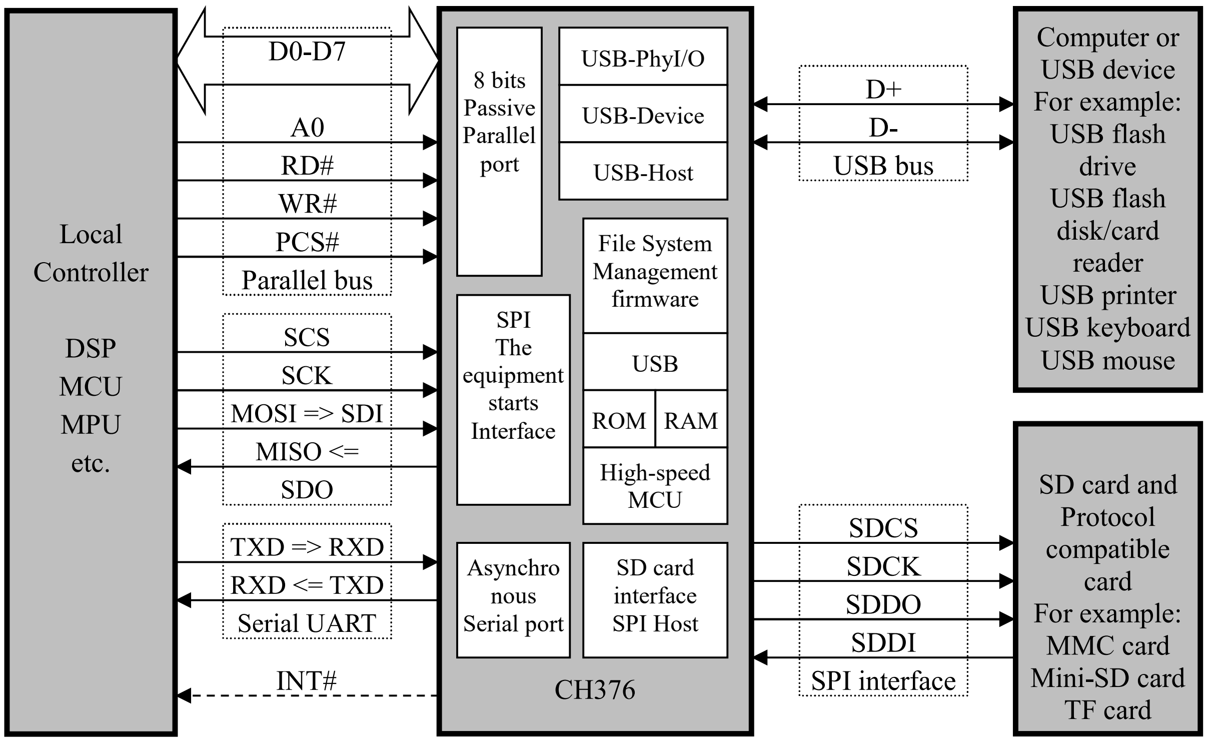
CH376 is a file management control chip, which is used for MCU system to read and write files in USB flash disk or SD card.

CH376 supports USB device mode and USB host mode, has the basic firmware of the USB communication protocol, the firmware for processing the special communication protocol of Mass-Storage device, the SD card communication interface firmware and the management firmware of FAT16 and FAT32 and FAT12 file systems, and supports the common USB storage devices (including USB flash disk/USB hard disk, USB flash drive/USB card reader) and SD card (including standard capacity SD card and high capacity HC-SD card as well as protocol compatible MMC card and TF card).

CH376 supports three communication interfaces: 8-bit parallel port, SPI interface or asynchronous serial interface. Controllers such as DSP/MCU/MPU can control CH376 through any of the above communication interfaces, access files in USB flash disk or SD card or communicate with the computer.

The USB device mode of CH376 is fully compatible with CH372, and the USB host mode of CH376 is basically compatible with CH375.

The figure below is an application block diagram of CH376.

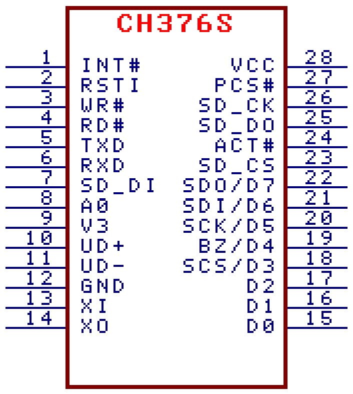

Local Controller
DSP
MCU
MPU
etc.
D0-D7
A0
RD#
WR#
PCS#
Parallel bus
SCS
SCK
MOSI => SDI
MISO <=
SDO
TXD => RXD
RXD <= TXD
Serial UART
INT#
8 bits Passive Parallel port
USB-PhyI/O
USB-Device
USB-Host
File System Management firmware
SPI The equipment starts Interface
USB
ROM
RAM
High-speed MCU
Asynchronous Serial port
SD card interface SPI Host
CH376
D+
D-
USB bus
SDCS
SDCK
SDDO
SDDI
SPI interface
Computer or USB device For example: USB flash drive USB flash disk/card reader USB printer USB keyboard USB mouse
SD card and Protocol compatible card For example: MMC card Mini-SD card TF card

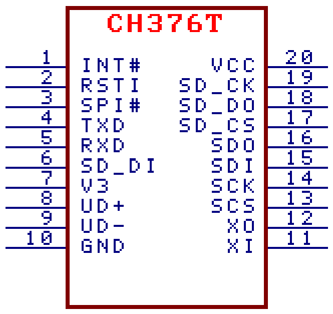
2. Features

- Support 1.5Mbps low-speed and 12Mbps full-speed USB communication, compatible with USB V2.0; only crystals and capacitors are required for peripheral components.

- Support USB-HOST interface and USB-Device interface, and support dynamic mode switching between HOST and DEVICE.
- Support USB device control transmission, bulk transmission and interrupt transmission.
- Automatically detecting the connection and disconnection of USB device, and providing event notifications for device connection and disconnection.
- Provide 6MHz SPI host interface, and support SD card as well as MMC card and TF card compatible with its protocol.
- Built-in USB control transmission protocol processor simplifies common control transmission.
- The built-in firmware processes special communication protocols for mass storage devices, and supports BULk-Only transport protocol and USB storage devices (including USB flash disk/USB hard disk/USB flash drive/USB card reader) with SCSI, UFI, RBC or equivalent command sets.
- The management firmware with FAT16, FAT32 and FAT12 file systems supports 32GB USB flash disk and SD card.
- Provide file management functions: open, create or delete files, enumerate and search files, create subdirectories, and support long filenames.
- Provide file read-write function: read and write files in the multi-level subdirectories in bytes or sectors.
- Provide the disk management function: initialize the disk, inquire the physical capacity, inquire the free space, read and write physical sectors.
- Provide a 2MB 8-bit passive parallel interface and support the parallel data bus connected to MCU.
- Provide 2MB/24MHz SPI device interface, and support SPI serial bus connected to MCU.
- Provide the asynchronous serial interface with the maximum speed of 3Mbps, support the serial port connected to MCU, and support the dynamic adjustment of communication baud rate.
- Support supply voltage of 5V, 3.3V and 3V as well as low power mode.
- The USB device mode is fully compatible with CH372; the USB host mode is basically compatible with CH375 chip.
- Provide SOP-28 and SSOP20 lead-free package, be compatible with RoHS, provide SOP28 to DIP28 conversion board, and SOP28 package pin is basically compatible with CH375.

3. Package





Package	Width of Plastic		Pitch of Pin		Instruction of Package	Ordering Information
SOP-28	7.62mm	300mil	1.27mm	50mil	Standard 28-pin patch	CH376S
SSOP-20	5.30mm	209mil	0.65mm	25mil	Ultra-small 20-pin patch	CH376T

4. Pins

CH376S Pin No.	CH376T Pin No.	Pin Name	Pin Type	Description
28	20	VCC	Power	Positive power input, an external 0.1uF power decoupling capacitor is required.
12	10	GND	Power	Common ground, shall be connected to the ground wire of the USB bus
9	7	V3	Power	Connected to the VCC input external power at the supply voltage of 3.3V Connected to an external 0.01uF decoupling capacitor at 5V supply voltage
13	11	XI	Input	Input terminal of the crystal oscillator, required to be externally connected to a 12MHz crystal
14	12	XO	Output	Inverted output terminal of crystal oscillator, required to be externally connected to a 12MHz crystal
10	8	UD+	USB signal	USB bus D+data line
11	9	UD-	USB signal	USB bus D - data cable
23	17	SD_CS	Open-drain output	Chip selection output of SD card SPI interface, active at low level, built-in pull-up resistor
26	19	SD_CK	Output	Serial clock output of SD card SPI interface
7	6	SD_DI	Input	Serial data input of SD card SPI interface, built-in pull-up resistor
25	18	SD_DO	Output	Serial data output of SD card SPI interface
25	18	RST	Output	Power-on rest output and external reset output before entering SD card mode, active high
22～15	None	D7～D0	Bi-directional three-state	8-bit bidirectional data bus on parallel port, built-in pull-up resistor
18	13	SCS	Input	Chip selection input of SPI interface, active low, built-in pull-up resistor
20	14	SCK	Input	Serial clock input of SPI interface, built-in pull-up resistor
21	15	SDI	Input	Serial data input of SPI interface, built-in pull-up resistor
22	16	SDO	Three-status output	Serial data output of SPI interface
19	None	BZ	Output	Busy status output of SPI interface, active at high level
8	None	A0	Input	Address input of parallel port, distinctive command port and data port, built-in pull-up resistor, Write commands or read statuses when A0=1; write and read data when A0=0
27	None	PCS#	Input	Chip selection control input of parallel port, active at low level, built-in pull-up resistor
4	None	RD#	Input	Read strobe input of parallel port, active at low level, built-in pull-up resistor
3	None	WR#	Input	Write strobe input of parallel port, active at low level, built-in pull-up resistor

None	3	SPI#	Input	Interface configuration input during chip internal reset, built-in pull-up resistor
5	4	TXD	Input Output	Interface configuration input during chip internal reset, built-in pull-up resistor, Serial data output of asynchronous serial interface after chip reset
6	5	RXD	Input	Serial data input of asynchronous serial interface, built-in pull-up resistor
1	1	INT#	Output	Interrupt request output, active at low level, built-in pull-up resistor
24	None	ACT#	Open-drain output	Status output, active at low level, built-in pull-up resistor. USB device is being connected to the status output in the USB host mode; SD card SPI communication success status output in the SD card host mode; USB device configuration completes the status output in the USB device mode with built-in firmware
2	2	RSTI	Input	External reset input, active at low level, built-in pull-down resistor

5. Commands

For the data in this datasheet, the suffix B is a binary number, and the suffix H is a hexadecimal number. Otherwise, it is a decimal number.

Double-word data (32 bits in total) with low bytes in the front (Little-Endian) refers to: the lowest byte (bits 7 to 0), followed by the lower byte (bits 15 to 8), followed by the higher byte (bits 23 to 16), followed by the highest byte (bits 31 to 24).

Data stream is a data block consisting of several consecutive bytes with a total length of at least 0 and at most 255.

The number in the brackets for the input data and output data in the following table is the number of bytes of the parameter. If there are no brackets, it will be 1 byte by default.

MCU referred to in this datasheet is basically applicable to DSP or MCU/MPU/SCM, etc.

USB flash disk referred to in this datasheet includes USB flash disk, USB external hard disk, USB flash drive, USB card reader, etc.

SD card referred to in this datasheet includes SD card, MMC card, HC-SD card (high-capacity SD card), TF card, etc.

This datasheet mainly provides commonly used file management control commands for USB flash disk and SD card. Refer to datasheet CH376DS2 (II).PDF for some less commonly used auxiliary commands and commands for performing USB basic transactions and control transmission.

CH376 contains all the functions of CH372 chip. This datasheet does not provide the description of CH376 in USB device mode. For related information, refer to datasheet CH372DS1.PDF.

Code	Command name CMD_	Input data	Output data	Command purpose
01H	GET_IC_VER		Version	Get the chip and firmware versions

02H	SET_BAUDRATE	Frequency division coefficient	(Wait for 1mS) Operation status	Set serial communication baud rate
		Frequency division constant		
03H	ENTER_SLEEP			Enter the low-power sleep suspend state
05H	RESET_ALL		(Wait for 35mS)	Execute hardware reset
06H	CHECK_EXIST	Any data	Bitwise NOT	Test the communication interface and operation status
0BH	SET_SDO_INT	Data 16H		Set the interrupt mode for SDO pin of SPI
		Interrupt mode		
0CH	GET_FILE_SIZE	Data 68H	File length (4)	Get the current file length
15H	SET_USB_MODE	Mode code	(Wait for 10uS) Operation status	Set USB working mode
22H	GET_STATUS		Interrupt status	Get the interrupt status and cancel the interrupt request
27H	RD_USB_DATA0		Data Length	Read the data block from the endpoint buffer of the current USB interrupt or from the receive buffer of the host endpoint
			Data stream (n)	
2CH	WR_HOST_DATA	Data Length		Write the data block to the transmit buffer of the USB host endpoint
		Data stream (n)		
2DH	WR_REQ_DATA		Data Length	Write the requested data block to the internal specified buffer
		Data stream (n)		
2EH	WR_OFS_DATA	Offset address		Write the data block to the specified offset address of the internal buffer
		Data Length		
		Data stream (n)		
2FH	SET_FILE_NAME	Character string (n)		Set the filename of the file to be operated
30H	DISK_CONNECT		Generate interrupt	Check whether the disk is connected
31H	DISK_MOUNT		Generate interrupt	Initialize the disk and test whether the disk is ready
32H	FILE_OPEN		Generate interrupt	Open files or directories, enumerate files and directories
33H	FILE_ENUM_GO		Generate interrupt	Continue enumerating files and directories
34H	FILE_CREATE		Generate interrupt	New File
35H	FILE_ERASE		Generate interrupt	Delete File
36H	FILE_CLOSE	Whether update is allowed	Generate interrupt	Close the currently opened file or directory
37H	DIR_INFO_READ	Directory index number	Generate interrupt	Read the directory information for the file
38H	DIR_INFO_SAVE		Generate interrupt	Save the directory information for the file

39H	BYTE_LOCATE	Number of offset bytes (4)	Generate interrupt	Move the current file pointer in bytes
3AH	BYTE_READ	Number of bytes requested (2)	Generate interrupt	Read data blocks from the current location in bytes
3BH	BYTE_RD_GO		Generate interrupt	Continue byte read
3CH	BYTE_WRITE	Number of bytes requested (2)	Generate interrupt	Write data blocks to the current location in bytes
3DH	BYTE_WR_GO		Generate interrupt	Continue byte write
3EH	DISK_CAPACITY		Generate interrupt	Inquire the physical capacity of the disk
3FH	DISK_QUERY		Generate interrupt	Inquire the space information of the disk
40H	DIR_CREATE		Generate interrupt	Create a new directory and open it or open an existing directory
4AH	SEC_LOCATE	Number of offset sectors (4)	Generate interrupt	Move the current file pointer in sectors
4BH	SEC_READ	Number of requested sectors	Generate interrupt	Read data blocks from the current location in sectors
4CH	SEC_WRITE	Number of requested sectors	Generate interrupt	Write data blocks to the current location in sectors
50H	DISK_BOC_CMD		Generate interrupt	Execute BO transport protocol commands on USB memory
54H	DISK_READ	LBA sector address (4)	Generate interrupt	Read physical sectors from USB memory
		Number of sectors		
55H	DISK_RD_GO		Generate interrupt	Continue the physical sector read operation of the USB memory
56H	DISK_WRITE	LBA sector address (4)	Generate interrupt	Write physical sectors to USB memory
		Number of sectors		
57H	DISK_WR_GO		Generate interrupt	Continue the physical sector write operation of the USB memory

If the output data of the command is the operation status, refer to the following table.

Status code	Status name	Status description
51H	CMD_RET_SUCCESS	Operated Successfully
5FH	CMD_RET_ABORT	Operation failure

It usually takes time to execute the commands marked as “generate interrupt”. CH376 requests an interrupt from MCU after the command is executed. MCU can read the interrupt status as the operation status of the command. If the interrupt status is USB_INT_SUCCESS, the operation will be successful. Some commands have return data (refer to CH376_CMD_DATA structure in the file CH376INC.H), and the returned data can be read through the command CMD_RD_USB_DATA0.

5.1. CMD_GET_IC_VER

This command is used to get the chip and firmware versions. One byte of data returned is the version number, the bit 7 is 0, the bit 6 is 1, and the bits 5-0 are the version number. If the returned value is 41H, remove bits 7 and 6, and the version number will be 01H.

5.2. CMD_SET_BAUDRATE

This command is used to set the baud rate of CH376 for serial communication. When CH376 works in serial communication mode, the default communication baud rate is set by the level combination of BZ/D4, SCK/D5 and SDI/D6 pins (refer to Section 6.4 of this datasheet) after reset. When these pins are suspended, the baud rate is 9600bps by default. If MCU supports high communication speed, the serial communication baud rate can be dynamically regulated through this command. The command requires the input of two data, namely, baud rate frequency division coefficient and frequency division constant. The following table shows the relationship with the baud rate.

Frequency division coefficient	Frequency division constant	Serial communication baud rate (bps)	Error
02H	B2H	9600	0.16%
02H	D9H	19200	0.16%
03H	98H	57600	0.16%
03H	CCH	115200	0.16%
03H	F3H	460800	0.16%
07H	F3H	921600	0.16%
03H	C4H	100000	0%
03H	FAH	1000000	0%
03H	FEH	3000000	0%
02H	Constant	Calculation formula: 750000/(256- constant)	
03H	Constant	Calculation formula: 6000000/(256- constant)	

Usually, the serial port communication baud rate is set within 1mS. After completion, CH376 outputs the operation state at the newly set communication baud rate. Therefore, MCU shall adjust its own communication baud rate in time after sending the command.

5.3. CMD_ENTER_SLEEP

This command enables CH376 in a low-power sleep suspended state. After entering the low power state, the clock of CH376 stops vibrating, thus saving power. The low power state is not quitted until one of the following two situations is detected: first, the signal is detected on the USB bus (such as the USB host starting transmission or USB device plugging event). Second, MCU writes new commands to CH376 (commands without input data, such as CMD_GET_IC_VER or CMD_ABORT_NAK). For SPI interface communication interface mode, active SCS chip selection will also cause CH376 to exit the low-power state, so MCU shall immediately disable the SCS chip selection after sending the command CMD_ENTER_SLEEP.

Typically, it takes several milliseconds for CH376 to exit the low-power state and return to normal operation. When fully restored to normal operation, CH376 will generate the event interrupt USB_INT_WAKE_UP.

5.4. CMD_RESET_ALL

This command enables CH376 to perform a hardware reset. Typically, hardware reset is completed within 35mS. In the parallel communication mode, hardware reset is generally completed within 1mS.

5.5. CMD_CHECK_EXIST

This command is used to test the communication interface and working state to check whether CH376 is working properly. This command needs to input 1 data, which can be any data. If CH376 is working properly, the output data of CH376 will be the bitwise reverse of the input data. For example, if the input data is 57H, the output data will be A8H. In addition, CH376 with parallel communication mode normally reads the data 00H from its parallel port before receiving no command after its reset.

5.6. CMD_SET_SDO_INT

This command is used to set the interrupt mode for SDO pin of SPI interface. The command requires to first input one data 16H, and then input one new interrupt mode. There are two interrupt modes: 10H disable SDO pin is used for interrupt output, and the three-state output is disabled when the SCS chip selection is invalid, so as to share the SPI bus of MCU with other devices; 90H set SDO pin is always in the output state. When the SCS chip selection is invalid, it serves as the interrupt request output, which is equivalent to the INT# pin for MCU to query the interrupt request status.

5.7. CMD_GET_FILE_SIZE

This command is used to get the length of the current file, namely the number of bytes. The command requires to input one data 68H and output the length of the file currently being opened, which is a double-word data (32 bits) expressed by 4 bytes with low bytes in front.

To set the new file length, refer to the command CMD_WRITE_VAR32 in Datasheet (II) to set the variable VAR_FILE_SIZE.

5.8. CMD_SET_FILE_SIZE

This command is used to set the length of the current file, namely the number of bytes. The command requires to input one data 68H and then a new file length, which is a double-word data (32 bits) expressed by 4 bytes with low bytes in front.

This command is only used to modify the file length variable in CH376 memory. Only after the commands like CMD_FILE_CLOSE are executed, the file length in the USB storage device or SD card can be actually updated.

5.9. CMD_SET_USB_MODE

This command is used to set USB working mode. This command needs to input 1 data, which is a mode code:

Switch to the disabled USB device mode (default mode after power-on or reset) when the mode code is 00H;
Switch to the enabled USB device mode and external firmware mode (serial connection mode is not supported) when the mode code is 01H;
Switch to the enabled USB device mode and built-in firmware mode when the mode code is 02H;
Switch to the SD card host mode to manage and access files in the SD card when the mode code is 03H;
Switch to the disabled USB host mode when the mode code is 04H;
Switch to the enabled USB host mode when the mode code is 05H, not generating SOF package;
Switch to the enabled USB host mode when the mode code is 06H, automatically generating SOF package;
Switch to the enabled USB host mode when the mode code is 07H, resetting USB bus;

Please refer to CH372 Datasheet for USB device mode. The USB device mode of CH376 is fully compatible with CH372 chip.

In USB host mode, "Not enabled" means that whether the USB device is connected is not automatically detected, so the external MCU shall detect; "Enabled" means that whether the USB device is connected is automatically detected. When the USB device is connected or disconnected, an interrupt will be generated to notify the external MCU. After switching to the mode code 06H, CH376 will automatically periodically generate a USB frame cycle SOF packet to be sent to a connected USB device. The mode code 07H is usually used to provide a USB bus reset state to a connected USB device, and the USB bus reset will not end until it is switched to other working mode. It is recommended to use mode 5 when there is no USB device. After the USB device is plugged, enter mode 7 first and then switch to mode 6.

Usually, the USB working mode is set within 10uS, and the operation status is output after completion.

5.10. CMD_GET_STATUS

This command is used to get the interrupt status of CH376 and notify CH376 to cancel the interrupt request. After CH376 requests an interrupt from MCU, MCU gets the interrupt status through the command, analyzes the interrupt cause and processes.

Interrupt status byte	Classification of interrupt status
00H～0FH	Please refer to CH372 Datasheet for the interrupt status in USB device mode.
10H～1FH	Operation interrupt status in SD card or USB host mode
20H～3FH	Communication failure status in USB host mode, used to analyze the cause of operation failure
40H～4FH	File system warning codes in SD card or USB host file mode
80H～BFH	File system error codes in SD card or USB host file mode

The following table shows operation interrupt statuses in SD card or USB host mode.

Status byte	Status name	Analysis and description of interrupt status
14H	USB_INT_SUCCESS	SD card or USB transaction or transmission operation or file operation is successful
15H	USB_INT_CONNECT	A USB device connection event is detected
16H	USB_INT_DISCONNECT	A USB device disconnection event is detected
17H	USB_INT_BUF_OVER	The transmitted data is wrong or the buffer overflows due to too many data
18H	USB_INT_USB_READY	USB device has been initialized (USB address has been assigned)
1DH	USB_INT_DISK_READ	Storage device performs read operation, and requests data read
1EH	USB_INT_DISK_WRITE	Storage device performs write operation, and requests data write
1FH	USB_INT_DISK_ERR	Storage device operation failed

The following table shows communication failure statuses in USB host mode, which are generally used to analyze the cause of operation failure.

Interrupt status byte	Name	Analysis and description of interrupt status
Bits 7-6	(Reserved bit)	Always 00
Bit 5	(Flag bit)	Always 1, indicating that this status is the operation failure status
Bit 4	IN transaction Sync flag	For IN transactions, if the bit is 0, data packets currently received are out of sync and the data may be invalid
Bits 3-0	Returned value	1010 = Device returns NAK

	of USB device that causes operation failure	1110 = Device returns STALL
		XX00 = Device returns timeout. The device did not return
		Other value is PID returned by the device

The following table shows file system warning codes and error codes in SD card or USB host file mode.

Status byte	Status name	Analysis and description of interrupt status
41H	ERR_OPEN_DIR	The directory of the specified path is opened
42H	ERR_MISS_FILE	The file of the specified path is not found. It may be a file name error
43H	ERR_FOUND_NAME	The matched filename is searched, or a directory is required to be opened but a file is opened actually
82H	ERR_DISK_DISCON	The disk is not connected. It may have been disconnected
84H	ERR_LARGE_SECTOR	The sector of the disk is too large, and only 512 bytes is supported per sector
92H	ERR_TYPE_ERROR	The disk partition type is not supported. The disk shall be repartitioned by the disk management tool
A1H	ERR_BPB_ERROR	The disk is not formatted, or the parameter is wrong, The disk shall be reformatted by WINDOWS using default parameters
B1H	ERR_DISK_FULL	The disk is full of files, there is too little free space or there is no free space
B2H	ERR_FDT_OVER	There are too many files in the directory, and there are no free directory entries. It is necessary to manage the disk, The number of files in the root directory of FAT12/FAT16 shall be less than 512
B4H	ERR_FILE_CLOSE	The file has been closed. If it is required to be used, reopen it

5.11. CMD_RD_USB_DATA0

This command is used to read the data block from the endpoint buffer of the current USB interrupt or from the receive buffer of the host endpoint. The output data firstly read is data block length, namely, the number of bytes of subsequent data streams. The effective value of the data block length is 0 to 255. For USB underlying transport, the data block length is 0 to 64. If the length is not 0, MCU must read the subsequent data one by one from CH376.

5.12. CMD_WR_HOST_DATA

This command is used to write data blocks to the send buffer of the USB host endpoint. The input data firstly written is data block length, namely, the number of bytes of subsequent data streams. The effective value of the data block length is 0 to 64. If the length is not 0, MCU must write the subsequent data one by one to CH376.

5.13. CMD_WR_REQ_DATA

This command is used to write the data block requested by CH376 to the internal specified buffer. The output data firstly read is data block length, namely, the number of bytes of subsequent data streams that CH376 requests MCU to write. The effective value of the data block length is 0 to 255. For USB underlying transport, the data block length is 0 to 64. If the length is not 0, MCU must write the subsequent data one by one to CH376.

5.14. CMD_WR_OFS_DATA

This command is used to write the data block to the specified offset address of the internal buffer. The input data firstly written is the offset address (the internal buffer start address plus the offset address gives the start address for writing the command block), and the input data written later is the data block length, namely, the number of bytes of subsequent streams. The effective value of the data block length is 0 to 32, and the sum of offset address and data block length shall not be more than 32. If the data block length is not 0, MCU must write the subsequent data one by one to CH376.

5.15. CMD_SET_FILE_NAME

This command is used to set the filename of the file or directory (folder) to be operated or the directory name (pathname). The input data is a string ending in 0, and the length must not exceed 14 characters, including the ending character 0. For files under multi-level subdirectories, the whole path can be decomposed into multiple subdirectory names and a file name to set the name several times and open step by step from the root directory. When the file operation is wrong, it is necessary to return to the root directory to reopen step by step.

Filenames (or directory names, or pathnames) are in the same format as short filenames on DOS systems, but do not require drive letter or colon. As root directory descriptors, the left slash "/" and the right slash "\" are equivalent, and the left slash "/" is recommended. All characters must be uppercase, numeric, or Chinese characters, and some special characters. The length of filename shall not be more than 11 characters. The length of primary filename shall not be more than 8 characters, the length of extension shall not be more than 3 characters. If there is an extension, it shall be the separated from the primary filename by the decimal point. Refer to EXAM11 to support long filenames.

When there is no character in the string (but there is an ending character 0, the same below), it indicates that the file system is initialized and any file is not opened;

When there is only one "/" or "\" (left slash or right slash) in the string, the root directory will be opened;

If the first character of the string is "/" or "\" and the following character is the filename, it will be the file under the root directory.

When the string is directly a filename, it indicates that the file is in the current directory.

For example, FILENAME.EXT file under the root directory can be set with the string “/FILENAME.EXT\0”. The entire string has a total of 14 characters including ending character, in which "\0" is 0 expressed in C language and used as the ending character of the string, "/" expresses the root directory, and "\\" (actually one "\" character) can also express the root directory in C language.

For example, the file \YEAR2004\MONTH05.NEW\DATE18\ADC.TXT with a long path under the three-level subdirectory, can be opened by the following steps:

① After setting the filename (directory name) with the string "/YEAR2004\0", open the first-level subdirectory with CMD_FILE_OPEN;

② After setting the filename (directory name) with the string “MONTH05.NEW\0”, open the second-level subdirectory with CMD_FILE_OPEN;

③ After setting the filename (directory name) with the string “DATE18\0”, open the third-level subdirectory with CMD_FILE_OPEN;

④ After setting the filename with the string “ADC.TXT\0”, open the final file with CMD_FILE_OPEN;

5.16. CMD_DISK_CONNECT

This command is used to check whether the disk is connected and does not support SD card. In the USB host mode, this command can inquire whether the disk is connected at any time, and CH376 requests an interrupt to MCU after the command is executed. If the operation status is USB_INT_SUCCESS, there will be a disk or USB device connected.

5.17. CMD_DISK_MOUNT

This command is used to initialize the disk and test whether the disk is ready. The newly connected USB storage device or SD card must be initialized through this command before file operation. Some USB storage devices may return the successful operation status USB_INT_SUCCESS after being initialized for several times. In addition, the command can also be used to test whether the disk is ready at any time during file operation.

If the interrupt status is USB_INT_SUCCESS when the command CMD_DISK_MOUNT is executed, the data can be gotten through the command CMD_RD_USB_DATA0. The data is usually 36 bytes, including the features of USB storage device and the identification information of manufacturer and product.

5.18. CMD_FILE_OPEN

This command is used to open files or directories (folders) and enumerate files and directories (folders).

Opening a file (or directory) is a necessary operation before reading or writing a file (or directory). Before opening the file command, set the filename of the file to be opened or enumerated through the command CMD_SET_FILE_NAME.

If the file is under the multi-level subdirectories, and the pathname is long, it can be opened from the root directory several times level by level. First open the first-level subdirectory, then the second-level subdirectory, until finally open the file. The first opening must start from the root directory, so the first character of the pathname must be a slash "/" or "\". Later, the first character must not be "/" or "\" when it is opened again followed by the previous level.

If the directory is opened successfully, the interrupt status ERR_OPEN_DIR will be returned. At this point, the file length is invalid, being 0FFFFFFFFH.

If the file is opened successfully, the interrupt status USB_INT_SUCCESS will be returned. At this point, the file length is valid.

If the specified file or directory (folder) is not found, the interrupt status ERR_MISS_FILE will be returned.

For example:

To open the file \TODAY1.TXT in the root directory, the steps are as follows:

① Set the filename with the string “/TODAY1.TXT\0” through the command CMD_SET_FILE_NAME;
② Open the file with the command CMD_FILE_OPEN.

To open the file \YEAR2004\MONTH05.NEW\DATE18\ADC.TXT under the three-level subdirectory, the steps are as follows:

① Set the subdirectory name with the string “/YEAR2004\0” through the command CMD_SET_FILE_NAME;
② Open the first-level subdirectory with the command CMD_FILE_OPEN. If the command CMD_GET_FILE_SIZE is executed after the directory is opened, the invalid file length 0FFFFFFFFH will be returned;
③ Set the subdirectory name with the string “MONTH05.NEW\0” through the command CMD_SET_FILE_NAME;
④ Open the second-level subdirectory with the command CMD_FILE_OPEN;
⑤ Set the subdirectory name with the string “DATE18\0” through the command

CMD_SET_FILE_NAME;

⑥ Open the third-level directory with the command CMD_FILE_OPEN.

⑦ Set the filename with the string "ADC.TXT\0" through the command CMD_SET_FILE_NAME;

⑧ Open the final file with the command CMD_FILE_OPEN. If the command CMD_GET_FILE_SIZE is executed after the directory is opened, the actual file length will be returned.

To initialize the file system without opening any files, the steps are as follows:

① Set the filename with the string "\0" through the command CMD_SET_FILE_NAME;

② Execute the command CMD_FILE_OPEN, and then the file system will be initialized (return directly if it has been already initialized).

To open the root directory (for example, when processing a long filename), the steps are as follows:

① Set the filename with the string "/\0" through the command CMD_SET_FILE_NAME;

② Execute the command CMD_FILE_OPEN, and then the root directory will be opened (it must be closed through CMD_FILE_CLOSE after use).

5.19. CMD_FILE_ENUM_GO

This command is used to continue enumerating files and directories (folders).

To search and inquire files, the steps are as follows:

① Replace all or part of the filename characters to be inquired with the wildcard character *. No characters are allowed behind the wildcard character *. Set the string containing the wildcard character * to the filename through the command CMD_SET_FILE_NAME. For example, the string "/*\0" indicates that all files or directories under the root directory will be enumerated; the string "USB*\0" indicates that all files or directories beginning with three characters "USB" in the current directory will be enumerated. Filenames (or directory names) meeting requirements include "USB.TXT", "USB1234", "USB", "USBC.H", etc., but exclude "XUSB", "USB", "U2SB", "MY.USB", etc.;

② Start enumerating files and directories through the command CMD_FILE_OPEN.

③ CH376 compares each filename. Every time a file meeting the requirements is found, an interrupt will be generated to MCU. The interrupt status is USB_INT_DISK_READ to request MCU to read data from CH376;

④ MCU reads the data through the command CMD_RD_USB_DATA0, analyzes and processes the data immediately or saves the data first. The data is FAT file directory information (refer to the definition of FAT_DIR_INFO structure in the file CH376INC.H);

⑤ MCU sends the command CMD_FILE_ENUM_GO to notify CH376 to continue enumeration;

⑥ CH376 continues to compare filenames. If it finds a file that meets the requirements again, go to the step ③, otherwise proceed to the next step.

⑦ CH376 generates an interrupt to MCU with the interrupt status of ERR_MISS_FILE, indicating that no more files meeting the requirements are found, and the whole enumeration operation ends.

In the step ④, MCU can analyze FAT_DIR_INFO structure to further confirm whether it is matched, or record relevant information in order to further process at the end of the entire enumeration operation. MCU can distinguish the ordinary file from the subdirectory (ATTR_DIRECTORY) through DIR_Attr file attribute unit in the structure. DIR_Name filename unit in the structure can be used for precise comparison of filenames. For example, compare the characters of file extensions DIR_Name[8], [9], [10] with "XLS" to filter particular EXCEL files.

5.20. CMD_FILE_CREATE

This command is used to create a file. If the file already exists, delete it before creating.

Before creating the file command, set the filename of the file to be created through the command CMD_SET_FILE_NAME. The format is the same as the command CMD_FILE_OPEN, but the wildcard character is not supported. If a file with the same name exists, it will be deleted first, and then a new file will be created. If the existing file is not expected to be deleted, confirm that the file does not exist through the command CMD_FILE_OPEN before creating a new file. The default file date and time of the new file are January 1, 2004 and 0:0:0, and the default file length is 1. If this information is required to be modified, it can be realized through the commands CMD_DIR_INFO_READ and CMD_DIR_INFO_SAVE.

5.21. CMD_FILE_ERASE

This command is used to delete a file. If the file has been opened, it will be deleted directly. Otherwise, the file will be opened and deleted automatically, and the subdirectory must be opened first.

For ordinary files, the deletion steps are as follows:

① Confirm that the previous file or directory has been closed, otherwise it will be deleted directly, not affected by the step ②;
② Set the filename to be deleted through the command CMD_SET_FILE_NAME. The wildcard character is not supported;
③ Automatically open the file and delete it through the command CMD_FILE_ERASE.

Subdirectories (or files) must be deleted by the following steps:

① For subdirectories, all files in subdirectories and subdirectories must be deleted in advance;
② Set the subdirectory name (or filename) to be deleted through the command CMD_SET_FILE_NAME. The wildcard character is not supported;
③ Open the subdirectory name (or filename) through the command CMD_FILE_OPEN;
④ Delete the subdirectory (or file) that has been opened in the step ② through the command CMD_FILE_ERASE.

5.22. CMD_FILE_CLOSE

This command is used to close the file or directory (folder) that has been opened. This command requires 1 input data to indicate whether the file length is allowed to be updated. If the data is 0, it will indicate that the file length is disabled to be updated; if the data is 1, it will indicate that the file length is allowed to be automatically updated.

The file shall be closed after the file or directory (folder) is read and written. For root directory operation, it is necessary to close the file. For ordinary file read operation, it is optional to close the file. For ordinary file write operation, choose whether the file length is automatically updated by CH376 while the file is closed.

If the file is read or written in sectors through the command CMD_SEC_LOCATE, CMD_SEC_READ or CMD_SEC_WRITE, the file length automatically updated by CH376 will be calculated in sectors. The file length is usually the multiple of the sector size 512. If the file length is not expected to be a multiple of the sector size, MCU can modify the file length variable through the command CMD_SET_FILE_SIZE before closing the file, or directly modify the file information through the commands CMD_DIR_INFO_READ and CMD_DIR_INFO_SAVE.

If the file is read or written in byte through the command CMD_BYTE_LOCATE, CMD_BYTE_READ or CMD_BYTE_WRITE, the file length automatically updated by CH376 will be calculated in byte, so an appropriate length can be obtained.

5.23. CMD_DIR_INFO_READ

This command is used to read the directory information of the file, namely, FAT_DIR_INFO structure. This command requires 1 input data to specify the index number of the directory information structure to be read in the sector, the index number ranges from 00H to 0FH, and the index number 0FFH corresponds to the currently opened file. The command just reads to the memory buffer. Later, MCU can read the data through the command CMD_RD_USB_DATA0.

Each time a file is opened, CH376 gets directory information of the adjacent 16 files from the USB storage device or SD card to be stored in the memory, and MCU can specify the index numbers 0-15 to respectively correspond to each FAT_DIR_INFO structure, and can also specify the index number 0FFH to get FAT_DIR_INFO structure of the file being opened, to analyze the file information such as date, time, length and attribute.

5.24. CMD_DIR_INFO_SAVE

This command is used to save the directory information of the file. This command refreshes and saves the directory information of 16 files in the memory to a USB storage device or SD card. The steps for modifying the file directory information are as follows:

① File has been opened. Then go to ②, otherwise open the file through the commands CMD_SET_FILE_NAME and CMD_FILE_OPEN;
② Read the FAT_DIR_INFO structure of the current file or adjacent file to the memory buffer through the command CMD_DIR_INFO_READ;
③ Read the data from the memory buffer through the command CMD_RD_USB_DATA0. If no modification is required, the step will end;
④ If it is necessary to modify, read FAT_DIR_INFO structure to the buffer again through the command CMD_DIR_INFO_READ;
⑤ Write the modified data to the internal buffer through the command CMD_WR_OFS_DATA. For example, write two bytes to the offset address 18H (DIR_WrtDate file date unit in the structure) as the new file date.
⑥ Save the modified file directory information to the USB storage device or SD card through the command CMD_DIR_INFO_SAVE.

5.25. CMD_BYTE_LOCATE

This command is used to move the current file pointer in bytes. The command requires to input the number of offset bytes, which is a double-word data (32 bits) expressed by 4 bytes with low bytes in front. If the interrupt status is USB_INT_SUCCESS after the command is executed, the absolute linear sector number LBA (32-bit double-word data expressed by 4 bytes with low bytes in front) corresponding to the current file pointer can be obtained through the command CMD_RD_USB_DATA0. The value will be 0FFFFFFFFH if it has reached the end of the file.

When a file is created or reopened, the current file pointer is set to 0. Move the current file pointer to usually read or write data from the specified location. For example, if MCU expects to skip the first 158 bytes of the file before reading and writing data, it can use the parameter 158 as the offset byte through the command CMD_BYTE_LOCATE. After the command is executed successfully, the read and write operation immediately followed will start from the byte 158. For write operation, if MCU is prepared to continue adding data at the tail of the original file without affecting the previous original data, a large byte offset can be specified, such as 0FFFFFFFFH, to move the file pointer to the end of the original file in order to append the data.

5.26. CMD_BYTE_READ

5.27. CMD_BYTE_RD_GO

The command CMD_BYTE_READ is used to read data blocks from the current location in bytes, and the command CMD_BYTE_RD_GO is used to continue the byte read operation. After successful reading, CH376 automatically and synchronously moves the file pointer so that the next read and write operation can be started from the data reading end position. This command requires to input the number of bytes requested to be read, which is word data (16 bits) expressed by 2 bytes with low bytes in the front (Little-Endian).

A complete byte read operation is typically started through the command CMD_BYTE_READ and consists of several interrupt notifications, several data block reads, and several CMD_BYTE_RD_GO commands. The steps for a complete byte read operation are as follows:

① Open the file and make sure it is in the right position (file pointer);
② MCU sends the command CMD_BYTE_READ and inputs the number of bytes requested to be read to start read operation;
③ CH376 calculates the remaining file length from the current file pointer to the file end position. If the current file pointer is already at the end position of the file, or the number of remaining bytes requested is 0, the read operation will be ended and an interrupt will be notified to MCU, with the interrupt status of USB_INT_SUCCESS. Otherwise, CH376 calculates the number of bytes allowed to be read according to the number of bytes requested, the remaining file length and the internal buffer status, and subtracts the number of bytes allowed this time from the number of bytes requested to get the number of remaining bytes requested, moves the current file pointer at the same time, and then inform MCU of an interrupt, with interrupt status of USB_INT_DISK_READ;
④ MCU analyzes the interrupt status. If the interrupt status is USB_INT_DISK_READ, read the data block through the command CMD_RD_USB_DATA0 and continue; if it is USB_INT_SUCCESS, go to the step ⑥;
⑤ MCU sends the command CMD_BYTE_RD_GO to inform CH376 to continue read operation, and CH376 automatically goes to the step ③;
⑥ At the end of the file or after the number of bytes requested to be read is read, the whole read operation is ended.

MCU accumulates the length of data block obtained after several interrupt notifications to get the total length actually read, and compares it with the number of bytes originally requested. If the latter is greater than the former, the file pointer will be already at the end of the file.

5.28. CMD_BYTE_WRITE

5.29. CMD_BYTE_WR_GO

The command CMD_BYTE_WRITE is used to write data blocks to the current location in bytes, and the command CMD_BYTE_WR_GO is used to continue the byte write operation. After successful writing, CH376 automatically and synchronously moves the file pointer so that the next read and write operation can be started from the data writing end position. This command requires to input the number of bytes requested to be written, which is word data (16 bits) expressed by 2 bytes with low bytes in the front (Little-Endian). When the number of bytes requested is 0, it is only used to refresh the file length.

A complete byte write operation is typically started through the command typically and consists of several interrupt notifications, several data block writes, and several CMD_BYTE_WR_GO commands. The steps for a complete byte write operation are as follows:

① Open or create the file and make sure it is in the right position (file pointer);
② MCU sends the command CMD_BYTE_WRITE and inputs the number of bytes requested to be written to start write operation;
③ CH376 checks the number of bytes requested. If it is 0, CH376 will perform the operation of refreshing the file length, save the file length variable in the memory to the USB storage device or SD card, then output the interrupt status USB_INT_SUCCESS, and go to the step ⑤;
④ CH376 checkes the number of remaining bytes requested. If it is 0, the write operation will be ended and an interrupt will be notified to MCU, with the interrupt status of USB_INT_SUCCESS. Otherwise, CH376 calculates the number of bytes allowed to be written according to the number of bytes requested and the internal buffer status, and subtracts the number of bytes allowed this time

from the number of bytes requested to get the number of remaining bytes requested, and moves the current file pointer at the same time. If it is appended data, CH376 will also update the file length variable in the memory, and then inform MCU of an interrupt, with interrupt status of USB_INT_DISK_WRITE;

⑤ MCU analyzes the interrupt status. If the interrupt status is USB_INT_DISK_WRITE, get the number of bytes allowed this time through the command CMD_WR_REQ_DATA, write the data block and continue; if the interrupt status is USB_INT_SUCCESS, go to the step ⑦;

⑥ MCU sends the command CMD_BYTE_WR_GO to inform CH376 to continue write operation, and CH376 automatically goes to the step ④;

⑦ After the number of bytes requested to be written is written, the whole write operation is ended.

CH376 will automatically update the file length variable in the memory if data is appended directly to the end of the file or if the automatically moved file pointer is beyond the end position of the original file during a write operation. If another write operation is not intended to be performed within a short period of time after the completion of the entire write operation, MCU shall inform CH378 to refresh the file length variable in the memory to the USB storage device or SD card. There are two methods: Write 0 length data similar to the steps ② and ③; execute the command CMD_FILE_CLOSE and allow the length to be updated.

5.30. CMD_DISK_CAPACITY

This command is used to inquire the physical capacity of the disk and supports the USB storage device or SD card. If the interrupt status is USB_INT_SUCCESS after the command is executed, the physical capacity of the disk can be obtained through the command CMD_RD_USB_DATA0, namely, the total number of sectors. The capacity is double-word data (32 bits) expressed by 4 bytes with low bytes in the front. If it is multiplied by the sector size 512, the total physical capacity can be obtained in bytes.

5.31. CMD_DISK_QUERY

This command is used to query the disk space information, including free space and file system type. If the interrupt status is USB_INT_SUCCESS after the command is executed, the total number of sectors (32-bit double word data expressed by 4 bytes with low bytes in the front), the number of residual sectors of the current logic disk (32-bit double word data expressed by 4 bytes with low bytes in the front) and the FAT file system type of the logic disk (refer to CH376_CMD_DATA structure in the file CH376INC.H) can be obtained in turn through the command CMD_RD_USB_DATA0.

5.32. CMD_DIR_CREATE

This command is used to create a new subdirectory (folder) and open it. If the subdirectory already exists, open it directly. Only the first level subdirectory is supported. See EXAM9 for creation of multi-level subdirectories.

Before creating the subdirectory command, set the directory name of the subdirectory to be created through the command CMD_SET_FILE_NAME. The format is the same as the command CMD_FILE_CREATE. If there are common files with the same name, the interrupt status will be ERR_FOUND_NAME; the interrupt status is USB_INT_SUCCESS if a new subdirectory is created successfully or if a preexisting subdirectory is opened. The file date and time of the new subdirectory are the same as that when a new file is created through the command CMD_FILE_CREATE, and the modification method is the same, except that the file attribute is ATTR_DIRECTORY, and the file length is always 0 (the file length of the subdirectory must be 0 according to the FAT specification).

5.33. CMD_SEC_LOCATE

This command is used to move the current file pointer in sector and does not support SD card. The command requires to input the number of offset sectors, which is a double-word data (32 bits) expressed by 4 bytes with low bytes in front. If the interrupt status is USB_INT_SUCCESS after the command is executed, the absolute linear sector number LBA (32-bit double-word data expressed by 4 bytes with low bytes in front) corresponding to the current file pointer can be obtained through the command CMD_RD_USB_DATA0. The value will be 0FFFFFFFFH if it has reached the end of the file.

When a file is created or reopened, the current file pointer is set to 0. Move the current file pointer to usually read or write data from the specified location. For example, if MCU expects to skip the first 18 sectors of the file before reading and writing data, it can use the parameter 18 as the number of offset sectors through the command CMD_SEC_LOCATE. After the command is executed successfully, the read and write operation immediately followed will start from the sector 18. For write operation, if MCU is prepared to continue adding data at the tail of the original file without affecting the previous original data, a large byte offset can be specified, such as 0FFFFFFFFH, to move the file pointer to the end of the original file in order to append the data.

5.34. CMD_SEC_READ

This command is used to get the parameter information for reading data blocks from the current location in sector, and does not support SD card. After the successful execution of command, CH376 automatically and synchronously moves the file pointer so that the next read and write operation can be started from the data reading end position. This command requires to input 1 data to specify the number of sectors to be read, with valid values of 1-255. If the interrupt status is USB_INT_SUCCESS after the command is executed, 8 bytes of returned result can be obtained through the command CMD_RD_USB_DATA0: the first byte is the number of sectors to be allowed to be read, and 0 indicates that the file pointer is already at the end of the file; the last four bytes are the initial absolute linear sector number LBA of the sector block allowed to be read (32-bit double-word data expressed by 4 bytes with the low bytes in the front).

The parameter information for a complete sector read operation is typically obtained through the command CMD_SEC_READ, and then the sector read operation is started through the command CMD_DISK_READ and consists of several interrupt notifications, several data block reads, and several CMD_DISK_RD_GO commands. The steps for a complete sector read operation are as follows:

① Open the file and make sure it is in the right position (file pointer);
② MCU sends the command CMD_SEC_READ and inputs the number of sectors requested to be read;
③ After calculating the parameters, CH376 notifies the notifies of the interrupt with interrupt status of USB_INT_SUCCESS;
④ MCU reads the parameters, the starting LBA of the sector block and the number of sectors allowed to read through the command CMD_RD_USB_DATA0. If the number of sectors allowed to be read is 0, it will indicate that the file is ended, then go to the step ⑨.
⑤ MCU sends the command CMD_DISK_READ and inputs the above parameters to start read operation;
⑥ Each sector is broken into eight 64-byte data blocks. If eight data blocks of all sectors allowed to be read are processed, end the read operation and notify MCU of an interrupt with the interrupt status of USB_INT_SUCCESS, otherwise CH376 reads a 64-byte data block from the USB storage device, then notifies MCU of an interrupt and requests to read data blocks, with the interrupt status of USB_INT_DISK_READ;
⑦ MCU analyzes the interrupt status. If the interrupt status is USB_INT_DISK_READ, read the data block through the command CMD_RD_USB_DATA0 and continue; if it is USB_INT_SUCCESS, go to the step ⑨;
⑧ MCU sends the command CMD_DISK_RD_GO to inform CH376 to continue read operation, and CH376 automatically goes to the step ⑥;
⑨ After the number of sectors allowed to be read is read, the whole read operation is ended.

5.35. CMD_SEC_WRITE

This command is used to get the parameter information for writing data blocks to the current location in sector, and does not support SD card. After the successful execution of command, CH376 automatically and synchronously moves the file pointer so that the next read and write operation can be started from the data writing end position. This command requires to input 1 data to specify the number of sectors requested to be written, with valid values of 0 to 255. When the number of requested sectors is 0, it is only used to refresh the file length. If the interrupt status is USB_INT_SUCCESS after the command is executed, 8 bytes of returned result can be obtained through the command CMD_RD_USB_DATA0: the first byte is the number of sectors to be allowed to be written; the last four bytes are the initial absolute linear sector number LBA of the sector block allowed to be written (32-bit double-word data expressed by 4 bytes with the low bytes in the front).

The parameter information for a complete sector write operation is typically obtained through the command CMD_SEC_WRITE, and then the sector read operation is started through the command CMD_DISK_WRITE and consists of several interrupt notifications, several data block writes, and several CMD_DISK_WR_GO commands. The steps for a complete sector write operation are as follows:

① Open or create the file and make sure it is in the right position (file pointer);
② MCU sends the command CMD_SEC_WRITE and inputs the number of sectors requested to be written;
③ CH376 checks the number of sectors requested. If it is 0, CH376 will perform the operation of refreshing the file length, save the file length variable in the memory to the USB storage device, then output the interrupt status USB_INT_SUCCESS. Otherwise, after calculating the parameters, CH376 notifies MCU of an interrupt with the interrupt status of USB_INT_SUCCESS;
④ MCU reads the parameters, the starting LBA of the sector block and the number of sectors allowed to be written through the command CMD_RD_USB_DATA0. If the number of sectors allowed to be written is 0, it will indicate that the file length is refreshed or the disk is full, then go to the step ⑨;
⑤ MCU sends the command CMD_DISK_WRITE and inputs the above parameters to start write operation;
⑥ Each sector is broken into eight 64-byte data blocks. If eight data blocks of all sectors allowed to be written are processed, end the write operation and notify MCU of an interrupt with the interrupt status of USB_INT_SUCCESS, otherwise notify MCU of an interrupt and requests to write data blocks, with the interrupt status of USB_INT_DISK_WRITE;
⑦ MCU analyzes the interrupt status. If the interrupt status is USB_INT_DISK_WRITE, write a 64-byte data block through the command CMD_WR_HOST_DATA and continue; if it is USB_INT_SUCCESS, go to the step ⑨;
⑧ MCU sends the command CMD_DISK_WR_GO to inform CH376 to continue write operation, and CH376 automatically goes to the step ⑥ after writing the above data blocks to the USB storage device;
⑨ After the number of sectors allowed to be written is written, the whole write operation is ended.

CH376 will automatically update the file length variable in the memory if data is appended directly to the end of the file or if the automatically moved file pointer is beyond the end position of the original file during a write operation. If another write operation is not intended to be performed within a short period of time after the completion of the entire write operation, MCU shall inform CH378 to refresh the file length variable in the memory to the USB storage device. There are two methods: Write 0 length data similar to the steps ② and ③; execute the command CMD_FILE_CLOSE and allow the length to be updated.

5.36. CMD_DISK_BOC_CMD

This command is used to execute BulkOnly transport protocol commands for USB storage devices. Before executing the command, MCU must first write the corresponding CBW packet to CH376 through the command CMD_WR_HOST_DATA (refer to BULK_ONLY_CBW structure in the file CH376INC.H). CH376 requests an interrupt from MCU after the command is executed. If the interrupt status is USB_INT_SUCCESS, the command will be executed successfully. For the data return operation, the data can be returned through the command CMD_RD_USB_DATA0.

5.37. CMD_DISK_READ

5.38. CMD_DISK_RD_GO

The command CMD_DISK_READ is used to read physical sectors from USB storage devices, the command CMD_DISK_RD_GO is used to continue executing the physical sector read operation of the USB storage device, and does not support SD cards.

The command CMD_DISK_READ requires two sets of parameters: a 4-byte sector start address and a 1-byte sector number. The sector start address is the linear sector number LBA, which is a 32-bit double-word data expressed by 4 bytes with the low bytes in front. This command requires to input 5 data, respectively the lowest byte of LBA address, the lower byte of LBA address, the higher byte of LBA address, the highest byte of LBA address and the number of sectors in turn. This command can read the data of 1-255 sectors at a time in the USB storage device.

A complete physical sector read operation is typically started through the command CMD_DISK_READ and consists of several interrupt notifications, several data block reads, and several CMD_DISK_RD_GO commands. The operation steps are as follows:

① MCU ends the command CMD_DISK_READ and specifies the initial LBA and the number of sectors to start the read operation;

② Each sector is broken into eight 64-byte data blocks. If eight data blocks of all sectors required to be read are processed, end the read operation and notify MCU of an interrupt with the interrupt status of USB_INT_SUCCESS, otherwise CH376 reads a 64-byte data block from the USB storage device, then notifies MCU of an interrupt and requests to read data blocks, with the interrupt status of USB_INT_DISK_READ;

③ MCU analyzes the interrupt status. If the interrupt status is USB_INT_DISK_READ, read the data block through the command CMD_RD_USB_DATA0 and continue; if it is USB_INT_SUCCESS, go to the step ⑤. If the interrupt status is USB_INT_DISK_ERR, it will indicate that the operation failed, then go to the step ⑤, and try again if necessary.

④ MCU sends the command CMD_DISK_RD_GO to inform CH376 to continue read operation, and CH376 automatically goes to the step ②;

⑤ After the number of sectors specified to be read is read, the whole read operation is ended.

Even if the command DISK_READ sent by MCU only reads 1 sector, MCU will normally receives 9 interrupts. The first 8 interrupts require MCU to take data, and the last interrupt is to return the final operation status. If 4 sectors are read, MCU will normally receive 33 interrupts, the first 32 interrupts require MCU to take data. If the read operation fails midway, MCU may receive the status USB_INT_DISK_ERR in advance, so as to end the read operation in advance.

5.39. CMD_DISK_WRITE

5.40. CMD_DISK_WR_GO

The command CMD_DISK_WRITE is used to write physical sectors to the USB storage device, the command CMD_DISK_WR_GO is used to continue executing the physical sector write operation of the USB storage device, and does not support SD cards.

The command CMD_DISK_WRITE requires two sets of parameters: a 4-byte sector start address and a 1-byte sector number. The sector start address is the linear sector number LBA, which is a 32-bit double-word data expressed by 4 bytes with the low bytes in front. This command requires to input 5 data, respectively the lowest byte of LBA address, the lower byte of LBA address, the higher byte of LBA address, the highest byte of LBA address and the number of sectors in turn. This command can write the data of 1-255 sectors at a time in the USB storage device.

A complete physical sector write operation is typically started through the command CMD_DISK_WRITE and consists of several interrupt notifications, several data block writes, and several CMD_DISK_WR_GO commands. The operation steps are as follows:

① MCU sends the command CMD_DISK_WRITE and specifies the initial LBA and the number of sectors to start the write operation;

② Each sector is broken into eight 64-byte data blocks. If eight data blocks of all sectors required to be written are processed, end the write operation and notify MCU of an interrupt with the interrupt status of USB_INT_SUCCESS, otherwise notify MCU of an interrupt and requests to write data blocks, with the interrupt status of USB_INT_DISK_WRITE;

③ MCU analyzes the interrupt status. If the interrupt status is USB_INT_DISK_WRITE, write a 64-byte data block through the command CMD_WR_HOST_DATA and continue; if it is USB_INT_SUCCESS, go to the step ⑤. If the interrupt status is USB_INT_DISK_ERR, it will indicate that the operation failed, then go to the step ⑤, and try again if necessary.

④ MCU sends the command CMD_DISK_WR_GO to inform CH376 to continue write operation, and CH376 automatically goes to the step ② after writing the above data blocks to the USB storage device;

⑤ After the number of sectors specified to be written is written, the whole write operation is ended.

Even if the command DISK_WRITE sent by MCU only reads 1 sector, MCU will normally receive 9 interrupts. The first 8 interrupts require MCU to provide data, and the last interrupt is to return the final operation status. If 4 sectors are written, MCU will normally receive 33 interrupts, the first 32 interrupts require MCU to provide data. If the write operation fails midway, MCU may receive the status USB_INT_DISK_ERR in advance, so as to end the write operation in advance.

6. Functional Specification

6.1. Communication Interfaces of MCU

There are three communication interfaces supported between CH376S and MCU: 8-bit parallel interface, SPI synchronous serial interface and asynchronous serial interface. During chip power on reset, CH376S will sample the statuses of WR#, RD#, PCS#, A0, RXD and TXD pins, and select the communication interface according to the combination of these pin statuses. Refer to the following table (X in the table means that this bit is not concerned, 0 means low level, 1 means high level or suspended).

WR# pin	RD# pin	PCS# pin	A0 pin	RXD pin	TXD pin	Select the communication interface
0	0	1	1	1	1	SPI interface
1	1	1	1	1	1	Asynchronous serial interface
1	1/X	1/X	X	1	0	8-bit parallel port
Other state						CH376S chip does not work RST pin always outputs high level

There are two communication interfaces supported between CH376T and MCU: SPI synchronous serial interface and asynchronous serial interface. During power on reset, CH376T chip will sample the state of SPI# pin. If SPI# is at low level, SPI interface will be selected; if SPI# is at high level, the asynchronous serial interface will be selected.

The interrupt request of INT# pin output of CH376 is active at low level by default and can be connected to the interrupt input pin or ordinary input pin of MCU. MCU can get the interrupt request of CH376 in interrupt mode or query mode. To save pins, MCU can get the interrupt in other way without being connected to INT# pin of CH376.

6.2. Parallel Interfaces

The parallel port signal line includes: 8-bit bidirectional data buses D7-D0, read strobe input pin RD#, write strobe input pin WR#, chip selection input pins PCS# and address input pin A0. PCS# pin of CH376 is driven by the address decoding circuit, which is used for device selection when MCU has multiple peripheral devices. CH376 can be easily hooked to the system buses of various 8-bit DSP and MCU through a passive parallel interface, and can coexist with multiple peripheral devices.

For MCU similar to the Intel parallel port timing sequence, RD# and WR# pins of CH376 can be connected to the read strobe output pin and write strobe output pin of MCU respectively. For MCU similar to Motorola parallel port time sequence, RD# pin of the CH376 shall be connected to the low level, and the WR# pin shall be connected to the reading and writing direction output pin R/-W of MCU.

The following table is the truth table of the parallel port I/O operation (X in the table means that this bit is not concerned, and Z means that three states of CH376 are disabled).

PCS#	WR#	RD#	A0	D7-D0	Actual operation on CH376
1	X	X	X	X/Z	CH376 is not selected, and no any operation is made
0	1	1	X	X/Z	Although selected, no any operation is made
0	0	1/X	1	Input	Write a command code to the command port of CH376
0	0	1/X	0	Input	Write data to the data port of CH376
0	1	0	0	Output	Read data from the data port of CH376
0	1	0	1	Output	Read interface status from the command port of CH376: Bit 7 is an interrupt flag, active low, equivalent to INT# pin, Bit 4 is a busy flag, active high, equivalent to BZ pin of SPI interface

CH376 occupies two address bits. When A0 pin is at high level, write a new command, or read the interface status; when A0 pin is at low level, select the data port to read and write the data.

MCU reads and writes CH376 through an 8-bit parallel port. All operations are composed of a command code, several input data and several output data. Some commands do not need input data, and some commands do not have output data. The command operation steps are as follows:

①、 MCU writes the command code to the command port when A0 is 1;
②、 If the command has input data, write the input data in sequence when A0 is 0, one byte at a time;
③、 If the command has output data, read the output data in sequence when A0 is 0, one byte at a time;
④、 The command is completed. The interrupt notification will be generated for some commands. MCU can be paused or go to ① to continue to execute the next command.

6.3. SPI

The SPI synchronous serial interface signal lines include: SPI chip selection input pin SCS, serial clock input pin SCK, serial data input pin SDI, serial data output pin SDO and interface busy status output pin BZ. CH376 can be hooked to SPI serial buses of various DSP and MCU with fewer connections through SPI serial interface by using less connecting wires, or be connected point-to-point over a longer distance.

The SCS pin of CH376 is driven by the SPI chip selection output pin or the general output pin of MCU. SCK pin is driven by the SPI clock output pin SCK of MCU. SDI pin is driven by the SPI data output pin SDO or MOSI, and SDO pin is connected to the SPI data input pin SDI or MISO of MCU. For the hardware SPI interface, it is recommended that the SPI setting is CPOL=CPHA=0 or CPOL=CPHA=1, and the data bit sequence is MSB first. SPI interface of CH376 supports MCU to simulate SPI interface for communication with the common I/O pins.

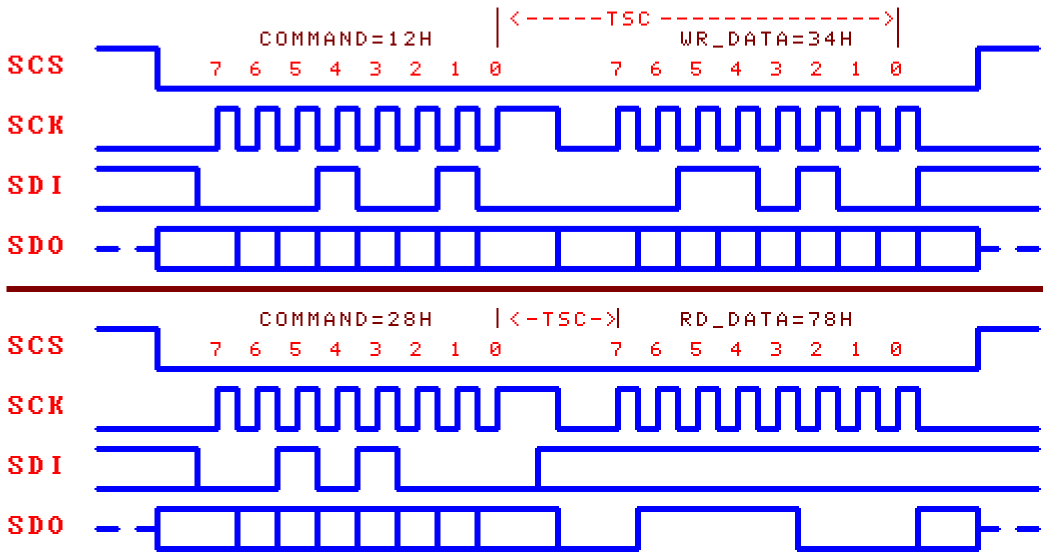
If INT# pin is not connected, the interrupt can be gotten by inquiring SDO pin. The method is as follows: make SDO pin use an input pin of MCU exclusively, and set the SDO pin as the interrupt request output through the command CMD_SET_SDO_INT if the chip selection of MCU is invalid.

The SPI interface of CH376 supports SPI mode 0 and SPI mode 3. CH376 always inputs data from the rising edge of the SPI clock SCK, and outputs data from the falling edge of SCK when the output is allowed. The data bit sequence is MSB first, and 8 full bits are a byte.

Operation procedure of SPI:

① MCU generates the SPI chip selection of CH376, which is active at low level;
② MCU sends a byte of data in SPI output mode. CH376 always takes the first byte received after SPI chip selection SCS is valid as the command code and takes the subsequent bytes as data;
③ MCU inquires that the SPI interface is free when BZ pin wait for CH376, or directly delays the TSC time (about 1.5uS);
④ If it is a write operation, MCU sends a byte of write data to CH376. After waiting for SPI interface to be free, MCU continues to send several bytes of write data, and CH376 receives them in turn until MCU disables SPI chip selection.
⑤ If it is a read operation, MCU receives a byte of data from CH376. After waiting for SPI interface to be free, MCU continues to receive several bytes of data from CH376 until MCU disables SPI chip selection;
⑥ MCU disables the SPI chip selection of CH376 to end the current SPI operation.

The figure below is an SPI interface logic sequence diagram. The first one sends the command 12H and writes 34H, and the second one sends the command 28H and reads the data 78H.



6.4. Asynchronous Serial Interface

The serial data format of CH376 asynchronous serial interface is not compatible with CH375 chip and does not support USB device mode of external firmware.

The signal line of the asynchronous serial interface includes serial data input pin RXD and serial data output pin TXD. CH376 can be connected point-to-point to DSP and MCU through a serial interface by using less connecting wires over a longer distance.

RXD and TXD of CH376 can be connected to the serial data output pin and serial data input pin of MCU respectively.

The serial data format of CH376 is the standard byte transmission mode, consisting of 1 start bit, 8 data bits and 1 stop bit.

CH376 not only supports hardware to set the default serial communication baud rate, but also supports MCU to select the appropriate communication baud rate through the command CMD_SET_BAUDRATE at any time. After each power on reset, the default serial communication baud rate of CH376 is set by the level combination of three pins BZ/D4, SCK/D5 and SDI/D6. Refer to the following table (0 represents low level, and 1 represents high level or suspended).

SDI/D6 pin	SCK/D5 pin	BZ/D4 pin	Default serial communication baud rate after power on reset
1	1	1	9600 bps
1	1	0	57600 bps
1	0	1	115200 bps
1	0	0	460800 bps
0	1	1	250000 bps
0	1	0	1000000 bps
0	0	1	2000000 bps
0	0	0	921600 bps

In order to distinguish the command code from the data, CH376 requires MCU to first send two synchronous code bytes (57H and ABH) through the serial port, then send the command code, and then send or receive the data. CH376 will check the interval between the above two synchronous code bytes and between the synchronous code and the command code. If the interval is greater than the serial input timeout of SER_CMD_TIMEOUT (approximately 32mS), CH376 will discard the synchronous code and the command packet. The operation steps of serial port command are as follows:

① MCU sends the first synchronous code 57H to CH376 through the serial port;
② MCU sends the second synchronous code 0ABH to CH376;
③ MCU sends the command code to CH376;
④ If the command has input data, MCU will send the input data to CH376 in turn, one byte at a time;
⑤ If the command has output data, MCU will receive the output data from CH376 in turn, one byte at a time;
⑥ The command is completed. Interrupt notification may be generated after some commands are executed, and the interrupt status code will be directly sent through the serial port. MCU can be paused or go to ① to continue to execute the next command.

6.5. Other Hardware

CH376 integrates USB-SIE and Phy-I/O, CRC data check, USB-Host controller, USB-Device controller, SD card SPI-Host controller, passive parallel interface, SPI-Slave controller, asynchronous serial interface, double-port SRAM, FIFO, high-speed MCU, firmware program, crystal oscillator and PLL frequency multiplier, power on reset circuit, etc.

ACT# pin of CH376 is used for the status indicator output. After the USB device is not configured or is unconfigured in USB device mode of internal firmware, this pin will output a high level; when the USB device is configured, this pin will output a low level. In the USB host mode, when the USB device is disconnected, the pin outputs a high level; when the USB device is connected, the pin outputs a low level. In the SD card host mode, the pin outputs a low level when the SPI communication of SD card is successful.

ACT# pin of CH376 can be externally connected to an LED with a current limiting resistor connected in series to indicate the relevant status.

The UD+ and UD- pins of CH376 are USB signal lines, which shall be directly connected to the USB bus when working in USB device mode; yet they can be directly connected to USB device when working in USB host mode. If a fuse resistor or inductor or ESD protection device is connected in series for chip safety, the AC and DC equivalent series resistors shall be within 5Ω.

CH376 has a built-in power on reset circuit. Generally, no external reset is required. RSTI pin is used to input an asynchronous reset signal from the outside; when RSTI pin is at high level, CH376 will be reset; when RSTI pin recovers to a low level, CH376 will continuously delay reset for about 35mS, and then enter the normal working status. In order to reliably reset and reduce external interference during the power-on period, a capacitor with a capacity of about 0.1uF can be connected across the RSTI pin and VCC. RST pin (with alias of SD_DO pin) is an active high reset status output pin, which can be used to provide a power on reset signal to the external MCU. RST pin outputs high level when CH376 is reset during power-on or externally forced to be reset and during reset delay; after CH376 is reset and the communication interface is initialized, RST pin recovers to the low level.

When CH376 works normally, 12MHz clock signal shall be provided for it externally. CH376 has a built-in crystal oscillator and an oscillating capacitor. Generally, the clock signal is generated by the built-in oscillator of CH376 through a crystal stable frequency oscillator, and the peripheral circuit is only required to be connected with a crystal with a nominal frequency of 12MHz between XI and XO pins (a 15pF capacitor may be required to be added on XO pin for some crystals). If the 12MHz clock signal is inputted directly from the outside, it shall be inputted from the XI pin, and the XO pin is suspended.

CH376 supports supply voltage of 3.3V or 5V. When the operating voltage is 5V (more than 4V), VCC pin of CH376 will input an external 5V power supply, and V3 pin shall be externally connected to a power decoupling capacitor with a capacity of about 4700pF-0.02uF. When the operating voltage is 3.3V (less than 4V), V3 pin of the CH376 shall be connected to VCC pin, and an external 3.3V power supply shall be inputted at the same time, and the operating voltage of other circuits connected to CH376 shall not exceed 3.3V.

7. Parameters

7.1. Absolute Maximum Value

Critical value or exceeding the absolute maximum value may cause the chip to work abnormally or even be damaged.

Name	Parameter description		Min.	Max.	Unit
TA	Ambient temperature during operation	VCC=5V	-40	85	℃
		VCC=V3=3.3V	-40	85	
		VCC=V3=3V	-40	70	
TS	Ambient temperature during storage		-55	125	℃
VCC	Supply voltage (VCC connects to power, GND to ground)		-0.5	6.0	V
VIO	Voltage on the input or output pins		-0.5	VCC+0.5	V

7.2. Electrical Parameters

Test Conditions: TA=25℃, VCC=5V, Excluding the Pins Connected to the USB Bus
(If the supply voltage is 3.3V, all current parameters in the table need to be multiplied by a factor of 40%)

Name	Parameter description		Min.	Typ.	Max.	Unit
VCC	Power supply voltage	V3 is not connected to VCC	4.3	5	5.3	V
		V3 is connected to VCC, V3=VCC	3.0	3.3	3.6	
ICC	Total supply current during operation	VCC=5V		12	30	mA
		VCC=3.3V		6	15	
ISLP	Supply current at the low power status I/O pin suspended/ internal pull-up	VCC=5V		0.15		mA
		VCC=3.3V		0.05		
VIL	Low level input voltage		-0.5		0.7	V
VIH	High level input voltage		2.0		VCC+0.5	V
VOL	Low level output voltage (4mA draw current)				0.5	V
VOH	High level output voltage (4mA output current)		VCC-0.5			V
IUP	Input current at the input terminal of built-in pull-up resistor		30	80	160	uA
IUP2	Open-drain output pin ACT# and SD_CS Input current at the input terminal of built-in pull-up resistor		100	230	500	uA
IDN	Input current at the input terminal of built-in pull-down resistor		-30	-80	-200	uA
VR	Voltage threshold of power-on reset		2.4	2.7	2.9	V

Note: Low level draw current of ACT# pins and SD_CS pins is 4mA, and the high level output current is 200uA.
During CH376 reset, INT# and TXD pins only provide the high level output current of 80uA.

7.3. Timing Parameters

Test Conditions: TA=25°C, VCC=5V or VCC=3.3V, refer to the attached figure.

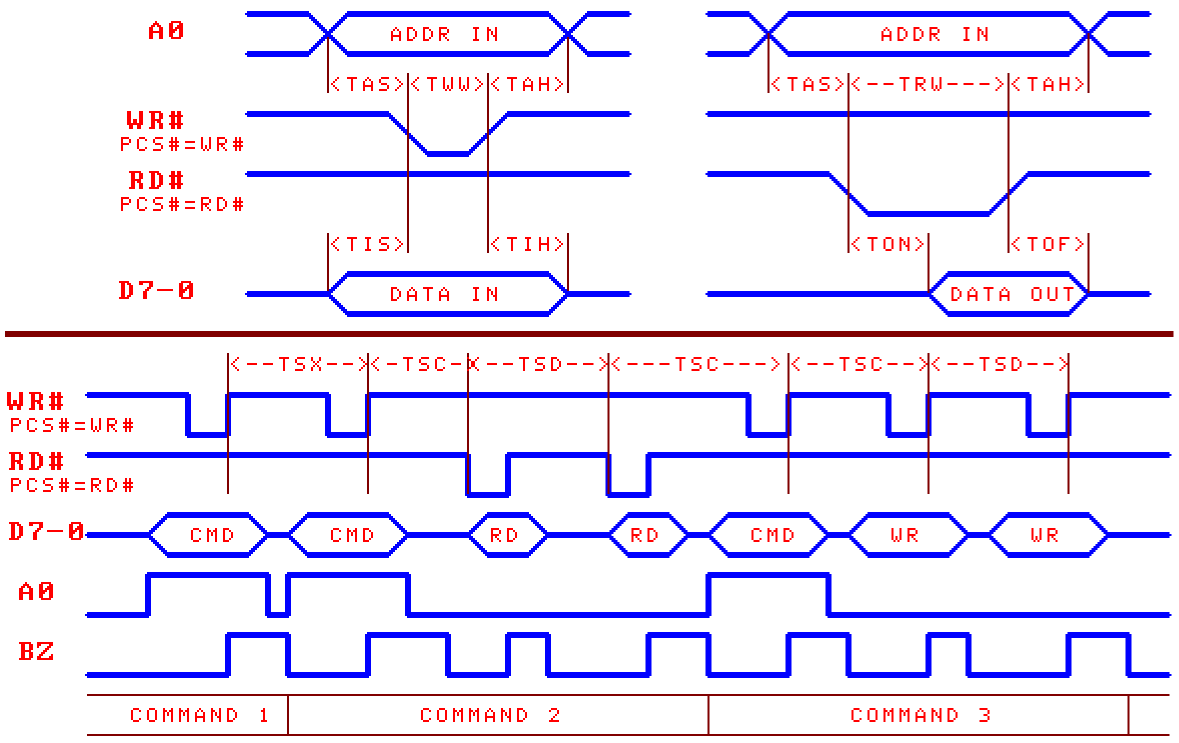
Name	Parameter description	Min.	Typ.	Max.	Unit
FCLK	Input clock frequency of XI pin in USB host mode	11.995	12.00	12.005	MHz
TPR	Internal power-on reset time	25	35	40	mS
TRI	Effective signal width of external reset input	100			nS
TRD	Reset delay after external reset input	25	32	35	mS
TWAK	Wake-up time when exiting from low-power state	3	7	12	mS
TE1	Execution time of command CMD_RESET_ALL		32	35	mS
TE2	Execution time of command CMD_SET_USB_MODE		4	10	uS
TE3	Execution time of command TEST_CONNECT or SET_ENDP?		2	3	uS
TE4	Execution time of command CMD_SET_BAUDRATE	200	1000	2000	uS

TE0	Execution time of other commands		1.5	2	uS
TSX	Interval time between command codes	1.5			uS
TSC	Interval time between command code and data	1.5			uS
TSD	Interval time between data and data	0.6			uS
TINT	Receive the command GET_STATUS until INT# pin undoes the interrupt		1.5	2	uS

7.4. Parallel Port Timing Parameters

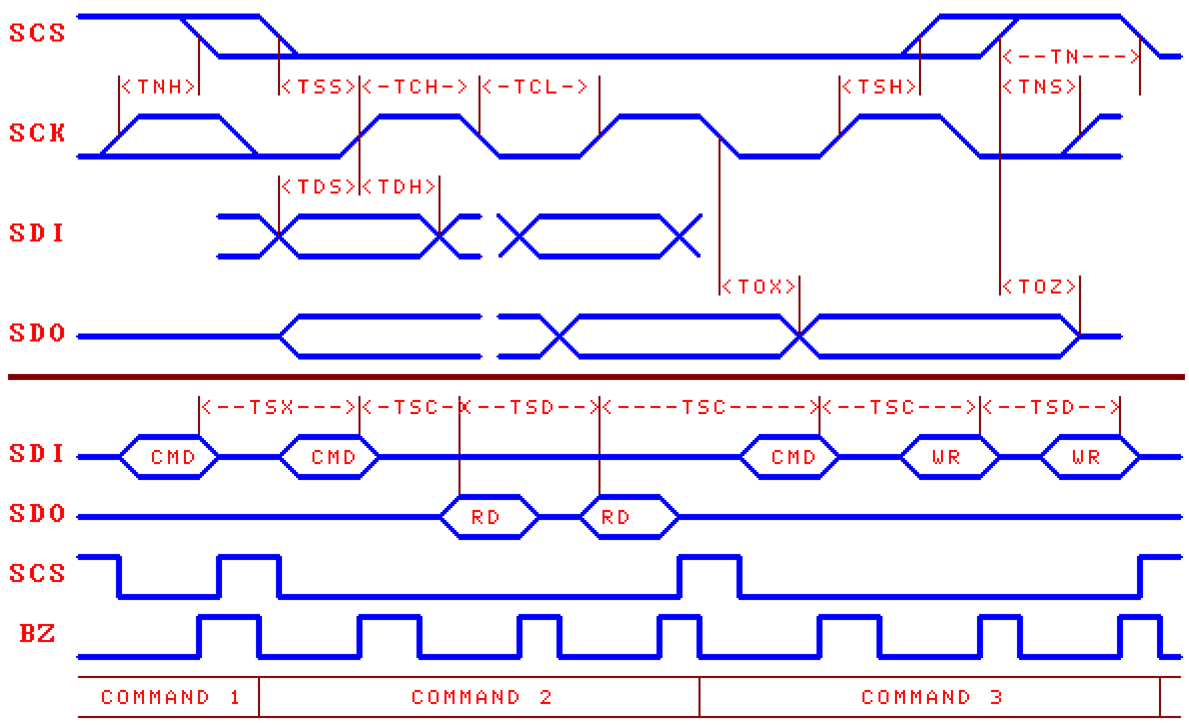
Test Conditions: TA=25℃, VCC=5V, Parameter in Brackets VCC=3.3V, refer to the attached figure
(RD means that RD# signal is valid and PCS# signal is valid; perform read operation when RD#=PCS#=0)
(WR means WR# signal is valid and PCS# signal is valid, perform write operation when WR#=PCS#=0)

Name	Parameter description	Min.	Typ.	Max.	Unit
TWW	Write pulse width	30 (45)			nS
TRW	Read pulse width	40 (60)			nS
TAS	Address input setup time before RD or WR	4 (6)			nS
TAH	Address input hold time after RD or WR	4 (6)			nS
TIS	Data setup time before Write HIGH	0			nS
TIH	Data hold time after Write HIGH	4 (6)			nS
TON	Data output valid after Read active	2	12	18 (30)	nS
TOF	Data output invalid after Read inactve	3	16	24 (40)	nS



7.5. SPI Timing Parameters

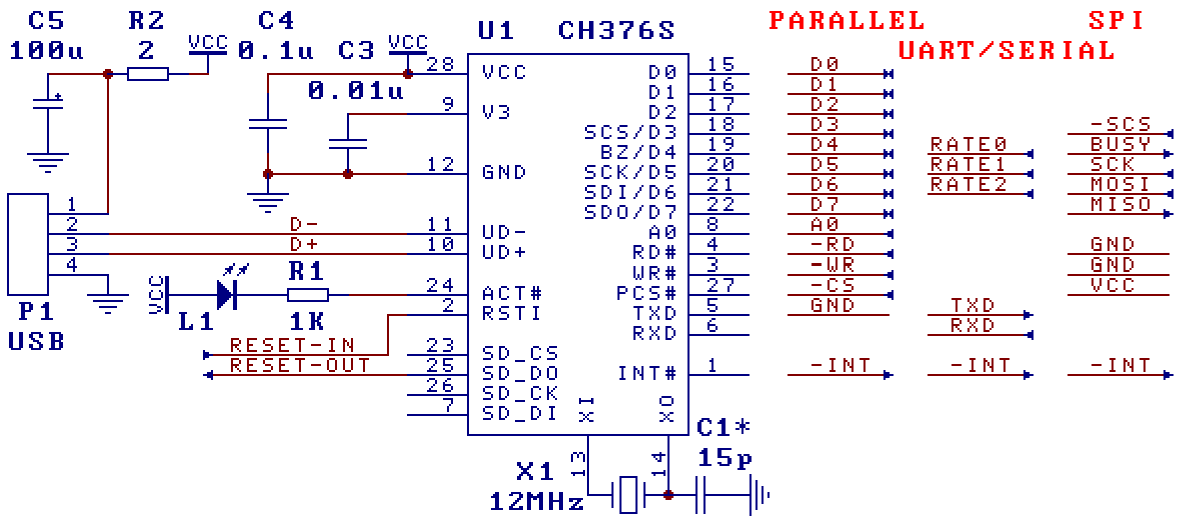
Test Conditions: TA=25℃, VCC=5V, the parameter in brackets VCC=3.3V, referring to the attached figure



Name	Parameter description	Min.	Typ.	Max.	Unit
TSS	Effective setup time of SCS before SCK rising edge	20 (30)			nS
TSH	Effective hold time of SCS after SCK rising edge	20 (30)			nS
TNS	Ineffective setup time of SCS before SCK rising edge	20 (30)			nS
TNH	Hold time of invalid SCS after SCK rising edge	20 (30)			nS
TN	Ineffective time of SCS (SPI operation interval time)	80 (120)			nS
TCH	SCK clock high-level time	14 (18)			nS
TCL	SCK clock low-level time	18 (24)			nS
TDS	SDI input setup time before SCK rising edge	6 (8)			nS
TDH	SDI input hold time after SCK rising edge	2			nS
TOX	Output change from SCK falling edge to SDO	3	8 (12)	12 (18)	nS
TOZ	SCS invalid to SDO output invalid	4		18 (25)	nS

8. Application

8.1. Application of USB Flash Disk, 5V Power Supply (Figure below)



This is the application circuit of CH376 operating USB flash disk at 5V supply voltage.

If CH376 is required to be configured as 8-bit parallel communication mode PARALLEL, TXD shall be connected to GND and other pins shall be suspended.

If CH376 is required to be configured as SPI serial communication mode, RD# and WR# shall be connected to GND and other pins shall be suspended.

If CH376 is required to be configured as the asynchronous serial communication mode UART/SERIA, all pins shall be suspended and the default serial communication baud rate shall be set by SDI/D6, SCK/D5 and BZ/D4 pins. If the communication baud rate of CH376 serial port is required to be dynamically modified, it will be suggested that the I/O pin of MCU control RSTI pin of CH376, so as to reset CH376 to return to the default communication baud rate when necessary. Because RSTI pin has a pull-down resistor, a pull-up resistor with resistance of several KΩ may be required to be added when the quasi-bidirectional I/O pin of MCS51 MCU drives it.

As INT# pins and TXD pin can only provide weak high level output current during CH376 reset, when a long distance connection is conducted, in order to avoid INT# or TXD interference caused by MCU misoperation during CH376 reset, a pull-up resistor with resistance of 2KΩ-5KΩ can be added on INT# pin or TXD pin to maintain a stable high level. After CH376 reset is completed, INT# and TXD pins will be able to provide either a high level output current of 4mA or a low level sinking current of 4mA.

To save pins, MCU can get the interrupt without being connected to INT# pin of CH376. The methods are as follow:

① In the 8-bit parallel port mode, the interface status is gotten by inquiring the status port of CH376 (namely, the command port). Bit 7 is the interrupt flag PARA_STATE_INTB, which is active low and equivalent to the query of INT# pin. If bit 7 is 0, there will be an interrupt request;

② In SPI interface mode, the interrupt is obtained by inquiring SDO pin (after power-on or reset, SDO pin shall be set as the interrupt request output through the command CMD_SET_SDO_INT when the SCS chip selection is invalid). If SDO is at low level, it will indicate that there is an interrupt request;

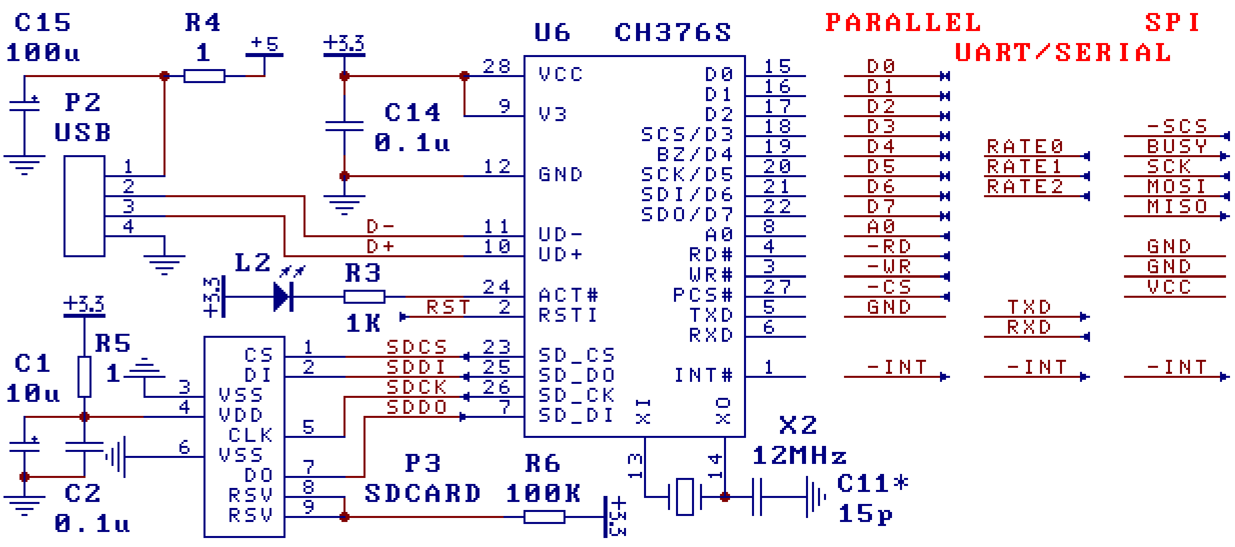
③ When CH376 generates an interrupt notification (INT# becomes low level), it sends an interrupt status code directly through the serial port. MCU receives the interrupt status code, indicating that there is an interrupt request.

R2 is used to limit the current supplied to the external USB device as a USB host. The high-speed electronic switches with current limiting function can be connected in series if necessary. USB supply voltage must be 5V.

The capacitor C3 is used for decoupling the internal power node of CH376. C3 is a monolithic or high-frequency ceramic capacitor with a capacity of 4700pF to 0.02µF. Capacitors C4 and C5 are used for decoupling the external power supply, and C4 is a monolithic or high-frequency ceramic capacitor with a capacity of 0.1µF. The crystal X1 is used for the clock oscillation circuit. The USB-HOST mode requires accurate clock frequency, and the frequency of the crystal X1 is 12MHz±0.4‰. The oscillating capacitor C1 is selectable, with capacity of 0-22pF according to the characteristics of crystal X1.

It shall be noticed that the decoupling capacitors C3 and C4 shall be as close as possible to the connected pins of CH376 when the printed circuit board PCB is designed; the D+ and D- signal lines shall be close to parallel wiring, and ground wire or covered copper shall be provided on both sides to reduce the external signal interference; the length of the signal lines related to the XI and XO pins shall be shortened as far as possible to reduce the high-frequency clock interference to the outside. The ground wire or covered copper shall surround the relevant components.

8.2. Application of SD card and USB Flash Disk, 3.3V Power Supply (Figure below)



This is the application circuit of CH376 operating USB flash disk and SD card at 3.3V or 3V supply voltage.

P3 is a simplified SD card slot. The pin can be directly connected to the I/O or interrupt input pin of MCU in the SD card plug state.

The communication interface configuration is the same as 5V voltage application. Refer to Section 8.1.

R4 is used to limit the current supplied to the external USB device as a USB host. The high-speed electronic switches with current limiting function can be connected in series if necessary. USB supply voltage must be 5V.

The supply voltage of CH376 is 3.3V. In the figure, V3 pin is short-circuited with VCC pin to jointly input 3.3V voltage.

Capacitors C14 and C15 are used for decoupling the external power supply, and C14 is a monolithic or high-frequency ceramic capacitor with a capacity of 0.1µF. The oscillating capacitor C1 is selectable according to the characteristics of crystal X1.

8.3. Application Basis

A USB flash disk (or SD card, the same below) provides a number of physical sectors for data storage, and the size of each sector is typically 512 bytes. As the computer usually organizes the physical sectors of the USB flash disk as the FAT file system, MCU shall also access the data of the USB flash disk in the form of file under the FAT specification in order to facilitate the data exchange between MCU and the computer through the USB disk or SD card.

A USB flash disk can have several files, each of which is a collection of a set of data, being distinguished and identified by the filename. The storage of actual file data may not be continuous, but by multiple blocks (i.e., allocation units or clusters) linked by a set of "pointers", so that the file length can be increased as needed to accommodate more data. Directories (folders) are designed to facilitate classified management. The administrator can artificially specify multiple files to be filed together, for example, files in 2004 are filed in a single directory (folder).

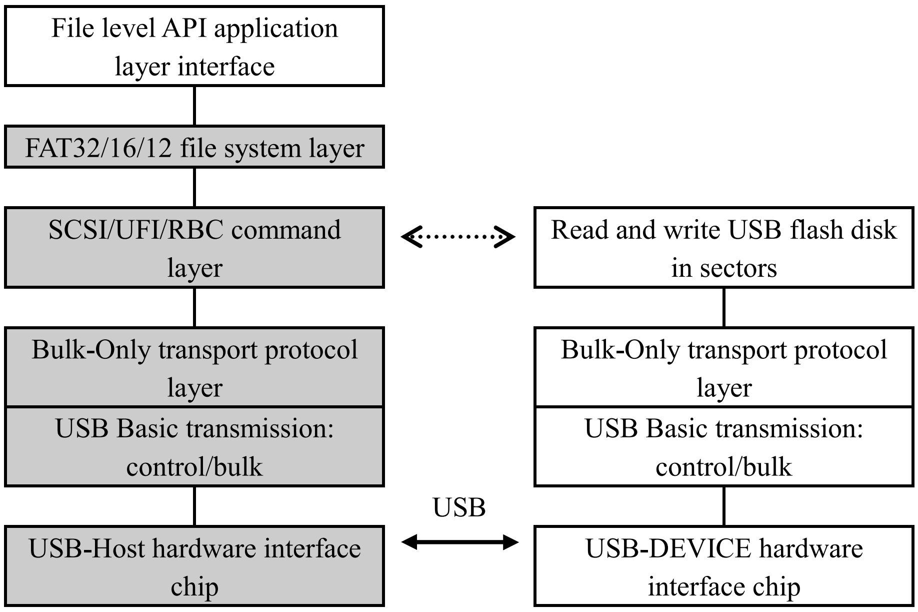
In the FAT file system, the disk capacity is allocated in the basic unit of cluster, and the size of cluster is always a multiple of the sector, so the space occupied by the file is always a multiple of the cluster and also a multiple of the sector. Although the space occupied by files is a multiple of the cluster or sector, in practice, the length of the valid data saved in the file is not necessarily the multiple of the sector, so the FAT file system records the length of the valid data for the current file in the file directory information FAT_DIR_INFO, namely the number of valid data bytes, known as the file size. The file length is always less than or equal to the space that the file occupies.

After the data is written to the file, the file length may not change if the original data is covered. When the file length is more than the original file length and becomes the appended data, it shall change (increase). If the file length in the file directory information is not modified after the data is appended to the file, the FAT file system will consider the data exceeding the file length invalid. Normally, the computer cannot read the data exceeding the file length, even though the data actually exists.

If the data size is small or the data is discontinuous, the file length in the file directory information can be immediately updated after the data is appended each time. However, if the data size is large and it is necessary to continuously write the data, immediate update of file directory information will reduce the efficiency, and frequent modification of file directory information will shorten the service life of the flash memory in the USB flash disk (because the flash memory can be only erased for limited times). In this case, the file length in the file directory information shall be updated after multiple sets of data is continuously written or until the file is closed. The file length in the memory can be refreshed to the file directory information of the USB flash disk file through the command CMD_FILE_CLOSE.

Although CH376 supports a single file of 1GB to the maximum, it is recommended that the length of a single file shall not exceed 100MB in order to improve efficiency. Generally, several KB to several MB is relatively normal. More data can be stored in multiple directories and files.

In general, it is necessary to realize the four levels on the left in the following figure for MCU or embedded system to process the files of the USB flash disk. The internal structure hierarchy of the USB flash disk is shown on the right. As CH376 not only is a general-purpose USB-HOST hardware interface chip, but also has related USB underlying transport firmware program, the Bulk-Only protocol transport firmware program, FAT file system management firmware program, containing 4 levels (parts marked in gray) on the left of the following figure, so the actual MCU program only needs to send file management and read/write command.


File level API application layer interface
FAT32/16/12 file system layer
SCSI/UFI/RBC command layer
Read and write USB flash disk in sectors
Bulk-Only transport protocol layer
USB Basic transmission: control/bulk
Bulk-Only transport protocol layer
USB Basic transmission: control/bulk
USB
USB-Host hardware interface chip
USB-DEVICE hardware interface chip

8.4. Rapid Application Reference Steps

Refer to the example program to call a subprogram that has packaged multiple commands. In the following steps, the original command codes are used for reference only.

8.4.1. Initialize (the necessary step before performing any of the file operations)

① Command CMD_SET_USB_MODE,entering the USB-HOST mode or SD card host mode (mode 3)
② When waiting for USB flash disk or SD card to be connected, CH376 can detect the USB flash disk automatically and generate an interrupt notification, or MCU sends the command CMD_DISK_CONNECT to CH376 and periodically inquires, and the SD card shall be detected by MCU.
③ Command CMD_DISK_MOUNT, initializing USB flash disk or SD card, and testing whether the disk is ready, and retrying 5 times to the maximum after failure;
④ The above steps only need to be performed once. Unless the USB or SD card is disconnected and reconnected, go to the step ②;

8.4.2. Read File

① Command CMD_SET_FILE_NAME + command CMD_FILE_OPEN, to open the file
② Command CMD_BYTE_READ + command CMD_RD_USB_DATA0 + command CMD_BYTE_RD_GO for multiple times, to read the data
③ Command CMD_FILE_CLOSE, closing the file (optional operation)

8.4.3. Overwrite File (overwrite original data, and convert to appended data after exceeding the length of original file)

① Command CMD_SET_FILE_NAME + command CMD_FILE_OPEN, opening the file
② Command CMD_BYTE_WRITE + command CMD_WR_REQ_DATA + command CMD_BYTE_WR_GO for multiple times, writing the data
③ Command CMD_FILE_CLOSE. Set the parameter to 1, close the file and allow to update the file length automatically

8.4.4. Append Data to Existing File

① Command CMD_SET_FILE_NAME + command CMD_FILE_OPEN, opening the file
② Command CMD_BYTE_LOCATE. Set the parameter to 0FFFFFFFFH, and move the file pointer to the end of the file

③ Command CMD_BYTE_WRITE + command CMD_WR_REQ_DATA + command CMD_BYTE_WR_GO for multiple times, writing the data
④ Command CMD_FILE_CLOSE. Set the parameter to 1, close the file and allow to update the file length automatically

8.4.5. Create New File and Write Data

① Command CMD_SET_FILE_NAME + command CMD_FILE_CREATE. Create a file
② Command CMD_BYTE_WRITE + command CMD_WR_REQ_DATA + command CMD_BYTE_WR_GO for multiple times, to write the data
③ Command CMD_FILE_CLOSE. Set the parameter to 1, close the file and allow to update the file length automatically

8.4.6. Read File First and Then Rewrite File

① Command CMD_SET_FILE_NAME + command CMD_FILE_OPEN, to open the file
② Command CMD_BYTE_READ + command CMD_RD_USB_DATA0 + command CMD_BYTE_RD_GO for multiple times, to read the data
③ Command CMD_BYTE_LOCATE. Set the parameter to 0, and move the file pointer to the head of the file
④ Command CMD_BYTE_WRITE + command CMD_WR_REQ_DATA + command CMD_BYTE_WR_GO for multiple times, to write the data
⑤ Command CMD_FILE_CLOSE. Set the parameter to 1, close the file and allow to update the file length automatically

8.4.7. Append data if the file already exists, and create a new file and then write data if the file does not exist

① Command CMD_SET_FILE_NAME + command CMD_FILE_OPEN. Open the file. If ERR_MISS_FILE is returned, it will indicate that the file does not exist, go to the step ③
② Command CMD_BYTE_LOCATE. Set the parameter to 0FFFFFFFFH, move the file pointer to the end of the file, and then go to the step ④
③ Command CMD_FILE_CREATE. Create a file
④ Command CMD_BYTE_WRITE + command CMD_WR_REQ_DATA + command CMD_BYTE_WR_GO for multiple times, writing the data
⑤ Command CMD_FILE_CLOSE. Set the parameter to 1, close the file and allow to update the file length automatically

8.4.8. Modify filename, file date/time, file length and other file directory information. Please refer to the relevant description in EXAM10.

① Command CMD_SET_FILE_NAME + command CMD_FILE_OPEN, opening the file
② Command CMD_DIR_INFO_READ. Set the parameter to 0FFH, and read the file directory information into the memory
③ Read the original file directory information through the command CMD_RD_USB_DATA0
④ Command CMD_DIR_INFO_READ. Set the parameter to 0FFH, and read the file directory information into the memory
⑤ Write the new file directory information through the command CMD_WR_OFS_DATA
⑥ Command CMD_DIR_INFO_SAVE, saving the file directory information
⑦ Optional. Command CMD_FILE_CLOSE, setting the parameter to 0, closing the file and forbidding to update the file length automatically

8.4.9. Create Subdirectory (Folder) (please refer to the description in EXAM9)

① Command CMD_SET_FILE_NAME + command CMD_DIR_CREATE, creating a new subdirectory

(folder)

② Command CMD_FILE_CLOSE, setting the parameter to 0, closing the file and forbidding to update the file length automatically

8.4.10. Process Lowercase and Long Filenames (refer to the description in EXAM11)

8.4.11. Search and Enumerate Filenames, Enumerate All Files (please refer to the description in EXAM13)

8.4.12. For master-slave switching, communication with the computer, USB flash disk or SD card file read and write, please refer to the instructions in EXAM0.

8.5. USB Device Application

Please refer to the datasheet CH372DS1.PDF for CH372 chip and its application information.