Renesas USB MCU

USB PMSC with Local File System and Flash Storage

Introduction

This document is an extension of the PMSC application note R01AN0514EJ0400, for USB Peripheral Mass Storage class using Renesas. Added to this version is the ability of the RX platform to locally access (r/w) files on the board using its own file system library, and also the ability to use the RSK63N’s serial flash chip. It is an add-on for the RSK63N, but the implementation can be applied to any of the devices in the RX600.

Target Device

RX63N Group. This program can be used with other RX600 Series microcontrollers that have the same USB module as the above target devices. When using this code in an end product or other application, its operation must be tested and evaluated thoroughly.

This program has been evaluated using the Renesas Starter Kit board for RX63N.

The default storage for demo is RAM. See 9 for how to use the board SPI flash.

The storage media block size is set to 4 kB which only works with Win7 or later. This can be changed to standard 512 B blocks in ram_disk.h and ff_conf.h.
Content

1. Overview .................................................................................................................................................. 4
1.1 Functions and Features ............................................................................................................................. 4
1.2 Related Documents ................................................................................................................................. 4
1.3 Terms and Abbreviations ....................................................................................................................... 5

2. Using E2studio .......................................................................................................................................... 6
2.1 Import the Project ................................................................................................................................... 6

3. Software Configuration ............................................................................................................................ 7
3.1 Module Configuration ............................................................................................................................... 7
3.2 File Structure ......................................................................................................................................... 9
3.3 Setting Up the Code For Your Board .................................................................................................... 9
3.4 System Resources .................................................................................................................................. 10

4. Peripheral MSC Firmware Application Functionality (APL) ............................................................... 13
4.1 Media R/W functionality as seen from USB Host .................................................................................. 13
4.2 Media R/W functionality as seen from MCU application layer ......................................................... 16

5. Peripheral Device Class Driver (PDCD) .................................................................................................. 17
5.1 Basic Functions ..................................................................................................................................... 17
5.2 BOT Protocol Overview ......................................................................................................................... 17

6. USB Peripheral Mass Storage Class Driver (PMSCD) ............................................................................. 24
6.1 Basic Functions ..................................................................................................................................... 24
6.2 List of API Functions .............................................................................................................................. 24
6.3 Class Driver Registration ....................................................................................................................... 30
6.4 User Definition Tables .......................................................................................................................... 30

7. Peripheral Mass Storage Device Driver (PMSDD) .................................................................................. 31
7.1 PMSDD Storage Command Structure ............................................................................................... 32
7.2 List of PMSDD Functions ...................................................................................................................... 33
7.3 PMSDD Task Description ....................................................................................................................... 34

8. Target File System ................................................................................................................................... 35
8.1 User Interactive Board Demo ............................................................................................................... 35
8.2 Long Filenames ..................................................................................................................................... 35

9. Media Driver Interface ............................................................................................................................. 36
9.1 Overview of Media Driver API Functions ............................................................................................. 36
9.2 Selecting Media Driver .......................................................................................................................... 36
9.3 Changing (adding) Storage Media ...................................................................................................... 37

10. The RAM Media Driver ......................................................................................................................... 39
10.1 RAM-disk Media Driver Default FAT Format .................................................................................... 39
10.2 RAM-disk Global Area Variables ....................................................................................................... 39
10.3 Constant Definitions .............................................................................................................................. 40
10.4 Operation Overview ............................................................................................................................. 41
1. Overview

This document is a manual describing use of the USB Peripheral Mass Storage class driver for Renesas.

1.1 Functions and Features

The USB Peripheral Mass Storage Class driver comprises a USB Mass Storage class bulk-only transport (BOT) protocol. When combined with a USB peripheral control driver and storage device driver, it enables communication with a USB host as a BOT-compatible storage device.

1.2 Related Documents

1. USB Revision 2.0 Specification
2. USB Mass Storage Class Specification Overview Revision 1.1
3. USB Mass Storage Class Bulk-Only Transport Revision 1.0, “BOT” protocol [http://www.usb.org/developers/docs/]
7. R8A66597 Data Sheet. (Document No.REJ03F0229RJJ03F.)
8. Renesas USB Basic Firmware Application Note. (Document No. R01AN0512EJ.)
9. RX600 Series USB PMSC with Local File System and Flash Storage installation guide. (Document No. R01AN0529EJ.)
10. Block Access Media Driver API. (Document No. R01AN1443EU.)
11. Block Storage Driver for Serial Flash via RSPI. (Document No. R01AN1466EU.)

Renesas Electronics Website [http://www.renesas.com/]
USB Devices Page [http://www.renesas.com/prod/usb/]
1.3 Terms and Abbreviations

<table>
<thead>
<tr>
<th>Term</th>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI</td>
<td></td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>APL</td>
<td></td>
<td>Application program</td>
</tr>
<tr>
<td>ASSP (asp)</td>
<td></td>
<td>Application Specific Standard Produce</td>
</tr>
<tr>
<td>BOT</td>
<td></td>
<td>USB mass storage class bulk only transport. See “Universal Serial Bus Mass Storage Class Bulk-Only Transport” at USB Implementers Forum.</td>
</tr>
<tr>
<td>cost</td>
<td></td>
<td>Prefix of Function and File for Host &amp; Peripheral USB-Basic-F/W</td>
</tr>
<tr>
<td>DDI</td>
<td></td>
<td>Device driver interface, or PMSDD API.</td>
</tr>
<tr>
<td>HEW</td>
<td></td>
<td>High-performance Embedded Workshop</td>
</tr>
<tr>
<td>H/W</td>
<td></td>
<td>Renesas USB device</td>
</tr>
<tr>
<td>ITRON, uITRON</td>
<td></td>
<td>Industrial The Real-time Operating system Nucleus</td>
</tr>
<tr>
<td>non-OS</td>
<td></td>
<td>USB basic firmware for OS less system</td>
</tr>
<tr>
<td>PCD</td>
<td></td>
<td>Peripheral control driver of USB-Basic-F/W</td>
</tr>
<tr>
<td>PCDC</td>
<td></td>
<td>Communications Devices Class for peripheral</td>
</tr>
<tr>
<td>PCI</td>
<td></td>
<td>PCD interface</td>
</tr>
<tr>
<td>PMSCD</td>
<td></td>
<td>Peripheral mass storage USB class driver (PMSCF + PCI + DDI)</td>
</tr>
<tr>
<td>PMSCF</td>
<td></td>
<td>Peripheral mass storage class function</td>
</tr>
<tr>
<td>PMSDD</td>
<td></td>
<td>Peripheral mass storage device driver (sample ATAPI driver)</td>
</tr>
<tr>
<td>PP</td>
<td></td>
<td>Pre-processed definition</td>
</tr>
<tr>
<td>pstd</td>
<td></td>
<td>Prefix of Function and File for Peripheral USB-Basic-F/W</td>
</tr>
<tr>
<td>R8A66597</td>
<td></td>
<td>Renesas Hi-Speed USB2.0 ASSP R8A66597 board (Use in combination with RX62N-RSK.)</td>
</tr>
<tr>
<td>RTOS</td>
<td></td>
<td>USB basic firmware for uITRON system</td>
</tr>
<tr>
<td>RX62N-RSK</td>
<td></td>
<td>Renesas Starter Kits for RX62N</td>
</tr>
<tr>
<td>RX630-RSK</td>
<td></td>
<td>Renesas Starter Kits for RX630</td>
</tr>
<tr>
<td>RX63N-RSK</td>
<td></td>
<td>Renesas Starter Kits for RX63N</td>
</tr>
<tr>
<td>Scheduler</td>
<td></td>
<td>Used to schedule functions, like a simplified OS.</td>
</tr>
<tr>
<td>Scheduler Macro</td>
<td></td>
<td>Used to call a scheduler function (non-OS)</td>
</tr>
<tr>
<td>Task</td>
<td></td>
<td>Processing unit</td>
</tr>
<tr>
<td>USB</td>
<td></td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>USB-Basic-F/W</td>
<td></td>
<td>USB basic firmware for Renesas USB device (non-OS &amp; RTOS)</td>
</tr>
</tbody>
</table>

(Note 1) When RX62N-RSK is used in conjunction with the R8A66597, SW1 is allocated to a port used by an interrupt. Therefore, do not use SW1.
2. Using E2studio

2.1 Import the Project

With e² studio, a project should not be moved, but first exported then imported. Follow the directions below.

2.1.1 New Workspace
1. Create empty folder where you want workspace.
2. Start E2S, and point to that folder as E2S asks what workspace to open.
3. Click Workbench icon (bottom right in blue intro-screen).
4. Continue with next step below.

2.1.2 Existing Workspace
1. Select Import.
2. Select General => Existing Projects into workspace. ("Create new projects from an archive file or directory.")
3. Browse to
   - Archive zip-file, or
   - Root directory

For both, make sure checkbox "Copy project to workspace" is checked.

You have now imported this project into the workspace. You can go ahead and import other projects into the same workspace... Follow below for importing to existing workspace.

Build with Ctrl+B.
3. **Software Configuration**

3.1 **Module Configuration**

As shown in Figure 3-1, PDCD comprises two layers: PMSCD and PMSDD.

PMSCD comprises three layers: PCD API (PCI), PMSDD API (DDI), and BOT protocol control and data sends and receives (PMSCF).

In the case of RTOS, PMSCD and PMSDD run on top of uITRON as tasks.

PMSCD uses the BOT protocol to communicate with the host via PCD.

PMSDD analyzes and executes storage commands received from PMSCD. PMSDD accesses media data via the media driver.

Figure 3-1 shows the configuration of the modules.

![Figure 3-1 Software Configuration Diagram](image)

3.1.1 **PDCD**

As shown in Figure 3-1, PDCD incorporates PMSDD and PMSCD. PDCD takes care of class requests from the USB host, and responds to USB host storage commands.

Table 3-1 provides an overview of the parts of PDCD, as well as PCD and the media driver. The media driver is implemented as an interchangeable block media type storage driver.

3.1.2 **PMSCD**

PMSCD comprises three layers: PMSCF, which performs BOT protocol control and data transmission/reception; DDI for interfacing with PMSDD; and a group of functions (PCI) for interfacing with PCD. The main functions of these layers are as follows.

1. **PMSCF:**
   - USB mass storage class BOT protocol control
   - CBW analysis, data transmission/reception, and CSW creation in coordination with PMSDD/PCD
   - Responding to class requests (MassStorageReset, GetMaxLUN)
2. **PCI:**
   - Processing of tasks, message boxes, and memory pools during configuration and detach
Receiving class requests
Clearing STALL states and setting related callback functions
Setting structures and callback functions for PCD transmit/receive data

3. DDI:
Driver registration.

Transferring data information and execution results during PMSDD execution - the ATAPI command result callback is executed.

### Table 3-1 Overview of Modules

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Module</th>
<th>Description</th>
<th>Reference folder/file</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB Basic FW</td>
<td>PCD</td>
<td>USB peripheral hardware control driver.</td>
<td>\usbstdfw</td>
<td>See “Renesas USB Device USB Basic Firmware Application note” Document no. R01AN0512.</td>
</tr>
<tr>
<td>Ch. 0</td>
<td>PCI</td>
<td>PMSCF-PCD interface functions.</td>
<td>\MSCFW\PMSC\r_usb_pmsc_pci.c</td>
<td></td>
</tr>
<tr>
<td>Ch. 0</td>
<td>PMSCF</td>
<td>Core component of PMSCD. Controls BOT protocol data and responds to USB class requests. Also transfers storage commands and data to and from storage (PMSDD).</td>
<td>\MSCFW\PMSC\r_usb_pmsc_request.c \r_usb_pmsc_driver.c</td>
<td></td>
</tr>
<tr>
<td>DDI</td>
<td>PMSDD-PMSCF interface: Driver registration and ATAPI result callback.</td>
<td>\MSCFW\PMSC\r_usb_pmsc_ddi.c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ch. 0</td>
<td>PMSDD</td>
<td>Peripheral mass storage media driver. It processes storage commands from PMSCD and accesses the media via the block media driver below. (To be modified to match the memory device.)</td>
<td>\MSCFW\MEDIA\r_usb_atapi_driver.c</td>
<td></td>
</tr>
<tr>
<td>Ch. 0</td>
<td>Block Media Driver</td>
<td>Block media storage driver. For either RAM or SPI flash data storage.</td>
<td>\MSCFW\MEDIA\r_usb_atapi_memory.c</td>
<td></td>
</tr>
</tbody>
</table>
3.2 File Structure
The following shows the folder structure for the files provided in this device class.

This source code does not use the ANSI-C type IO interface.

```
+---WorkSpace
  (USB-BASIC-FW) [Common USB code that is used by all USB firmware]
  +---USBSTDFW USB Basic FW
  +---include USB Basic FW Common header file
  +---nonOS / RTOS

(ANSI-C File I/O System Calls)
  +---ANSI [open(), close(), read(), write(), etc of the USB class driver]

(Class F/W)
  +---MSCFW
  |   +---include Peripheral MSC header file
  |   +---PMSC Peripheral MSC driver

(Sample Code) [user application]
  +---SmplMain
  |   +---APL Sample Application

(HW Setting) [Hardware access layer; to manipulate the MCU’s USB register]
  +---HwResourceForUSB Hardware resource for RX63N/RX631 Group
  (media_driver) File system and storage media
  +---elm ELM FAT32 file system (long filenames and 4 kB storage media possible)
  +---r_hsp FIT BSP
  +---ram_disk Driver when using RAM instaed of serial flash (default)
  +---r_rspi_rx FIT RSPI
  +---r_spi_flash FIT Flash
  +---r_switches FIT Switches package
  +---spi_flash_block_if Block storage interface

(uITRON) [uItron OS code]
  +---RI600_4 ITRON Folder (not included in non-OS version)
```

3.3 Setting Up the Code For Your Board
In this chapter we will set up the code for the MCU and board you want to use.

3.3.1 Endian
Set target for Little Endian as this is the default setup of the SW.

3.3.2 Add the Right “HW Resource” Code
Replace the existing code in folder HwResourceForUSB with the content of the relevant folder HwResourceForUSB_devicename, as mentioned in 3.2.

3.3.3 Selecting Platform
Open file platform.h and uncomment one include file corresponding to your board
/* RSKRX63N */
#include "./board/rskrx63n/r_bsp.h"

### 3.3.4 Selecting Build Configuration

This version only supports RSK63N.

![High-performance Embedded Development Environment](image)

Figure 2. In HEW, Select the Build Configuration for the board you are going to use. Selecting configuration determines which files are included in the project, include file directory paths, build options, etc.

Table 3-2 shows the file structure supplied with PDCD.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEDIA/r_usb_atapi_memory.c</td>
<td>FAT (16) /FAT(12) data</td>
<td>Sample</td>
</tr>
<tr>
<td>MEDIA/r_usb_atapi_driver.c</td>
<td>Device driver (PMSDD/media driver)</td>
<td>Sample</td>
</tr>
<tr>
<td>include/r_usb_catapi_define.h</td>
<td>Device driver header file</td>
<td>Sample</td>
</tr>
<tr>
<td>PMSC/r_usb_pmsc_ddi.c</td>
<td>PMSDD interface functions (DDI) Driver registration, storage command callback.</td>
<td></td>
</tr>
<tr>
<td>PMSC/r_usb_pmsc_driver.c</td>
<td>USB class driver (PMSCF)</td>
<td></td>
</tr>
<tr>
<td>PMSC/r_usb_pmsc_pci.c</td>
<td>PCD interface functions (PCI)</td>
<td></td>
</tr>
<tr>
<td>PMSC/r_usb_pmsc_request.c</td>
<td>PCD interface functions (class requests)</td>
<td></td>
</tr>
<tr>
<td>include/r_usb_pmsc_define.h</td>
<td>PMSCD header file</td>
<td></td>
</tr>
<tr>
<td>APL/r_usb_pmsc_descriptor.c</td>
<td>Mass storage class descriptor</td>
<td>Sample</td>
</tr>
<tr>
<td>include/r_usb_cmsc_define.h</td>
<td>PDCD(PMSCD+PMSDD) common header file</td>
<td></td>
</tr>
<tr>
<td>include/r_usb_pmscExtern.h</td>
<td>External reference header file</td>
<td></td>
</tr>
</tbody>
</table>

### 3.4 System Resources

#### 3.4.1 RTOS version
Table 3-3 shows the μITRON resources used by PDCD on the μITRON version.

These resources are defined in the r_usb_peri.cfg file.

For details on how to define, refer to the Renesas USB Device USB Basic Firmware Application note.
### Table 3-3 uITRON Resources

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Name/Task ID</th>
<th>Task Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>USB_PMSC_TSK</td>
<td>PMSCD or <code>usb_pmsc_Task</code> (File: <code>r_usb_pmsc_driver.c</code>) Priority=3</td>
</tr>
<tr>
<td></td>
<td>USB_PFLSH_TSK</td>
<td>PMSDD or <code>usb_pmsc_SmpAtapiTask</code> (File: <code>r_usb_atapi_driver.c</code>) Priority: 4</td>
</tr>
<tr>
<td>Mailboxes</td>
<td>USB_PMSC_MBX</td>
<td>PDCD -&gt; PMSCD / PMSDD -&gt; PMSCD mailbox ID</td>
</tr>
<tr>
<td></td>
<td>USB_PFLSH_MBX</td>
<td>PMSCD -&gt; PMSDD mailbox ID</td>
</tr>
<tr>
<td>Memory pool</td>
<td>USB_PMSC_MPL</td>
<td>PMSCD memory pool ID</td>
</tr>
<tr>
<td></td>
<td>USB_PFLSH_MPL</td>
<td>PMSDD memory pool ID</td>
</tr>
</tbody>
</table>

#### 3.4.2 Non-OS Version

In the Non-OS version of PMSC, there is a scheduler that invokes a “task” when it has message(s) pending in the task’s mailbox, and according to the task’s priority. Table 3-4 lists the ID and priority definitions used to register PMSC in the scheduler.

These are defined in the `r_usb_cKernelId.h` header file.

For details on how to define, refer to the Renesas USB Device USB Basic Firmware Application note.

### Table 3-4 ‘Tasks’ (Mailboxes)

<table>
<thead>
<tr>
<th>Object</th>
<th>Task Name / ID / Mailbox</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>USB_PMSC_TSK / USB_TID_3</td>
<td>PMSCD, or <code>usb_pmsc_Task</code> (File: <code>r_usb_pmsc_driver.c</code>) Priority: USB_PMSC_PRI (default=1)</td>
</tr>
<tr>
<td></td>
<td>USB_PFLSH_TSK / USB_TID_4</td>
<td>PMSDD, or <code>usb_pmsc_SmpAtapiTask</code> (File: <code>r_usb_atapi_driver.c</code>) Priority: USB_PFLSH_PRI (default=2)</td>
</tr>
<tr>
<td>Mailbox ID</td>
<td>USB_PMSC_MBX / USB_PMSC_TSK</td>
<td>PDCD =&gt; PMSCD / PMSDD =&gt; PMSCD (File: <code>r_usb_pmsc_pci.c, r_usb_pmsc_driver.c, r_usb_pmsc_ddi.c</code>)</td>
</tr>
<tr>
<td></td>
<td>USB_PFLSH_MBX / USB_PFLSH_TSK</td>
<td>PMSCD =&gt; PMSDD mailbox ID (File: <code>r_usb_atapi_driver.c</code>)</td>
</tr>
<tr>
<td>Memory pool ID</td>
<td>USB_PMSC_MPL / USB_PMSC_TSK</td>
<td>PMSCD memory pool ID</td>
</tr>
<tr>
<td></td>
<td>USB_PFLSH_MPL / USB_PFLSH_TSK</td>
<td>PMSDD memory pool ID</td>
</tr>
</tbody>
</table>
4. Peripheral MSC Firmware Application Functionality (APL)

Both the USB host and the MCU’s firmware application can read and write to the storage media. This is possible since both have FAT file system software. To avoid “collisions” due to writes from both sides to the FAT table and media, the local file system should not be able to write data when attached to a USB host.

4.1 Media R/W functionality as seen from USB Host

PMSC’s main function is to enable file read/write operations on the connected USB mass storage device. The USB peripheral is to be recognized by the host as a removable disk, so the host (e.g. PC) can perform operations such as read and write files. Mass Storage Class specification defines the transport protocol (BOT), however, various command sets could be used to control a storage device. The following are the command sets which can be used over USB:

SFF-8070i, (ATAPI) * – Command set used in this sample code.
SFF-8020i, MMC-2 (ATAPI)
QIC-157
UFI
SCSI transparent command set

This sample mass storage device driver supports the storage command set SFF-8070i (ATAPI) *.

* As listed in “Mass Storage Specification Overview v1.2”, command block specification SFF-8070i is used (bInterfaceSubClass = 05h) together with protocol code “USB Mass Storage Class Bulk-Only” (BBB; bInterfaceProtocol = 050h).

4.1.1 Operating Environment

The following diagram illustrates data transfer.

Figure 4.1 shows the operating environment example and Figure 4.2 shows the application operations example.
Figure 4.2 Application Operations Example
4.1.2 Application Program Flow

In a sense, there is no “user application”. The mass storage class driver and mass storage device driver solely executes requests from the host.

Figure 4.3 shows the application processing flow overview.

![Application Processing Sequence Diagram]

**Figure 4.3 Application Processing Sequence**
4.1.3 API tasks

Table 4-1 shows lists the APL tasks.

Table 4-1 Lists of APL tasks

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>usb_cstd_task_start</td>
<td>Starts Task</td>
</tr>
<tr>
<td>usb_pmsc_task_start</td>
<td>A variety of Stars Task for peripheral USB</td>
</tr>
<tr>
<td>usb_papl_task_start</td>
<td>Start Application task</td>
</tr>
<tr>
<td>usb_apl_task_switch</td>
<td>Switches Task (non-OS version only)</td>
</tr>
</tbody>
</table>

4.2 Media R/W functionality as seen from MCU application layer

See chapter 9.1 “Overview of Media Driver API Functions”.
5. Peripheral Device Class Driver (PDCD)

5.1 Basic Functions

The functions of PDCD are to:

1. Respond to mass storage class requests from USB host.
2. Respond to USB host storage commands which are encapsulated in the BOT protocol (Bulk Only Transport), see below.

5.2 BOT Protocol Overview

BOT (USB MSC Bulk-Only Transport) is a transfer protocol that, encapsulates command, data, and status (results of commands) using only two endpoints (one bulk in and one bulk out).

The ATAPI storage commands and the response status are embedded in a “Command Block Wrapper” (CBW) and a “Command Status Wrapper” (CSW).

Figure 5-1 shows an overview of how the BOT protocol progresses with command and status data flowing between USB host and peripheral.

![Figure 5-1 BOT protocol Overview. Command and status flow between USB host and peripheral.](image)

5.2.1 CBW processing

When PMSCD receives a command block wrapper (CBW) from the host, it first verifies the validity of the CBW. If the CBW is valid, PMSCD notifies PMSDD of the storage command contained in the CBW and requests analysis of the command. PMSCD finally performs processing based on the analysis by PMSDD (command validity, data transfer direction and size) and the information contained in the wrapper (data communication direction and size).
5.2.2 Sequence for storage command with no data transmit/receive

Figure 5-2 shows the sequence of storage commands without data transfer.

(1) CBW transfer stage

PMSCD issues a CBW receive request to PCD and registers a callback function. When PCD receives the CBW, it executes a callback function which starts the CBW transfer stage. PMSCD verifies the validity of the CBW and transfers the storage command (CBWC) to PMSDD. PMSCD requests PMSDD to execute storage commands. PMSDD executes the storage command and returns the result to PMSCD.

(2) CSW transfer stage

Based on the execution result at the time of callback, PMSCD creates a command status wrapper (CSW) and transmits it to the host via PCD.

For details on PCD operation refer to the USB basic firmware Application note.

---

**Figure 5-2 Sequence of Storage Command for No Transmit/Receive Data**
5.2.3 Sequence with storage command for transmit (IN) data

Figure 5-3 shows the sequence of storage command when there is transmit (IN) data from the peripheral side.

(1) CBW transfer stage

PMSCD executes a CBW receive request to PCD, and sets up a callback. When PCD receives the CBW it executes the callback. PMSCD verifies the validity of the CBW and transfers the storage command (CBWCB) to PMSDD. PMSDD analyzes the data transmit command and returns the result to PMSCD. PMSCD then reads the CBW and sends an ATAPI storage command execution request to PMSDD together with a callback registration.

(2) Data IN transfer stage

Based on the execution result at the time of callback, PMSCD notifies PCD of the data storage area and data size, and data communication with the USB host takes place. When the peripheral PCD issues a transmit end notification (status), PMSCD once again sends a continuation request to PMSDD, and data transmission is repeated.

(3) CSW transfer stage

When PMSCD receives a command processing end result from PMSDD, PMSCD creates a command status wrapper (CSW) and transmits it to the host via PCD.

For PCD operation details refer the USB Basic Firmware Application note.
Figure 5-3 Sequence of Storage Command for Transmit (IN) Data
5.2.4 Sequence for storage command with receive (OUT) data

Figure 5-4 shows the sequence of storage command when there is transmit (OUT) data from the peripheral.

(1) CBW transfer stage

In the CBW transfer stage, PMSCD issues a CBW receive request to PCD and sets up a callback. When PCD receives the CBW it executes the callback. PMSCD verifies the validity of the CBW and transfers the storage command (CBWCB) to PMSDD. PMSDD analyzes the data transmit command, and returns the result to PMSCD. PMSCD then compares the analysis result from PMSDD with the information contained in the CBW and sends an ATAPI storage command execution request to PMSDD together with a callback registration.

(2) Data OUT transfer stage

Based on the callback execution result, PMSCD notifies PCD of the data storage area and data size, and data communication with the host takes place. When it receives transmit end notification from PCD, PMSCD once again sends a common continuation request to PMSDD, and data transmission is repeated.

(3) CSW transfer stage

When it receives a command processing end result from PMSDD, PMSCD creates a command status wrapper (CSW) and transmits it to the host via PCD.

For PCD operation details refer to the USB Basic Firmware Application note.
Figure 5-4  Sequence of Storage Command for Receive (OUT) Data
### 5.2.5 Access sequence for class request

Figure 5-5 shows the sequence when a mass storage class request is received.

1. **Setup Stage**
   
   When PCD receives a class request in the control transfer setup stage, it sends a request received notification to PMSCD.

2. **Data Stage**
   
   PMSCD executes the control transfer data stage and notifies PCD of data stage end by means of a callback function.

3. **Status Stage**
   
   PCD executes the status stage and ends the control transfer.

![Figure 5-5](image-url)  
**Figure 5-5** Sequence for Class Request
6. USB Peripheral Mass Storage Class Driver (PMSCD)

6.1 Basic Functions
The basic interface functions of PMSCD are to register, open, and close the Peripheral Mass Storage Class Driver.

The rest of the functionality inside PMSCD was already described in the sequence charts in chapter 0 Peripheral Device Class Driver (PDCD).

6.2 List of API Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_usb_pmsc_Registration</td>
<td>Registers PMSC driver</td>
</tr>
<tr>
<td>R_usb_pmsc_Open</td>
<td>Open PMSC driver</td>
</tr>
<tr>
<td>R_usb_pmsc_Close</td>
<td>Close PMSC driver</td>
</tr>
</tbody>
</table>
R_usb_pmsc_Registration

Registers PMSC driver

Format

void  R_usb_pmsc_Registration(USB_UTR_t *ptr)

Arguments

USB_UTR_t  *ptr  : Pointer to USB Transfer Structure

Return Values

-

Description

Register for the USB Peripheral Mass Storage Class.

Make your changes to the registration function according to the application program.

Use the USB_PCDREG_t type structure to register the driver to PCD.

For details, refer to "USB basic firmware Application note".

The information registered with the USB_PCDREG_t type structure members are as follows.

- pipetbl  Pipe information table address
- devicetbl  Device descriptor address
- qualitbl  Device qualifier descriptor address
- configtbl  Configuration descriptor address
- othertbl  Other speed descriptor address
- stringtbl  String descriptor address table
- classinit  callback function to start at registration PDCD
- devdefault  Callback function to start at transition to the default state
- devconfig  Callback function to start at transition to the configuration state
- devdetach  Callback function to start at transition to the detach starte
- devsuspend  Callback function to start at transition to the suspend state
- devresume  Callback function to start at transition to the resume state
- interface  Callback function to start at change of interface
- ctrltrans  Callback function to start at control transfer for the user

Notes

1. Please set the following member of USB_UTR_t structure.

   USB_REGADR_t  ipp: USB register base address
   uint16_t  ip : USB IP Number

   If the callback process is not necessary, register to prepare a dummy function.

   For USB device state detail, refer to " Universal Serial Bus Specification Revision 2.0 " Figure 9-1 Device State Diagram

   String descriptor address table is the more string descriptor address table. Reference case is as follows
\begin{verbatim}
uint8_t *usb_gpmsc_StrPtr[USB_STRINGNUM] =
{
  usb_gpmsc_StringDescriptor0, /* Language ID String Descriptor Address */
  usb_gpmsc_StringDescriptor1, /* iManufacturer String Descriptor Address */
  usb_gpmsc_StringDescriptor2, /* iProduct String Descriptor Address */
  usb_gpmsc_StringDescriptor3, /* iInterface String Descriptor Address */
  usb_gpmsc_StringDescriptor4, /* iConfiguration String Descriptor Address */
  usb_gpmsc_StringDescriptor5, /* iConfiguration String Descriptor Address */
  usb_gpmsc_StringDescriptor6 /* iSerialNumber String Descriptor Address */
};

Example

void usb_pmsc_task_start( void )
{
  USB_UTR_t utr;
  USB_UTR_t *ptr;

  ptr = &utr;
  ptr->ip = USB_PERI_USBIP_NUM;
  if( USB_NOUSE_PP != ptr->ip )
  {
    ptr->ipp = R_usb_cstd_GetUsbIpAdr( ptr->ip );
  }

  R_usb_pmsc_Registration( ptr ); /* Peripheral Application Registration */
  R_usb_pmsc_driver_start( ptr ); /* Peripheral Class Driver Task Start setting */
  usb_pstd_usbdriver_start( ptr ); /* Peripheral USB Driver Start Setting */
  usb_papl_task_start( ptr ); /* Peripheral Application Task Start setting */
}
\end{verbatim}
R_usb_pmsc_Open

Open PMSC driver

Format

USB_ER_t R_usb_pmsc_Open(USB_UTR_t *ptr, uint16_t data1, uint16_t data2)

Argument

USB_UTR_t *ptr : Pointer to USB Transfer Structure
uint16_t   data1 : Not used
uint16_t   data2 : Not used

Return Value

uint16_t — Processing result 0:USB_E_OK

Description

This function is called when the USB device is connected to the USB host device and the USB communication has enabled, and sets the CBW reception setting.

Note

This function is registered as a callback function to the member(devconfig) of USB_PCDREG_t structure.

Example

void R_usb_pmsc_Registration(USB_UTR_t *ptr)
{

    USB_PCDREG_t driver;

    /* Driver registration */
    /* Pipe Define Table address */
    driver.pipetbl = &usb_gpmsc_EpPtr[0];
    /* Device descriptor Table address */
    driver.devicetbl = (uint8_t *)&usb_gpmsc_DeviceDescriptor;
    /* Qualifier descriptor Table address */
    driver.qualitbl = (uint8_t *)&usb_gpmsc_QualifierDescriptor;
    /* Configuration descriptor Table address */
    driver.configtbl = (uint8_t *)&usb_gpmsc_ConPtr;
    /* Other configuration descriptor Table address */
    driver.othertbl = (uint8_t *)&usb_gpmsc_ConPtrOther;
    /* String descriptor Table address */
    driver.stringtbl = (uint8_t *)&usb_gpmsc_StrPtr;
    /* Device default */
    driver.devdefault = &R_usb_pmsc_DescriptorChange;
    /* Device configured */
    driver.devconfig = (USB_CB_INFO_t *)&R_usb_pmsc_Open;
    /* Device detach */
    driver.devdetach = (USB_CB_INFO_t *)&R_usb_pmsc_Close;
    /* Device suspend */
    driver.devsuspend = &usb_cstd_DummyFunction;
    /* Device resume */
    driver.devresume = &usb_cstd_DummyFunction;
    /* Interfaced change */
}
driver.interface = &R_usb_pmsc_SetInterface;
/* Control Transfer */
driver.ctrltrans = &usb_pmsc_UsrCtrlTransFunction;
R_usb_pstd_DriverRegistration(ptr, &driver);
R_usb_pmsc_Close

Close PMSC driver

Format

```c
USB_ER_t R_usb_pmsc_Close(USB_UTR_t *ptr, uint16_t data1, uint16_t data2)
```

Argument

- `USB_UTR_t *ptr` : Pointer to a USB Transfer Structure
- `uint16_t data1` : Not used
- `uint16_t data2` : Not used

Return Value

- `uint16_t` : `USB_E_OK`

Description

This function is called at transition to the detached state. There are no operations. Add if necessary.

Note

This function is registered as a callback function to the members (devdetach) of USB_PCDREG_t structure.
Example

```c
void R_usb_pmsc_Registration(USB_UTR_t *ptr)
{
    USB_PCDREG_t driver;

    /* Driver registration */
    /* Pipe Define Table address */
    driver.pipetbl = &usb_gpmsc_EpPtr[0];
    /* Device descriptor Table address */
    driver.devicetbl = (uint8_t*)&usb_gpmsc_DeviceDescriptor;
    /* Qualifier descriptor Table address */
    driver.qualitbl = (uint8_t*)&usb_gpmsc_QualifierDescriptor;
    /* Configuration descriptor Table address */
    driver.configtbl = (uint8_t*)&usb_gpmsc_ConfigPtr;
    /* Other configuration descriptor Table address */
    driver.othertbl = (uint8_t*)&usb_gpmsc_ConPtrOther;
    /* String descriptor Table address */
    driver.stringtbl = (uint8_t*)&usb_gpmsc_StrPtr;
    /* Driver init */
    driver.classinit = &usb_cstd_DummyFunction;
    /* Device default */
    driver.devdefault = &R_usb_pmsc_DescriptorChange;
    /* Device configured */
    driver.devconfig = (USB_CB_INFO_t)&R_usb_pmsc_Open;
    /* Device detach */
    driver.devdetach = (USB_CB_INFO_t)&R_usb_pmsc_Close; //

    /* Device suspend */
    driver.devsuspend = &usb_cstd_DummyFunction;
    /* Device resume */
    driver.devresume = &usb_cstd_DummyFunction;
    /* Interfaced change */
    driver.interface = &R_usb_pmsc_SetInterface;
    /* Control Transfer */
    driver.ctrltrans = &usb_pmsc_UsrCtrlTransFunction;
    R_usb_pstd_DriverRegistration(ptr, &driver);
}
```

6.3 Class Driver Registration
The device class driver PMSCD must be registered with PCD to function. Use the PeripheralRegistration() function to register PMSCD, using the sample code as reference. For details, refer to the Renesas USB Device USB Basic Firmware Application note.

6.4 User Definition Tables
It is necessary to create a descriptor table and pipe information table for use by PCD. Refer to the sample files `r_usb_PMSCdescriptor.c` and `r_usb_PMSCdefEp.h` when creating these tables. For details, refer to the Renesas USB Device USB Basic Firmware Application note.
7. Peripheral Mass Storage Device Driver (PMSDD)

The main function of PMSDD is to analyze and call for execution of storage commands received from the host via PMSCD. Enumeration is set up with InterfaceSubClass as SFF-8070i (ATAPI). This command set is therefore used by the host to control the storage media. These are the storage commands:

- READ10
- INQUIRY
- REQUEST_SENSE
- MODE_SENSE6
- MODE_SENSE10
- READ_FORMAT_CAPACITY
- READ_CAPACITY
- WRITE10
- WRITE_AND_VERIFY
- MODE_SELECT6
- MODE_SELECT10
- FORMAT_UNIT
- TEST_UNIT_READY
- START_STOP_UNIT
- SEEK
- VERIFY10
- PREVENT_ALLOW

PMSDD notifies PMSCD of communication data and execution results related to storage command execution.

PMSDD divides the data transfer into pieces when the transfer data length exceeds the user-specified block count.

A master boot record (FAT16) sample table is provided.
### 7.1 PMSDD Storage Command Structure

The “storage command structure” is USB_PMSC_CDB_t. The format of a storage command (SFF-8070i) differs depending on the command category, so a union is used. Four patterns sort out from ten kinds of command type details as shown in Table 7-1.

<table>
<thead>
<tr>
<th>Union Member</th>
<th>Type</th>
<th>Structure Member</th>
<th>Bit Count</th>
<th>Command Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_usb_ptn0</td>
<td>uint8_t</td>
<td>uc_OpCode</td>
<td></td>
<td>Command determination (common)</td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>b_LUN</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>s_LUN</td>
<td>b_reserved</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>uc_data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s_usb_ptn12</td>
<td>uint8_t</td>
<td>uc_OpCode</td>
<td></td>
<td>INQUIRY / REQUEST_SENSE</td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>b_LUN</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>s_LUN</td>
<td>b_reserved4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>b_immed</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>uc_rsv2[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>uc_Allocation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>uc_rsv1[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>uc_rsv6[6]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s_usb_ptn378</td>
<td>uint8_t</td>
<td>uc_OpCode</td>
<td></td>
<td>Not used (FORMAT UNIT)</td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>b_LUN</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>s_LUN</td>
<td>b_FmtData</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>b_CmpList</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>b_Defect</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>ul_LBA0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>ul_LBA1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>ul_LBA2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>ul_LBA3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>uc_rsv6[6]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s_usb_ptn4569</td>
<td>uint8_t</td>
<td>uc_OpCode</td>
<td></td>
<td>READ10 / WRITE10 / WRITE_AND_VERIFY /</td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>b_LUN</td>
<td>3</td>
<td>MODE_SENSE / FORMAT_CAPACITY / MODE_SELECT</td>
</tr>
<tr>
<td></td>
<td>s_LUN</td>
<td>b_1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>b_reserved2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>b_ByteChk</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>b_SP</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>ul_LogicalBlock0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>ul_LogicalBlock1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>ul_LogicalBlock2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>ul_LogicalBlock3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>uc_rsv1[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>us_Length_Hi</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>us_Length_Lo</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>uc_rsv3[3]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 7-2 shows storage commands analysis result.

### Table 7-2  The USB_PMSC_CBM_t Structure
- Contains “analysis” result of usb_pmsc_SmpAtapi AnalyzeCbwCb.

<table>
<thead>
<tr>
<th>Member</th>
<th>PMSDD storage command analysis RESULT</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t ar_rst</td>
<td>Data direction.</td>
<td>Direction of data transported in last ATAPI command.</td>
</tr>
<tr>
<td>uint32_t ul_size</td>
<td>Data size</td>
<td>Size of data in last ATAPI command.</td>
</tr>
</tbody>
</table>

### 7.2 List of PMSDD Functions

Table 7-3 lists the functions of PMSDD.

### Table 7-3 List of PMSDD Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>usb_pmsc_SmpAtapiAnalyzeCbwCb</td>
<td>Analyzes storage command.</td>
</tr>
<tr>
<td>usb_pmsc_SmpAtapiTask</td>
<td>Main task of PMSDD</td>
</tr>
<tr>
<td>usb_pmsc_SmpAtapiGetReadData</td>
<td>Returns transmit data storage address and data size</td>
</tr>
<tr>
<td>usb_pmsc_SmpAtapiGetReadMemory</td>
<td>Read data address and data size</td>
</tr>
<tr>
<td>usb_pmsc_SmpAtapiGetWriteMemory</td>
<td>Write data address and data size</td>
</tr>
<tr>
<td>usb_pmsc_SmpAtapiInitMedia</td>
<td>Initialization at PMSDD start</td>
</tr>
<tr>
<td>usb_pmsc_SmpAtapiCloseMedia</td>
<td>Processing at PMSDD end</td>
</tr>
<tr>
<td>usb_pmsc_SmpAtapiCommandExecute</td>
<td>Transmits message from PMSCD to PMSDD main task.</td>
</tr>
</tbody>
</table>
7.3 PMSDD Task Description

PMSDD receives storage commands from PMSCD and executes the storage command. PMSDD also receives host data transfer results from PMSCD. Table 7-4 lists PMSDD command processing. When the transfer data size exceeds USB_ATAPI_TRANSFER_UNIT, the data is divided into smaller units and transferred.

For commands that do not involve memory access, the transmitted data is created from the response data tables usp_gpmsec_AtapiDataSize[], usp_gpmsec_AtapiDataIndx[], usp_gpmsec_AtapiReqIndx[], and usp_gpmsec_AtapiRdDataTbl[]. (*1)

(*1) The response data table follows storage command set SFF-8070i, and the index into the table is determined by the command. Refer to uc_OpCode in Table 7-1 USB_PMSC_CDB_t Structure) provided in the subclass.

<table>
<thead>
<tr>
<th>Storage command</th>
<th>Corresponding Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ10</td>
<td>usp_pmsc_SmpAtapiGetReadMemory()</td>
<td>Gets start address and size.</td>
</tr>
<tr>
<td>INQUIRY</td>
<td>usp_pmsc_SmpAtapiGetReadData()</td>
<td>Selects response data from array usp_gpmsec_AtapiRdDataTbl.</td>
</tr>
<tr>
<td>REQUEST_SENSE</td>
<td>usp_pmsc_SmpAtapiGetReadData()</td>
<td>Selects response data from array usp_gpmsec_AtapiRdDataTbl.</td>
</tr>
<tr>
<td>MODE_SENSE10</td>
<td>usp_pmsc_SmpAtapiGetReadData()</td>
<td>Selects response data from array usp_gpmsec_AtapiRdDataTbl.</td>
</tr>
<tr>
<td>READ_FORMAT_CAPACITY</td>
<td>usp_pmsc_SmpAtapiGetReadData()</td>
<td>Selects response data from array usp_gpmsec_AtapiRdDataTbl.</td>
</tr>
<tr>
<td>READ_CAPACITY</td>
<td>usp_pmsc_SmpAtapiGetReadData()</td>
<td>Selects response data from array usp_gpmsec_AtapiRdDataTbl.</td>
</tr>
<tr>
<td>WRITE10</td>
<td>usp_pmsc_SmpAtapiGetWriteMemory()</td>
<td>Gets start address and size.</td>
</tr>
<tr>
<td>WRITE_AND_VERIFY</td>
<td>usp_pmsc_SmpAtapiGetWriteMemory()</td>
<td>Gets start address and size.</td>
</tr>
<tr>
<td>MODE_SELECT10</td>
<td>usp_pmsc_SmpAtapiGetWriteMemory()</td>
<td>Gets start address and size.</td>
</tr>
<tr>
<td>FORMAT_UNIT</td>
<td>usp_pmsc_SmpAtapiGetWriteMemory()</td>
<td>Gets start address and size.</td>
</tr>
<tr>
<td>TEST_UNIT_READY</td>
<td>usp_pmsc_SmpAtapiTask()</td>
<td>Status = USB_PMSC_CMD_COMPLETE</td>
</tr>
<tr>
<td>START_STOP_UNIT</td>
<td>usp_pmsc_SmpAtapiTask()</td>
<td>Status = USB_PMSC_CMD_COMPLETE</td>
</tr>
<tr>
<td>SEEK</td>
<td>usp_pmsc_SmpAtapiTask()</td>
<td>Status = USB_PMSC_CMD_COMPLETE</td>
</tr>
<tr>
<td>VERIFY10</td>
<td>usp_pmsc_SmpAtapiTask()</td>
<td>Status = USB_PMSC_CMD_COMPLETE</td>
</tr>
<tr>
<td>PREVENT_ALLOW</td>
<td>usp_pmsc_SmpAtapiTask()</td>
<td>Status = USB_PMSC_CMD_FAILED</td>
</tr>
<tr>
<td>Else</td>
<td>usp_pmsc_SmpAtapiTask()</td>
<td>Status = USB_PMSC_CMD_ERROR</td>
</tr>
</tbody>
</table>
8. Target File System

A FAT32 file system is incorporated with PMSC. This was added so that the target application can locally read and write files to the media, separately from the USB host.

Before using this file system acknowledge the license conditions at [http://creativecommons.org/licenses/by/3.0](http://creativecommons.org/licenses/by/3.0)

This file system has many features, yet it is rather small in object size. Among them:

- It is a Windows compatible FAT32 file system.
- Long file name support is added. To test this in the demo, set USE_LONG_FILENAMES to 1 in `browse_files_from_rsk.c`. The preprocessor output will then direct you to adjust a few more things to use this.
- Multiple storage sector size support is available. That is, larger than 512 byte memory blocks. This is needed for many storage flash chips, for example the RSK63N has sector (block) storage size 4096 Bytes.

See [http://elm-chan.org/sw/ff/00index_e.html](http://elm-chan.org/sw/ff/00index_e.html) for details on the file system and its API.

8.1 User Interactive Board Demo

A local file browsing demo is also added.

SW1: List next file in current directory.

SW2: View file. Since the board only has a 2*8 character display, the content is only visible in the Debug Console window.

SW3. Add a test file.

In the demo, files are only written when PMSC is in the non-configured USB state. That is, the USB cable is disconnected. This was put in place to avoid duplicate - different - cached copies of FAT tables; the USB host’s and the local one in PMSC. Non-identical copies of FAT tables could arise if both USB host and the PMSC APL both add files while the media is mounted.

The source code for this demo is mainly in the file `browse_files_from_rsk.c`. By maneuvering switches SW1-3 the user browses the media and add test files.

The file system and demo can be removed if desired, e.g. to save space. In that case exclude files of the HEW project window pane “Board_app_file_access”, and any calls to these modules or demo switches etc.

8.2 Long Filenames

Please be aware of any license restrictions on using long filenames in your product.

To use long file names, set USE_LONG_FILENAMES to 1 in `browse_files_from_rsk.c` and follow the text from the preprocessor output to enable this. When enabled, The demo then write files with long names when SW3 is pressed.
9. Media Driver Interface

This chapter is an introduction to the block **USB PMSC with Local File System and Flash Storage**, and how it is used for PMSC. For complete details on this API and how to create new media drivers that interface through it, see application note no. r01an1443eu_rx.

PMSC is able to operate with a variety of devices as data storage media. It uses a block storage type driver API described in the application note **USB PMSC with Local File System and Flash Storage** (Document No. r01an1443eu_rx). The storage media interface is an abstract set of functions (R_MEDIA_Read, R_MEDIA_Write, etc) which are the same regardless of the underlying driver that will be called behind the interface. PMSC can interface any media driver that supports this API.

By default, this application uses a RAM-disk media driver that is supplied preconfigured to use a specific RAM memory area as example storage media. For the RSK63N there is the option to use the board’s SPI-flash as storage. See 9.3.2.

### 9.1 Overview of Media Driver API Functions

The Block Access Media Driver API serves to interface the PMSC application to a specific media device driver. The selection of media is made through configuration files that the user must customize. There is one configuration file for the Block Access Media Driver API, r_media_driver_api_config.h, which has a list of media devices, and another configuration file for PMSC, r_usb_atapi_driver_config.h, which assigns the selected media driver to be used for PMSC.

The transport layer subtype in this application is SFF-8070i (ATAPI). This layer processes the storage commands that are contained in the Command Blocks that are tunneled through the BOT transport layer. Most of the work done to process the command set is accomplished by routines in the file r_usb_atapi_driver.c. This is where the ATAPI data storage commands that write or read the storage media, that is, the block API calls, originate. Storage commands pass through the Block Access Media Driver API layer where they are directed to drivers for the assigned storage device.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_MEDIA_Initialize</td>
<td>Registers the media driver</td>
</tr>
<tr>
<td>R_MEDIA_Open</td>
<td>Open media driver</td>
</tr>
<tr>
<td>R_MEDIA_Close</td>
<td>Close media driver</td>
</tr>
<tr>
<td>R_MEDIA_Read</td>
<td>Read from a media device</td>
</tr>
<tr>
<td>R_MEDIA_Write</td>
<td>Write to a media device</td>
</tr>
<tr>
<td>R_MEDIA_ioctl</td>
<td>Perform control and query operations on a media device</td>
</tr>
</tbody>
</table>

### 9.2 Selecting Media Driver

A media driver has a structure that contains the pointers to its implementation of the API’s abstract functions shown in Table 9-1 above. The name of this driver implementation structure must be assigned to a macro used by the ATAPI task: ATAPI_MEDIA_DEVICE_DRIVER in r_media_driver_api_config.h.

The media driver has a logical unit number, LUN, assigned to it. The LUN used by the ATAPI task must be defined by the macro ATAPI_MEDIA_LUN. The storage media used will be determined by how ATAPI_MEDIA_LUN is defined.

Example:

```c
/* r_usb_atapi_driver_config.h */
#define ATAPI_MEDIA_DEVICE_DRIVER g_RamMediaDriver
#define ATAPI_MEDIA_LUN RAM_DISK_LUN
#define USB_ATAPI_BLOCK_UNIT RAMDISK_SECTSIZE
```

There are a few examples where this is done. To change (only RSK63N can be changed for now) uncomment one of the macros under
/* UNCOMMENT ONE STORAGE MEDIA */

In the default example code for PSC, and as can be seen above, `g_RamMediaDriver` is a RAM-disk driver media structure instantiated in the `r_ram_disk.c` source module. RAM_DISK_LUN is from the enumerated list of media drivers in `r_media_driver_api_config.h`. The section, or block, size must match both what the host can handle (e.g. Windows FAT) and what the bottom layer driver can handle, e.g. 512 or 4096 bytes.

### 9.2.1 Initializing the Media Driver Function Set

Once the block media driver functions listed in the `g_RamMediaDriver` structure actually exist, all that is needed is to call the abstract function

```c
R_MEDIA_Initialize(ATAPI_MEDIA_LUN, &ATAPI_MEDIA_DEVICE_DRIVER);
```

which will write the actual driver member functions to `g_MediaDriverList[]` at runtime. Once the member functions have populated MediaDriverList, calls to the other abstract block media functions; R_MEDIA_Open, R_MEDIA_Read, R_MEDIA_Write,... will redirect to call the user’s particular driver functions. In other words, it all happens behind the scene without the user having to replace the abstract call with the actual driver calls.

This runtime registration of the drivers can be omitted and the member functions be called directly. In that case the member functions cannot be declared static in the driver source code.

This initialization call is already done in PMSC in file `r_media_driver_api.c`.

### 9.3 Changing (adding) Storage Media

Suppose you would like to change what media the data is stored on. Default is RAM, which being volatile memory will not survive a power cycling. To use a different storage media, the read, write, etc functions must first be made to conform to the block media driver API described above.

#### 9.3.1 Steps to conform a driver to the block media API

1. In `r_media_driver_api_config.h` add the new media logical unit number to the enumeration.
2. Add the media driver interface function source code, using return types and arguments as specified in `r_media_driver_api.h`. Add a `media_driver_s` structure containing pointers to these members at the top.
3. In `r_usb_atapi_driver_config.h`, add the definitions as described in 9.2.
4. Make sure to call the abstract initialize function.

#### 9.3.2 Example - Serial Flash as Storage Media

As an example of a different permanent block storage media (as opposed to volatile RAM) a SPI-flash driver for a serial SPI flash (U4 on the RSK63N) is added to the source tree. The serial flash chip is connected to the MCU over the Renesas SPI interface; “RSPI”.

The member functions of this SPI flash driver, and all other files needed to use the SPI flash are in the folder `spi_flash_block_if`. This folder contains the files shown in the SPI folder in HEW’s project pane. The file `spi_flash_block_if.c` holds the API function definitions to interface the storage chip.

#### Setup To Use Serial Flash

The serial flash is available for the RSK63N only (PMSC v4.00), and is activated by following these steps.

1. Change `r_usb_atapi_driver_config.h` as shown below. This will change the build so that the code will use the serial flash instead of RAM.

   ```c
   /* UNCOMMENT ONE STORAGE MEDIA */
   //#define USE_RAM       1
   #define USE_SPI_FLASH   1
   ```

2. Include the files grouped under the SPI folder in the project / build configuration. That is, the files `spi_flash_block_if.c`, `(spi_flash_test.c)`, `r_rspi_rx.c`, and `r_spi_flash.c` with associated include files.

3. Switch to 4 kB block size: Change `MAX_SS` in `ffconf.h` to be `0x1000` (4096 bytes). Also make sure that `SPIFLASH_DISK_SECTSIZE` in `spi_flash_block_if.c` is the same size. See Restrictions below for more information regarding block size change.
Restrictions

The default demo, using RAM, with 512 byte sector size, will run fine on any Windows USB host.

To use the serial flash on the RSK63N with Windows as the USB host, Windows 7 or later must be used. This is because the SPI flash needs a 4 kB file system allocation block size since that is the erase size of the U4 chip on the RSK63N. Windows 7 supports 4 kB physical blocks as opposed to previous versions of Windows.

The industry is rapidly moving towards 4 kB size for storage media. Using 512 bytes as sector size for the SPI flash driver, being non-native for this particular SPI-flash would cause the code to be much complex, wear out the flash chip up to 8 times as fast, and make writing some 8 times as slow. More on 4 kB sector (block) size for media storage can be found at

http://www.support.microsoft.com/kb/2510009

Figure 6. When using the SPI-flash for RSK63N, the drive must be formatted using 4 kB sectors with Windows7 (standard block size going forward) as that is the erase size of the serial flash chip used.
10. The RAM Media Driver

The RAM-disk media driver that is supplied preconfigured to use RAM memory area as a virtual storage media device. While this is provided primarily as a simplified example of a media driver to demonstrate the functionality of USB-PMSC, it can still be useful as a means to transfer data to or from the MCU to the USB host. The RAM-disk uses either on-chip RAM or external SDRAM if available.

1. The RX62N and RX63N sample code operates as media with 8 MB SDRAM
2. The RX630 operates as media with internal 32-kByte RAM.
3. Media is preformatted to appear as removable FAT file storage device.
4. Media driver supports connection to Windows OS (2000, XP, 7, etc) host.
5. Default format may be overwritten by format command from the host.
6. Host can read files from RAM-disk or write files to it, and can update the FAT format information as needed.
7. Size of the RAM-disk can be configured at build time by settings in a configuration file.

10.1 RAM-disk Media Driver Default FAT Format

The RAM-disk virtual media device is preformatted to appear as removable FAT file storage device. This permits files to be placed in RAM and read back by the USB host. This format is implemented as a pre-initialized data section in RAM with boot sector, FAT tables, and directory areas defined by hard-coded values. The file \texttt{r\_ram\_disk\_format\_data.c} contains the section declarations and the pre-initialized data that will get copied to RAM at system startup.

The beginning of the RAM-disk area (lowest memory address) is considered to be the boot sector (sector 0), as block zero is the default boot sector area in a FAT formatted storage device. Therefore, all storage blocks are addressed relative to this location. When a host device accesses the media device it will always communicate in terms of starting logical block number (LBN) and block count (how many blocks to transfer.) Since the host knows how to navigate a FAT formatted storage device, it will read the first sector to gain additional information about the specific format on the RAM-disk, and from there it will discover where to look for additional file information. From that the host will know which block number to access.

Alternatively, the host can re-format the RAM-disk, replacing the default boot sector, FAT tables, etc, with its own format. In this case the host still knows where in the RAM image a specific block of data resides.

Note: It is not strictly necessary for a USB-PMSC device to have a FAT file system format, however most host systems will expect to use the PMSC device for file storage with FAT as the file system type.

10.2 RAM-disk Global Area Variables

The entire RAM-disk RAM section is of global scope with a number of named variables. Table 10-1 lists the global area variables of the media driver.

Table 10-1 Media Driver Global Areas

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td>g_ram_disk_boot_sector</td>
<td>Primary Boot Record area (sector 0)</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_ram_disk_table1</td>
<td>Dummy area (sector 1) *</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_ram_disk_table_fat1</td>
<td>FAT table (sector 2) *</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_ram_disk_table_fat2</td>
<td>FAT table (sector 3/66) *</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_ram_disk_root_dir</td>
<td>Directory entry area (sector 4/130) *</td>
</tr>
</tbody>
</table>

*Note: This RAM-disk media driver implementation only references the \texttt{g\_ram\_disk\_boot\_sector} variable, which corresponds to the beginning of the RAM-disk memory section.
10.3 Constant Definitions

Table 10-2 shows the constant definitions used for the RAM-disk media driver.

### Table 10-2 PMSDD Constant Definitions

<table>
<thead>
<tr>
<th>Description</th>
<th>Definition name</th>
<th>Value</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media type</td>
<td>RAMDISK_MEDIATYPE</td>
<td>0xF8u</td>
<td>Modifiable</td>
</tr>
<tr>
<td>Signature</td>
<td>RAMDISK_SIGNATURE</td>
<td>0xAA55u</td>
<td>Not modifiable</td>
</tr>
<tr>
<td>Sector size</td>
<td>RAMDISK_SECTSIZEx</td>
<td>512ul</td>
<td>Modifiable</td>
</tr>
<tr>
<td>Cluster size</td>
<td>RAMDISK_CLSTSIZE</td>
<td>0x01u</td>
<td>Modifiable</td>
</tr>
<tr>
<td>FAT number</td>
<td>RAMDISK_FATNUM</td>
<td>0x02u</td>
<td>Modifiable</td>
</tr>
<tr>
<td>Media size *1</td>
<td>RAMDISK_MEDIASIZE</td>
<td>RX62N, 6X63N: 8<em>1024</em>1024 (=8 MB) (\times 32*1024 \text{ (=32 kB)})</td>
<td>Modifiable</td>
</tr>
<tr>
<td>Total number of sectors*2</td>
<td>RAMDISK_TOTALSECT</td>
<td>USB_MEDIA_SIZE / RAMDISK_SECTSIZEx</td>
<td>Not modifiable</td>
</tr>
<tr>
<td>FAT Table Length*2</td>
<td>RAMDISK_FATLENGTH</td>
<td>341ul (\text{（FAT12, 256ul（FAT16）)})</td>
<td>Not modifiable</td>
</tr>
<tr>
<td>FAT table length*2</td>
<td>RAMDISK_FATSIZEx</td>
<td>(((\text{RAMDISK_TOTALSECT}-8) / \text{RAMDISK_FATLENGTH})+1)</td>
<td>Not modifiable</td>
</tr>
<tr>
<td>Root directory</td>
<td>RAMDISK_ROOTTOP</td>
<td>(((\text{RAMDISK_FATSIZE} * \text{RAMDISK_FATNUM}+1)/8+1)*8) (\text{Not modifiable (Not used)})</td>
<td>Not modifiable</td>
</tr>
<tr>
<td>FAT start</td>
<td>RAMDISK_FATTOP</td>
<td>(\text{RAMDISK_ROOTTOP} - (\text{RAMDISK_FATSIZE} * \text{RAMDISK_FATNUM}))</td>
<td>Not modifiable</td>
</tr>
<tr>
<td>Root Directory size</td>
<td>RAMDISK_ROOTSIZE</td>
<td>1ul</td>
<td>Not modifiable</td>
</tr>
</tbody>
</table>

*1 A minimum 20K byte capacity is required when connecting the device to a PC running WindowsXP.

FAT12 is selected when the media size is set to under 2M bytes.

FAT16 is selected when the media size is set to under 32M bytes.

“RAMDISK_MEDIASIZE” is defined in r_ram_disk.h and is 8M bytes for RX62N and RX63N SDRAMs and 32K bytes for RX630 built-in RAM.

*2 Total number of sectors, FAT data length, and FAT table length are automatically calculated based on the media size.
## 10.4 Operation Overview

Table 10-3 lists the RAM media variables and Figure 10.1 shows the RAM media block diagram.

### Table 10-3 Media Variables

<table>
<thead>
<tr>
<th>Sector No.</th>
<th>Physical Address</th>
<th>Accessible Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sector 0</td>
<td>g_ram_disk_boot_sector[]</td>
<td>512Byte</td>
</tr>
<tr>
<td>Sector 1</td>
<td>g_ram_disk_table1[]</td>
<td>512Byte</td>
</tr>
<tr>
<td>Sector 2</td>
<td>g_ram_disk_table_fat1[]</td>
<td>512Byte× RAMDISK_FATSIZE</td>
</tr>
<tr>
<td>Sector 3/66</td>
<td>g_ram_disk_table_fat2[]</td>
<td>512Byte× RAMDISK_FATSIZE</td>
</tr>
<tr>
<td>Sector 4/130</td>
<td>g_ram_disk_root_dir[]</td>
<td>512Byte×16</td>
</tr>
</tbody>
</table>

![Figure 10.1 Media Block](image-url)
11. OS and Non-OS Resources

The methods of including resources when using the RTOS and non-OS versions are described below.

Please refer to USB Basic Firmware Application note for detail of how to register.

11.1 Including Resources in RTOS

When using the RTOS version (RI600), it is necessary to register tasks, mailboxes, memory pools, and interrupt vectors in the ITRON configuration file (r_usb_peri.cfg).

11.1.1 PMSC Registration

In the sample program, the PMSC task is registered as follows.

```c

task[]{
    entry_address = usb_pmsc_Task();
    name    = USB_PMSC_TSK;
    stack_size  = 512;
    stack_section = SURI_STACK;
    priority   = 5;
    initial_start = OFF;
    exinf   = 0x0;
};
```

11.1.2 APL Registration

In the sample program, the APL task is registered as follows. The priority is defined as the lowest level, except for idle tasks.

```c

task[]{
    entry_address = usb_pmsc_SmpAtapiTask();
    name    = USB_PFLSH_TSK;
    stack_size  = 512;
    stack_section = SURI_STACK;
    priority   = 5;
    initial_start = OFF;
    exinf   = 0x0;
};
```

11.1.3 PMSC Mailbox Registration

In the sample program, the PMSC mailbox is registered as follows.

```c

mailbox[]{
    name = USB_CLS_MBX;
    wait_queue  = TA_TFIFO;
    message_queue = TA_MFIFO;
    max_pri   = 1;
};
```

11.1.4 APL Mailbox Registration

In the sample program, the APL mailbox is registered as follows.

```c

mailbox[]{
    name = USB_PMSC_MBX;
    wait_queue  = TA_TFIFO;
    message_queue = TA_MFIFO;
    max_pri   = 1;
};
```
11.1.5 PMSC Memory Pool Registration
In the sample program, the PMSC memory pool is defined as ten 64-byte blocks.

```c
memorypool[]{
    name        = USB_PCD_MPL;
    wait_queue  = TA_TFIFO;
    section     = BRI_HEAP;
    siz_block   = 64;
    num_block   = 10;
}
```

11.1.6 APL Memory Pool Registration
In the sample program, the APL memory pool is defined as ten 64-byte blocks.

```c
memorypool[]{
    name        = SB_PMSC_MPL;
    wait_queue  = TA_TFIFO;
    section     = BRI_HEAP;
    siz_block   = 64;
    num_block   = 10;
}
```

11.2 Resource Registration in Non-OS Scheduler
When using the non-OS scheduler, it is necessary to register resources such as task IDs, mailbox IDs, and memory pool IDs in the file "r_usb_cKernelId.h".

In the sample file, the registrations are as follows.

```c
/* Peripheral MSC Driver Task */
#define USB_PMSC_TSK  USB_TID_3    /* Task ID */
#define USB_PMSC_MBX  USB_PMSC_TSK   /* Mailbox ID */
#define USB_PMSC_MPL  USB_PMSC_TSK   /* Memorypool ID */

/* Peripheral MSC Sample Task */
#define USB_PFLSH_TSK  USB_TID_4   /* Task ID */
#define USB_PFLSH_MBX  USB_PFLSH_TSK /* Mailbox ID */
#define USB_PFLSH_MPL  USB_PFLSH_TSK /* Memorypool ID */
```
12. **Limitations**

The following limitations apply to PMSC.

1. Structures are composed of members of different types.
13. Using the Renesas Debug Console

The Renesas Debug Console means you have the ability to use `printf()` statements in C to send trace strings to the standard output. Standard output will in this case be the E1/E20 debug register. To use this feature, the following must be true.

1. `INIT_JOLIB()` must be called. This is sometimes commented out in `resetprog.c` to save object code space.
2. In ../media_driver/rbsp/board/rskrx63nr_bsp_config.h, set
   
   ```
   #define BSP_CFG_IO_LIB_ENABLE (1)
   ```
3. The code in 13.1 constitutes the `putchar` and `getchar` functions to reside in `lowlvl.src` so that the E1/E20 debug ports are used for I/O processing. Replace the existing code with this if is not already in `lowlvl.src`.
4. Include `<stdio.h>` in any files where you wish to use printf-statements.
5. In e’ studio, add the Debug Console window by switching on both icons

   “I/O” and

   “Pin Console” as shown below.

   Both must be on so that the print buffer in E1/E20 can be emptied, and not block.

13.1 STDIO Low Level Source Code

Use the following code in `lowlvl.src` to get printf statements to the E1/E20 Debug Console.

```c
; FILE : lowlvl.s
; DATE : Wed, Jul 01, 2009
; DESCRIPTION : Program of Low level
; CPU TYPE : RX
;-----------------------------------------------------------------------
; _charput
; _charget

.FC2E0 .EQU 00084080h
.FE2C0 .EQU 00084090h
DBGSTAT .EQU 000840C0h
RXFL0EN .EQU 00001000h
TXFL0EN .EQU 00000100h

.SECTION P, CODE
;-----------------------------------------------------------------------
; _charput:
;-----------------------------------------------------------------------
;__C2ESTART:
;MOV.L #TXFL0EN,R3
;MOV.L #DBGSTAT,R4
;__TXLOOP:
;MOV.L [R4],R5
;AND     R3,R5
;BNZ     __TXLOOP
;__WRITEFC2E0:
;MOV.L #FC2E0,R2
;MOV.L R1,[R2]
;__CHARPUTEXIT:
;RTS
;-----------------------------------------------------------------------
; _charget:
;-----------------------------------------------------------------------
;__E2CSTART:
;MOV.L #RXFL0EN,R3
;MOV.L #DBGSTAT,R4
;__RXLOOP:
;MOV.L [R4],R5
;AND     R3,R5
;BZ      __RXLOOP
;__READFE2C0:
;MOV.L #FE2C0,R2
;MOV.L R1,[R2]
;__CHARGETEXIT:
;RTS
;-----------------------------------------------------------------------
; End of conditional code (section)
; .END
```
Website and Support

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

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

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

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>May 26, 2014</td>
<td>All</td>
<td>Adapted from r01an0514ej_usb_pmsc, v.210.</td>
</tr>
</tbody>
</table>

**Document:**
- Added chapter “Target File System”, and refined the chapter “Media Driver Interface”.
- Added chapter “Using the Renesas Debug Console”.
- Added chapter “Using E2studio”.

**Source code:** Major feature upgrades.
- Added target file system (Elm FAT32) to code. A local application on board can now read and write files to the media, aside from the USB host.
- A local file browsing demo is added. User uses switches SW1-3 to browse, add test files etc.
- Long filenames are fully incorporated (R/W), and multiple drives are supported (TFAT only one drive).
- SPI flash on RSK63N can be used as storage. For Win7 and later only.
### General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

<table>
<thead>
<tr>
<th>1. Handling of Unused Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.</td>
</tr>
<tr>
<td>- The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2. Processing at Power-on</th>
</tr>
</thead>
<tbody>
<tr>
<td>The state of the product is undefined at the moment when power is supplied.</td>
</tr>
<tr>
<td>- The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.</td>
</tr>
<tr>
<td>In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.</td>
</tr>
<tr>
<td>In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3. Prohibition of Access to Reserved Addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access to reserved addresses is prohibited.</td>
</tr>
<tr>
<td>- The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. Clock Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.</td>
</tr>
<tr>
<td>- When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5. Differences between Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.</td>
</tr>
<tr>
<td>- The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.</td>
</tr>
</tbody>
</table>
Notice

1. Descriptions of circuits, software, and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error-free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors or omissions from the information included herein.

3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.

5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunction under certain use conditions. Further, Renesas Electronics products are not subject to radiation design. 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 Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.

11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

Reneses LA

(SALES OFFICES) Renesas Electronics Corporation

http://www.renesas.com

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

Reneses LA

Renesas Electronics America Inc.,
2820 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
2101 Nicholson Road, Nepean, Ontario L3Y 9C1, Canada
Tel: +1-613-897-5441, Fax: +1-613-897-2220

Renesas Electronics Europe Limited
Dukes Meadow, Millbridge Road, Basingstoke, Hampshire, RG24 9FH, U.K.
Tel: +44-1628-585-500, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiaplatz 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1237

Reneses LA

Reneses LA

Reneses LA

Reneses LA