

S1C17801

USB Mass Storage

Application Note

When using the commands, follow the instructions of NOTICE_Application Notes Sample Programs.pdf included in the downloaded compressed file.

NOTICE

No part of this material may be reproduced or duplicated in any form or by any means without the written permission of Seiko Epson. Seiko Epson reserves the right to make changes to this material without notice. Seiko Epson does not assume any liability of any kind arising out of any inaccuracies contained in this material or due to its application or use in any product or circuit and, further, there is no representation that this material is applicable to products requiring high level reliability, such as, medical products. Moreover, no license to any intellectual property rights is granted by implication or otherwise, and there is no representation or warranty that anything made in accordance with this material will be free from any patent or copyright infringement of a third party. This material or portions thereof may contain technology or the subject relating to strategic products under the control of the Foreign Exchange and Foreign Trade Law of Japan and may require an export license from the Ministry of Economy, Trade and Industry or other approval from another government agency.

All other product names mentioned herein are trademarks and/or registered trademarks of their respective companies.

OVERVIEW

This document is a reference to use the USB (mass storage) function of S1C17801.

OPERATING ENVIRONMENT

- S5U1C17801T1100 (Hereafter SVT17801: Software eValuation Tool for S1C17801)
SVT17801 CPU board and SVT17 ICD board sets
- USB miniB cable
- PC
 - The GNU17 development tool must be installed.
 - The USB driver for the SVT17 ICD board must be installed.
- The latest version of FLS17801 (file name: fls17801.elf) must be installed.
This file is indispensable for writing the program to the built-in Flash memory.

Table of Contents

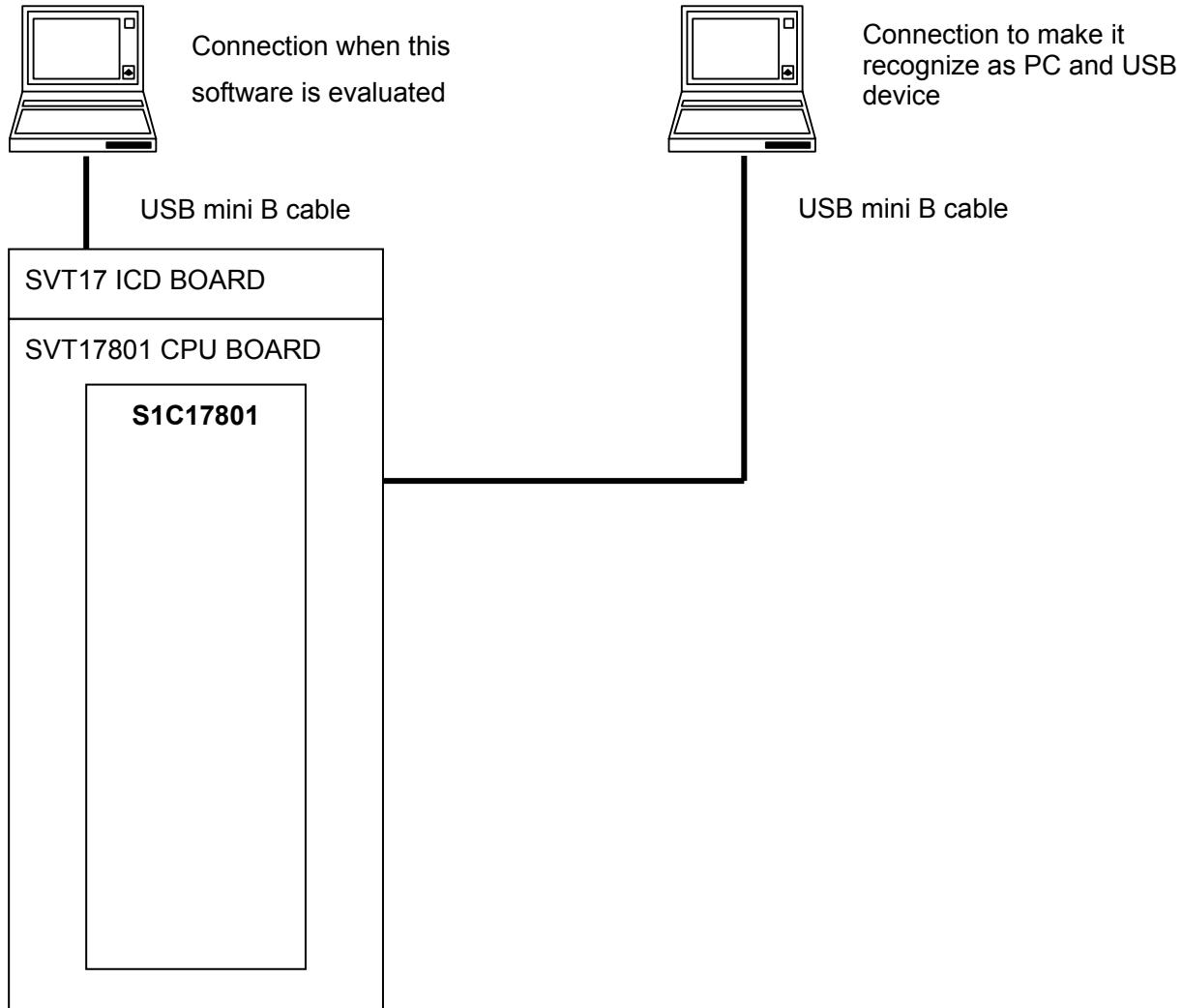
1. SPECIFICATIONS	1
2. FUNCTIONAL DESCRIPTION	2
3. SOFTWARE DESCRIPTION	4
3.1 File Configuration	4
3.2 Firmware Configuration	5
3.3 Memory Map	6
3.4 Module Description.....	7
3.5 Global Variables	7
3.6 Structures	8
3.7 Descriptors	9
3.8 Operation Procedure	10
3.9 Outline of sample program operations	11
3.10 OPERATION FLOWCHARTS	14
3.10.1 Read command operation flow	14
3.10.2 Write command operation flow	15
3.11 Flowchart	16
3.12 Description of API Specifications.....	17
3.13 Detailed Description of USB Drivers	18
3.13.1 Low-level USB driver API details	18
3.13.2 High-level USB driver API details	45
3.14 HEADER DEFINITIONS	52
3.15 Compile Options	53
REVISION HISTORY	54

1. SPECIFICATIONS

When this sample program is used, the host recognizes the Multi-Media Card (MMC) that is inserted into the SD/MMC card slot as a mass-storage class USB device.

The following shows the system connection.

PC (having the GNU17 development tool)



2. FUNCTIONAL DESCRIPTION

2. FUNCTIONAL DESCRIPTION

The following provides the functional overview of the USB driver.

The USB mass-storage (MMC version) sample software uses this USB driver to provide the following functions. The USB driver functions are detailed in the “USB Driver Detailed Description” section.

■ Control and setting of operation clock

The 48MHz clock must be entered into the OSC3.

The CMU (0x4900: D[7]=0 OSC3 disable during SLEEP/D[1]=0 OSC3 Oscillator on/off) must be set. Also, the CMU (0x4906: D[5:4]=1 USB SAPB I/F clock control/USB-IP 48MHz clock control) must be enabled.

The USB driver must be set by the host application before it is accessed.

■ Initializing MMC

Initializes the USB IP.

Also, the host application must set an access waiting to the USB-IP using the CMU driver.

■ Snooze Control

The Snooze Control function controls the CMU driver and sets the system in the power-saving mode. If the USB-IP is used, the Snooze Control function controls the CMU driver and enables its operation using this driver.

■ Auto Address Control

Sets the Auto Address Return function.

■ Descriptor Control

Initializes the descriptor to enable the automatic descriptor return function, sets the descriptor data, and enables descriptor return control according to the request.

■ Endpoint Control

Initializes the system to use EPa to EPd, and executes data transmission processing to/from EPa to EPd.

■ Control-Pipe Control

Initializes the system to use the EP0, and executes IN/OUT control, EP0 control, and data transmission processing to/from the EP0.

■ USB Control

Controls the functions according to the changes on the USB bus. They are Bus Reset, Suspend, Resume, VBUS connection, VBUS disconnection, and others.

2. FUNCTIONAL DESCRIPTION

- FIFO Control

Controls the descriptor and endpoint assignment for FIFO operation, and controls data writing/reading to/from the FIFO storage.

- Test Mode Control

Controls the Test Mode operation according to a request from the USB authentication tool and others.

- Interrupt Control

Identifies an interrupt request from USB-IP, and controls to clear the interrupt cause and others.

- Pull-Up Control

Disconnects the pull-up resistors and controls their connection.

- Standard request processing

Automatically executes the standard request processing that is required for the USB device.

3. SOFTWARE DESCRIPTION

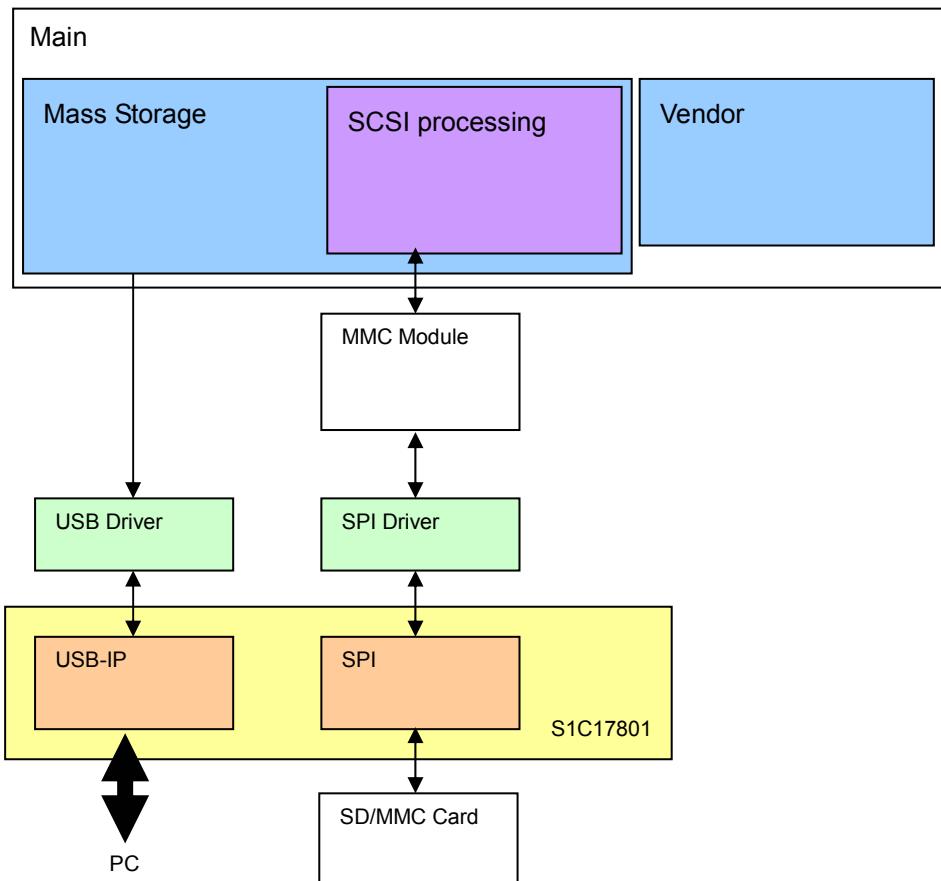
3. SOFTWARE DESCRIPTION

3.1 File Configuration

File name	Function
boot.c	Startup module
main.c	Main function
inthdlr.c	Interrupt function
vector.c	Vector table settings
buffer.c	Work buffer file
descriptor.c	Sets the descriptor information.
descriptor.h	Descriptor information definition
massstorage.c	Mass-storage class processing
massstorage.h	Mass-storage class definition
scsicommand.c	SCSI command processing
scsicommand.h	SCSI command definition
vendorclass.c	VendorClass processing
vendorclass.h	VendorClass definition
header/	System header (Register definition and vector table definition)
usb_drv/ usb_api.h	USB driver API definition
usb_drv/ usb_drv.c	USB driver API
usb_drv/ usb_usb.c	USB slave driver
usb_drv/ usb_usb.h	USB slave driver definition
usb_drv/ usb_control.c	USB Control Transfer processing
usb_drv/ usb_control.h	USB Control Transfer definitions
usb_drv/ usb_request.c	USB standard request processing
usb_drv/ usb_request.h	USB standard request definition
usb_drv/ usb_bulk.c	USB Bulk Transfer processing
usb_drv/ usb_bulk.h	USB Bulk Transfer definition
cmu_drv/	CMU driver group folder
flashc_drv/	FLASHC driver group folder
gpio_drv/	GPIO driver group folder
itc_drv/	ITC driver group folder
sramc_drv/	SRAMC driver group folder
clg_drv/	CLG driver
spi_drv/	SPI driver
mmc_drv/	MMC driver
usb_ms mmc gnu17IDE.lds	Linker script file
usb_ms mmc gnu17IDE.cmd	GDB command file
usb_ms mmc gnu17IDE.par	Parameter configuration file
usb_ms mmc gnu17IDE.mak	Make file
.cdtproject	Project file
.gnu17project	Project file
.project	Project file
GDB17 Launch for usb_ms mmc.launch	Project startup file

3.2 Firmware Configuration

The following shows the simple and standard configuration of USB MS application.



The Mass Storage receives CBW packets from the USB, receives data, and sends the CSW.

The SCSI Process analyzes the commands included in the CBW packets, issues commands to the connected devices, and sends or receives data.

The Vendor processes the vendor requests.

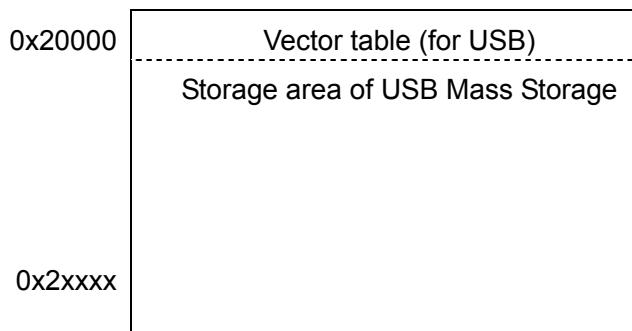
3. SOFTWARE DESCRIPTION

3.3 Memory Map

The memory map shows the memory that is used when the USB Mass Storage sample program operates.

- Internal Flash memory

Contains the vector table or its main part of the USB Mass Storage.

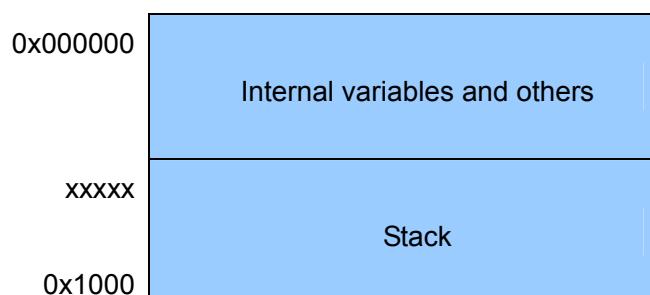


- External Flash memory

Not used by the USB Mass Storage.

- IRAM

Used to store the USB Mass Storage variables, or as the stack area.

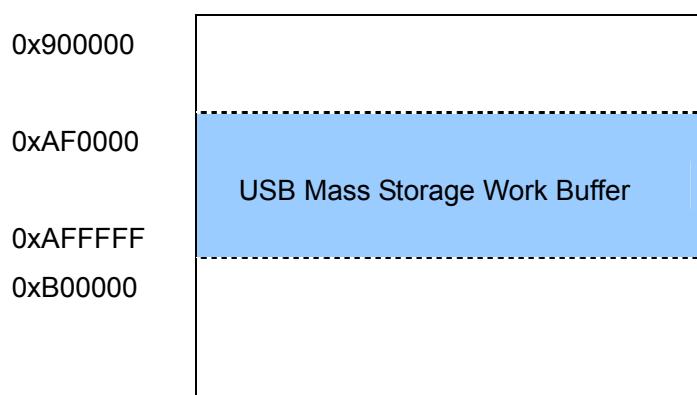


- IVRAM

Not used by the USB Mass Storage.

- External SRAM

Used as the work area during data transmission to/from the host of USB Mass Storage.



3.4 Module Description

File name : main.c

Function name	Function
Main	Initializes the system to use the USB functions. The Main function monitors the USB bus and provides the storage device functions.
ItcInitialize	Initializes the interrupt controller.
ClgInitialize	Sets the clock for the SPI.
PortInitialize	Sets the port for the SPI.
UsbInitialize	Initializes the USB macros.
SnoozeUsb	Snoozes the USB macros.
ActiveUsb	Activates the USB macros.
USBIntEnableFunc	Enables a USB interrupt.
USBIntDisableFunc	Disables a USB interrupt.

For the CMU driver functions, refer to the “Detailed description of CMU driver” chapter.

For the FLASHC driver functions, refer to the “Detailed description of FLASHC Driver” chapter.

For the GPIO driver functions, refer to the “Detailed description of GPIO driver” chapter.

For ITC driver functions, refer to the “Detailed description of ITC driver” chapter.

For SRAMC driver functions, refer to “Detailed descriptions of SRAMC driver” chapter.

For the CLG driver function, refer to the chapter of “Detailed Explanation of CLG Driver.”

For the MMC/SPI driver functions, refer to the “Detailed description of MMC/SPI Driver” chapter.

3.5 Global Variables

The following shows the global variables used in the sample program.

Variable name	Type	Function
—	—	—

3. SOFTWARE DESCRIPTION

3.6 Structures

The following describes the structure used in the sample program.

Definition name		
T_USB_CFG		
Members		
descriptorData	Unsigned char *	Descriptor data pointer
descriptorSize	unsigned short	Descriptor size
negotiationMode	unsigned char	Auto negotiation mode
addressMode	unsigned char	Auto address reply mode
managementMode	unsigned char	BUS processing mode of USB
activeFunc	Void (*) (void)	Active clock function
snoozeFunc	Void (*) (void)	Snooze clock function
resetFunc	Void (*) (void)	Reset process function
remoteFunc	int (*) (void)	RemoteWakeup Switch function
classFunc	int (*) (T_USB_REQUEST_PACKET*)	Class request function
vendorFunc	int (*) (T_USB_REQUEST_PACKET*)	Vendor request function
intEnableFunc	Void (*) (void)	USB Interrupt enable function
intDisableFunc	Void (*) (void)	USB Interrupt disable function
Remarks		
USB driver initialization structures		

Definition name		
T_USB_REQUEST_PACKET		
Members		
requestType	unsigned char	Request type
request	unsigned char	Request number
value	unsigned short	Value
index	unsigned short	Index
length	unsigned short	Length
Remarks		
Request packet structures		

3.7 Descriptors

The following lists the USB Mass Storage descriptors.

- Device Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	12h	Descriptor size (18 bytes)
1	bDescriptorType	1	01h	Descriptor type (Device)
2	bcdUSB	2	0110h	Release number of USB standard (USB1.1)
4	bDeviceClass	1	00h	Device class code
5	bDeviceSubClass	1	00h	Device subclass code
6	bDeviceProtocol	1	00h	Device protocol code
7	bMaxPacketSize0	1	40h	Maximum packet size (64 bytes) at endpoint 0
8	idVendor	2	04B8h	Vendor ID (SEIKO EPSON)
10	idProduct	2	0506h	Product ID
12	bcdDevice	2	0100h	Device release number
14	iManufacturer	1	01h	String descriptor index that identifies the manufacturer
15	iProduct	1	02h	String descriptor index that identifies the product name
16	iSerialNumber	1	03h	String descriptor index that identifies the serial number
17	bNumConfigurations	1	01h	Number of devices that can be configured

- Configuration Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	09h	Descriptor size (9 bytes)
1	bDescriptorType	1	02h	Descriptor type (Configuration)
2	wTotalLength	2	0020h	Total length of return data (32 bytes)
4	bNumInterfaces	1	01h	Number of supported interfaces
5	bConfigurationValue	1	01h	Selection value of this configuration
6	iConfiguration	1	00h	String descriptor index
7	bmAttributes	1	C0h	Configuration attributes
8	MaxPower	1	F0h	Maximum power consumption

- Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	09h	Descriptor size (9 bytes)
1	bDescriptorType	1	04h	Descriptor type (Interface)
2	bInterfaceNumber	1	00h	Interface number
3	bAlternateSetting	1	00h	Selection value of alternate settings
4	bNumEndpoints	1	02h	Number of endpoints used for this interface
5	bInterfaceClass	1	08h	Interface class code
6	bInterfaceSubClass	1	06h	Interface subclass code
7	bInterfaceProtocol	1	50h	Interface protocol code
8	iInterface	1	00h	String descriptor index that identifies the interface

3. SOFTWARE DESCRIPTION

- Endpoint Descriptor1

Offset	Field	Size	Value	Description
0	bLength	1	07h	Descriptor size (7 bytes)
1	bDescriptorType	1	05h	Descriptor type (Endpoint)
2	bEndpointAddress	1	81h	Direction IN endpoint number: 01
3	bmAttributes	1	02h	Bulk
4	wMaxPacketSize	2	0040h	Max Packet Size
5	blInterval	1	FFh	Polling interval

- Endpoint Descriptor2

Offset	Field	Size	Value	Description
0	bLength	1	07h	Descriptor size (7 bytes)
1	bDescriptorType	1	05h	Descriptor type (Endpoint)
2	bEndpointAddress	1	02h	Direction OUT endpoint number: 02
3	bmAttributes	1	02h	Bulk
4	wMaxPacketSize	2	0040h	Max Packet Size
5	blInterval	1	FFh	Polling interval

- String Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	??h	Descriptor size (? bytes)
1	bDescriptorType	1	03h	Descriptor type (String)
2	bString	?	?	String coded in UNICODE system

Enter a value or a character string within the length that has been set by the customer.

- * String1: “EPSON”
- * String2: “EPSON C17801/501 Sample”
- * String3: “Rev.0.01”

3.8 Operation Procedure

Import the project

- (1) Start the IDE and import the “usb_ms_mmc” project.

* For the import procedure, refer to Section 3 “Software Development Procedure” of the S5U1C17001C Manual.

Build

- (1) Build the “usb_ms_mmc” project using the IDE.

Cabling and power-on

- (1) Connect the SVT17801, USB miniB port and PC using their cables.
- (2) Reset SVT17 ICD board.

Execution

- (1) Change the path to the “fls17801.elf” written in the command file if necessary.
By default (GNU17v1.2.0 and later), the path is set as “c:/epson/gnu17/mcu_model/17801/fls/fls17801.elf”.
* For command file modification, refer to the “S5U1C17001C Manual”.
- (2) Execute the “usb_ms_mmc” project using the IDE.
- (3) Once the “usb_ms_mmc” program is executed, the USB memory is recognized as a removable disk by the host PC.

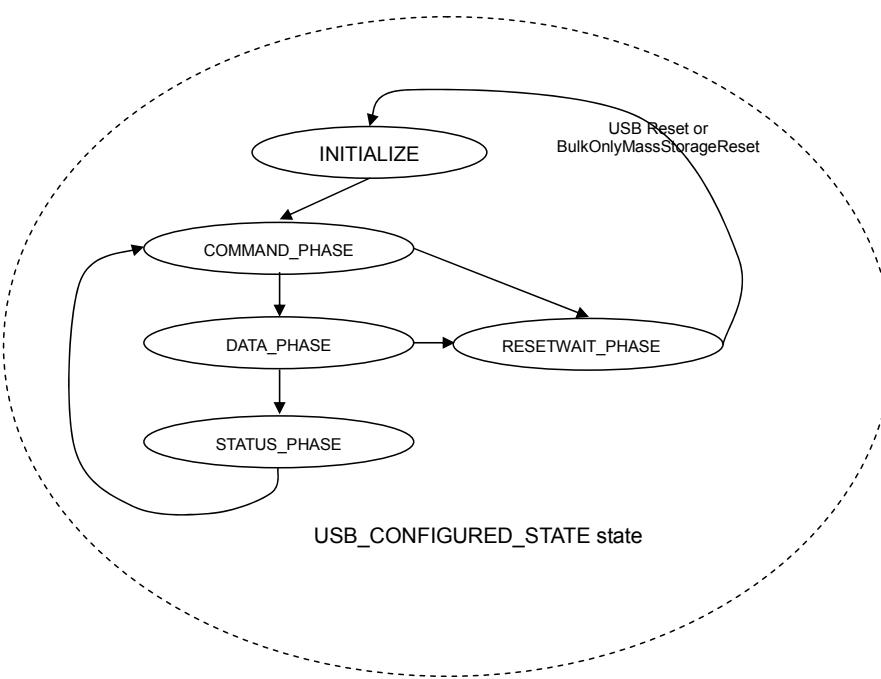
3.9 Outline of sample program operations

- (1) The USB interrupt of ITC is set.
- (2) The CLG is set for SPI operation voltage supply.
- (3) The SPI send/receive port is set.
- (4) The ITC interrupt is enabled.
- (5) The MMC is initialized.
- (6) The USB functions are initialized.
- (7) The Storage process is initialized.
- (8) The USB communication is made with the host PC, and the USB memory is recognized as a removable disk.

- Mass Storage process

Mass Storage status transition

The Mass Storage process in the sample program is executed and its status transitions as follows.
If the “USB Device State” is not in the CONFIGURED state, this process is not executed as illustrated below.
* The CONFIGURED state occurs when the host recognizes the mass storage as a device and when it issues the “SetConfiguration” request.



Also, each Bulk Process function is called and the Bulk process is executed according to the Bulk IN or OUT state.

INITIALIZE

The INITIALIZE state occurs when this program is started, when a “USB Bus Reset” occurs, or when the “BulkOnlyMassStorageReset” command is received.

When this state occurs, the Bulk-IN is set to “IDLE” state and the Bulk-OUT is set to “DATA RCV” state. The FIFO of OUT pipe is cleared. Also, the CBW storage pointer is set for CBW reception, the NAK of Bulk-OUT is released, and data reception is started.

COMMAND_PHASE

In the COMMAND_PHASE state, the CBW packet or not is determined after the OUT data transfer. Then, the CommandBlock included in the CBW packet is saved.

DATA_PHASE

In the DATA_PHASE state, the completion of data transfer relating the CommandBlock is waited. Nothing

3. SOFTWARE DESCRIPTION

occurs during data transfer.

STATUS_PHASE

The CSW packets are sent.

RESETWAIT_PHASE

If an inconsistent state, a command error or others occurs, the error state occurs. This state can be recovered by either “USB Bus Reset”, “BulkOnlyMassStorageReset”, “Power Off and On”, or “HardReset”.

MassStorage flag

The MassStorage processing and SCSI command processing are executed by setup and monitoring of the following flags.

```
typedef struct {
    unsigned char PowerOnFlag:1;           // Power-ON flag
    unsigned char Direction:2;             // Data transfer direction
    unsigned char CmdReceive:1;            // Command reception
    unsigned char DivideFlag:1;             // Divide (or split) flag
    unsigned char NextData:1;              // Next data flag
    unsigned char CmdComplete:1;            // Command completion
    unsigned char Lun;                     // Process LUN number
} T_TRANS_FLAG;
```

Power-ON flag

This flag is set when the system is initialized.

It is used to return the “Power On Reset” or “Bus Device Reset” sense code in response to the first command issued by the host. When the sense code is returned, this flag is cleared.

Transfer direction

The data transfer direction is set by the CBW command when it is received.

Command receive flag

This flag is set when the CBW command is received. When the command is accepted by the SCSI process, this flag is cleared.

Divide flag

This flag is set if the data transfer size is greater than the intermediate buffer size of USB, and the issue of command to the connected device and the data transfer are divided and processed.

Next data flag

This flag is set when data transfer is completed. When this process is accepted by the SCSI command process, this flag is cleared.

Command completion flag

This flag is set when command process is completed. The MassStorage process executes the Status processing according to this flag state.

LUN number

The LUN number of this flag is set by the CBW command when it is received.

● SCSI command processing

The SCSI command process of the sample program checks both the “CmdReceive” command receive flag and the “NextData” data flag, and executes operations. In other cases, no processing starts.

The command receive flag and the data flag are set by the MassStorage process.

Command reception

If the Command Reception flag is set, the command is checked for its validity. When the command is

determined to be normal, it is executed.

Actually, however, the data reading commands are only executed.

A data write command is issued to the connected device when data is received.

Data flag

If this flag is set, it indicates that the data transfer has completed. For a data read command, the remaining data to be sent is checked. If the send data remains, the next command is issued. If not, the Command Completion flag is set and the process is terminated.

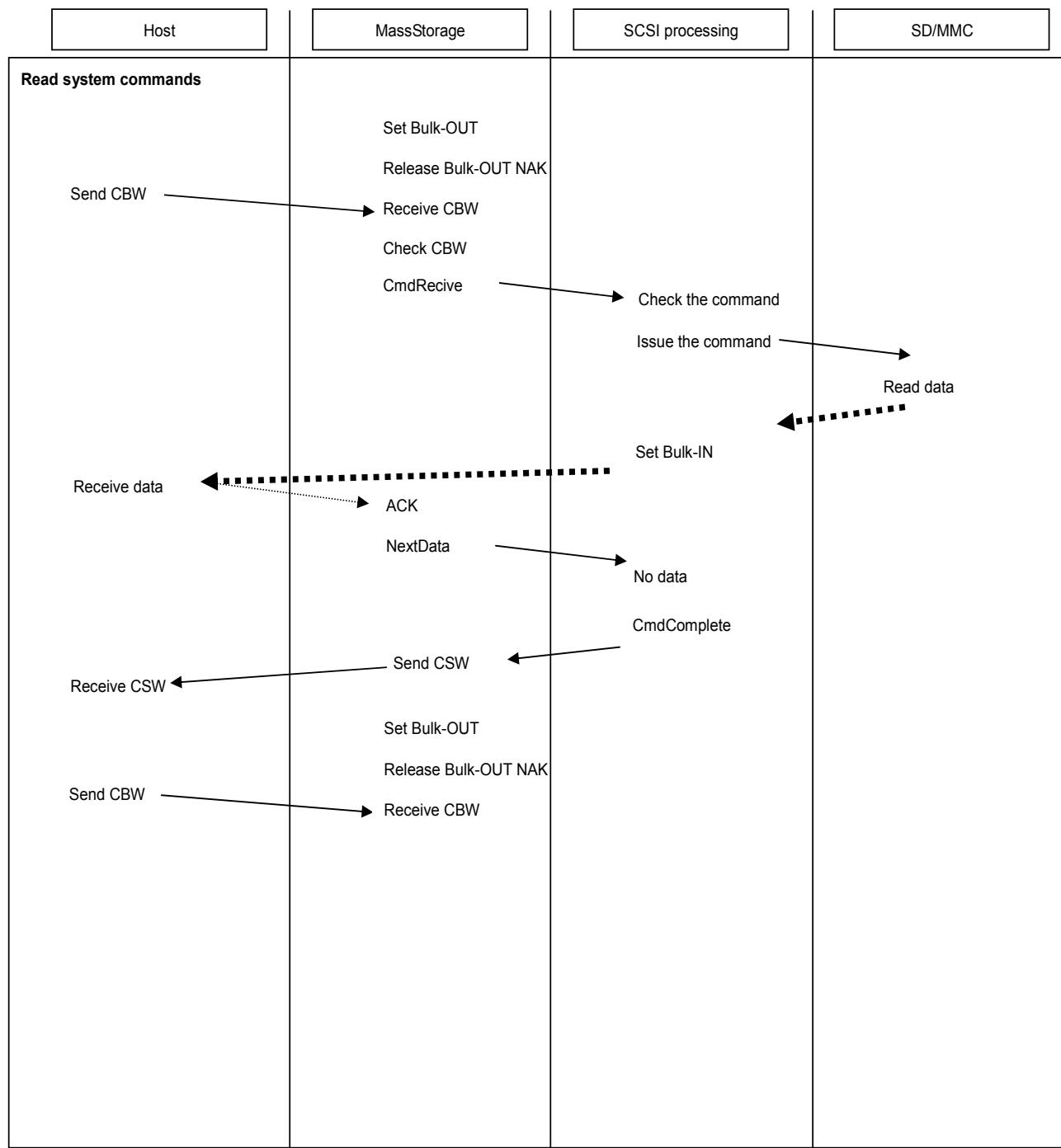
For a data write command, the command is issued to the connected device and the data is passed to it. If no data remains, the Command Completion flag is set and the process is terminated.

3. SOFTWARE DESCRIPTION

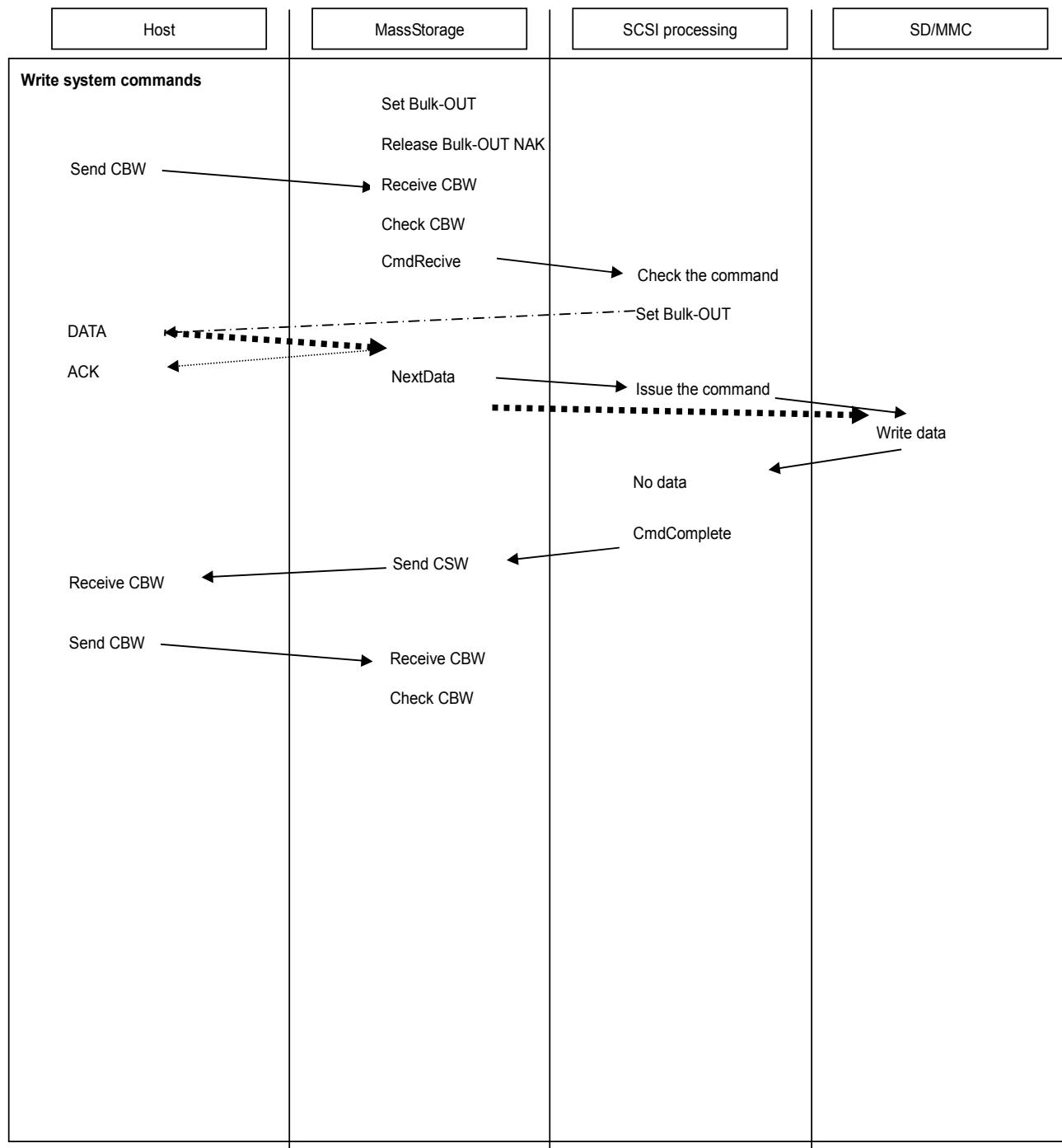
3.10 OPERATION FLOWCHARTS

The following outlines the system operation image.

3.10.1 Read command operation flow



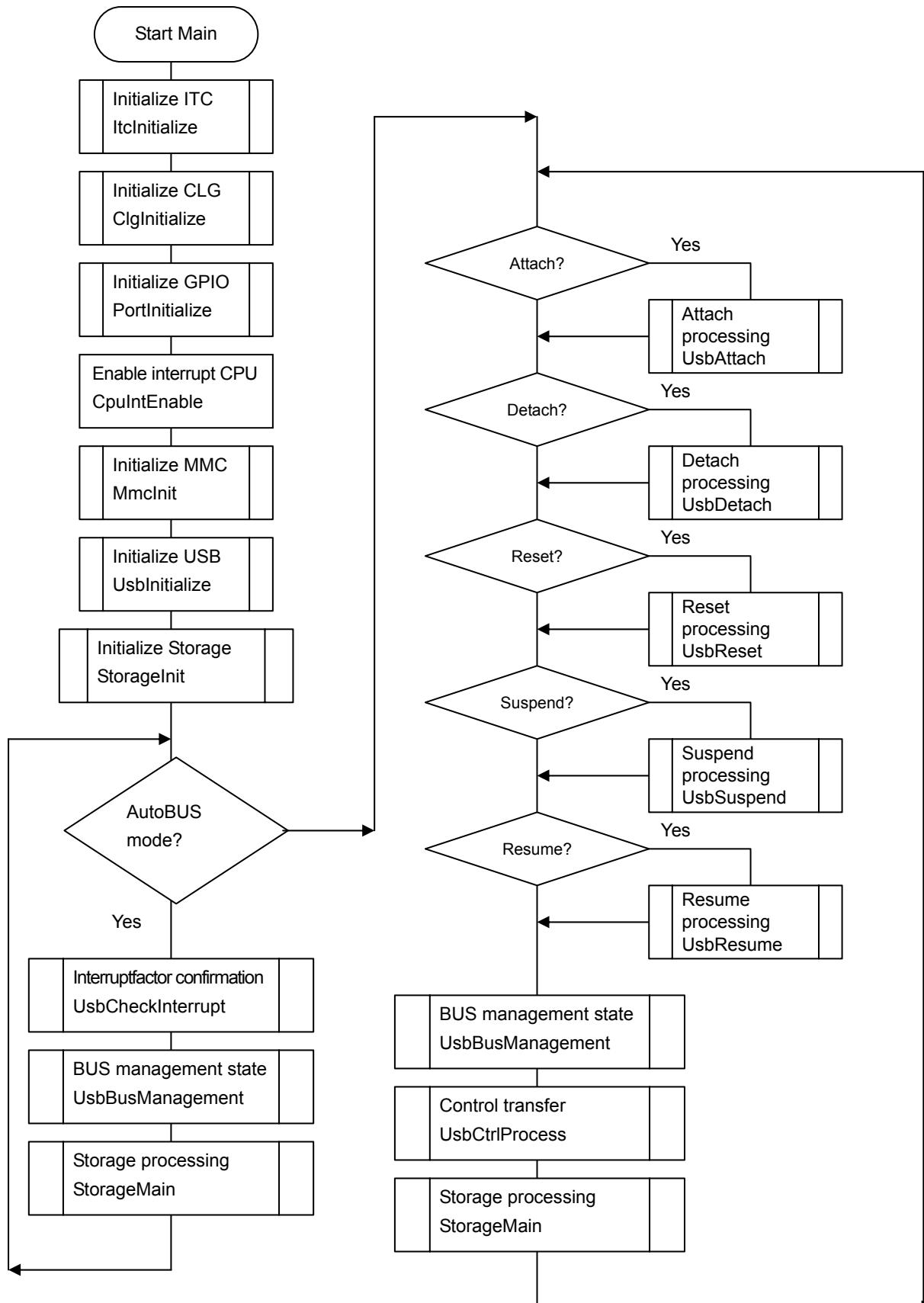
3.10.2 Write command operation flow



3. SOFTWARE DESCRIPTION

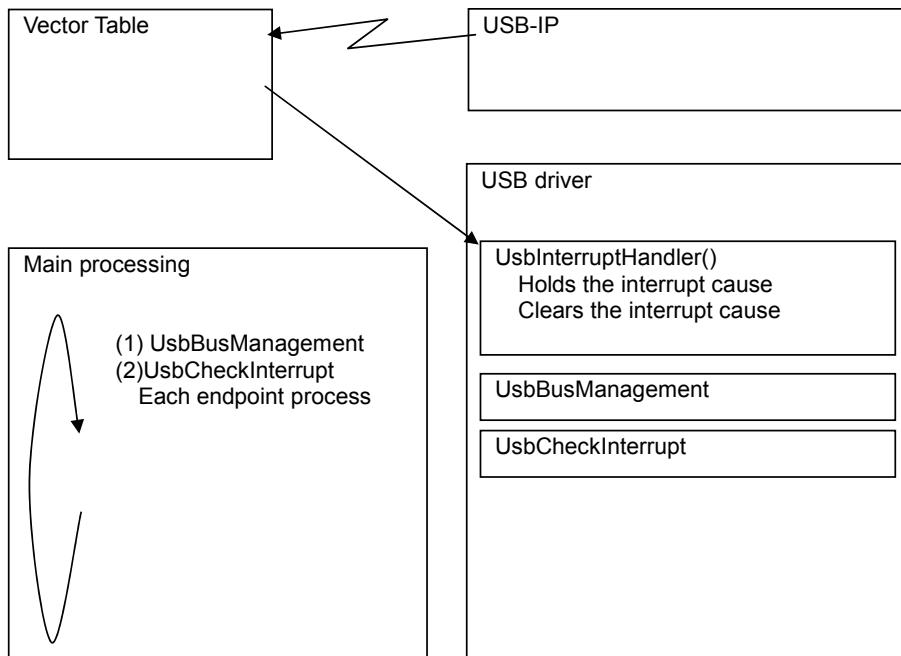
3.11 Flowchart

The following shows the flowchart of the main routine and interrupt handler functions.



3.12 Description of API Specifications

- Interrupt processing



The UsbInterruptHandler contained in the USB driver is called from the Vector. During interrupt processing, the hardware (HW) interrupt cause is kept in internal variables, and the HW interrupt cause is cleared.

The UsbCheckInterrupt determines the interrupt cause being kept and saves the result in the State Result variable at each endpoint.

The Main process calls the UsbCheckInterrupt and executes processing based on the Result value.

- Bus Management process

The UsbBusManagement function is provided for normal system operations so that the user needs not consider the BusReset, Suspend, Resume, VBUS-ON/OFF and others.

As the chart of “Interrupt process” shows, the Bus Management process needs to be called at regular interval by the Main process function.

If the USB and another process (implemented functions) are executed simultaneously, the response to a change of bus state may be delayed.

If a BusReset occurs and if you wish to issue the Reset to another device (for example, if you wish to use SD/MMC in MassStorage), you can set the Reset process function for another device in the ResetFunc of the “config” of UsbInit function, set a pointer to this function, and execute the Reset process.

- Standard request processing

The Standard Request process is executed inside the driver.

- Class Request process

Class Request can be used for execution of any process if the Pointer of the process function, contained in “classFunc” of the “config” of UsbInit, is set. The return value of the process function must be “SUCCESS” during normal termination. It must be “ERROR” during abnormal termination.

- Vendor Request process

Vendor Request can be used for execution of any process if the Pointer of the process function, contained in “vendorFunc” of the “config” of UsbInit, is set.

The return value of the process function must be “SUCCESS” during normal termination. It must be

3. SOFTWARE DESCRIPTION

“ERROR” during abnormal termination.

● RemoteWakeUp process

RemoteWakeUp can be used to set the pointer of the process function that is contained in “remoteFunc” of the “config” of UsbInit.

In such case, if the “SUCCESS” value of the process function is returned as the external cause for wakeup (for example, when the Wakeup switch is pressed), the RemoteWakeUp is executed. If no cause exists and if “ERROR” is returned, the Wakeup is not executed.

3.13 Detailed Description of USB Drivers

The following lists the functions described in files usb_drv.c and usb_api.h.

The USB drivers can be classified into two groups: low-level USB drivers and high-level USB drivers. For their function details, refer to the API list.

3.13.1 Low-level USB driver API details

API List

Function	API	Description
VBUS State Check	UsbVBUSCheck	Checks the VBUS state.
USB Line State Check	UsbLineStateCheck	Checks the line state.
Attach process	UsbAttach	Executes Attach processing.
Detach process	UsbDetach	Executes Detach processing.
Bus Reset process	UsbReset	Executes Reset processing.
Suspend process	UsbSuspend	Executes Suspend processing.
Resume process	UsbResume	Executes Resume processing.
FIFO Read process	UsbFIFORead	Executes FIFO Read processing.
FIFO Write Process	UsbFIFOWrite	Executes FIFO Write processing.
Get Available FIFO Read Size	UsbFIFOReadRemain	Gets the available FIFO read size.
Get Available FIFO Write Size	UsbFIFOWriteRemain	Gets the available FIFO write size.
FIFO Clear	UsbFIFOClear	Clears the FIFO memory.
FIFO All Clear	UsbFIFOAllClear	Clears all FIFO memories.
Get FIFO Read Connection	UsbGetFIFOReadJoin	Gets the FIFO read connection state.
Get FIFO Write Connection	UsbGetFIFOWriteJoin	Gets the FIFO write connection state.
Enable FIFO Interrupt	UsbFIFOIntEnable	Enables a FIFO interrupt cause.
Disable FIFO Interrupt	UsbFIFOIntDisable	Disables a FIFO interrupt cause.
RemoteWakeUp process	UsbRemoteWakeUp	Executes the RemoteWakeUp process.
TestMode Setup Process	UsbSetTestMode	Executes the TestMode setting process.
TestMode Start Process	UsbStartTestMode	Executes the TestMode start process.
Auto Addressing	UsbSetAutoAddress	Executes automatic addressing.
Get Address	UsbGetAddress	Executes the address acquisition.
Set Address	UsbSetAddress	Sets the USB address.
Get Operation Mode	UsbGetOpMode	Executes the operation mode acquisition.
Set Operation Mode	UsbSetOpMode	Executes the operation mode setup process.
Get Setup Packet	UsbGetSetupPacket	Obtains the Setup packet.
Get EP0 Direction	UsbGetEP0Direction	Gets the EP0 transfer direction.
Set EP0 Direction	UsbSetEP0Direction	Sets the EP0 transfer direction.
Enable Endpoint	UsbEnableEndpoint	Enables an endpoint.
Disable Endpoint	UsbDisableEndpoint	Disables an endpoint.
Set EP0 Control	UsbSetEP0Control	Sets EP0 Control.
Get EP0 Control	UsbGetEP0Control	Gets the EP0 control set value.
Set Endpoint Control	UsbSetEndpointControl	Sets the endpoint control.
Get Endpoint Control	UsbGetEndpointControl	Gets the endpoint control set value.
Enable Endpoint Interrupt	UsbEndpointIntEnable	Enables an endpoint interrupt.
Disable Endpoint Interrupt	UsbEndpointIntDisable	Disables an endpoint interrupt.
Get Endpoint Interrupt	UsbGetEndpointInt	Get the enabled endpoint interrupt.
Clear Endpoint Interrupt	UsbClearEndpointInt	Clears an endpoint interrupt.
Enable Main Interrupt	UsbMainIntEnable	Enables the Main interrupt.
Disable Main Interrupt	UsbMainIntDisable	Disables the Main interrupt.

3. SOFTWARE DESCRIPTION

Function	API	Description
Enable SIE Interrupt	UsbSIEIntEnable	Enables an SIE interrupt.
Disable SIE Interrupt	UsbSIEIntDisable	Disables an SIE interrupt.
Enable EPr Interrupt	UsbEPrIntEnable	Enables an EPr interrupt.
Disable EPr Interrupt	UsbEPrIntDisable	Disables an EPr interrupt.
Get Main Interrupt Cause	UsbGetMainInt	Gets the Main interrupt cause.
Clear Main Interrupt Cause	UsbClearMainInt	Clears the Main interrupt cause.
Get SIE Interrupt Cause	UsbGetSIEInt	Gets the SIE interrupt cause.
Clear SIE Interrupt Cause	UsbClearSIEInt	Clears the SIE interrupt cause.
Get EPr Interrupt Cause	UsbGetEPrInt	Gets the EPr interrupt cause.
Clear EPr Interrupt Cause	UsbClearEPrInt	Clears the EPr interrupt cause.
Get FIFO Interrupt Cause	UsbGetFIFOInt	Gets the FIFO interrupt cause.
Clear FIFO Interrupt Cause	UsbClearFIFOInt	Clears the FIFO interrupt cause.
Get EP0 Interrupt Cause	UsbGetEP0Int	Gets the EP0 interrupt cause.
Clear EP0 Interrupt Cause	UsbClearEP0Int	Clears the EP0 interrupt cause.
Get EPa Interrupt Cause	UsbGetEPaInt	Gets the EPa interrupt cause.
Clear EPa Interrupt Cause	UsbClearEPaInt	Clears the EPa interrupt cause.
Get EPb Interrupt Cause	UsbGetEPbInt	Gets the EPb interrupt cause.
Clear EPb Interrupt Cause	UsbClearEPbInt	Clears the EPb interrupt cause.
Get EPc Interrupt Cause	UsbGetEPcInt	Gets the EPc interrupt cause.
Clear EPc Interrupt Cause	UsbClearEPcInt	Clears the EPc interrupt cause.
Get EPd Interrupt Cause	UsbGetEPdInt	Gets the EPd interrupt cause.
Clear EPd Interrupt Cause	UsbClearEPdInt	Clears the EPd interrupt cause.
Set Max Packet Size	UsbSetEndpointMaxSize	Sets the maximum packet size at the specified endpoint.
Get Max Packet Size	UsbGetEndpointMaxSize	Gets the maximum packet size at the specified endpoint.
Set Endpoint Number	UsbSetEndpointNumber	Sets an endpoint number.
Get Endpoint Number	UsbGetEndpointNumber	Gets the endpoint number.
Set Endpoint Direction	UsbSetEndpointDirection	Sets an endpoint transfer direction.
Get Endpoint Direction	UsbGetEndpointDirection	Gets the endpoint transfer direction.
Set Endpoint Mode	UsbSetEndpointMode	Sets the endpoint mode.
Get Endpoint Mode	UsbGetEndpointMode	Gets the endpoint mode.
Set Endpoint Toggle Mode	UsbSetEndpointToggleMode	Sets the endpoint toggle mode.
Get Endpoint Toggle Mode	UsbGetEndpointToggleMode	Gets the endpoint toggle mode.
Set FIFO Area	UsbSetFIFOArea	Assigns a FIFO area.
Get FIFO Area	UsbGetFIFOArea	Gets the assigned FIFO area.
Write Descriptor Data	UsbWriteDescriptor	Writes the descriptor data.
Read Descriptor Data	UsbReadDescriptor	Reads the descriptor data.
Send Descriptor Data	UsbReplyDescriptor	Sends the descriptor data.
Get Remaining Descriptor Count	UsbRemainDescriptor	Gets the remaining data count of descriptor.
Get USB State	UsbGetUSBState	Gets the USB state.

3. SOFTWARE DESCRIPTION

VBUS Status Check

Format	unsigned char UsbVBUSCheck(void)
Function	Checks the VBUS state.
Argument	None
Return value	USB_VBUS_ON : VBUS-ON USB_VBUS_OFF : VBUS-OFF
(Description)	The current VBUS state is returned.

USB Line State Check

Format	unsigned char UsbLineStateCheck(void)
Function	Checks the line state.
Argument	None
Return value	USB_LINE_SE1 : SE1 state USB_LINE_K : K state USB_LINE_J : J state USB_LINE_SE0 : SE0 state
(Description)	The signal state on the USB cable is returned.

Attach Process

Format	int UsbAttach(void)
Function	Executes Attach processing.
Argument	None
Return value	SUCCESS : Normal termination ERROR : Abnormal termination
(Description)	The USB is activated and the Attach process is executed. The ERROR value is returned if the J state is not detected when the USB is activated. This function needs to be called again until the SUCCESS value is returned.

Detach Process

Format	void UsbDetach(void)
Function	Executes Detach processing.
Argument	None
Return value	None
(Description)	The USB is deactivated.

Bus Reset Process

Format	void UsbReset(void)
Function	Executes Reset processing.
Argument	None
Return value	None
(Description)	The BusReset process is executed.

Suspend Process

Format	void UsbSuspend(void)
Function	Executes Suspend processing.
Argument	None
Return value	None
(Description)	The process for bus suspend state is executed.

Resume Process

Format	void UsbResume(void)
Function	Executes Resume processing.
Argument	None
Return value	None
(Description)	The process for bus resume state is executed.

FIFO Read Process

Format	int UsbFIFORead(unsigned char number, unsigned char *buffer, unsigned short requestLength, unsigned short *actualLength)		
Function	Executes FIFO Read processing.		
Argument	number	-in	Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
	buffer	-in	Pointer to the data storage buffer
	requestLength	-in	Length of read request
	actualLength	-out	Length of actually read data
Return value	SUCCESS	: Normal termination	
	ERROR	: An error in argument	
(Description)	Data is read from the FIFO memory of the specified endpoint, and it is stored in the specified buffer. The specified length of request data may not be read. If "actualLength" is zero (0) and if the SUCCESS is returned, there is no data in the FIFO memory. The length of the actually read data must be checked by reading the "actualLength" value.		

3. SOFTWARE DESCRIPTION

FIFO Write Process

Format	int UsbFIFOWrite(unsigned char number, unsigned char *buffer, unsigned short requestLength, unsigned short *actualLength)														
Function	Executes FIFO Write processing.														
Argument	<table><tr><td>number</td><td>-in</td><td>Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D</td></tr><tr><td>buffer</td><td>-in</td><td>Pointer to the data storage buffer</td></tr><tr><td>requestLength</td><td>-in</td><td>Length of write request</td></tr><tr><td>actualLength</td><td>-out</td><td>Length of actually written data</td></tr></table>			number	-in	Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D	buffer	-in	Pointer to the data storage buffer	requestLength	-in	Length of write request	actualLength	-out	Length of actually written data
number	-in	Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D													
buffer	-in	Pointer to the data storage buffer													
requestLength	-in	Length of write request													
actualLength	-out	Length of actually written data													
Return value	SUCCESS : Normal termination ERROR : An error in argument														
(Description)	<p>Data of the specified buffer is written in the FIFO memory of the specified endpoint. The specified length of request data may not be written. If “actualLength” is zero (0) and if the SUCCESS is returned, there is no space in the FIFO memory. The length of the actually written data must be checked by reading the “actualLength” value.</p>														

Get Available FIFO Read Size

Format	int UsbFIFOReadRemain(unsigned char number, unsigned short *remain)								
Function	Gets the available FIFO read size.								
Argument	<table><tr><td>number</td><td>-in</td><td>Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D</td></tr><tr><td>remain</td><td>-out</td><td>Length of data that can be read</td></tr></table>			number	-in	Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D	remain	-out	Length of data that can be read
number	-in	Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D							
remain	-out	Length of data that can be read							
Return value	SUCCESS : Normal termination ERROR : An error in argument								
(Description)	The length of data that can be read from FIFO memory of the specified endpoint is returned.								

Get Available FIFO Write Size

Format	int UsbFIFOWriteRemain(unsigned char number, unsigned short *remain)				
Function	Gets the available FIFO write size.				
Argument	number -in	Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D			
	remain -out	Length of data that can be written			
Return value	SUCCESS : Normal termination ERROR : An error in argument				
(Description)	The length of data that can be written in FIFO memory of the specified endpoint is returned.				

FIFO Clear

Format	int UsbFIFOClear(unsigned char number)		
Function	Clears the FIFO memory.		
Argument	number -in	Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D	
Return value	SUCCESS : Normal termination ERROR : An error in argument		
(Description)	The FIFO memory of the specified endpoint is cleared.		

FIFO All Clear

Format	void UsbFIFOAllClear(void)		
Function	Clears all FIFO memories.		
Argument	None		
Return value	None		
(Description)	All FIFO memories are cleared.		

Get FIFO Read Connection

Format	unsigned short UsbGetFIFOReadJoin(void)		
Function	Gets the FIFO read connection state.		
Argument	None		
Return value	USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D		
(Description)	The endpoint that is connected to the FIFO memory read port is returned.		

3. SOFTWARE DESCRIPTION

Get FIFO Write Connection

Format	unsigned short UsbGetFIFOWriteJoin(void)
Function	Gets the FIFO write connection state.
Argument	None
Return value	USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
(Description)	The endpoint that is connected to the FIFO memory write port is returned.

Enable FIFO Interrupt

Format	int UsbFIFOIntEnable(unsigned char event)	
Function	Enables a FIFO interrupt cause.	
Argument	event	-in Interrupt enable bits (Can be ORed) USB_FIFO_INT_DESCCMP USB_FIFO_INT_INCMP USB_FIFO_INT_OUTCMP
Return value	SUCCESS : Normal termination ERROR : An error in argument	
(Description)	An interrupt due to the specified cause is enabled.	

Disable FIFO Interrupt

Format	int UsbFIFOIntDisable(unsigned char event)	
Function	Disables a FIFO interrupt cause.	
Argument	event	-in Interrupt disable bits (Can be ORed) USB_FIFO_INT_DESCCMP USB_FIFO_INT_INCMP USB_FIFO_INT_OUTCMP
Return value	SUCCESS : Normal termination ERROR : An error in argument	
(Description)	An interrupt due to the specified cause is disabled.	

RemoteWakeup Process

Format	void UsbRemoteWakeUp(void)	
Function	Executes the RemoteWakeup process.	
Argument	None	
Return value	None	
(Description)	The RemoteWakeup process is executed.	

TestMode Setup Process

Format	int UsbSetTestMode(unsigned char testMode)	
Function	Executes the TestMode setting process.	
Argument	testMode	-in Sets the Test mode. USB_TESTMODE_J USB_TESTMODE_K USB_TESTMODE_SE0_NAK USB_TESTMODE_PACKET
Return value	SUCCESS : Normal termination ERROR : An error in argument	
(Description)	The test mode specified by the argument is set.	

TestMode Start Process

Format	void UsbStartTestMode(void)	
Function	Executes the TestMode start process.	
Argument	None	
Return value	None	
(Description)	The TestMode that has been specified by UsbSetTestMode is executed. No value is returned by this function. The System Reset is required.	

Auto Addressing

Format	void UsbSetAutoAddress(unsigned char mode)	
Function	Auto Addressing	
Argument	mode	-in Specifies the Auto Address mode. USB_AUTOADDRESS_ON USB_AUTOADDRESS_OFF
Return value	None	
(Description)	The Auto Address Return function is turned On or Off.	

Get Address

Format	unsigned char UsbGetAddress(void)	
Function	Executes the address acquisition.	
Argument	None	
Return value	USB Address	
(Description)	The address that has been set by the SetAddress request from the host is returned.	

3. SOFTWARE DESCRIPTION

Set Address

Format	void UsbSetAddress(unsigned char address)	
Function	Sets the USB address.	
Argument	address	-in The address to be set
Return value	None	
(Description) The specified USB address is set. When this API is called, the Auto Address Set is turned Off even if the Auto Address has been set by UsbSetAutoAddress.		

Get Operation Mode

Format	unsigned char UsbGetOpMode(void)	
Function	Executes the operation mode acquisition.	
Argument	None	
Return value	Operation Mode USB_OPMODE_NORMAL USB_OPMODE_NONDRIVE USB_OPMODE_DISBITSTUFF	
(Description) The current operation mode is returned.		

Set Operation Mode

Format	int UsbSetOpMode(unsigned char opMode)	
Function	Executes the operation mode setup process.	
Argument	opMode -in The opMode to be set USB_OPMODE_NORMAL USB_OPMODE_NONDRIVE USB_OPMODE_DISBITSTUFF	
Return value	SUCCESS : Normal termination ERROR : An error in argument	
(Description) The operation mode is set.		

Get Setup Packet

Format	void UsbGetSetupPacket(T_USB_REQUEST_PACKET *packet)	
Function	Obtains the Setup packet.	
Argument	packet -in SETUP Packet	
Return value	None	
(Description) The received setup packet is obtained.		

Get EP0 Direction

Format	unsigned char UsbGetEP0Direction(void)
Function	Gets the EP0 transfer direction.
Argument	
Return value	USB_EP0_IN USB_EP0_OUT
(Description)	The data transfer direction at endpoint 0 is returned.

Set EP0 Direction

Format	void UsbSetEP0Direction(unsigned char direction)
Function	Sets the EP0 transfer direction.
Argument	direction -in Data transfer direction USB_EP0_IN USB_EP0_OUT
Return value	None
(Description)	The data transfer direction at endpoint 0 is set.

Enable Endpoint

Format	void UsbEnableEndpoint(unsigned char number)
Function	Enables an endpoint.
Argument	number -in Endpoint number USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
Return value	None
(Description)	The specified endpoint is enabled.

Disable Endpoint

Format	void UsbDisableEndpoint(unsigned char number)
Function	Disables an endpoint.
Argument	number -in Endpoint number USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
Return value	None
(Description)	The specified endpoint is disabled.

3. SOFTWARE DESCRIPTION

Set EP0 Control

Format	int UsbSetEP0Control(unsigned char direction, unsigned char set, unsigned char clear)		
Function	Sets the EP0 control.		
Argument	direction -in	Direction at EP0 USB_EP0_IN USB_EP0_OUT	
	set -in	The set flag (Can be ORed) USB_EP_CONTROL_STALL USB_EP_CONTROL_NAK USB_EP_CONTROL_TOGGLE_CLR USB_EP_CONTROL_TOGGLE_SET USB_EP_CONTROL_DISAFNAK USB_EP_CONTROL_ENSHORTPKT USB_EP_CONTROL_AUTOFORCENAK	
	clear -in	The release flag (Can be ORed) USB_EP_CONTROL_STALL USB_EP_CONTROL_NAK USB_EP_CONTROL_DISAFNAK USB_EP_CONTROL_ENSHORTPKT USB_EP_CONTROL_AUTOFORCENAK	
Return value	SUCCESS: Normal termination ERROR : An error in argument		
(Description)	The "set" bit is set to 0 in the EP0 Control register. Also, "clear" bit is set to 0. If both "set" and "clear" have the same bit value, the ERROR is returned.		
Example:	UsbSetEP0Control(USB_EP0_IN, USB_EP_CONTROL_STALL USB_EP_CONTROL_NAK, USB_EP_CONTROL_ENSHORTPKT USB_EP_CONTROL_AUTOFORCENAK);		
	The ForceSTALL or ForceNAK of the EP0 ControlIN is set, and the EnShortPkt or AutoForceNAK is cleared.		

Get EP0 Control

Format	unsigned char UsbGetEP0Control(unsigned char direction)		
Function	Gets the EP0 control set value.		
Argument	direction -in	Direction at EP0 USB_EP0_IN USB_EP0_OUT	
Return value	USB_EP_CONTROL_STALL USB_EP_CONTROL_NAK USB_EP_CONTROL_TOGGLE_0 USB_EP_CONTROL_TOGGLE_1 USB_EP_CONTROL_DISAFNAK USB_EP_CONTROL_ENSHORTPKT USB_EP_CONTROL_AUTOFORCENAK		
(Description)	The EP0 Control register value is returned.		

Set Endpoint Control

Format	int UsbSetEndpointControl(unsigned char number, unsigned char set, unsigned char clear)		
Function	Sets the endpoint control.		
Argument	number	-in	Endpoint number USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
	set	-in	The set flag (Can be ORed) USB_EP_CONTROL_STALL USB_EP_CONTROL_NAK USB_EP_CONTROL_TOGGLE_CLR USB_EP_CONTROL_TOGGLE_SET USB_EP_CONTROL_DISAFNAK USB_EP_CONTROL_ENSHORTPKT USB_EP_CONTROL_AUTOFORCENAK
	clear	-in	The release flag (Can be ORed) USB_EP_CONTROL_STALL USB_EP_CONTROL_NAK USB_EP_CONTROL_DISAFNAK USB_EP_CONTROL_ENSHORTPKT USB_EP_CONTROL_AUTOFORCENAK
Return value	SUCCESS: Normal termination ERROR: An error in argument		
(Description)	The “set” bit is set to 1 in the Control register at the specified endpoint. Also, “clear” bit is set to 0. If both “set” and “clear” have the same bit value, the ERROR is returned.		
Example:	UsbSetEndpointControl(USB_ENDPOINT_A, USB_EP_CONTROL_STALL USB_EP_CONTROL_NAK, USB_EP_CONTROL_ENSHORTPKT USB_EP_CONTROL_AUTOFORCENAK);		
	The ForceSTALL or ForceNAK of the Control register at endpoint A is set, and the EnShortPkt or AutoForceNAK is cleared.		

3. SOFTWARE DESCRIPTION

Get Endpoint Control

Format	unsigned char UsbGetEndpointControl(unsigned char number)		
Function	Gets the endpoint control set value.		
Argument	number	-in	Endpoint number USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
Return value	USB_EP_CONTROL_STALL USB_EP_CONTROL_NAK USB_EP_CONTROL_TOGGLE_0 USB_EP_CONTROL_TOGGLE_1 USB_EP_CONTROL_DISAFNAK USB_EP_CONTROL_ENSHORTPKT USB_EP_CONTROL_AUTOFORCENAK		
(Description)	The Control register value of the specified endpoint is returned.		

Enable Endpoint Interrupt

Format	int UsbEndpointIntEnable(unsigned char number, unsigned char event)		
Function	Enables an endpoint interrupt.		
Argument	number	-in	Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
	Event	-in	Interrupt enable bits USB_EP_INT_OUT_TRAN_ERR USB_EP_INT_IN_TRAN_ERR USB_EP_INT_OUT_TRAN_NAK USB_EP_INT_IN_TRAN_NAK USB_EP_INT_OUT_TRAN_ACK USB_EP_INT_IN_TRAN_ACK USB_EP_INT_OUT_SHORT_ACK
Return value	SUCCESS: Normal termination ERROR : An error in argument		
(Description)	An interrupt cause of the specified endpoint is enabled.		

Disable Endpoint Interrupt

Format	int UsbEndpointIntDisable(unsigned char number, unsigned char event)		
Function	Disables an endpoint interrupt.		
Argument	Number -in	Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D	
	event -in	Interrupt disable bits USB_EP_INT_OUT_TRAN_ERR USB_EP_INT_IN_TRAN_ERR USB_EP_INT_OUT_TRAN_NAK USB_EP_INT_IN_TRAN_NAK USB_EP_INT_OUT_TRAN_ACK USB_EP_INT_IN_TRAN_ACK USB_EP_INT_OUT_SHORT_ACK	
Return value	SUCCESS: Normal termination ERROR : An error in argument		
(Description)	An interrupt cause of the specified endpoint is disabled.		

Get Endpoint Interrupt

Format	unsigned char UsbGetEndpointInt(unsigned char number)		
Function	Get the enabled endpoint interrupt.		
Argument	number -in	Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D	
Return value	(The following values may be ORed.) USB_EP_INT_OUT_TRAN_ERR USB_EP_INT_IN_TRAN_ERR USB_EP_INT_OUT_TRAN_NAK USB_EP_INT_IN_TRAN_NAK USB_EP_INT_OUT_TRAN_ACK USB_EP_INT_IN_TRAN_ACK USB_EP_INT_OUT_SHORT_ACK		
(Description)	An interrupt enabled state of the specified endpoint is returned.		

3. SOFTWARE DESCRIPTION

Clear Endpoint Interrupt

Format	void UsbClearEndpointInt(unsigned char number, unsigned char event)		
Function	Clears an endpoint interrupt.		
Argument	number	-in	Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
	event	-in	The clear cause (Can be ORed) USB_EP_INT_OUT_TRAN_ERR USB_EP_INT_IN_TRAN_ERR USB_EP_INT_OUT_TRAN_NAK USB_EP_INT_IN_TRAN_NAK USB_EP_INT_OUT_TRAN_ACK USB_EP_INT_IN_TRAN_ACK USB_EP_INT_OUT_SHORT_ACK
Return value	None		
(Description)	The specified interrupt cause of the specified endpoint is cleared.		

Enable Main Interrupt

Format	void UsbMainIntEnable(unsigned char event)		
Function	Enables the Main interrupt.		
Argument	event	-in	The interrupt cause (Can be ORed) USB_MAIN_INT_RCVSETUP USB_MAIN_INT_EP0 USB_MAIN_INT_FIFO USB_MAIN_INT_DMA USB_MAIN_INT_EPR USB_MAIN_INT_SIE
Return value	None		
(Description)	The specified interrupt cause of the Main interrupt register is enabled.		

Disable Main Interrupt

Format	void UsbMainIntDisable(unsigned char event)		
Function	Disables the Main interrupt.		
Argument	event	-in	The interrupt cause (Can be ORed) USB_MAIN_INT_RCVSETUP USB_MAIN_INT_EP0 USB_MAIN_INT_FIFO USB_MAIN_INT_DMA USB_MAIN_INT_EPR USB_MAIN_INT_SIE
Return value	None		
(Description)	The specified interrupt cause of the Main interrupt register is disabled.		

Enable SIE Interrupt

Format	void UsbSIEIntEnable(unsigned char event)	
Function	Enables an SIE interrupt.	
Argument	event -in	The interrupt cause (Can be ORed) USB_SIE_INT_SETADDRESSCMP USB_SIE_INT_DETECTJ USB_SIE_INT_RCVSOF USB_SIE_INT_DETECTSUSPEND USB_SIE_INT_DETECTRESET USB_SIE_INT_NONJ USB_SIE_INT_VBUS
Return value	None	
(Description)	The specified interrupt cause of the SIE interrupt register is disabled.	

Disable SIE Interrupt

Format	void UsbSIEIntDisable(unsigned char event)	
Function	Disables an SIE interrupt.	
Argument	event -in	The interrupt cause (Can be ORed) USB_SIE_INT_SETADDRESSCMP USB_SIE_INT_DETECTJ USB_SIE_INT_RCVSOF USB_SIE_INT_DETECTSUSPEND USB_SIE_INT_DETECTRESET USB_SIE_INT_NONJ USB_SIE_INT_VBUS
Return value	None	
(Description)	The specified interrupt cause of the SIE interrupt register is disabled.	

Enable EPr Interrupt

Format	void UsbEPrIntEnable(unsigned char event)	
Function	Enable an EPr interrupt.	
Argument	event -in	The interrupt cause (Can be ORed) USB_EPR_INT_EPA USB_EPR_INT_EPB USB_EPR_INT_EPC USB_EPR_INT_EPD
Return value	None	
(Description)	The specified interrupt cause of the EPr interrupt register is enabled.	

3. SOFTWARE DESCRIPTION

Disable EPr Interrupt

Format	void UsbEPrIntDisable(unsigned char event)	
Function	Disables an EPr interrupt.	
Argument	event -in	The interrupt cause (Can be ORed) USB_EPR_INT_EPA USB_EPR_INT_EPB USB_EPR_INT_EPC USB_EPR_INT_EPD
Return value	None	
(Description)	The specified interrupt cause of the EPr interrupt register is disabled.	

Get Main Interrupt Cause

Format	unsigned char UsbGetMainInt(void)	
Function	Gets the Main interrupt cause.	
Argument	None	
Return value	(The following values may be ORed.) USB_MAIN_INT_RCVSETUP USB_MAIN_INT_EP0 USB_MAIN_INT_FIFO USB_MAIN_INT_DMA USB_MAIN_INT_EPR USB_MAIN_INT_SIE	
(Description)	The Main interrupt register value is returned.	

Clear Main Interrupt Cause

Format	void UsbClearMainInt(unsigned char event)	
Function	Clears the Main interrupt cause.	
Argument	event -in	The interrupt cause (Can be ORed) USB_MAIN_INT_RCVSETUP USB_MAIN_INT_EP0 USB_MAIN_INT_FIFO USB_MAIN_INT_DMA USB_MAIN_INT_EPR USB_MAIN_INT_SIE
Return value	None	
(Description)	The specified Main interrupt cause is cleared.	

Get SIE Interrupt Cause

Format	unsigned char UsbGetSIEInt(void)
Function	Gets the SIE interrupt cause.
Argument	None
Return value	(The following values may be ORed.) USB_SIE_INT_SETADDRESSCMP USB_SIE_INT_DETECTJ USB_SIE_INT_RCVSOF USB_SIE_INT_DETECTSUSPEND USB_SIE_INT_DETECTRESET USB_SIE_INT_NONJ USB_SIE_INT_VBUS
(Description) The SIE interrupt register value is returned.	

Clear SIE Interrupt Cause

Format	void UsbClearSIEInt(unsigned char event)	
Function	Clears the SIE interrupt cause.	
Argument	event	-in The interrupt cause (Can be ORed) USB_SIE_INT_SETADDRESSCMP USB_SIE_INT_DETECTJ USB_SIE_INT_RCVSOF USB_SIE_INT_DETECTSUSPEND USB_SIE_INT_DETECTRESET USB_SIE_INT_NONJ USB_SIE_INT_VBUS
Return value	None	
(Description) The specified SIE interrupt cause is cleared.		

Get EPr Interrupt Cause

Format	unsigned char UsbGetEPrInt(void)	
Function	Gets the EPr interrupt cause.	
Argument	None	
Return value	(The following values may be ORed.) USB_EPR_INT_EPA USB_EPR_INT_EPB USB_EPR_INT_EPC USB_EPR_INT_EPD	
(Description) The EPr interrupt register value is returned.		

3. SOFTWARE DESCRIPTION

Clear EPr Interrupt Cause

Format	void UsbClearEPrInt(unsigned char event)		
Function	Clears the EPr interrupt cause.		
Argument	event	-in	The interrupt cause (Can be ORed) USB_EPR_INT_EPA USB_EPR_INT_EPB USB_EPR_INT_EPC USB_EPR_INT_EPD
Return value	None		
(Description)	The specified EPr interrupt cause is cleared.		

Get FIFO Interrupt Cause

Format	unsigned char UsbGetFIFOInt(void)		
Function	Gets a FIFO interrupt cause.		
Argument	None		
Return value	(The following values may be ORed.) USB_FIFO_INT_DESCCMP USB_FIFO_INT_INCMP USB_FIFO_INT_OUTCMP		
(Description)	The FIFO interrupt register value is returned.		

Clear FIFO Interrupt Cause

Format	void UsbClearFIFOInt(unsigned char event)		
Function	Clears the FIFO interrupt cause.		
Argument	event	-in	The interrupt cause (Can be ORed) USB_FIFO_INT_DESCCMP USB_FIFO_INT_INCMP USB_FIFO_INT_OUTCMP
Return value	None		
(Description)	The specified FIFO interrupt cause is cleared.		

Get EP0 Interrupt Cause

Format	unsigned char UsbGetEP0Int(void)		
Function	Gets the EP0 interrupt cause.		
Argument	None		
Return value	(The following values may be ORed.) USB_EP_INT_OUT_TRAN_ERR USB_EP_INT_IN_TRAN_ERR USB_EP_INT_OUT_TRAN_NAK USB_EP_INT_IN_TRAN_NAK USB_EP_INT_OUT_TRAN_ACK USB_EP_INT_IN_TRAN_ACK		
(Description)	The EP0 interrupt register value is returned.		

Clear EP0 Interrupt Cause

Format	void UsbClearEP0Int(unsigned char event)	
Function	Clears the EP0 interrupt cause.	
Argument	event -in	The interrupt cause (Can be ORed) USB_EP_INT_OUT_TRAN_ERR USB_EP_INT_IN_TRAN_ERR USB_EP_INT_OUT_TRAN_NAK USB_EP_INT_IN_TRAN_NAK USB_EP_INT_OUT_TRAN_ACK USB_EP_INT_IN_TRAN_ACK
Return value	None	
(Description)	The specified EP0 interrupt cause is cleared.	

Get EPa Interrupt Cause

Format	unsigned char UsbGetEPaInt(void)	
Function	Gets an EPa interrupt cause.	
Argument	None	
Return value	(The following values may be ORed.) USB_EP_INT_OUT_TRAN_ERR USB_EP_INT_IN_TRAN_ERR USB_EP_INT_OUT_TRAN_NAK USB_EP_INT_IN_TRAN_NAK USB_EP_INT_OUT_TRAN_ACK USB_EP_INT_IN_TRAN_ACK USB_EP_INT_OUT_SHORT_ACK	
(Description)	The EPa interrupt register value is returned.	

Clear EPa Interrupt Cause

Format	void UsbClearEPaInt(unsigned char event)	
Function	Clears the EPa interrupt cause.	
Argument	event -in	The interrupt cause (Can be ORed) USB_EP_INT_OUT_TRAN_ERR USB_EP_INT_IN_TRAN_ERR USB_EP_INT_OUT_TRAN_NAK USB_EP_INT_IN_TRAN_NAK USB_EP_INT_OUT_TRAN_ACK USB_EP_INT_IN_TRAN_ACK USB_EP_INT_OUT_SHORT_ACK
Return value	None	
(Description)	The specified EPa interrupt cause is cleared.	

3. SOFTWARE DESCRIPTION

Get EPb Interrupt Cause

Format	unsigned char UsbGetEPbInt(void)
Function	Gets the EPb interrupt cause.
Argument	None
Return value	(The following values may be ORed.) USB_EP_INT_OUT_TRAN_ERR USB_EP_INT_IN_TRAN_ERR USB_EP_INT_OUT_TRAN_NAK USB_EP_INT_IN_TRAN_NAK USB_EP_INT_OUT_TRAN_ACK USB_EP_INT_IN_TRAN_ACK USB_EP_INT_OUT_SHORT_ACK
(Description) The EPb interrupt register value is returned.	

Clear EPb Interrupt Cause

Format	void UsbClearEPbInt(unsigned char event)	
Function	Clears an EPb interrupt cause.	
Argument	event	-in The interrupt cause (Can be ORed) USB_EP_INT_OUT_TRAN_ERR USB_EP_INT_IN_TRAN_ERR USB_EP_INT_OUT_TRAN_NAK USB_EP_INT_IN_TRAN_NAK USB_EP_INT_OUT_TRAN_ACK USB_EP_INT_IN_TRAN_ACK USB_EP_INT_OUT_SHORT_ACK
Return value	None	
(Description) The specified EPb interrupt cause is cleared.		

Get EPc Interrupt Cause

Format	unsigned char UsbGetEPcInt(void)
Function	Gets the EPc interrupt cause.
Argument	None
Return value	(The following values may be ORed.) USB_EP_INT_OUT_TRAN_ERR USB_EP_INT_IN_TRAN_ERR USB_EP_INT_OUT_TRAN_NAK USB_EP_INT_IN_TRAN_NAK USB_EP_INT_OUT_TRAN_ACK USB_EP_INT_IN_TRAN_ACK USB_EP_INT_OUT_SHORT_ACK
(Description) The EPc interrupt register value is returned.	

Clear EPc Interrupt Cause

Format	void UsbClearEPcInt(unsigned char event)	
Function	Clears the EPc interrupt cause.	
Argument	event -in	The interrupt cause (Can be ORed) USB_EP_INT_OUT_TRAN_ERR USB_EP_INT_IN_TRAN_ERR USB_EP_INT_OUT_TRAN_NAK USB_EP_INT_IN_TRAN_NAK USB_EP_INT_OUT_TRAN_ACK USB_EP_INT_IN_TRAN_ACK USB_EP_INT_OUT_SHORT_ACK
Return value	None	
(Description)	The specified EPc interrupt cause is cleared.	

Get EPd Interrupt Cause

Format	unsigned char UsbGetEPdInt(void)	
Function	Gets the EPd interrupt cause.	
Argument	None	
Return value	(The following values may be ORed.) USB_EP_INT_OUT_TRAN_ERR USB_EP_INT_IN_TRAN_ERR USB_EP_INT_OUT_TRAN_NAK USB_EP_INT_IN_TRAN_NAK USB_EP_INT_OUT_TRAN_ACK USB_EP_INT_IN_TRAN_ACK USB_EP_INT_OUT_SHORT_ACK	
(Description)	The EPd interrupt register value is returned.	

Clear EPd Interrupt Cause

Format	void UsbClearEPdInt(unsigned char event)	
Function	Clears the EPd interrupt cause.	
Argument	event -in	The interrupt cause (Can be ORed) USB_EP_INT_OUT_TRAN_ERR USB_EP_INT_IN_TRAN_ERR USB_EP_INT_OUT_TRAN_NAK USB_EP_INT_IN_TRAN_NAK USB_EP_INT_OUT_TRAN_ACK USB_EP_INT_IN_TRAN_ACK USB_EP_INT_OUT_SHORT_ACK
Return value	None	
(Description)	The specified EPd interrupt cause is cleared.	

3. SOFTWARE DESCRIPTION

Set Max Packet Size

Format	void UsbSetEndpointMaxSize(unsigned char number, unsigned short size)		
Function	Sets the maximum packet size at the specified endpoint.		
Argument	number	-in	Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
	size	-in	Maximum packet size
Return value	None		
(Description)	The specified maximum packet size is set for the specified endpoint.		

Get Max Packet Size

Format	unsigned short UsbGetEndpointMaxSize(unsigned char number)		
Function	Gets the maximum packet size at the specified endpoint.		
Argument	number	-in	Endpoint number USB_ENDPOINT_0 USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
Return value	The maximum packet size that has been set		
(Description)	The maximum packet size at the specified endpoint is returned.		

Set Endpoint Number

Format	void UsbSetEndpointNumber(unsigned char number, unsigned char setNumber)		
Function	Sets an endpoint number.		
Argument	number	-in	Endpoint number (USB IP endpoint number) USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
	setNumber	-in	Endpoint number (Endpoint number on protocol) to be set
Return value	None		
(Description)	The specified number is set for the specified endpoint.		

Get Endpoint Number

Format	unsigned char UsbGetEndpointNumber(unsigned char number)	
Function	Gets the endpoint number.	
Argument	number -in	Endpoint number (USB IP endpoint number) USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
Return value	The number that has been set for the specified endpoint	
(Description)	The number that has been set for the specified endpoint is returned.	

Set Endpoint Direction

Format	void UsbSetEndpointDirection(unsigned char number, unsigned char direction)	
Function	Sets an endpoint transfer direction.	
Argument	number -in	Endpoint number (USB IP endpoint number) USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
	direction -in	Data transfer direction USB_EP_IN USB_EP_OUT
Return value	None	
(Description)	The data transfer direction that has been specified for the specified endpoint is set.	

Get Endpoint Direction

Format	unsigned char UsbGetEndpointDirection(unsigned char number)	
Function	Gets the endpoint transfer direction.	
Argument	number -in	Endpoint number (USB IP endpoint number) USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
Return value	The data transfer direction at the endpoint that has been set	
	USB_EP_IN USB_EP_OUT	
(Description)	The data transfer direction at the endpoint that has been set is returned.	

3. SOFTWARE DESCRIPTION

Set Endpoint Mode

Format	void UsbSetEndpointMode(unsigned char number, unsigned char mode)	
Function	Sets the endpoint mode.	
Argument	Number -in mode -in	Endpoint number (USB IP endpoint number) USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D The mode set value USB_EP_ISO USB_EP_ISO_CRCMODE USB_EP_BULK_INTR
Return value	None	
(Description)	The specified mode is set for the specified endpoint.	

Get Endpoint Mode

Format	unsigned char UsbGetEndpointMode(unsigned char number)	
Function	Gets the endpoint mode.	
Argument	number -in	Endpoint number (USB IP endpoint number) USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
Return value	The mode of the endpoint that has been set USB_EP_ISO USB_EP_ISO_CRCMODE USB_EP_BULK_INTR	
(Description)	The mode of the endpoint that has been set is returned.	

Set Endpoint Toggle Mode

Format	void UsbSetEndpointToggleMode(unsigned char number, unsigned char mode)	
Function	Sets the endpoint toggle mode.	
Argument	number -in mode -in	Endpoint number (USB IP endpoint number) USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D The mode set value USB_EP_TOGGLE_NORMAL USB_EP_TOGGLE_ALWAYS
Return value	None	
(Description)	The specified mode is set for the specified endpoint.	

Get Endpoint Toggle Mode

Format	unsigned char UsbGetEndpointToggleMode(unsigned char number)		
Function	Gets the endpoint toggle mode.		
Argument	number	-in	Endpoint number (USB IP endpoint number) USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
Return value	The toggle mode of the endpoint that has been set USB_EP_TOGGLE_NORMAL USB_EP_TOGGLE_ALWAYS		
(Description)	The toggle mode of the endpoint that has been set is returned.		

Assign FIFO Area

Format	void UsbSetFIFOArea(unsigned short *map)												
Function	Assigns a FIFO area.												
Argument	map	-in	The FIFO assignment map										
			<table border="1"> <tr><td>[0]</td><td>Descriptor Area Top</td></tr> <tr><td>[1]</td><td>Endpoint-a Area Top</td></tr> <tr><td>[2]</td><td>Endpoint-b Area Top</td></tr> <tr><td>[3]</td><td>Endpoint-c Area Top</td></tr> <tr><td>[4]</td><td>Endpoint-d Area Top</td></tr> </table>	[0]	Descriptor Area Top	[1]	Endpoint-a Area Top	[2]	Endpoint-b Area Top	[3]	Endpoint-c Area Top	[4]	Endpoint-d Area Top
[0]	Descriptor Area Top												
[1]	Endpoint-a Area Top												
[2]	Endpoint-b Area Top												
[3]	Endpoint-c Area Top												
[4]	Endpoint-d Area Top												
Return value	None												
(Description)	The specified FIFO area is assigned and set.												

Get FIFO Area

Format	void UsbGetFIFOArea(unsigned short *map)												
Function	Gets the assigned FIFO area.												
Argument	map	-out	The FIFO assignment map										
			<table border="1"> <tr><td>[0]</td><td>Descriptor Area Top</td></tr> <tr><td>[1]</td><td>Endpoint-a Area Top</td></tr> <tr><td>[2]</td><td>Endpoint-b Area Top</td></tr> <tr><td>[3]</td><td>Endpoint-c Area Top</td></tr> <tr><td>[4]</td><td>Endpoint-d Area Top</td></tr> </table>	[0]	Descriptor Area Top	[1]	Endpoint-a Area Top	[2]	Endpoint-b Area Top	[3]	Endpoint-c Area Top	[4]	Endpoint-d Area Top
[0]	Descriptor Area Top												
[1]	Endpoint-a Area Top												
[2]	Endpoint-b Area Top												
[3]	Endpoint-c Area Top												
[4]	Endpoint-d Area Top												
Return value	None												
(Description)	The FIFO area assignment value that is currently set is returned.												

3. SOFTWARE DESCRIPTION

Write Descriptor Data

Format	void UsbWriteDescriptor(unsigned char *buffer, unsigned short offset, unsigned short length)		
Function	Writes the descriptor data.		
Argument	buffer	-in	Pointer to the descriptor storage data
	offset	-in	An offset of the destination storage
	length	-in	Data length
Return value	None		
(Description) Data is written in the FIFO descriptor area. “offset” shows an offset to the FIFO descriptor area.			

Read Descriptor Data

Format	void UsbReadDescriptor(unsigned char *buffer, unsigned short offset, unsigned short length)		
Function	Reads the descriptor data.		
Argument	buffer	-out	Pointer to the descriptor storage data
	offset	-in	An offset of the destination storage
	length	-in	Data length
Return value	None		
(Description) Data is read into the FIFO descriptor area. “offset” shows an offset to the FIFO descriptor area.			

Send Descriptor Data

Format	void UsbReplyDescriptor(unsigned short offset, unsigned short length)		
Function	Sends the descriptor data.		
Argument	Offset	-in	The data offset
	Length	-in	Data length
Return value	None		
(Description) The descriptor of FIFO storage identified by the “offset” is sent for the “length”.			

Get Remaining Descriptor Count

Format	unsigned short UsbRemainDescriptor(void)		
Function	Gets the remaining data count of descriptor.		
Argument	None		
Return value	The number of remaining bytes of the descriptor data		
(Description) The number of remaining bytes of the descriptor to be sent is returned.			

Get USB State

Format	unsigned char UsbGetUSBState(void)
Function	Gets the USB state.
Argument	None
Return value	USB_DEFAULT_STATE USB_ADDRESS_STATE USB_CONFIGURED_STATE
(Description)	The State value of the USB device is returned.

3.13.2 High-level USB driver API details

API List

Function	API	Description
USB Initialization	UsbInit	Initializes USB functions.
USB Bus Management	UsbBusManagement	Bus Management process
Control Process	UsbCtrlProcess	Control Process
Set Control Data Parameter	UsbSetCtrlParameter	Set the Control Transfer parameter.
Remaining Control Data	UsbCtrlRemainData	Remaining Control data
End Control Data Transfer	UsbCtrlFinish	Ends the Control data transfer.
Set EPx Data Parameter	UsbSetEPxParameter	Sets the EPx transfer parameter.
Remaining EPx Data	UsbEPxRemainData	Remaining EPx data
Set EPx State	UsbEPxSetState	Sets the EPx state.
Get EPx State	UsbEPxGetState	Gets the EPx state.
Clear Endpoint Interrupt Result	UsbEPxClearResult	Clears the endpoint interrupt result.
Get Endpoint Interrupt Result	UsbEPxGetResult	Gets the endpoint interrupt result.
Set Endpoint Data	UsbEPxDataSet	Writes data in FIFO memory of the specified endpoint.
Get Endpoint Data	UsbEPxDataGet	Reads data from FIFO memory of the specified endpoint.
Interrupt Handler	UsbInterruptHandler	Interrupt Handler
Check Interrupt Cause	UsbCheckInterrupt	Interrupt processing

3. SOFTWARE DESCRIPTION

USB Initialization

Format	int UsbInit(T_USB_CFG *Config)			
Function	Initializes USB functions.			
Argument	Config -in The USB initialization parameter			
Return value	SUCCESS: Normal termination ERROR : An error in argument			
(Description) The USB functions are initialized. Each endpoint is set automatically based on the descriptor data.				
<pre>typedef struct { unsigned char *descriptorData; // Descriptor storage pointer unsigned short descriptorSize; // Descriptor data size unsigned char negotiationMode; // Auto Negotiation mode // USB_NEGOTIATION_AUTO : Auto Negotiation // USB_NEGOTIATION_MANUAL : Manual Negotiation unsigned char addressMode; // Automatic Address Return mode // USB_ADDRESS_AUTO : Automatic Address Return // USB_ADDRESS_MANUAL : Manual Address Return unsigned char managementMode; // USB Bus Process mode // USB_MANAGEMENT_AUTO : Auto Management // USB_MANAGEMENT_MANUAL : Manual Management void (*activeFunc)(void); // Clock supply process function to USB memory void (*snoozeFunc)(void); // Clock stop process function to USB int (*remoteFunc)(void); // RemoteWakeup process function int (*resetFunc)(void); // External process function at USB BusReset time int (*classFunc)(T_USB_REQUEST_PACKET*); // Class Request process function int (*vendorFunc)(T_USB_REQUEST_PACKET*); // Vendor Request process function void (*intEnableFunc)(void); // Enable USB Interrupt function void (*intDisableFunc)(void); // Disable USB Interrupt function } T_USB_CFG;</pre>				

USB Bus Management

Format	void UsbBusManagement(void)	
Function	Executes the Bus Management process.	
Argument	None	
Return value	None	
(Description) The USB bus state change process is executed simultaneously. This is valid if the ManagementMode of config has been set to USB_MANAGEMENT_AUTO. The VBUS ON/OFF, Bus-Reset, Suspend, and Resume processes are executed simultaneously.		

Control Process

Format	void UsbCtrlProcess(void)
Function	Executes the Control process.
Argument	None
Return value	None
(Description) The data send/receive process is executed at endpoint 0. The process at endpoint 0 is executed as the State Machine process. If USB_MANAGEMENT_AUTO is not set, this function needs to be called at fixed intervals. Prepare the process functions in the host application for Class Request/Vendor Request, and register the process functions in the UsbInit config.classFunc and config.vendorFunc. If not implemented, the Dummy functions needs to be prepared and set.	

Set Control Data Parameter

Format	void UsbSetCtrlParameter(unsigned char *buffer, unsigned short length)		
Function	Sets the Control Transfer parameter.		
Argument	buffer	-in	Data storage pointer
	length	-in	Data size
Return value	None		
(Description) The data storage pointer and the data size are set for any data transfer at endpoint 0. The area indicated as the buffer must be a multiple of the MaxPacketSize (64 bytes), and it must be less than 64K bytes. If data is divided and sent and if the data length that cannot be divided by 64 bytes is set, this data is sent as the ShortPacket.			

Remaining Control data

Format	int UsbCtrlRemainData(unsigned short *remain)		
Function	Returns the remaining control data length.		
Argument	remain -out The remaining data length		
Return value	SUCCESS: Normal termination ERROR: An error occurrence during processing		
(Description) The remaining data length at endpoint 0 is returned during data transfer. If the ERROR is returned, an error has occurred during the process. If the SUCCESS is returned, "remain" indicates the remaining data length. When data transfer is completed, specify the SUCCESS for UsbCtrlFinish and call the function. If this API is terminated with the ERROR, specify the ERROR for UsbCtrlFinish and call the function.			

End Control Data Transfer

Format	void UsbCtrlFinish(int end)		
Function	Ends the Control data transfer.		
Argument	end	-in	Termination cause SUCCESS ERROR
Return value	None		
(Description) The Status process is executed.			

3. SOFTWARE DESCRIPTION

Set EPx Data Parameter

Format	void UsbSetEPxParameter(unsigned char number, unsigned char *buffer, unsigned short length)		
Function	Sets the EPx transfer parameter.		
Argument	number	-in	Endpoint number USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
	buffer	-in	Data storage pointer
	length	-in	Data size
Return value	None		
(Description)	The data storage pointer and the data size are set for data transfer at the specified endpoint. The area indicated as the buffer must be a multiple of the MaxPacketSize (64 bytes), and it must be less than 64K bytes. If data is divided and sent and if the data length that cannot be divided by 64 bytes is set, this data is sent as the ShortPacket.		

Remaining EPx data

Format	unsigned short UsbEPxRemainData(unsigned char number)		
Function	Returns the remaining EPx data length.		
Argument	number	-in	Endpoint number USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
Return value	The remaining transfer data length		
(Description)	The remaining data length at the specified endpoint is returned during data transfer. When value 0 is returned, the data transfer has completed.		

Set EPx State

Format	void UsbEPxSetState(unsigned char number, unsigned char state)		
Function	Sets the EPx state.		
Argument	number	-in	Endpoint number USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
	state	-in	The change state USB_EPX_DATA_SET USB_EPX_DATA_SEND USB_EPX_DATA_GET USB_EPX_DATA_RCV USB_EPX_IDLE USB_EPX_STALL USB_EPX_RESET
Return value	None		
(Description)	The specified endpoint is changed into the specified state.		

Get EPx State

Format	unsigned char UsbEPxGetState(unsigned char number)		
Function	Gets the EPx state.		
Argument	number	-in	Endpoint number USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
Return value	USB_EPX_DATA_SET USB_EPX_DATA_SEND USB_EPX_DATA_GET USB_EPX_DATA_RCV USB_EPX_IDLE USB_EPX_STALL (It is ORed with another value.) USB_EPX_RESET USB_EPX_FEATURE		
(Description)	The state of the specified endpoint is returned. If the read result is FEATURE, the STALL has been released by the ClearFeature. Use this result to determine postprocessing such as Status reply based on the protocol specifications. To determine if the state is STALL or not: if(UsbEPxGetState(USB_ENDPOINT_A) & USB_EPX_STALL){ Should be used to check it by ANDing as above.		

3. SOFTWARE DESCRIPTION

Clear Endpoint Interrupt Result

Format	void UsbEPxClearResult(unsigned char number)		
Function	Clears the endpoint interrupt result.		
Argument	number	-in	Endpoint number USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
Return value	None		
(Description)	The interrupt result of the specified endpoint is cleared.		

Get Endpoint Interrupt Result

Format	unsigned char UsbEPxGetResult(unsigned char number)		
Function	Gets the endpoint interrupt result.		
Argument	number	-in	Endpoint number USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
Return value	No result : USB_EPX_NO_RESULT Error at IN side : USB_EPX_IN_ERR Error at OUT side : USB_EPX_OUT_ERR ACK at IN side : USB_EPX_IN_ACK ACK at OUT side : USB_EPX_OUT_ACK		
(Description)	The interrupt result of the specified endpoint is obtained.		

Set Endpoint Data

Format	void UsbEPxDataSet(unsigned char number)		
Function	Writes data in FIFO memory of the specified endpoint.		
Argument	number	-in	Endpoint number USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
Return value	None		
(Description)	Data is read from the buffer being set by the UsbSetEPxParameter, and it is written in FIFO memory of the specified endpoint.		

Get Endpoint Data

Format	void UsbEPxDataGet(unsigned char number)		
Function	Reads data into FIFO memory of the specified endpoint.		
Argument	number	-in	Endpoint number USB_ENDPOINT_A USB_ENDPOINT_B USB_ENDPOINT_C USB_ENDPOINT_D
Return value	None		
(Description)	Data is read from FIFO memory of the specified endpoint, and it is written in the buffer being set by the UsbSetEPxParameter.		

Interrupt Handler

Format	void UsbInterruptHandler(void)		
Function	Interrupt handler		
Argument	None		
Return value	None		
(Description)	The USB IP interrupt cause is held in the internal variables, and the interrupt cause is cleared. This API is used for registration in the interrupt handler.		

Check Interrupt Cause

Format	void UsbCheckInterrupt(void)		
Function	Interrupt processing		
Argument	None		
Return value	None		
(Description)	The interrupt cause being held by the UsbInterruptHandler is closely checked. The result is set in the variables for internal processing. This API needs to be called before other processing (such as UsbBusManagement and UsbCtrlProcess).		

3. SOFTWARE DESCRIPTION

3.14 HEADER DEFINITIONS

The tables below show the definitions used in the driver functions.

Definition name	Value	Description
USB_NEGOTIATION_MANUAL	0	Manual Negotiation
USB_NEGOTIATION_AUTO	1	Auto Negotiation
USB_ADDRESS_MANUAL	0	Manual Address Reply
USB_ADDRESS_AUTO	1	Auto Address Reply
USB_MANAGEMENT_MANUAL	0	Manual Bus Management
USB_MANAGEMENT_AUTO	1	Auto Bus Management
USB_VBUS_ON	0x80	VBUS On (reg_801.h: USB_USBStatus_VBUS)
USB_VBUS_OFF	0x00	VBUS Off
USB_LINE_SE0	0x00	SE0 State
USB_LINE_J	0x01	J State
USB_LINE_K	0x02	K State
USB_LINE_SE1	0x03	SE1 State
USB_ENDPOINT_0	0	Endpoint 0
USB_ENDPOINT_A	1	Endpoint a
USB_ENDPOINT_B	2	Endpoint b
USB_ENDPOINT_C	3	Endpoint c
USB_ENDPOINT_D	4	Endpoint d
USB_TESTMODE_J	0x04	Test J Mode (reg_801.h: USB_USBTest_TestJ)
USB_TESTMODE_K	0x02	Test K Mode (reg_801.h: USB_USBTest_TestK)
USB_TESTMODE_SE0_NAK	0x08	Test SE0 NAK Mode (reg_801.h: USB_USBTest_TestSE0NAK)
USB_TESTMODE_PACKET	0x01	Test Packet Mode (reg_801.h: USB_USBTest_TestPacket)
USB_AUTOADDRESS_ON	0x80	Auto Address ON (reg_801.h: USB_USBAddress_AutoSetAddress)
USB_AUTOADDRESS_OFF	0	Auto Address OFF
USB_OPMODE_NORMAL	0x00	Normal
USB_OPMODE_NONDRIVE	0x01	Non-Drive
USB_OPMODE_DISBITSTUFF	0x02	Disable Bitstuffing and NRZI Encoding
USB_OPMODE_RESERVED	0x03	
USB_EP0_IN	0x80	IN (reg_801.h: USB_EP0Control_INxOUT)
USB_EP0_OUT	0	OUT
USB_EP_IN	0x80	IN
USB_EP_OUT	0x00	OUT
USB_EP_ISO	0x80	Isochronous Mode
USB_EP_ISO_CRCMODE	0x40	Isochronous CRC Mode
USB_EP_BULK_INTR	0x00	Bulk or Interrupt
USB_EP_TOGGLE_NORMAL	0x00	Normal Toggle Mode
USB_EP_TOGGLE_ALWAYS	0x40	Always Toggle Mode
USB_EP_CTL_STALL	0x01	Force STALL
USB_EP_CTL_NAK	0x02	Force NAK
USB_EP_CTL_TOGGLE_CLR	0x04	Toggle Clear
USB_EP_CTL_TOGGLE_SET	0x08	Toggle Set
USB_EP_CTL_DISAFNAK	0x20	Disable Auto Force NAK when Short Packet
USB_EP_CTL_ENSHORTPKT	0x40	Enable Short Packet
USB_EP_CTL_AUTOFORCENAK	0x80	Auto Force NAK
USB_EP_CTL_TOGGLE_0	0x00	Toggle State 0
USB_EP_CTL_TOGGLE_1	0x10	Toggle State 1
USB_DEFAULT_STATE	(0x00)	
USB_ADDRESS_STATE	(0x01)	
USB_CONFIGURED_STATE	(0x02)	
USB_INT_ENABLE	0x00	Enable USB Interrupt
USB_INT_DISABLE	0x01	Disable USB Interrupt

3.15 Compile Options

This sample program can enable/disable unused functions using the compiling option (or Symbol definition). As the unused functions are disabled by default, it should be noted during compiling.

Symbol List

Symbol name	Description
FOR_CLEAR_FEATURE	Enables ClearFeature of the Standard request.
FOR_SET_FEATURE	Enables SetFeature of the Standard request.
FOR_GET_INTERFACE	Enables GetInterface of the Standard request.
FOR_SET_INTERFACE	Enables SetInterface of the Standard request.
FOR_CLASS_REQUEST	Enables the Class request.
FOR_TEST_MODE	Enables TestMode.
FOR_BULK_TRANSFER	Enables the Bulk Transfer process.
FOR_NEGOTIATION	Enables the manual negotiation process.
FOR_ADDRESS	Enables the manual SetAddress process.
FOR_MANAGEMENT	Enables the manual management of USB processing.
FOR_REMOTEWAKEUP	Enables the RemoteWakeup function.
FOR_SNOOZE_OFF	Enables the Snooze OFF function.
FOR_USB_MINIMUM_API	Makes the minimum required API configuration.

REVISION HISTORY

REVISION HISTORY

AMERICA

EPSON ELECTRONICS AMERICA, INC.

HEADQUARTERS

2580 Orchard Parkway
San Jose , CA 95131,USA
Phone: +1-800-228-3964 FAX: +1-408-922-0238

SALES OFFICES

Northeast

301 Edgewater Place, Suite 210
Wakefield, MA 01880, U.S.A.
Phone: +1-800-922-7667 FAX: +1-781-246-5443

EUROPE

EPSON EUROPE ELECTRONICS GmbH

HEADQUARTERS

Riesstrasse 15 Muenchen Bayern,
80992 GERMANY
Phone: +49-89-14005-0 FAX: +49-89-14005-110

ASIA

EPSON (CHINA) CO., LTD.

7F, Jinbao Bldg., No.89 Jinbao St.,
Dongcheng District,
Beijing 100005, China
Phone: +86-10-6410-6655 FAX: +86-10-6410-7320

SHANGHAI BRANCH

7F, Block B, Hi-Tech Bldg., 900, Yishan Road,
Shanghai 200233, CHINA
Phone: +86-21-5423-5522 FAX: +86-21-5423-5512

EPSON HONG KONG LTD.

20/F., Harbour Centre, 25 Harbour Road
Wanchai, Hong Kong
Phone: +852-2585-4600 FAX: +852-2827-4346
Telex: 65542 EPSCO HX

EPSON (CHINA) CO., LTD.

SHENZHEN BRANCH
12/F, Dawning Mansion, Keji South 12th Road,
Hi-Tech Park, Shenzhen
Phone: +86-755-2699-3828 FAX: +86-755-2699-3838

EPSON TAIWAN TECHNOLOGY & TRADING LTD.

14F, No. 7, Song Ren Road,
Taipei 110
Phone: +886-2-8786-6688 FAX: +886-2-8786-6660

EPSON SINGAPORE PTE., LTD.

1 HarbourFront Place,
#03-02 HarbourFront Tower One, Singapore 098633
Phone: +65-6586-5500 FAX: +65-6271-3182

SEIKO EPSON CORPORATION

KOREA OFFICE

50F, KLI 63 Bldg., 60 Yoido-dong
Youngdeungpo-Ku, Seoul, 150-763, KOREA
Phone: +82-2-784-6027 FAX: +82-2-767-3677

GUMI OFFICE

2F, Grand B/D, 457-4 Songjeong-dong,
Gumi-City, KOREA
Phone: +82-54-454-6027 FAX: +82-54-454-6093

SEIKO EPSON CORPORATION SEMICONDUCTOR OPERATIONS DIVISION

IC Sales Dept.

IC International Sales Group

421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-42-587-5814 FAX: +81-42-587-5117