APPLICATION NOTE

V850E2/ML4

USB Host Software

R01AN1217EJ0102
Rev.1.02
Jan. 10, 2013

Summary

This application note provides an example of controlling a USB function by using the USB host controller.

Operation-verified devices

- V850E2/ML4 group (product name: μPD70F4022)

Target board

- V850E2/ML4 CPU board (model: R0K0F4022C000BR)

Cautions

- Multiple USB devices can be controlled. The number of supported USB devices is up to 3 devices include hub device.
- The hub layer is one layer (except for root hub) and maximum number of hub port is 7 ports.
- Bulk Only Transport of the mass storage class, Abstract Control Model of the communication class and hub class are supported.

Contents

1. Preface

1.1 Specifications

- Multiple USB devices can be controlled. The number of supported USB devices is up to 3 devices include hub device .
- The hub layer is one layer (except for root hub) and whose number of port is up to 7 ports..
- Bulk Only Transport of the mass storage class, Abstract Control Model of the communication class and hub class are supported.
- If another class is required, you will need to implement that class yourself.
- The maximum capacity of the memory (USB memory, SD memory card, etc.) that can be mounted is 32 GB.
- The following operations can be performed on the memories, V850E2/ML4 USB Function sample software[1] and SH7216 USB Function sample software[4] by using the file control application:
  - Creation of 10 files in a root directory.
  - Creation of another directory and creation of 10 files in that directory.
  - Writing, reading, and verifying 1 to 40 blocks in block (512-byte) units.
  - Writing, reading, and verifying 1 to 1024 bytes in byte units.
- FAT file system is used M3S-TFAT-Tiny that is Renesas original one.
- The communication class operations are checked only to V850E2/ML4 USB Function sample software[2].
- The following operations can be performed on the V850E2/ML4 USB Function sample software[2] by using the communication class sample application:
  - Set transfer rate(bps), stop bits, parity, data bits, etc.
  - Get transfer rate(bps), stop bits, parity, data bits, etc.
  - Writing, reading, and verifying 1 to 64 bytes in byte units.
- The communication class operations are checked only to V850E2/ML4 USB Function sample software[2].
- Support multiple interface (up to 3 interface). This software can also control V850E2/ML4 USB Function Multifunction Operation sample software[3] support mass storage class and communication class.

1.2 Features Used

- H bus
- USB Host Controller
- Interrupts

1.3 Applicable Conditions

Microcontroller: V850E2/ML4 (μPD70F4022)
Evaluation board: V850E2/ML4 CPU board (model: R0K0F4022C000BR)
Operating frequency:
  Input clock: 10 MHz
  Internal system clock (f_{CLK}): 200 MHz
  MII transmission clock (f_{MIITX}): 25 MHz
  MII reception clock (f_{MIIRX}): 25 MHz
  H bus clock (f_{HCLK}): 33.3 MHz
Operating mode: Normal operating mode
Integrated development environment: CubeSuite+ V1.02.00 [12 Apr 2012] from Renesas Electronics
C compiler: CX V1.21 from Renesas Electronics
Emulator: EI emulator from Renesas Electronics

1.4 Related Application Note

[1] V850E2/ML4 Microcontrollers Example of USB Function Mass Storage Class Application Note (to be published)

[2] V850E2/ML4 Microcontrollers Example of USB Communication Class Function Application Note (to be published)

[3] V850E2/ML4 Microcontrollers Example of USB Function Multifunction Operation Application Note (R01AN1037EJ)

[4] SH7216 Group USB Function Module: USB Mass Storage Class Application Note (REJ06B0897)

[5] V850E2M FAT File System Software M3S-TFAT-Tiny: Introduction Guide (R01AN1028EJ)

2. Correlation Between Functions

The figure below shows the correlation between the functions. Each function is described in detail in section 3 and subsequent sections.

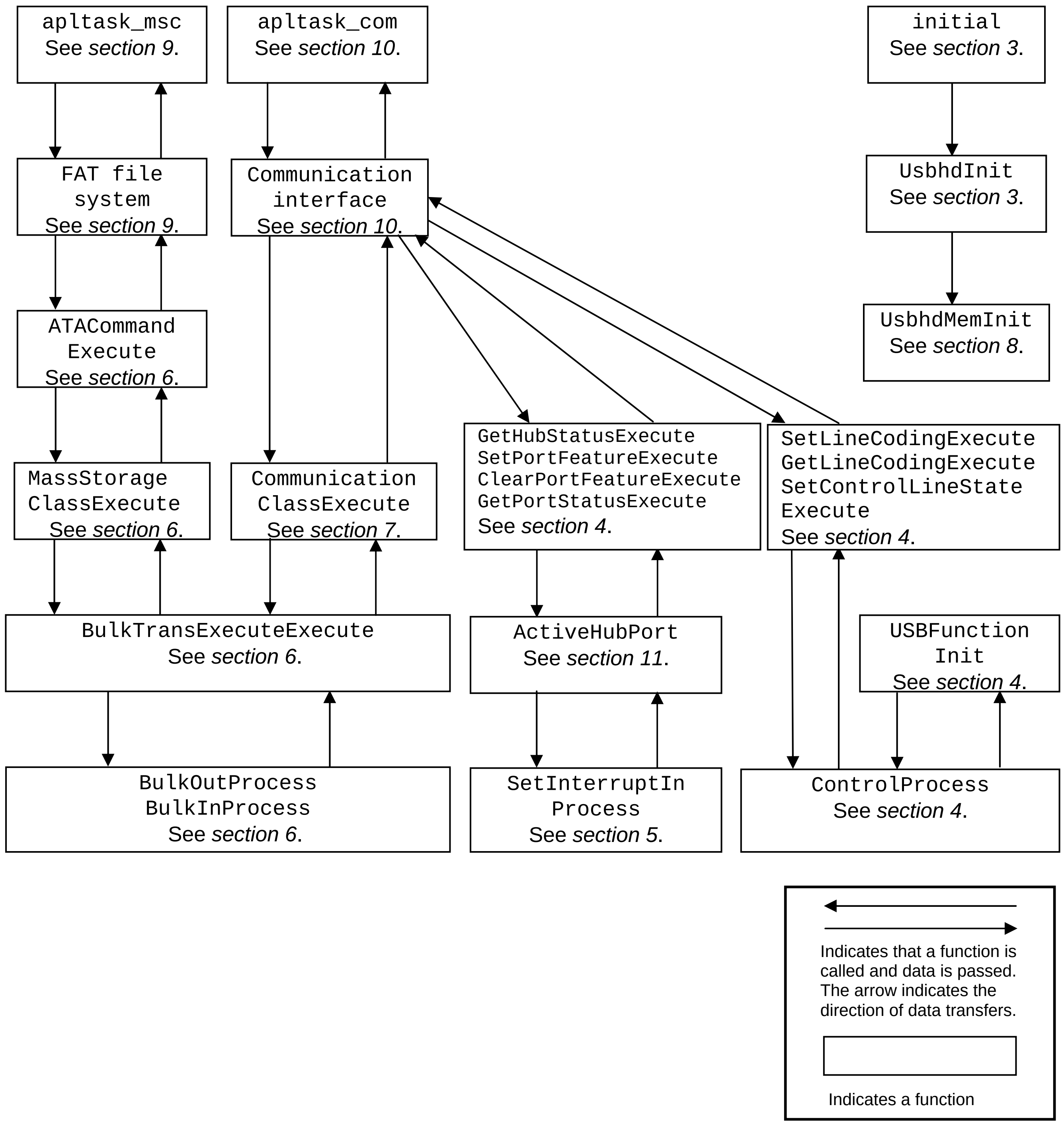

Figure 1. Correlation Between Functions

3. Initialization

This section provides a diagram indicating the correlation between the initialization functions, describes the function specifications and operations.

3.1 Correlation Between Initialization Functions

The `initial` function calls each function. Each function initializes the corresponding peripheral module.

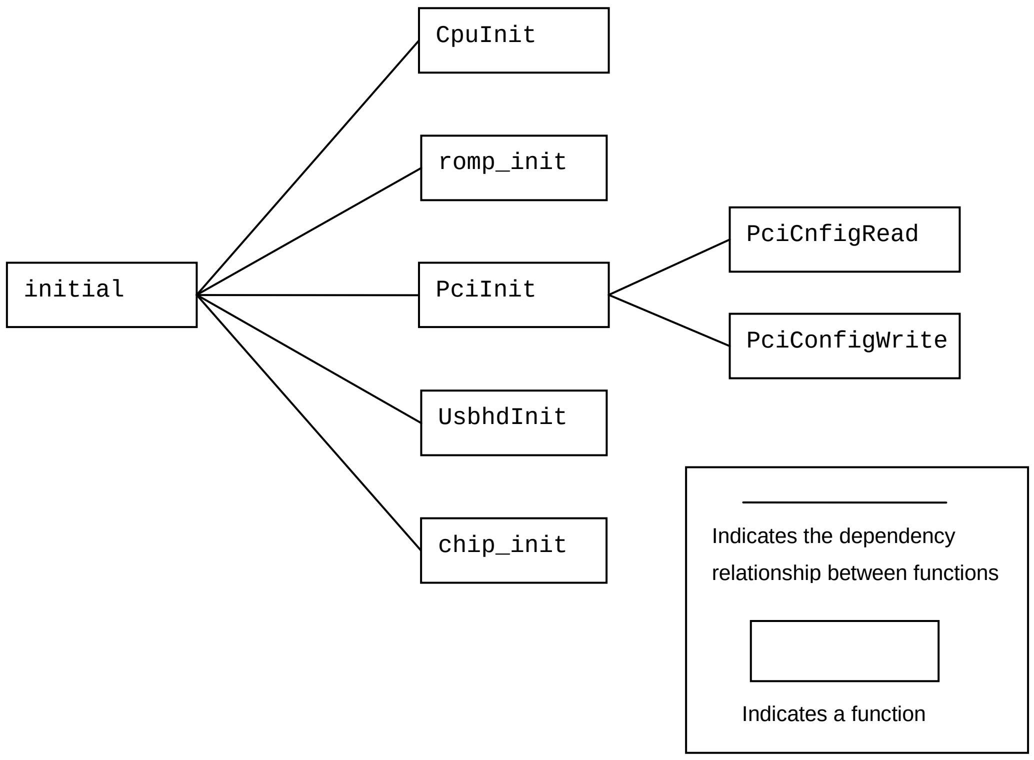

Figure 2. Correlation Between Initialization Functions

3.2 Description of Functions

The following shows the function names and processing details.

Function	Description
`initial`	This function is called when the CPU is turned on and calls various initialization functions. The actual hardware initialization is executed by each *xxx*`Init` function; this function only calls the processing.
`CpuInit`	This function initializes the CPU incorporated in the V850E2/ML4.
`romp_init`	ROMization package initialization processing This function calls the copy function (`_rcopy`), which copies the information stored at the specified address to the RAM area byte by byte.
`PciInit`	This function initializes the PCI bridge to access USB host controller registers.
`PciConfigRead`	This function is used to read the base address of the USB host driver registers via the PCI configuration space.
`PciConfigWrite`	This function is used to write the base address of the USB host driver registers via the PCI configuration space.
`UsbhdInit`	This function initializes the USB host controller. This function must be called after `PciInit` processing because the PCI bridge is used to access the USB host controller registers.
`UsbhdMemInit`	This function initializes the global memory used by the EDs (endpoint descriptors) and TDs (transfer descriptors).
`chip_init`	This function performs initial setup of ports.

3.3 Function Interfaces

The following shows the interface of each function.

Function	`initial`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	void
Return value usage	–
Return value meaning	–

Function	`CpuInit`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	void
Return value usage	–
Return value meaning	–

Function	`romp_init`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	void
Return value usage	–
Return value meaning	–

Function	`PciInit`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	void
Return value usage	–
Return value meaning	–

Function	`PciConfigRead`	
Parameter	`RegisterNumber`	`ReadData`
Parameter information	PCI configuration register number	Data read out
Specifiable value	0x10	32-bit data
Return value	Status	
Return value usage	Used to judge whether execution of `PciConfigRead` finished successfully.	
Return value meaning	0: Normal completion 1: Abnormal termination	

Function	`PciConfigWrite`	
Parameter	`RegisterNumber`	`WriteData`
Parameter information	PCI configuration register number	USB host controller base address
Specifiable value	0x10	32-bit data
Return value	Status	
Return value usage	Used to judge whether execution of `PciConfigWrite` finished successfully.	
Return value meaning	0: Normal completion 1: Abnormal termination	

Function	`UsbhdInit`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	void
Return value usage	–
Return value meaning	–

Function	`UsbhdMemInit`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	STATUS
Return value usage	Used to judge whether execution of `UsbhdMemInit` finished successfully.
Return value meaning	0: Normal completion (no error reported)

Function	`chip_init`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	void
Return value usage	–
Return value meaning	–

3.4 Initialization Operation

The figure below shows the functions during initialization. Each function is called from the `initial` function. Because these functions return no value (`void`), the figure below omits the returned message.

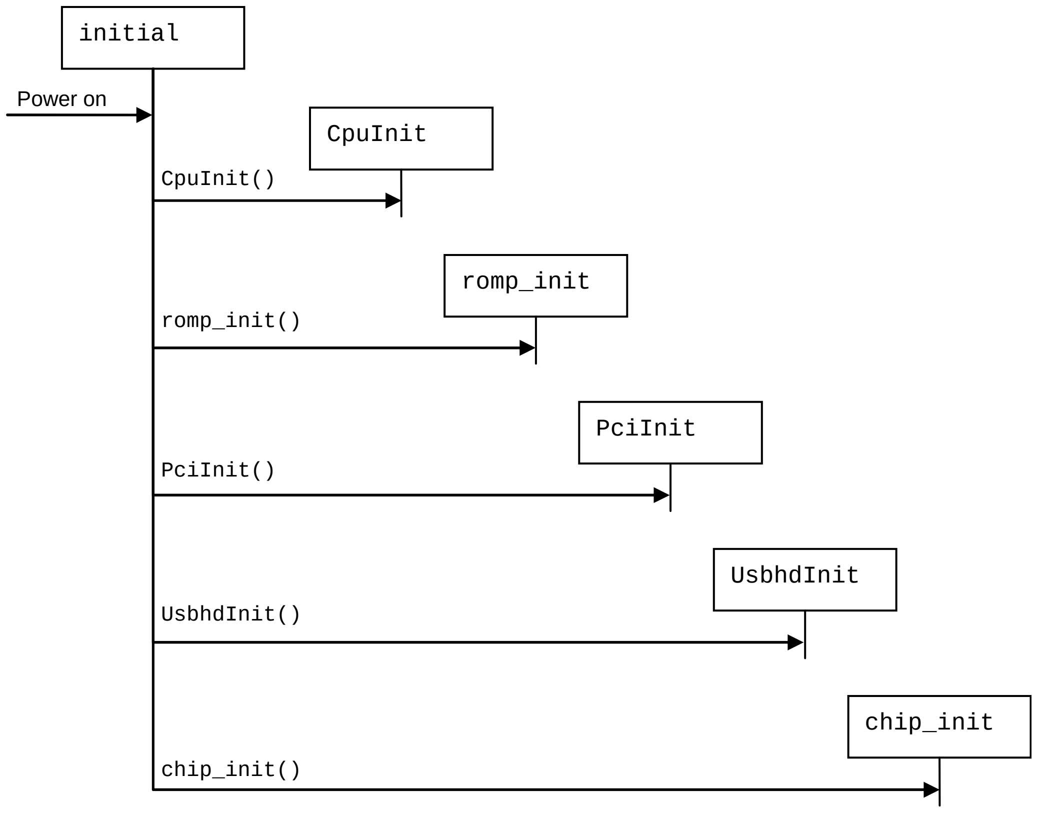

Figure 3. Initialization Function Behavior

4. Control Transfer

This section provides a diagram indicating the correlation between the control transfer functions, describes the function specifications and operations in the control transfer.

4.1 Correlation Between Control Transfer Functions

When the 1st USB device is connected or the user application accesses 3rd USB device connected to the hub's downstream port, the `UsbFunctionInit` is executed and then each function is executed according to the enumeration sequence.
The hub class commands are executed to get the hub status, reset and enable the hub's downstream ports, get the status of the hub's downstream ports when the hub enumeration sequence is completed or waiting the detection to a device on the hub's downstream port.
The mass storage class commands are executed when the USB mass storage device is initialized after enumeration sequence.
The communication class commands are executed by the demand from the communication class layer.

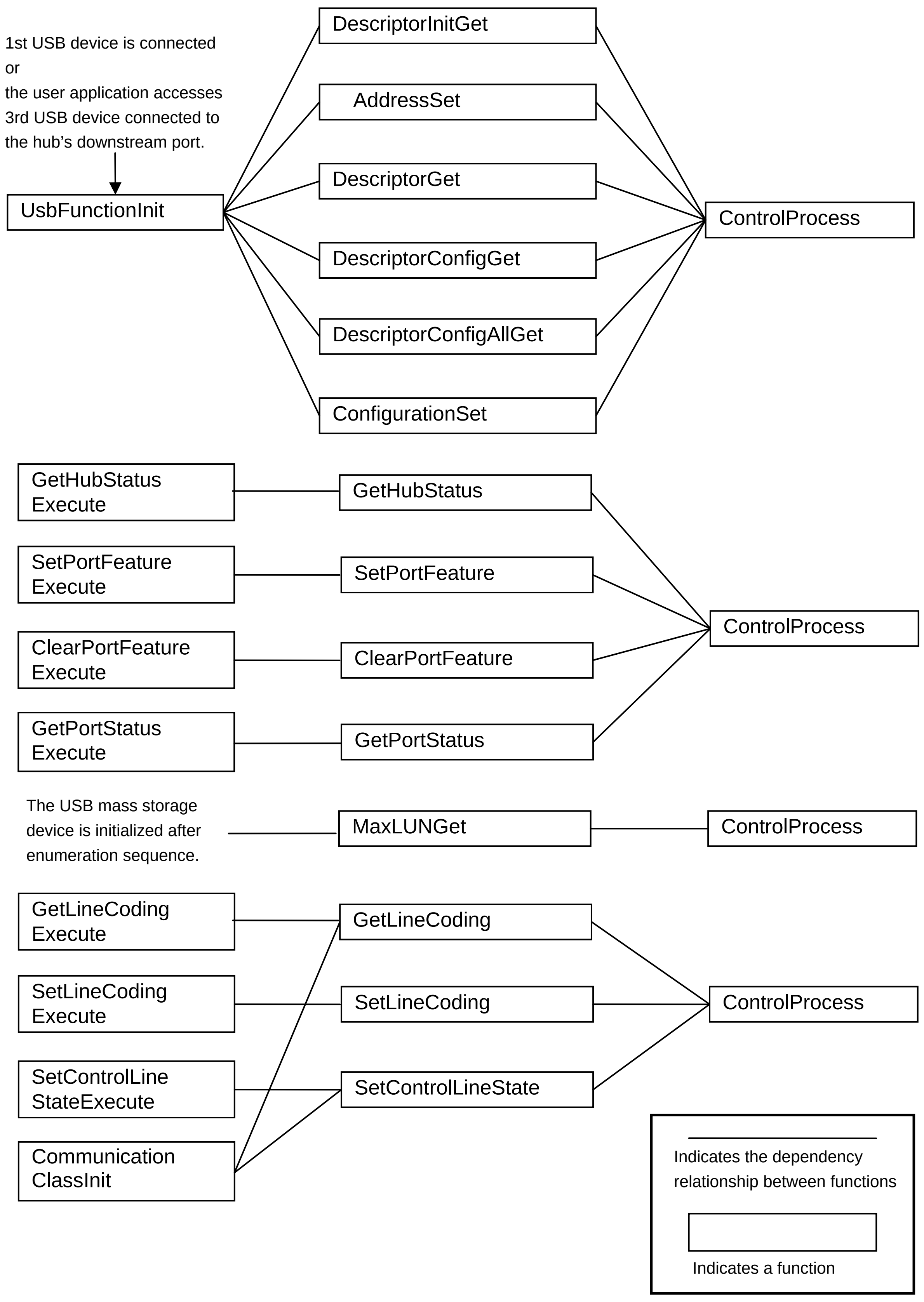

Figure 4. Correlation Between Control Transfer Functions

4.2 Description of Functions

The following shows the function names and processing details.

Function	Description
`UsbFunctionInit`	This function is called when the 1st USB device is connected or the user application accesses 3rd USB device connected to the hub's downstream port. This function calls functions that assign the address to the USB device and acquire descriptor information. 1. Acquires standard device descriptor (device) and set a maximum packet size of EP0 to a driver structure (USBHD_DEV_INFO). 2. Assigns a address to USB device. 3. Acquires standard device descriptor (device) 4. Acquires standard device descriptor (configuration) and set the length of standard device descriptor (configuration), the number of interface and configuration to the driver structure. 5. Acquires all of the standard device descriptor (configuration) and set the endpoint information for all interface (up to 3) to the driver structure. 6. Finishes the enumeration sequence of the USB device by issuing `Set Configuration`. 7. Waits the detection to a device on the hub's downstream port. If the device detected, make enable the downstream port.
`ControlProcess`	This function is called during setup data transfer processing and performs the following: 1. Acquires a transmission/reception ED, data stage TD, and a TD area for the status stage. 2. Specifies various parameters for the ED and 8-byte standard device request data for the current buffer pointer of the setup stage TD. 3. Specifies a data reception buffer for the data stage TD, if there is a data stage and the direction is in (function to host). 4. Specifies a data transmit buffer for the data stage TD, if there is a data stage and the direction is out (host to function). 5. Links the ED and TD, sets the ED address to the ControlHeadED register, writes 0x02 to the HcCommandStatus register, and then starts control transfer. 6. After transfer finishes, checks the `Done` queue to confirm that the transfer was executed successfully.
`DescriptorInitGet`	This function acquires a standard device descriptor (device) of 64 bytes or less from the USB device before assigning an address to the USB device. 1. Creates 8-byte standard device request data and assigns `Get Descriptor` (0x06) to the request. 2. Calls `ControlProcess` to execute transfer processing.
`AddressSet`	This function assigns an address to the USB device. 1. Creates 8-byte standard device request data and assigns `Set Address` (0x05) to the request. 2. Calls `ControlProcess` to execute transfer processing.

Function	Description
`DescriptorGet`	This function acquires a standard device descriptor (device) from the USB device. 1. Creates 8-byte standard device request data and assigns `Get Descriptor` (0x06) to the request. 2. Calls `ControlProcess` to execute transfer processing.
`DescriptorConfigGet`	This function acquires a standard device descriptor (configuration) from the USB device (9 bytes). 1. Creates 8-byte standard device request data and assigns `Get Descriptor` (0x06) to the request. 2. Calls `ControlProcess` to execute transfer processing.
`DescriptorConfigAllGet`	This function acquires all standard device (configuration) descriptors from the USB device. 1. Creates 8-byte standard device request data and assigns `Get Descriptor` (0x06) to the request. 2. Calls `ControlProcess` to execute transfer processing.
`GetHubDescriptor`	This function acquires hub descriptors from the USB device. 1. Creates 8-byte class specific request data and assigns `Get Descriptor` (0x06) to the request. 2. Calls `ControlProcess` to execute transfer processing.
`DeviceStatusGet`	This function acquires the device status from the USB device. 1. Creates 8-byte standard device request data and assigns `Get Status Request` (0x00) to the request. 2. Calls `ControlProcess` to execute transfer processing.
`ConfigurationSet`	This function issues `Set Configuration` to the USB device. 1. Creates 8-byte standard device request data and assigns `Set Configuration` (0x09) to the request. 2. Calls `ControlProcess` to execute transfer processing.
`GetHubStatus`	This function acquires the hub status from the USB device. 1. Creates 8-byte class specific request data and assigns `Get Status Request` (0x00) to the request. 2. Calls `ControlProcess` to execute transfer processing.
`SetPortFeature`	This function issues `Set Port Feature` to the USB device. 1. Creates 8-byte class specific request data, assigns `Set Feature Request` (0x03) to the request and controlled port number to the index. 2. Calls `ControlProcess` to execute transfer processing.
`ClearPortFeature`	This function issues `Set Port Feature` to the USB device. 1. Creates 8-byte class specific request data, assigns `Clear Feature Request` (0x01) to the request and controlled port number to the index. 2. Calls `ControlProcess` to execute transfer processing.
`GetPortStatus`	This function issues `Get Port Status` to the USB device. 1. Creates 8-byte class specific request data, assigns `Get Status Request` (0x00) to the request and controlled port number to the index. 2. Calls `ControlProcess` to execute transfer processing.
`MaxLUNGet`	This function issues `Max LUN Get` to the USB device. 1. Creates 8-byte class specific request data and assigns `Max LUN Get` (0xFE) to the request. 2. Calls `ControlProcess` to execute transfer processing.

Function	Description
`SetLineCoding`	This function issues `Set Line Coding` to the USB device. 1. Creates 8-byte class specific request data and assigns `Set Line Coding` (0x20) to the request. 2. Calls `ControlProcess` to execute transfer processing.
`GetLineCoding`	This function issues `Get Line Coding` to the USB device. 1. Creates 8-byte class specific request data and assigns `Get Line Coding` (0x21) to the request. 2. Calls `ControlProcess` to execute transfer processing.
`SetControlLineState`	This function issues `Set Control Line State` to the USB device. 1. Creates 8-byte class specific request data and assigns `Set Control Line State` (0x22) to the request. 2. Calls `ControlProcess` to execute transfer processing.

4.3 Function Interfaces

The following shows the interface of each function. For details about Driver structure(=`USBDevInfo`), see *section 14*.

Function	`UsbFunctionInit`	
Parameter	`UsbDevInfo`	`dev_num`
Parameter information	Driver structure (USB device information, Transfer information)	USB Device number
Specifiable value	Other than NULL	0, 1, 2
Return value	STATUS	
Return value usage	Used to judge whether execution of `UsbFunctionInit` finished successfully.	
Return value meaning	0: Normal completion	
	Other than 0: Abnormal termination	

Function	`ControlProcess`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `ControlProcess` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`DescriptorInitGet`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `DescriptorInitGet` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`AddressSet`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `AddressSet` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`DescriptorGet`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `DescriptorGet` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`DescriptorConfigGet`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `DescriptorConfigGet` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`DescriptorConfigAllGet`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `DescriptorConfigAllGet` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`GetHubDescriptor`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `GetHubDescriptor` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`DeviceStatusGet`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `DeviceStatusGet` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`ConfigurationSet`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `ConfigurationSet` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`GetHubStatus`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `GetHubStatus` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`SetPortFeature`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `SetPortFeature` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`ClearPortFeature`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `ClearPortFeature` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`GetPortStatus`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `GetPortStatus` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`MaxLUNGet`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `MaxLUNGet` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`SetLineCoding`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `SetLineCoding` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`GetLineCoding`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `GetLineCoding` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`SetControlLineState`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `SetControlLineState` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

4.4 Control Transfer Operation

This section describes the dynamic behavior of the functions during control transfer.
There are two cases for the control transfer. Those cases are the enumeration sequence operation when a USB device is connected and the class command sequence operation.

(1) Enumeration

The `UsbFunctionInit` function is executed when the 1st USB device is connected or the user application accesses 3rd USB device connected to the hub's downstream port. For details about the interrupt that occurs when the USB device is connected, see *section 13*.

Figure 5 shows the enumeration sequence.

Acquires the maximum packet size of the USB device by the `DescriptorInitGet` function. Hereafter, the address of the USB device is assigned by the `AddressSet` function. Functions such as the `DescriptorGet` function are used to acquire information from the USB device at this address.
If the USB device is hub device, acquires hub descriptor from the USB device at this address.
The acquired information is stored in `USBDevInfo`.

(2) Class commands

Figure 6 shows the communication class command sequence for `SetLineCoding` and `GetLineCoding` as typical example.
Set communication parameter such as transfer rate, data length and so on to USB device by the `SetLineCoding` when the application requires it.
Get communication parameter such as transfer rate, data length and so on from USB device by the `GetLineCoding` when the application requires it.
Skip sequences of the class commands belong to the hub, mass storage and communication class because those behavior are almost same.

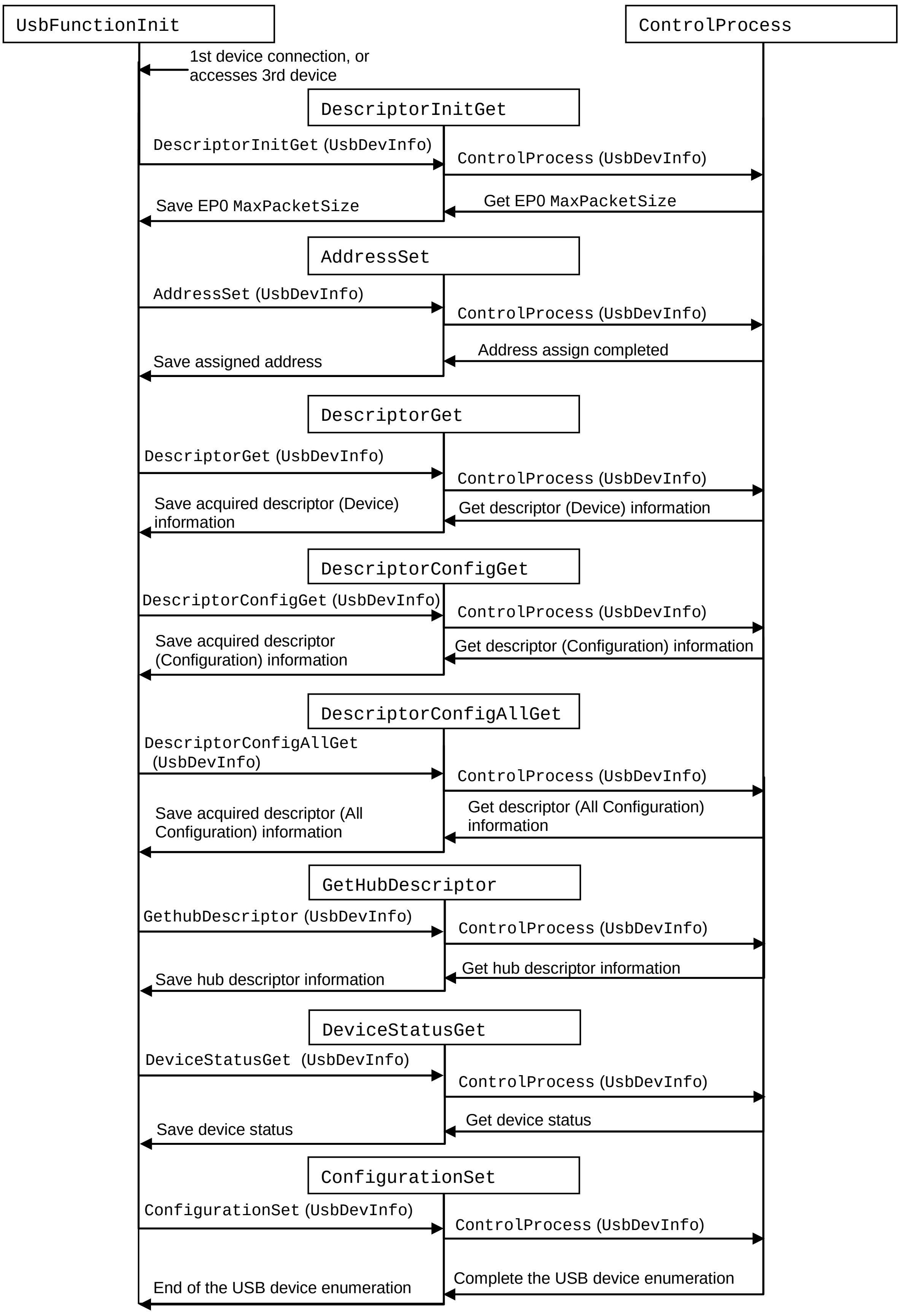
UsbFunctionInit
ControlProcess
1st device connection, or accesses 3rd device
DescriptorInitGet
DescriptorInitGet (UsbDevInfo)
ControlProcess (UsbDevInfo)
Save EP0 MaxPacketSize
Get EP0 MaxPacketSize
AddressSet
AddressSet (UsbDevInfo)
ControlProcess (UsbDevInfo)
Address assign completed
Save assigned address
DescriptorGet
DescriptorGet (UsbDevInfo)
ControlProcess (UsbDevInfo)
Save acquired descriptor (Device) information
Get descriptor (Device) information
DescriptorConfigGet
DescriptorConfigGet (UsbDevInfo)
ControlProcess (UsbDevInfo)
Save acquired descriptor (Configuration) information
Get descriptor (Configuration) information
DescriptorConfigAllGet
DescriptorConfigAllGet (UsbDevInfo)
ControlProcess (UsbDevInfo)
Save acquired descriptor (All Configuration) information
Get descriptor (All Configuration) information
GetHubDescriptor
GethubDescriptor (UsbDevInfo)
ControlProcess (UsbDevInfo)
Save hub descriptor information
Get hub descriptor information
DeviceStatusGet
DeviceStatusGet (UsbDevInfo)
ControlProcess (UsbDevInfo)
Save device status
Get device status
ConfigurationSet
ConfigurationSet (UsbDevInfo)
ControlProcess (UsbDevInfo)
End of the USB device enumeration
Complete the USB device enumeration

Figure 5. Correlation Transfer Function Behavior (Enumeration)

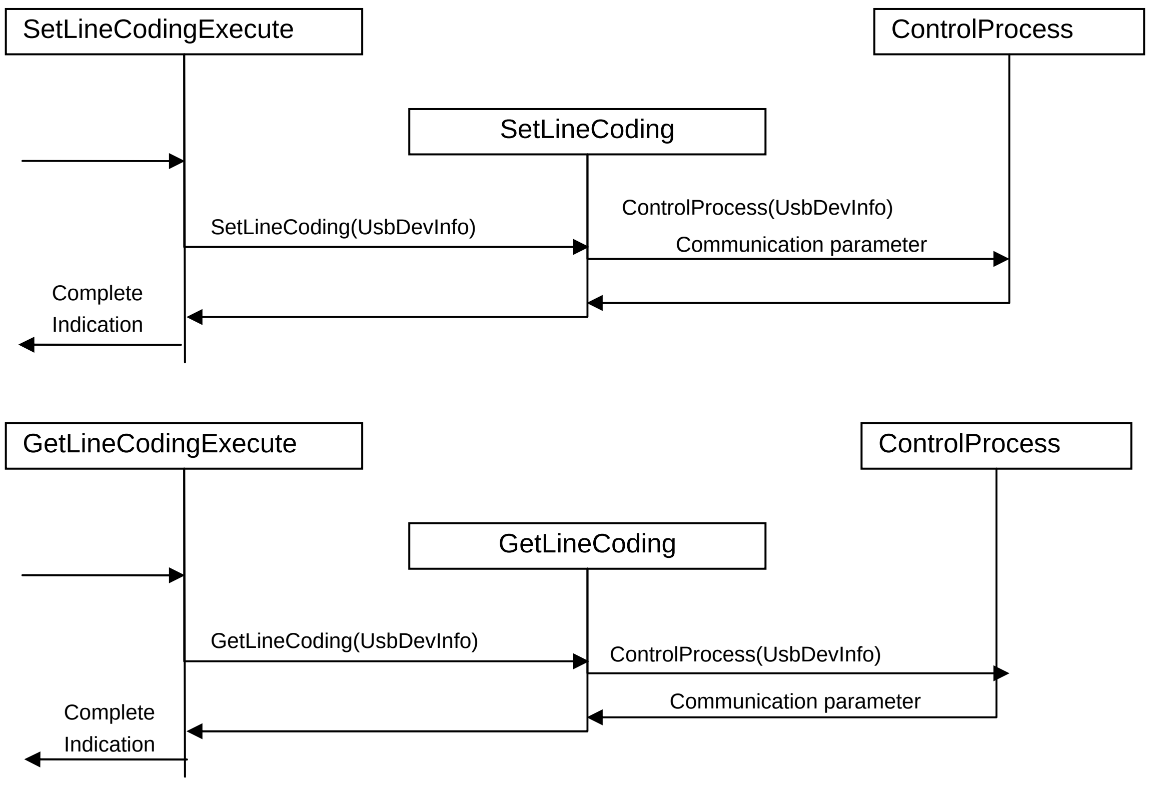

Figure 6. Correlation Transfer Function Behavior (Class commands)

5. Interrupt-in Transfer

This section provides a diagram indicating the correlation between the interrupt-in transfer functions, describes the function specifications and operations in the interrupt-in transfer.

5.1 Correlation Between Interrupt-In Transfer Functions

When the enumeration completed for the hub device, executes the `ActiveHubPort` to enable the hub’s downstream port.
Starts issue the interrupt-in transfer to detect the status change of the downstream ports. This operation is executed in the `SetInterruptInProcess` called by the `ActiveHubPort`.

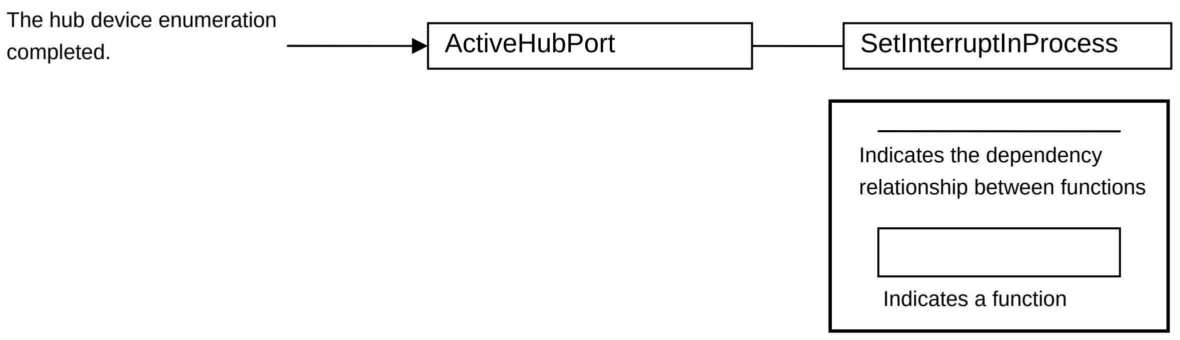
The hub device enumeration completed.
ActiveHubPort
SetInterruptInProcess
Indicates the dependency relationship between functions
Indicates a function

Figure 7. Correlation Between Interrupt-In Transfer Functions (Hub class)

5.2 Description of Functions

The following shows the function names and processing details.

Function	Description
`ActiveHubPort`	This function is called when the enumeration completed for the hub device. Detects the change of the downstream ports which are connected device and makes enable them. 1. Starts issue interrupt-in transfer to detect the status change of the hub device. 2. Transfer all connecting ports from “Powered-off state” to “Disconnected state”. 3. Waits the status change of the hub device by interrupt-in transfer. 4. Resets one of the device connected port. 5. Waits the status change of the hub device by interrupt-in transfer.. 6. Verifies the reset port transferred to “Enable state”.
`SetInterruptInProcess`	This function is called when the interrupt-in transfer is started to detect the status change of the hub device. Thereafter, starts issue interrupt-in transfer. 1. Acquires multiple EDs, one TD area for the interrupt-in transfer. The number of EDs is 32. 2. Specifies various parameters for the ED. 3. Specifies various parameters for the TD. 4. Links the multiple ED and one TD. 5. Sets the multiple ED’s Skip bit depend on the interbal of the endpoint descriptor. 6. Starts issue interrupt-in transfer setting the multiple ED address to the HccaInterruptTable in the HCCA..

5.3 Function Interfaces

The following shows the interface of each function. For details about Driver structure (=`USBDevInfo`), see *section 14*.

Function	`ActiveHubPort`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `ActiveHubPort` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`SetInterruptInProcess`	
Parameter	`UsbDevInfo`	`Num`
Parameter information	Driver structure (USB device information, Transfer information)	Interface number
Specifiable value	Other than NULL	0, 1, 2
Return value	STATUS	
Return value usage	Used to judge whether execution of `SetInterruptInProcess` finished successfully.	
Return value meaning	0: Normal completion	
	Other than 0: Abnormal termination	

5.4 Interrupt-In Transfer Operation

The Interrupt-in transfer is started by setting the multiple ED (Endpoint Descriptor) addresses to the HccaInterruptTable field in the HCCA (Host Controller Communication Area). Total number of ED is 32.
HC (Host Controller) make a polling each ED addresses 32msec interval.
If the polled ED is linked to a TD (Transfer Descriptor), HC do the interrupt-in transfer refer to the ED and the TD.
Figure 8 shows the example for the 8msec interval. Links the IntInEd[0] to IntInED[4] (total four) to the IntInTD and permutates each IntInEDs to balanced order, the 8msec interval interrupt-in transfer is executed by the HC.
Figure 9 shows the operation flow of the interrupt-in transfer function

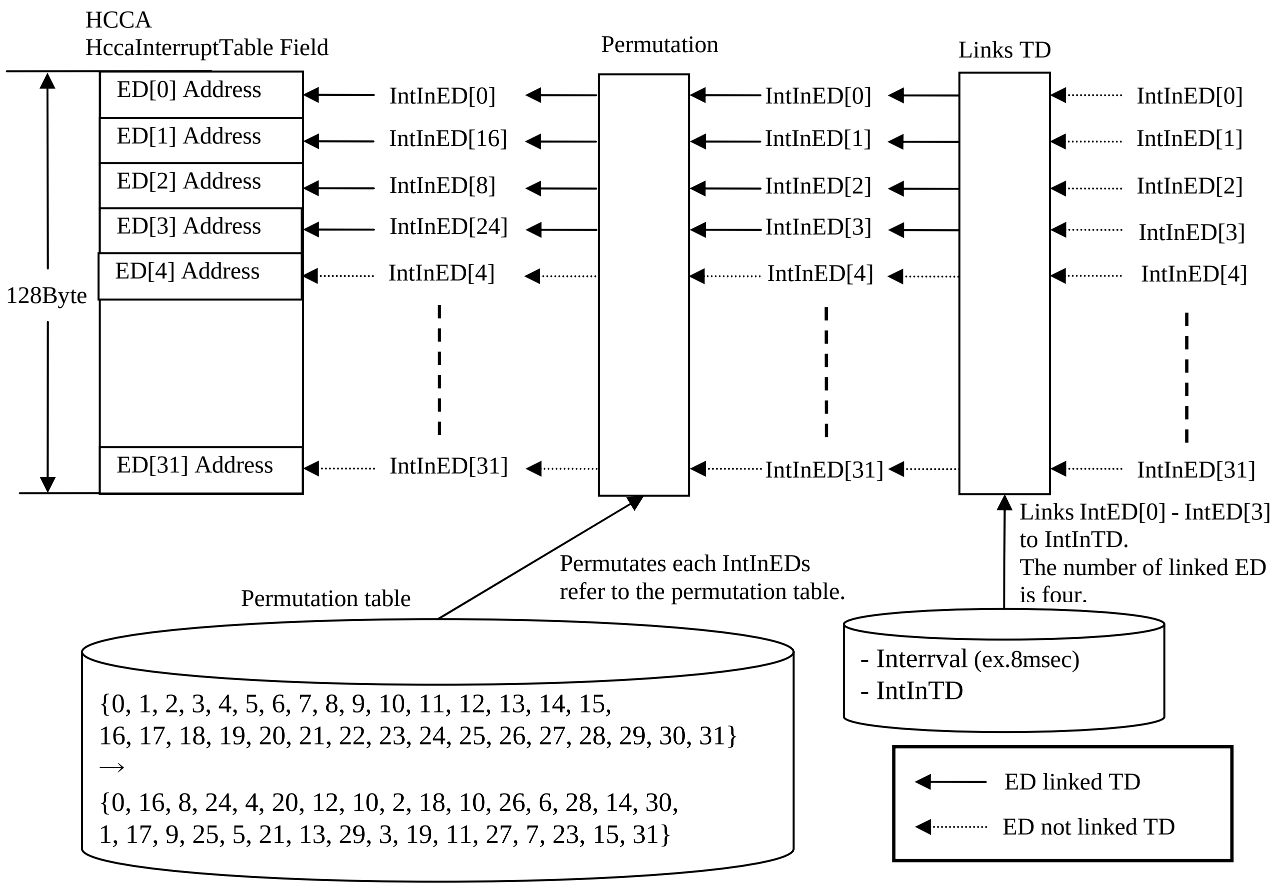

Figure 8. ED/TD Setting Interrupt-In Transfer

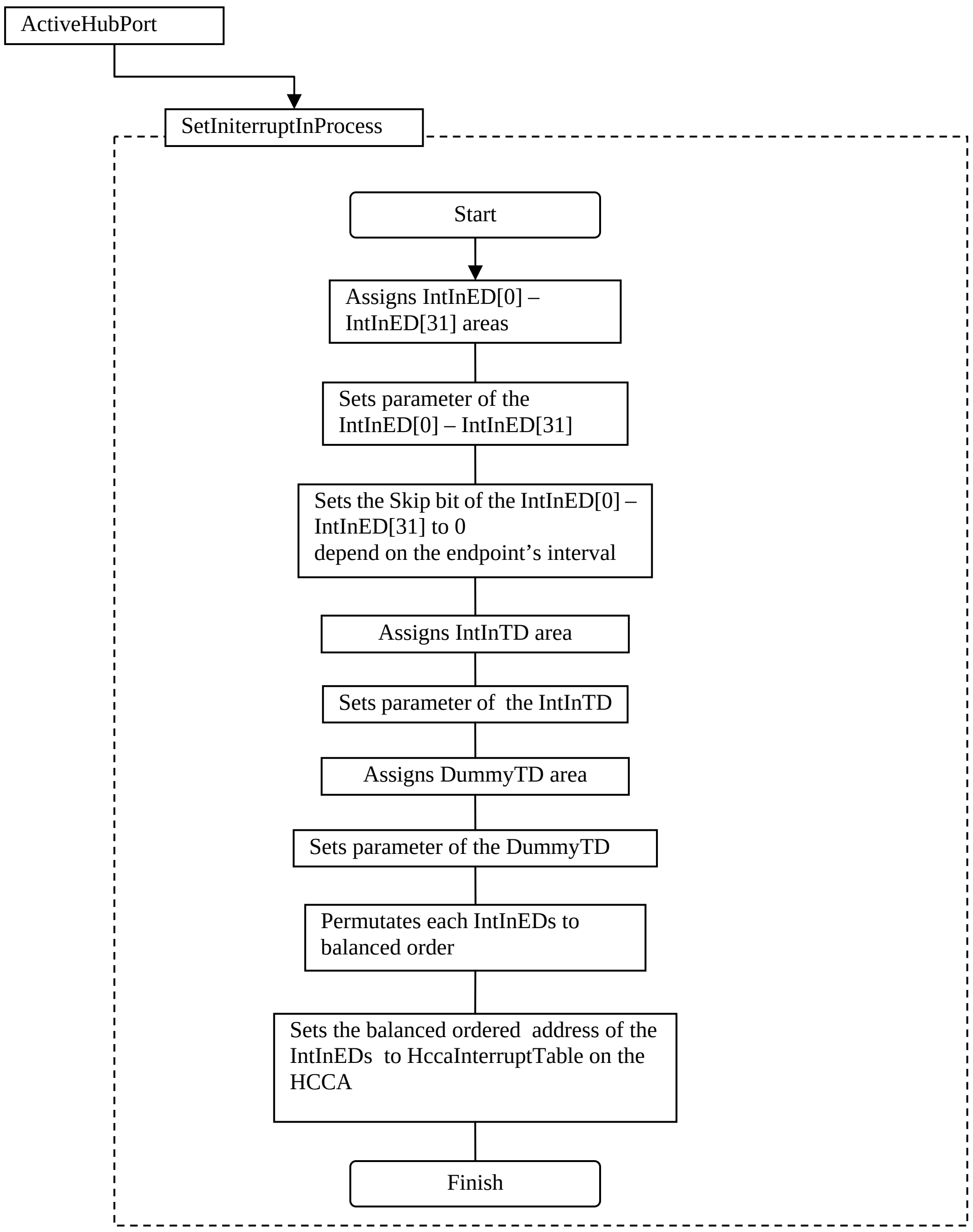
ActiveHubPort
SetIniterruptInProcess
Start
Assigns IntInED[0] – IntInED[31] areas
Sets parameter of the IntInED[0] – IntInED[31]
Sets the Skip bit of the IntInED[0] – IntInED[31] to 0 depend on the endpoint's interval
Assigns IntInTD area
Sets parameter of the IntInTD
Assigns DummyTD area
Sets parameter of the DummyTD
Permutates each IntInEDs to balanced order
Sets the balanced ordered address of the IntInEDs to HccaInterruptTable on the HCCA
Finish

Figure 9. Interrupt-In Transfer Operation Flow

6. Bulk Transfer (Mass Storage Class)

This section provides a diagram indicating the correlation between the bulk transfer functions, describes the function specifications and operations in the bulk transfer (mass storage class).

6.1 Correlation Between Bulk Transfer Functions

The `ATACommandExecute` function is called from the file system and converted to an ATAPI command, based on which bulk transfer is executed.

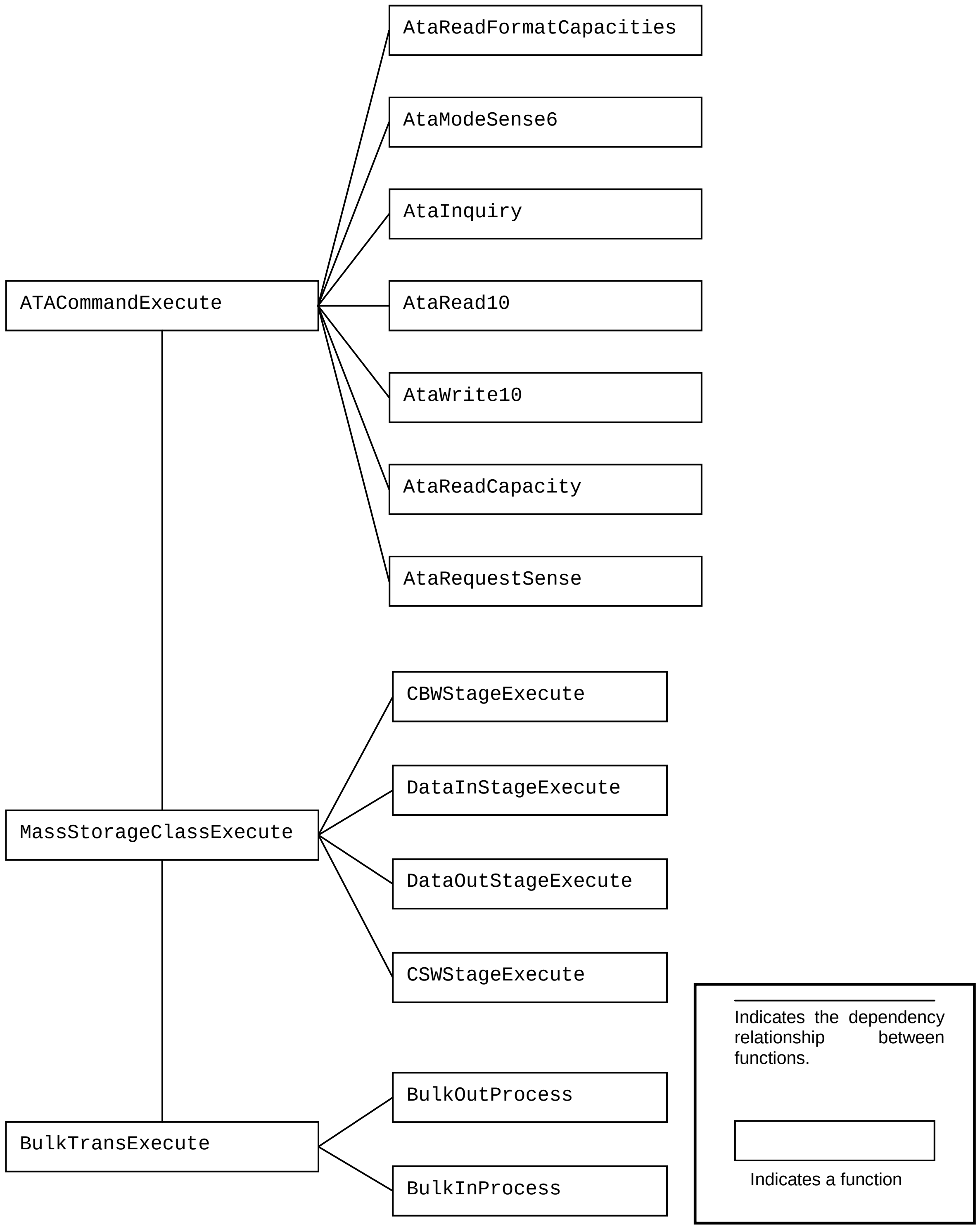

Figure 10. Correlation Between Bulk Transfer Functions (Mass Storage Class)

6.2 Description of Functions

The following shows the function names and processing details.

Function	Description
`ATACommandExecute`	This is an interface function between the FAT file system and the ATA layer functions. This function then executes functions such as `AtaWrite10` based on commands received from the file system.
`AtaRead10`	This function creates the data structure used to store data such as the number of the block to be read and the reception buffer area and calls the `MassStorageClassExecute` function.
`AtaWrite10`	This function creates the data structure used to store data such as the number of the block to be written and the data area and calls the `MassStorageClassExecute` function.
`AtaModeSense6`	This function is used to read the mode sense table. This function creates the data structure used to store data such as the reception buffer area and calls the `MassStorageClassExecute` function.
`AtaInquiry`	This function is used to read the inquiry table. This function creates the data structure used to store data such as the reception buffer area and calls the `MassStorageClassExecute` function.
`AtaRequestSense`	This function is used to read the sense data. This function creates the data structure used to store data such as the reception buffer area and calls the `MassStorageClassExecute` function.
`AtaReadFormatCapacities`	This function is used to read the `ReadFormatCapacity` table. This function creates the data structure used to store data such as the reception buffer area and calls the `MassStorageClassExecute` function.
`AtaReadCapacitys`	This function is used to read the `ReadCapacity` table. This function creates the data structure used to store data such as the reception buffer area and calls the `MassStorageClassExecute` function.
`MassStorageClassExecute`	This function analyzes the data structure and calls the `CBWStageExecute`, `DataInStageExecute`, `DataOutStageExecute`, or `CSWStageExecute` function. After transfer finishes, confirms that the transfer was executed successfully. If transfer terminated abnormally, this function performs the appropriate processing, such as retransferring the data.
`CBWStageExecute`	This function creates CBW data and calls the `BulkTransExecute` function.
`DataInStageExecute`	This function specifies the area for receiving data and calls the `BulkTransExecute` function.
`DataOutStageExecute`	This function specifies the area for transmitting data and calls the `BulkTransExecute` function.
`CSWStageExecute`	This function creates CSW data and calls the `BulkTransExecute` function.
`BulkTransExecute`	This function analyzes the received command and calls the `BulkOutProcess` or `BulkInProcess` function. After transfer finishes, confirms the completion status.

Function	Description
`BulkOutProcess`	This function performs the following: 1. Acquires the bulk-out ED and data TD area. 2. Assigns an address for transmitting data to the current buffer pointer. 3. Links the ED and TD, sets the ED address to the HcBulkHeadED register, writes 0x04 to the HcCommandStatus register, and then starts bulk transfer. 4. After transfer finishes, checks the `Done` queue to confirm that the transfer was executed successfully.
`BulkInProcess`	This function performs the following: 1. Acquires the bulk-in ED and data TD area. 2. Assigns an address for receiving data to the current buffer pointer. 3. Links the ED and TD, sets the ED address to the HcBulkHeadED register, writes 0x04 to the HcCommandStatus register, and then starts bulk transfer. 4. After transfer finishes, checks the `Done` queue to confirm that the transfer was executed successfully.

6.3 Function Interfaces

The following shows the interface of each function. For details about `USBDevInfo`, see *section 14*.

Function	`ATACommandExecute`	
Parameter	`UsbDevInfo`	Command
Parameter information	Driver structure	Command type
Specifiable value	Other than NULL	`ATA_INQUIRY`, `ATA_RREAD_FORMAT`, `ATA_RREAD_CAPACITY`, `ATA_RREAD_10`, `ATA_MODE_SENCE6`, `ATA_REQUEST_SENCE`, `ATA_WRITE_10`
Return value	STATUS	
Return value usage	Used to judge whether execution of `ATACommandExecute` finished successfully.	
Return value meaning	0: Normal completion Other than 0: Abnormal termination	

Function	`AtaRead10`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `AtaRead10` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`AtaWrite10`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `AtaWrite10` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`AtaModeSense6`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `AtaModeSense6` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`AtaInquiry`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `AtaInquiry` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`AtaRequestSense`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `AtaRequestSense` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`AtaReadFormatCapacities`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `AtaReadFormatCapacities` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`AtaReadCapacity`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `AtaReadCapacity` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`MassStorageClassExecute`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `MassStorageClassExecute` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`CBWStageExecute`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `CBWStageExecute` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`DataInStageExecute`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `DataInStageExecute` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`DataOutStageExecute`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `DataOutStageExecute` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`CSWStageExecute`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `CSWStageExecute` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`BulkTransExecute`		
Parameter	`UsbDevInfo`	`Command`	`Num`
Parameter information	Driver structure (USB device information, Transfer information)	Command type	Interface number
Specifiable value	Other than NULL	BULK_OUT_COMMAND, BULK_IN_COMMAND	0, 1, 2
Return value	STATUS		
Return value usage	Used to judge whether execution of BulkTransExecute finished successfully.		
Return value meaning	0: Normal completion Other than 0: Abnormal termination		

Function	`BulkOutProcess`	
Parameter	`UsbDevInfo`	`Num`
Parameter information	Driver structure (USB device information, Transfer information)	Interface number
Specifiable value	Other than NULL	0, 1, 2
Return value	STATUS	
Return value usage	Used to judge whether execution of `BulkOutProcess` finished successfully.	
Return value meaning	0: Normal completion Other than 0: Abnormal termination	

Function	`BulkInProcess`	
Parameter	`UsbDevInfo`	Parameter
Parameter information	Driver structure (USB device information, Transfer information)	Parameter information
Specifiable value	Other than NULL	Specifiable value
Return value	STATUS	
Return value usage	Used to judge whether execution of `BulkInProcess` finished successfully.	
Return value meaning	0: Return value meaning Other than 0: Abnormal termination	

6.4 Read/Write Operation

The following shows the operation when `AtaRead10` or `AtaWrite10` is executed.

When an access is made from the file system, the `ATACommandExecute` function is executed, and functions such as `AtaRead10` and `AtaWrite10` are executed according to the type of access. The `MassStorageClassExecute` function is subsequently called from the `AtaRead10` or `AtaWrite10` function, and bulk transfer is performed.

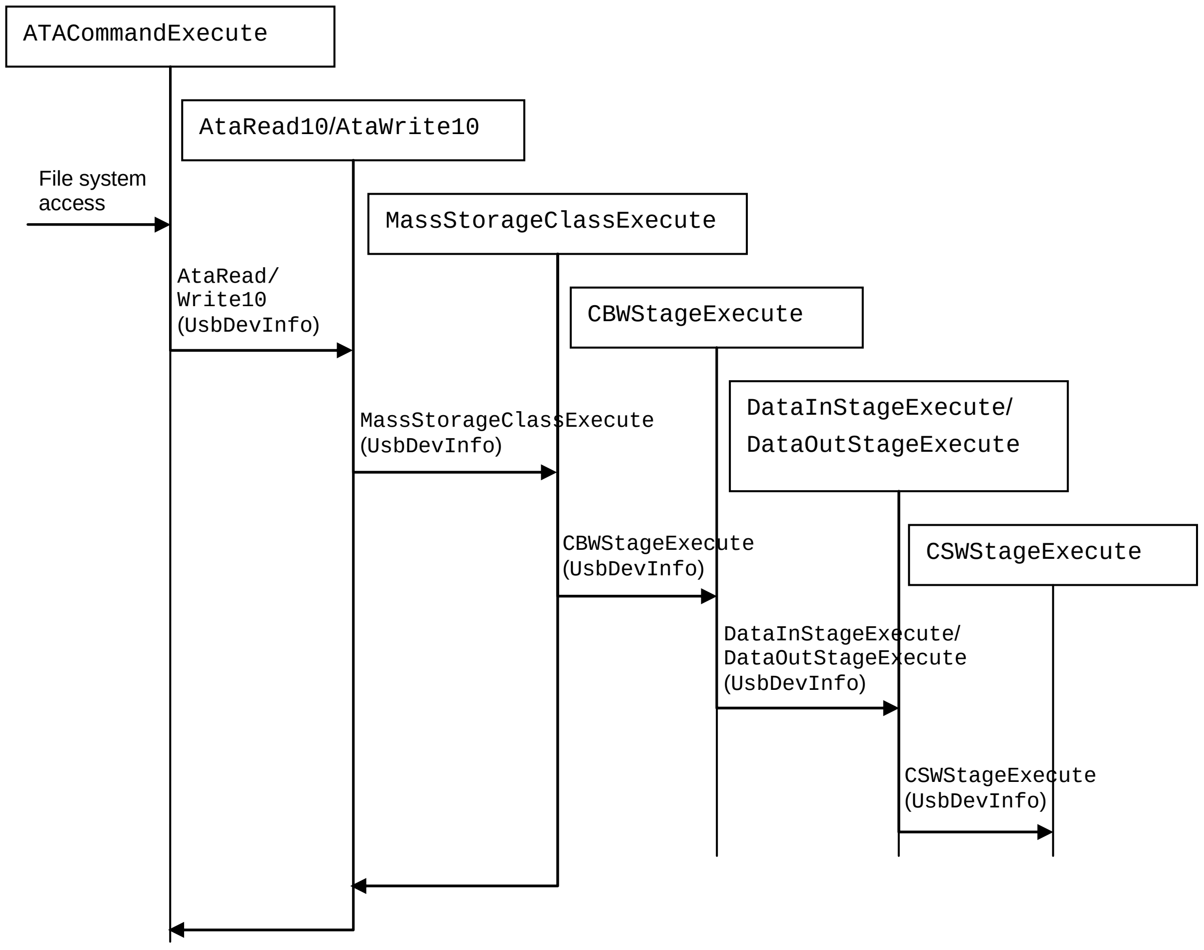
ATACommandExecute
AtaRead10/AtaWrite10
File system access
MassStorageClassExecute
AtaRead/ Write10 (UsbDevInfo)
CBWStageExecute
DataInStageExecute/ DataOutStageExecute
MassStorageClassExecute (UsbDevInfo)
CSWStageExecute
CBWStageExecute (UsbDevInfo)
DataInStageExecute/ DataOutStageExecute (UsbDevInfo)
CSWStageExecute (UsbDevInfo)

Figure 11. Read/Write Function Behavior

7. Bulk Transfer (Communication Class)

This section provides a diagram indicating the correlation between the bulk transfer functions, describes the function specifications in the bulk transfer (communication class).

7.1 Correlation Between Bulk Transfer Functions

The `CommunicationClassExecute` function is called from the `ComRead` and `ComWrite` functions of the communication class interface and executes bulk transfer.

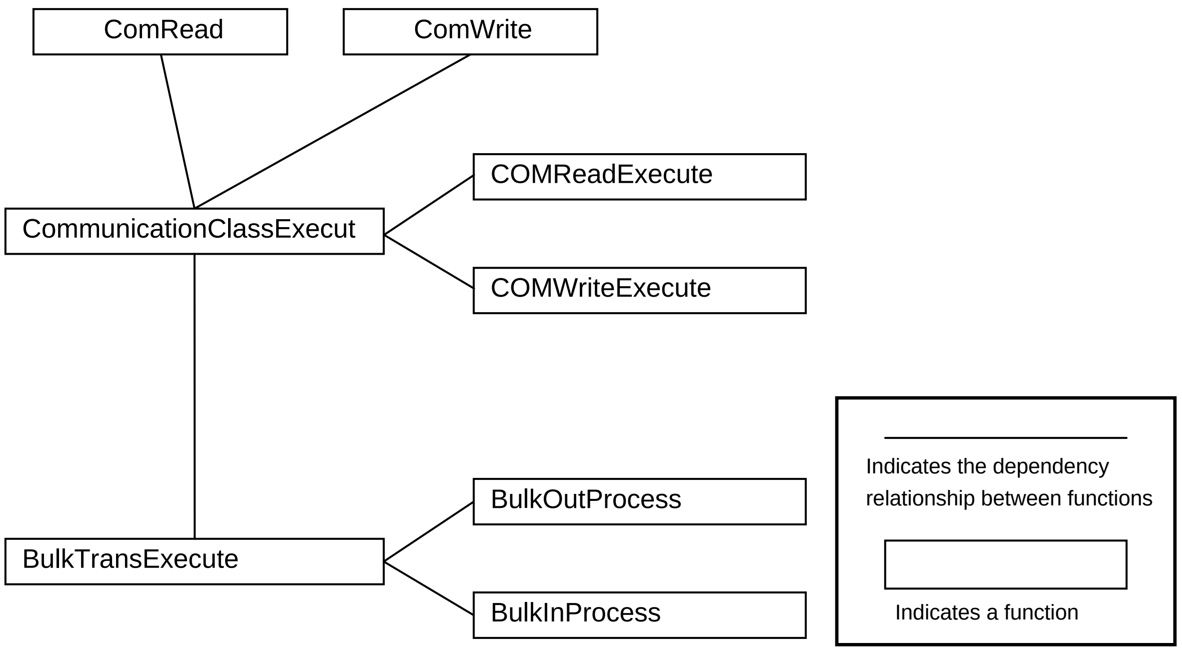

Figure 12. Correlation Between Bulk Transfer Functions (Communication Class)

7.2 Description of Functions

The following shows the function names and processing details.

Function	Description
`ComRead`	This is called from user application as read requirement and executes `CommunicationClassExecute` function.
`ComWrite`	This is called from user application as write requirement and executes `CommunicationClassExecute` function.
`CommunicationClassExecute`	This function analyzes the driver structure (USBHD_DEV_INFO) and calls the `Com ReadExecute or` the `COMWriteExecute` function depend on the analyzed result. After transfer finishes, confirms that the transfer was executed successfully. If transfer terminated abnormally, this function performs the appropriate processing, such as retransferring the data.
`COMReadExecute`	This function specifies the area for receiving data and calls the `BulkTransExecute` function.
`COMWriteExecute`	This function specifies the area for transmitting data and calls the `BulkTransExecute` function.
`BulkTransExecute`	This function analyzes the received command and calls the `BulkOutProcess` or `BulkInProcess` function. After transfer finishes, confirms the completion status.
`BulkOutProcess`	This function performs the following: 1. Acquires the bulk-out ED and data TD area. 2. Assigns an address for transmitting data to the current buffer pointer. 3. Links the ED and TD, sets the ED address to the HcBulkHeadED register, writes 0x04 to the HcCommandStatus register, and then starts bulk transfer. 4. After transfer finishes, checks the `Done` queue to confirm that the transfer was executed successfully.
`BulkInProcess`	This function performs the following: 1. Acquires the bulk-in ED and data TD area. 2. Assigns an address for receiving data to the current buffer pointer. 3. Links the ED and TD, sets the ED address to the HcBulkHeadED register, writes 0x04 to the HcCommandStatus register, and then starts bulk transfer. 4. After transfer finishes, checks the `Done` queue to confirm that the transfer was executed successfully.

7.3 Function Interfaces

The following shows the interface of each function. For details about `USBDevInfo`, see *section 14*. Furthermore, for details about `BulkTransExecute, BulkOutProcess and BulkInProcess, see` *section 6.3.*

Function	`ComRead`	
Parameter	`buffer`	`CntByte`
Parameter information	Read data buffer	Read require byte
Specifiable value	Other than NULL	From 1 to 64
Return value	The number of read complete bytes	
Return value usage	Used to judge whether execution of `ComRead` finished successfully and get the number of read complete byte	
Return value meaning	0: Normal completion Other than 0: Abnormal termination	

Function	`ComWrite`	
Parameter	`buffer`	`CntByte`
Parameter information	Write data buffer	Write require byte
Specifiable value	Other than NULL	From 1 to 64
Return value	The number of write complete bytes	
Return value usage	Used to judge whether execution of `ComWrite` finished successfully and get the number of read complete byte	
Return value meaning	0: Normal completion Other than 0: Abnormal termination	

Function	`CommunicationClassExecute`	
Parameter	`UsbDevInfo`	`Command`
Parameter information	Driver structure (USB device information, Transfer information)	Command type
Specifiable value	Other than NULL	COMReadExecute、COMWriteExecute
Return value	STATUS	
Return value usage	Used to judge whether execution of `CommunicationClassExecute` finished successfully.	
Return value meaning	0: Normal completion Other than 0: Abnormal termination	

Function	COMReadExecute
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `COMReadExecute` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	COMWriteExecute
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `COMWriteExecute` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

8. Creation of Descriptors

This section provides a diagram indicating the correlation between the descriptor creation functions, describes the function specifications and operations in the descriptor creation.

8.1 Correlation Between Descriptor Creation Functions

A diagram indicating the correlation between descriptor creation functions is shown below. EDs and TDs are generated by the `UsbhdMemInit` function during initialization.

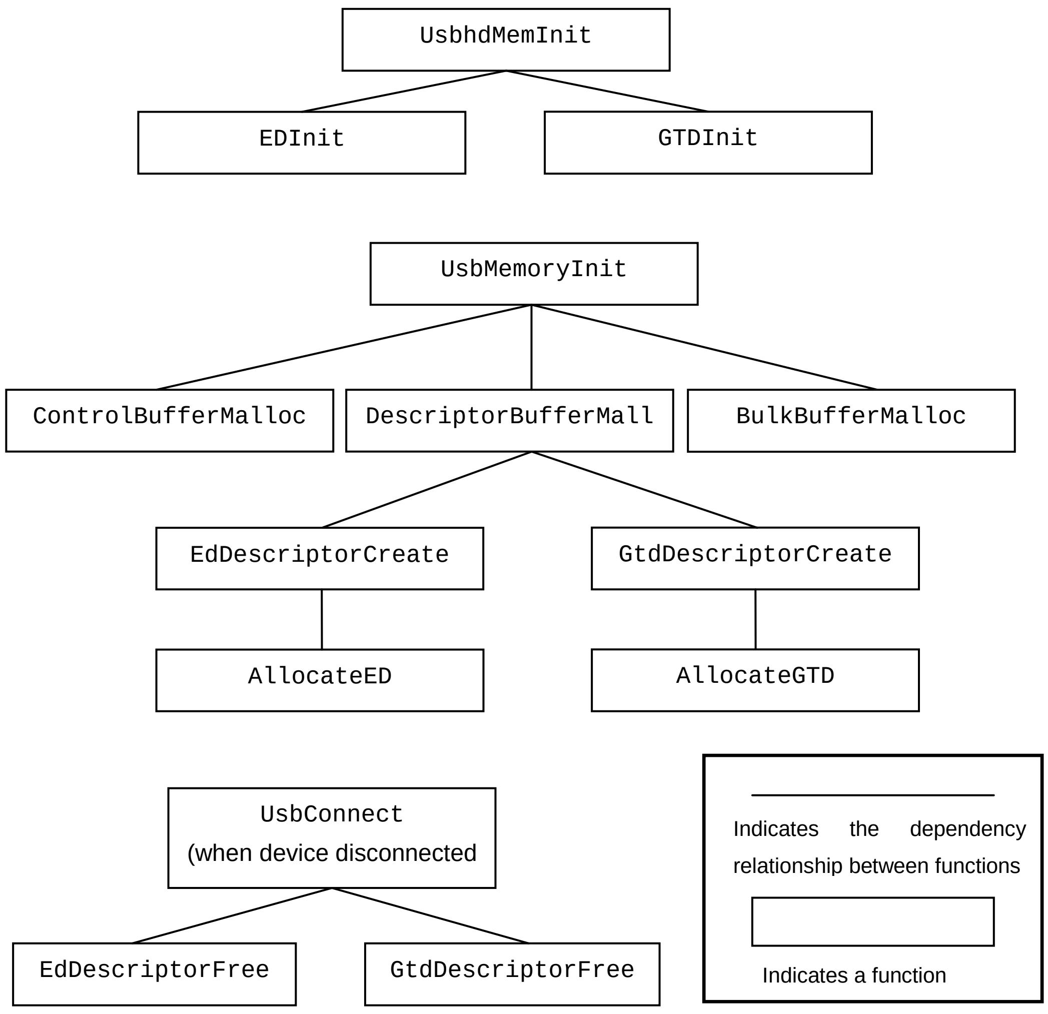

Figure 13. Correlation Between Descriptor Creation Functions

8.2 Description of Functions

The following shows the function names and processing details.

Function	Description
`UsbhdMemInit`	This function acquires the ED and TD parameters and calls the EDInit and GTDInit functions.
`EDInit`	This function initializes the ED descriptor area.
`GTDInit`	This function initializes the TD descriptor area.
`UsbMemoryInit`	This function initializes the buffers for control and bulk transfer when a USB device is connected.
`ControlBufferMalloc`	This function allocates data area for control transfer.
`BulkBufferMalloc`	This function allocates data area for bulk and interrupt-in transfer.
`DescriptorBufferMalloc`	This function sets the EDs and TDs. The control transfer uses one EDs and four TDs. The bulk transfer uses two EDs and four TDs per interfaces (total three). The interrupt-in transfer uses thirty two EDs and two TDs. (total three).
`EdDescriptorCreate`	This function assigns one ED from the ED area and returns the address of that ED.
`GtdDescriptorCreate`	This function assigns one TD from the TD area and returns the address of that TD.
`AllocateED`	This function searches for an unused (free) ED in the ED area and returns the address of that ED.
`AllocateGTD`	This function searches for an unused (free) TD in the TD area and returns the address of that TD.
`EdDescriptorFree`	This function removes the specified ED from the chain and updates its status to unused.
`GtdDescriptorFree`	This function removes the specified TD from the chain and updates its status to unused.

8.3 Function Interfaces

The following shows the interface of each function.

Function	`UsbhdMemInit`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	STATUS
Return value usage	Used to judge whether execution of `UsbhdMemInit` finished successfully.
Return value meaning	Normal completion

Function	`EDInit`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	STATUS
Return value usage	Used to judge whether execution of `EDInit` finished successfully.
Return value meaning	0: Normal completion

Function	`GTDInit`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	STATUS
Return value usage	Used to judge whether execution of `GTDInit` finished successfully.
Return value meaning	0: Normal completion

Function	`UsbMemoryInit`
Parameter	`UsbDevInfo`
Parameter information	USB device information
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `UsbMemoryInit` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`ControlBufferMalloc`
Parameter	`UsbDevInfo`
Parameter information	USB device information
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `ControlBufferMalloc` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`BulkBufferMalloc`
Parameter	`UsbDevInfo`
Parameter information	USB device information
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `BulkBufferMalloc` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`DescriptorBufferMalloc`
Parameter	`UsbDevInfo`
Parameter information	USB device information
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `DescriptorBufferMalloc` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`EdDescriptorCreate`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	`USBHD_ED_DESC`
Return value usage	Address of the assigned ED
Return value meaning	Other than NULL: Normal completion NULL: No free ED

Function	`GtdDescriptorCreate`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	`USBHD_GTD_DESC`
Return value usage	Address of the assigned TD
Return value meaning	Other than NULL: Normal completion NULL: No free TD

Function	`AllocateED`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	`DESC_ED_MEMORY`
Return value usage	Address of unused ED
Return value meaning	Other than NULL: Normal completion NULL: No free ED

Function	`AllocateGTD`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	`DESC_GTD_MEMORY`
Return value usage	Address of unused TD
Return value meaning	Other than NULL: Normal completion NULL: No free TD

Function	`EdDescriptorFree`
Parameter	`ED`
Parameter information	Address of the ED to be released
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `EdDescriptorFree` finished successfully.
Return value meaning	0: Normal completion –1: Abnormal termination

Function	`GtdDescriptorFree`
Parameter	`GTD`
Parameter information	Address of the TD to be released
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `GtdDescriptorFree` finished successfully.
Return value meaning	0: Normal completion –1: Abnormal termination

8.4 ED/TD Acquisition Operation

The following shows the operation when EDs and TDs are acquired.
`UsbhdMemInit` specifies the ED and TD base addresses and calls the `EDInit` and `GTDInit` functions.

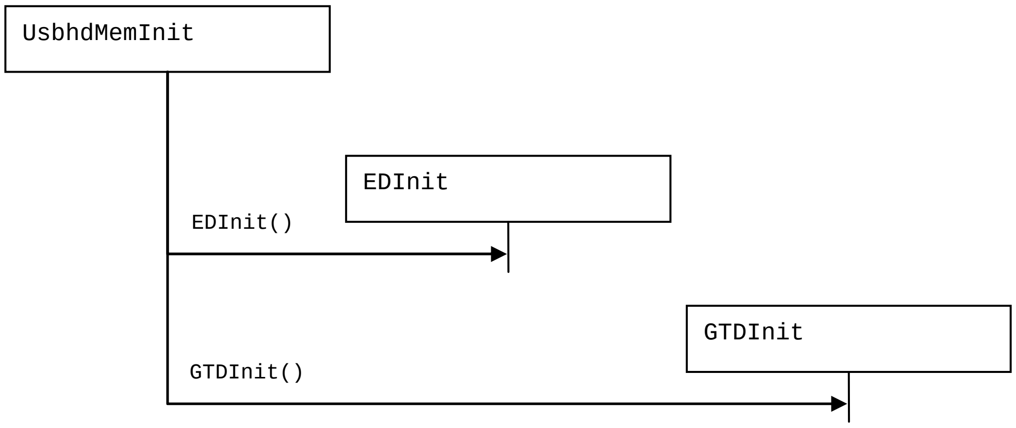

Figure 14. ED/TD Acquisition Function Behavior

8.5 ED/TD Setting Operation

The following shows the operation when EDs and TDs are set.
`UsbhdMemInit` allocates the areas for EDs and TDs when performing control, bulk and interrupt-in transfer.
`UsbhdMemInit` also allocates the data areas which are used control, bulk and interrupt-in transfer.

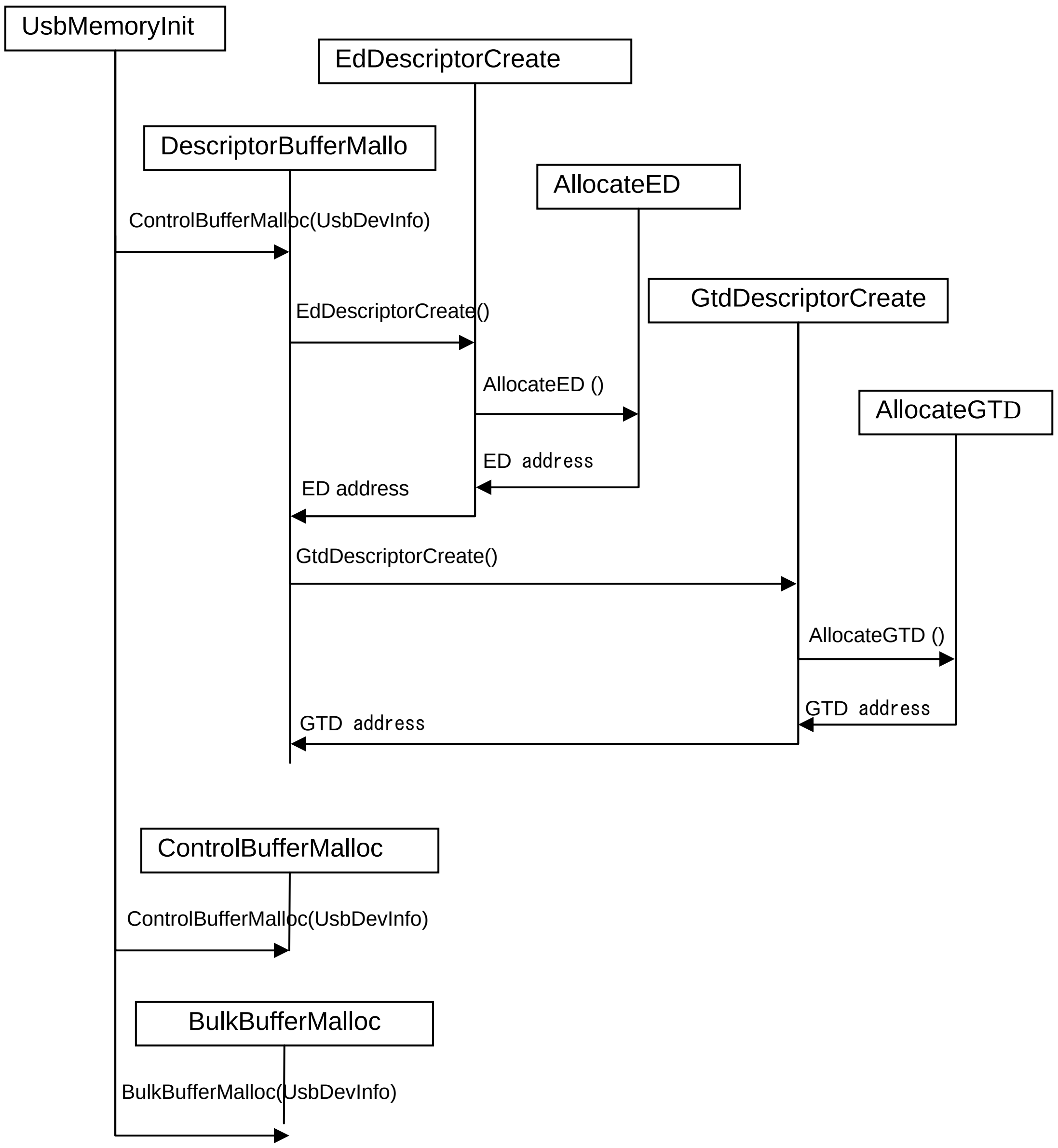

Figure 15. ED/TD Setting Function Behavior

9. Application (Mass Storage Class)

This section provides a diagram indicating the correlation between the functions (mass storage class), describes the function specifications and operations in the application.

9.1 Application Correlation

A diagram indicating the correlation between application functions is shown below.
FAT file system is used M3S-TFAT-Tiny (following is TFAT) that is Renesas original one.
This section describes the typical usage of the application. As for other usage for other applications, please refer to TFAT application note[5].

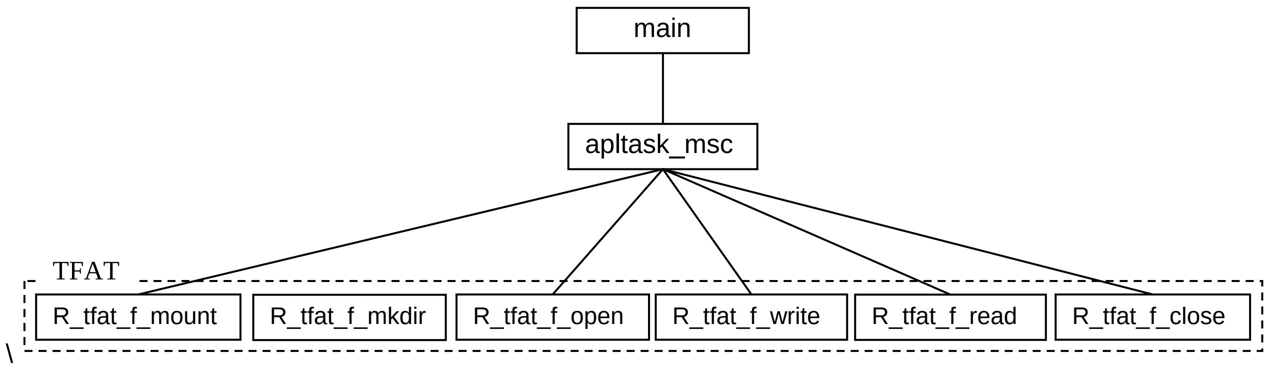

Figure 16. Correlation Between Application Functions

9.2 Description of Functions

The following shows the function names and processing details.

Function	Description
`Apltask_msc`	This function is called from the `main` function and executes application processing.
`R_tfat_f_mount`	This function allocates the work area for FAT file system and connects it to the device drivers.
`R_tfat_f_mkdir`	This function makes the directories.
`R_tfat_f_open`	This function opens the existing file or creates the new file if specified file is not existed.
`R_tfat_f_write`	This function writes data to the opened file.
`R_tfat_f_read`	This function reads data from the opened file.
`R_tfat_f_close`	This function closes the opened file.

9.3 Function Interfaces

The following shows the interface of each function.

Function	`apltask_msc`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	STATUS
Return value usage	Used to judge whether execution of `apltask_msc` finished successfully.
Return value meaning	0: Normal completion –1: Abnormal termination

Function	`R_tfat_f_mount`	
Parameter	Logical drive number	Work area
Parameter information	Logical drive number whose work area is allocated and removed by this function	Pointer to the work area
Specifiable value	Always 0 (Only 1 drive supported)	Other than NULL
Return value	Result of allocation or removal	
Return value usage	Used to judge whether execution of `R_tfat_f_mount` finished successfully.	
Return value meaning	TFAT_FR_OK: Normal completion Other than TFAT_FR_OK: Abnormal termination	

Function	`R_tfat_f_mkdir`
Parameter	Directory name
Parameter information	Pointer to the creating directory
Specifiable value	ASCII code, short file name (8.3 format)
Return value	Result of creating directory
Return value usage	Used to judge whether execution of `R_tfat_f_mkdir` finished successfully.
Return value meaning	TFAT_FR_OK: Normal completion Other than TFAT_FR_OK: Abnormal termination

Function	R_tfat_f_open		
Parameter	File pointer	File name	Attribute
Parameter information	Pointer to the structure that contains information about the opened file	File to be opened	Attribute of file
Specifiable value	Other than NULL	ASCII code, short file name (8.3 format)	Example: TFAT_FA_OPEN_ALWAYS: Open or create if file is not existed. TFAT_FA_WRITE: Write allowed TFAT_FA_READ: Read allowed
Return value	Result of file opened or created		
Return value usage	Used to judge whether execution of R_tfat_f_open finished successfully.		
Return value meaning	TFAT_FR_OK: Normal completion Other than TFAT_FR_OK: Abnormal termination		

Function	R_tfat_f_write			
Parameter	File pointer	Buffer	Requirement size	Completion size
Parameter information	Pointer to the structure that contains information about the opened file	Pointer to the write data buffer	Write requirement size (byte units)	Write completion size (byte units)
Specifiable value	File pointer get by R_tfat_f_open	Other than NULL	Other than NULL	-
Return value	Result of write data to the file			
Return value usage	Used to judge whether execution of R_tfat_f_write finished successfully.			
Return value meaning	TFAT_FR_OK: Normal completion Other than TFAT_FR_OK: Abnormal termination			

Function	R_tfat_f_read			
Parameter	File pointer	Buffer	Requirement size	Completion size
Parameter information	Pointer to the structure that contains information about the opened file	Pointer to the read data buffer	Read requirement size (byte units)	Read completion size (byte units)
Specifiable value	File pointer get by R_tfat_f_open	Other than NULL	Other than NULL	-
Return value	Result of read data from the file			
Return value usage	Used to judge whether execution of R_tfat_f_read finished successfully.			
Return value meaning	TFAT_FR_OK: Normal completion Other than TFAT_FR_OK: Abnormal termination			

Function	`R_tfat_f_close`
Parameter	File pointer
Parameter information	Pointer to the structure that contains information about the opened file
Specifiable value	File pointer get by `R_tfat_f_open`
Return value	Result of close the file
Return value usage	Used to judge whether execution of `R_tfat_f_close` finished successfully.
Return value meaning	TFAT_FR_OK: Normal completion Other than TFAT_FR_OK: Abnormal termination

9.4 Application Operation

The application performs the following operations:

- Creation of 10 files in a root directory.
- Creation of another directory and creation of 10 files in that directory.
- Writing, reading, and verifying data (1 to 40 blocks) in block (512-byte) units.
- Writing, reading, and verifying data (1 to 1024 bytes) in byte units.

Figure 17 shows an example of the write-read-verify operation.

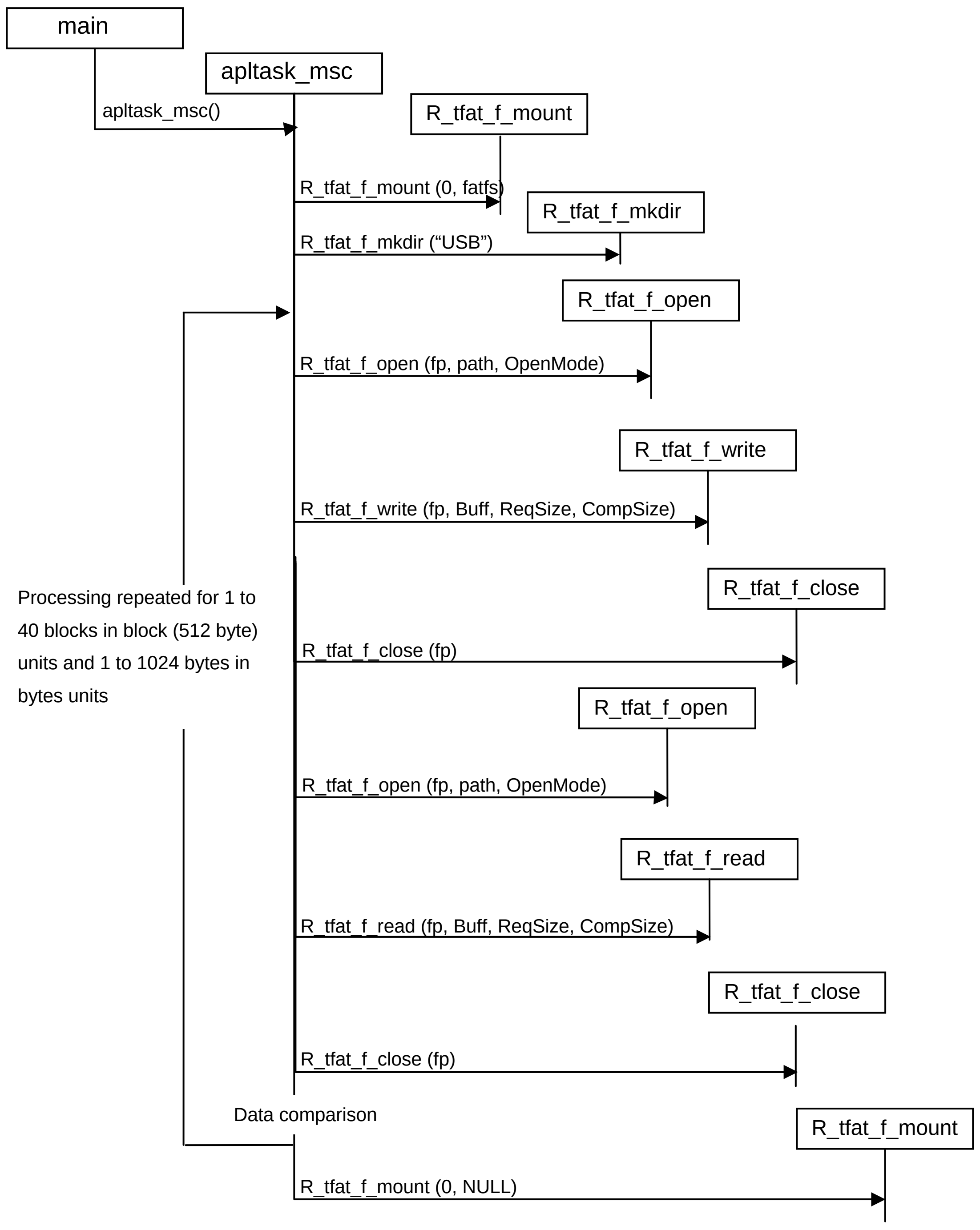
main
apltask_msc
apltask_msc()
R_tfat_f_mount
R_tfat_f_mount (0, fatfs)
R_tfat_f_mkdir
R_tfat_f_mkdir (“USB”)
R_tfat_f_open
R_tfat_f_open (fp, path, OpenMode)
R_tfat_f_write
R_tfat_f_write (fp, Buff, ReqSize, CompSize)
R_tfat_f_close
R_tfat_f_close (fp)
R_tfat_f_open
R_tfat_f_open (fp, path, OpenMode)
R_tfat_f_read
R_tfat_f_read (fp, Buff, ReqSize, CompSize)
R_tfat_f_close
R_tfat_f_close (fp)
Processing repeated for 1 to 40 blocks in block (512 byte) units and 1 to 1024 bytes in bytes units
Data comparison
R_tfat_f_mount
R_tfat_f_mount (0, NULL)

Figure 17. Application Operation

10. Application (Communication Class)

This section provides a diagram indicating the correlation between the functions (communication class), describes the function specifications and operations in the application.

10.1 Application Correlation

A diagram indicating the correlation between application functions is shown below.

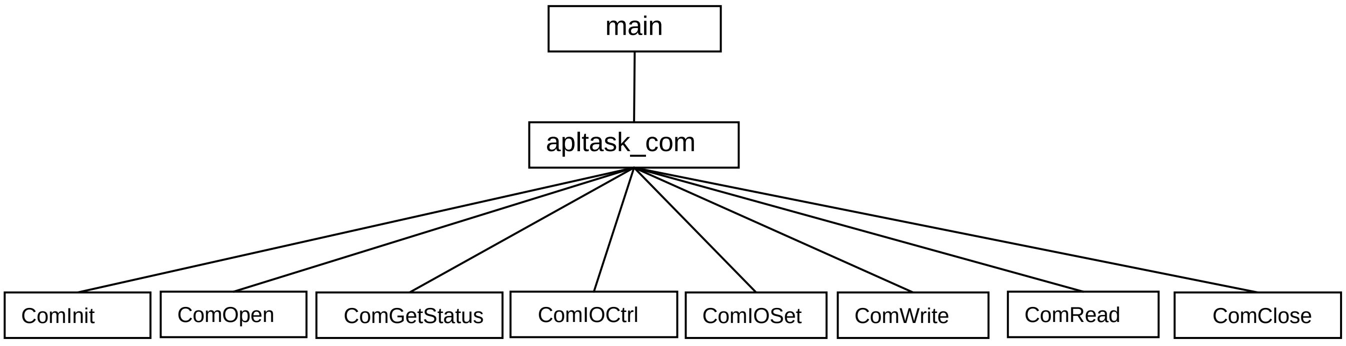

Figure 18. Correlation Between Application Functions

10.2 Description of Functions

The following shows the function names and processing details.

Function	Description
`apltask_com`	This function is called from the `main` function and executes application processing.
`ComInit`	This function acquires initial communication state from the USB device.
`ComOpen`	This function sets the software flag indicating the USB device started.
`ComClose`	This function clears the software flag indicating the USB device started.
`ComGetStatus`	This function acquires the communication parameter that is composed of transfer speed, stop bits, parity and start bit.
`ComIOCtrl`	This function sets the communication parameter that is composed of transfer speed, stop bits, parity and start bit.
`ComIOSet`	This function sends carrier control signal to the USB device.
`ComWrite`	This function specifies the pointer to the buffer storing the data to be written, and writes the data with the number of bytes to be write.
`ComRead`	This function specifies the pointer to the buffer storing the data to be read, and reads the data with the number of bytes to be read.

10.3 Function Interfaces

The following shows the interface of each function.

Function	`apltask_com`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	STATUS
Return value usage	Used to judge whether execution of `apltask_com` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`ComInit`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	STATUS
Return value usage	Used to judge whether execution of `ComInit` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`ComOpen`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	STATUS
Return value usage	Used to judge whether execution of `ComOpen` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`ComClose`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	–

Function	`ComGetStatus`
Parameter	`param`
Parameter information	Communication parameter structure (Transfer speed, Stop bit, Parity and Start bit)
Specifiable value	none
Return value	STATUS
Return value usage	Used to judge whether execution of `ComGetStatus` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`ComIOCtrl`
Parameter	`param`
Parameter information	Communication parameter structure (Transfer speed, Stop bit, Parity and Start bit)
Specifiable value	none
Return value	STATUS
Return value usage	Used to judge whether execution of `ComIOCtrl` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`ComIOSet`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	STATUS
Return value usage	Used to judge whether execution of `ComIOSet` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`ComWrite`	
Parameter	`buffer`	`CntByte`
Parameter information	Data to write	Requirement size of the data to write
Specifiable value	Data of the size specified by `WriteSize`	1 to 64 (bytes)
Return value	Size of the written data	
Return value usage	Used to judge whether execution of `ComWrite` finished successfully.	
Return value meaning	More than 1: Normal completion (Write complete size) Negative value: Abnormal termination	

Function	`ComRead`	
Parameter	`buffer`	`CntByte`
Parameter information	Data to read	Requirement size of the data to read
Specifiable value	Data of the size specified by `ReadSize`	1 to 64 (bytes)
Return value	Size of the read data	
Return value usage	Used to judge whether execution of `ComRead` finished successfully.	
Return value meaning	More than 1: Normal completion (Read complete size) Negative value: Abnormal termination	

10.4 Application Operation

The application performs the following operations:

1. Starts access operation to the USB device. (Execute: ComOpen)
2. Sets the communication parameter. (ComIOCtrl)
3. Acquires the communication parameter got sequence 2., and verifies its contents. (ComGetStatus)
4. Sends the carrier control signal. (ComIOSet)
5. Sets the communication parameter again. (ComIOCtrl)
6. Acquires the communication parameter got sequence 5., and verifies its contents. (ComGetStatus)
7. Writes data to the USB device. (ComWrite)
8. Acquires the communication parameter written by sequence 7., and verifies its contents. (ComGetStatus)
9. Reads data from the USB device. (ComRead)
10. Finalizes access operation to the USB device. (ComClose)

From sequence 1. to sequence 10. are repeated for 1 to 64 bytes in byte units.

Table 1. Shows the communication parameter and Figure 19 shows an example of the write-read-verify operation.

Table 1. Communication Parameter

Communication Parameter	Value (Contents)
Transfer speed	115200 (115200bp), 57600 (57600bps), 38400 (38400bps), 14400（14400bps）, 9600 (9600bps), 4800 (4800bps)
Stop bit	0 (1bit), 1 (1.5bits), 2 (2bits)
Parity	0 (None), 1 (Odd), 2 (Even), 3 (Mark), 4 (Space)
Data bit	5 (5bits), 6 (6bits), 7 (7bits), 8 (8bits), 16 (16bits)

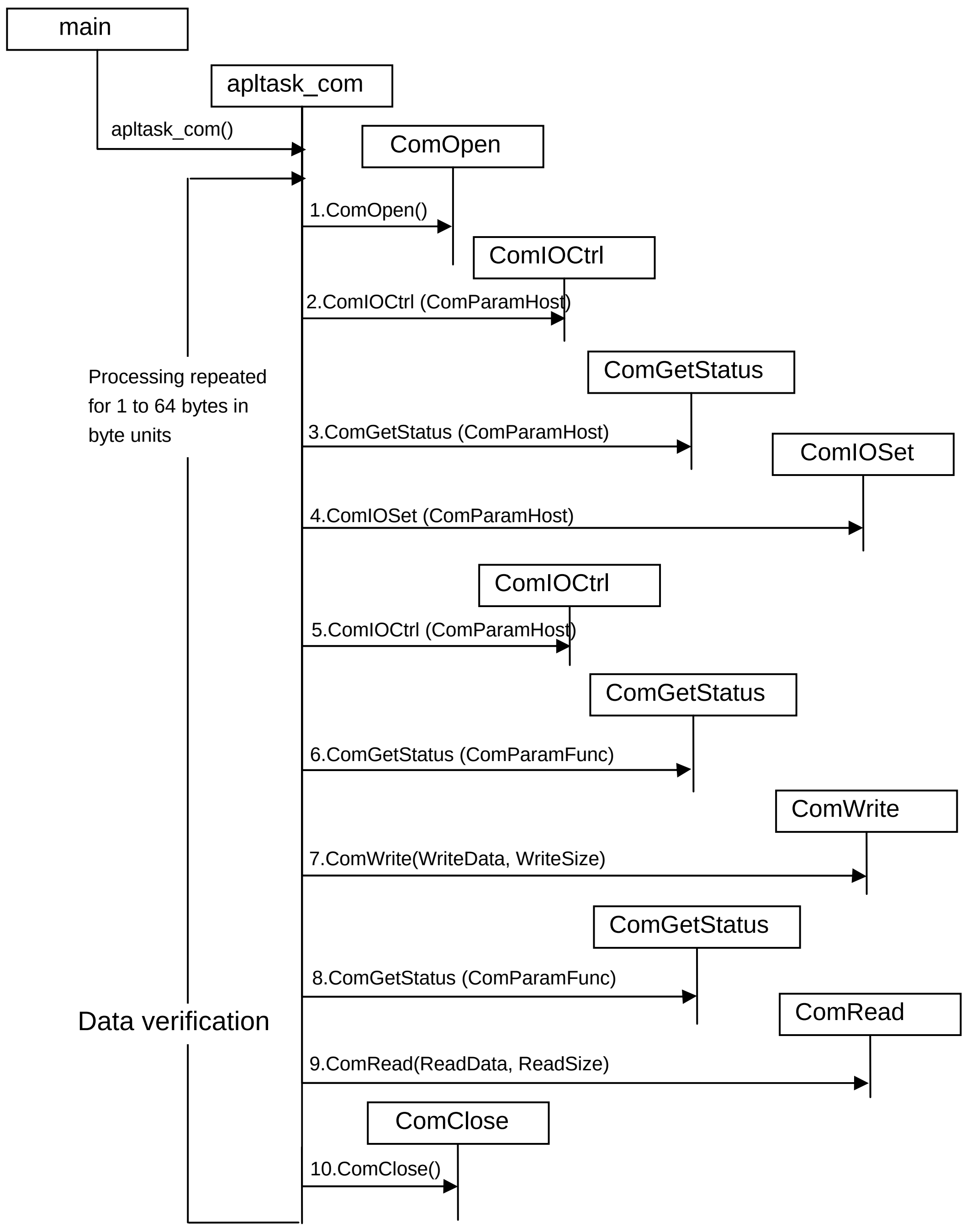
main
apltask_com
apltask_com()
ComOpen
1.ComOpen()
ComIOCtrl
2.ComIOCtrl (ComParamHost)
ComGetStatus
Processing repeated for 1 to 64 bytes in byte units
3.ComGetStatus (ComParamHost)
ComIOSet
4.ComIOSet (ComParamHost)
ComIOCtrl
5.ComIOCtrl (ComParamHost)
ComGetStatus
6.ComGetStatus (ComParamFunc)
ComWrite
7.ComWrite(WriteData, WriteSize)
ComGetStatus
8.ComGetStatus (ComParamFunc)
Data verification
ComRead
9.ComRead(ReadData, ReadSize)
ComClose
10.ComClose()

Figure 19. Application Operation

11. Hub Class Driver

This section describes the function specifications.

11.1 Description of Functions

The following shows the function names and processing details.

Function	Description
`HubClassInit`	Transfer all connecting ports of the hub device from “Powered-off state” to “Disconnected state”.
`WaitHubChange`	Waits the status change of the hub device by interrupt-in transfer.
`ResetHubPort`	Resets one of the device connected port and waits the reset completion of the hub device by interrupt-in transfer.
`ReadHubStatus`	Saves the received data by the interrupt-in transfer.

11.2 Function Interfaces

The following shows the interface of each function.

Function	`HubClassInit`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `HubClassInit` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

Function	`WaitHubChange`	
Parameter	`UsbDevInfo`	`port`
Parameter information	Driver structure (USB device information, Transfer information)	Port number
Specifiable value	Other than NULL	0 to 6 (max number of port: 7)
Return value	STATUS	
Return value usage	Used to judge whether execution of `WaitHubChange` finished successfully.	
Return value meaning	0: Normal completion	
	Other than 0: Abnormal termination	

Function	`ResetHubPort`	
Parameter	`UsbDevInfo`	`port`
Parameter information	Driver structure (USB device information, Transfer information)	Port number
Specifiable value	Other than NULL	0 to 6 (max number of port: 7)
Return value	STATUS	
Return value usage	Used to judge whether execution of `ResetHubPort` finished successfully.	
Return value meaning	0: Normal completion	
	Other than 0: Abnormal termination	

Function	`ReadHubStatus`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `ReadHubStatus` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

12. Hierarchical Structure

The hierarchical structure of the USB host software is shown below.

12.1 Hierarchical Structure

The hierarchical structure is shown in Figure 20 below.

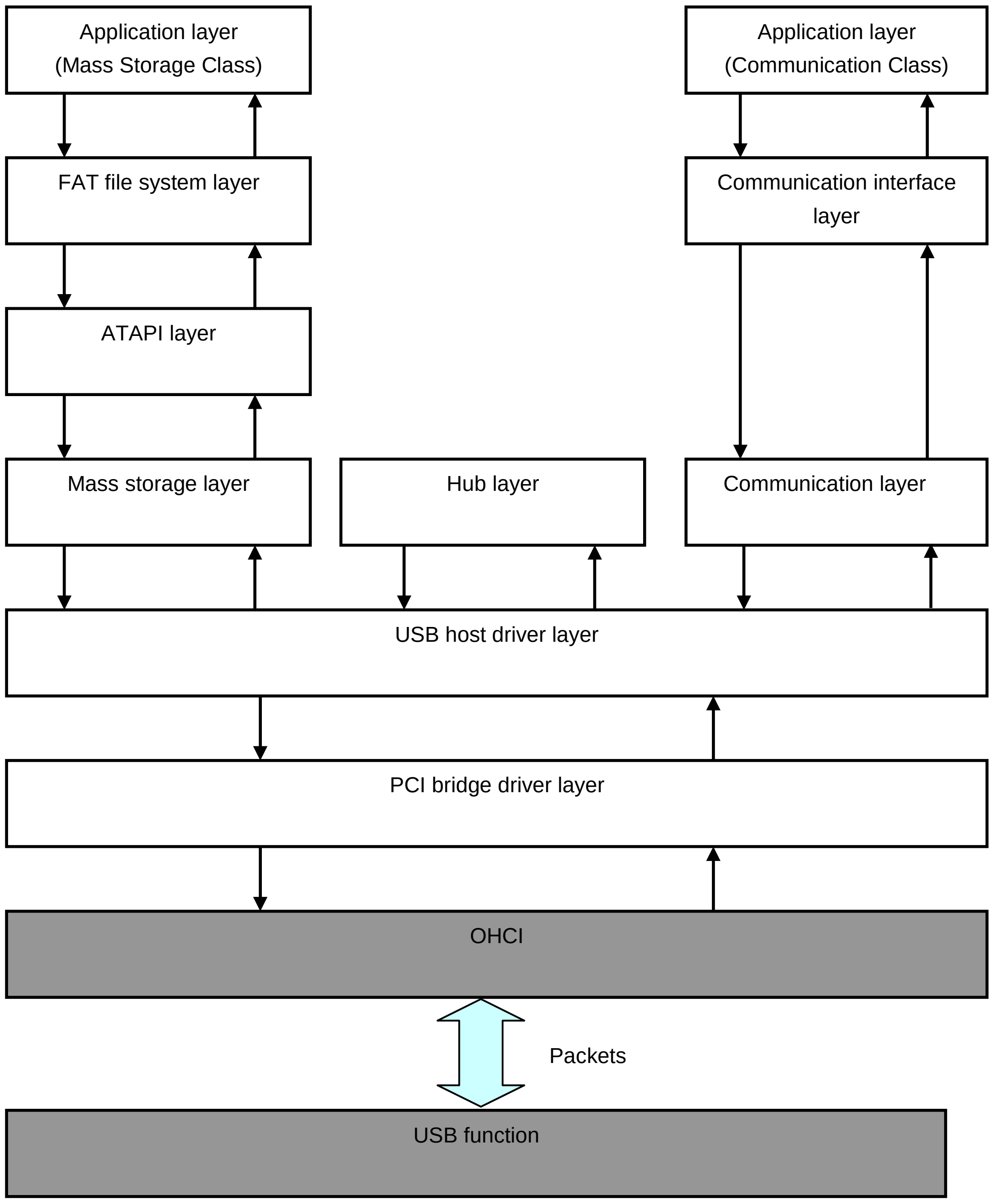

Figure 20. Hierarchy Diagram

12.2 Description of Hierarchy

The following shows the function names and processing details for each layer in the hierarchy.

Layer	Function	Description
Application	`apltask_msc` `apltask_com`	In this layer, data is sent to and received from lower layers.

FAT file system	`cfs_mountfs`
	`cfs_open`
	`cfs_write`
	`cfs_read`
	`cfs_close`
	`cfs_umountfs`
		In this layer, commands sent from the application layer, such as to open files and write and read data, are executed. Information such as the block number, the address of the reception buffer for the Read command and the address of the data buffer for the Write command are passed from this layer to lower layers.

Communication interface	`ComInit`
	`ComOpen`
	`ComClose`
	`ComRead`
	`ComWrite`
	`ComGetStatus`
	`ComIOCtrl`
	`ComIOSet`
		In this layer, commands sent from the application layer, such as to open and write and read data, are executed to/from the USB device. Information such as the block number, the address of the reception buffer for the Read command and the address of the data buffer for the Write command are passed from this layer to lower layers.

ATAPI	`ATACommandExecute`
	`AtaReadFormatCapacities`
	`AtaModeSense6`
	`AtaInquiry`
	`AtaRead10`
	`AtaWrite10`
	`AtaReadCapacity`
	`AtaRequestSense`
		In this layer, Read and Write commands from the file system are converted into ATAPI commands and passed to lower layers.

Mass storage	`MassStorageClassExecute`
	`CBWStageExecute`
	`DataInStageExecute`
	`DataOutStageExecute`
	`CSWStageExecute`
		In this layer, ATAPI commands and the addresses for writing and reading data are received from the ATAPI layer and bulk transfers of CBW, data, and CSW are performed.

Communication	`CommunicationClassExecute`
	`COMReadExecute`
	`COMWriteExecute`
	`GetLineCodingExecute`
	`SetLineCodingExecute`
	`SetControlLineStateExecute`
		In this layer, the USB device control requests are received from the communication interface layer and class command is executed (control transfer is performed).
		The writing or reading requests are received from the communication interface layer and bulk transfers are performed.

Layer	Function	Description

Hub	`HubClassInit`
	`GetHubStatusExecute`
	`SetPortFeatureExecute`
	`ClearPortFeatureExecute`
	`GetPortStatusExecute`
	`WaitHubChange`
	`RsetHubPort`
	`ReadHubStatus`
		In this layer, executes the hub class commands to control the hub device.
		Saves the received data by the interrupt-in transfer.

USB host driver	`ControlProcess`
	`DescriptorInitGet`
	`AddressSet`
	`DescriptorGet`
	`DescriptorConfigGet`
	`DescriptorConfigAllGet`
	`ConfigurationSet`
	`GetHubDescriptor`
	`DeviceStatusGet`
	`ConfigurationSet`
	`GetHubStatus`
	`SetPortFeature`
	`ClearPortFeature`
	`GetPortStatus`
	`MaxLUNGet`
	`SetLineCoding`
	`GetLineCoding`
	`SetControlLineState`
	`BulkTransExecute`
	`BulkOutProcess`
	`BulkInProcess`
	`UsbhdInit`
		In this layer, enumeration or class command are executed by using control transfer.
		The writing or reading requests are received from the mass storage class layer and the communication layer and bulk transfer is performed.
		Executed the interrupt-in transfer when a hub device is connected.
		The host driver is initialized by using the PCI bridge driver.

PCI bridge driver	`PciInit`
	`PciConfigRead`
	`PciConfigWrite`
		In this layer, the OHCI host bridge is accessed via the PCI bridge.

Memory module	`UsbhdMemInit`
	`EDInit`
	`GTDInit`
	`UsbMemoryInit`
	`EdDescriptorCreate`
	`GtdDescriptorCreate`
	`AllocateED`
	`AllocateGTD`
	`EdDescriptorFree`
	`GtdDescriptorFree`
		In this layer, ED and TD memory areas are allocated.

13. Interrupts

This section provides a diagram indicating the correlation between functions when an interrupt occurs, describes the function specifications and operations related to the interrupt.

13.1 Correlation Between Functions When an Interrupt Occurs

A diagram indicating the correlation between functions when an interrupt occurs is shown below. When the USB device is connected, the RootHubStatusChange event occurs (the root hub detects the change in the port status) or when the operation of the TD is completed, WritebackDoneHead event occurs, this interrupt notifies those events to the CPU.

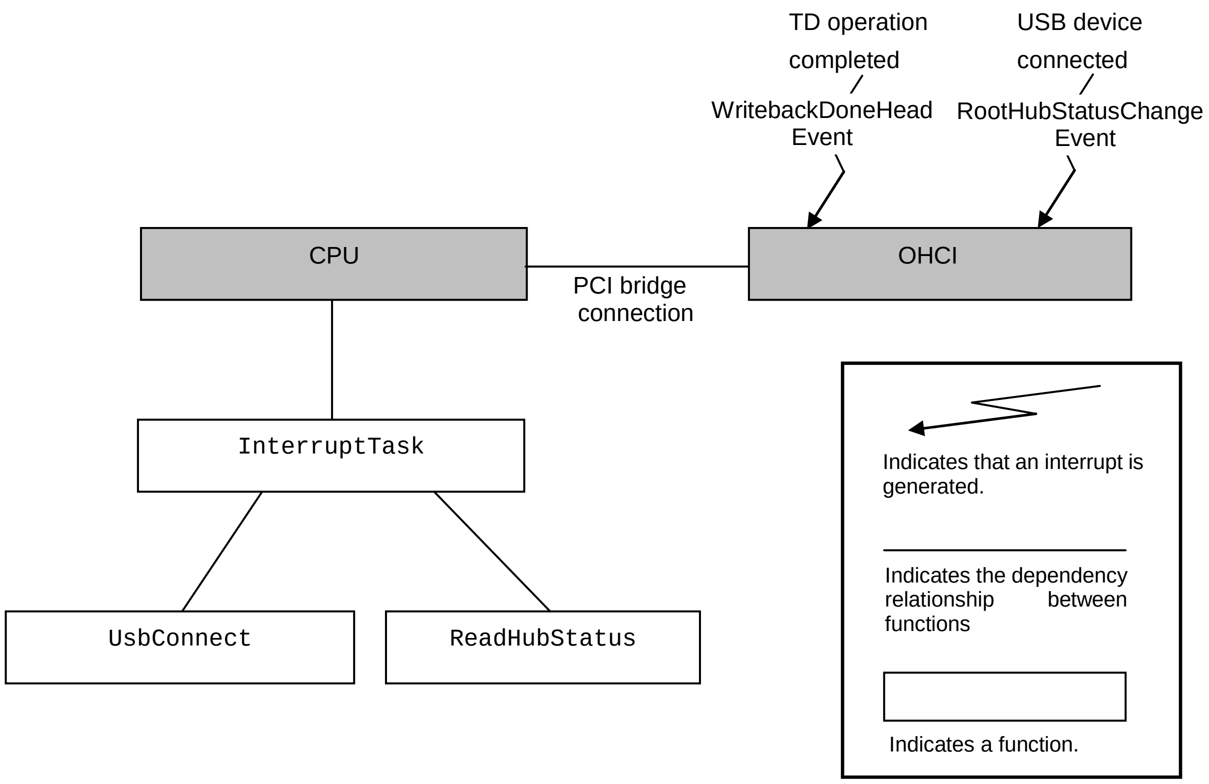

Figure 21. Correlation Between Functions When an Interrupt Occurs

13.2 Description of Functions

The following shows the function names and processing details.

Function	Description
`InterruptTask`	This function is called by the INTUSBH0 interrupt from the CPU. This function is used to read the interrupt register (HcInterruptStatus register) of the OHCI. If a USB device is connected (RootHubStatusChange bit equals to 1'b), the `UsbConnect` function is executed. If a TD's operation is completed (WritebackDoneHead bit equals to 1'b), this function checks the contents of the TD by reading HccaDoneHead register on the OHCI. If that TD is interrupt-in transfer one, the `ReadHubStatus` function is executed. If that TD is but for interrupt-in transfer one, the `write_back_done` flag is set. Note that interrupt masking for RootHubStatusChange bit and WritebackDoneHead bit must be disabled in the OHCI, PCI, and CPU in order to report interrupts.
`UsbConnect`	This function reads the port status register (HcRhPortStatus register) of the OHCI and verifies the USB device is connected to the root hub's port.
`ReadHubStatus`	Saves the received data by the interrupt-in transfer.

13.3 Function Interfaces

The following shows the interface of this function.

Function	`InterruptTask`
Parameter	void
Parameter information	–
Specifiable value	–
Return value	–
Return value usage	–
Return value meaning	–

Function	`UsbConnect`
Parameter	`void`
Parameter information	–
Specifiable value	–
Return value	STATUS
Return value usage	Used to judge whether execution of `UsbConnect` finished successfully.
Return value meaning	0: Normal completion

Function	`ReadHubStatus`
Parameter	`UsbDevInfo`
Parameter information	Driver structure (USB device information, Transfer information)
Specifiable value	Other than NULL
Return value	STATUS
Return value usage	Used to judge whether execution of `ReadHubStatus` finished successfully.
Return value meaning	0: Normal completion Other than 0: Abnormal termination

13.4 Interrupt Handling Operation

The operation when the interrupt is occurred shown below. When the USB device is connected, the RootHubStatusChange event occurs (the root hub detects the change in the port status) or when the operation of the TD is completed, WritebackDoneHead event occurs, this interrupt notifies those events to the CPU. The CPU executes the interrupt task registered to the INTUSBH0 interrupt vector.

The interrupt register in the OHCI is read and if it can be confirmed that the port status has changed, the `UsbConnect` function is executed.

The interrupt register in the OHCI is read and if it can be confirmed that WritebackDoneHead event has occured, and then read HccaDoneHead register in the OHCI to check the completed TD.

If that TD is interrupt-in transfer one, the `ReadHubStatus` function is executed.

If that TD is but for interrupt-in transfer one, the `write_back_done` flag is set.

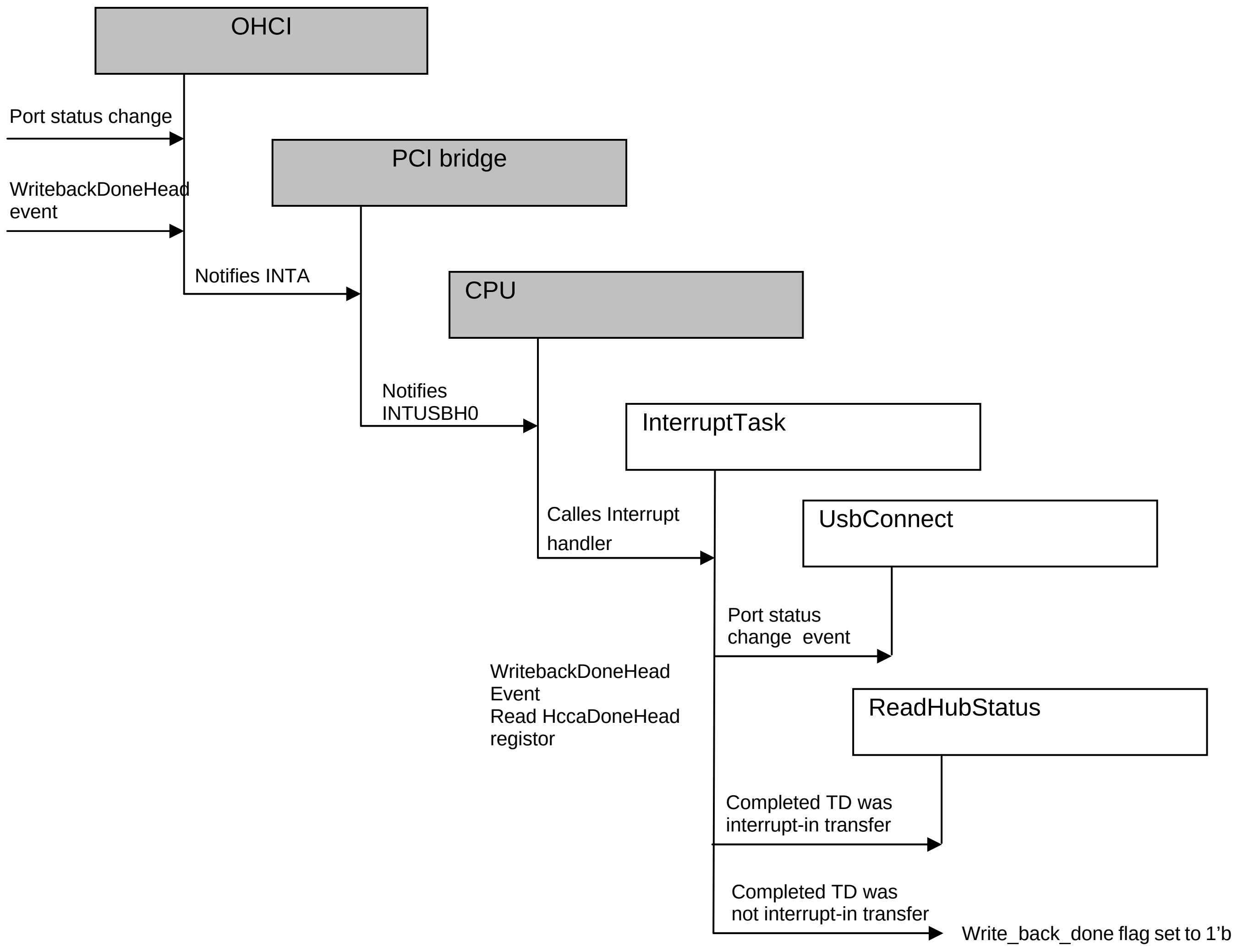

Figure 22. Interrupt Handling Operation

14. Data Structures

The data structures used by the USB host driver are shown below.

Structure	Description
USBHD_DEV_INFO	Enumeration end flag Port number USB device address USB device address to assign new connected device Maximum size of packets transferred on endpoint 0 Total size of descriptors returned by the `Get Configuration` descriptor The number of interface Configuration value Number of endpoints on the interface (Interface: 0 to 2) Interface class (Interface: 0 to 2) Interface subclass (Interface: 0 to 2) Interface protocol (Interface: 0 to 2) Number of endpoint used for interrupt transfer (Interface: 0 to 2) Attribute of endpoint used for interrupt transfer (Interface: 0 to 2) Maximum size of packets transferred on endpoint used for interrupt transfer (Interface: 0 to 2) Interval of transfer on endpoint used for interrupt transfer (Interface: 0 to 2) Number of endpoint used for bulk-in transfer (Interface: 0 to 2) Attribute of endpoint used for bulk-in transfer (Interface: 0 to 2) Maximum size of packets transferred on endpoint used for bulk-in transfer (Interface: 0 to 2) Number of endpoint used for bulk-out transfer (Interface: 0 to 2) Attribute of endpoint used for bulk-out transfer (Interface: 0 to 2) Maximum size of packets transferred on endpoint used for bulk-out transfer (Interface: 0 to 2) HCCA base address DeviceStatus Number of interface used for the mass storage class Number of interface used for the communication class Number of interface used for the data interface class Number of interface used for the hub class MaxLUN value CBW tag Start flag of the USB device applied to the communication class Number of downstream ports of the hub device Hub status change bitmap data got by interrupt-in transfer Hub status Port Status Port change status Address for device requests Address of receive data in control transfer Size of receive data in control transfer Address of transmit data in control transfer Size of transmit data in control transfer Address of data for bulk transfer Size of data for bulk-out transfer Size of data for bulk-in transfer Descriptor (ED, TD) information CBW data address

Structure	Description
	Address of data for mass storage CSW data address Address of data for the communication class Address of parameter for the communication class Address of parameter for the hub class Buffer address for storing received data Buffer address for storing data to transmit Data size Pointer to the `FileInfo` structure
USBHD_DESC_INFO	Address of ED for control transfer Address of TD for setup data Address of TD for data stage Address of TD for status stage Address of dummy TD for control transfer Address of ED for bulk-out transfer (Interface: 0 to 2) Address of ED for bulk-in transfer (Interface: 0 to 2) Address of TD for bulk-out transfer (Interface: 0 to 2) Address of TD for bulk-in transfer (Interface: 0 to 2) Address of dummy TD for bulk-out transfer (Interface: 0 to 2) Address of dummy TD for bulk-in transfer (Interface: 0 to 2) Address of ED for interrupt-in transfer (0 to 31) Address of TD for interrupt-in transfer Address of dummy TD for interrupt-in transfer
FILEINFO	Logical block number Data size Data count ATAPI command Sector size Sector number Pointer to data to be written Pointer to data to be read Data pointer File name Mode
LINE_CODING_STRUCT	Transfer speed (bps) Stop bit Parity Data bit
ED	For details about data, see *Table 2* and *Table 3*.
TD	For details about data, see *Table 4* and *Table 5*.

Table 2. Overview of Endpoint Descriptor (ED) Format

	31 27	26 16	15	14	13	12 11	10 7	6 4	3 2	1	0
Word0	–	`MPS`	`F`	`K`	`S`	`D`	`EN`	`FA`			
Word1	TD Queue Tail Pointer (`TailP`)								–		
Word2	TD Queue Head Pointer (`HeadP`)								0	C	H
Word3	Next Endpoint Descriptor (`NextED`)								–		

Table 3. Details of Endpoint Descriptor (ED) Format

Name	Read/write from the host controller	Description
`MPS`	R	Indicates the maximum number of bytes that can be sent or received in one USB packet by the endpoint on the USB device.
`F`	R	0: Isochronous transfer 1: Control/bulk/interrupt transfer
`K`	R	By setting this bit, control can be transferred to the next ED without performing the communication specified for this endpoint.
`S`	R	0: Full speed 1: Low speed
`D`	R	Direction of communication 00: Specified by the TD field 01: OUT 10: IN
`EN`	R	Number of the communication partner endpoint.
`FA`	R	Address of the communication partner USB device
`C`	R/W	This bit holds the final toggle information before the TD is retired.
`H`	R/W	This bit is set when the host controller wants to stop the processing of this ED. This bit is set if an error occurs during normal TD processing.
`TailP`	R	This bit holds the address of the last TD linked to this ED. If `TailP` and `HeadP` are the same, there will be no TD for this ED to process.
`HeadP`	R/W	First TD linked to this ED This field is updated to the address of the next TD each time the TD indicated by this field has finished processing.
`NextED`	R	Address of ED linked to this ED Set to 0 if there is no ED to link.

Table 4. Overview of Transfer Descriptor (TD) Format

	31 28	27 26	25 24	23 21	20 19	18	17 4	3 0
Word0	`CC`	`EC`	`T`	`DI`	`DP`	`R`	–	
Word1	Current Buffer Pointer (`CBP`)							
Word0	Next TD (`NextTD`)							
Word2	Buffer End (`BE`)							

Table 5. Details of Transfer Descriptor (TD) Format

Name	Read/write from the host controller	Description
`CC`	R/W	Indicates the status of the last transfer created from this TD
`EC`	R/W	This bit is incremented each time a transfer error occurs. If a transfer error occurs when this bit is 2, the error type is recorded to the CC bit and the TD is shifted to the `Done` queue.
`T`	R/W	Used to create and compare the PID of the data packet (DATA0/DATA1). This bit is updated each time transmission or reception is completed successfully.
`DI`	R	Specifies the timing at which an interrupt will occur when TD processing is complete. 0: Interrupt occurs as soon as processing is complete 1: Interrupt occurs at the end of the frame following completion of TD processing 111: Interrupt does not occur
`DP`	R	Specifies the transfer direction and PID 00: SETUP To endpoint 01: OUT To endpoint 10: IN From endpoint
`R`	R	An error occurs if the last packet created from this TD is a short packet. If this bit is set to 1, the error is ignored.
`CBP`	R/W	Indicates the buffer area for transferring data to and from endpoints Always indicates the address of the buffer area that should be accessed next. If this bit is set to 0, it indicates that data of size 0 is specified to be transferred or that transfer is complete.
`NextTD`	R/W	Pointer the next TD
`BE`	R	Last address of the buffer area

15. Reference Documents

- Hardware manual
  V850E2/ML4 Hardware User's Manual [R01UH0262EJ]
  (Download the latest revision from the Renesas Electronics website.)

- Software manual
  V850E2M Architecture User's Manual [R01US0001EJ]
  (Download the latest revision from the Renesas Electronics website.)

- Universal Serial Bus Revision 2.0 specification

- OpenHCI Open Host Controller Interface specification for USB Release 1.0a

- Universal Serial Bus Mass Storage Class Bulk-Only Transport Revision 1.0

- Universal Serial Bus Class Definitions for Communications Devices Revision 1.2

- Universal Serial Bus Class Subclass Specification for PSTN Devices s Revision 1.2

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/

All trademarks and registered trademarks are the property of their respective owners.

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jun. 22, 2012	—	First edition issued
1.01	Sep. 14, 2012	—	Implements M3S-TFAT-Tiny (Renesas original FAT File System)
1.02	Jan. 10, 2013	—	Hub class added (Multiple device control)

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
   - Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on
   - The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses
   - Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals
   - After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products
   - Before changing from one product to another, i.e. to one with a different part number, confirm that the change will not lead to problems.

   — The characteristics of MPU/MCU in the same group but having different part numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different part numbers, implement a system-evaluation test for each of the products.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.
(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

© 2012 Renesas Electronics Corporation. All rights reserved.
Colophon 2.0