To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

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

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. 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.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. 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.
5. When exporting the 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. You should not use Renesas Electronics products or the 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. 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.
6. 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.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. 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 categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. 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 an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

"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.
"High Quality":	Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
"Specific":	Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.

8. 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.
9. 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 system manufactured by you.
10. 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.
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.

Application Note

32

SH7727 Group USB Host Module

Application Note

Renesas 32 bit RISC Microcomputer SuperH^{TM} RISC engine Family/ SH7700 Series

Renesas Electronics
www.renesas.com

Rev.1.00 2003.4

Cautions

Keep safety first in your circuit designs!

1. Renesas Technology Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage. Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
2. Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors. Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (http://www.renesas.com).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
   Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.

Microsoft® Windows® 98 and Windows® 2000 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Preface

This application note describes the firmware that uses the USB Host Module in the SH7727. This is provided to be used as a reference when the user creates USB Host Module firmware.

This application note and the described software are application examples of the USB Host Module, and their contents and operation are not guaranteed.

In addition to this application note, the manuals listed below are also available for reference when developing applications.

[Related manuals]

- Universal Serial Bus Specification Revision 1.1
- Open Host Controller Interface Specification for USB Revision 1.0a
- SH7727 Hardware Manual
- SH7727 Solution Engine (MS7727SE01) Instruction Manual
- SH7727 E10A Emulator User's Manual

[Caution] The sample programs described in this application note do not include firmware related to isochronous transfer of USB transfer types. When using this transfer type, the user needs to create the program for it.

Also, the hardware specifications of the SH7727 and SH7727 Solution Engine, which will be necessary when developing the system described above, are described in this application note, but more detailed information is available in the SH7727 Hardware Manual and the SH7727 Solution Engine Instruction Manual.

Contents

RENESAS

Section 1   Overview

This application note describes how to use the USB host module that is incorporated in the SH7727, and explanation of sample programs.

The features of the USB host module incorporated in the SH7727 are listed below.

- Conforming to the Open Host Controller Interface (OHCI) 1.0 register set
- Conforming to USB1.1
- Root hub function
- Low-speed (1.5 Mbps) and full-speed (12 Mbps) transfer supported
- Detection of overcurrent supported
- User memory area connected to the SH7727 can be used for transfer data and descriptor storage

Figure 1.1 shows an example of a system configuration for executing a sample program.

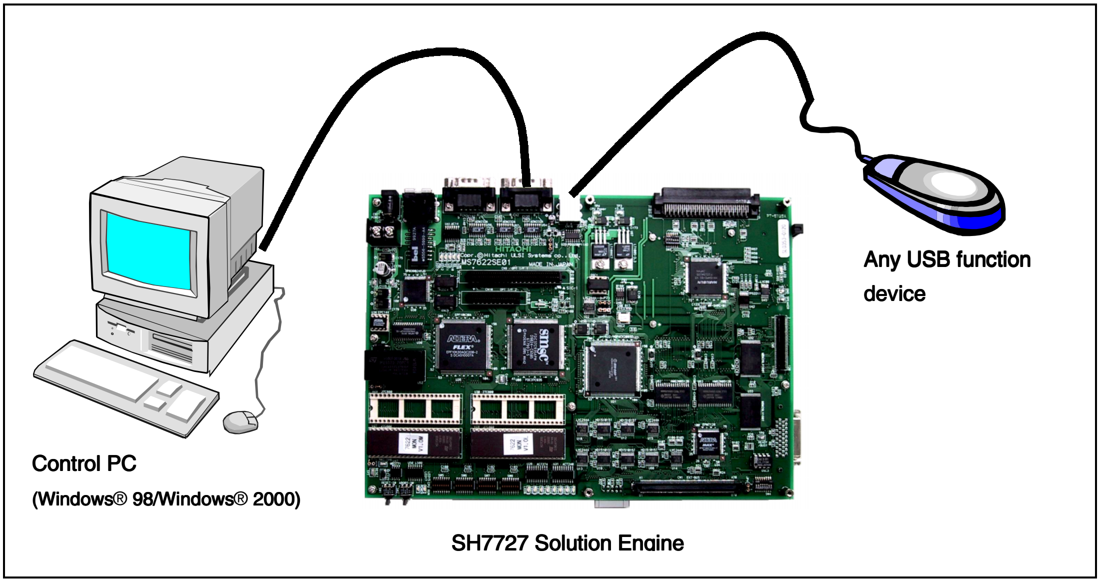


Figure 1.1   System Configuration Example

This system is configured with the SH7727 Solution Engine made by Hitachi ULSI Systems Co., Ltd. (hereafter referred to as the SH7727SE) and a PC running on Windows® 98/2000.

In this system, the host PC and the SH7727SE are connected through serial cable. USB packets can be generated to the USB function devices connected to the SH7727SE by the USB packet generation tool, RequestGenerator.

This system offers the following features.

1. The sample program can be used to evaluate the USB host module of the SH7727 quickly.
2. The sample program supports control, bulk, and interrupt transfers.
3. Additional programs can be created to support isochronous transfer. *

Note: * Isochronous transfer program is not provided, and will need to be created by the user.

Section 2 Overview of the Open Host Controller Interface (OpenHCI) Specification

The USB host module incorporated in the SH7727 supports to the Open Host Controller Interface (hereafter referred to as OpenHCI) specification. The OpenHCI specification is explained in this section. Refer to this section when developing a USB host system. For details of the OpenHCI specification, see Revision 1.0a of the Open Host Controller Interface Specification for USB.

2.1 OpenHCI Standard

The hierarchical structure of a USB-host system consists of the USB device, host controller (HC), host controller driver (HCD), USB driver (USBD), and client-software levels. The OpenHCI specification prescribes the functions of the HCD and HC and the interface between them: that is, the interface between the software and hardware.

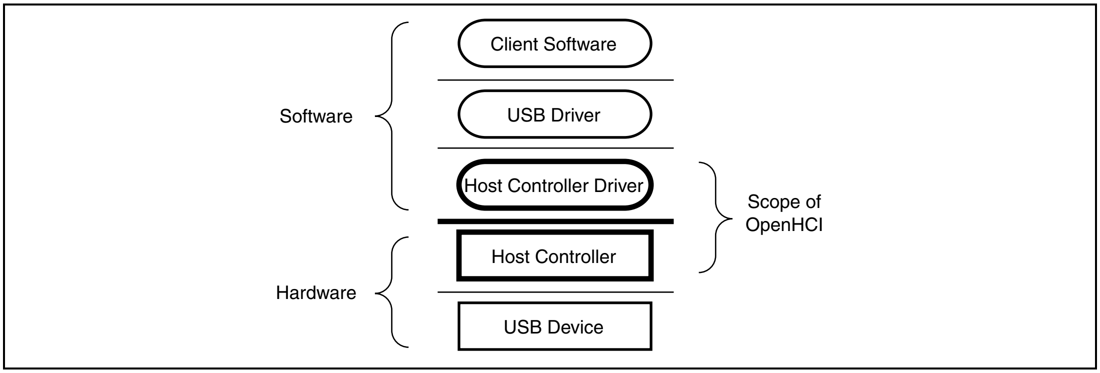


Figure 2.1 USB Focus Areas

2.2 Data-Transfer Types

The four data-transfer types defined for the USB are listed below.

Interrupt Transfer: Small amounts of data are transferred periodically. The transfer interval can be optimized to suit the requirements of the device.

Isochronous Transfer: Transfer is performed periodically with a constant data rate.

Control Transfer: Configuration, command, and status information of the device is transferred nonperiodically.

Bulk Transfer: Large amounts of data are transferred nonperiodically.

In the OpenHCI specification, the four data-transfer types are classified into two categories: periodic and non-periodic. Interrupt and isochronous transfers are classified as periodic, since they are scheduled to run at periodic intervals. Control and bulk transfer are classified as non-periodic, since they are not scheduled to run at periodic intervals.

2.3 Host-Controller Interface

Communication between the HCD and HC is through the transmission/reception of Endpoint Descriptors (EDs) and Transfer Descriptors (TDs). An ED contains the information on an endpoint: the device address, speed, and maximum packet size of the device to which the corresponding endpoint belongs and the endpoint number. The TD contains the information on the data packets to be transferred to an endpoint: PID, data-toggle information, address of transmitted/received data in memory, and status information after the transfer has been completed. The HCD gathers the descriptors and places them in lists according to the transfer type, then sends the head addresses of the lists to the HC. The head address of a list is transmitted or received via an internal register of the HC or the Host Controller Communications Area (HCCA) which is set up in the memory. The HC passes each TD for which transfer has been completed to the HCD by placing it at the head address of the Done Queue in the HCCA.

Details of the processing of lists, EDs, TDs, the HCCA, and the interface between the HCD and HC are given below.

2.3.1 Lists

A maximum of 127 USB-function devices can be connected in a single USB system, and each USB-function device has a maximum of 15 endpoints. As more than one endpoint can be specified, one ED is prepared to gather the information for each endpoint. The OpenHCI specification provides for groups of EDs, which include all EDs for a given transfer type. A group thus linked is called a list. Each TD has to have a target endpoint, and so is placed in the queue of the ED that corresponds to this endpoint. In other words, one or more TDs are linked, in an FIFO queue, to each ED. The EDs and TDs for each transfer type are aggregated in lists of the type shown in figure 2.2.

The HCD controls the address of the head ED of a list as a pointer to the list (the head pointer of figure 2.2). Passing the head pointer of a list to the HC (writing the information to a register of the HC) allows the HC to access that list.

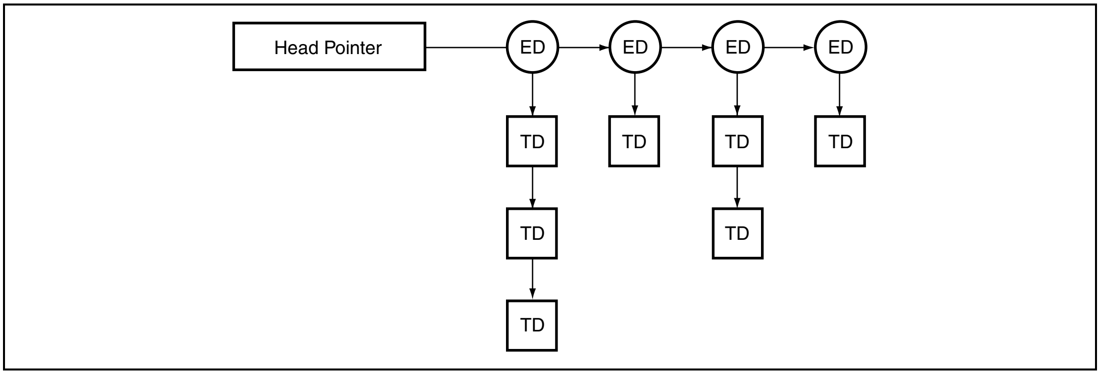


Figure 2.2 Typical List Structure

Lists for Non-Periodic Transfer: Figure 2.2 shows the structure of the lists for bulk transfer and control transfer, which are categorized as non-periodic. Each list has one head pointer and these head pointers are stored in the registers of the HC (HcControlHeadED and HcBulkHeadED). Although bulk transfer and control transfer are non-periodic, i.e., they occur asynchronously, the HC still executes these forms of transfer by reading their head pointers.

Lists for Periodic Transfer: Interrupt transfer and isochronous transfer are categorized as periodic forms of transfer and their lists are treated as a single unit. The first ED of the isochronous-transfer list is linked to the last ED of the interrupt-transfer list. The Open HCI specification thus has three lists: the non-periodic control-transfer and bulk-transfer lists and the single periodic-transfer list.

The periodic-transfer list is as shown in figure 2.3. This list differs from those for non-periodic transfer in the number of head pointers. There are 32 head pointers and each is referred to at a 32-ms interval (i.e., 32-frame interval). Since the EDs for periodic transfer have to be referred to at specific intervals, the list for periodic transfer is organized into the tree structure shown in figure 2.3.

The OpenHCI specification provides fixed polling rates of 32 ms, 16 ms, 8 ms, 4 ms, 2 ms, and 1 ms for interrupt transfer. The polling rate for each interrupt-transfer ED is chosen from among these rates. Since the isochronous-transfer ED has to be accessed at 1-ms intervals, it is linked to the interrupt-transfer ED which has a polling rate of 1 ms. As is shown in figure 2.3, each interrupt-transfer ED, which is polled every 32 ms, is linked to a head pointer. Each of the interrupt-transfer EDs with a 16-ms polling rate is obtained by linking it to two 32-ms head pointers. These EDs are thus accessed twice in every 32-ms interval; i.e., once in every 16-ms interval. The interrupt EDs with polling rates of 8, 4, and 2 ms are accessed from four, eight, and 16 head pointers, respectively. The interrupt-transfer ED polled every 1 ms is accessed from all of the head pointers. The isochronous ED is linked with the tree after the interrupt ED that is polled every 1 ms. The periodic-transfer EDs are thus organized into a tree structure.

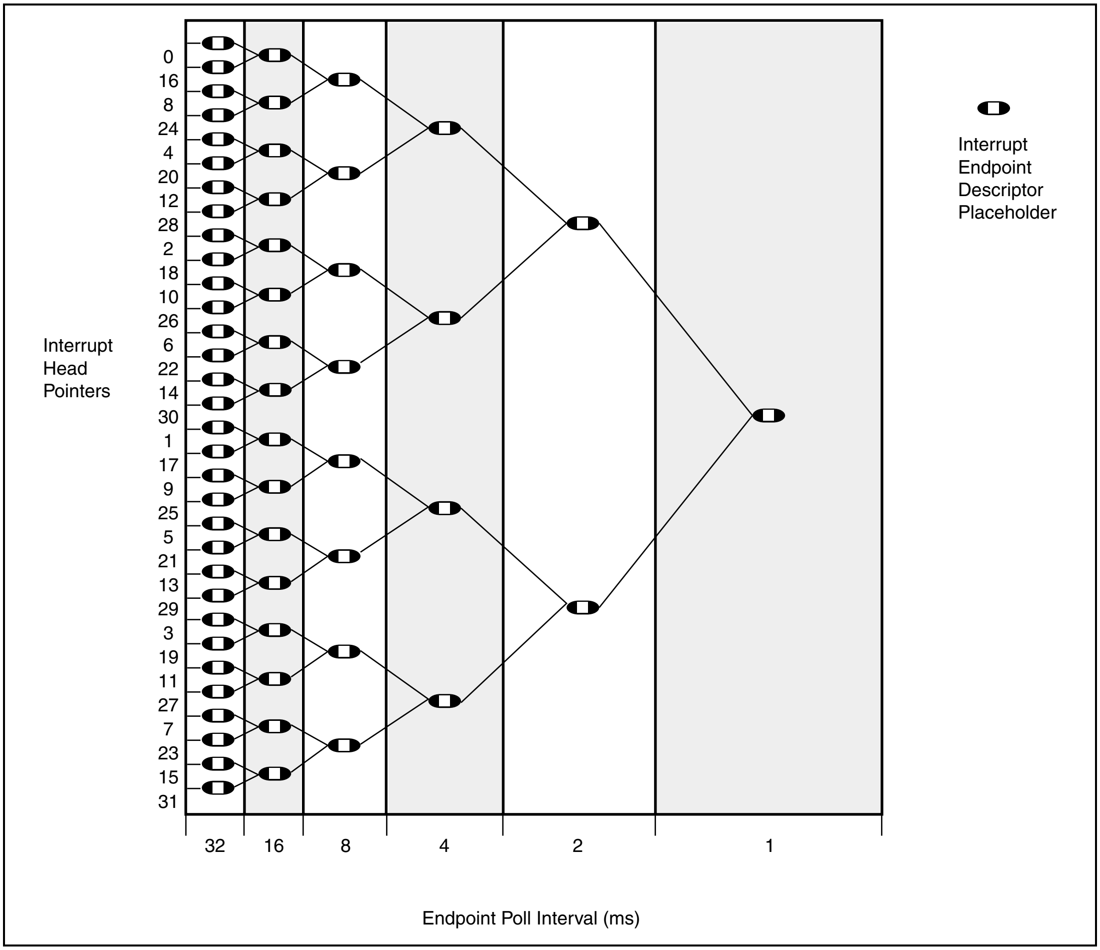


Figure2.3 Interrupt List Structure

An example of a list for periodic transfer is shown in figure 2.4. This sample consists of a single interrupt-transfer ED polled every 4 ms, two EDs each for polling every 32, 16, 8, 2, and 1 ms, and the single isochronous ED.

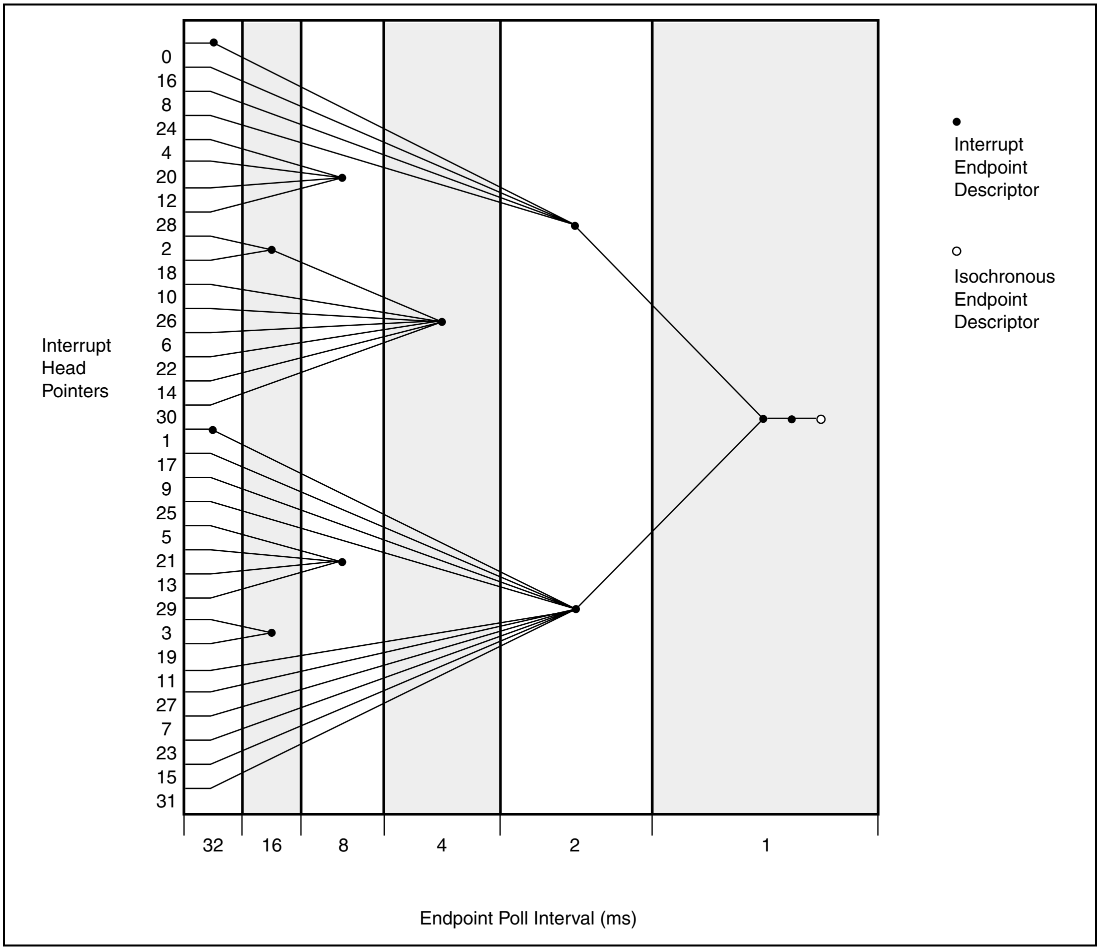

Interrupt Head Pointers
0
16
8
24
4
20
12
28
2
18
10
26
6
22
14
30
1
17
9
25
5
21
13
29
3
19
11
27
7
23
15
31
Interrupt Endpoint Descriptor
Isochronous Endpoint Descriptor
32
16
8
4
2
1
Endpoint Poll Interval (ms)

Figure 2.4 Example of Periodic ED List

2.3.2 Endpoint Descriptors (EDs)

The HCD prepares an Endpoint Descriptor (ED) for each endpoint of communications and gathers the information necessary to communicate with the endpoints. An ED consists of 16 bytes and must be aligned with a 16-byte boundary. Figure 2.5 shows the format of an ED and table 2.1 lists the details of the individual fields.

As was explained in section 2.3.1, all EDs for a given transfer type are linked to form a list by placing the address of the next ED in the NextED field of each ED. A zero in this field indicates that the ED is not linked with any other ED.

After confirming that neither the sKip nor the Halted bit is set to 1, the HC compares TailP to HeadP. If they are the same, the HC recognizes that no TD is queued for processing and starts processing the next ED. If TailP and HeadP have different values, the TD indicated by HeadP will be processed. Data is transmitted from the buffer area of the TD, packet-by-packet, or a received packet is written to the buffer area. After processing of a TD has been completed, the value in the NextTD field of this TD is copied to the HeadP field of ED, and the processed TD is moved to the Done queue.

	31	26 … 16	15	14	13	12 11	10 … 07	06 05 04	03 02	01	00
Dword 0	–	MPS	F	K	S	D	EN	FA			
Dword 1	TD Queue Tail Pointer (TailP)								–		
Dword 2	TD Queue Head Pointer (HeadP)								0	C	H
Dword 3	Next Endpoint Descriptor (NextED)								–		

Notes: 1. Fields '–' are not accessed or modified by the HC and are available for use by the HC for any purpose.
2. Fields '0' must be processed by the HC after clearing the HCD to 0.

Figure 2.5 Endpoint Descriptor

Table 2.1 Field Definitions for Endpoint Descriptor

Name	HC Access	Description
FA	R	FunctionAddress Address of a USB-function device
EN	R	EndpointNumber Number of the corresponding endpoint
D	R	Direction This field indicates the direction of data flow (IN or OUT). The direction may also be determined by using the PID field of the TD. Code / Direction: 00b — Get direction from TD 01b — OUT 10b — IN 11b — Get direction from TD
S	R	Speed Indicates the speed of the endpoint: full-speed (S = 0) or low-speed (S = 1)
K	R	sKip When this bit is set, HC continues on to the next ED without attempting communication for this endpoint.
F	R	Format This bit indicates the format of the TDs linked to this ED. If this is a control, bulk, or interrupt endpoint, then F = 0, indicating that the General TD format is used. If this is an isochronous endpoint, then F = 1, indicating that the Isochronous TD format is used.

Name	HC Access	Description
MPS	R	MaximumPacketSize This field indicates the maximum number of bytes that can be sent to or received from the endpoint in a single USB packet.
TailP	R	TDQueueTailPointer The address of the last TD linked with this ED is stored. If TailP and HeadP are the same, this ED contains no TD that the HC can process. If TailP and HeadP are different, this ED contains a TD to be processed by the HC.
H	R/W	Halted This bit is set by the HC to halt the processing of ED. This bit is set, usually due to an error in processing a TD.
C	R/W	ToggleCarry Whenever a TD is retired, this bit is written to contain the last data toggle value (LSb of DataToggle field) from the retired TD. This field is not used for Isochronous Endpoints.
Head P	R/W	TDQueueHeadPointer The first TD linked with this ED. Whenever the processing of TD indicated by this field has been completed, the value set to the address of next TD linked with this TD is incremented.
NextED	R	NextED Address of the ED linked with this ED. If no ED to link, clear this bit to 0.

The HCD sets the sKip bit to stop processing of an ED. When a sKip bit is set to 1, the HC skips processing of that ED and begins processing the ED indicated by NextED.

If an error occurs during the processing of a TD, the Halt bit is set, the TD that caused the error is moved to the Done Queue, and the HeadP bit is updated by the HC. After the source of the error has been eliminated, the HCD clears the Halt bit and processing of the ED recommences.

2.3.3 Transfer Descriptor (TD)

A TD is a data structure placed in memory by the HC to define the packets of data that are transmitted to/received from an endpoint. TDs are categorized into General Transfer Descriptors (GTDs) and Isochronous Transfer Descriptors (ITDs). The GTD is used in control transfer, bulk transfer, and interrupt transfer, while the ITD is used in isochronous transfer. Both the GTD and ITD specify a buffer with a length of 0 to 8192 bytes.

A TD is linked to the ED which corresponds to the endpoint for the transfer. The HCD generates TDs and links TDs to EDs. The HC processes each TD and, after the processing has been completed, the TD is moved from the ED to the Done Queue.

The details of the GTD and ITD are explained below.

General Transfer Descriptor (GTD): The Transfer Descriptors (TDs) that are used in control, bulk, and interrupt transfer all have the same format and are called General TDs (GTDs). The format and individual fields of the GDT are shown in figure 2.6 and table 2.2, respectively. The GTD is 16 bytes long and should be aligned with a 16-byte boundary.

	31 28	27 26	25 24	23 21	20 19	18		03 00
Dword 0	CC	EC	T	DI	DP	R	–	
Dword 1	Current Buffer Pointer (CBP)							
Dword 2	Next TD (NextTD)							0
Dword 3	Buffer End (BE)							

Notes: 1. The all fields should not be modified while a TD is accessed by the HC.
2. Fields '–' are not written to nor changed the values by the HC.

Figure 2.6 General Transfer Descriptor

RENESAS

Table 2.2 Field Definitions for General TD

Name	HC Access	Description
R	R	bufferRounding If the last packet generated by GTD is short packet, a DataUnderrun error is generated. When this field is set to 1, a DataUnderrun error is neglected.
DP	R	Direction/PID Direction of transfer and PID are set. Code | PID Type | Data Direction 00b | SETUP | to endpoint 01b | OUT | to endpoint 10b | IN | from endpoint 11b | Reserved |
DI	R	DelayInterrupt The timing of an interrupt (WriteBackDoneHead) generation which is generated after the TD has been completed is defined. The HC waits for frames set by this field before generating an interrupt. If setting value is 0, an interrupt is generated immediately. If the setting value is 1, an interrupt is generated after waiting for one frame. If setting value is 111b, no interrupt is generated.
T	R/W	DataToggle Sets the PID (DATA0/DATA1) of data packet by LSb. It is updated after each successful transmission/reception of a data packet. The MSb of this bit is 0 when the data toggle value is acquired from the toggleCarry field in the ED. The MSb of this bit is 1 when the data toggle value is acquired from this field.
EC	R/W	ErrorCount For each transmission error, this value is incremented. If ErrorCount is 2 and another transfer error occurs, the error type is recorded in the ConditionCode field and placed on the Done Queue. When a transaction completes without error, ErrorCount is reset to 0.
CC	R/W	ConditionCode Indicates the status of the last transfer generated by GTD.
CBP	R/W	CurrentBufferPointer Indicates the buffer area for transfer to/from the endpoint. The address of buffer to be accessed next is always indicated. When CurrentBufferPointer is 0, data of size 0 will be transferred or the transfer has been completed.
NextTD	R/W	NextTD Points the next TD.
BE	R	BufferEnd Indicates the last address of the buffer area.

The CurrentBufferPointer of the GTD indicates the address of the data buffer and is referred to when the HC generates a data packet for the endpoint indicated by the ED with which the GTD is linked. On completion of the transfer of each packet generated from the GTD, the result of the transfer is written to the ConditionCode field. When the transfer has been completed without error, the value of the CurrentBufferPointer is updated by the amount transferred.

The MSb of the DataToggle field indicates the source of the data PID value of the first data packet generated from this TD. When MSb = 0, the source is the toggleCarry bit of the ED and is not LSb of DataToggle field of the GTD. When MSb is 0, the source is the LSb of DataToggle field.

Bulk and interrupt transfers usually start with the DataToggle field set to 00b. When a TD is retired and processing switches to the next TD, the toggle information is carried over; the LSb of the DataToggle field is copied to the toggleCarry bit and then to the LSb of the DataToggle field in the next TD. Toggle information is thus carried over and the DataPID is toggled when processing moves from one TD to the next.

For a control transfer, the DataPacketPID is Data0 in the Setup stage, Data1 in the first packet of the Data stage, and Data1 in the status stage. Therefore, the more significant bit of DataToggle is set to 1 in the TD of each stage, so that information is obtained from the less significant bit of the DataToggle field rather than from the previous TD via the ED.

Processing of the TD is completed in either of two ways: all data between the CurrentBufferPointer and BufferEnd specifications of the TD has been transferred or a data packet from the endpoint was smaller than MaxPacketSize. In the former case, the TD is moved from the ED to the Done Queue. In the latter case, if BufferRounding is set, normal transfer-completion processing is performed. If BufferRounding is not set, a DataUnderrun error occurs and the Halt field in the ED is set.

Isochronous Transfer Descriptor (ITD): The Isochronous TD replaces the General TD in transfer to/from the isochronous endpoints. The format of the ITD and details of its fields are shown in figure 2.7 and table 2.3, respectively.

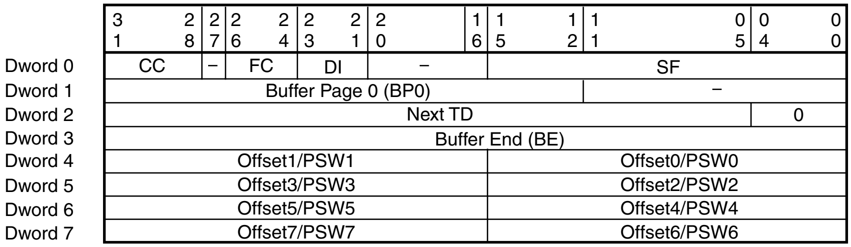


Figure 2.7 Isochronous TD Format

The data packets generated from an Isochronous TD have to be transferred within specific frames. From 1 to 8 consecutive frames of data to be transferred are set in an Isochronous TD, FrameCount +1 frames of data are consecutively transferred, with the first packet placed in the frame specified by StartingFrame. When the difference between the HcFmNumber register and the StartingFrame is R, data packets are isochronously transferred until R has been incremented from 0 to the number in FrameCount. Offset[R] determines the first buffer address of each frame. The lower 12 bits are always specified by Offset[R]. The upper 20 bits are given by BufferPage0 when the 12th bit of Offset[R] is 0 and by the 20 higher-order bits of BufferEnd when the 12th bit of Offset [R] is 1. The last address in the buffer for each frame from R = 0 to FrameCount−1 is defined by Offset[R+1] −1 and the same 20 higher-order bits as are defined for the first address. The last address of the last data packet (R = FrameCount) is BufferEnd. On completion of the transfer of each frame, the Offset/PSWN fields of the frame become PSWN fields; the ConditionCode and SizeOfPacket data are written to these fields.

Table 2.3 Field Definitions for Isochronous TD

Name	HC Access	Description
SF	R	StartingFrame Indicates the transfer start frame of data packet.
DI	R	DelayInterrupt As the field of GTD, indicates how long the interrupt generation is delayed when ITD is completed.

FC	R	FrameCount
		Indicates the number of data packet transferred (the number of frame to be transferred) from ITD.
		FrameCount = 0: One data packet
		FrameCount = 7: Eight data packets

CC	R/W	ConditionCode Stores the condition code when the Isochronous TD is moved to the Done Queue.
BP0	R	BufferPage0 Indicates the first address of the data buffer.
NextTD	R/W	NextTD Indicates the address of the next IsochronousTD.
BE	R	BufferEnd Indicates the last address of a buffer.

Name	HC Access	Description
OffsetN	R	Offset Indicates the head address of Isochronous Data Packets.

15 13	12 0
7	Offset

Name	HC R/W	Description
Offset	R	**Offset** 13 bits. Set the lower 12-bit of transfer start address by lower 12-bit of Offset. The way of obtaining the value of upper 20-bit is selected by the 12th bit. When the value is 0, the value of the upper 20-bit of BufferPage0 is used. When the value is 1, the value of the upper 20-bit of BufferEnd is used.

Name	HC Access	Description
PSWN	W	PacketStatusWord Stores the condition code when Isochronous Data Packets are transferred completely.

15 12	11	10 0
CC	0	Size

Name	HC R/W	Description
Size	R	**Size of Packet** 11 bits. Indicates the size of reception on an IN transfer. This bit is 0, on an OUT transfer.
CC	R	**Condition Code** When this field indicates NotAccessed, format of OffsetN/PSWN is Offset. When this field indicates other than NotAccessed, format of OffsetN/PSWN is PacketStatusWord.

Condition Code: The values that can be set in ConditionCode fields of TDs are listed in table 2.4. The ConditionCode for a General TD is set when the TD is moved to the Done Queue. Condition codes appear in two places within an Isochronous TD: in the ConditionCode field of Dword0 and the Offset/PacketStatusWord fields. After each data packet has been transferred, the condition code is placed in the corresponding Offset/PacketStatusWord. When the TD is moved to the Done Queue, condition codes are placed in the ConditionCode field of Dword0.

Table 2.4 Condition Code

Code	Meaning	Description
0000	NOERROR	Data packet processing completed with no detected errors.
0001	CRC	Last data packet from endpoint contained a CRC error.
0010	BITSTUFFING	Last data packet from endpoint contained a bit stuffing violation.
0011	DATATOGGLEMISMATCH	Last packet from endpoint had data toggle PID that did not match the expected value.
0100	STALL	TD was moved to the Done Queue because the endpoint returned a STALL PID.
0101	DEVICENOTRESPONDING	Device did not respond to token (IN) or did not provide a handshake (OUT).
0110	PIDCHECKFAILURE	DataPID (IN) and Handshake (OUT) are error PID.
0111	UNEXPECTEDPID	Receive PID was not valid.
1000	DATAOVERRUN	The amount of data packet more than MaxPacketSize of endpoint has been received or the amount of total receive size is more than that was expected.
1001	DATAUNDERRUN	The amount of data less than MaxPacketSize has been received or the amount of total transfer size is less than that was expected.
1010	Reserved	
1011	Reserved	
1100	BUFFEROVERRUN	During an IN transfer, received data from endpoint faster than it could be written to system memory. Only for the isochronous TD.
1101	BUFFERUNDERRUN	During an OUT transfer, read access to the system memory can not be executed fast enough to keep up with USB transfer rate.
111x	NOTACCESSED	This code is set by software before TD is generated and listed. If the value of the ConditionCode is not changed, processing have not be done by the HC.

When an error occurs, the corresponding Condition Code is set in the ConditionCode field of the TD and the Halt bit of the ED is set.

Errors are categorized into the four types shown below:

- Transmission Errors
- Sequence Errors
- System Errors
- Time Errors

Transmission errors are errors that occur in communicating information over the USB wires and manifest themselves as CRC errors, BITSTUFFING errors, DEVICENOTRESPONDING errors. When this error occurs, the TD is not immediately moved to the Done Queue and a transaction is retried. If, however, this error occurs three times in a row, or another error occurs after two transmission errors, the TD is moved to the Done Queue.

Sequence errors occur when the amount of data received from an endpoint does not match the expected amount. The categories of sequence error are STALL, DATAOVERRUN, and DATAUNDERRUN. Once such errors have occurred, the corresponding TDs are moved to the Done Queue.

A system error is the occurrence of trouble to do with the system environment of the HC. The categories of system error are BUFFEROVERRUN and BUFFERUNDERRUN. Such errors only occur in the processing of Isochronous TDs. Those do not occur in the processing of General TDs.

Time errors also only occur in the processing of Isochronous TDs. The categories of time error are skipped packets and late retirement. Since each data packet of an Isochronous TD has to be transferred in a specific frame, the situation can arise where it is impossible to transfer an isochronous data packet in the corresponding frame. If it’s not possible to send a data packet in the frame in which it should have been sent, that transfer is skipped and the condition code for the data packet is set to NOTACCESSED. The packet is skipped and the next one is processed. When a packet has been skipped but processing of the last data packet is completed, the ConditionCode in Dword0 is set to NOERROR and the TD is retired. However, if the last data packet is skipped, the ConditionCode DATAOVERRUN is placed in Dword0, and the TD is retired. In this case, processing of the ED is not halted and processing of the next Isochronous TD soon starts.

2.3.4 Host Controller Communications Area (HCCA)

The Host Controller Communications Area (HCCA) is used to transfer various information between the HCD and HC. The format of the HCCA is shown in table 2.5. The HCCA consists of 256 bytes and should be located on a 256-byte boundary. The HCD is able to get information from the HCCA through memory access alone, i.e., without directly accessing the HC.

Table 2.5 Host Controller Communications Area

Offset	Size (bytes)	Name	R/W	Description
0	128	HccaInterruptTable	R	32 pointers to Interrupt ED
0x80	2	HccaFrameNumber	W	Current frame number. This field is updated by the HC before it begins processing EDs of each frame.
0x82	2	HccaPad1	W	When the HC updates HccaFrameNumber, it sets this word to 0.
0x84	4	HccaDoneHead	W	When the HC reaches the end of a frame and WriteBackDoneHead interrupt is enabled, the HC writes the value of HcDoneHead in this field. Once the HC writes in this field, the HC never write in this field until software clears the WD bit in the HcInterruptStatus. When LSb of this field is set to1, indicates that interrupts other than WriteBackDoneHead are generated when this field is written by the HC.
0x88	116	reserved	R/W	Reserved

Pointers to the 32 Interrupt EDs are placed in the HccaInterruptTable. As was explained in section 2.3.1, the list for periodic transfer has 32 head pointers. The HccaInterruptTable is used to store these pointers. The HC accesses the HccaInterruptTable once per frame and obtains the corresponding pointer.

The HccaFrameNumber field is updated by the HC on every frame. The HC updates HcFrameNumber before issuing the start-of-frame packet (SOF). After that, the HC updates the HccaFrameNumber and then reads the ED to be processed first in that frame.

The value of HcDoneHead is written to the HccaDoneHead field. After a TD has been processed and the number of frame-periods indicated by the DelayInterrupt field has passed, processing of the next frame starts and the value in HcDoneHead is written to the HccaDoneHead.

2.3.5 List Processing

Figure 2.8 shows a situation where there are four EDs (ED1, ED2, ED3, and ED4), with one or more TDs linked to each ED. The HC neither processes a single ED until all TDs linked to that ED have been retired nor processes a single TD until the TD has been retired. The HC repeatedly generates single data packets for transfer from the first TD of each ED, so that processing is shared among the EDs.

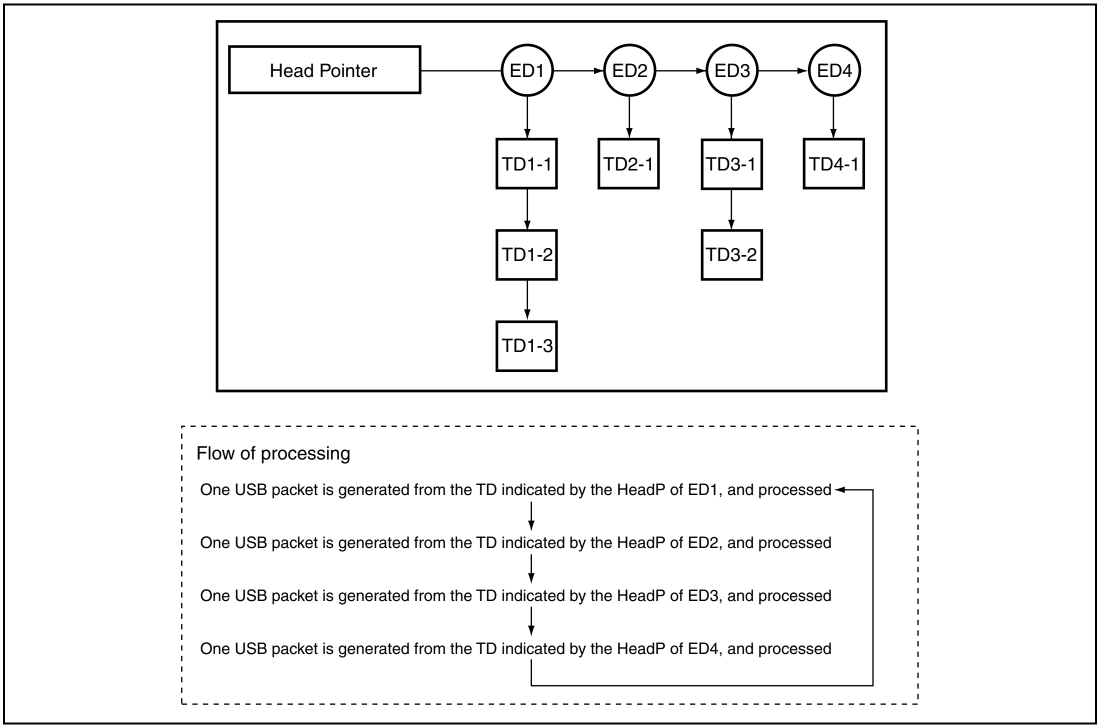

Head Pointer
ED1
ED2
ED3
ED4
TD1-1
TD2-1
TD3-1
TD4-1
TD1-2
TD3-2
TD1-3
Flow of processing
One USB packet is generated from the TD indicated by the HeadP of ED1, and processed
One USB packet is generated from the TD indicated by the HeadP of ED2, and processed
One USB packet is generated from the TD indicated by the HeadP of ED3, and processed
One USB packet is generated from the TD indicated by the HeadP of ED4, and processed

Figure 2.8 List Processing

2.3.6 Done Queue

The HC links TDs for which transfer has been completed in another list. This is called the Done Queue. An example of a Done Queue is shown in figure 2.9.

Consider the situation where the list indicated by List as the one to be processed by the HC (initial condition) and transfer for TD1-1, TD2-1, TD3-1, and TD4-1 in the list is completed during Frame-R processing. Each time the transfer for a TD is completed, that TD is linked to the Done Queue. As is shown in figure 2.11, the latest transfer-completed TD for which processing has most recently been completed is always linked to the head of the Done Queue. The oldest transfer-completed TD is always at the end of the queue.

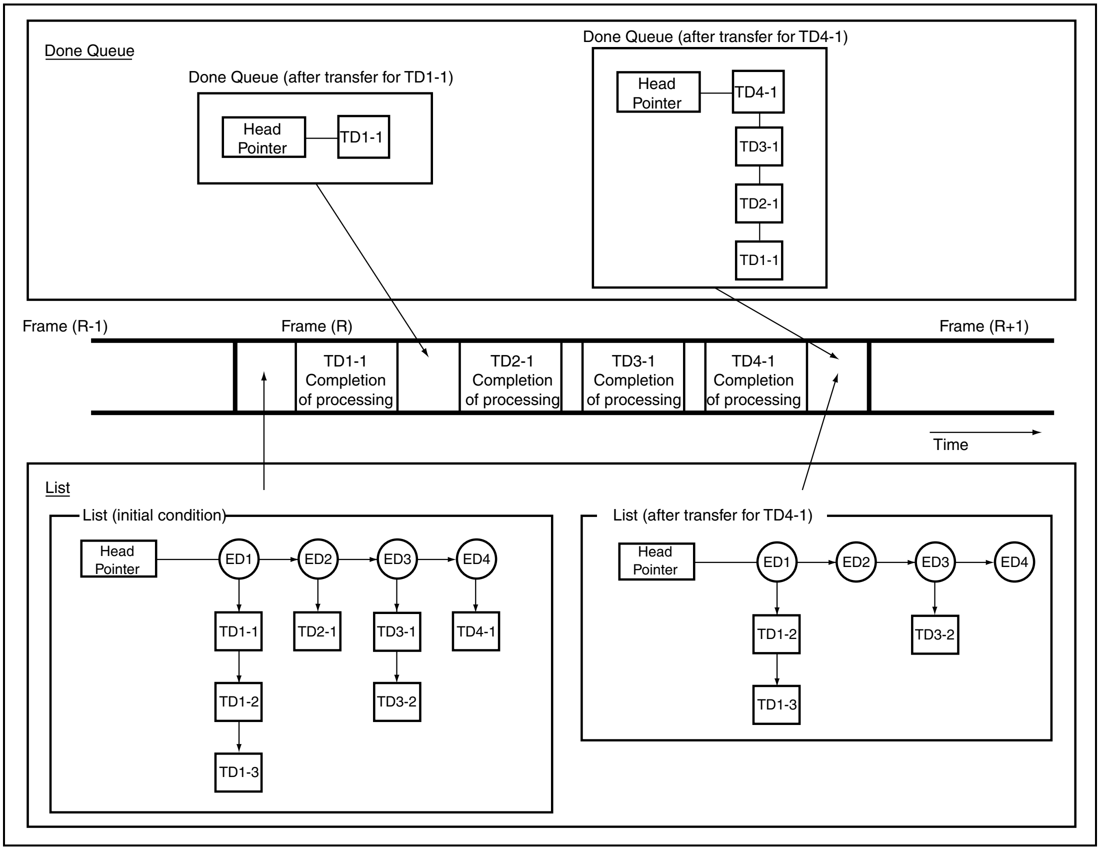

Done Queue
Done Queue (after transfer for TD1-1)
Head Pointer
TD1-1
Done Queue (after transfer for TD4-1)
Head Pointer
TD4-1
TD3-1
TD2-1
TD1-1
Frame (R-1)
Frame (R)
Frame (R+1)
TD1-1 Completion of processing
TD2-1 Completion of processing
TD3-1 Completion of processing
TD4-1 Completion of processing
Time
List
List (initial condition)
Head Pointer
ED1
ED2
ED3
ED4
TD1-1
TD2-1
TD3-1
TD4-1
TD1-2
TD3-2
TD1-3
List (after transfer for TD4-1)
Head Pointer
ED1
ED2
ED3
ED4
TD1-2
TD3-2
TD1-3

Figure 2.9 The Done Queue

Figure 2.10 shows the flow for the linking of TDs to the Done Queue on completion of the corresponding transfers. On completion of the transfer for TD1 in figure 2.10, processing by the HC is as listed below.

1. The value of the NextTD of the transfer-completed TD (TD1) is written to the HeadP field of ED (ED1).
2. The value of the HcDoneHead register is written to the NextTD field of TD1.
3. The first address of the TD1 is written to the HcDoneHead register.

TD1 is thus linked to the Done Queue on completion of the corresponding transfer. On completion of transfer for TD2, processes 1 to 3 are applied to link TD2 to the Done Queue.

The HC links the transfer-completed TD to the Done Queue. When a WriteBackDoneHead interrupt is generated, the value of HcDoneHead is written to the HccaDoneHead field by the HC. The HCD is able to recognize the TDs in the Done Queue by reading the HccaDoneHead field after having detected the WriteBackDoneHead interrupt.

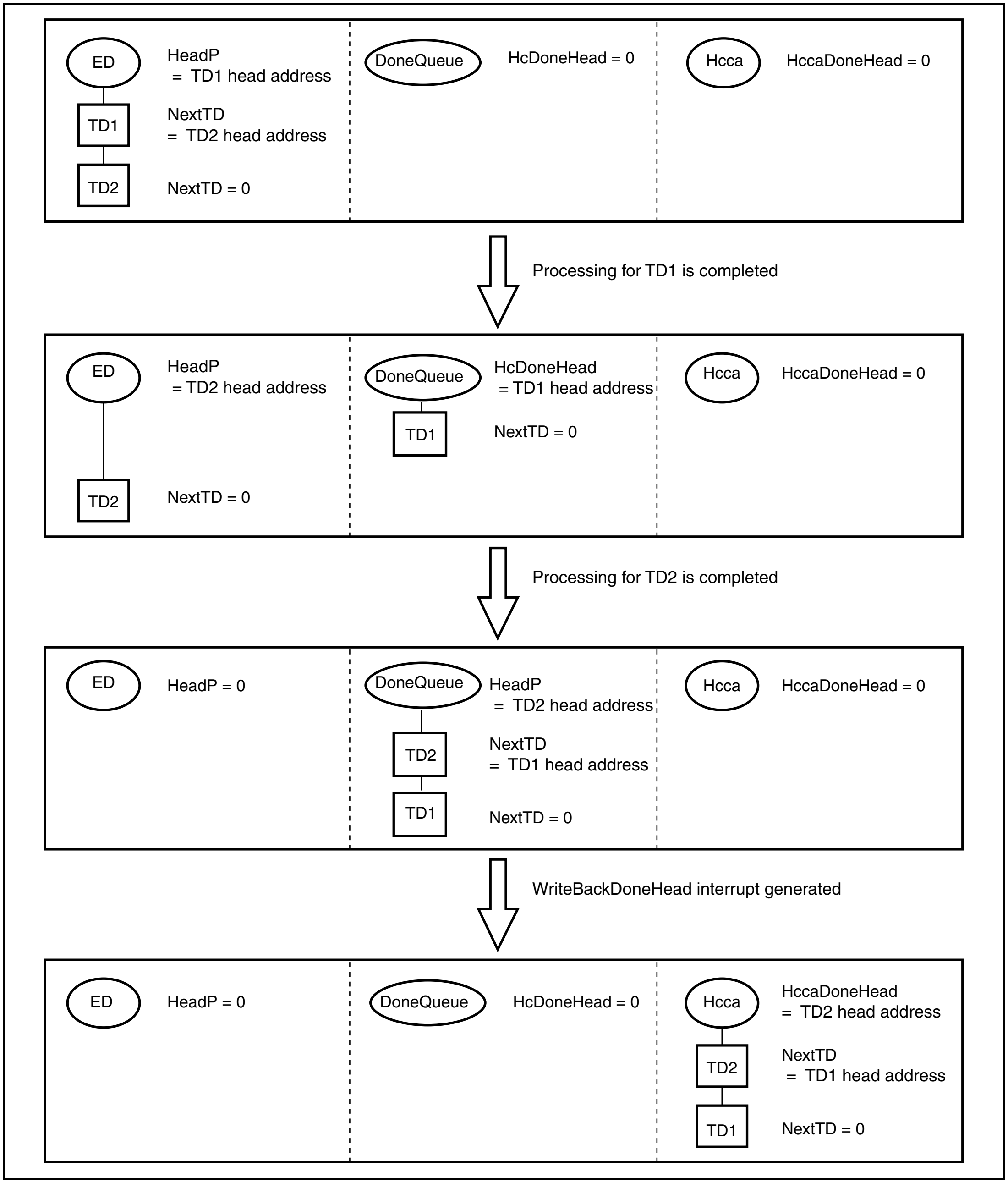


Figure 2.10 Operation of The Done Queue

2.3.7 Communication Channels

There are two channels for communication between the HC and the HCCA. Communications are performed via registers in the HC or via the HCCA in memory.

EDs and TDs are transferred between the HCD and HC. The HCD creates an ED for a transfer and the TDs for the data to be transferred, links them in a list, and then sends the list to the HC. The HC generates USB packets from the received ED and TDs and transfers the packets to the endpoint indicated by the ED. The HC places the transfer-completed TDs in the Done Queue and sends the head pointer to the Done Queue of completed TDs back to the HCD at specified intervals.

Four items of information are transferred between the HCD and HC: the control list and bulk list for nonperiodic transfer, the list for periodic transfer, and the Done Queue. Communication between the HCD and HC is through the transfer of pointers to the respective head addresses. The two head pointers for nonperiodic transfer are transferred via registers, while the head pointers for periodic transfer and the Done Queue are transferred via the HCCA; in other words, the latter two lists are transferred via memory. This is shown in figure 2.11.

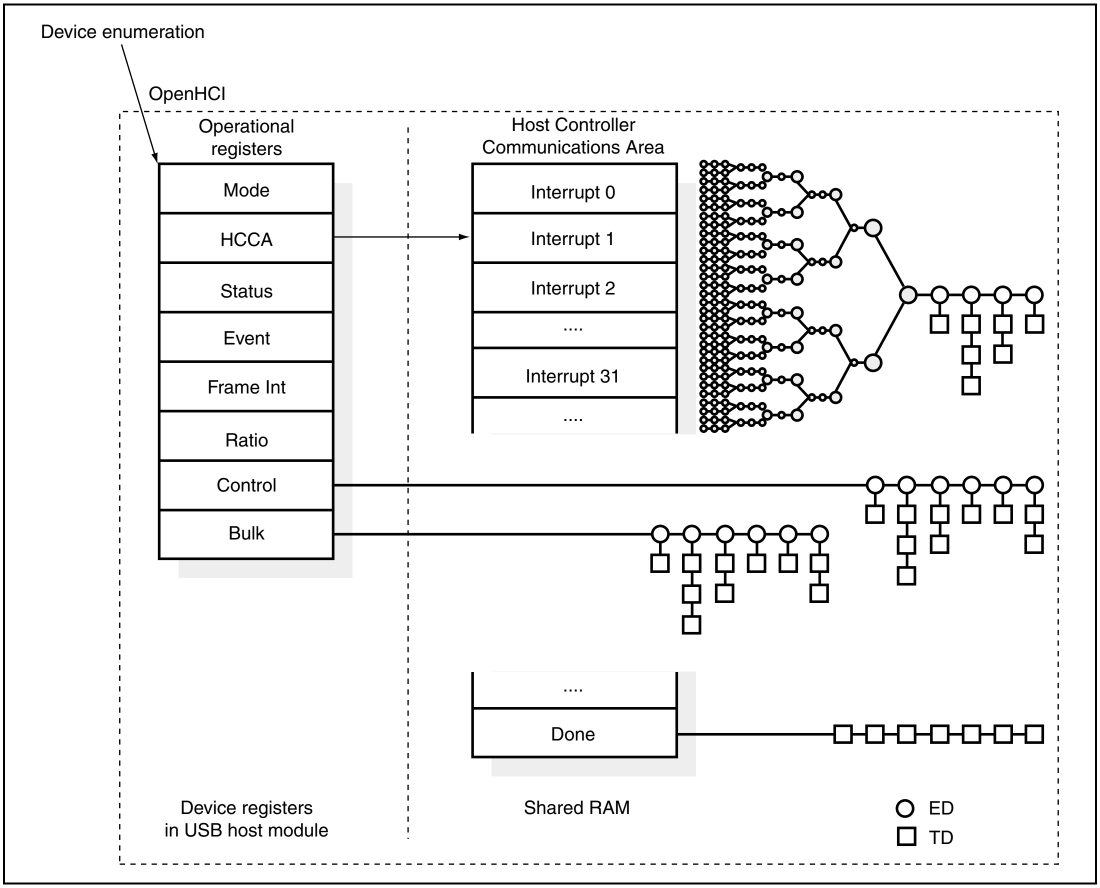


Figure 2.11 Communication Channels

2.4 Responsibilities of Host Controller Drive

2.4.1 Management of Host Controller

The HCD maintains and controls the HC. The HCD controls the HC by directly accessing the registers of the HC and registering the head pointers in the interrupt Endpoint Descriptor list of the HCCA.

The HCD thus maintains the state of the HC, head pointers to each list, enabling and disabling of list processing, and enabling and disabling of interrupts.

2.4.2 Bandwidth Allocation

In the USB, a frame is generated every 1.0 ms. As is shown in figure 2.12, in the OpenHCI, a frame is internally divided into three parts, with periods for the processing of nonperiodic and periodic lists. On completion of SOF-packet generation, nonperiodic transfers are carried out until the value of the Frame Remaining field in HcFmRemaining reaches that of the Frame Interval field in HcFmInterval. Periodic transfer begins when the value in the Frame Remaining field has exceeded the value in the Interval field. After all of the periodic transfers have been completed, nonperiodic transfers are again carried out. The HCD determines whether or not to accept requests for periodic transfer by judging whether or not there is sufficient bandwidth for the transfers.

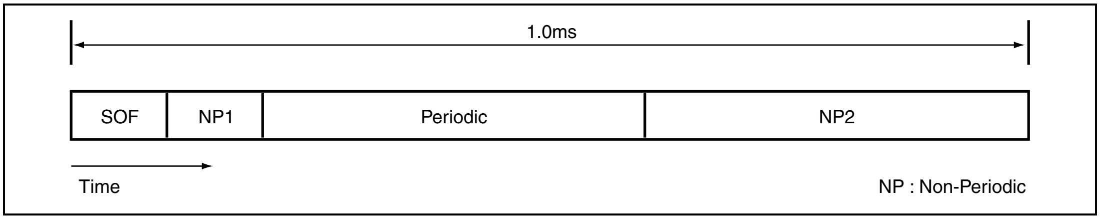


Figure 2.12 Frame Bandwidth Allocation

2.4.3 List Management

The USB data packets are generated from the TDs, which are linked to the EDs. The HCD creates the structures of linked EDs and TDs, i.e., lists.

A new ED can easily be added to the end of a list without halting list processing by the HC. On the other hand, deleting an existing ED requires a halt to list processing by the HC; the ED is then deleted, after which list processing is enabled again. List processing is enabled and disabled by the ControlListEnable, BulkListEnable, PeriodicListEnable, and IsochronousListEnable bits in the HcControl register. List processing stops after all of these bits have been cleared to 0 and the next SOF has been transferred. TDs are added and deleted in the same way as EDs.

2.4.4 Root Hub

The HC includes the registers for the Root Hub. The HCD needs to control both the HC and the Root Hub.

2.5 Responsibilities of Host Controller

2.5.1 USB State

Four USB states are defined in the OpenHCI specification: Operational, Reset, Suspend, and Resume. The HC places the USB bus in the proper state.

Operational State: SOF Tokens can be generated and processing of the respective lists proceeds.

Reset State: This state follows a hardware reset. The USB bus is in the reset state. No SOF Token generation, list processing, or frame-number incrementation is carried out.

Suspend State: No SOF Token generation, list processing, or frame-number incrementation is carried out. The HC's remote wakeup circuit waits to be activated by the wakeup signal.

Resume State: This state is only entered from the 'Suspend' state. Register access by the HCD or acceptance of a remote-wakeup signal makes the system enter this state.

2.5.2 Frame Management

At the beginning of the processing of each frame, an SOF packet is generated and the frame counter in memory is incremented.

2.5.3 List Processing

The HC implements transfer defined by the EDs and TDs generated by the HCD. During the periodic-transfer period of each frame, the HC reads one of the 32 interrupt-list head pointers from the HCCA and processes the interrupt list and isochronous list. A USB packet is generated from the head TD linked to each ED of the lists.

The lists for control and bulk transfer are processed during the nonperiodic-transfer period. These lists are processed in succession. The timing of the changeover is set by the HCD. The HC continues to process the control and bulk lists throughout the nonperiodic-transfer period, transferring n control packets to 1 bulk packet, followed by n control packets, and so on; the value of n is set by the HCD.

A TD for which processing has been completed is moved from its ED to the Done Queue, whether or not the transfer was successful. A completed TD is linked to the Done Queue, with the most recently completed TD at the head of the queue. The HC sets the NextTD field of a transfer-completed TD to the head TD of the Done Queue, thus placing the most recently completed TD in the Done Queue. Information on the Done Queue is periodically sent from the HC to the HCD via the HCCA.

2.6 Register Specifications

The HC contains registers as shown in table 2.6. Every register is accessed as a 32-bit unit by the HCD. The registers of the HC are divided into four partitions, specifically for Control and Status, Memory Pointer, Frame Counter and Root Hub.

Table 2.6 Host Controlled Operational Registers

Offset	31 … 0
0	HcRevision
4	HcControl
8	HcCommandStatus
C	HcInterruptStatus
10	HcInterruptEnable
14	HcInterruptDisable
18	HcHCCA
1C	HcPeriodCurrentED
20	HcControlHeadED
24	HcControlCurrentED
28	HcBulkHeadED
2C	HcBulkCurrentED
30	HcDoneHead
34	HcFmInterval
38	HcFmRemaining
3C	HcFmNumber
40	HcPeriodicStart
44	HcLSThreshold
48	HcRhDescriptorA
4C	HcRhDescriptorB
50	HcRhStatus
54	HcRhPortStatus[1]
...	...
54 + 4*NDP	HcRhPortStatus[NDP]

Legend

NDP: Number Downstream Ports (NDP = 2 for the SH7727 Series)

2.6.1 Control and Status Partition

HcRevision Register:

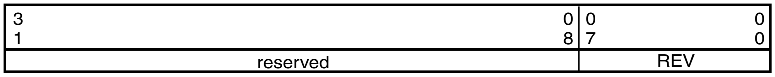


Figure 2.13 HcRevision Register

Table 2.7 HcRevision Register

Key	Reset	Read/Write HCD	Read/Write HC	Description
REV	10h	R	R	Revision This read-only field contains the BCD representation of the version of the HCI specification that is implemented by this HC. For example, a value of H'11 corresponds to version 1.1. All of the HC implementations that are compliant with this specification will have a value of H'10.

HcControl Register: This register defines the operating modes for the HC.

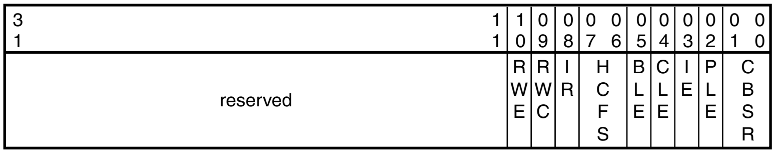


Figure 2.14 HcControl Register

Table 2.8 HcControl Register

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
CBSR	00b	R/W	R	ControlBulkServiceRatio This specifies the ratio between Control ED and Bulk EDs. During processing the nonperiodic lists, HC processes one to four Control EDs according to the ratio specified for this field before processing a Bulk ED. During processing the nonperiodic lists, this process is repeated. After a Control ED has been processed, according to the ratio for this field, HC decides whether to continue serving another Control ED or switching to Bulk EDs.
PLE	0b	R/W	R	PeriodicListEnable This bit is set to enable the processing of the periodic list in the next frame. If cleared by HCD, processing of the periodic list does not occur after the next SOF. HC must check this bit before it starts processing the list.
IE	0b	R/W	R	IsochronousEnable This bit is used by HCD to enable/disable processing of isochronous EDs. While processing the periodic list in a Frame, HC checks the status of this bit when it finds an Isochronous ED (F=1). If set (enabled), HC continues processing the EDs. If cleared (disabled), HC halts processing of the periodic list (which now contains only isochronous EDs) and begins processing the Bulk/Control lists.
CLE	0b	R/W	R	ControlListEnable This bit is set to enable the processing of the Control list in the next Frame. If cleared by HCD, processing of the Control list does not occur after the next SOF. HC must check this bit whenever it determines to process the list. When disabled, HCD may modify the list. If HcControlCurrentED is pointing to an ED to be removed, HCD must advance the pointer by updating *HcControlCurrentED* before re-enabling processing of the list.

CBSR sub-table:

CBSR	**No. of Control EDs Over Bulk EDs Served**
0	1:1
1	2:1
2	3:1
3	4:1

Key	Reset	Read/Write HCD	Read/Write HC	Description
BLE	0b	R/W	R	BulkListEnable This bit is set to enable the processing of the Bulk list in the next Frame. If cleared by HCD, processing of the Bulk list does not occur after the next SOF. HC checks this bit whenever it determines to process the list. If HcBulkCurrentED is pointing to an ED to be removed, HCD must advance the pointer by updating HcBulkCurrentED before re-enabling processing of the list.
HCFS	00b	R/W	R/W	HostControllerFunctionalState Defines the state of the HC. 00b: USBRESET 01b: USBRESUME 10b: USBOPERATIONAL 11b: USBSUSPEND A transition to USBOPERATIONAL from another state causes SOF generation to begin 1 ms later. This field may be changed by HC only when in the USBSUSPEND state. HC may move from the USBSUSPEND state to the USBRESUME state after detecting the resume signaling from a downstream port. HC enters USBSUSPEND after a software reset, whereas it enters USBRESET after a hardware reset. The latter also resets the Root Hub.
IR	0b	R/W	R	InterruptRouting This bit determines the routing of interrupts generated by events registered in HcInterruptStatus. If clear, all interrupts are routed to the normal host bus interrupt mechanism. If set, interrupts are routed to the System Management Interrupt. HCD uses this bit as a tag to indicate the ownership of HC.
RWC	0b	R/W	R/W	RemoteWakeupConnected This bit indicates whether HC supports remote wakeup signaling. If remote wakeup is supported and used by the system it is the responsibility of system firmware to set this bit during POST. HC clears the bit upon a hardware reset but does not alter it upon a software reset.
RWE	0b	R/W	R	RemoteWakeupEnable This bit is used by HCD to enable or disable the remote wakeup feature upon the detection of upstream resume signaling. When this bit is set and the **ResumeDetected** bit in *HcInterruptStatus* is set, a remote wakeup is signaled to the host system.

HcCommandStatus Register: The HCD requests the list processing, HC reset, and honor ship change to the HC by using this register. The number of frames with which the HC has detected the scheduling overrun error is counted by this register.

31 … 18	17 16	15 … 04	03	02	01	00
reserved	SOC	reserved	OCR	BLF	CLF	HCR

Figure 2.15 HcCommandStatus Register

Table 2.9 HcCommandStatus Register

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
HCR	0b	R/W	R/W	HostControllerReset This bit is set by HCD to initiate a software reset of HC. Regardless of the functional state of HC, it moves to the USBSUSPEND state in which most of the operational registers are reset except those stated otherwise; e.g., the InterruptRouting field of *HcControl*, and no Host bus accesses are allowed. This bit is cleared by HC upon the completion of the reset operation. The reset operation must be completed within 10 µs. This bit, when set, should not cause a reset to the Root Hub and no subsequent reset signaling should be asserted to its downstream ports.
CLF	0b	R/W	R/W	ControlListFilled This bit is used to indicate whether there are any TDs on the Control list. It is set by HCD whenever it adds a TD to an ED in the Control list. When HC begins to process the head of the Control list, it checks CLF. As long as ControlListFilled is 0, HC will not start processing the Control list. If CF is 1, HC will start processing the Control list and will set ControlListFilled to 0. If HC finds a TD on the list, then HC will set ControlListFilled to 1 causing the Control list processing to continue. If no TD is found on the Control list, and if the HCD does not set ControlListFilled, then ControlListFilled will still be 0 when HC completes processing the Control list and Control list processing will stop.

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
BLF	0b	R/W	R/W	BulkListFilled This bit is used to indicate whether there are any TDs on the Bulk list. It is set by HCD whenever it adds a TD to an ED in the Bulk list. When HC begins to process the head of the Bulk list, it checks BF. As long as BulkListFilled is 0, HC will not start processing the Bulk list. If BulkListFilled is 1, HC will start processing the Bulk list and will set BF to 0. If HC finds a TD on the list, then HC will set BulkListFilled to 1 causing the Bulk list processing to continue. If no TD is found on the Bulk list, and if HCD does not set BulkListFilled, then BulkListFilled will still be 0 when HC completes processing the Bulk list and Bulk list processing will stop.
OCR	0b	R/W	R/W	OwnershipChangeRequest This bit is set by an OS HCD to request a change of control of the HC. When set HC will set the OwnershipChange field in HcInterruptStatus. After the changeover, this bit is cleared and remains so until the next request from OS HCD.
SOC	00b	R	R/W	SchedulingOverrunCount These bits are incremented on each scheduling overrun error. It is initialized to B'00b and wraps around at B'11. This will be incremented when a scheduling overrun is detected even if SchedulingOverrun in HcInterruptStatus has already been set. This is used by HCD to monitor any persistent scheduling problems.

HcInterruptStatus Register: This register provides status on various events that cause hardware interrupts. When an event occurs, corresponding bit is set. When a bit is set, if the event is enabled in the HcInterruptEnable register and the MasterInterruptEnable bit is set, a hardware interrupt is generated. To clear specific bits in this register, write a 1 to the bit by the HCD. The HCD can clear but not set any of these bits. While, the HC can set but not clear the bit.

31	30	29	07	06	05	04	03	02	01	00
0	OC	reserved		RHSC	FNO	UE	RD	SF	WDH	SO

Figure 2.16 HcInterruptStatus Register

Table 2.10 HcInterruptStatus Register

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
SO	0b	R/W	R/W	SchedulingOverrun This bit is set when the USB schedule for the current Frame overruns and after the update of HccaFrameNumber. A scheduling overrun will also cause the SchedulingOverrunCount of HcCommandStatus to be incremented.
WDH	0b	R/W	R/W	WritebackDoneHead This bit is set immediately after HC has written HcDoneHead to HccaDoneHead. Further updates of the HccaDoneHead will not occur until this bit has been cleared. HCD should only clear this bit after it has saved the content of HccaDoneHead.
SF	0b	R/W	R/W	StartofFrame This bit is set by HC at each start of a frame and after the update of HccaFrameNumber. HC also generates a SOF token at the same time.
RD	0b	R/W	R/W	ResumeDetected This bit is set when HC detects that a device on the USB bus is asserting resume signaling. This bit is not set when HCD sets HC in USBResume state by modifying the HostControllerFunctionalState field in the HcControl register.
UE	0b	R/W	R/W	UnrecoverableError This bit is set when HC detects a system error not related to USB. HC should not proceed with any processing nor signaling before the system error has been corrected. HCD clears this bit after HC has been reset.
FNO	0b	R/W	R/W	FrameNumberOverflow This bit is set when the MSb of HcFmNumber (bit 15) changes value, from 0 to 1 or from 1 to 0, and after HccaFrameNumber has been updated.
RHSC	0b	R/W	R/W	RootHubStatusChange This bit is set when the content of HcRhStatus or the content of any of HcRhPortStatus[NumberofDownstreamPort] has changed.
OC	0b	R/W	R/W	OwnershipChange This bit is set by HC when HCD sets the OwnershipChangeRequest field in HcCommandStatus. This event, when unmasked, will always generate a System Management Interrupt (SMI) immediately. This bit is tied to 0b when the SMI pin is not implemented.

HcInterruptEnable Register: Each bit in the HcInterruptEnable register corresponds to each bit in the HcInterruptStatus register. The HcInterruptEnable register is used to control which events generate a hardware interrupt. When a bit is set in the HcInterruptStatus register and the corresponding bit in the HcInterruptEnable register is set and the MasterInterruptEnable bit is set, a hardware interrupt is generated.

Writing a 1 to a bit in this register sets the corresponding bit. Writing a 0 to a bit in this register leaves the corresponding bit unchanged. To clear the corresponding bit, write 1 to a bit in the HcInterruptDisable register.

31	30	29	07	06	05	04	03	02	01	00
MIE	OC	reserved		RHSC	FNO	UE	RD	SF	WDH	SO

Figure 2.17 HcInterruptEnable Register

Table 2.11 HcInterruptEnable Register

Key	Reset	Read/Write HCD	Read/Write HC	Description
SO	0b	R/W	R	0: Ignore 1: Enable interrupt generation due to Scheduling Overrun.
WDH	0b	R/W	R	0: Ignore 1: Enable interrupt generation due to HcDoneHead Writeback.
SF	0b	R/W	R	0: Ignore 1: Enable interrupt generation due to Start of Frame.
RD	0b	R/W	R	0: Ignore 1: Enable interrupt generation due to Resume Detect.
UE	0b	R/W	R	0: Ignore 1: Enable interrupt generation due to Unrecoverable Error.
FNO	0b	R/W	R	0: Ignore 1: Enable interrupt generation due to Frame Number Overflow.
RHSC	0b	R/W	R	0: Ignore 1: Enable interrupt generation due to Root Hub Status Change.
OC	0b	R/W	R	0: Ignore 1: Enables interrupt generation due to Ownership Change.
MIE	0b	R/W	R	0: Ignored 1: Enables interrupt generation due to other bits of this register. This is used by HCD as a Master Interrupt Enable.

HcInterruptDisable Register: Each bit in the HcInterruptDisable register corresponds to each bit in the HcInterruptStatus register. The HcInterruptDisable register is coupled with the HcInterruptEnable register. Writing a 1 to a bit in the HcInterruptDisable register clears the corresponding bit in the HcInterruptEnable register. Writing a 0 to a bit in the HcInterruptDisable register leaves the corresponding bit in the HcInterruptEnable register unchanged. On read, the current value of the HcInterruptEnable register is returned.

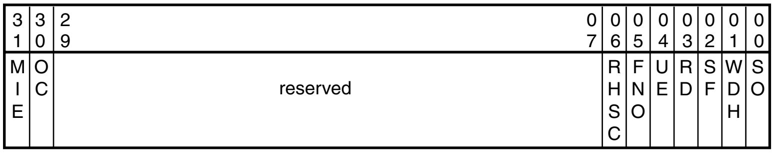


Figure 2.18 HcInterruptDisable Register

Table 2.12 HcInterruptDisable Register

Key	Reset	Read/Write		Description
		HCD	HC	
SO	0b	R/W	R	0: Ignore 1: Disable interrupt generation due to Scheduling Overrun.
WDH	0b	R/W	R	0: Ignore 1: Disable interrupt generation due to HcDoneHead Writeback.
SF	0b	R/W	R	0: Ignore 1: Disable interrupt generation due to Start of Frame.
RD	0b	R/W	R	0: Ignore 1: Disable interrupt generation due to Resume Detect.
UE	0b	R/W	R	0: Ignore 1: Disable interrupt generation due to Unrecoverable Error.
FNO	0b	R/W	R	0: Ignore 1: Disable interrupt generation due to Frame Number Overflow.
RHSC	0b	R/W	R	0: Ignore 1: Disable interrupt generation due to Root Hub Status Change.
OC	0b	R/W	R	0: Ignore 1: Disable interrupt generation due to Ownership Change.
MIE	0b	R/W	R	0: Ignore 1: Disables interrupt generation due to other bits of this register. This field is set after a hardware or software reset.

2.6.2 Memory Pointer Partition

HcHCCA Register: The HcHCCA register contains the physical address of the Host Controller Communications Area (HCCA). The HCCA should be aligned to a 256-byte boundary. Therefore, the lower 8-bit of this register is always read as 0.

31	08	07	00
HCCA		0	

Figure 2.19 HcHCCA Register

Table 2.13 HcHCCA Register

Key	Reset	Read/Write HCD	Read/Write HC	Description
HCCA	0h	R/W	R	Indicates the base address of the Host Controller Communications Area.

HcPeriodCurrentED Register: The HcPeriodCurrentED register indicates the physical address of the current Isochronous ED or Interrupt ED.

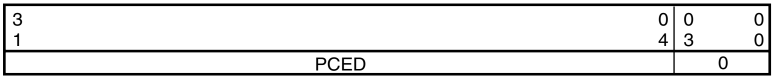


Figure 2.20 HcPeriodCurrentED Register

Table 2.14 HcPeriodCurrentED Register

Key	Reset	Read/Write HCD	Read/Write HC	Description
PCED	0h	R	R/W	PeriodCurrentED This is used by HC to point to the head of one of the Periodic lists which will be processed in the current Frame. The content of this register is updated by HC after a periodic ED has been processed. HCD may read the content in determining which ED is currently being processed at the time of reading.

HcControlHeadED Register: The HcControlHeadED register indicates the physical address of the first ED of the Control list.

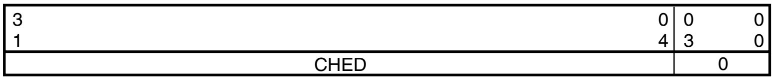


Figure 2.21 HcControlHeadED Register

Table 2.15 HcControlHeadED Register

Key	Reset	Read/Write HCD	Read/Write HC	Description
CHED	0h	R/W	R	ControlHeadED HC traverses the Control list starting with the HcControlHeadED pointer.

HcControlCurrentED Register: The HcControlCurrentED register indicates the physical address of the current Control ED.

31	04	03	00
CCED		0	

Figure 2.22 HcControlCurrentED Register

Table 2.16 HcControlCurrentED Register

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
CCED	0h	R/W	R/W	ControlCurrentED This field indicates a pointer of the current ED. This pointer is advanced to the next ED after serving the present one. When it reaches the end of the Control list, HC checks the ControlListFilled of in HcCommandStatus. If set, it copies the content of HcControlHeadED to HcControlCurrentED and clears the bit. If not set, it does nothing. HCD is allowed to modify this register only when the ControlListEnable of HcControl is cleared. When set, HCD only reads the instantaneous value of this register.

HcBulkHeadED Register: The HcBulkHeadED register indicates the physical address of the first ED of the Bulk list.

31	04	03	00
BHED		0	

Figure 2.23 HcBulkHeadED Register

Table 2.17 HcBulkHeadED Register

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
BHED	0h	R/W	R	BulkHeadED HC traverses the Bulk list starting with the HcBulkHeadED pointer.

HcBulkCurrentED Register: The HcBulkCurrentED register indicates the physical address of the current Bulk ED.

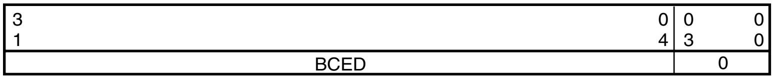


Figure 2.24 HcBulkCurrentED Register

Table 2.18 HcBulkCurrentED Register

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
BCED	0h	R/W	R/W	BulkCurrentED This field indicates a pointer of the current ED. This is advanced to the next ED after the HC has served the present one. When it reaches the end of the Bulk list, HC checks the BulkListFilled of HcControl. If set, it copies the content of *HcBulkHeadED* to *HcBulkCurrentED* and clears the bit. If it is not set, it does nothing. HCD is only allowed to modify this register when the **BulkListEnable** of *HcControl* is cleared. When set, the HCD only reads the instantaneous value of this register.

HcDoneHead Register: The HcDoneHead register indicates the physical address of the last completed TD that was added to the Done queue. As the value of this register is gained from the HCCA, usually this register is not accessed by HCD.

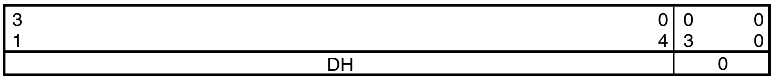


Figure 2.25 HcDoneHead Register

Table 2.19 HcDoneHead Register

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
DH	0h	R	R/W	DoneHead When a TD is completed, HC writes the content of HcDoneHead to the NextTD field of the TD. HC then overwrites the content of HcDoneHead with the address of this TD. This is set to zero whenever HC writes the content of this register to HCCA. It also sets the WritebackDoneHead of HcInterruptStatus.

2.6.3 Frame Counter Partition

HcFmInterval Register: The HcFmInterval register contains a 14-bit value which indicates the bit time interval in a frame (between two consecutive SOFs), and a 15-bit value indicating the maximum packet size that the HC can transfer without causing scheduling overrun. The HCD can carry out minor adjustment on the frame interval. This register provides the programmability necessary for the HC to synchronize with an external clock source and to adjust any unknown local clock.

31	30 … 16	15 14	13 … 0
FIT	FSMPS	reserved	FI

Figure 2.26 HcFmInterval Register

Table 2.20 HcFmInterval Register

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
FI	2EDFh	R/W	R	FrameInterval This specifies the interval between two consecutive SOFs in bit times. The nominal value is set to be 11,999. HCD should store the current value of this field before resetting HC. By setting the HostControllerReset field of HcCommandStatus as this will cause the HC to reset this field to its nominal value.
FSMPS	TBD	R/W	R	FSLargestDataPacket This field specifies a value which is loaded into the Largest Data Packet Counter at the beginning of each frame. The counter value represents the largest amount of data in bits which can be sent or received by the HC in a single transaction at any given time without causing scheduling overrun. The field value is calculated by the HCD.
FIT	0b	R/W	R	FrameIntervalToggle HCD toggles this bit whenever it loads a new value to FrameInterval.

HcFmRemaining Register: The HcFmRemaining register is a 14-bit down counter which shows the bit time remaining in the current frame.

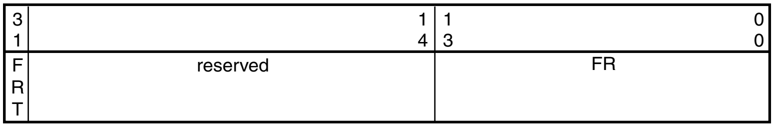


Figure2.27 HcFmRemaining Register

Table2.21 HcFmRemaining Register

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
FR	0h	R	R/W	FrameRemaining Indicates bit time. When it reaches zero, it is reset by loading the FrameInterval value specified in HcFmInterval at the next bit time boundary. When entering the USBOPERATIONAL state, HC re-loads the content with the FrameInterval of *HcFmInterval* and uses the updated value from the next SOF.
FRT	0b	R	R/W	FrameRemainingToggle This bit is loaded from the FrameIntervalToggle field of HcFmInterval whenever FrameRemaining reaches 0. This bit is used by HCD for the synchronization between FrameInterval and FrameRemaining.

HcFmNumber Register: The HcFmNumber register is a 16-bit counter which indicates the current frame number. The HCD can generate a 32-bit frame number by using the FrameNumberOverrun interrupt.

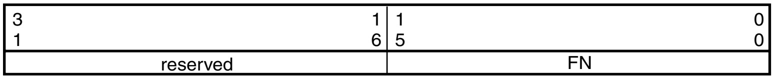


Figure2.28 HcFmNumber Register

Table2.22 HcFmNumber Register

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
FN	0h	R	R/W	FrameNumber This is incremented when HcFmRemaining is re-loaded. It will be rolled over to H'0 after H'FFFF. When entering the USBOPERATIONAL state, this will be incremented automatically. The content will be written to HCCA after HC has incremented the FrameNumber at each frame boundary and sent a SOF but before HC reads the first ED in that Frame. After writing to HCCA, HC will set the StartofFrame in HcInterruptStatus.

HcPeriodicStart Register: The HcPeriodicStart register determines when is the earliest time HC should start processing the periodic list.

31 14	13 0
reserved	PS

Figure 2.29 HcPeriodicStart Register

Table 2.23 HcPeriodicStart Register

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
PS	0h	R/W	R	PeriodicStart After a hardware reset, this field is cleared. The value is calculated roughly as 10% off from HcFmInterval. A typical value will be H'3E67. When HcFmRemaining reaches the value specified, HC will start processing the Periodic list after having completed the current Control or Bulk transaction.

HcLSThreshold Register: The HcLSThreshold register referred when the HC to decide whether to transfer a maximum of 8-byte LS packet before EOF.

31 12	11 0
reserved	LST

Figure 2.30 HcRhDescriptorA Regis

Table 2.24 HcLSThreshold Regis

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
LST	0628h	R/W	R	LSThreshold This field contains a value which is compared to the FrameRemaining field prior to initiating a Low Speed transaction. The transaction is started only if FrameRemaining ≥ this field. The value is calculated by HCD with the consideration of transmission and setup overhead.

2.6.4 Root Hub Partition

HcRhDescriptorA Register: The HcRhDescriptorA register is the first register of two describing the characteristics of the Root Hub.

31 24	23 13	12	11	10	09	08	07 00
POTPGT	reserved	NOCP	OCPM	DT	NPS	PSM	NDP

Figure 2.31 HcRhDescriptorA Regis

Table 2.25 HcRhDescriptorA Regis

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
NDP	IS	R	R	NumberDownstreamPorts These bits specify the number of downstream ports supported by the Root Hub. The minimum number of ports is 1. The maximum number of ports supported by OpenHCI is 15.
NPS	IS	R/W	R	NoPowerSwitching These bits are used to specify whether power switching is supported or port are always powered. When this bit is cleared, in other words, power switching is enabled, the PowerSwitchingMode specifies global or per-port switching. 0: Ports are power switched 1: Ports are always powered on when the HC is powered on
PSM	IS	R/W	R	PowerSwitchingMode This bit is used to specify how the power switching of the Root Hub ports is controlled. This field is only valid if the NoPowerSwitching field is cleared. 0: All ports are powered at the same time. 1: Each port is powered individually. This mode allows port power to be controlled by either the global switch or per-port switching. If the PortPowerControlMask bit is set, the port responds only to port power commands (Set/ClearPortPower). If the port mask is cleared, then the port is controlled only by the global power switch (Set/ClearGlobalPower).

		Read/Write		
Key	Reset	HCD	HC	Description
DT	0b	R	R	DeviceType This bit specifies that the Root Hub is not a compound device. The Root Hub is not permitted to be a compound device. This field should always read/write 0.
OCPM	IS	R/W	R	OverCurrentProtectionMode This bit describes how the overcurrent status for the Root Hub ports are reported. At reset, this fields should reflect the same mode as PowerSwitchingMode. This field is valid only if the NoOverCurrentProtection field is cleared. 0: Over-current status is reported collectively for all downstream ports 1: Over-current status is reported on a per-port basis
NOCP	IS	R/W	R	NoOverCurrentProtection This bit describes how the overcurrent status for the Root Hub ports are reported. When this bit is cleared, the OverCurrentProtectionMode field specifies global or per-port reporting. 0: Over-current status is reported collectively for all downstream ports 1: No overcurrent protection supported
POTPGT	IS	R/W	R	PowerOnToPowerGoodTime This byte specifies the duration HCD has to wait before accessing a powered-on port of the Root Hub. The unit of time is 2 ms. The duration is calculated as POTPGT × 2 ms.

HcRhDescriptorB Register: The HcRhDescriptorB register is the second register of two describing the characteristics of the Root Hub.

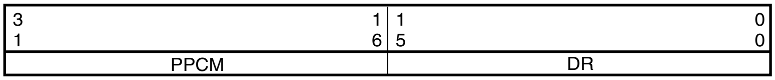


Figure 2.32 HcRhDescriptor Register

Table 2.26 HcRhDescriptor Register

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
DR	IS	R/W	R	DeviceRemovable Each bit is dedicated to a port of the Root Hub. When cleared, the attached device is removable. When set, the attached device is not removable. bit 0: Reserved bit 1: Device attached to Port #1 bit 2: Device attached to Port #2 ... bit15: Device attached to Port #15
PPCM	IS	R/W	R	PortPowerControlMask Each bit indicates if a port is affected by a global power control command when PowerSwitchingMode is set. When set, the port's power state is only affected by per-port power control (Set/ClearPortPower). When cleared, the port is controlled by the global power switch (Set/ClearGlobalPower). If the device is configured to global switching mode (PowerSwitchingMode=0), this field is not valid. bit 0: Reserved bit 1: Ganged-power mask on Port #1 bit 2: Ganged-power mask on Port #2 ... bit15: Ganged-power mask on Port #15

HcRhStatus Register: The HcRhStatus register specifies the status of the Root Hub.

31	18	17	16	15	14 02	01	00
CRWE	reserved	OCIC	LPSC	DRWE	reserved	OCI	LPS

Figure 2.33 HcRhStatus Register

Table 2.27 HcRhStatus Register

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
LPS	0b	R/W	R	(read) LocalPowerStatus The Root Hub does not support the local power status feature; thus, this bit is always read as '0'. (write) ClearGlobalPower In global power mode (PowerSwitchingMode=0), this bit is written to '1' to turn off power to all ports (clear PortPowerStatus). In per-port power mode, it clears PortPowerStatus only on ports whose PortPowerControlMask bit is not set. Writing a '0' has no effect.
OCI	0b	R	R/W	OverCurrentIndicator This bit reports overcurrent conditions when the global reporting is implemented. When set, an overcurrent condition exists. When cleared, all power operations are normal. If per-port overcurrent protection is implemented this bit is always '0'
DRWE	0b	R/W	R	(read) DeviceRemoteWakeupEnable This bit enables or disables remote wakeup. This bit enables a ConnectStatusChange bit as a resume event, causing a USBSUSPEND to USBRESUME state transition and setting the ResumeDetected interrupt. 0: ConnectStatusChange is not a remote wakeup event. 1: ConnectStatusChange is a remote wakeup event. (write) SetRemoteWakeupEnable Writing a '1' sets DeviceRemoveWakeupEnable. Writing a '0' has no effect.
LPSC	0b	R/W	R	(read) LocalPowerStatusChange The Root Hub does not support the local power status feature; thus, this bit is always read as '0'. (write) SetGlobalPower In global power mode (PowerSwitchingMode=0), this bit is written to '1' to turn on power to all ports (clear PortPowerStatus). In per-port power mode, it sets PortPowerStatus only on ports whose PortPowerControlMask bit is not set. Writing a '0' has no effect.

		Read/Write		
Key	Reset	HCD	HC	Description
CCIC	0b	R/W	R/W	OverCurrentIndicatorChange This bit is set by the HC when a change has occurred to the OCI field of this register. The HCD clears this bit by writing a '1'. Writing a '0' has no effect.
CRWE	—	W	R	(write) ClearRemoteWakeupEnable Writing a '1' clears DeviceRemoveWakeupEnable. Writing a '0' has no effect.

HcRhPortStatus 1 and NDP Registers: The HcRhPortStatus register is used to control and report port events on a per-port basis. The lower word is used to set the port status. Whereas, the upper word monitors the status change of ports.

31 ... 21	20	19	18	17	16	15 ... 10	09	08	07 ... 05	04	03	02	01	00
reserved	PRSC	OCIC	PSSC	PESC	CSC	reserved	LSDA	PPS	rsvd	PRS	POCI	PSS	PES	CCS

Figure 2.34 HcRhPortStatus Registe

Table 2.28 HcRhPortStatus Registe

		Read/Write		
Key	Reset	HCD	HC	Description
CCS	0b	R/W	R/W	(read) CurrentConnectStatus This bit reflects the current state of the downstream port. 0: No device connected 1: Device connected (write) ClearPortEnable The HCD writes a '1' to this bit to clear the PortEnableStatus bit. Writing a '0' has no effect. The CurrentConnectStatus is not affected by any write. Note: This bit is always read '1b' when the attached device is nonremovable (DeviceRemoveable[NDP]).

		Read/Write		
Key	**Reset**	**HCD**	**HC**	**Description**
PES	0b	R/W	R/W	(read) **PortEnableStatus** This bit indicates whether the port is enabled or disabled. The Root Hub may clear this bit when an overcurrent condition, disconnect event, switched-off power, or operational bus error such as babble is detected. This change also causes PortEnabledStatusChange to be set. HCD sets this bit by writing SetPortEnable and clears it by writing ClearPortEnable. This bit cannot be set when CurrentConnectStatus is cleared. This bit is also set, if not already, at the completion of a port reset when ResetStatusChange is set or port suspend when SuspendStatusChange is set. 0: Port is disabled 1: Port is enabled (write) SetPortEnable The HCD sets PortEnableStatus by writing a '1'. Writing a '0' has no effect. If CurrentConnectStatus is cleared, this write does not set PortEnableStatus, but instead sets ConnectStatusChange. This informs the driver that it attempted to enable a disconnected port.
PSS	0b	R/W	R/W	(read) PortSuspendStatus This bit indicates the port is suspended or in the resume sequence. It is set by a SetSuspendState write and cleared when PortSuspendStatusChange is set at the end of the resume interval. This bit cannot be set if CurrentConnectStatus is cleared. This bit is also cleared when PortResetStatusChange is set at the end of the port reset or when the HC is placed in the USBRESUME state. If an upstream resume is in progress, it should propagate to the HC. 0: Port is not suspended 1: Port is suspended (write) SetPortSuspend The HCD sets the PortSuspendStatus bit by writing a '1' to this bit. Writing a '0' has no effect. If CurrentConnectStatus is cleared, this write does not set PortSuspendStatus; instead it sets ConnectStatusChange. This informs the driver that it attempted to suspend a disconnected port.

Key	Reset	Read/Write HCD	Read/Write HC	Description
POCI	0b	R/W	R/W	(read) PortOverCurrentIndicator This bit is only valid when the Root Hub is configured in such a way that overcurrent conditions are reported on a per-port basis. If per-port overcurrent reporting is not supported, this bit is set to 0. If cleared, all power operations are normal for this port. If set, an overcurrent condition exists on this port. This bit always reflects the overcurrent input signal 0: No overcurrent condition. 1: Overcurrent condition detected. (write) ClearSuspendStatus The HCD writes a '1' to initiate a resume. Writing a '0' has no effect. A resume is initiated only if PortSuspendStatus is set.
PRS	0b	R/W	R/W	(read) PortResetStatus When this bit is set by a write to SetPortReset, port reset signaling is asserted. When reset is completed, this bit is cleared when PortResetStatusChange is set. This bit cannot be set if CurrentConnectStatus is cleared. 0: Port reset signal is not active 1: Port reset signal is active (write) SetPortReset The HCD sets the port reset signaling by writing a '1' to this bit. Writing a '0' has no effect. If CurrentConnectStatus is cleared, this write does not set PortResetStatus, but instead sets ConnectStatusChange. This informs the driver that it attempted to reset a disconnected port.

		Read/Write		
Key	Reset	HCD	HC	Description
PPS	0b	R/W	R/W	(read) PortPowerStatus This bit reflects the port's power status, regardless of the type of power switching implemented. This bit is cleared if an overcurrent condition is detected. HCD sets this bit by writing SetPortPower or SetGlobalPower. HCD clears this bit by writing ClearPortPower or ClearGlobalPower. Which power control switches are enabled is determined by PowerSwitchingMode and PortPortControlMask[NDP]. In global switching mode (PowerSwitchingMode=0), only Set/ClearGlobalPower controls this bit. In per-port power switching (PowerSwitchingMode=1), if the PortPowerControlMask[NDP] bit for the port is set, only Set/ClearPortPower commands are enabled. If the mask is not set, only Set/ClearGlobalPower commands are enabled. When port power is disabled, CurrentConnectStatus, PortEnableStatus, PortSuspendStatus, and PortResetStatus should be reset. 0: port power is off 1: port power is on (write) SetPortPower The HCD writes a '1' to set the PortPowerStatus bit. Writing a '0' has no effect. Note: This bit is always reads '1b' if power switching is not supported.
LSDA	Xb	R/W	R/W	(read) LowSpeedDeviceAttached This bit indicates the speed of the device attached to this port. When set, a Low Speed device is attached to this port. When clear, a Full Speed device is attached to this port. This field is valid only when the CurrentConnectStatus is set. 0: Full speed device attached 1: Low speed device attached (write) ClearPortPower The HCD clears the PortPowerStatus bit by writing a '1' to this bit. Writing a '0' has no effect.

Key	Reset	Read/Write HCD	Read/Write HC	Description
CSC	0b	R/W	R/W	ConnectStatusChange This bit is set whenever a connect or disconnect event occurs. The HCD writes a ‘1’ to clear this bit. Writing a ‘0’ has no effect. If CurrentConnectStatus is cleared when a SetPortReset, SetPortEnable, or SetPortSuspend write occurs, this bit is set to force the driver to re-evaluate the connection status since these writes should not occur if the port is disconnected. 0: No change in CurrentConnectStatus 1: Change in CurrentConnectStatus Note: If the DeviceRemovable[NDP] bit is set, this bit is set only after a Root Hub reset to inform the system that the device is attached.
PESC	0b	R/W	R/W	PortEnableStatusChange This bit is set when hardware events cause the PortEnableStatus bit to be cleared. Changes from HCD writes do not set this bit. The HCD writes a ‘1’ to clear this bit. Writing a ‘0’ has no effect. 0: No change in PortEnableStatus 1: Change in PortEnableStatus
PSSC	0b	R/W	R/W	PortSuspendStatusChange This bit is set when the full resume sequence has been completed. This sequence includes the 20-msec resume pulse, LS EOP, and 3-mssec resychronization delay. The HCD writes a ‘1’ to clear this bit. Writing a ‘0’ has no effect. This bit is also cleared when ResetStatusChange is set. 0: Resume is not completed 1: Resume completed
OCIC	0b	R/W	R/W	PortOverCurrentIndicatorChange This bit is valid only if overcurrent conditions are reported on a per-port basis. This bit is set when Root Hub changes the PortOverCurrentIndicator bit. The HCD writes a ‘1’ to clear this bit. Writing a ‘0’ has no effect. 0: no change in PortOverCurrentIndicator 1: PortOverCurrentIndicator has changed

		Read/Write		
Key	Reset	HCD	HC	Description
PRSC	0b	R/W	R/W	PortResetStatusChange This bit is set at the end of the 10-ms port reset signal. The HCD writes a ‘1’ to clear this bit. Writing a ‘0’ has no effect. 0: port reset is not complete 1: port reset is complete

Section 3 Development Environment

This section describes the development environment used to develop this system. The devices (tools) listed below were used when developing the system.

- SH7727 Solution Engine (type number: MS7727SE01, hereafter referred to as SH7727SE) manufactured by Hitachi ULSI Systems Co., Ltd.
- SH7727 E10A Emulator manufactured by Renesas Technology Corp.
- PC (Windows® 95/98) equipped with a PCMCIA slot
- Control PC (Windows® 98/Windows® 2000)
- Serial cable (cross cable)
- USB cable
- High-Performance Debugging Interface (hereafter called HDI) manufactured by Renesas Technology Corp.
- High-Performance Embedded Workshop (hereafter called HEW) manufactured by Renesas Technology Corp.
- USB function device (any device)

3.1 Hardware Environment

Figure 3.1 shows the device connections.

1. SH7727SE

   The DIP switch settings on the SH7727SE board must be changed from those at shipment. Before turning on the power, ensure that the DIP switches are set as shown in table 3.1. There is no need to change any other DIP switches.

Table 3.1 DIP Switch Settings

Switch	At Time of Shipment	After Change	DIP Switch Function
Baseboard SW1-6	OFF	ON	Select endian mode
Baseboard SW1-8	OFF	ON	Select E10A emulator mode

2. Control PC

   A PC with Windows® 98/Windows® 2000 installed, and with a serial interface, is used as a PC for USB packet generation tool, RequestGenerator.

3. PC equipped with a PCMCIA slot (E10A PC)

   The E10A should be inserted into a PC card slot and connected to the SH7727SE via an interface cable. After connection, start the HDI and perform emulation.

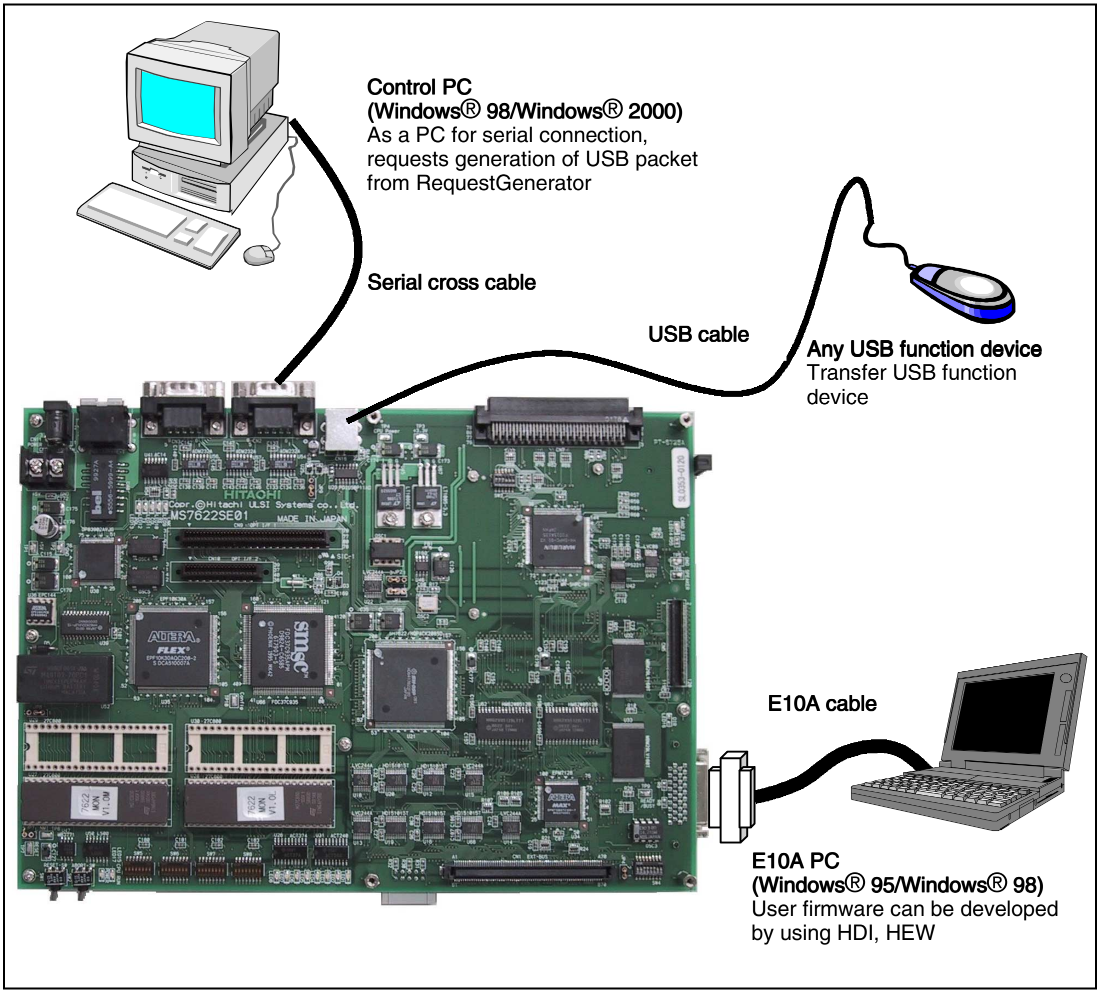

Control PC
(Windows® 98/Windows® 2000)
As a PC for serial connection, requests generation of USB packet from RequestGenerator
Serial cross cable
USB cable
Any USB function device
Transfer USB function device
E10A cable
E10A PC
(Windows® 95/Windows® 98)
User firmware can be developed by using HDI, HEW

Figure 3.1 Device Connections

3.2 Software Environment

A sample program, as well as the compiler and linker used, are explained.

3.2.1 Sample Program

Files required for the sample program are all stored in the SH7727 folder. When this entire folder with its contents is moved to a PC on which HEW and HDI have been installed, the sample program can be used immediately. Files included in the folder are shown in figure 3.2.

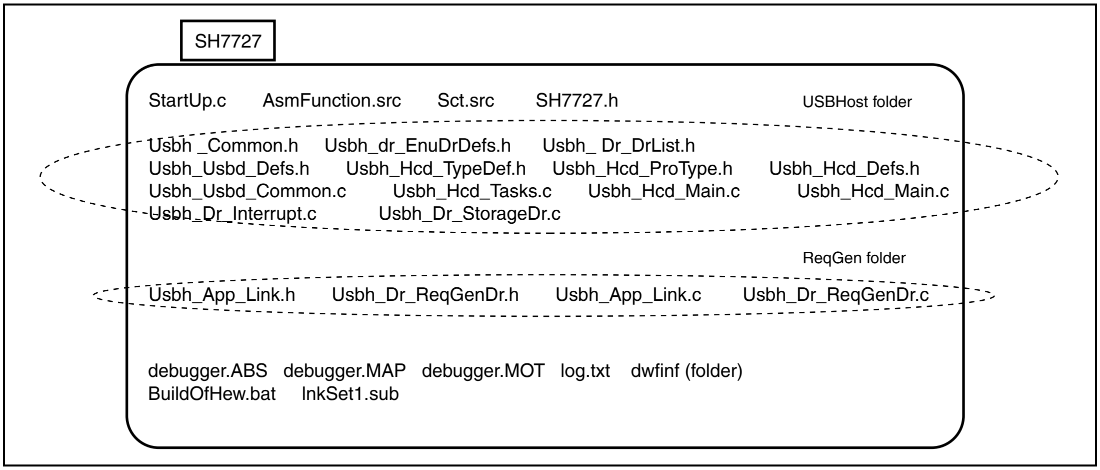

SH7727
StartUp.c AsmFunction.src Sct.src SH7727.h
USBHost folder
Usbh _Common.h Usbh_dr_EnuDrDefs.h Usbh_ Dr_DrList.h
Usbh_Usbd_Defs.h Usbh_Hcd_TypeDef.h Usbh_Hcd_ProType.h Usbh_Hcd_Defs.h
Usbh_Usbd_Common.c Usbh_Hcd_Tasks.c Usbh_Hcd_Main.c Usbh_Hcd_Main.c
Usbh_Dr_Interrupt.c Usbh_Dr_StorageDr.c
ReqGen folder
Usbh_App_Link.h Usbh_Dr_ReqGenDr.h Usbh_App_Link.c Usbh_Dr_ReqGenDr.c
debugger.ABS debugger.MAP debugger.MOT log.txt dwfinf (folder)
BuildOfHew.bat lnkSet1.sub

Figure 3.2 Files Included in SH7727 Folder

3.2.2 Compiling and Linking

The sample program is compiled and linked using the following software: High-Performance Embedded Workshop Version 1.0 (release 9) (hereafter called HEW).

When HEW is installed in C:\Hew*, the procedure for compiling and linking the program is as follows.

Firstly, a folder named Tmp should be created below the C:\Hew folder for use in compiling (figure 3.3).

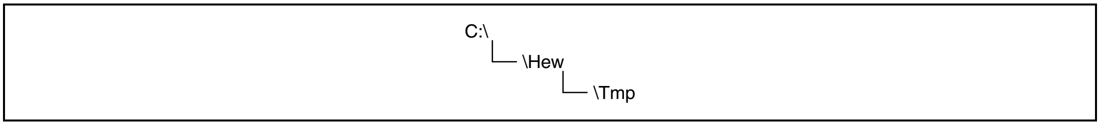


Figure 3.3 Creating a Working Folder

Next, the folder in which the sample program is stored (SH7727) should be copied to any drive. In addition to the sample program, this folder contains a batch file named BuildOfHew.bat. This batch file sets the path, specifies compile options, specifies a log file indicating the compile and linking results, and performs other operations. When BuildOfHew.bat is executed, compiling and linking are performed. As a result, a file named debugger.ABS, which is an executable file, is created within the folder. At the same time, a map file named debugger.MAP and a log file named log.txt are created. The map file indicates the program size and variable addresses. The compile results (whether there are any errors, etc.) are recorded in the log file (figure 3.4).

Note: * If HEW is installed to a folder other than C:\Hew, the compiler path setting and settings for environment variables used by the compiler in BuildOfHew.bat, as well as the library settings in InkSet1.sub, must be changed. Here the compiler path setting should be changed to the path of shc.exe, and the setting for the environment variable shc_lib used by the compiler should be set to the folder of shc.exe, the shc_inc setting should be changed to the folder of machine.h, and the setting of shc_tmp should specify the working folder for the compiler. The path of shcpic.lib should be specified for the library.

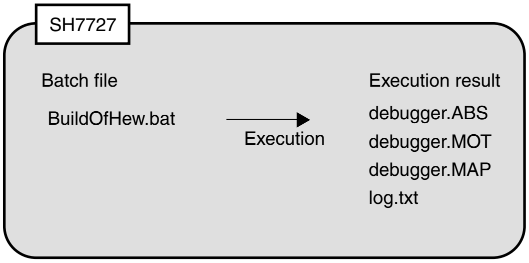


Figure 3.4 Compile Results

3.2.3 Request Generator

Files required for the USB packet generation tool, RequestGenerator, are all stored in the ReqGen folder.

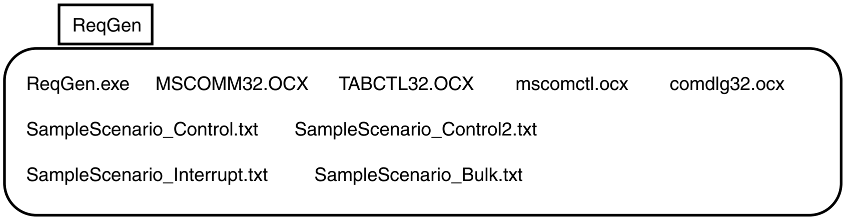
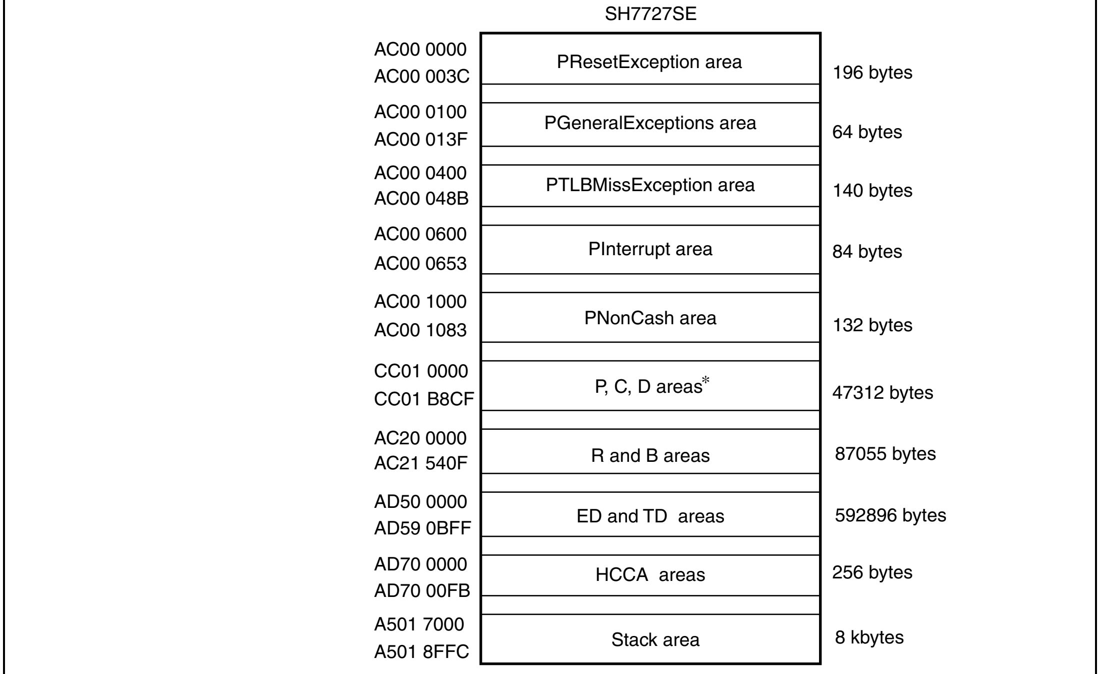


Figure 3.5 Files Included in ReqGen Folder

3.3 Loading and Executing the Program

Figure 3.6 shows the memory map for the sample program.



Notes: The memory map differs according to the compiler version, compiling conditions, firmware upgrade, etc.

* Placed in the P3 cache write-through space. Consequently the address bits A31 to A29 are 110.

Figure 3.6 Memory Map

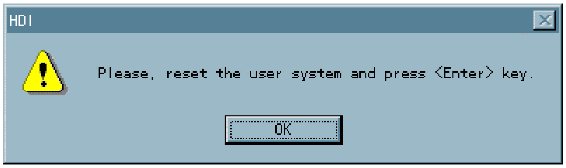
As shown in figure 3.6, this sample program allocates areas of P, C, D, R, and B to the SDRAM. In order to use the E10A for break and other functions, the program must be placed in RAM in this way. These memory allocations are specified by the lnkSet1.sub file in the SH7727 folder. When incorporating the program in ROM by writing it to flash memory or some other media, this file must be modified.

3.3.1 Loading the Program

In order to load the sample program into the SDRAM of the SH7727SE, the following procedure is used.

- Insert the E10A in which the HDI has been installed into the E10A PC, connect the E10A to the SH7727SE via a user cable, and connect the COM1 port of serially-connected PC to the SH7727SE via a serial cable.
- Turn on the power to the E10A PC and serially-connected PC for start up.
- Initiate the HDI.
- Turn on the power to the SH7727SE.
- A dialog (figure 3.7) is displayed on the PC screen; turn the SH7727SE reset switch (SW1) on, and after resetting the CPU, click the OK button or press the Enter key.
- Select CommandLine in the View menu to open a window (figure 3.8), click the BatchFile button on the upper left, and specify the 7727e10a.hdc file in the SH7727 folder. As a result, the BSC is set and access to the SDRAM is enabled.
- Select LoadProgram... from the File menu; in the Load Program dialog box, specify debugger.ABS in the SH7727 folder.
- Select Go from the Run menu bar to execute the program

Through the above procedure, the sample program can be loaded into the RAM of the SH7727SE.



Please, reset the user system and press <Enter> key.

Figure 3.7 Reset Request Dialog

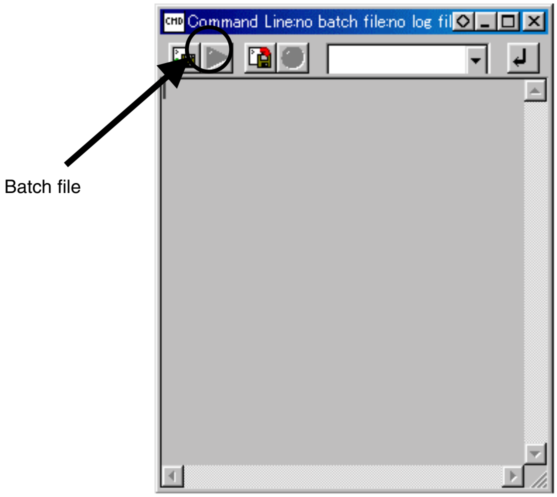
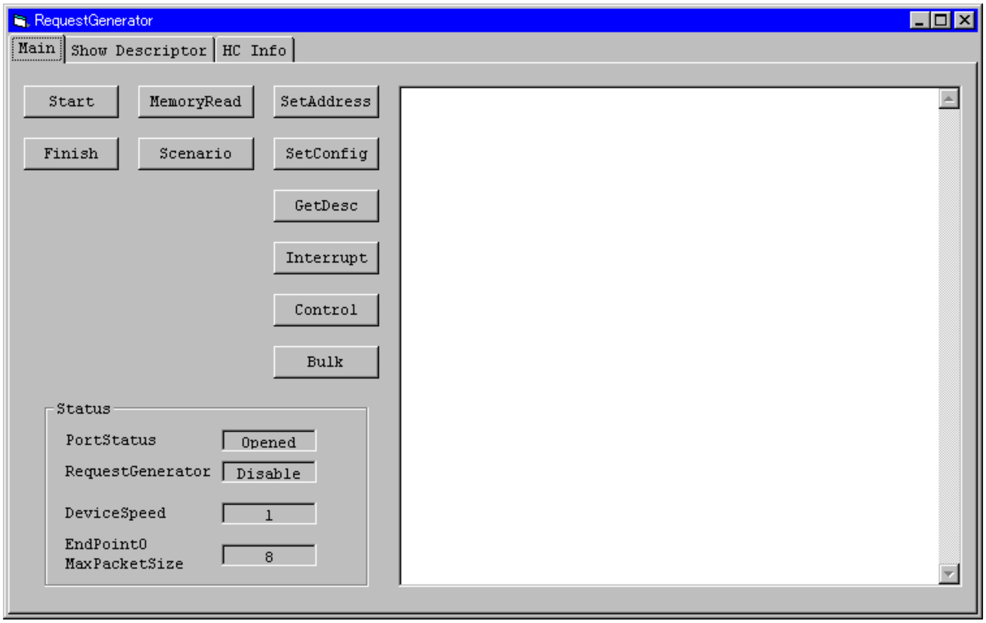


Figure 3.8 Command Line Input

3.4 Execution

Activation of RequestGenerator

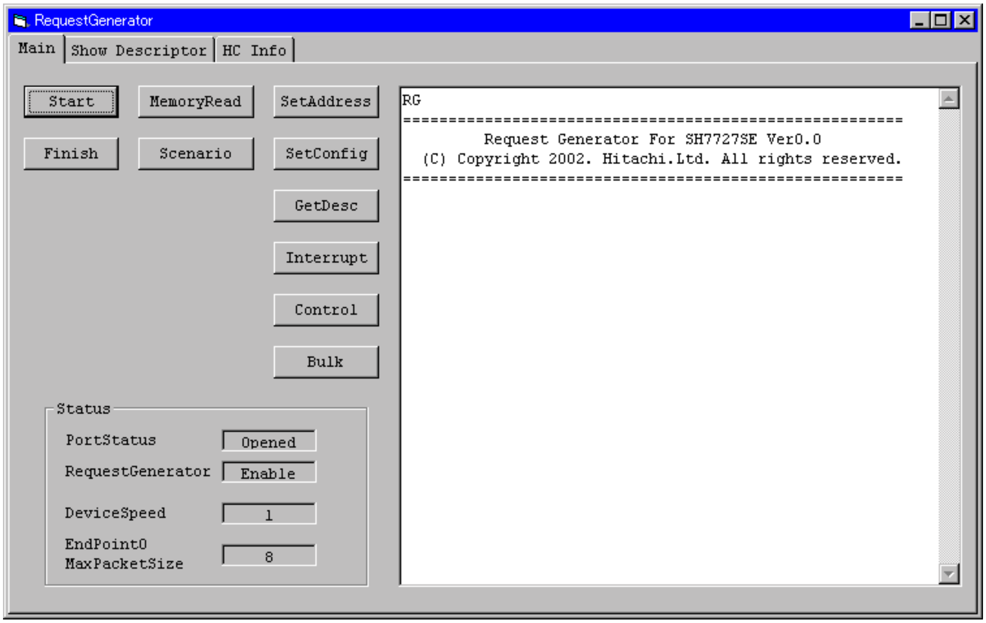
1. Execute the sample program according to the procedure of section 3.3.1. After successful launching of the sample program, "0xAA" is displayed on the SH7727SE's 8-bit LEDs.
2. Copy the .ocx file from the ReqGen folder to the Windows\System32 folder (Windows®98) or WinNT\System32 (Windows®2000).
3. Execute ReqGen.exe, which is in the ReqGen folder. This opens the COM1 port.



PortStatus Opened
RequestGenerator Disable
DeviceSpeed 1
EndPoint0 MaxPacketSize 8

Figure 3.9 Initial Screen of the RequestGenerator

4. Click Start.



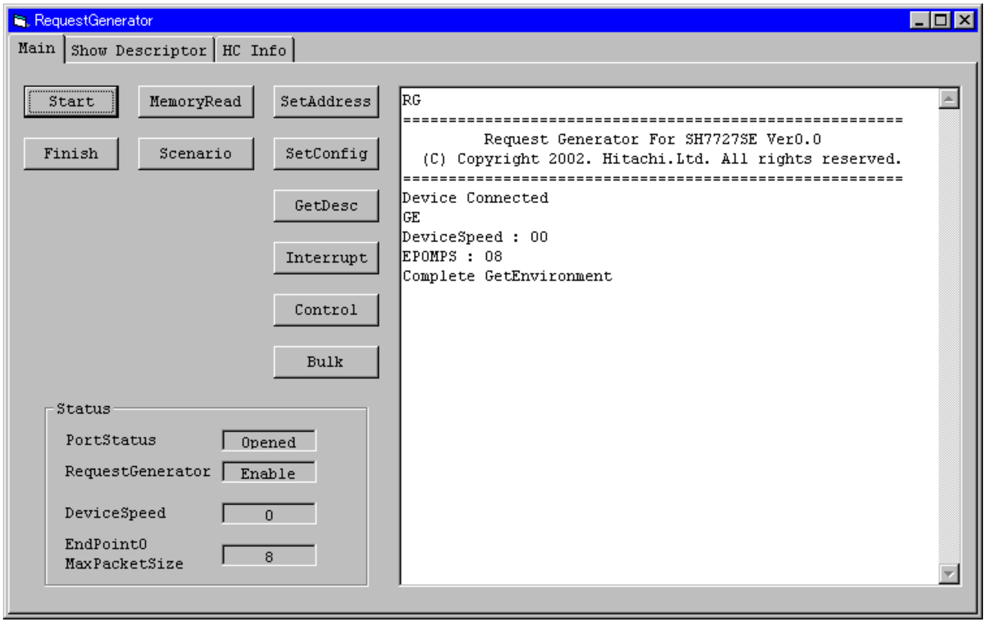
PortStatus: Opened
RequestGenerator: Enable
DeviceSpeed: 1
EndPoint0 MaxPacketSize: 8

```
RG
==============================================================
          Request Generator For SH7727SE Ver0.0
  (C) Copyright 2002. Hitachi.Ltd. All rights reserved.
==============================================================
```

Figure 3.10 Screen when the RequestGenerator is Active

5. Connect the USB-function device to the USB-A connector of the SH7727 SE. When the connection is detected, the GetDescriptor(Device) command is executed and the MaxPacketSize information on endpoint0 is obtained. Information on the speed of the connected device can also be obtained.

Note: To use the RequestGenerator, click Start before connecting the device.



```
RG
=======================================================
         Request Generator For SH7727SE Ver0.0
  (C) Copyright 2002. Hitachi.Ltd. All rights reserved.
=======================================================
Device Connected
GE
DeviceSpeed : 00
EP0MPS : 08
Complete GetEnvironment
```

Status

PortStatus	Opened
RequestGenerator	Enable
DeviceSpeed	0
EndPoint0 MaxPacketSize	8

Figure 3.11 RequestGenerator (Screen when a Device is Connected)

6. Perform the desired operation.
7. Pull the plug for the USB-function device from the SH7727SE's USB-A socket.
8. Click the Finish button.
9. Various USB-function devices can be placed under the control of the RequestGenerator by repeating steps 4 to 8.

The functions of the Request Generator's buttons are explained below.

- Start

  Activate the RequestGenerator by clicking the Start button. When the Request Generator is activated, the status display against "RequestGenerator" changes from Disable to Enable.
- Finish

  Click the Finish button to close the RequestGenerator. When the RequestGenerator is closed, the status display against "RequestGenerator" changes from Enable to Disable.
- SetAddress

  Click the SetAddress button to issue a SetAddress command. Input a USB device address (2 or greater) for allocation on the parameter-input screen.
- SetConfig

  Click the SetConfig button to issue a SetConfig command. Input the value for the required configuration on the parameter-input screen.

- GetDesc

  Click the GetDesc button to issue a GetDescriptor command. Input the descriptor type (only the Device and Configuration are supported) and size on the parameter-input screen.
- Control

  Any control transfer can be executed by clicking the Control button. Specify an 8-byte DeviceRequest value, the transfer direction for the data stage, and data and amount of data to be transferred in the data stage (this setting is prohibited with “In” transfers) on the parameter-input screen. Set the direction of data-stage transfer for commands that have no DataStage, such as SetAddress and SetConfiguration, as “Out”.

Notes: The SetAddress command cannot be executed through this function. Execute this command by clicking the SetAddress button.
Erroneous settings may lead to incorrect operation.

- Interrupt

  Click the Interrupt button to execute an interrupt transfer. Specify the endpoint number for the transfer, the direction of transfer, MaxPacketSize, PollingRate, amount of data to be transferred, the data to be transferred (setting data is unnecessary for “In” transfers), and the number of interrupt packets to be generated on the parameter-input screen.
- Bulk

  Click the Bulk button to execute a bulk transfer. Specify the endpoint number for the transfer, the direction of transfer, MaxPacketSize, PollingRate, amount of data to be transferred, and the data to be transferred (setting data is unnecessary for “In” transfers) on the parameter-input screen.
- MemoryRead

  Data can be read from a specified address of the SH7727SE by clicking the MemoryRead button. Specify the address, amount of data to be read, and access width (byte, word, or longword) on the parameter-input screen.
- Scenario

  Clicking the Scenario button allows you to set up a text file containing a list of processes for execution. Any button function other than Start, Finish, and Scenario can be included in the text file.
  The command names that correspond to the respective buttons, along with their parameters, are listed in table 3.2. Spaces must be included between the command name and parameters and between the individual parameters, and the line must end with a line-feed. For example, to include SetConfig for execution, enter “SC 1” then begin a new line. Table 3.3 gives some examples and descriptions.
  Parameters should be written in the order shown in figure 3.2. The listed parameters are the same as are specified by clicking the individual buttons on the parameter-input screen. The order of parameters is also the same as on the parameter-input screen.

Table 3.2 Commands in the Scenario Function

Button	Command	Parameter (D: decimal, H: hexadecimal)
SetAddress	SA	USB device address (D, 2 or greater)
SetConfig	SC	Configuration value (D)
GetDesc	DeviceDescriptor: GDD ConfigDescriptor: GDC	Amount to be transferred (H)
Interrupt	INT	Endpoint number (D), amount to be transferred (H), polling rate (D), direction of transfer (D, 1: OUT, 2: IN), data for transfer (H), MaxPacketSize (H), transfer count (H)
Control	CNT	DeviceRequest (H), amount to be transferred (H), direction of transfer (D, 1: OUT, 2: IN), data for transfer (H)
Bulk	BLK	Amount to be transferred (H), direction of transfer (D, 1: OUT, 2: IN), data for transfer (H), Endpoint number (D), MaxPacketSize (H)
MemoryRead	MR	Address to be accessed (H), amount of data (D), access unit (D, 1 : byte, 2 : word, 3 : longword)

Table 3.3 Examples of Scenario-File Entries

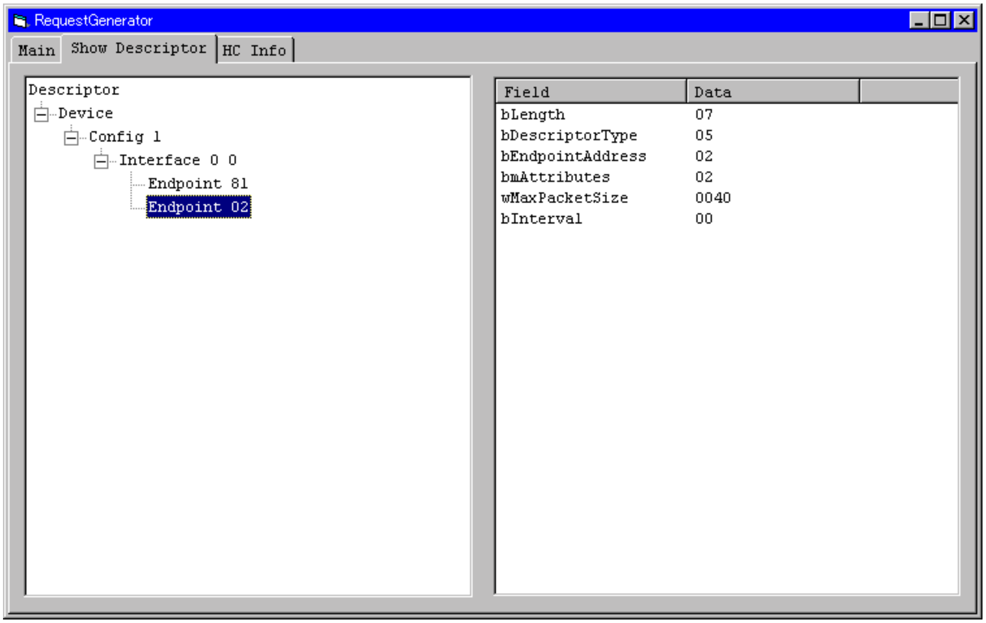
Examples of Writing	Transfer
SA 2 (line feed)	Issues the SetAddress command for address = 2
SC 1 (line feed)	Issues the SetConfig command for configuration = 1
GDD 12 (line feed)	Issues the GetDescriptor (Device) command for transfer size = 0x12
GDC 400 (line feed)	Issues the GetDescriptor(Config) command for transfer size = 0x400
CNT 0009010000000000 0 1 (line feed)	Issues the SetConfig command for configuration = 1
CNT 8006000100001200 12 2 (line feed)	Issues the GetDescriptor (Device) command for transfer size = 0x12
CNT 8006000200000004 0400 2 (line feed)	Issues the GetDescriptor(Config) command for transfer size = 0x400
INT 1 4 10 2 0 4 20 (line feed)	Interrupt transfer is carried out with the following settings. Endpoint = 1, amount for transfer = 0x4 bytes, polling rate = 10 msec, direction of transfer = 2 (IN), data for transfer = 0, MaxPacketSize = 0x4 bytes, transfer count = 0x20 times
BLK 40 1 (transfer data) 1 40 (line feed)	Bulk transfer is carried out with the following settings. Amount for transfer = 0x40 bytes, direction of transfer = 1 (OUT), data for transfer = (omitted), endpoint = 1, MaxPacketSize = 0x40 bytes
BLK 40 2 0 2 40 (line feed)	Bulk transfer is carried out on following settings. Amount for transfer = 0x40 bytes, direction of transfer = 2 (IN), data for transfer = 0, endpoint = 2, MaxPacketSize = 0x40 bytes

Notes: Write the command from the first column of each line.
Characters after // are ignored (treated as comments).
Tab codes are ignored.
The line feed must be included. A command that does not end with a line feed is not processed.
SetAddress is not executable by the CNT command. Set Address must be executed by using the SA command.

Other functions and points to note are explained below.

- Descriptor display function

  Select the Show Descriptor tab. Click the Descriptor button to view Descriptor information that has been obtained by clicking the GetDesc button .



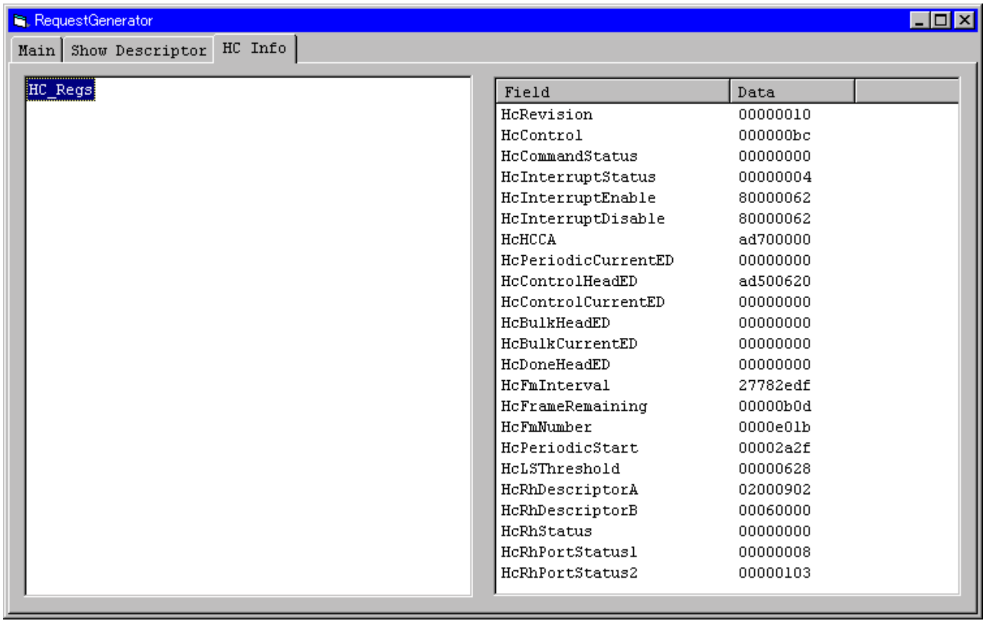
Field	Data
bLength	07
bDescriptorType	05
bEndpointAddress	02
bmAttributes	02
wMaxPacketSize	0040
bInterval	00

Figure 3.12 Request Generator (Descriptor Display)

A Descriptor's information can only be displayed when the Descriptor has been obtained through the GetDesc button. Information on Descriptors that is gained by the Control button or Scenario button cannot be displayed.

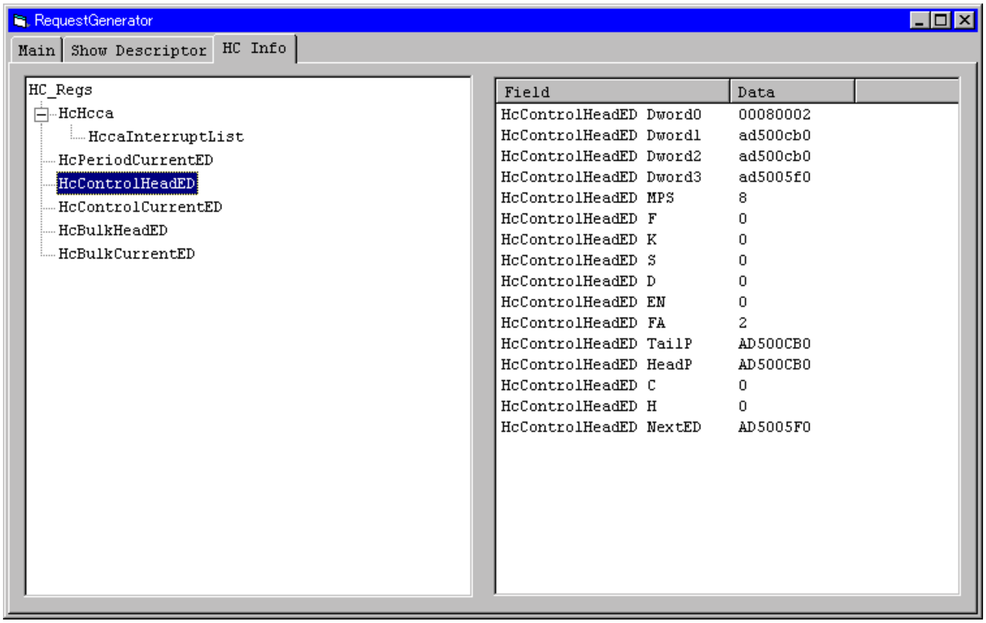
- Register browsing function

  Select the HC Info tab. The latest values of the USB host-module registers are displayed whenever the HC_Regs item is clicked. Double-click on HC_Regs to view the values in the fields of HcControlHeadED, HcControlCurrentED, HcBulkHeadED, and HcBulkCurrentED.



Field	Data
HcRevision	00000010
HcControl	000000bc
HcCommandStatus	00000000
HcInterruptStatus	00000004
HcInterruptEnable	80000062
HcInterruptDisable	80000062
HcHCCA	ad700000
HcPeriodicCurrentED	00000000
HcControlHeadED	ad500620
HcControlCurrentED	00000000
HcBulkHeadED	00000000
HcBulkCurrentED	00000000
HcDoneHeadED	00000000
HcFmInterval	27782edf
HcFrameRemaining	00000b0d
HcFmNumber	0000e01b
HcPeriodicStart	00002a2f
HcLSThreshold	00000628
HcRhDescriptorA	02000902
HcRhDescriptorB	00060000
HcRhStatus	00000000
HcRhPortStatus1	00000008
HcRhPortStatus2	00000103

Figure 3.13-1 RequestGenerator (Register Display)



Field	Data
HcControlHeadED Dword0	00080002
HcControlHeadED Dword1	ad500cb0
HcControlHeadED Dword2	ad500cb0
HcControlHeadED Dword3	ad5005f0
HcControlHeadED MPS	8
HcControlHeadED F	0
HcControlHeadED K	0
HcControlHeadED S	0
HcControlHeadED D	0
HcControlHeadED EN	0
HcControlHeadED FA	2
HcControlHeadED TailP	AD500CB0
HcControlHeadED HeadP	AD500CB0
HcControlHeadED C	0
HcControlHeadED H	0
HcControlHeadED NextED	AD5005F0

Figure 3.13-2 RequestGenerator (Register Display)

Section 4 Overview of the Sample Program

The features and structure of the sample program are explained in this section. The sample program runs on the SH7727SE and handles USB-host processing in response to interrupts from the USB-host module and branches from the main routine. The SCIF module handles communication with the RequestGenerator, which is the USB-packet generation tool. Of the interrupts from modules in the SH7727, the three that concern the USB-host module are RootHubStatusChange, WritebackDoneHead, and FrameNumberOverflow. Three other interrupts concern the SCIF module: ERI2 (receive error), BRI2 (break error), and RXI2 (receive-error FIFO full).

Features of this sample program are as follows:

- Control transfer can be performed.
- Bulk transfer can be used.
- Interrupt transfer can be performed.
- Transfer requests can be generated by using the RequestGenerator packet-generation tool on the PC which is connected to the SH7727SE with serial cable.

Note: Isochronous transfer is not supported.
The Suspend and Resume states are not supported.

RENESAS

4.1 State-Transition Diagram

Figure 4.1 shows a state transition diagram for this sample program. In this sample program, as shown in figure 4.1, there are transitions between six states.

- Reset State

  Upon power-on reset and manual reset, this state is entered. In this reset state, the SH7727 mainly performs initial settings.
- Connection-Wait State

  This state is entered after initial settings have been completed in the reset state. The program returns to this state when a device is disconnected from the Root Hub while the program is in the steady state. In this state, the program waits for a RootHubStatusChange interrupt. When this interrupt is generated by the connection of a device to the Root Hub, processing for connection is carried out, and the steady state is then entered.
- Steady State

  The program enters this state after a device has been connected to the Root Hub while the program was in the connection-wait state. In this state, EDs and TDs are generated and transfer requests are generated for the HC. Processing for the requested transfer is then completed. The program also controls the SCIF module to perform serial output.
- Root Hub Processing

  The program enters this state in response to a RootHubStatusChange interrupt that occurs while the program is in the connection-wait or steady state. Since this interrupt occurs when the root hub condition changes, the program decides the interrupt source and handles the following: connection processing when the root hub condition changes from the disconnection state to connection state, disconnection processing when the root hub condition changes from the connection state to the disconnection state, and overcurrent clearing processing when the overcurrent state is entered.
- Done-Queue Processing

  The program enters this state in response to a WriteBackDoneHead interrupt that occurs while the program is in the steady state. The Done Queue, which is the list of completed TDs, is received in this state.
- Serial-Communications State

  The program enters this state in response to a serial-receive interrupt that occurs while the program is in the steady state. Communications with the RequestGenerator, which is a tool running on the PC, take place in this state.

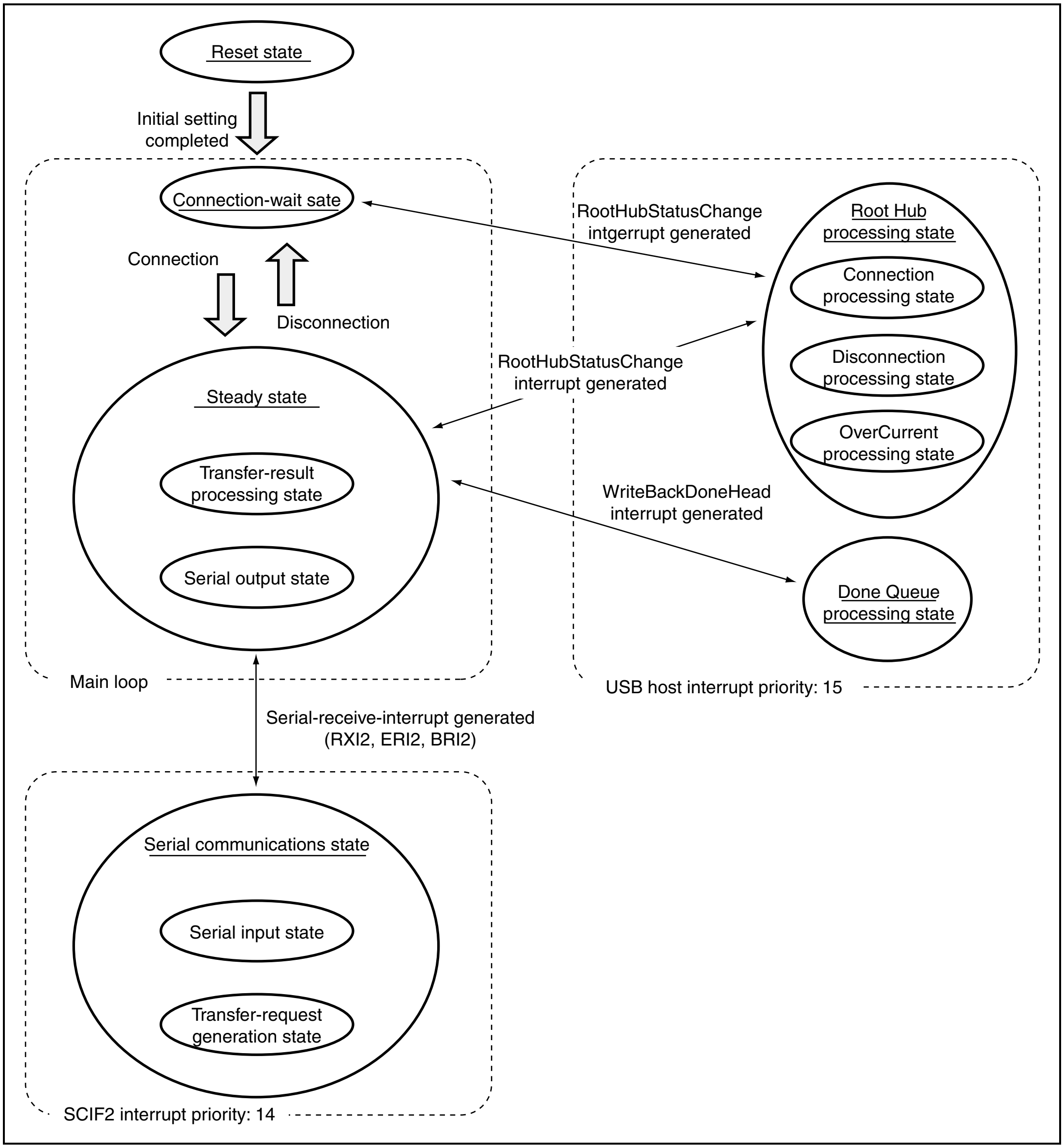

Reset state
Initial setting completed
Connection-wait sate
Connection
Disconnection
Steady state
Transfer-result processing state
Serial output state
Main loop
RootHubStatusChange intgerrupt generated
RootHubStatusChange interrupt generated
Root Hub processing state
Connection processing state
Disconnection processing state
OverCurrent processing state
WriteBackDoneHead interrupt generated
Done Queue processing state
USB host interrupt priority: 15
Serial-receive-interrupt generated (RXI2, ERI2, BRI2)
Serial communications state
Serial input state
Transfer-request generation state
SCIF2 interrupt priority: 14

Figure 4.1 State-Transition Diagram

4.2 Types of Interrupts

As was explained at the beginning of section 4, the USB host module and SCIF interrupts are used in this sample program. The interrupt sources in use are indicated by the HcInterruptStatus register of the USB host module and the serial status register, SCSSR2, for the SCIF; there are three interrupts for each. When an interrupt occurs, the flag bit for the corresponding interrupt source is set to 1 and an interrupt request is sent to the CPU. The sample program includes appropriate interrupt handling. Figure 4.2 shows the types of interrupts.

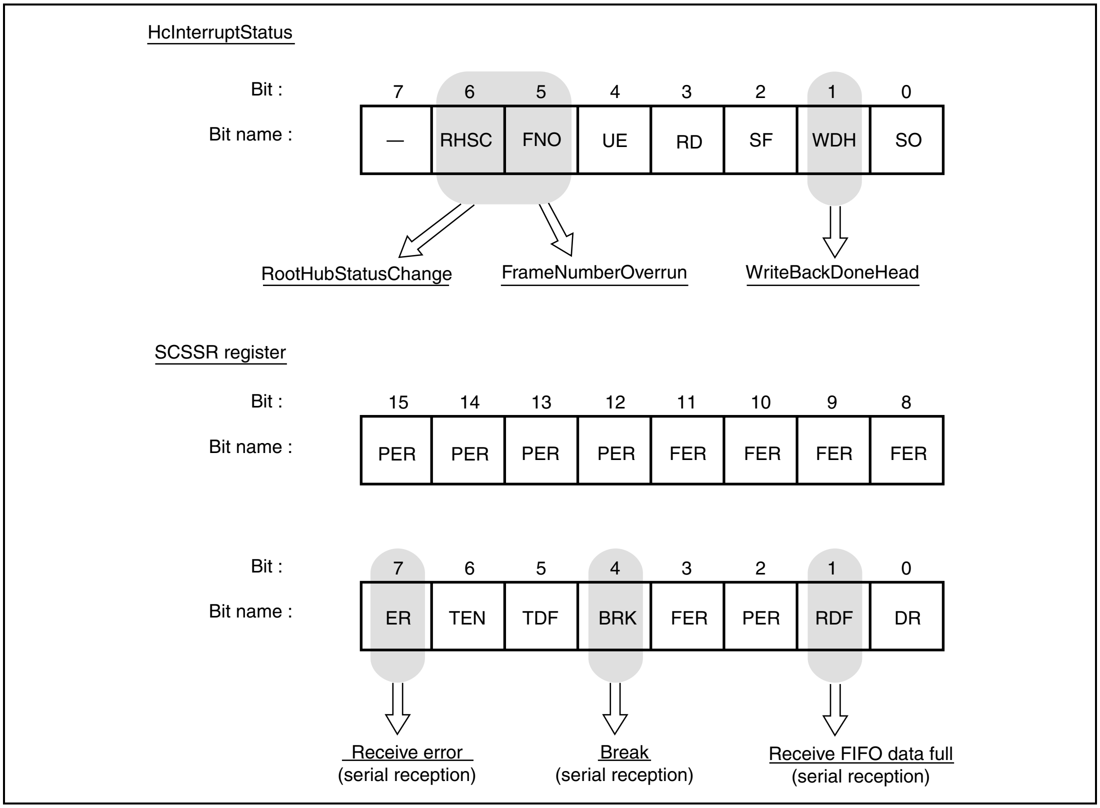


Figure 4.2 Types of Interrupts

4.3 File Structure

This sample program consists of nine source files and ten header files. The overall file structure is shown in table 4.1. The relationships among files are shown as layered configuration in figure 4.3.

Table 4.1 File Structure

File Name	Principle Role
StartUp.c	Microcomputer initial setttings
Usbh_Hcd_Tasks.c	OpenHCI specification functions
Usbh_Hcd_Others.c	HCD layer common function
Usbh_Hcd_Main.c	Main routine of HCD layer, interrupt functions, etc.
Usbh_Usbd_Common.c	USBD layer common function
Usbh_Dr_EnuDr.c	Driver layer function for Enumeration processing
Usbh_Dr_ReqGenDr.c	Driver layer function for RequestGenerator
Usbh_App_Log.c	Serial output function
Usbh_App_Link.c	Function for USB host module and SCIF module conjunction processing
SH7727.h	SH7727 register definition
Usbh_Hcd_TypeDef.h	HCD layer structure declaration
Usbh_Hcd_ProType.h	Prototype declaration of HCD layer
Usbh_Hcd_Defs.h	Various declarations of HCD layer
Usbh_Usbd_Defs.h	Various declarations of USBD layer
Usbh_Dr_EmuDrDefs.h	Various declarations of EnumerationDriver
Usbh_Dr_DrList.h	DeviceDriver list called by EnumerationDriver
Usbh_Dr_ReqGenDr.h	Various declarations of ReqGenDr
Usbh_App_Link.h	Various declarations required for USB host module and SCIF module conjunction processing
Usbh_Common.h	Common declaration of whole USB host

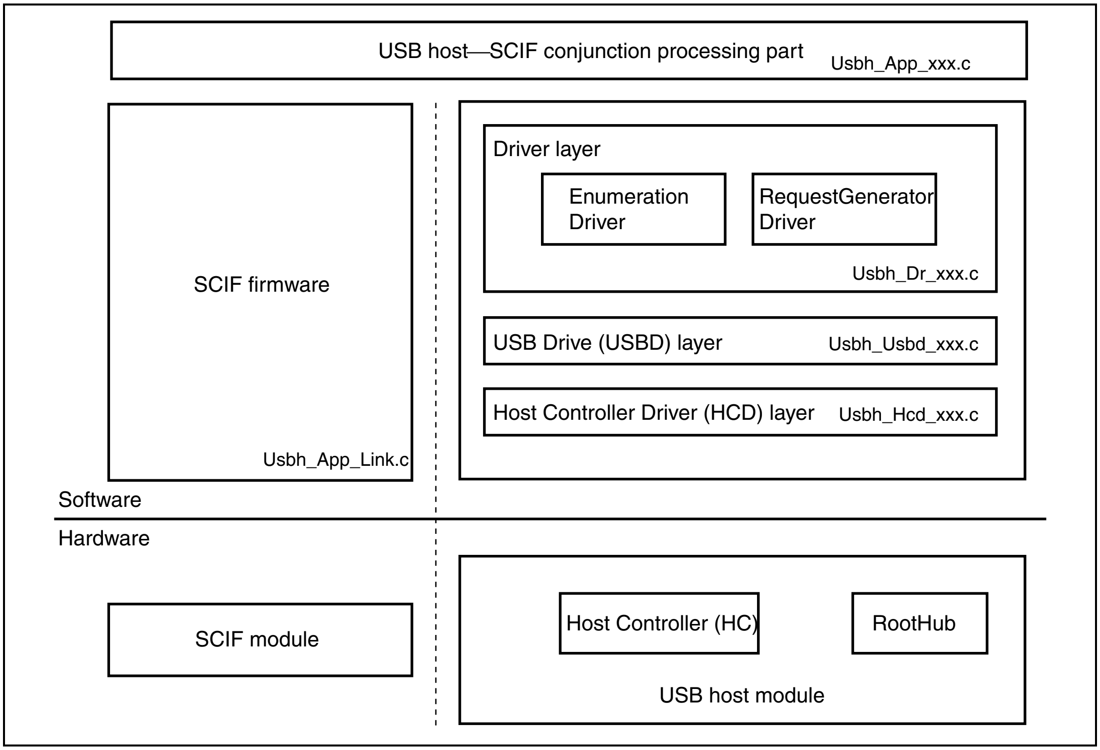


Figure 4.3 Layer Structure of Firmware

4.4 Purposes of Functions

Table 4.2 shows functions contained in each file and their purposes.

Table 4.2-1 StartUp.c

File in Which Stored	Function Name	Purpose
StartUp.c	CallReseException	Operates for reset exception and calls function to be executed
	CallGeneralException	Calls function corresponding to general exception other than TLB miss occurrence
	CallTLBMissException	Calls function corresponding to TLB miss occurrence
	CallInterrupt	Calls function corresponding to interrupt request
	SetPowerOnSection	Module and memory initialization, and shift to main loop
	_INITSCT	Copies variables that have initial settings to the RAM work area
	InitMemory	Clears RAM area used in bulk transfer
	InitSystem	Pull-up control of the USB bus
	SciInit	SCIF module initialization

When a power-on reset or manual reset is carried out, SetPowerOnSection in StartUp.c is called. At this point, the SH7727 initial settings are carried out. After that, the RAM area used for the transfers is cleared.

Table 4.2-2 Usbh_Hcd_Main.c

File in Which Stored	Function Name	Purpose
Usbh_Hcd_Main.c	HCD_Setup	Initializes the HCD
	HCD_IntRoutine	Interrupt processing routine
	HCD_MainRoutine	HCD layer main routine
	HCD_ControlRootHub	Control RootHub

In Usbh_Hcd_Main.c, the HCD is initialized and various interrupts are processed when they are occurred. Also, when the RootHubStatusChange interrupt is occurred, Root Hub is controlled.

Table 4.2-3 Usb_Hcd_Tasks.c

File in Which Stored	Function Name	Purpose
Usbh_Hcd_Tasks.c	InsertEDForEndpoint	Generate ED and add it to list
	QueueGeneralRequest	Generate TD and link it to ED
	ProcessDoneQueue	DoneQueue processing
	InitializeInterruptLists	Build Interrupt list
	OpenPipe	Generate periodic ED and add it to list
	CheckBandwidth	Check bandwidth
	PauseED	Pause ED
	ProcessPausedED	Pause ED
	RemoveED	Delete existing ED
	CancelRequest	Cancel processing of existing USBDRequest. Delete generated TD.
	UnscheduleIsochronousOr InterruptEndpoint	Delete periodic ED
	SetFrameInterval	Arrange value of frame interval
	Get32BitFrameNumber	Generate 32-bit frame number from 16-bit length HccaFrameNumber

Functions of Usbh_Hcd_Tasks.c are all based on the sample codes in section 5 of the OpenHCI specification. The function carries out generation and deletion of EDs and TDs.

Table 4.2-4 Usbh_Hcd_Others.c

File in Which Stored	Function Name	Purpose
Usbh_Hcd_Others.c	HCD_CreateDeviceData	Initialize DeviceData structure variable
	HCD_CreateEndPoint	Initialize Endpoint structure variable
	HCD_SetupEndpoint	Setup Endpoint structure variable
	AllocateEndpointDescriptor	Reserve ED variable
	PhysicalAddressOf	Obtain address
	IsListEmply	Define whether List_Entry variable is null or not
	AllocateTransferDescriptor	Reserve TD variable
	InsertHeadList	Insert List_Entry variable in start of List
	InsertTailList	Insert List_Entry variable in end of List
	Containing_Record	Find ED and TD which meet condition
	InitializeListHead	Initialize List_Entry variable

File in Which Stored	Function Name	Purpose
Usbh_Hcd_Others.c	min	Define minimum value
	CompleteUsbdRequest	Send completed USBDRequest variable to USBD layer
	FreeTransferDescriptor	Delete TD variable
	RemoveListHead	Delete LIST_ENTRY variable
	RemoveListEntry	Delete LIST_ENTRY
	VirtualAddressOf	Obtain address
	HCD_InitializeEndpoint	Generate Endpoint variable and make ED variable
	HCD_GetRootDeviceSpeed	Obtain information of device speed connected to RootHub
	HCD_ScanEndpoint	Access to array for Endpoint control
	HCD_StoreEndpoint	Access to array for Endpoint control
	HCD_Request	Generate Endpoint variable, ED variable, and TD variable from received USBDRequest
	HCD_CheckDiffEPs	Confirm whether setting of Endpoint has changed
	HCD_CheckRemainedTDs	Check the number of remaining TD variable
	HCD_CheckRemainedEDs	Check the number of remaining ED variable
	HCD_CheckRemainedEPs	Check the number of remaining Endpoint
	HCD_PauseEndpoint	Pause ED processing
	HCD_RemoveEndpoint	Delete Endpoint variable and ED variable
	HCD_CancelRequest	Delete transfer requested USBDRequest
	HCD_FreeEndpoint	Clear Endpoint variable
	FreeEndpointDescriptor	Clear ED variable
	HCD_ClearDeviceData	Clear DeviceData variable
	HCD_ClearList	Initialize array for Endpoint, ED, and TD variables
	HCD_ClearHCCA	Initialize HCCA area
	HCD_WaitConnection Complete	Connect device to RootHub
	HCD_WaitRoutine	Wait routine

In Usbh_Hcd_Others.c contains common functions called from the HCD layer function and functions called from the USBD layer.

Table 4.2-5 Usb_h_Usbd_Common.c

File in Which Stored	Function Name	Purpose
Usbh_Usbd_Common.c	USBD_SetupDriverRequest ToUSBDRequestndpoint	Make USBDRequest structure variable from DriverRequest structure variable
	USBD_ReceiveDriver Request	Receive DriverRequest structure variable from Dr layer
	USBD_ReceiveUSBDR Request	Receive completed USBDRequest structure variable from HCD layer
	USBD_CreateRequest	Reserve USBDRequest structure variable and initialize
	USBD_FreeRequest	Initialize USBDRequest structure variable
	USBD_GetDeviceAddress	Obtain value of usable DeviceAddress
	USBD_ReportConnection	Called when a device is connected to RootHub and initialize USBD layer
	USBD_ReportDisConnection	Called when a device is disconnected from RootHub and processing of disconnection is carried
	USBD_RemoveUSBDRequest	Delete processing requested USBDRequest to HCD
	USBD_RemoveDevice	Delete endpoint structure variable in specified DeviceAddress
	USBD_RemoveEndpoint	Delete specified Endpoint structure variable
	USBD_ReadDAArray	Access to array for DeviceAddress control
	USBD_WriteDAArray	Access to array for DeviceAddress control
	USBD_RootDeviceSpeedInfo	Obtain information of device speed connected to RootHub

Usbh_Usbd_Common.c is the function group in the USBD layer which receives transfer request from the Driver layer and transfers the request to the HCD layer. Then, the group receives the transfer result from the HCD layer and informs it to the requesting driver.

Table 4.2-6 Usbh_Dr_EnuDr.c

File in Which Stored	Function Name	Purpose
Usbh_Dr_EnuDr.c	EnumeDriver_Start	Initialize EnumerationDr and ready to start transfer
	EnumeDriver_Result	Receive the result of transfer request to USBD layer and request next transfer
	EnumeDriver_Request	Transfer request to USBD layer
	EnumeDriver_GetInfoFrom Descriptor	Selects necessary information among obtained Descriptor information
	EnumeDriver_GetDevice Descriptor	Selects necessary information from obtained DeviceDescriptor
	EnumeDriver_GetConfig Descriptor	Selects necessary information from obtained ConfigDescriptor
	EnumeDriver_GetInterface Descriptor	Selects necessary information from obtained InterfaceDescriptor
	EnumeDriver_GetEndpoint Descriptor	Selects necessary information from obtained EndpointDescriptor
	EnumeDriver_Initialize	Initialize EnumerationDriver
	EnumeDriver_Clear	Clear internal flag of EnumerationDriver
	EnumeDriver_EnableDriver	Define class of connected device and call most appropriate Driver function
	EnumeDriver_Enable	Enable EnumerationDriver
	EnumeDriver_Disable	Disable EnumerationDriver
	DummyDriver_Start	Dummy Dreiver execution function
	DriverCommon_SetupDriver Request	Gather transfer request to DriverRequest
	DriverCommon_DoDrivr Request	Send DriverRequest function to USB layer and request transfer
	DriverCommon_Endian	Endian conversion

Usbh_Dr_EmuDr.c carries out the processing for EnumerationDriver. The function carries SetAddress for the connected device, distinguishes DeviceClass, and defines which Driver to call.

Table 4.2-7 Usbh_App_Log.c

File in Which Stored	Function Name	Purpose
Usbh_App_Log.c	App_LogOut_Char	Function to output character string
	App_LogOut_Data	Function to output character string
	App_ExOutPut	Function to output 1-byte string variable

Usbh_App_Log.c serially outputs information of debugging.

Table 4.2-8 Usbh_Dr_ReqGenDr.c

File in Which Stored	Function Name	Purpose
Usbh_Dr_ReqGenDr.c	App_ReceiveCommand	Receive command data from RequestGenerator
	App_CheckReceiveCommand	Define received command data
	DisableRequestGenerator	Carry out RequestGenerator end processing
	DoRequestGenerator	Initialize RequestGeneratorDriver
	DoSetAddress	Request SetAddress
	ReceiveSetAddressResult	Receive result of SetAddress request
	DoGetDescriptorDevice	Request GetDescriptor (Device) request
	ReceiveGetDescriptorDevice Result	Receive result of GetDescriptor (Device) request
	DoGetDescriptorConfig	Request GetDescriptor (Config) request
	ReceiveGetDescriptorConfig Result	Receive the result of GetDescriptor (Config) request
	DoSetConfiguration	Request SetConfiguration
	ReceiveSetConfigurationResult	Receive the result of SetConfiguration request
	DoIntTransfer	Request interrupt transfer
	ReceiveIntTransferResult	Receive the result of interrupt transfer request
	DoSetEnvironment	Set DeviceSpeed information and information of MaxPacketSize of Ep0
	DoGetEnvironment	Request obtaining DeviceSpeed information and information of MaxPacketSize of Ep0
	ReceiveGetEnvironmentResult	Receive DeviceSpeed information and information of MaxPacketSize of Ep0
	DoControlTransfer	Request any control transfer
	ReceiveControlTransferResult	Receive result of control transfer request
	DoBulkTransfer	Request bulk transfer
	ReceiveBulkTransferResult	Receive result of bulk transfer request
	DoMemoryRead	Data is read from specified address
	mystrtoul	Convert character string to hexadecimal

File in Which Stored	Function Name	Purpose
Usbh_Dr_ReqGenDr.c	DisplayTitle	Output title of RequestGenerator
	DisplayHelp	Indicate Help information

Usbh_Dr_ReqGenDr.c receives a transfer request from the USB packet generation tool RequestGenerator on the PC and returns the result.

Table 4.2-9 Usbh_App_Link.c

File in Which Stored	Function Name	Purpose
Usbh_App_Link.c	App_SerialLn	SerialIn processing
	App_SerialOut	Serial out processing

Usbh_App_Link.c performs communication between RequestGeneratotrDriver and RequestGenerator on the PC by controlling the SCIF module.

Figure 4.4 shows the interrelations between the functions explained in table 4.2. The upper-side functions call the lower-side functions. Also, multiple functions may call the same function. In the steady state, CallResetException calls other functions, and in the case of a transition to the USB communication state which occurs on an interrupt, CallInterrupt which is an interrupt function calls HCD_IntRoutine function. In the SCIF interrupt, CallInterrupt calls AppSerialIn. Figure 4.4 shows the hierarchical relation of functions; there is no order for function calling. For information on the order in which functions are called, refer to the flowcharts in section 6.

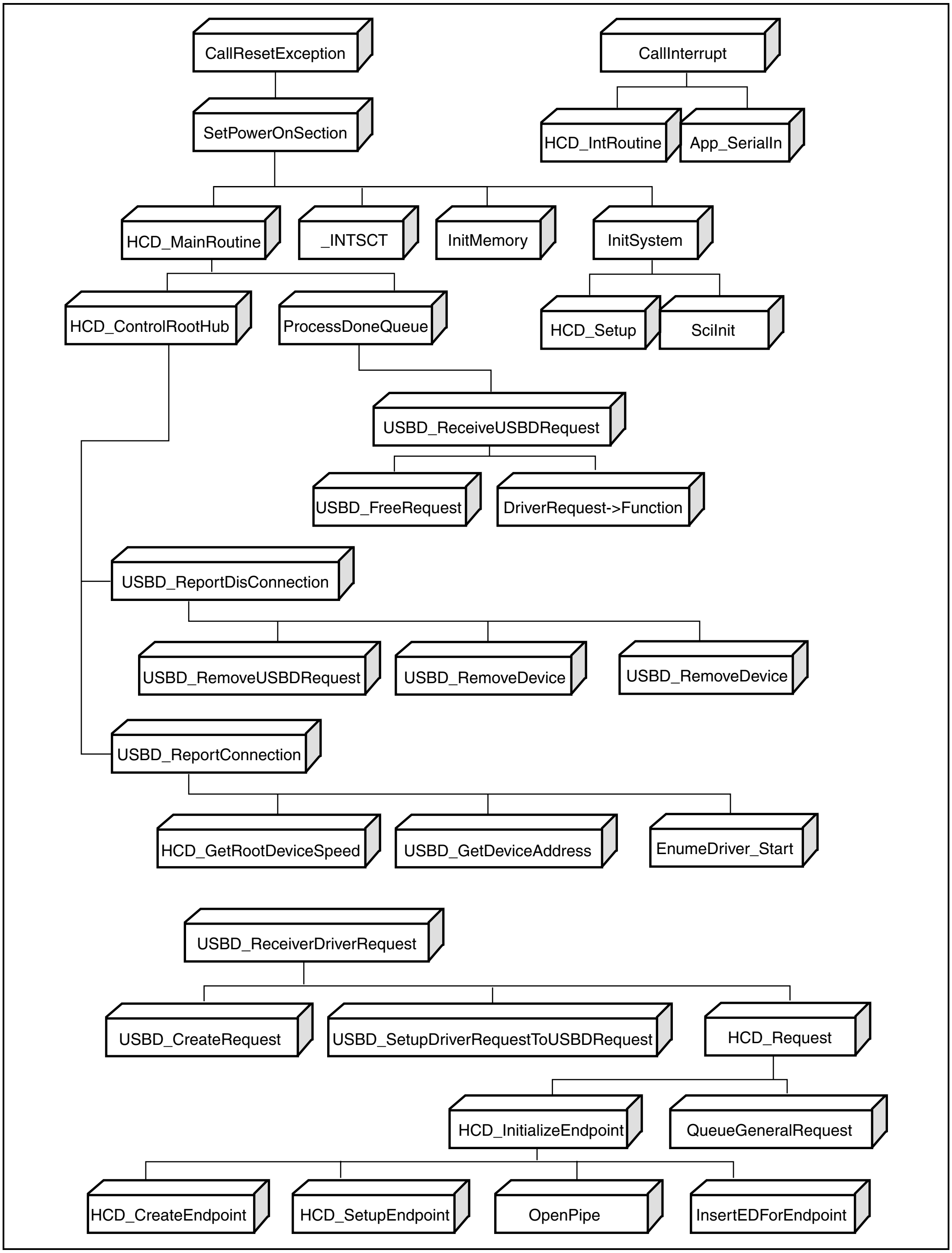


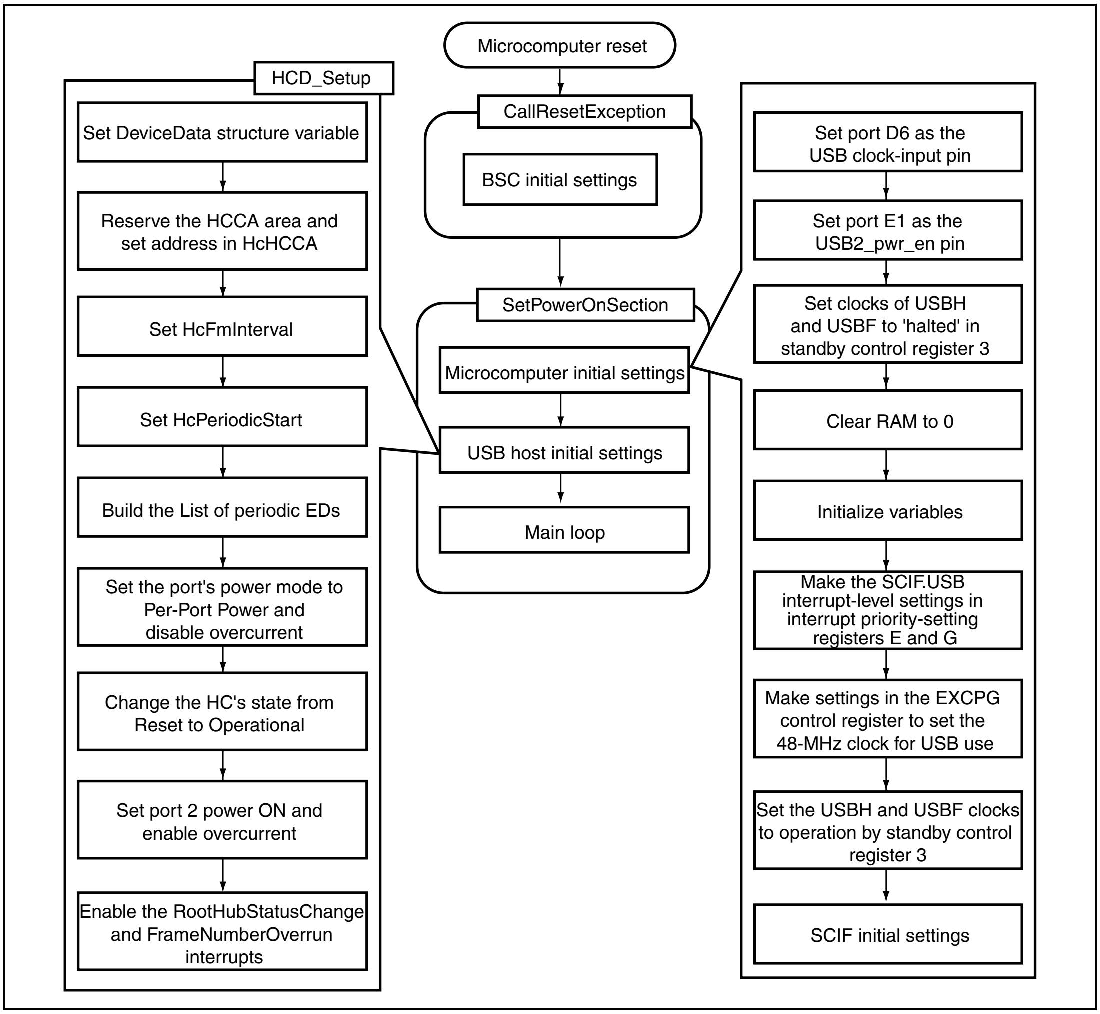
Figure 4.4 Interrelationship between Functions

Section 5 Operation of Sample Program

The operation of the sample program is explained in this section.

5.1 Reset State

The internal state of the CPU and the registers of the on-chip peripheral modules are initialized while the microcomputer is in the reset state. Next, the reset-interrupt function CallResetException is called, reset-exception processing is handled, and the SetPowerOnSection function is called. Figure 5.1 is a flow chart of operation from occurrence of the reset interrupt occurs to entry to the steady state (main loop).



Microcomputer reset
CallResetException
BSC initial settings
SetPowerOnSection
Microcomputer initial settings
USB host initial settings
Main loop
HCD_Setup
Set DeviceData structure variable
Reserve the HCCA area and set address in HcHCCA
Set HcFmInterval
Set HcPeriodicStart
Build the List of periodic EDs
Set the port's power mode to Per-Port Power and disable overcurrent
Change the HC's state from Reset to Operational
Set port 2 power ON and enable overcurrent
Enable the RootHubStatusChange and FrameNumberOverrun interrupts
Set port D6 as the USB clock-input pin
Set port E1 as the USB2_pwr_en pin
Set clocks of USBH and USBF to 'halted' in standby control register 3
Clear RAM to 0
Initialize variables
Make the SCIF.USB interrupt-level settings in interrupt priority-setting registers E and G
Make settings in the EXCPG control register to set the 48-MHz clock for USB use
Set the USBH and USBF clocks to operation by standby control register 3
SCIF initial settings

Figure 5.1 Processing in the Reset State

Set items such as pins and interrupt levels that are related to operation of the USB host so that the host is in a usable state before it is initialized. When the USB host is initialized, it enters the connection-wait state.

5.2 Main Loop (Connection-Wait and Steady States)

This loop is entered after initial settings have been completed in the reset state. In the main loop, the program waits for USB-host interrupts. Figure 5.2 is the flowchart of the main loop.

HCD_IntRoutine is called in response to a USB-host interrupt. This function sets the flag (IntFlag) which indicates generation of the USB-host interrupt, disables the interrupt, and ends immediately after that. The actual processing in response to the interrupts is carried out in the main routine.

The main routine constantly checks for the occurrence of USB-host interrupts. This is done by checking whether or not IntFlag is set to 1. When IntFlag is set to 1, processing that corresponds to the generated interrupt is carried out, and interrupts are enabled again.

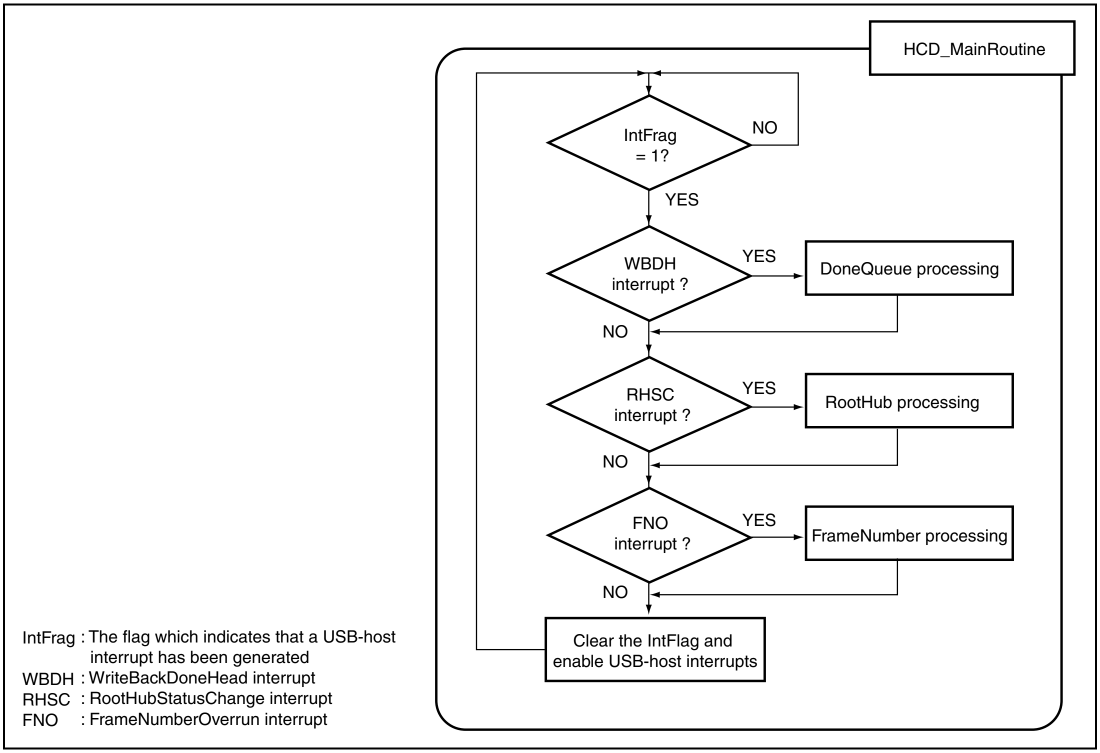

HCD_MainRoutine
IntFrag = 1?
NO
YES
WBDH interrupt ?
YES
DoneQueue processing
NO
RHSC interrupt ?
YES
RootHub processing
NO
FNO interrupt ?
YES
FrameNumber processing
NO
Clear the IntFlag and enable USB-host interrupts
IntFrag : The flag which indicates that a USB-host interrupt has been generated
WBDH : WriteBackDoneHead interrupt
RHSC : RootHubStatusChange interrupt
FNO : FrameNumberOverrun interrupt

Figure 5.2 Main Loop

5.3 Root-Hub Processing State

The flow chart for Root-Hub processing is given in figure 5.3. This state is entered when the RootHubStatusChange interrupt occurs while the program is in the connection-wait state and steady state, i.e., in the main loop. This interrupt occurs when the state of the Root Hub has been changed. Processing is carried out as shown in the following figure in response to the detection of any of three states: connection of a device to the Root Hub, disconnection of a device from the Root Hub, or when a port of Root Hub is overcurrented.

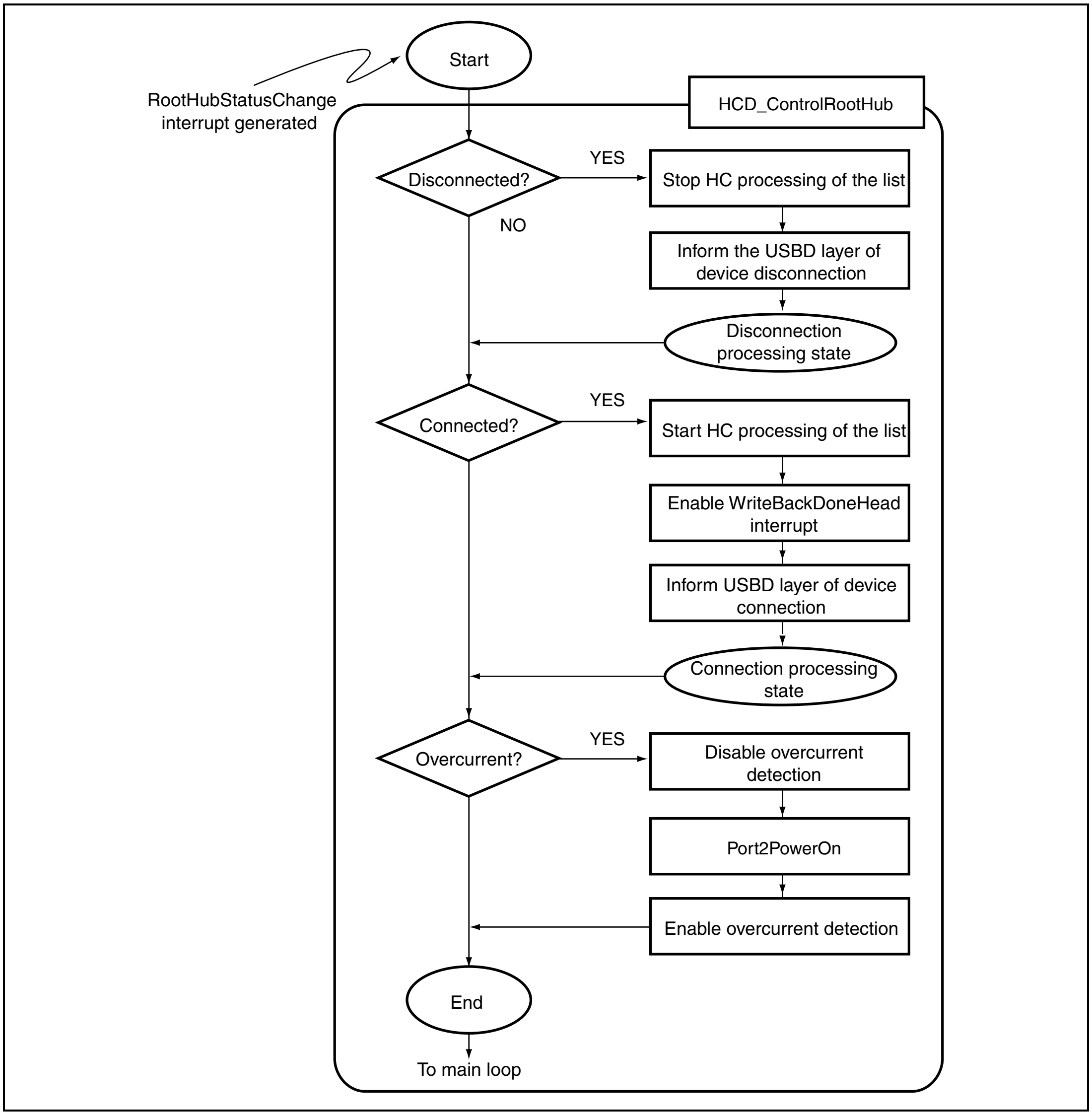

Start
RootHubStatusChange interrupt generated
HCD_ControlRootHub
Disconnected?
YES
NO
Stop HC processing of the list
Inform the USBD layer of device disconnection
Disconnection processing state
Connected?
YES
Start HC processing of the list
Enable WriteBackDoneHead interrupt
Inform USBD layer of device connection
Connection processing state
Overcurrent?
YES
Disable overcurrent detection
Port2PowerOn
Enable overcurrent detection
End
To main loop

Figure 5.3 Root-Hub Processing

5.4 Connection-Processing State

Figure 5.4 shows the connection processing for a connected device. As was explained in section 5.3, connection of a device to the Root Hub generates a RootHubStatusChange interrupt, and the USBD layer is informed of the connection. The function USBD_ReportConnection is called from the HCD_ControlRootHub function. USBD_ReportConnection initializes the USBD layer and calls the EnumerationDriver. The EnumerationDriver carries out the SetAddress processing to allocate the DeviceAddress, and, through GetDescriptor processing, obtains the Descriptor information on the device that has been connected. Of this information, the Class information on the connected device is used to select the most appropriate driver.

Note: In this sample program, the connected device is controlled from the RequestGenerator, a tool which runs on the PC. Therefore, drivers are not called.

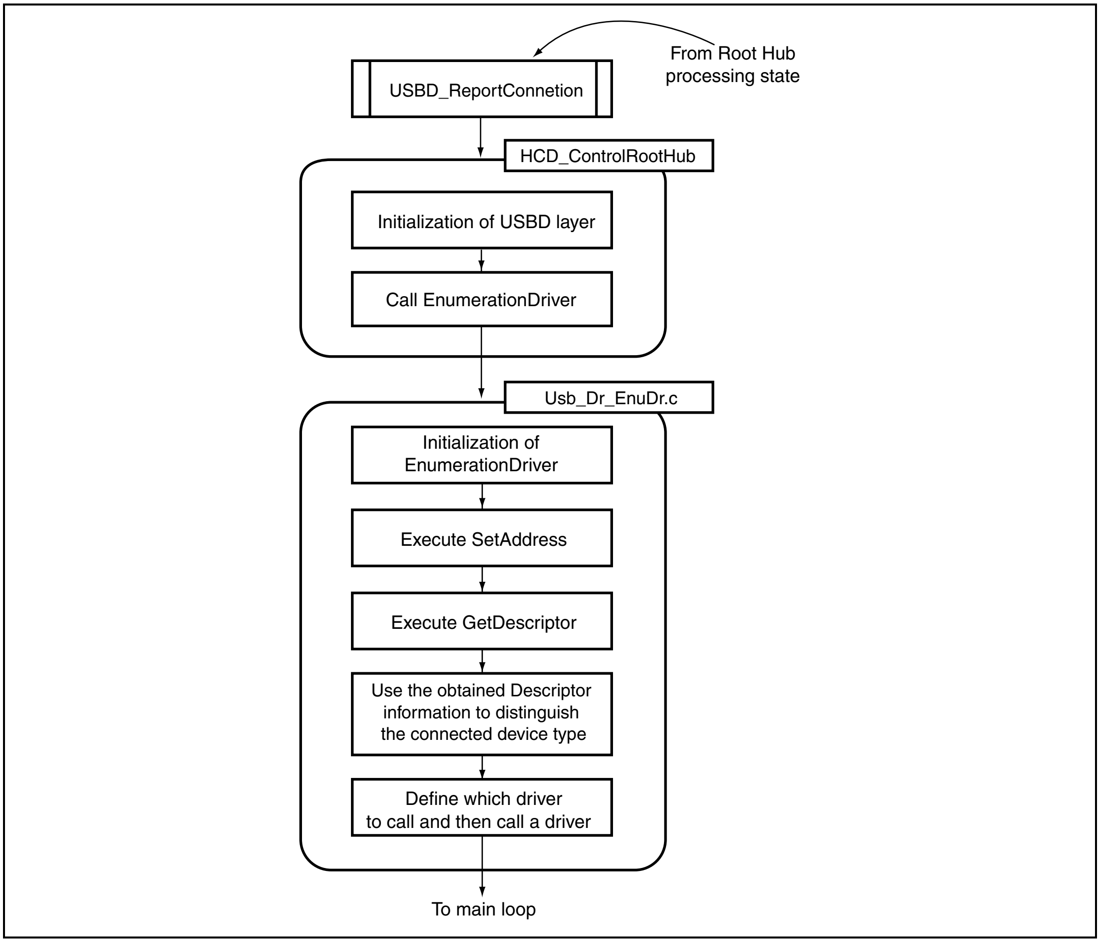
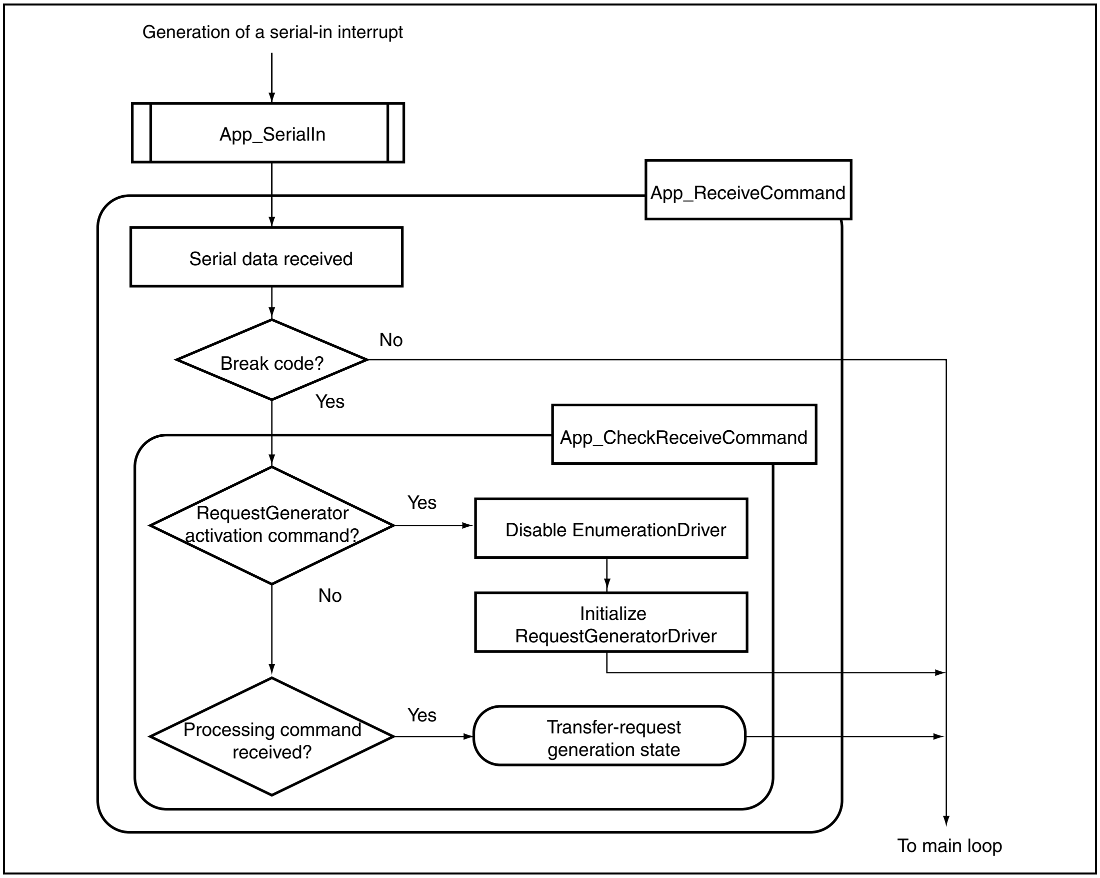

From Root Hub processing state
USBD_ReportConnetion
HCD_ControlRootHub
Initialization of USBD layer
Call EnumerationDriver
Usb_Dr_EnuDr.c
Initialization of EnumerationDriver
Execute SetAddress
Execute GetDescriptor
Use the obtained Descriptor information to distinguish the connected device type
Define which driver to call and then call a driver
To main loop

Figure 5.4 Connection Processing

5.5 Serial Input State (RequestGeneratorDriver-Processing State)

The flow chart for serial input processing is given in figure 5.5.

Clicking the RequestGenerator's Start button sends the command RG, which enables the USB-packet function. In response to this command, the RequestGeneratorDriver is read and the EnumerationDriver is disabled on the SH7727SE side. Thus, when a USB-function device is connected to the USB-A connector, the EnumerationDriver is not read, this allows USB control by the RequestGeneratorDriver (i.e., RequestGenerator). After that, the RequestGeneratorDriver decodes and executes requests for transfers from the RequestGenerator, and returns the results to the RequestGenerator.



Generation of a serial-in interrupt
App_SerialIn
App_ReceiveCommand
Serial data received
Break code?
No
Yes
App_CheckReceiveCommand
RequestGenerator activation command?
Yes
Disable EnumerationDriver
No
Initialize RequestGeneratorDriver
Processing command received?
Yes
Transfer-request generation state
To main loop

Figure 5.5 Serial Receive State (RequestGeneratorDriver Processing State)

5.6 Transfer-Request Generation State

The program enters this state on completion of connection processing after a device has been connected to the Root Hub and generation of the corresponding RootHubStatusChange interrupt. In this state, requests for transfers are received from the Driver layer by the HCD via the USBD layer, and the list in which EDs and TDs are linked is generated and sent to the HC. The flow chart for transfer requests is given in figure 5.6.

In this sample software, transfer requests from the RequestGenerator (on the PC) are received by the RequestGeneratorDriver by the serial-receive interrupt and transfer requests for the HCD are carried. In response to these transfer requests, the HCD generates and sends EDs and TDs to the HC.

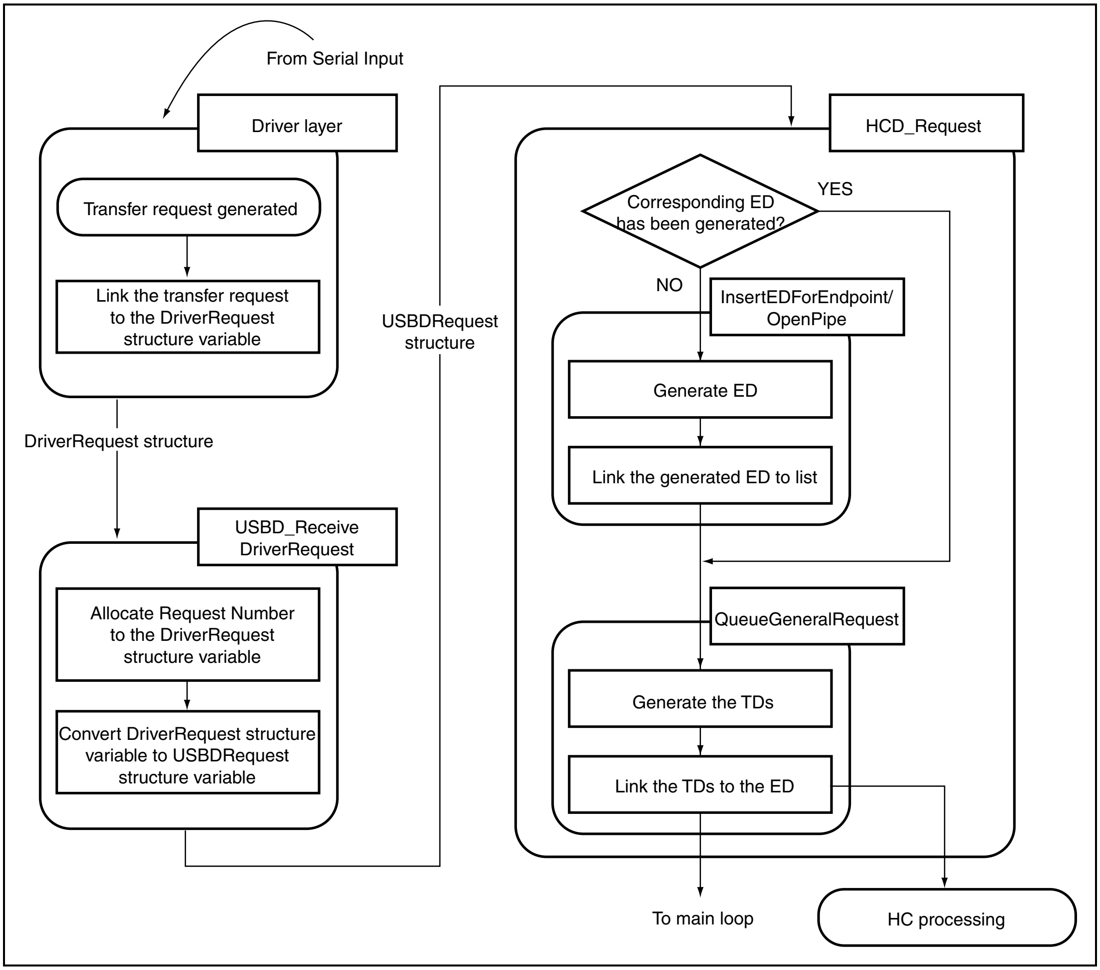
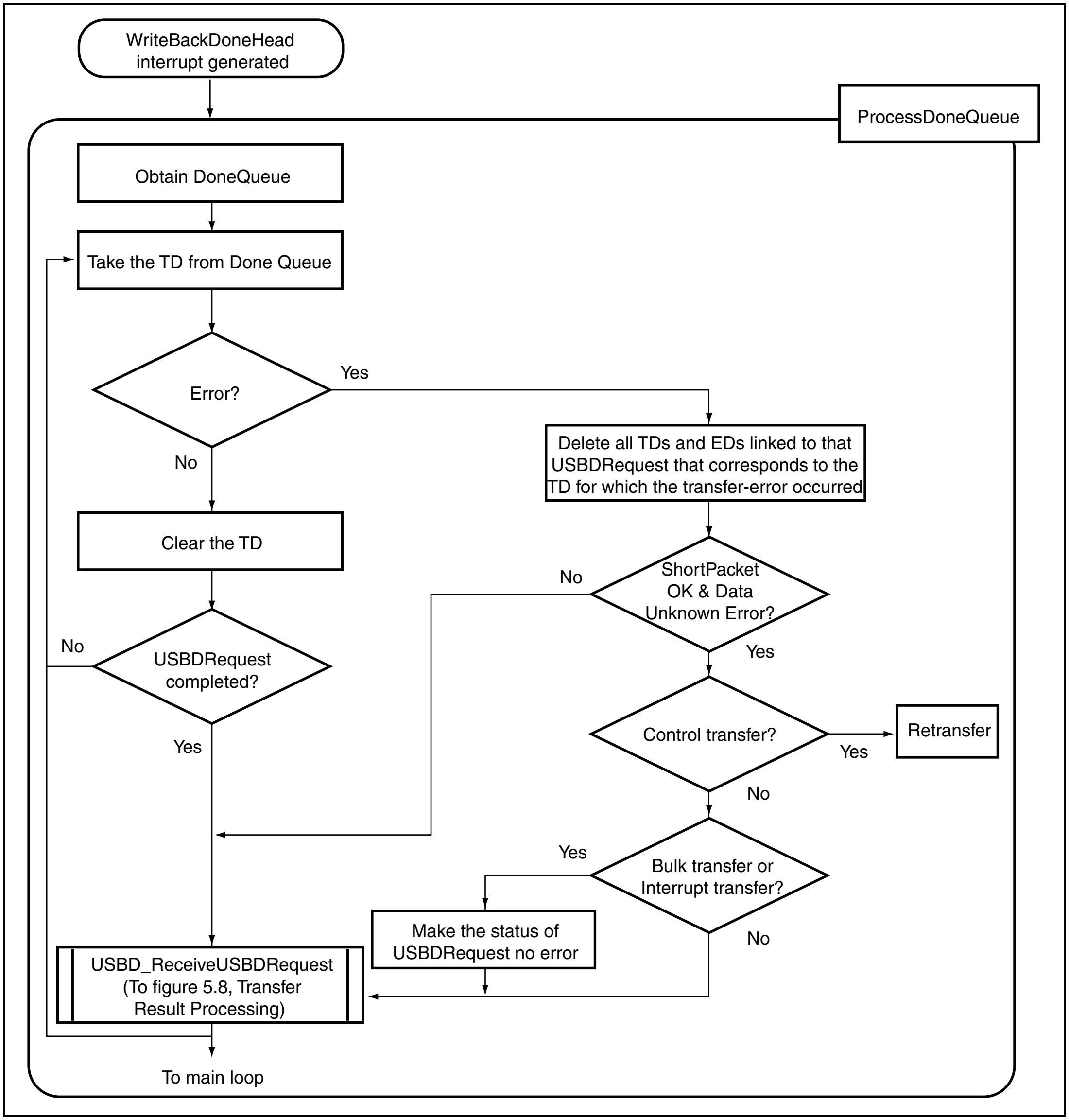

From Serial Input
Driver layer
Transfer request generated
Link the transfer request to the DriverRequest structure variable
DriverRequest structure
USBD_Receive DriverRequest
Allocate Request Number to the DriverRequest structure variable
Convert DriverRequest structure variable to USBDRequest structure variable
USBDRequest structure
HCD_Request
Corresponding ED has been generated?
YES
NO
InsertEDForEndpoint/ OpenPipe
Generate ED
Link the generated ED to list
QueueGeneralRequest
Generate the TDs
Link the TDs to the ED
To main loop
HC processing

Figure 5.6 Transfer-Request Processing

5.7 DoneQueue Processing State

The DoneQueue processing state is entered when a WriteBackDoneHead interrupt is generated while the program is in the steady state. In response to transfer requests, the HC generates data packets from received EDs and TDs and carries out USB communication in the way shown in figure 5.6. When processing of a set of received TDs has been completed, a WriteBackDoneHead interrupt is generated. The flow chart is given in figure 5.7.



WriteBackDoneHead interrupt generated
ProcessDoneQueue
Obtain DoneQueue
Take the TD from Done Queue
Error?
Yes
No
Delete all TDs and EDs linked to that USBDRequest that corresponds to the TD for which the transfer-error occurred
Clear the TD
No
ShortPacket OK & Data Unknown Error?
Yes
No
USBDRequest completed?
Yes
Control transfer?
Yes
Retransfer
No
Yes
Bulk transfer or Interrupt transfer?
No
Make the status of USBDRequest no error
USBD_ReceiveUSBDRequest (To figure 5.8, Transfer Result Processing)
To main loop

Figure 5.7 DoneQueue Processing

5.8 Transfer-Result Processing State

The program enters this state from a DoneQueue processing state which has been executed by the WriteBackDoneHead interrupt. A completed TD is received as a USBDRequest from the DoneQueue processing function, the result of the transfer is checked, and the result is returned to the Driver that requested the transfer. Details of the transfer-result processing state are given in figure 5.8.

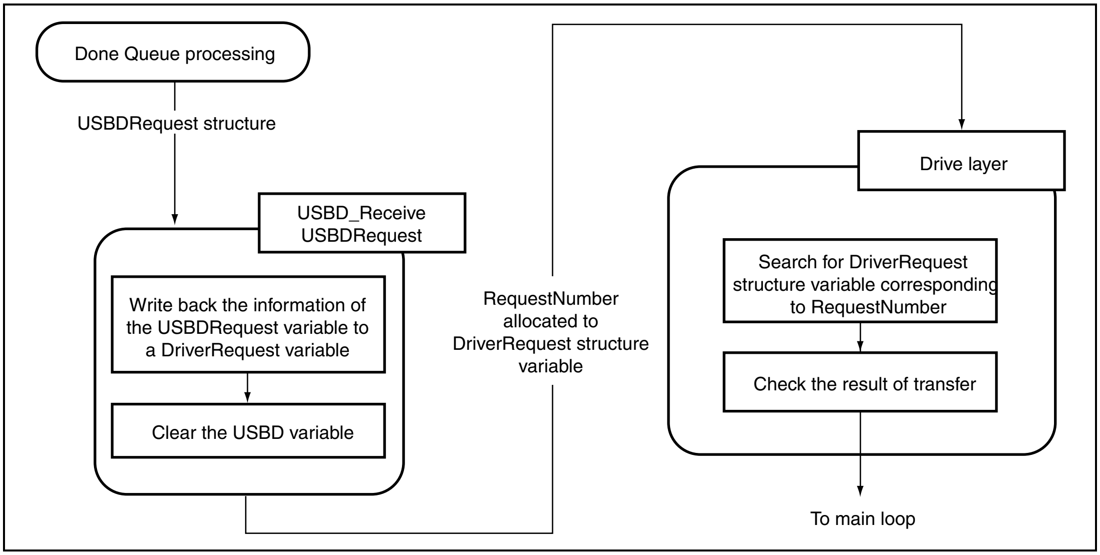

Done Queue processing
USBDRequest structure
USBD_Receive USBDRequest
Write back the information of the USBDRequest variable to a DriverRequest variable
Clear the USBD variable
RequestNumber allocated to DriverRequest structure variable
Drive layer
Search for DriverRequest structure variable corresponding to RequestNumber
Check the result of transfer
To main loop

Figure 5.8 Transfer-Result Processing

SH7727 USB Host Module Application Note

Publication Date: Rev.1.00, April 15, 2003
Published by: Sales Strategic Planning Div.
Renesas Technology Corp.
Edited by: Technical Documentation & Information Department
Renesas Kodaira Semiconductor Co., Ltd.

©2003 Renesas Technology Corp. All rights reserved. Printed in Japan.

SH7727 Group USB Host Module Application Note

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ05B0015-0100Z