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.

RENESAS

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.

H8SX Family

Mass Storage Class Demonstration

Introduction

This application note introduces the USB Mass Storage class and shows an example of how to configure the USB block on the H8SX/1664 and use the microcontroller as a Mass Storage Class Device. This document refers to the RSK H8SX/1664 USB kit and specifically to the included MSC application example.

Target Device

H8SX1664 (RSKH8SX1664)

Contents

1. Overview

The Renesas USB Stack and MSC sample program is an example of how to create a Mass Storage class device and enumerate on a PC. This class of device is more commonly known as the MSC and examples include memory sticks and other USB based storage devices.

2. Introduction to USB

The USB (Universal Serial Bus) is an interface and a protocol that allows a single host computer to communicate with a variety of peripheral devices. The USB 2.0 spec defines this interface. Although it is dependant on the application most USB projects will require a host side interface app and the device firmware. Every USB communication is between a host and a device, where the host controls the bus and initiates communication all the time, except in case of devices with the remote-wakeup feature (USB On-The-Go allows for devices to negotiate for the role of host and thus bus control). In comparison with other interfaces, USB offers a host of advantages which include, automatic configuration (enumeration), minimum IRQ lines used, hot pluggable, low cost, low power consumption, speed and reliability. Depending on the application, the developers can chose one of four USB transfer types for his project; Control, Bulk, Interrupt and Isochronous. These classifications are based on frequency of transfer, amount of data to be transferred and the kind of data being transferred.

In USB terminology, individual devices are referred to as *functions*, which are linked in series through *hubs*. The hubs are special-purpose devices that are not considered functions. There always exists one hub known as the root hub, which is attached directly to the host controller.

Endpoints:

Functions and hubs have associated *pipes* (logical channels). Pipes are connections from the host controller to a logical entity on the device named an *endpoint.* The end point thus serves as a data buffer; typically it is a block of data memory or a register in the device; each endpoint can transfer data in one direction only (except endpoint 0), either into or out of the device/function, thus making pipes unidirectional. Every device has endpoint zero configured for bidirectional control transfer. The number of available endpoints and supported transfer types vary with each device. The different kinds of endpoints are Bulk, Control, Interrupt and Isochronous. Since endpoints are unidirectional, they will be followed by an “in” or “out” specification (e.g. Bulk-In).

Device Information:

To identify itself as a USB device and to conform to the spec for a certain class, a device needs to have in its firmware certain elements of information that the host can access in order to successfully enumerate and then communicate with the device. These elements are broadly known as Descriptors and are further classified into:

a. Device descriptor: Information such as the device class, the device sub-class, number of configurations, max packet size and other info about the device as a whole are present in this descriptor.

b. Configuration Descriptor: Information about the number of interface supported and power consumption is provided in this descriptor; most devices usually support only a single configuration, but multiple configurations are allowed.

c. Interface descriptor: Each interface on the device has its own descriptor and subordinate descriptors (descriptors for endpoints used in the interface.

d. Endpoint Descriptors (at least 2): Endpoint descriptors contain information about the endpoints to be used in that interface. This includes maximum packet size, polling rate, endpoint type (Interrupt, Bulk, Control or Isochronous) and endpoint direction (in or out).

e. Report Descriptor: This descriptor is required only in case of HID class devices and contains information on the format of data being transmitted.

f. String Descriptor: Human readable information i.e. messages to be displayed on device enumeration etc are stored in this descriptor. It is optional.

Enumeration:

Before the host can begin using a USB device, it has to learn about the device capabilities, resources and other features in order to assign a device driver. The procedure by which a device identifies itself (including all resources and capabilities available) to the host is known as enumeration. When a function or hub is attached to the host controller through any hub on the bus (including the root hub), it is given a unique 7 bit address on the bus by the host controller. On any USB system all communication is initiated by the host. The host uses a specific set of requests to retrieve required information from the device. These requests can be classified as standard requests and class-specific requests. There are eleven standard requests in the USB. An example of this is Get_Descriptor.

The Get_Descriptor command is used to retrieve descriptors. The Set_Descriptor request lets the host change descriptors in the device.The host controller then polls the bus for traffic, usually in a round-robin fashion, so no function can transfer any data on the bus without explicit request from the host controller.

Frames:

USB establishes a 1 millisecond time base called a frame on a full-/low-speed bus. A frame can contain several transactions. Each transfer type defines what transactions are allowed within a frame for an endpoint. Isochronous and interrupt endpoints are given opportunities to access the bus every N frames. This information is set in the "*bInterval*" or Polling Interval field in the endpoint descriptor. For Bulk endpoints, this filed is not applicable.

2.1 Transfer types

There are four transfer types defined by the USB specification:

2.1.1 Control transfers

Control transfers are facilitated by the device control endpoint (endpoint zero). The host uses control transfers to configure the device, request device information and other settings. Control transfers are different from other transfers in that they have stages; typically three stages. The host sends a request in the Setup stage; the Data stage is used by the host/device to send data (not all requests have this stage) and the device reports the status information in the Status stage. Control transfers may also be used to send vendor specific requests.

2.1.2 Interrupt Transfers

Interrupt transfers are typically non-periodic communication requiring bounded latency. An Interrupt request is queued by the device until the host polls the USB device asking for data. These transfers require an Interrupt-In endpoint on the device.

2.1.3 Bulk transfers:

Bulk transfers can be used for large bursty data. It is ideal in situations where the transfer rate is not critical. Data transfer using bulk transfers are very fast if the bus is idle; if the bus is busy, the transfers are delayed. This type of transfer is supported only by Full-Speed and High-Speed devices and require a Bulk-In endpoint and a Bulk-Out endpoint for data to and from the PC respectively.

2.1.4 Isochronous transfers:

Isochronous transfers occur continuously and periodically. They typically contain time sensitive information, such as an audio or video stream. There is no retry or guarantee of delivery, although for the kind of application it is designed for, loss of a packet or frame does not cause critical issues with application performance e.g. audio or video glitches too small to be noticed by the user. This transfer mode is supported only by Full and High speed USB devices.

Refer to the Universal Serial Bus Specification [b] on usb.org for more details

2.2 Storage media and the FAT File System:

There are many different kinds of storage media and they can be broadly grouped into hard drives and general flash memory from an embedded storage media point of view. While hard drives are preferred for larger capacities, flash memory is the media of choice in smaller capacity applications. While using flash memory, device firmware must implement "wear-leveling" in order to extend the life of the memory chip.

2.2.1 FAT file system:

The FAT (File Allocation Table) file system was created (and patented) by Microsoft for disk management purposes. The system essentially keeps a table of file names, location of contents, usable areas of memory etc. This file system is supported by virtually every OS, and hence is an ideal format in which to store data. The FAT file system is used extensively in embedded data storage systems that have to directly interface with an OS.

File system support is not a requirement for Mass Storage class devices but, dependant on the application, the developer may choose to implement this as well. If the firmware supports Logical Block Addressing, then any required data can be readily accessed; however, if there was need for this storage media to be useful in other environments outside that controlled by the device firmware (e.g. if the storage media were to be used in a Windows environment), then additional support/interface as required by that system will have to be provided. In most cases a file system provides the requisite interface.

For more information and articles on Embedded File Systems and Storage media refer to エラー! 参照元が見つかりません。

2.3 The Mass Storage class

USB devices are categorized in to various classes based on common behavior and protocols for devices that serve similar functions.

2.3.1 Mass Storage Class Requirements:

In addition to supporting standard USB requirements, a mass storage class device must conform to the following as well:

1. Interface Descriptor with a class code of 08h.
2. A mass storage interface with Bulk-in and Bulk-out end points and endpoint zero for control transfer.
3. Storage media (hard drive, flash-memory cards, CD/DVD, MMC cards etc).
4. Firmware support for logical block addressing (LBA) of the storage media.
5. Firmware support for Mass Storage class requests.
6. Support for one or more industry-standard command-block sets to exchange control, data and status information. (e.g. SCSI)

2.3.2 The SCSI command Interface

The SCSI (Small Computer Standard Interface) standard contains definitions of command sets of specific peripheral device types and using it to transfer data between computers and the specific peripheral device. However the presence of “unknown” as one of the device types, theoretically allows SCSI to be used to interface with practically any device.

Devices are classified based on the type of SCSI commands they support; the SCSI Block Commands (SBC) document specifies commands used by flash drives and other direct access block devices. In SCSI terminology, communication takes place between an initiator and a target. The initiator sends a command to the target which then responds. SCSI commands are sent in a Command Descriptor Block (CDB). The CDB consists of a one byte operation code followed by five or more bytes containing command-specific parameters.

At the end of the command sequence the target returns a Status Code byte which is usually 00h for success, 02h for an error (called a Check Condition), or 08h for busy. When the target returns a Check Condition in response to a command, the initiator usually then issues a SCSI Request Sense command in order to obtain the status.

The total number of commands defined by the protocol are about 60; but how many are implemented is dependant on the application. At a minimum, dependant on the application, the following primary commands have to be implemented:

1. INQUIRY: This command is used by the host to request information about the device. The response from the device is in the form of a structure (at least 36 bytes in length) where information such as peripheral device type, vendor identification number, and other information about the devices capabilities is sent to the host. The response structure is sent in the data-transfer phase of the request.
2. READ CAPACITY: Tells the host the media sector information.
3. READ (10): Reads the specified sector volume data from a specified sector.
4. REQUEST SENSE: The host requests sense data via this command. In the event that the device experiences a problem, status information is filled into a structure; this data is called sense data.
5. TEST UNIT READY: This command is used to determine if the storage device is ready for use.
6. WRITE (10). Write the specified sector volume data to a specified sector.

The following additional commands have been implemented by the sample program as well.

1. PREVENT/ALLOW MEDIUM REMOVAL: This command requests the device to prevent or allows removal of the storage media from the device. A 2 bit field is used to set/unset the option; support is optional for this command.
2. VERIFY: Verifies if the data in a medium can be accessed.
3. STOP/START UNIT: Controls installation and removal of media.

4. MODE SENSE (6): Tells the host the drive status.

3. Program Description

Below is a layout of different layers of the USB stack

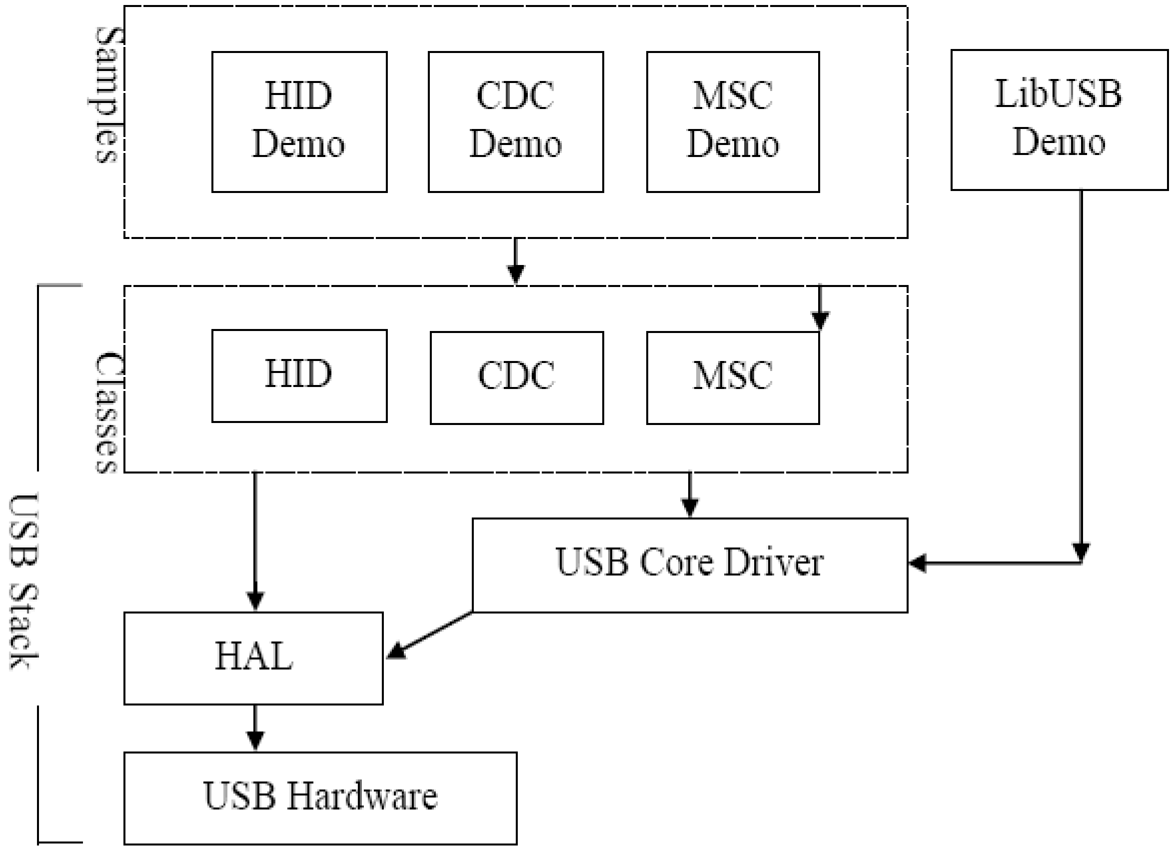

Figure 1: Sample program layout

The USB stack is split into three layers, namely the Hardware Abstraction Layer, the USB Core Layer and the USB Class layer.

The Hardware Abstraction layer interfaces directly with the hardware and provides the higher layers with a uniform hardware independent API. Thus when moving between devices, only the functions in this layer have to be modified since the higher layers are abstracted to the hardware features and operation. This implementation is maintained across all the versions of this program developed for different Renesas devices. The Hal currently supports the following transfer modes

Control (Setup, Data IN/OUT, Status)

Bulk (IN and OUT)

Interrupt (IN)

Isochronous transfer modes are not currently supported.

The USB Core layer implements protocol specific commands and response decoding. This layer uses the HAL layer for actual data transfer and reception. For example the USB Core layer handles the Get_Descriptor requests from the host *This layer can be further expanded to support additional USB host requests* . The HAL layer notifies the USB core layer of USB events via call back functions.

The individual class layer implements functions that are specific to that particular class. In case of the CDC class device, the application support the class requests required for communication over hyper-terminal namely (GET_LINE_CODING, SET_LINE_CODING and SET_CONTROL_STATE)

3.1 Configuring the device for different USB events

The following section talks about the operation of the USB clock and how to configure the registers to respond to different USB events

3.1.1 Interrupt Vectors

This device has a total of 5 USB interrupt vectors.

- Vector 232; USBINTN0
- Vector 233; USBINTN1
- Vector 234; USBINTN2
- Vector 235; USBINTN3
- Vector 238; RESUME

All USB events are channeled through these interrupt vector. In the ISR, the user has to check the flag register to determine the actual cause of the interrupt. USBINTN0 and USBINTN1 are only required when combining USB and DMAC transfers. In the included sample program, this feature is not used.

The RESUME signal is used to wake the device out of software stand-by mode. This is useful in applications where the device has to go into low power and will be waken up remotely when required.

All of the remaining interrupts are channeled though USBINTN2 and USBINTN3.

3.1.2 Initialization

This section lists the initialization sequence for the USB block

(a) **Pin Configuration**

This Device has a total of 5 USB pins all of which are dedicated for USB operation and hence do not need to be configured as inputs or outputs.

- VBUS
- DrVSS
- USD-
- USD+
- DrVCC

During initialization, before enabling the pull-up for USBD+, the port is forced low. This is to ensure that if the USB device is already plugged in before the software runs, then the forced pull down and release will start the enumeration process.

(b) **Module Enable**

By default, all peripheral modules including USB are disabled on the H8SX/1664.

To access any USB registers, the module needs to be enabled by setting `MSTP._CRC.BIT._USB = 0`

(c) **Clock Configuration**

Once the initialization is done, the device has to be configured to respond to different USB events. The following sections explain the configuration required to respond to each of the different USB events.

3.1.3 Cable Connection/Disconnection

When properly configured, this LSI generates an interrupt to the CPU when the USB cable is plugged in or disconnected. To enable this interrupt, the VBUSF bit in IER1 has to be set (`USB.IER1.BIT.VBUSF = 1`).

Since this interrupt is usually followed by the Bus Reset interrupt, ensure that this is enabled as well by setting the BRST bit in IER0 (`USB.IER1.BIT.VBUSF = 1`).

Since both of these interrupts are routed via USBINTN2, in the interrupt handler check the flag registers to determine the interrupt source (`USB.IFR1.BIT.VBUSF` and USB.IFR0.BIT.BRST). In case of the VBUS interrupt, clear the flag register, notify the software layer, and if the cable is connected, initialize the data elements used by the software stack.

When the Bus Reset interrupt occurs after this, make sure to initialize the block by clearing all the FIFOs and stall states. This is done by

```
USB.FCLR.BYTE = 0x73;    /* Clear FIFOS    */
USB.EPSTL.BYTE = 0x00;   /* Clear all stalls */
```

3.1.4 Control Transfer

Once the Bus Reset has occurred, the enumeration process begins. The details of the enumeration process and the control transfer used for this process are described in section 2.1.1.

(a) **Setup Stage**

When the host sends the SETUP command, if the SETUPTS bit in IER0 has been set, the device will issue an interrupt request. The source can be determined by interrogating the SETUPTS bit in the flag register IFR0. Since the reception of the Setup command packet indicates the start of a new command sequence, it is necessary to flush out the FIFOs to prevent the software from reading data packets from a prior command. Once this is done, read all the 8 byts of the Setup packet and notify the higher layer for processing. Then set the hardware bit indicating that the 8 bytes have been read. This is done as shown below:

```
  /*Clear EP0i and EP0o FIFOs*/
USB.FCLR.BYTE = 0x03;/* EP0oCLR = 1 and EP0iCLR = 1 */
  /*Read 8 bytes of data from EP0 into tempory buffer*/
  for(index = 0; index < USB_SETUP_PACKET_SIZE; index++)
  {
   SetupCmdBuffer[index] = USB.EPDR0s;
  }
  /*Set EP0 Status Read Complete*/
  USB.TRG.BIT.EP0sRDFN = 1;
```

The H8S/1664 USB block automatically decodes the following Setup commands. These commands do not generate an interrupt request to the CPU and all three stages including command, data and status stages are automatically processed by the hardware.

- Clear Feature
- Get Configuration
- Get Interface
- Get Status
- Set Address
- Set Configuration
- Set Feature
- Set Interface

The following commands require firmware support to proceed with the subsequent stages. An interrupt request is generated when these commands are received.

- Get Descriptor
- Class/Vendor commands
- Set Descriptor
- Sync Frame

If the application does not support a particular command, the slave can respond by setting the Control endpoint to a “stall” state. This will cause the block to send a “stall” packet to the host indicating that the particular command is not supported. The stall state can also be used by other endpoints to let the host know that it is not yet ready to respond to further host requests. These two stall states are known as *protocol stall* and *commanded stall* respectively.

(b) **Control-Data Stage**

As mentioned in section 2.1.1, the command stage is followed by an optional data stage and a mandatory status stage. Depending on the command sent during the Setup stage, there may be a Data-IN or Data-OUT stage. An important thing to remember is that the subsequent *Status stage has a direction opposite to the preceding Data stage*. The interrupt sources are configured based on this.

To handle Data-IN, enable the EP0iTS and EP0oTS interrupts. The data is sent to the host via the EPDR0i register. If there are more packets still to send, the deice waits for either the EP0iTS interrupt (indicating that the host is asking for the rest of the data) or the EP0oTS interrupt (indicating that the host is done and now sending the Status). Once all the packets have been sent, set the EP0iPKTE bit to indicate to the host that all the data has been sent.

The EP0iTR (Request for information by the host) interrupt will not be used in this case since the host requests for further data packets by means of the EP0iTS interrupt. And since the subsequent Status stage will consist of *receiving* information from the host, the EP0iTR is not required.

To handle Data-OUT, enable the EP0iTS, EP0oTS and the EP0iTR interrupts. The data from the host is received on the EPDR0o register. Once the data has been read, set the EP0oRDFN to indicate to the host that it has been read so that the next packet can be sent by the host. Once all the data has been read, the host attempts to read the Status from the device. This will cause the EP0iTR interrupt to trigger.

(c) **Control-Status Stage**

The direction of the Status stage is opposite to that of the preceding data stage (if there was one) or the Setup stage. If the host has requested for the status by means of the EP0iTR interrupt, then the device will send one of the following packets depending on the outcome of the process: ACK, NAK or STALL. The ACK is generated by sending a zero-byte packet to the host. This is done by setting the EP0iPKTE bit without writing anything to the EPDR0i register.

Conversely, if the host is the source of the Status stage, then the EP0oTS interrupt will trigger on receiving an ACK.

3.1.5 Bulk-In Transfer

On the H8SX/1664, endpoint 2 is the Bulk-IN endpoint. Interrupts requests for this endpoint are enabled by setting the EP2TR bit.

When the host wants to receive over this endpoint, it sends a request which triggers the EP2TR interrupt. In the interrupt handler, the application writes out a packet to the EPDR2 register byte by byte and then sets the EP2PKTE bit so that the data is sent to the host. Once the data is successfully sent, the interrupt will be retriggered. The application will then write out any more remaining packets and repeat the process until all the data is sent to the host.

If at any point the EP2PKTE bit is set without writing a full packet worth of data (64 bytes for this device) to EPDR2, then the host will consider this as an indication that the device has finished sending all the available data.

If then total data to be sent is a multiple of 64, then to end the transmission, send an empty packet by setting the EP2PKTE bit without writing any data to EPDR2.

3.1.6 Bulk-Out Transfer

On the H8SX/1664, endpoint 1 is the Bulk-OUT endpoint. Interrupts requests for this endpoint are enabled by setting the EP1FULL bit.

When the host sends data over this endpoint, the EP1FULL interrupt is triggered. The total size of the data sent by the host is in the EPSZ1 register and the actual data in the EPDR1 register. Once the application reads out all the data from the register, it sets the EP1RDFN bit to indicate to the host that all the data has been read out.

3.1.7 Interrupt-In Transfer

On the H8SX/1664, endpoint 3 is the Bulk-OUT endpoint. Interrupts requests for this endpoint are enabled by setting the EP3TR the EP3TS bits.

To send data to the host over this endpoint, the application writes a packet (or less) worth of data to the EPDR3 register one byte at a time and then sets the EP3PKTE bit to send the data. If a full packet was sent, the EP3TS interrupt will be generated again indicating that the data was sent and the host is requesting the rest of the data. The procedure is repeated until all the data is sent or until the EP3PKTE bit is set without writing a full packet worth of data to the EP3DR register.

If the host attempts to read the EP3DR register and finds no data in the buffer, the EP3TR interrupt will be triggered. This case is not handled in the provided sample code and hence this interrupt is disabled.

4. Using the software in your application

The USB software stack provided is mostly self sufficient and provides adequate functionality for most applications.

The sample program uses the internal RAM for to demonstrate the Mass Storage functionality. This means that a power cycle will cause any new data to be lost. The program can be reconfigured to use and external memory quite easily. To do this, locate the g_RAM_DATA array to the external memory area. Other associated macros also have to be redefined depending on the size of the external memory. Refer to the ram_disk.c file on the sample project for more details.

Reference

a. H8SX/1664 group manual. Document number: REJ09B0294

b. Universal Serial Bus Specification Revision 2.0

c. "USB Complete: Everything You Need to Develop Custom USB Peripherals" by Jan Axelson.

Website and Support

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

Inquiries
http://www.renesas.com/inquiry
csc@renesas.com

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Nov.10.09	—	First edition issued

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, 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 products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas 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 applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.