U disk and SD card file management control chip CH376

manual

Version: 1

http://wch.cn

1 Overview

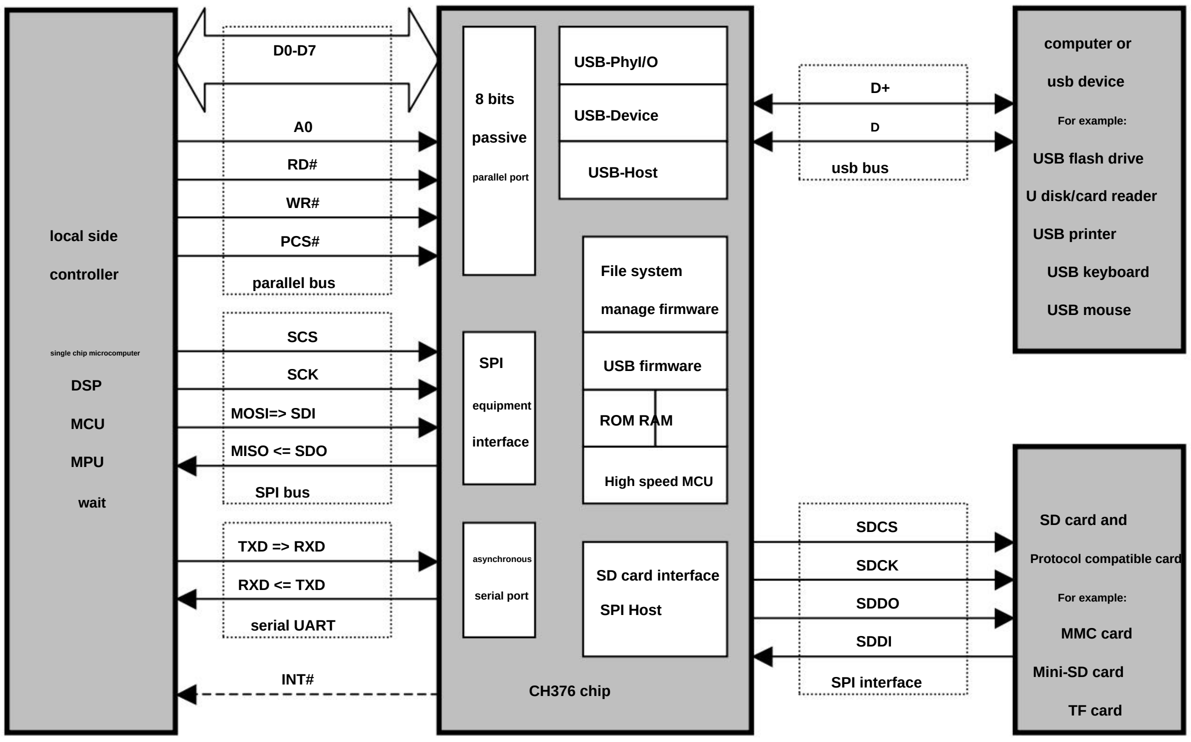
CH376 is a file management control chip, which is used for reading and writing files in U disk or SD card by single chip microcomputer system.

CH376 supports USB device mode and USB host mode, and built-in basic firmware of USB communication protocol, built-in processing Mass-Storage mass storage device dedicated communication protocol firmware, built-in SD card communication interface firmware, built-in FAT16 and Management firmware for FAT32 and FAT12 file systems, supports commonly used USB storage devices (including U disk/USB hard disk/USB flash disk /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 kinds of communication interfaces: 8-bit parallel port, SPI interface or asynchronous serial port, controllers such as microcontroller/DSP/MCU/MPU can It can control the CH376 chip through any of the above communication interfaces, access the files in the U disk or SD card or communicate with the computer.

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

The figure below is the application block diagram of CH376.



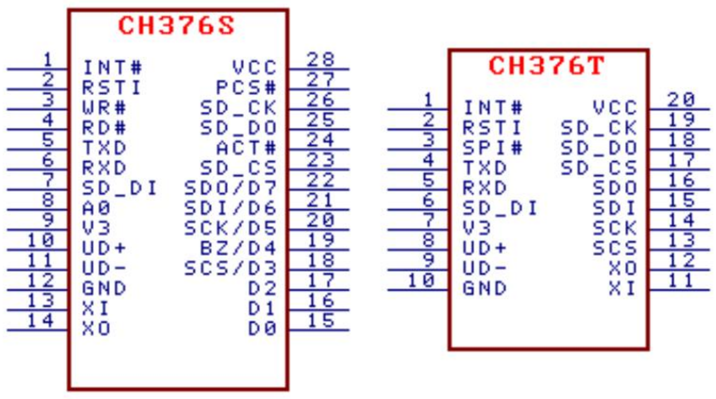
2. Features

- Support 1.5Mbps low-speed and 12Mbps full-speed USB communication, compatible with USB V2.0, only crystal and capacitor are needed for external components.
- Support USB-HOST host interface and USB-DEVICE device interface, and support dynamic switching of host mode and device mode.
- Support control transfer, bulk transfer and interrupt transfer of USB devices.
- Automatically detect the connection and disconnection of USB devices, and provide event notifications of device connection and disconnection.
- Provide 6MHz SPI host interface, support SD card and MMC card and TF card compatible with its protocol.
- Built-in protocol processor for USB control transmission, which simplifies common control transmission.
- Built-in firmware handles dedicated communication protocols for mass storage devices, supports Bulk-Only transfer protocols and SCSI, UFI, RBC or USB storage devices with equivalent command sets (including U disk/USB hard disk/USB flash disk/USB card reader).
- Built-in management firmware of FAT16, FAT32 and FAT12 file system, supports U disk and SD card with capacity up to 32GB.
- Provide file management functions: open, create or delete files, enumerate and search files, create subdirectories, and support long file names.
- Provide file read and write functions: read and write files in multi-level subdirectories with byte as the minimum unit or sector as the unit.

- Provide disk management functions: initialize disk, query physical capacity, query remaining space, and read and write physical sectors.
- Provide 8-bit passive parallel interface with 2MB speed and support parallel data bus connected to MCU.
- Provide SPI device interface with 2MB/24MHz speed, and support SPI serial bus connected to MCU.
- Provide the asynchronous serial port with the highest speed of 3Mbps, support the serial port connected to the single-chip microcomputer, and support the dynamic adjustment of the communication baud rate.
- Support 5V power supply voltage, 3.3V power supply voltage and 3V power supply voltage, and support low power consumption mode.
- USB device mode is fully compatible with CH372 chip; USB host mode is basically compatible with CH375 chip.
- Provide SOP-28 and SSOP20 lead-free package, RoHS compatible, provide SOP28 to DIP28 conversion board, SOP28 package

  The pins are basically compatible with CH375 chip.

3. Packaging



Package form	Plastic body width		Pin Spacing Package Description Ordering Number			
SOP-28	7.62mm	300mil	1.27mm	50mil standard 28-pin patch CH376S 25mil ultra-small 20-pin patch		
SSOP-20	5.30mm	209 miles	0.65mm	CH376T		

4. Pin

CH376S pin number	CH376T pin number	pin name	type	Pin Description
28	20	VCC	power supply	Positive power input terminal, need external 0.1uF power decoupling capacitor
12	10	GND	Power	Common ground terminal, which needs to be connected to the ground wire of the USB bus
9	7	V3	power supply	When the power supply voltage is 3.3V, connect VCC input external power supply, When the power supply voltage is 5V, the external capacity is 0.01uF decoupling capacitor
13	11	XI	Input	the input terminal of the crystal oscillator, need an external 12MHz crystal
14	12	XO	output	The inverting output of the crystal oscillator requires an external 12MHz crystal
10	8	UD+	USB signal	D+ data line of USB bus
11	9	UD-	USB signal	D-data line of USB bus
23	17	SD_CS	open-drain output	SD card SPI interface chip select output, active low, built-in pull-up resistor
26	19	SD_CK	output	Serial clock output for 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 for SD card SPI interface
25	18	RST	output	Before entering SD card mode is Power-on reset and external reset output, active high
22ÿ15	None	D7ÿD0	Bi-directional tri-state	8-bit bidirectional data bus of parallel port, built-in pull-up resistor
18	13	SCS	input	Chip select 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-state output		Serial data output for SPI interface
19 No BZ output				Busy status output of SPI interface, active high
8 No A0 input				Parallel port address input, distinguish between command port and data port, built-in pull-up resistor, When A0=1, you can write commands or read status; when A0=0, you can read and write data
27 Chip select control input of parallel port without PCS# input, active low, built-in pull-up resistor				
4 No RD# input parallel port read strobe input, active low, built-in pull-up resistor				
3 Write strobe input of parallel port without WR# input, active low, built-in pull-up resistor				
none	3	SPI# input is the interface configuration input during chip internal reset, built-in pull-up resistor		
5	4	TXD	enter output	Configure input for the interface during chip internal reset, built-in pull-up resistor, The serial data output of the asynchronous serial port after the chip reset is completed
6	5	RXD input		Serial data input of asynchronous serial port, built-in pull-up resistor
1	1	INT# output		Interrupt request output, active low, built-in pull-up resistor
24 No ACT# open drain output				Status output, active low, built-in pull-up resistor. In the USB host mode, the USB device is connecting status output; In the SD card host mode, it is the SD card SPI communication success status output; In the USB device mode of the built-in firmware, it is the status output of the USB device configuration completion
2	2	RSTI input		External reset input, active high, built-in pull-down resistor

5. Order

The data in this manual, suffix B means binary number, suffix H means hexadecimal number, otherwise it is decimal number.

Low byte first (Little-Endian) double-word data (total 32 bits) means: first the lowest byte (bit 7 ~ bit 0), followed by The lower byte (bit 15 to bit 8), then the higher byte (bit 23 to bit 16), and finally the highest byte (bit 31 to bit 24).

A data stream refers to a data block composed of several consecutive bytes, and the total length of the data block is at least 0 and at most 255.

The numbers in the parentheses of the input data and output data in the table below are the number of bytes of the parameter, and if there are no parentheses, it defaults to one byte.

The MCU mentioned in this manual is basically suitable for DSP or MCU/MPU/SCM etc.

The U disk referred to in this manual includes U disk, USB external hard disk, USB flash disk, USB card reader, etc.

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

This manual mainly provides common file management and control commands for U disk and SD card. About some less commonly used auxiliary life Please refer to the manual (2) CH376DS2.PDF for commands and commands to execute USB basic transactions and control transmission.

The CH376 chip contains all the functions of the CH372 chip. This manual does not provide the description of the CH376 in the USB device mode. Related information can refer to CH372 manual CH372DS1.PDF.

Code	Command Name CMD_	Input Data	Output Data	command purpose
01H	GET_IC_VER		Version	Get chip and firmware version
02H	SET_BAUDRATE	Frequency division factor Frequency division constant	number (wait 1mS) operating state	Set the serial communication baud rate
03H	ENTER_SLEEP ÿÿ			Entering Low Power Sleep Suspend State
05H	RESET_ALL 35mSÿ			Perform a hardware reset
06H	CHECK_EXIST Any data bitwise inverted data 16H Interrupt mode			Test communication interface and working status
0BH	SET_SDO_INT			Set the interrupt mode of the SDO pin of SPI
0CH	GET_FILE_SIZE data 68H		File length (4) (wait	Get the current file length
15H	SET_USB_MODE mode code		10uS) operating state	Set USB working mode
22h	GET_STATUS		Interrupt Status Get the interrupt status and cancel the interrupt request	
27h	RD_USB_DATA0		Data Length Data Stream(n)	From the endpoint buffer of the current USB interrupt or The receive buffer of the host endpoint reads data blocks

2CH	WR_HOST_DATA	Data Length Data Stream(n)		Send buffer to USB host endpoint write data block
2DH	WR_REQ_DATA	 Data stream (n)	Data length	Internally specify the buffer Write the requested data block
2EH	WR_OFS_DATA	offset address data length stream(n)		Specify an offset address to the internal buffer write data block
2FH	SET_FILE_NAME	string(n)		Set the filename of the file to be operated on
30H	DISK_CONNECT		Generate an	Check if the disk is connected
31H	DISK_MOUNT		interrupt Generate an	interrupt to initialize the disk and test whether the disk is ready
32H	FILE_OPEN		Generate an interrupt	to open a file or directory, enumerate files and directories
33h	FILE_ENUM_GO		Generate an interruption and	continue enumerating files and directories
34h	FILE_CREATE		generate interrupt	new file
35H	FILE_ERASE		generate interrupt	delete file
36h	FILE_CLOSE	Whether to allow updates	to generate interrupts	and close currently opened files or directories
37h	DIR_INFO_READ	directory index number	generated to interrupt	reading the directory information of the file
38h	DIR_INFO_SAVE		Generate directory	information for interrupted save files
39h	BYTE_LOCATE	Offset byte count (4)	Generate an interrupt	to move the current file pointer in bytes
3AH	BYTE_READ	Number of requested bytes(2)	Generate an interrupt	to read a block of data from the current position in bytes
3BH	BYTE_RD_GO		generate interrupt	continue byte read
3CH	BYTE_WRITE	request bytes (2)	generate an interrupt	to write data block to the current location in bytes
3DH	BYTE_WR_GO		generate	continue byte writing
3EH	DISK_CAPACITY		interrupt	Query the physical capacity of the disk
3FH	DISK_QUERY		generate	Query disk space information
40H	DIR_CREATE		interrupt generate interrupt	generate new directory and open or open existing directory
4AH	SEC_LOCATE	offset sector number (4)	generate an interrupt	to move the current file pointer in units of sectors
4BH	SEC_READ	The number of requested sectors	generates an interrupt	to read data blocks from the current position in units of sectors
4 ONLY	SEC_WRITE	requests the number of sectors	to generate an interrupt	and writes a data block at the current position in units of sectors
50H	DISK_BOC_CMD		Generate an interrupt	to execute the BO transfer protocol command on the USB memory
54h	DISK_READ	LBA Sector Address(4) Number of Sectors	generate interrupt	Read physical sectors from USB storage
55H	DISK_RD_GO		Generate an interrupt	to continue the physical sector read operation of the USB memory
56h	DISK_WRITE	LBA Sector Address(4) number of sectors	generate interrupt	Write physical sector to USB memory
57h	DISK_WR_GO		Generate an interrupt	to continue the physical sector write operation of the USB memory

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

Status Code	Status Name	Status Description
51H	CMD_RET_SUCCESS	Successful operation
5FH	Commands in the	operation failed

CMD_RET_ABORT table that are marked "causing an interrupt" usually take time to execute. CH376 requests to the MCU after the command execution is completed Interrupt, the MCU can read the interrupt status as the operation status of this command. If the interrupt status is USB_INT_SUCCESS then say indicates that the operation is successful, and some commands have return data (refer to the CH376_CMD_DATA structure in the CH376INC.H file), which can be passed CMD_RD_USB_DATA0 command reads the returned data.

5.1. CMD_GET_IC_VER

This command is used to get the chip and firmware version. One byte of data returned is the version number, its bit 7 is 0, bit 6 is 1, bit 5ÿ Bit 0 is the version number. If the return value is 41H, remove bit 7 and bit 6, and the version number is 01H.

5.2. CMD_SET_BAUDRATE

This command is used to set the serial communication baud rate of CH376. When CH376 works in serial port communication mode, the default communication mode after resetting The baud rate is set by the level combination of the three pins BZ/D4, SCK/D5, and SDI/D6 (refer to Section 6.4 of this manual), these pins are suspended The default is 9600bps when empty. If the MCU supports higher communication speed, you can dynamically adjust the serial communication baud rate through this command. This command needs to input two data, which are baud rate frequency division coefficient and frequency division constant. The following table shows the corresponding relationship with baud rate.

Frequency division factor	Frequency division constant	Serial communication baud rate (bps)	error
02H	B2H	9600	0.16ÿ
02H	D9H	19200	0.16ÿ
03H	98h	57600	0.16ÿ
03H	CCH	115200	0.16ÿ
03H	F3H	460800	0.16ÿ
07h	F3H	921600	0.16ÿ
03H	C4 SOLUTION	100000	0ÿ
03H	FAH	1000000	0ÿ
03H	FEH	3000000	0ÿ
02H		Calculation formula: 750000/(256-constant)	
03H		Calculation formula: 6000000/(256-constant)	

Constant Constant Under normal circumstances, the setting of the serial communication baud rate is completed within 1mS, and after completion, CH376 outputs Out of the operating state, so the single-chip microcomputer should adjust its own communication baud rate in time after issuing the command.

5.3. CMD_ENTER_SLEEP

This command makes the CH376 chip enter the low-power sleep suspend state. After entering the low power consumption state, the clock of the CH376 chip stops oscillating, from To save power, it will not exit the low power consumption state until one of the following two situations is detected: one is to detect that there is a signal on the USB bus (such as The USB host initiates the transmission or the USB device is plugged and unplugged); the second is that the MCU writes new commands to the CH376 chip (no input data commands, such as CMD_GET_IC_VER or CMD_ABORT_NAK commands). For the SPI communication interface mode, SCS chip selection will also be valid It causes CH376 to exit the low power consumption state, so after the MCU issues the CMD_ENTER_SLEEP command, it should immediately disable the SCS chip selection.

Normally, it takes a few milliseconds for the CH376 chip to exit from the low power consumption state and return to the normal working state. After recovering to normal working status, CH376 will generate USB_INT_WAKE_UP event interrupt.

5.4. CMD_RESET_ALL

This command makes CH376 execute hardware reset. Normally, hardware reset is completed within 35mS. For parallel communication In this way, the hardware reset is usually completed within 1mS.

5.5. CMD_CHECK_EXIST

This command is used to test the communication interface and working status to check whether CH376 works normally. This command needs to input 1 data, It can be any data. If CH376 is working normally, then the output data of CH376 is the inverse of the input data. For example, enter The data is 57H, then the output data is A8H. In addition, for CH376 with parallel port communication mode, after it resets and does not receive any command, Before, the data 00H can usually be read from its parallel port.

5.6. CMD_SET_SDO_INT

This command is used to set the interrupt mode of SDO pin of SPI interface. This command first needs to input a data 16H, and then needs to Enter the new interrupt method. There are two interrupt methods: 10H prohibits the SDO pin from being used for interrupt output, and it is a three-state output when the SCS chip selection is invalid. It is convenient to share the SPI bus of the microcontroller with other devices; 90H sets the SDO pin to be in the output state all the time, and it is in the SCS chip selection When it is invalid, it is also used as interrupt request output, which is equivalent to INT# pin, which is used for single-chip microcomputer to query the interrupt request status.

5.7. CMD_GET_FILE_SIZE

This command is used to get the length of the current file, that is, the number of bytes. This command needs to input a data 68H, and the output is currently being printed The length of the opened file, the length is double-word data (32 bits) represented by the low byte first 4 bytes.

If you want to set a new file length, please refer to the manual (2) command CMD_WRITE_VAR32 to set the VAR_FILE_SIZE variable.

5.8. CMD_SET_FILE_SIZE

This command is used to set the length of the current file, that is, the number of bytes. This command first needs to input 1 data 68H, and then needs to input The new file length, which is double-word data (32 bits) represented by 4 bytes with the low byte first.

This command only modifies the file length variable in the memory of CH376, and it will be true only after the commands such as CMD_FILE_CLOSE are executed. Updating the length of the file in the USB storage device or SD card.

5.9. CMD_SET_USB_MODE

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

When the mode code is 00H, switch to the unenabled USB device mode (the default mode after power-on or reset);

When the mode code is 01H, switch to the enabled USB device mode, external firmware mode (serial port connection mode does not support);

When the mode code is 02H, switch to the enabled USB device mode, built-in firmware mode;

When the mode code is 03H, switch to the SD card host mode, which is used to manage and access files in the SD card;

Switch to the disabled USB host mode when the mode code is 04H;

When the mode code is 05H, switch to the enabled USB host mode, and no SOF packet will be generated;

When the mode code is 06H, switch to the enabled USB host mode and automatically generate SOF packets;

When the mode code is 07H, switch to the enabled USB host mode and reset the USB bus;

For the USB device mode, please refer to the CH372 manual. The USB device mode of the CH376 is fully compatible with the CH372 chip.

In USB host mode, not enabled means that it does not automatically detect whether the USB device is connected, so an external microcontroller is required for detection; enabled It means to automatically detect whether the USB device is connected, and when the USB device is connected or disconnected, an interrupt will be generated to notify the external microcontroller. switching After reaching the mode code 06H, CH376 will automatically generate the USB frame period start packet SOF regularly and send it to the connected USB device. model Code 07H is usually used to provide the USB bus reset state to the connected USB device, after switching to other working modes, the USB bus Line reset will end. It is recommended to use mode 5 when there is no USB device, and enter mode 7 and then switch to mode 6 after inserting the USB device.

Normally, setting the USB working mode is completed within 10uS, and the operating status is output after completion.

5.10. CMD_GET_STATUS

This command is used to obtain the interrupt status of CH376 and notify CH376 to cancel the interrupt request. When CH376 requests interrupt from MCU, The MCU obtains the interrupt status through this command, analyzes the cause of the interrupt and handles it.

Interrupt Status Byte	Classification of Interrupt Status
00Hÿ0FH	For the interrupt status of USB device mode, please refer to the CH372 manual
10Hÿ1FH	Operation interruption status of SD card or USB host mode
20Hÿ3FH	The communication failure status of the USB host mode is used to analyze the cause of the operation failure
40Hÿ4FH	File system warning codes in SD card or USB host file mode
80HÿBFH	File system error codes in SD card or USB host file mode

Machine Translated by Google

The following is the operation interruption status of SD card or USB host mode.

status byte	status name	Interrupt Status Analysis Instructions
14H	USB_INT_SUCCESS	SD card or USB transaction or transfer operation or file operation succeeded
15H	USB_INT_CONNECT	USB device connection event detected
16H	USB_INT_DISCONNECT	USB device disconnect event detected
17H	USB_INT_BUF_OVER	Incorrect data transmitted or too much data buffer overflow
18H	USB_INT_USB_READY	The USB device has been initialized (USB address has been assigned)
1DH	USB_INT_DISK_READ	Storage device read operation, request data read
1EH	USB_INT_DISK_WRITE	Storage device write operation, request data write
1FH	USB_INT_DISK_ERR	storage device operation failed

The following is the communication failure status of the USB host mode, which is usually used to analyze the cause of operation failure.

Interrupt status byte	name	Interrupt Status Analysis Instructions
bit 7~bit 6	(reserved bit)	Always 00
bit 5	(flag bit)	Always 1, indicating that the state is an operation failure state
bit 4	IN transactional synchronization flag	For IN transactions, if this bit is 0 then The currently received packet is out of sync and the data may be invalid
bit 3 to bit 0	cause operation at the time of failure usb device The return value	1010=The device returns NAK
		1110=The device returns STALL
		XX00=The device returns timeout, the device does not return
		other values are the PID returned by the device

Below are the file system warning codes and error codes in SD card or USB host file mode.

Status Byte	Status Name	Interrupt Status Analysis Description
41H	ERR_OPEN_DIR	The directory at the specified path is opened
42H	ERR_MISS_FILE	The file in the specified path cannot be found, the file name may be wrong
43h	ERR_FOUND_NAME	A matching filename is found, Or a request to open a directory turns out to open a file
82H	ERR_DISK_DISCON	The disk is not connected, maybe the disk has been disconnected
84H	ERR_LARGE_SECTOR	Disk sector is too large, only supports 512 bytes per sector
92h	ERR_TYPE_ERROR	The disk partition type is not supported and needs to be repartitioned by the disk management tool
A1H	ERR_BPB_ERROR	The disk has not been formatted, or the parameters are wrong, Needs to be reformatted by WINDOWS with default parameters
B1H	ERR_DISK_FULL	The disk file is too full, the remaining space is too little or no
B2H	ERR_FDT_OVER	There are too many files in the directory, there are no free directory entries, disk defragmentation is required, The number of files in the FAT12/FAT16 root directory should be less than 512
B4H	ERR_FILE_CLOSE	The file is closed, it should be reopened if needed

5.11. CMD_RD_USB_DATA0

This command is used to read a block of data from the endpoint buffer of the current USB interrupt or the receive buffer of the host endpoint. first read input The output data is the data block length, that is, the number of bytes of the subsequent data stream. Valid values for data block length are 0 to 255 for file read and write, For the USB bottom layer transmission is 0 to 64, if the length is not 0, then the MCU must read the follow-up data from CH376 one by one.

5.12. CMD_WR_HOST_DATA

This command is used to write a block of data to the transmit buffer of the USB host endpoint. The input data written first is the data block length, that is, is the number of bytes in the subsequent data stream. The valid value of the data block length is 0 to 64, if the length is not 0, the MCU must transfer the subsequent data Write to CH376 one by one.

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 read first is the data block length, also It is the number of bytes of follow-up data stream that CH376 requests MCU to write. The effective value of data block length is 0 to 255 for file reading and writing, and 0 to 64 for USB bottom layer transmission. If the length is not 0, the microcontroller must write subsequent data into CH376 one by one.

5.14. CMD_WR_OFS_DATA

This command is used to write a data block to the specified offset address of the internal buffer. The input data written first is the offset address (adding the offset address to the start address of the internal buffer to obtain the write start address of the command data block), and then the input data written is the length of the data block, that is, the subsequent The number of bytes in the data stream. The valid value of the data block length is 0 to 32, and the sum of the offset address plus the data block length cannot be greater than 32. If the data block length is not 0, the MCU must write subsequent data into CH376 one by one.

5.15. CMD_SET_FILE_NAME

This command is used to set the file name or directory name (path name) of the file or directory (folder) to be operated. Input data is a 0-terminated string and must not exceed 14 characters in length including the terminating 0. For files under multi-level subdirectories, you can decompose the entire path into multiple subdirectory names and a file name, set the name multiple times and open it step by step from the root directory. When a file operation error occurs, you must go back to the root directory and start again Open level by level.

The format of the file name (or directory name, path name) is the same as the short file name format of the DOS system, but the drive letter and colon are not required. The left slash / as the root directory character is equivalent to the right slash \. It is recommended to use the left slash /. All characters must be uppercase letters, numbers or Chinese characters and some special characters. The length of the file name does not exceed 11 characters, of which the main file name does not exceed 8 characters, and the extension does not exceed 3 characters. If there is an extension, then Separate the main filename with a dot. Refer to EXAM11 example to

support long file names. When there is no character in the string (but there is a terminator 0, the same below), it means that the file system is initialized

and no file is opened; when there is only one / or \ (left slash or right slash) in the string, it means that it is opened The

root directory; when the first character of the string is / or \ and the subsequent character is a file name, it means that it is a

file in the root directory; when the string is directly a file name, it means that it is a file

in the current directory. For example, for the FILENAME.EXT file in the root directory, it can be set with the string "/FILENAME.EXT\0". String terminator, "/" in the string indicates the root directory, and "\\" (actually a \ character) can also be used in C language to indicate the root directory. For example, for the file \YEAR2004\MONTH05.NEW\DATE18\ADC.TXT with a long path in the third-level subdirectory, you can open it according

to the following steps: ÿ Use the character string "/YEAR2004\0" to set the file name (directory name) After that, use CMD_FILE_OPEN to open the first-level subdirectory; ÿ After

setting the file name (directory name) with the string "MONTH05.NEW\0", use CMD_FILE_OPEN to open the second-level subdirectory; ÿ Use the string "DATE18\0" to set After the file name (directory name), use CMD_FILE_OPEN to open the third-level subdirectory; ÿ After setting the file name with the string "ADC.TXT\0", use CMD_FILE_OPEN to open the final file.

5.16. CMD_DISK_CONNECT

This command is used to check whether the disk is connected, SD card is not supported. In the USB host mode, this command can check whether the disk is connected at any time, and CH376 will request an interrupt from the MCU after the command is executed. If the operation status is USB_INT_SUCCESS, then there is 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 operations can be performed. Some USB storage devices may need to be initialized multiple times before returning to the successful operation status USB_INT_SUCCESS. In addition, in the process of file operation, this command can also be used to test whether the disk is ready at any time.

Executing the CMD_DISK_MOUNT command for the first time, if the interrupt status is USB_INT_SUCCESS, then it can be executed by CMD_RD_USB_DATA0 Command to obtain data, the data is usually 36 bytes, including the characteristics of the USB storage device and the identification information of the manufacturer and product.

5.18. CMD_FILE_OPEN

This command is used to open a file or directory (folder), enumerate files and directories (folder).

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

If the file is in a multi-level subdirectory with a long path name, it can be opened step by step from the root directory multiple times, first open the first-level subdirectory, then the second-level subdirectory, and finally open the file. Among them, the first opening must start from the root directory, so the first character of the path name must be a slash / or \, and the first character must not be / or \ when opening after the previous level.

If the directory is successfully opened, the interrupt status returns ERR_OPEN_DIR, and the file length is invalid at this time, which is 0FFFFFFFFH. If the file is successfully opened, the interrupt status returns USB_INT_SUCCESS, and the file length is valid at this time. If the specified file or directory (folder) is not found, the interrupt status returns ERR_MISS_FILE. For

example: To open the file \TODAY1.TXT in the root directory, the steps

are as follows: ÿ Use the character string "/TODAY1.TXT\0" to set the file name through the CMD_SET_FILE_NAME

command; ÿ Use the CMD_FILE_OPEN command to

open the file. To open the file \YEAR2004\MONTH05.NEW\DATE18\ADC.TXT under the third-level subdirectory, the steps

are as follows: ÿ Use the character string "/YEAR2004\0" to set the subdirectory name through the

CMD_SET_FILE_NAME command; ÿ Use the CMD_FILE_OPEN command to open the first Level subdirectory, after opening the

directory, if the CMD_GET_FILE_SIZE command is executed, the invalid file length 0FFFFFFFFH will be returned;

ÿ Use the character string "MONTH05.NEW\0" to set the subdirectory name through the CMD_SET_FILE_NAME

command; ÿ Use the CMD_FILE_OPEN command to open the second-

level subdirectory; ÿ Use the character string "DATE18\0" to set the subdirectory name through the

CMD_SET_FILE_NAME command; ÿ Use the CMD_FILE_OPEN

command command to open the third-level subdirectory; ÿ Use the string "ADC.TXT\0" to set the file

name through the CMD_SET_FILE_NAME command; ÿ Use the CMD_FILE_OPEN command to open the final file. After opening the file,

if the CMD_GET_FILE_SIZE command is executed, the actual file length will be returned.

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

follows: ÿ Use the character string "\0" to set the file name through the CMD_SET_FILE_NAME

command; ÿ Execute the CMD_FILE_OPEN command, then the file system will be initialized (if it has been initialized, it will return

directly). To open the root directory (for example, when dealing with long filenames), the steps are as follows:

ÿ Use the character string "/\0" to set the file name through the CMD_SET_FILE_NAME

command; ÿ Execute the CMD_FILE_OPEN command, then the root directory will be opened (it must be closed with CMD_FILE_CLOSE after use).

5.19. CMD_FILE_ENUM_GO

This command is used to continue enumerating files and

directories (folders). If you need to search and query

files, the steps are as follows: ÿ Use the wildcard * to replace all or part of the characters in the file name to be queried. There can be no more

characters after the wildcard *. Set the string containing the wildcard * as the file name through the CMD_SET_FILE_NAME command.

For example, the string "/*\0" indicates that all files or directories in the root directory are to be enumerated, and the string "USB*\0"

indicates that all names in the current directory start with "USB" File or directory, the file name (or directory name) that meets the requirements includes "USB

"USB1234", "USB", "USBC.H", etc., but excluding "XUSB", "U.SB", "U2SB", "MY.USB", etc.;

ÿ Use the CMD_FILE_OPEN command to start enumerating files and

directories; ÿ CH376 compares each file name, and whenever it finds a file that meets the requirements, it will generate an interrupt to the

microcontroller, and the interrupt status is USB_INT_DISK_READ, requesting the

microcontroller to read data from CH376; ÿ The MCU reads the data through the CMD_RD_USB_DATA0 command, analyzes and processes it immediately or saves it

Directory information (refer to the FAT_DIR_INFO structure definition in the CH376INC.H file);

ÿ The MCU issues the CMD_FILE_ENUM_GO command to inform CH376 to continue

enumeration; ÿ CH376 continues to compare the file names, if it finds a file that meets the requirements again, then go to step ÿ, otherwise continue to the next step;

ÿ CH376 generates an interrupt to the MCU, and the interrupt status is ERR_MISS_FILE, indicating that no more files that meet the requirements are found, and the entire enumeration

operation ends. In the above step ÿ, the microcontroller can analyze the obtained FAT_DIR_INFO structure to further confirm whether it matches, or record relevant information for further processing after the entire enumeration operation is completed. The microcontroller can distinguish whether it is a normal file or a subdirectory (ATTR_DIRECTORY) through the DIR_Attr file attribute unit in the structure, and can perform accurate comparison of file names through the DIR_Name file name unit in the structure. For example, compare the characters of three units of file extension DIR_Name[8], [9], [10] with "XLS" to filter specific EXCEL type files.

5.20. CMD_FILE_CREATE

This command is used to create a new file. If the file already exists, delete it first and then

create a new one. Before creating a new file command, you should first set the file name of the new file through the CMD_SET_FILE_NAME command. The format is the same as that of the CMD_FILE_OPEN command, but wildcards are not supported. If a file with the same name exists, the file with the same name will be deleted first, and then a new file will be created. If you do not want the existing file to be deleted, you should confirm that the file does not exist through the CMD_FILE_OPEN command before creating a new one. The default date and time of the new file is 0:00:00 on January 1, 2004, and the default length of the file is 1. If you need to modify these information, you can use the CMD_DIR_INFO_READ and CMD_DIR_INFO_SAVE commands.

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 first and then deleted, and the subdirectory must be opened first. For

ordinary files, the deletion steps are as

follows: ÿ Confirm that the previous file or directory is closed, otherwise it will be deleted directly without being affected by

step ÿ; ÿ Set the file name to be deleted through the CMD_SET_FILE_NAME command, wildcards are not supported;

ÿ Pass The CMD_FILE_ERASE command opens the file and deletes it by itself.

For subdirectories (or files) must be deleted according to the following

steps: ÿ For subdirectories, all files in the subdirectory and subdirectories must be deleted in advance;

ÿ Set the subdirectory name (or file name) to be deleted through the CMD_SET_FILE_NAME command, Wildcards are not supported; ÿ Open

the subdirectory name (or file name) through the CMD_FILE_OPEN command; ÿ Delete

the subdirectory (or file) that has been opened in step ÿ through the CMD_FILE_ERASE command.

5.22. CMD_FILE_CLOSE

This command is used to close the currently opened file or directory (folder). This command requires 1 input data indicating whether Allow to update the file length, 0 to prohibit the update of the file length, and 1 to allow the automatic update of the file length.

After opening a file or directory (folder) for reading and writing, the file should be closed. For operations on the root directory, closing files is required. For ordinary file read operations, closing the file is optional. For the write operation of ordinary files, when closing the file, you can choose whether to automatically update the file length by CH376.

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

If the file is processed in bytes by the CMD_BYTE_LOCATE, CMD_BYTE_READ or CMD_BYTE_WRITE command Read and write, then the length of the file automatically updated by CH376 is calculated in bytes, 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, that is, the FAT_DIR_INFO structure. This command requires 1 input data, specifying 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. This command just reads the memory buffer, and then the MCU can read the data through the CMD_RD_USB_DATA0 c

After opening a file each time, CH376 takes out the directory information of 16 adjacent files from the USB storage device or SD card and stores them in the internal memory. The single-chip microcomputer can specify the index number 0~15 corresponding to each FAT_DIR_INFO structure, or specify the index number 0FFH To get the FAT_DIR_INFO structure of the file currently being opened, to analyze the file date, time, length, attributes and other information.

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 the USB storage device. device or SD card. The steps to modify the file directory information are as follows:

ÿ If the file is already opened, turn to ÿ, otherwise, open the file through the CMD_SET_FILE_NAME and CMD_FILE_OPEN commands; ÿ Read the FAT_DIR_INFO structure of the current file or adjacent files into the memory buffer through the CMD_DIR_INFO_READ command; ÿ Read data from the memory buffer through the CMD_RD_USB_DATA0 command, if If modification is not required, the step ends; ÿ If modification is required, read the FAT_DIR_INFO structure to the buffer again through the CMD_DIR_INFO_READ command; ÿ Write the modified data to the specified offset address of the internal buffer through the CMD_WR_OFS_DATA command, for example, to the offset address 18H (that is, the DIR_WrtDate file date unit in the structure) writes two bytes as the new file date; ÿ Save the modified file directory information to the USB storage device or SD card through the CMD_DIR_INFO_SAVE command.

5.25. CMD_BYTE_LOCATE

This command is used to move the current file pointer in bytes. This command needs to input the number of offset bytes, the number of offset bytes is low Double word data (32 bits) represented by the first 4 bytes. If the interrupt status is USB_INT_SUCCESS after the command is executed, then the absolute linear sector number LBA corresponding to the current file pointer can be obtained by the CMD_RD_USB_DATA0 command (32-bit double-word data represented by the 4 bytes before the low byte). end of file, then the value is 0FFFFFFFFH.

When the file is newly created or reopened, the current file pointer is 0, and the current file pointer is moved, which is usually used to read and write data from the specified position. For example, if the MCU wants to skip the first 158 bytes of the file before reading and writing data, then you can use the CMD_BYTE_LOCATE command with the parameter 158 as the offset byte number. After the command is executed successfully, the following read and write operations will start from the 158 bytes start. For the write operation, if the MCU is going to continue adding data at the end of the original file without affecting the previous original data, you can specify a large byte offset, such as 0FFFFFFFFH, to move the file pointer to the end of the original file for appending data.

5.26. CMD_BYTE_READ

5.27. CMD_BYTE_RD_GO

The CMD_BYTE_READ command is used to read the data block from the current position in bytes, and the CMD_BYTE_RD_GO command is used to continue the byte read operation. After reading successfully, CH376 automatically moves the file pointer synchronously, so that the next read and write operation can start from the end position of the data read this time. This command needs to input the number of bytes requested to be read, and the requested number of bytes is word data (16 bits) represented by 2 bytes with the low byte first (Little-Endian).

A complete byte read operation is usually initiated by a CMD_BYTE_READ command, and is notified by several interrupts and several Data block reading consists of several CMD_BYTE_RD_GO commands. The complete byte read operation steps are as follows: ÿ Open the file, and confirm that it is in the appropriate position (file pointer); ÿ The MCU sends the CMD_BYTE_READ command and enters the number of bytes requested to read, and starts the read operation; ÿ CH376 calculates from the current file pointer The remaining length of the file between the beginning and the end of the file. If the current file pointer is already at the end of the file, or the remaining number of requested bytes is 0, then end the read operation and interrupt the notification to the MCU. The interrupt status is USB_INT_SUCCESS, otherwise, according to the request The number of sections, the remaining length of the file, and the internal buffer state calculate the number of bytes allowed to be read this time, and subtract the number of bytes allowed this time from the number of bytes requested to obtain the remaining number of bytes requested, and move the current file at the same time Pointer, then the interrupt notification MCU, the interrupt status is USB_INT_DISK_READ; ÿ The MCU analyzes the interrupt status, if it is USB_INT_DISK_READ, then read the data through the CMD_RD_USB_DATA0 command According to the block and continue, if it is USB_INT_SUCCESS, then go to step ÿ; ÿ The MCU sends the CMD_BYTE_RD_GO command to notify CH376 to continue the read operation, and CH376 automatically goes to step ÿ; ÿ When the file ends or all the bytes requested to be read are read, the entire read operation ends. The single-chip computer accumulates the length of the data block obtained after several interrupt notifications to obtain the total length actually read, which is different from the initially requested number of bytes

Comparison, if the latter is greater than the former, then the file pointer is already at the end of the file.

5.28. CMD_BYTE_WRITE

5.29. CMD_BYTE_WR_GO

The CMD_BYTE_WRITE command is used to write a data block to the current location in bytes, and the CMD_BYTE_WR_GO command is used to continue the byte write operation. After writing successfully, CH376 automatically moves the file pointer synchronously, so that the next read and write operation can start from the end position of the data written this time. This command needs to input the number of bytes requested to be written, and the requested number of bytes is word data (16 bits) represented by 2 bytes with the low byte first (Little-Endian). When the number of requested bytes is 0, it is only used to refresh the file length.

A complete byte write operation is usually initiated by a CMD_BYTE_WRITE command, and is notified by several interrupts and several It is composed of data block writing and several CMD_BYTE_WR_GO commands. The complete byte writing operation steps are as follows: ÿ Open or create a new file, and confirm that it is in an appropriate position (file pointer); ÿ The MCU sends the CMD_BYTE_WRITE command and enters the number of bytes requested to be written to start the writing operation; ÿ CH376 checks the requested The number of bytes, if it is 0, execute the operation of refreshing the file length, save the file length variable in the memory to the USB storage device or SD card, and output the interrupt status as USB_INT_SUCCESS after completion, go to step ÿ; ÿ CH376 checks the remaining request words The number of sections, if it is 0, then end the write operation and interrupt the notification of the microcontroller, the interrupt status is USB_INT_SUCCESS, otherwise calculate the number of bytes allowed to be written this time according to the number of bytes requested and the state of the internal buffer, and subtract the number of bytes allowed this time from the number of bytes requested to obtain the remaining number of bytes requested, and move at the same time The current file pointer, if it is to append data, then it is necessary to update the file length variable in the memory, and then interrupt the microcontroller, and the in USB_INT_DISK_WRITEÿ

ÿ MCU analyzes the interruption status, if it is USB_INT_DISK_WRITE, then use the CMD_WR_REQ_DATA command to get the number of bytes allowed this time and write the data block to continue, if it is USB_INT_SUCCESS, then go to step 7; CH376 automatically turns to step ÿ; ÿ After all the bytes requested to be written are written in, the entire write operation ends. If adding data directly to the end of the file, or during the writing operation, the automatically moved file pointer exceeds the end position of the original file, then CH376 will automatically update the file length variable in the memory. After the entire write operation is completed, if another write operation is not planned within a short time, the MCU should notify CH376 to refresh the file length variable in the internal memory to the USB storage device or SD card. There are two methods: similar to the above step ÿ and ÿ Write 0-length data; execute the CMD_FILE_CLOSE command and allow the length to be updated.

5.30. CMD_DISK_CAPACITY

This command is used to query the physical capacity of the disk, and supports USB storage device or SD card. If the interrupt status is USB_INT_SUCCESS after the command is executed, then the physical capacity of the disk can be obtained by the CMD_RD_USB_DATA0 command, that is, the total number of sectors. If multiplied by the sector size 512, the total physical capacity in bytes can be obtained.

5.31. CMD_DISK_QUERY

This command is used to query disk space information, including remaining space and file system type. If the interrupt status is USB_INT_SUCCESS after the command execution is completed, then the total sector number of the logical disk (32-bit double-word data represented by the first 4 bytes with the low Sector number (32-bit double-word data represented by 4 bytes with low byte first), FAT file system type of logical disk (refer to CH376_CMD_DATA structure in CH376INC.H file).

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. Directory, refer to the EXAM9 example to support the creation of multi-level subdirectories.

Before the new subdirectory command, the directory name of the subdirectory to be created should be set through the CMD_SET_FILE_NAME command, and the format is the same as that of the CMD_FILE_CREATE command. If an ordinary file with the same name exists, the interrupt status is ERR_FOUND_NAME;

If the subdirectory is successfully created or an existing subdirectory is opened, the interrupt status is USB_INT_SUCCESS. The file date and time information of the new subdirectory is the same as that of the new file created by the CMD_FILE_CREATE command, and the modification method is the same, except that the file attribute is ATTR_DIRECTORY, and the file length is always 0 (according to the FAT specification, the file length of the subdirectory must be 0).

5.33. CMD_SEC_LOCATE

This command is used to move the current file pointer by sector and does not support SD card. This command needs to input the offset sector number, which is double-word data (32 bits) represented by 4 bytes before the low byte. If the interrupt status is USB_INT_SUCCESS after the command is executed, then the absolute linear sector number LBA (32-bit double-word data represented by the 4 bytes before the low byte) corresponding to the current file pointer can be obtained by the CMD_RD_USB_DATA0 command. end of file, then the value is 0FFFFFFFFH.

When the file is created or reopened, the current file pointer is 0, and the current file pointer is moved, which is usually used to read and write data from the specified position. For example, if the microcontroller wants to skip the first 18 sectors of the file before reading and writing data, then you can use the CMD_SEC_LOCATE command with a parameter of 18 as the offset sector number. After the command is executed successfully, the following read and write operations will start from the first 18 sectors start. For the write operation, if the MCU is going to continue adding data at the end of the original file without affecting the previous original data, you can specify a large byte offset, such as 0FFFFFFFFH, to move the file pointer to the end of the original file for appending data.

5.34. CMD_SEC_READ

This command is used to obtain the parameter information of reading the data block from the current position in units of sectors, and does not support SD cards. After the command is successfully executed, CH376 automatically moves the file pointer synchronously, so that the next read and write operation can start from the end position of the data read this time. This command needs to input 1 data, which specifies the number of sectors to be read, and the valid value is 1 to 255. If the interrupt status is USB_INT_SUCCESS after the command execution is completed, then the CMD_RD_USB_DATA0 command can obtain a total of 8 bytes of return results: the first byte is the number of sectors that are allowed to be read, if it is 0, it means that the file pointer is already at the end of the file; The 4 bytes are the starting absolute linear sector number LBA of the sector block that is allowed to be read (32-bit double-

word data represented by the 4 bytes before the low byte). A complete sector read operation usually consists of a CMD_SEC_READ command to obtain parameter information, and then a CMD_DISK_READ command to start the operation, and consists of several interrupt notifications, several data block reads and several CMD_DISK_RD_GO commands.

The complete sector read operation steps are as follows: ÿ Open the file, and confirm
that it is in the proper position (file pointer); ÿ The MCU issues the CMD_SEC_READ command and
enters the number of sectors requested to be read; ÿ CH376 interrupts the MCU after calculating the parameters, and
interrupts The status is USB_INT_SUCCESS; ÿ The MCU reads the parameters, the starting LBA of the sector block and the number of sectors allowed to be read through the CMD_RI
The number of sectors allowed to be read is 0, indicating the end of the file, then go to step
ÿ; ÿ The MCU issues the CMD_DISK_READ command and enters the above parameters to start the read
operation; ÿ Each sector is decomposed into 8 64-byte data blocks, If the 8 data blocks of all sectors allowed to be read have been processed, then end the
read operation and interrupt the MCU, the interrupt status is USB_INT_SUCCESS, otherwise CH376 reads a 64-byte data block from the USB storage
device, and then The interrupt notifies the microcontroller to request to read the data block, and the interrupt status is
USB_INT_DISK_READ; ÿ SCM
analysis interrupt status, if it is USB_INT_DISK_READ, then read the data through CMD_RD_USB_DATA0 command
Data block and continue, if it is USB_INT_SUCCESS, then go to step ÿ;
ÿ The MCU sends the CMD_DISK_RD_GO command to inform CH376 to continue the read operation, and CH376 automatically turns to
step ÿ; ÿ After reading all the sectors allowed to be read, the entire read operation ends.

5.35. CMD_SEC_WRITE

This command is used to obtain the parameter information of writing the data block to the current location in units of sectors, and does not support SD cards. After the command is successfully executed, CH376 automatically moves the file pointer synchronously, so that the next read and write operation can start from the end position of the data written this time. This command needs to input 1 data, which specifies the number of sectors to be written. The valid value is 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 execution is completed, then the return result of 8 bytes can be obtained by the CMD_RD_USB_DATA0 command: the first byte is the number of sectors that are allowed to be written; the last 4 bytes are the sector blocks that are allowed to be written The starting absolute linear sector number LBA (32-bit double-word data represented by the 4 bytes before the low

A complete sector write operation usually consists of a CMD_SEC_WRITE command to obtain parameter information, and then a CMD_DISK_WRITE command to start the operation, and consists of several interrupt notifications, several data block writes and several CMD_DISK_WR_GO commands. The complete sector write operation steps are as follows:

ÿ Open or create a new file, and confirm that it is in a suitable position (file pointer); ÿ The MCU sends the CMD_SEC_WRITE command and enters the number of sectors requested to be written;

ÿ CH376 checks the number of sectors requested, if it is 0, then executes to refresh the file length Operation, save the file length variable in the internal memory to the USB storage device, and output the interrupt status as USB_INT_SUCCESS after completion, otherwise CH376 will interrupt and notify the MCU after calculating the parameters, and the interrupt status will be USB_INT_SUCCESS;

ÿ The MCU reads the parameters through the CMD_RD_USB_DATA0 command, the starting LBA of the sector block and the number of sectors allowed to be written. If the number of sectors allowed to be written is 0, it means that the file length is refreshed or the disk is full, then go to the step ÿ;

ÿ The MCU issues the CMD_DISK_WRITE command and enters the above parameters to start the write operation; ÿ Each sector is decomposed into 8 data blocks of 64 bytes. End the write operation and interrupt the notification to the microcontroller, the interrupt status is USB_INT_SUCCESS, otherwise the interrupt notifies the microcontroller to request to write the data block, the interrupt status is USB_INT_DISK_WRITE;

ÿ Analyze the interrupt status of the microcontroller, if it is USB_INT_DISK_WRITE, then write it through the CMD_WR_HOST_DATA command A 64-byte data block and continue, if it is USB_INT_SUCCESS, then go to step ÿ;

ÿ The MCU sends the CMD_DISK_WR_GO command to inform CH376 to continue writing, and CH376 writes the above data blocks into the USB storage After the device is installed, it will automatically go to step ÿ; ÿ After the number of sectors allowed to be written is all written, the entire write operation is over. If adding data directly to the end of the file, or during the writing operation, the automatically moved file pointer exceeds the end position of the original file, then CH376 will automatically update the file length variable in the memory. After the entire write operation is completed, if another write operation is not planned in a short time, the MCU should notify CH376 to refresh the file length variable in the internal memory to the USB storage device. There are two methods: write similar to the above steps ÿ and ÿ 0 length data; execute CMD_FILE_CLOSE command and allow to updat

5.36. CMD_DISK_BOC_CMD

This command is used to execute the command of BulkOnly transfer protocol to USB storage device. Before executing this command, the microcontroller must first write the corresponding CBW package to CH376 through the CMD_WR_HOST_DATA command (refer to the BULK_ONLY_CBW structure in the CH376INC.H file). The command is executed successfully. For the operation with return data, the return data can be obtained by the CMD_RD_USB_DATA0 command.

5.37. CMD_DISK_READ

5.38. CMD_DISK_RD_GO

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

The CMD_DISK_READ command requires two sets of parameters: the sector start address represented by 4 bytes and the number of sectors represented by 1 byte. 32-bit double word data represented by a byte. This command requires 5 input data, which are 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. This command can arbitrarily read data from 1 to 255 sectors in the USB storage device.

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

ÿ The MCU issues the CMD_DISK_READ command and specifies the starting LBA of the sector and the number of sectors to start the read operation; ÿ Each sector is decomposed into 8 data blocks of 64 bytes. If all the 8 data blocks of the sector to be read Blocks are all processed, then end the read operation and interrupt the notification to the microcontroller, the interrupt status is USB_INT_SUCCESS, otherwise CH376 reads a 64-byte data block from the USB storage device, and then interrupts to notify the microcontroller to request to read the data block, the interru USB_INT_DISK_READ; ÿ MCU analyzes the interruption status, if it is USB_INT_DISK_READ, then read the data block through the CMD_RD_USB_DATA0 command and continue, if it is USB_INT_SUCCESS, then go to step ÿ, if it is USB_INT_DISK_ERR, explain the operation

If the operation fails, go to step ÿ and try again if necessary;

ÿ The MCU sends the CMD_DISK_RD_GO command to inform CH376 to continue the read operation, and CH376 automatically turns to step ÿ;

ÿ The specified number of read sectors is all read, and the entire read operation ends.

Even if the MCU sends DISK_READ command to read only 1 sector, under normal circumstances, the MCU will receive 9 interrupts, the previous The 8 interrupts are to request the MCU to take away the data, and the last interrupt is to return to the final operating state. If read 4 sectors, then normal Under normal circumstances, the single-chip microcomputer will receive 33 interrupts, and the previous 32 interrupts require the single-chip microcomputer to take away the data. If the read operation fails midway, MCU may receive USB_INT_DISK_ERR status ahead of time, thus ending the read operation ahead of time.

5.39. CMD_DISK_WRITE

5.40. CMD_DISK_WR_GO

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

The CMD_DISK_WRITE command requires two sets of parameters: the sector start address represented by 4 bytes and the number of sectors represented by 1 byte. The starting address of the area is the linear sector number LBA, which is 32-bit double-word data represented by the 4 bytes before the low byte. This command requires 5 Input data, in order the lowest byte of the LBA address, the lower byte of the LBA address, the upper byte of the LBA address, the last byte of the LBA address High byte, number of sectors. This command can arbitrarily write data from 1 to 255 sectors in the USB storage device.

A complete physical sector write operation is usually initiated by a CMD_DISK_WRITE command, and is notified by several interrupts and It consists of several data block writes and several CMD_DISK_WR_GO commands. The operation steps are as follows:

ÿ The MCU issues the CMD_DISK_WRITE command and specifies the starting LBA of the sector and the number of sectors to start the write operation;

ÿ Each sector is decomposed into 8 data blocks of 64 bytes, if all 8 data blocks of the sectors that need to be written are processed Complete, then end the write operation and interrupt the notification to the microcontroller, the interrupt status is USB_INT_SUCCESS, otherwise the interrupt notification to the microcontroller The computer requests to write a data block, and the interrupt status is USB_INT_DISK_WRITE;

ÿ MCU analyzes the interrupt status, if it is USB_INT_DISK_WRITE, then write it by CMD_WR_HOST_DATA command A 64-byte data block and continue, if it is USB_INT_SUCCESS, then go to step ÿ, if it is USB_INT_DISK_ERR, indicating that the operation failed, then go to step ÿ and try again if necessary;

ÿ The MCU sends the CMD_DISK_WR_GO command to inform CH376 to continue writing, and CH376 writes the above data blocks into the USB storage After equipment, automatically turn to step ÿ;

ÿ The specified number of sectors to be written is all written, and the entire write operation ends.

Even if the MCU sends DISK_WRITE command to write only 1 sector, under normal circumstances, the MCU will receive 9 interrupts. The 8 interrupts are to request the microcontroller to provide data, and the last interrupt is to return the final operating status. If writing 4 sectors, then positive Normally, the single-chip microcomputer will receive 33 interrupts, and the previous 32 interrupts require the single-chip microcomputer to provide data. If a write operation is lost in the middle of If it fails, the MCU may receive the USB_INT_DISK_ERR status ahead of time, thus ending the write operation ahead of time.

6. Function description

6.1. MCU communication interface

Three kinds of communication interfaces are supported between CH376 and MCU: 8-bit parallel interface, SPI synchronous serial interface and asynchronous serial interface. in chip When power-on reset, CH376 will sample the status of WR#, RD#, PCS#, A0, RXD, TXD pins, according to the state of these configuration pins Combination selection communication interface, refer to the following table (X in the table means don't care about this bit, 0 means low level, 1 means high level or floating).

WR# pin	RD# pin	PCS# pin	A0 pin	RXD pin	TXD pin	select communication interface
0	0	1	1	1	1	SPI interface
1	1	1	1	1	1	Asynchronous serial port
1	1/X	1/X	X	1	0	8-bit parallel port
other status						CH376 chip does not work, RST pin always outputs high level

The interrupt request output by the INT# pin of the CH376 chip is low-level active by default, and can be connected to the interrupt input pin of the single-chip microcomputer or the common

Machine Translated by Google

Through the input pin, the MCU can know the interrupt request of CH376 through the interrupt mode or query mode. In order to save pins, the MCU can Not connect the INT# pin of CH376, but know the interrupt through other ways.

6.2. Parallel Interface

Parallel port signal lines include: 8-bit bidirectional data bus D7~D0, read strobe input pin RD#, write strobe input pin WR#, chip select Input pin PCS# and address input pin A0. The PCS# of the CH376 chip is driven by the address decoding circuit, which is used when the MCU has multiple Device selection is performed when a peripheral device is selected. Through the passive parallel interface, the CH376 chip can be easily connected to various 8-bit microcontrollers, DSP, MCU system bus, and can coexist with multiple peripheral devices.

For MCU similar to Intel parallel port timing, RD# pin and WR# pin of CH376 chip can be connected to MCU respectively. Read Strobe Output Pin and Write Strobe Output Pin. For MCU similar to Motorola parallel port timing, the RD# pin of CH376 chip should be It should be connected to a low level, and the WR# pin is connected to the read/write direction output pin R/-W of the microcontroller.

The following table is the truth table of parallel port I/O operation (X in the table means don't care about this bit, Z means CH376 three-state prohibition).

PCS# WR# RD# A0 D7-D0					Actual operation of CH376 chip
1	X	X	X	X/Z	CH376 is not selected, do not perform any operation
0	1 1		X	X/Z	Selected but do nothing, do nothing
0	0 1/X 1 input				Write command code to the command port of CH376
0	0 1/X 0 input 0 output				Write data to the data port of CH376
0	1	0			Read data from the data port of CH376
0	1	0	1 output		Read the interface status from the command port of CH376: Bit 7 is the interrupt flag, low effective, equivalent to INT# pin, Bit 4 is a busy flag, high effective, equivalent to the BZ pin of the SPI interface

The CH376 chip occupies two address bits. When the A0 pin is high level, the command port is selected, and new commands can be written or read. Output interface state; when the A0 pin is low level, the data port is selected, and data can be read and written.

The MCU reads and writes the CH376 chip through the 8-bit parallel port. All operations are performed by a command code, several input data and if It consists of output data, some commands do not need input data, and some commands have no output data. The command operation steps are as follows:

ÿ. The MCU writes the command code to the command port when A0=1;

ÿ. If the command has input data, then write the input data sequentially when A0=0, one byte at a time;

ÿ. If the command has output data, then read the output data sequentially when A0=0, one byte at a time;

ÿ, the command is completed, some commands will generate an interrupt notification after the execution is completed, the microcontroller can pause or go to ÿ to continue execution next order.

6.3. SPI serial interface

SPI synchronous serial interface signal lines include: SPI chip select input pin SCS, serial clock input pin SCK, serial data input pin Pin SDI, serial data output pin SDO and interface busy state output pin BZ. Through SPI serial interface, CH376 can use less The connection line of the computer is connected to the SPI serial bus of various single-chip microcomputers, DSPs, and MCUs, or a point-to-point connection over a long distance is carried out.

The SCS pin of the CH376 chip is driven by the SPI chip select output pin or common output pin of the single-chip microcomputer, and the SCK pin is driven by the single-chip microcomputer The SPI clock output pin SCK is driven, the SDI pin is driven by the SPI data output pin SDO or MOSI of the microcontroller, and the SDO pin is Connect to the SPI data input pin SDI or MISO of the microcontroller. For hardware SPI interface, the recommended SPI setting is CPOL=CPHA=0 Or CPOL=CPHA=1, and the data bit order is MSB first. The SPI interface of CH376 also supports the common The I/O pins simulate the SPI interface for communication.

If the INT# pin is not connected, the interrupt can be known by querying the SDO pin, and the method is to let the SDO pin monopolize the MCU It is an input pin, and the SDO pin is set as an interrupt request output when the SCS chip selection is invalid through the CMD_SET_SDO_INT command.

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

The operation steps of SPI are:

ÿ The single chip microcomputer generates the SPI chip selection of the CH376 chip, which is active at low level;

ÿ The MCU sends a byte of data according to the SPI output mode, and the CH376 always selects the first byte received after the SPI chip selection SCS is valid.

The byte is regarded as the command code, and the following bytes are regarded as the data;

ÿ The MCU queries the BZ pin and waits for the SPI interface of CH376 to be idle, or directly delays the TSC time (about 1.5uS);

ÿ If it is a write operation, the single-chip microcomputer sends a byte of data to be written to CH376, and waits for the SPI interface to be idle, and the single-chip microcomputer continues

Continue to send several bytes of data to be written, and CH376 receives them sequentially until the SPI chip selection is prohibited by the MCU;

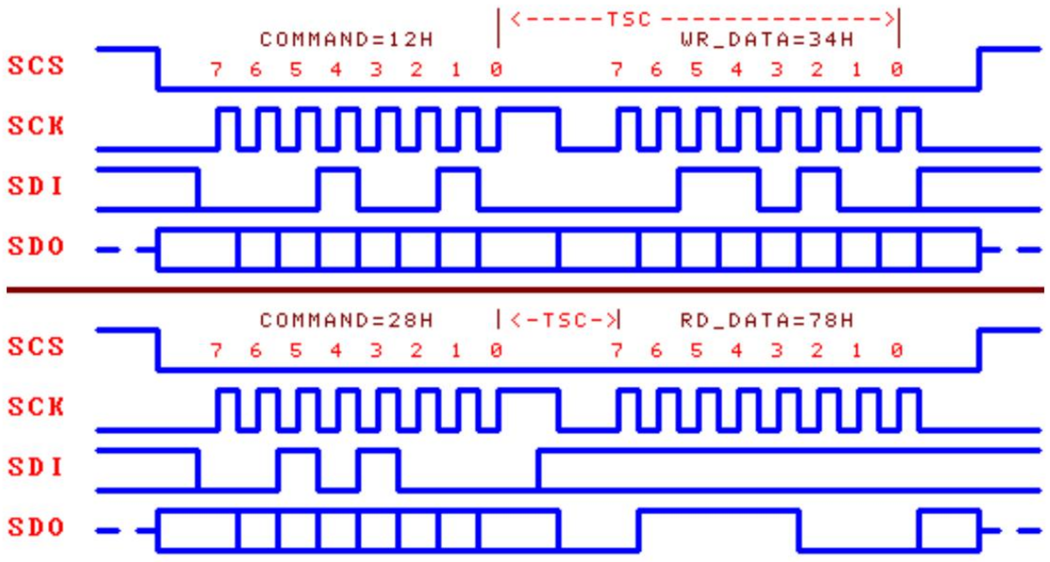
ÿ If it is a read operation, the MCU receives one byte of data from CH376, waits for the SPI interface to be free, and then continues to read from

CH376 receives several bytes of data until the MCU disables SPI chip selection;

ÿ The MCU prohibits the SPI chip selection of the CH376 chip to end the current SPI operation.

The following figure is the logic timing diagram of the SPI interface, the former one is to issue the command 12H and write the data 34H, the latter one is to issue the command 28H

And read data 78H.



6.4. Asynchronous serial port

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

Asynchronous serial port signal lines include: serial data input pin RXD and serial data output pin TXD. Through serial interface, CH376

It can use the least connection line to carry out long-distance point-to-point connection with single-chip microcomputer, DSP, MCU.

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

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

CH376 not only supports the hardware to set the default serial communication baud rate, but also supports the MCU to pass the CMD_SET_BAUDRATE command at any time

Select the appropriate communication baud rate. After each power-on reset, the default serial communication baud rate of CH376 is changed from BZ/D4, SCK/D5, SDI/D6

Refer to the following table for the level combination setting of the three pins (in the table, 0 means low level, 1 means high level or floating).

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 460800
1	0	0	bps 250000 bps
0	1	1	1000000 bps 2000000
0	1	0	bps 921600 bps In
0	0	1	order to distinguish
0	0	0	the command code

and data, CH376 requires the MCU to send two synchronization code bytes (57H and ABH) through the serial port first, and then

Then send the command code, followed by sending or receiving data. CH376 will check the interval between the above two synchronization code bytes, the synchronization code and the command

Interval time between command codes, if the interval time is greater than the serial port input timeout time SER_CMD_TIMEOUT (about 32mS), then CH376

The synchronization code and command packet will be discarded. The serial port command operation steps are as follows:

ÿ The MCU sends the first synchronization code 57H to CH376 through the serial port;

ÿ The MCU sends the second synchronization code 0ABH to CH376;

ÿ The MCU sends a command code to CH376;

ÿ If the command has input data, then send input data to CH376 in turn, one byte at a time;

ÿ If the command has output data, then receive the output data from CH376 in turn, one byte at a time;

ÿ The command is completed, and some commands will generate an interrupt notification after the execution is completed and send an interrupt status code directly through the serial port.

The machine can pause or go to ÿ to continue to execute the next command.

6.5. Other hardware

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

The ACT# pin of CH376 chip is used for status indication output. In the USB device mode of built-in firmware, when the USB device has not been configured Or after canceling configuration, this pin outputs high level; when the USB device configuration is completed, this pin outputs low level. In USB host mode Under this condition, when the USB device is disconnected, the pin outputs high level; when the USB device is connected, the pin outputs low level. On the SD card host In this mode, when the SD card SPI communication is successful, the pin outputs low level. The ACT# pin of CH376 can be externally connected with a current limiting resistor Light-emitting diode LEDs for indicating the relevant status.

The UD+ and UD- pins of the CH376 chip are USB signal lines, and when working in the USB device mode, they should be directly connected to the USB bus When working in the USB host mode, it can be directly connected to the USB device. If a safety resistor or inductor is connected in series for chip safety Or ESD protection devices, then the AC/DC equivalent series resistance should be within 5ÿ.

The CH376 chip has a built-in power-on reset circuit, and generally, no external reset is required. The RSTI pin is used to externally Input asynchronous reset signal; when the RSTI pin is high level, the CH376 chip is reset; when the RSTI pin returns to low level, the CH376 It will continue to delay reset for about 35mS, and then enter the normal working state. For reliable reset during power-up and to reduce external interference Interference, you can connect a capacitor with a capacity of about 0.1uF between the RSTI pin and VCC. RST pin (alias SD_DO pin) It is a high-level active reset status output pin, which can be used to provide a power-on reset signal to an external microcontroller. When CH376 is powered on bit or is externally forced to reset and during the reset delay, the RST pin outputs high level; CH376 reset is completed and the communication interface is initialized After initialization is complete, the RST pin returns to low level.

When CH376 chip works normally, it needs external 12MHz clock signal. The CH376 chip has a built-in crystal oscillator and oscillator In general, the clock signal is generated by the built-in oscillator of CH376 through crystal stable frequency oscillation, and the peripheral circuit only needs to be A nominal 12MHz crystal is connected between the XI and XO pins. If the 12MHz clock signal is directly input from the outside, then it should be It is input from the XI pin, and the XO pin is floating.

CH376 chip supports 3.3V or 5V power supply voltage. When the working voltage is 5V (when the voltage is higher than 4V), the CH376 chip The VCC pin inputs an external 5V power supply, and the V3 pin should be connected with a power decoupling capacitor with a capacity of about 4700pF to 0.02uF. when When the working voltage is 3.3V (when the voltage is lower than 4V), the V3 pin of the CH376 chip should be connected with the VCC pin, and at the same time input the external 3.3V power supply, and the working voltage of other circuits connected with CH376 chip cannot exceed 3.3V.

7. Parameters

7.1. Absolute maximum value (critical or exceeding the absolute maximum value may cause the chip to work abnormally or even be damaged)

name	Parameter Description		min max unit		
FACING	Ambient temperature at work	VCC=5V	-40	85	ÿ
		VCC=V3=3.3V	-40	85	
		VCC=V3=3V	-40	70	
TS	Ambient temperature during		-55	125	ÿ
VCC	storage Power supply voltage (VCC connected to power		-0.5	6.0	IN
SAW	supply, GND connected to ground) Voltage on input or output pins		-0.5	VCC+0.5	IN

7.2. Electrical parameters (test conditions: TA=25ÿ, VCC=5V, excluding the pins connected to the USB bus)

(If the power 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 supply voltage	V3 not connected to VCC	4.3	5	5.3	IN
	V3 is connected to VCC, V3=VCC	3.0	3.3	3.6	
Total Supply Current During ICC Operation	VCC=5V		12	30	mA
	VCC=3.3V		6	15	
ISLP	Supply Current in Low Power States VCC=5V		0.15		mA
	I/O pin floating/internal pull-up VCC=3.3V low-level input		0.05		
WILL	voltage high-level input	-0.5		0.7	IN
HIV	voltage low-level output	2.0		VCC+0.5	IN
VOL	voltage (4mA sink current) high-level output voltage			0.5	IN
VOH	(4mA output current) VCC-0.5 IUP built-in The input current of the input				IN
terminal of the pull-up resistor The input current of the input terminal		30	80	160	uA
IUP2	of the open-drain output pin ACT# and SD_CS Built-in pull-up resistor	100	230	500	uA
IDN	The input current of the input terminal of the built-in pull-down resistor	-30 The	-80	-200	uA
VR	voltage threshold of power-on reset	2.4	2.7	2.9	IN

Note: The low-level sink current of ACT# pin and SD_CS pin is 4mA, and the high-level output current is 200uA.

During CH376 chip reset, INT# pin and TXD pin can only provide 80uA high-level output current.

7.3. Timing parameters (test conditions: TA=25ÿ, VCC=5V or VCC=3.3V, refer to the attached picture)

name	Parameter Description	Min Typ Max Unit			
FCLK	USB host mode Input clock frequency of XI pin 11.995 Reset time of internal		12.00	12.005	MHz
TPR	power supply Power-on external reset	25	35	40	mS
THREE	input Valid signal width External reset input	100			nS
TRD	Reset delay after external reset input Wake-	25	32	35	mS
TWACK	up time from low power state Execution time	3	7	12	mS
THE1	of CMD_RESET_ALL command		32	35	mS
TE2	CMD_SET_USB_MODE command Execution time of		4	10	uS
TE3	TEST_CONNECT or SET_ENDP? Command execution time of		2	3	uS
of	CMD_SET_BAUDRATE command TE4 Execution time	200	1000	2000	uS
TE0	other commands Interval time between command code		1.5	2	uS
TSX	and command code Interval time between	1.5			uS
TSC	command code and data Interval time between	1.5			uS
TSD	data and data	0.6			uS
TINT	receives the GET_STATUS command to the INT# pin to cancel the interrupt		1.5	2	uS

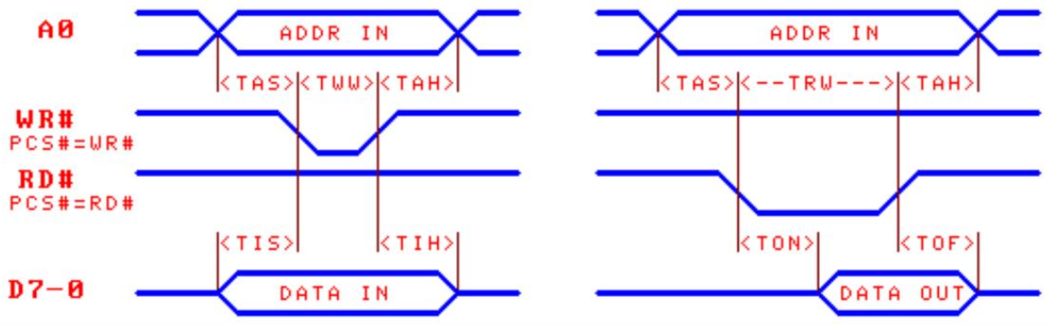
7.4. Parallel port timing parameters (test conditions: TA=25°C, VCC=5V, parameters in brackets VCC=3.3V, refer to the attached picture)

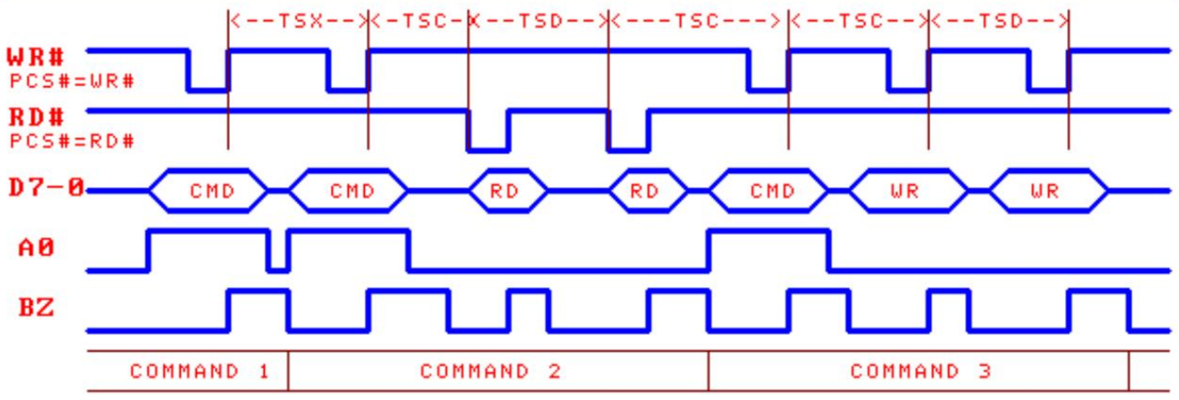
(RD means that the RD# signal is valid and the PCS# signal is valid, RD#=PCS#=0 executes the read operation)

(WR means that the WR# signal is valid and the PCS# signal is valid, and WR#=PCS#=0 executes the write operation)

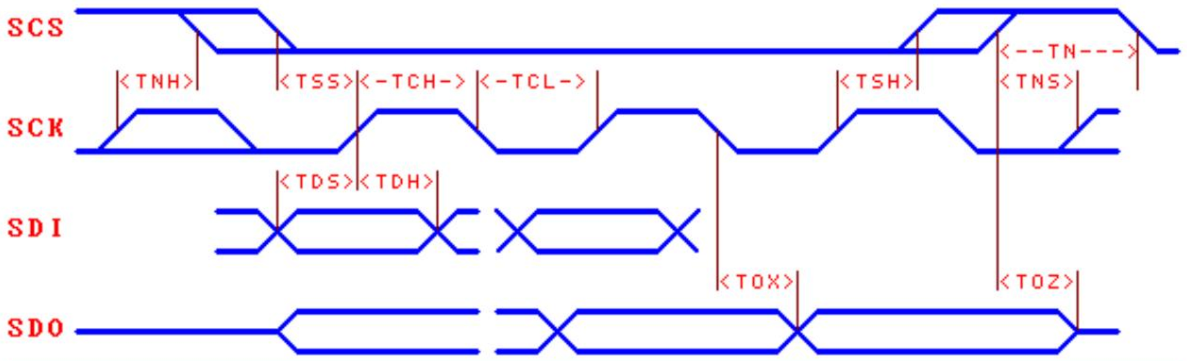
Name Minimum Typical Value Maximum Unit	Parameter				
TWW	Description The width of the effective write	30 (45) 40			nS
TRW	strobe WR and the effective width of the read strobe RD	(60)			nS

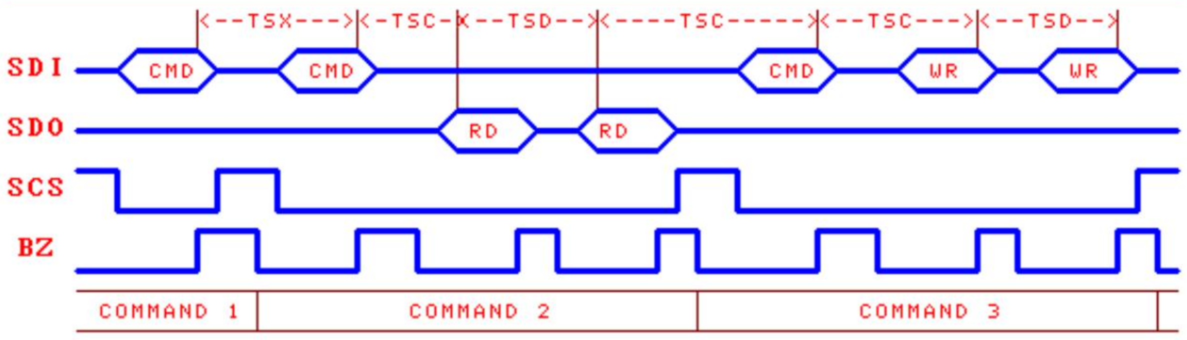
THAT	Address input setup time before RD or WR	4 (6) 4			nS
broken	Address input hold time after RD or WR	(6) 0 4			nS
TIS Data	input setup time before write strobe WR				nS
TIH Data	in hold time after write strobe WR Read strobe RD active to data	(6)			nS
TON	out active Read strobe RD inactive to data out	2	12	18 (30) 24	nS
TOF	inactive	3	16	(40)	nS





7.5. SPI timing parameters (test conditions: TA=25°C, VCC=5V, the parameters in brackets VCC=3.3V, refer to the attached picture)

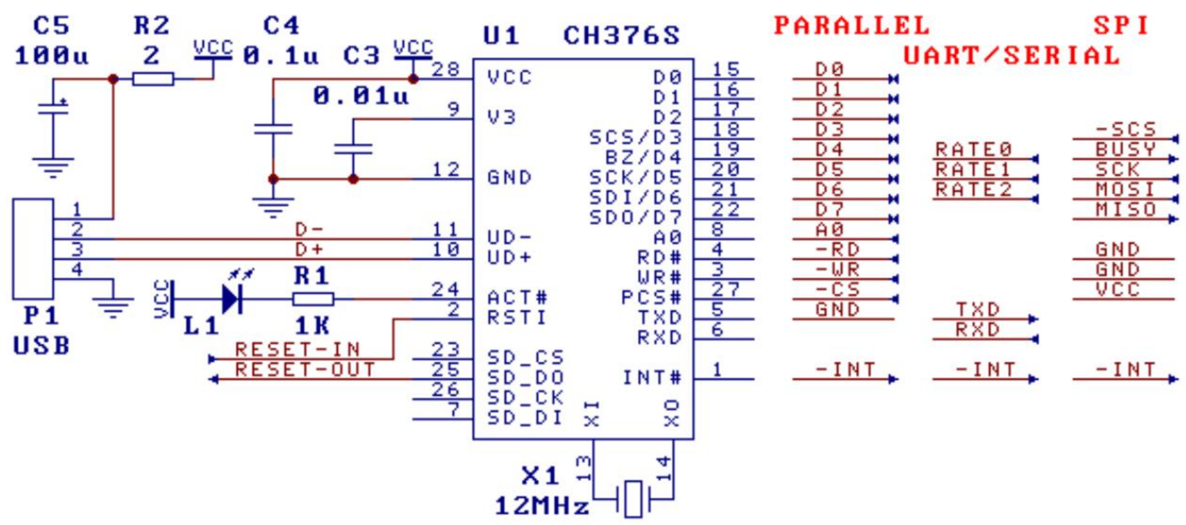




name	Parameter Description	Min Typ Max Unit			
TSS	SCS valid setup time before SCK rising edge 20 (30)				nS
TSH	SCS valid hold time after SCK rising edge 20 (30)				nS
TNS	SCS inactive setup time before SCK rising edge 20 (30)				nS
THINKING	SCS inactive hold time after SCK rising edge 20 (30)				nS
TN	SCS invalid time (SPI operation interval time) 80 (120)				nS
TCH	SCK clock high time 14 (18)				nS
TCL	SCK clock low time 18 (24)				nS
TDS	SDI input setup time before SCK rising edge6 (8)				nS
TDH	Hold time of SDI input after SCK rising edge	2			nS
TOX	SCK falling edge to SDO output change	3	8 (12)	12 (18) 18	nS
DUST	SCS invalid to SDO output invalid	4		(25)	nS

8. Application

8.1. U disk application, 5V power supply (below)



This is the application circuit of CH376 chip operating U disk under 5V power supply voltage.

If CH376 needs to be configured as 8-bit parallel port communication mode PARALLEL, then TXD should be connected to GND, and other pins should be left floating.

If CH376 needs to be configured as SPI serial communication mode SPI, then RD# and WR# should be connected to GND, and other pins should be left floating.

If it is necessary to configure CH376 as an asynchronous serial port communication mode UART/SERIAL, then all pins should be suspended, and the default serial port The communication baud rate is set by the three pins SDI/D6, SCK/D5 and BZ/D4. If it is necessary to dynamically modify the communication baud rate of the CH376 serial port, Then it is recommended to control the RSTI pin of CH376 by the I/O pin of the single chip microcomputer, so as to reset CH376 to restore to the default communication mode when necessary. Signal baud rate. Since the RSTI pin has a built-in pull-down resistor, it may need to be driven by the quasi-bidirectional I/O pin of MCS51 and other microcontrollers A pull-up resistor with a resistance of about a few Kÿ should be added.

Since the INT# pin and TXD pin can only provide a weak high-level output current during the CH376 reset period, the When connected, in order to avoid INT# or TXD being disturbed during the reset period of CH376 and causing the misoperation of the single chip microcomputer, you can Add a pull-up resistor with a resistance value of 2Kÿ~5Kÿ on the pin or TXD pin to maintain a relatively stable high level. After CH376 chip reset After completion, INT# pin and TXD pin will be able to provide 4mA high-level output current or 4mA low-level sink current.

In order to save pins, the MCU can not be connected to the INT# pin of the CH376 chip, and the method of getting the interrupt notification is as follows:

ÿ In the 8-bit parallel port mode, the interface status can be obtained by querying the status port (command port) of CH376, bit 7 is the interrupt flag PARA_STATE_INTB, low effective, equivalent to querying the INT# pin, when bit 7 is 0, it means there is an interrupt request;

ÿ In the SPI interface mode, the interrupt is known by querying the SDO pin (after power-on or reset, the CMD_SET_SDO_INT command must be passed first) Set the SDO pin as an interrupt request output when the SCS chip selection is invalid), and when SDO is low, it indicates that there is an interrupt request;

ÿ In the serial port mode, while CH376 generates an interrupt notification (INT# becomes low level), it will directly issue an interrupt status through the serial port.

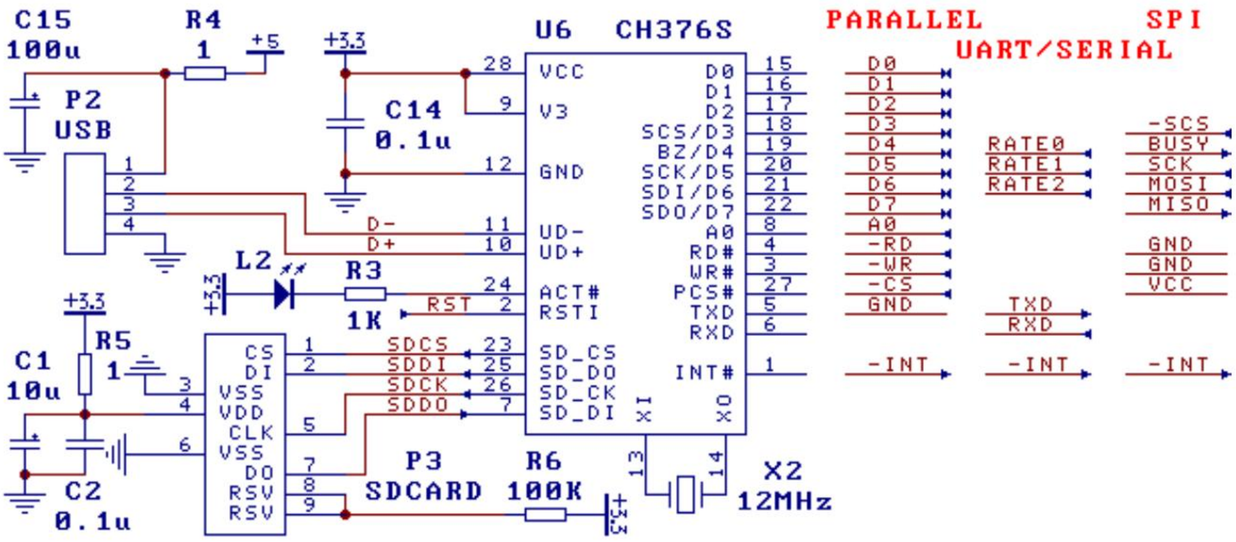
status code, the MCU receives an interrupt status code indicating that there is an interrupt request.

R2 is used to limit the current provided to the external USB device as a USB host, if necessary, it can be connected in series with a fast power supply with current limiting effect sub-switch, the USB supply voltage must be 5V.

Capacitor C3 is used for CH376 internal power node decoupling, C3 is a monolithic or high-frequency ceramic capacitor with a capacity of 4700pF to 0.02ÿF. Capacitors C4 and C5 are used for external power decoupling, and C4 is a monolithic or high-frequency ceramic capacitor with a capacity of 0.1ÿF. Crystal X1 is used in the clock oscillation circuit. The USB-HOST mode requires a relatively accurate clock frequency. The frequency of crystal X1 is 12MHz±0.4‰.

When designing the printed circuit board PCB, it is necessary to pay attention to: the decoupling capacitors C3 and C4 should be as close as possible to the connecting pins of CH376; make the D+ and D- signal lines close to parallel wiring, and provide ground wires or copper pours on both sides as much as possible to reduce the noise from the outside world. Signal interference; shorten the length of the XI and XO pin-related signal lines as much as possible. In order to reduce the interference of high-frequency clocks to the outside world, you can surround the ground wire or pour copper around the relevant components.

8.2. SD card and U disk application, 3.3V power supply (below)



This is the application circuit for CH376 chip operating U disk and SD card under 3.3V or 3V power supply voltage.

P3 is a simplified SD card slot, and the SD card plug status pin can be directly connected to the I/O or interrupt input pin of the microcontroller. The configuration of the communication interface is the same as that of the 5V voltage application, refer to Section 8.1.

R4 is used to limit the current provided to the external USB device as a USB host, if necessary, it can be connected in series with a fast power supply with current limiting effect sub-switch, the USB supply voltage must be 5V.

The power supply voltage of CH376 is 3.3V. In the figure, V3 pin and VCC pin are short-circuited to input 3.3V voltage together. Capacitors C14 and C15 are used for external power decoupling, and C14 is a monolithic or high-frequency ceramic capacitor with a capacity of 0.1ÿF.

8.3. Application Basics

U disk (or SD card, the same below) provides several physical sectors for data storage, and the size of each sector is usually 512 bytes. Since the computer usually organizes the physical sectors in the U disk as a FAT file system, in order to facilitate the exchange of data between the MCU and the computer through the U disk or SD card, the MCU should also access the U disk in the form of files under the FAT specification. data.

There can be several files in a USB flash drive, and each file is a collection of data, distinguished and identified by file name. The storage of actual file data may not be continuous, but multiple blocks (that is, allocation units or clusters) linked by a set of "pointers", so that the file length can be increased at any time to accommodate more data as needed. The directory (folder) is for the convenience of classified management. The administrator can manually designate multiple files to be archived together. For example, the files in 2004 are grouped into one directory (folder).

In the FAT file system, disk capacity is allocated based on clusters, and the size of clusters is always a multiple of sectors, so the space occupied by files is always a multiple of clusters and a multiple of sectors. Although the space occupied by the file is a multiple of clusters or sectors, in practical applications, the length of valid data stored in the file is not necessarily a multiple of sectors, so the FAT file system specifically records in the file directory information FAT_DIR_INFO The length of valid data in the current file, that is, the number of bytes of valid data, is commonly referred to as the file length, and the file length is always less than or equal to the space occupied by the file. After writing data to the file, if the original data is overwritten, the file

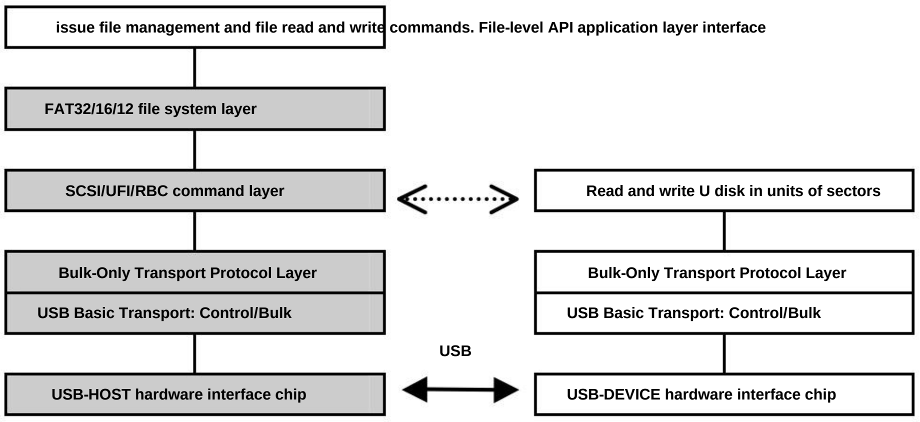
length may not change. When the original file length is exceeded

After that, it becomes append data, then the file length should change (increase). If the file length in the file directory information is not modified after adding data to the file, the FAT file system will consider the data exceeding the file length invalid. Under normal circumstances, the computer cannot read the data exceeding the file length, although the data actually exists . If the amount of data is small or the data is

discontinuous, the file length in the file directory information can be updated immediately after each additional data, but if the amount of data is large and the data needs to be written continuously, immediately updating the file directory information will reduce efficiency, and Frequently modifying the file directory information will also shorten the service life of the flash memory in the U disk (because the flash memory can only be erased and written for a limited number of times), so in this case, you should update the file directory information after writing multiple sets of data continuously or wait until the file is closed and then update the file length, the CMD_FILE_CLOSE command can refresh the file length in the memory to the file directory information of the U disk file.

Although CH376 supports a maximum single file of 1GB, in order to improve efficiency, it is recommended that the length of a single file should not exceed 100MB. Usually it is quite normal in the range of a few KB to several MB. When there is a lot of data, it can be divided into multiple directories and stored in multiple files.

In general, the single-chip microcomputer or embedded system needs to realize the 4 levels on the left side of the figure below to process the file system of the USB flash drive, and the right side is the internal structure level of the USB flash drive. Since CH376 is not only a general-purpose USB-HOST hardware interface chip, but also has built-in relevant USB underlying transmission firmware programs, Bulk-Only protocol transmission firmware programs, and FAT file system management firmware programs, including the four levels on the left of the figure below (Marked in gray), so the actual microcontroller program only needs to


issue file management and file read and write commands. File-level API application layer interface
FAT32/16/12 file system layer
SCSI/UFI/RBC command layer
Read and write U disk in units of sectors
Bulk-Only Transport Protocol Layer
USB Basic Transport: Control/Bulk
Bulk-Only Transport Protocol Layer
USB Basic Transport: Control/Bulk
USB
USB-HOST hardware interface chip
USB-DEVICE hardware interface chip

8.4. Quick Application Reference Steps

Please refer to the example program to call the subroutine that has packaged multiple commands. The following steps use the original command codes for reference only.

8.4.1. Initialization, the necessary steps before any file operation ÿ CMD_SET_USB_MODE

command, enter the USB-HOST working mode or SD card host working mode (mode 3) ÿ Wait for the U disk or SD card to be connected, the U disk can be used by CH376 Automatically detect and generate an interrupt notification, or send an interrupt notification to CH376 by the MCU

Issue the CMD_DISK_CONNECT command to query regularly, the SD card must be detected by the single chip microcomputer

ÿ CMD_DISK_MOUNT command, initialize the U disk or SD card, and test whether the disk is ready. If it fails, you can retry up to 5 times. ÿ

8.4.2. Read files sequentially ÿ

CMD_SET_FILE_NAME command + CMD_FILE_OPEN command, open the file ÿ Multiple

CMD_BYTE_READ command + CMD_RD_USB_DATA0 command + CMD_BYTE_RD_GO command, read data ÿ CMD_FILE_CLOSE command, close the file, optional operation

8.4.3. Rewrite files sequentially (overwrite the original data, and change to additional data after the original file length is

exceeded) ÿ CMD_SET_FILE_NAME command + CMD_FILE_OPEN command, open the file ÿ Multiple

CMD_BYTE_WRITE command + CMD_WR_REQ_DATA command + CMD_BYTE_WR_GO command, write data ÿ CMD_FILE_CLOSE command, The parameter is 1, close the file and allow automatic update of the file length

8.4.4. Add data to the existing file ÿ

CMD_SET_FILE_NAME command + CMD_FILE_OPEN command, open the file ÿ CMD_BYTE_LOCATE

command, the parameter is 0FFFFFFFFH, move the file pointer to the end of the file ÿ Multiple CMD_BYTE_WRITE command + CMD_WR_REQ_DATA command + CMD_BYTE_WR_GO command, write data ÿ CMD_FILE_CLOSE command, parameter is 1, close the file and allow automatic update of file length

8.4.5. Create a new file and write data ÿ
CMD_SET_FILE_NAME command + CMD_FILE_CREATE command, create a new file ÿ Multiple
CMD_BYTE_WRITE command + CMD_WR_REQ_DATA command + CMD_BYTE_WR_GO command, write data ÿ CMD_FILE_CLOSE
command, the parameter is 1, close the file and allow the file length to be automatically updated

8.4.6. Read the file first and then rewrite the
file ÿ CMD_SET_FILE_NAME command + CMD_FILE_OPEN command, open the file ÿ Multiple
CMD_BYTE_READ command + CMD_RD_USB_DATA0 command + CMD_BYTE_RD_GO command, read data ÿ CMD_BYTE_LOCATE
command, the parameter is 0, move the file pointer to the file head ÿ Multiple CMD_BYTE_WRITE
command + CMD_WR_REQ_DATA command + CMD_BYTE_WR_GO command, write data ÿ CMD_FILE_CLOSE command, the parameter
is 1, close the file and allow automatic update of file length

8.4.7. If the file already exists, add data, if the file does not exist, create a new file and then write data ÿ CMD_SET_FILE_NAME
command + CMD_FILE_OPEN command, open the file, if it returns ERR_MISS_FILE indicating that the file does not exist, then go to step ÿ

ÿ CMD_BYTE_LOCATE command, the parameter is 0FFFFFFFFH, move the file pointer to the end of the file, and then go to step ÿ ÿ
CMD_FILE_CREATE command, create a new file ÿ Multiple
CMD_BYTE_WRITE command + CMD_WR_REQ_DATA command + CMD_BYTE_WR_GO command, write data ÿ CMD_FILE_CLOSE
command, the parameter is 1, close the file and allow automatic file length updates

8.4.8. To modify file name, file date/time, file length and other file directory information, please refer to the description in the EXAM10 example ÿ
CMD_SET_FILE_NAME command + CMD_FILE_OPEN command, open the file ÿ Use the
CMD_RD_USB_DATA0 command to read the original file directory information ÿ CMD_DIR_INFO_READ
command, the parameter is 0FFH, read the file directory information into the
memory ÿ Use the CMD_WR_OFS_DATA command to write the new file directory information ÿ
CMD_DIR_INFO_SAVE command, save the file directory information ÿ
CMD_FILE_CLOSE command, The parameter is 0, close the file and disable
automatic update of file length, optional operation

8.4.9. To create a subdirectory (folder), please refer to the description in the EXAM9 example ÿ
CMD_SET_FILE_NAME command + CMD_DIR_CREATE command, create a new subdirectory (folder) ÿ
CMD_FILE_CLOSE command, the parameter is 0, close the file and prohibit automatic file length update

8.4.10. To deal with lowercase filenames and long filenames, please refer to the instructions in the EXAM11 example

8.4.11. Search and enumerate file names, enumerate all files in the whole disk, please refer to the instructions in the EXAM13 example

8.4.12. Master-slave switching, communication with computer, reading and writing U disk or SD card files, please refer to the instructions in the EXAM0 example

8.5. USB Device Application

Please refer to the manual CH372DS1.PDF of the CH372 chip and its application documents.