R-Car Series, 3rd Generation
Capacity Aware Migration Strategy

Introduction

Purpose

Modern in-vehicle information systems are required to be capable of running several applications, including navigation, the playback of music and video, and the display of information on meters, at the same time. For use cases such as this, we recommend the use of a multi-core processing environment, which allows multiple CPUs to operate in parallel. Doing so improves the computational capability of the system and leads to the applications behaving in a user-friendly fashion.

Third-generation R-Car series products have two types of CPU: Arm® Cortex®-A57 and Cortex®-A53 cores (for a total of eight cores in the R-Car H3 and six in the R-Car M3-W/R-Car M3-W+). Applying a capacity aware migration strategy (CAS) is a way to obtain efficient utilization of multi-core processing. This document describes the CAS functionality and associated debugging.

Target Readers

Readers of this document are assumed to have general knowledge in the fields and specific technologies listed below.

- Engineering, logic circuits, microcontrollers, and Linux.
- The functionality of the multiple processor cores of R-Car H3, R-Car M3-W and R-Car M3-W+ products.
- The electrical specifications of the multiple processor cores of R-Car H3, R-Car M3-W and R-Car M3-W+ products.
- The functions of the BSP drivers for R-Car H3, R-Car M3-W and R-Car M3-W+ products.

Note

- Statements in relation to operating systems in this document apply to Yocto v3.7.0 from Renesas.

Target Devices

- R-Car H3
- R-Car M3-W/ R-Car M3-W+
## Contents

1. Overview .................................................................................................................. 3  
   1.1 What is CAS? ........................................................................................................ 3  
   1.2 Outline of the CAS Functionality .......................................................................... 3  
   1.3 CAS Operations in Outline .................................................................................. 4  

2. Guide to Debugging .................................................................................................. 5  
   2.1 Monitoring Tasks with htop .................................................................................. 5  
      2.1.1 Preparations in Advance .............................................................................. 5  
      2.1.2 Procedure for Handling ............................................................................. 5  
   2.2 Monitoring Tasks with ftrace .............................................................................. 6  
      2.2.1 Preparations in Advance .............................................................................. 6  
      2.2.2 Procedure for Handling ............................................................................. 6  

3. Usage Notes on CAS Functionality ......................................................................... 7  
   3.1 Load Balancing by the Scheduler ........................................................................ 7  
   3.2 Realizing High Response Speeds in Task Processing ......................................... 8  
      3.2.1 Processor Affinity as a Measure .................................................................. 8  
      3.2.2 Measure in the Form of Making Processes Realtime ................................. 13  

4. Disabling CAS Functionality ................................................................................... 14  

Appendix ....................................................................................................................... 15  
   A1. Times for Task Switching between the Cortex-A57 and Cortex-A53 Cores ........ 15  
   A2. Building an Environment for htop ....................................................................... 16  
   A3. Building an Environment for ftrace .................................................................... 17  
   A4. Building an Environment for taskset ................................................................... 18
1. Overview

1.1 What is CAS?

CAS is a function Renesas provides for use with asymmetric multiprocessing systems. It involves the consideration of SoC-dependent parameters in addition to support by the Linux kernel for asymmetric CPU capacities. Here, “CPU capacity” refers to the computational capability of a CPU, and “support for asymmetric CPU capacities” refers to functionality which is provided as part of the energy-aware scheduler (EAS) from Arm Limited.

![Figure 1-1 Configuration and Background of CAS](image)

1.2 Outline of the CAS Functionality

In devices in which the Cortex-A57 and Cortex-A53 cores are operating at the same time, CAS assigns tasks in consideration of the performance of the cores.

The standard multi-processor scheduler of Linux is the completely fair scheduler (CFS). This allows the effective use of multi-core processing by assigning tasks to CPUs such that the load on each CPU is equal (see CFS in Figure 1-2).

On the other hand, the Cortex-A57 and Cortex-A53 cores differ in performance. Assigning more tasks to the Cortex-A57 cores with their greater computational capability instead of making the load on each CPU equal improves the performance of the system as a whole.

In CAS, the scheduler is made aware of the difference between the CPU capacities of Cortex-A57 and Cortex-A53 cores and gives higher priority to the Cortex-A57 cores in assigning tasks (see CAS in Figure 1-2).

![Figure 1-2 Assigning Tasks](image)
1.3 CAS Operations in Outline

CAS features the scheduling of tasks in awareness of the capacities of the target CPUs. The concrete behavior of the scheduler is described below.

The capacity of each of the CPUs is defined in the Linux kernel. The CPU capacities are divided into two parts in terms of the current assignment of tasks: the capacity in use and the idle capacity. When a task is assigned to a CPU, the part of the capacity of the given CPU that is in use is increased by the load of the task and its idle capacity is thus decreased by the same amount.

As the Cortex-A53 and Cortex-A57 cores have different computational capabilities, their maximum CPU capacities differ accordingly. Enabling CAS has the scheduler assign tasks to the CPU with the largest idle capacity. As a result, the scheduler gives higher priority to the Cortex-A57 cores in assigning tasks because the maximum CPU capacity of a Cortex-A57 core is greater than that of a Cortex-A53 core.

Specifically, an example of assigning tasks with different loads is given in Figure 1-3. Note that the configuration of CPUs in the figure is simplified for the sake of explanation and is different from those in the actual devices.

Example of a low load: Idle capacity (a) of a Cortex-A57 core > Idle capacity (b) of a Cortex-A53 core
    The scheduler starts assigning tasks to the Cortex-A57 cores.

Example of a medium load: Idle capacity (a) of a Cortex-A57 core > Idle capacity (b) of a Cortex-A53 core
    The scheduler starts assigning tasks to the Cortex-A57 cores.

Example of a heavy load: Idle capacity (a) of a Cortex-A57 core < Idle capacity (b) of a Cortex-A53 core
    The scheduler starts assigning tasks to the Cortex-A53 cores.

As described above, enabling CAS gives higher priority to the Cortex-A57 cores in assigning tasks because of the difference in the CPU capacity, and this allows capacity-aware scheduling of tasks for the CPUs.

![Figure 1-3 Example of Assigning Tasks According to Loads Imposed on the Individual CPUs](image-url)
2. Guide to Debugging

Improving the performance of applications by using CAS strongly depends on the use cases of users. To make full use of CAS in your use case, monitoring how tasks are being assigned and the transitions and progress of tasks is very important. This section describes two methods of monitoring tasks (htop and ftrace) which are effective in debugging.

2.1 Monitoring Tasks with htop

This covers htop as a simplified method of monitoring how tasks are being assigned.

htop is a tool with which loads imposed on CPUs, the state of assignment to CPUs of tasks in progress, and other information can be monitored on the console. htop can be used by adding it to the file system beforehand.

Note that htop displays information as a snapshot. Therefore, htop is not capable of continuously capturing the transitions and progress of tasks. To monitor continuous changes in the situation regarding tasks, we recommend using the ftrace tracing tool for the Linux kernel as the method of monitoring tasks (see section 2.2).

2.1.1 Preparations in Advance

See appendix A2, Building an Environment for htop.

2.1.2 Procedure for Handling

Enter the following command in a console window on the host PC.

```bash
$ htop
```

Executing the command displays the states of tasks in the console window as shown in the figure below.

![Figure 2-1 Example of a Display Produced by htop for the R-Car H3](image)

For details on how to use htop, see the “Help” for htop.
2.2 Monitoring Tasks with ftrace

To continuously monitor the detailed state of the assignment of tasks, such as the transitions and progress of tasks, use the ftrace tracing tool for the Linux kernel.

ftrace is the internal tracer of the Linux kernel and can output logs of events in kernel operations that have been executed.

Use trace-cmd to analyze the output logs and KernelShark to display results from trace-cmd in a GUI.

2.2.1 Preparations in Advance

See appendix A3, Building an Environment for ftrace.

2.2.2 Procedure for Handling

(1) Handling of Tracing (Recording)

Enter the following command in a console window to start tracing while tasks to be monitored are running.

```bash
$ trace-cmd record -e sched -b 4000
```

To end tracing, press the Ctrl + C (‘Ctrl’ + ‘C’) keys in the console window in which you entered the command.

The traced data are stored as a file named “trace.dat” in the current folder.

(2) Display of traced data

Use KernelShark in the following way to display the results of tracing that were obtained.

Use the following command to analyze an obtained trace.dat file on a Linux host PC on which KernelShark has been installed.

```bash
$ kernelshark {position of the file/}trace.dat
```

![Figure 2-2 Example of Using KernelShark to Display the Results of Tracing by ftrace for the R-Car H3](image)
3. Usage Notes on CAS Functionality

3.1 Load Balancing by the Scheduler

Although using CAS causes the scheduler to give higher priority to Cortex-A57 cores in assigning tasks, the scheduler may momentarily assign a task to a Cortex-A53 core due to the load balancing process which is the operation of the standard Linux scheduler, that is, completely fair scheduling (CFS). In such cases, the scheduler subsequently reassigns the task to a Cortex-A57.

This behavior of the scheduler leads to lowering of the processing performance, which may lead to poor response by a task handled in the way described above relative to the case where the task is run solely on a Cortex-A57 core, due to the difference in performance between the Cortex-A57 and Cortex-A53 cores.

---

**Figure 3-1  Momentary Assignment of a Task to a Cortex-A53**

![Diagram showing momentary assignment of a task to a Cortex-A53 core](image)
3.2 Realizing High Response Speeds in Task Processing

3.2.1 Processor Affinity as a Measure

Processor affinity is the method of allowing users to bind a specified application to a particular CPU or range of CPUs for running the application. Applying processor affinity in an environment where the Cortex-A57s and Cortex-A53s are booted up at the same time allows users to bind an application to the Cortex-A57s to prevent a Cortex-A53 from unexpectedly having to handle a heavy load of processing.

There are several methods of realizing processor affinity under Linux. They are listed and described in Table 3-1. We recommend control group (cgroup) from the viewpoint of controlling processes in groups.

<table>
<thead>
<tr>
<th>Method of Realizing Processor Affinity</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cgroup</td>
<td>Cgroup is a Linux standard feature, where processes are classified into groups for control of the assignment of resources such as CPUs and memory. Cgroup can be handled through sysfs.</td>
</tr>
<tr>
<td>taskset</td>
<td>The taskset command can be used to realize processor affinity by specifying process IDs (PIDs) and the CPUs to run the processes from the command line. With affinity through the taskset command, CPUs must be assigned per process.</td>
</tr>
</tbody>
</table>

We recommend the methods described up to this point in this document. Meanwhile, the Linux community suggests another method in which the bLsched daemon in the user space assigns tasks to CPUs based on the idea of realizing processor affinity. Details on bLsched are available on the Web page at the URL below. Refer to the page if you wish to know more it.

https://github.com/BayLibre/bLsched
3.2.1.1 Procedure for Setting Cgroup

Cgroup is a standard Linux feature in which processes are classified into groups for control of the assignment of resources such as CPUs and memory. Multiple groups can be created. Processes classified in the same group run by using specified resources. In an environment in which the Cortex-A57s and Cortex-A53s are booted up at the same time, applying cgroup allows the classification of applications as being for execution on a Cortex-A57 or a Cortex-A53 and assigning the applications to specified CPU resources as shown in Figure 3-2.

Here, the setting to prevent a Cortex-A53 from unexpectedly having to handle a task intended for a Cortex-A57 by assigning applications solely to a Cortex-A57 is described as an example.

![Figure 3-2 Example of Assignment of Applications through Cgroup](image-url)
Creating Groups for Use with Cgroup

How to use cgroup to set up a group in which the Cortex-A57 cores handle tasks as the “big” group is described below.

Step 1: Confirming the CPUs that have been booted up
Execute the following command to confirm the CPU number and type (Cortex-A57 or Cortex-A53).

```
$ cat /proc/cpuinfo
/* The types of CPU can be confirmed from the CPU part in the result of
execution as follows. Cortex-A57 -> 0xd07 and Cortex-A53 -> 0xd03 */
```

Step 2: Creating the “big” group
Execute the following command to create the “big” group.

```
$ mkdir /sys/fs/cgroup/cpuset/big
```

Step 3: Setting the memory node
The memory node is a parameter for use in setting the assignment of memory to the cgroup. If you do not wish to specify the assignment of memory, set it to 0 as in our example.

```
$ echo 0 > /sys/fs/cgroup/cpuset/big/cpuset.mems
```

Step 4: Assigning CPU resources to the groups
Register the CPU numbers for the Cortex-A57 processors confirmed in step 1 with the “big” group.

- Registration for the R-Car H3

```
$ echo 0-3 > /sys/fs/cgroup/cpuset/big/cpuset.cpus
/* For the R-Car H3, the Cortex-A57s have been registered with the CPU
numbers 0 to 3. */
```

- Registration for the R-Car M3-W/ R-Car M3-W+

```
$ echo 0-1 > /sys/fs/cgroup/cpuset/big/cpuset.cpus
/* For the R-Car M3-W, R-Car M3-W+, the Cortex-A57s have been registered
with the CPU numbers 0 and 1. */
```
3.2.1.1.2 Assigning Applications for Use with Cgroup

How to assign applications to the “big” group created on the previous page is described below.

Step 1: Confirming the PIDs of applications
Execute the following command to confirm the PIDs for applications to be assigned with cgroup.

```bash
$ ps
/* Confirm the PID lines in the result of executing the command. */
```

Step 2: Assigning the applications
Execute the following command to assign applications to the “big” group. Replace [PID] in the command with the PIDs for the applications to be assigned before execution.

- Registration of assignment to the “big” group

```bash
$ echo [PID] > /sys/fs/cgroup/cpuset/big/tasks
```

Executing the following command releases the settings which had been made for assignment to the “big” group.

- Release from assignment to the “big” group

```bash
$ echo [PID] > /sys/fs/cgroup/cpuset/tasks
```
3.2.1.2 Procedure for Setting taskset

taskset is used to apply CPU affinity to processes in progress.
CPU affinity is a property of the scheduler and is used to link processes to specific CPUs in a system.
This is used to specify a CPU for each of the processes that are in progress.
To use taskset, taskset must be added when the file system is configured.
For details, see appendix A4.

A specified process is made to run on a specified CPU by executing the following command.

```bash
$ taskset -pc X [process ID]
/* X is replaced with the CPU number (0, 1, 2, ...). */
```
3.2.2 Measure in the Form of Making Processes Realtime

Processor affinity can be used to prevent task switching unexpectedly assigning processes that are intended to be assigned solely to a Cortex-A57 to a Cortex-A53 through the method described in section 3.2.1, Processor Affinity as a Measure. Moreover, changing the attribute of a task to a realtime process is a further way of giving higher priority to the execution of the process.

Table 3-2 shows the method of making a process realtime, taking chrt as an example.

### Table 3-2 Example of Setting the Priority Level for Realtime Processes

<table>
<thead>
<tr>
<th>Outline</th>
<th>Details</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Setting the priority level for a realtime</td>
<td>Give a unique priority level to a realtime process before executing it.*</td>
<td>chrt -r X [process name]</td>
</tr>
<tr>
<td>process and executing it</td>
<td></td>
<td>X represents a priority level in the range from 0 to 99. Specify the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name of the process to be run with that priority level in [process</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name]. Example: The command for starting the /usr/local/bin/sample</td>
</tr>
<tr>
<td></td>
<td></td>
<td>process with the priority level specified as 99 is as follows. chrt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-r 99 /usr/local/bin/sample</td>
</tr>
</tbody>
</table>

Note: * Use root access to do this.
4. Disabling CAS Functionality

If you wish to confirm the effect of applying CAS or CAS is not required, you can disable the CAS functionality.

To disable the CAS functionality, disable the big.LITTLE architecture.

For the concrete details of the method, see the description titled “Step 6 change the bootargs by U-Boot” in section 5, Confirm starting of U-Boot and Linux, of RENESAS_RCH3M3M3NE3_YoctoStartupGuide_UME_v3.7.0 (Linux Interface Specification Yocto recipe Start-Up Guide).
Appendix

A1. Times for Task Switching between the Cortex-A57 and Cortex-A53 Cores

Table A1-1 shows the times for task switching.

<table>
<thead>
<tr>
<th>Straddling of clusters</th>
<th>Cortex-A57 -&gt; Cortex-A57</th>
<th>Cortex-A57 -&gt; Cortex-A53</th>
<th>Cortex-A53 -&gt; Cortex-A57</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same cluster, so none</td>
<td>14 µs</td>
<td>19 µs</td>
<td>21 µs</td>
</tr>
</tbody>
</table>

<Measurement environment>

- Software: Yocto v3.7.0 (Linux BSP 3.6.2)
- SoC: Version 3.0 of the R-Car H3 on a Salvator-XS board
- Measurement method
  1. Awaken any executable process (process A) which endlessly loops on CPU1.
  2. Use the taskset command* to switch the CPU on which process A runs to CPU2.

Note: * taskset is a command used to change the settings for assigning processes to CPUs.

---

1. CPU1 is running process A.

   ![Run queue](CPU1)  
   Run queue  
   A

   ![Run queue](CPU2)  
   Run queue  
   CPU 2

2. Process A is suspended.

   ![Run queue](CPU1)  
   Run queue  
   A

   ![Run queue](CPU2)  
   Run queue  
   CPU 2

3. The CPU to run process A is switched to CPU2.

   ![Run queue](CPU1)  
   Run queue  
   CPU 1

   ![Run queue](CPU2)  
   Run queue  
   CPU 2

4. Process A is awakened on CPU2.

   ![Run queue](CPU1)  
   Run queue  
   CPU 1

   ![Run queue](CPU2)  
   Run queue  
   CPU 2

---

Figure A1-1 Flow of Task Switching
A2. Building an Environment for htop

To use htop, edit the given file in the way shown below and rebuild the file system.

```
[Add the following line to build/conf/local.conf.]

CORE_IMAGE_EXTRA_INSTALL += "htop"
```

Figure A2-1   Adding htop
A3. Building an Environment for ftrace

(1) Enabling ftrace
To enable ftrace, make the following setting in the kernel configuration.

```
        [ ] Tracers       -->   *
        [*]   Kernel Function Tracer   *
        [*]   Kernel Function Graph Tracer   *
```

**Figure A3-1 Kernel Configuration**

Note: * Check these boxes to enable ftrace.

(2) Installing trace-cmd in the file system.
To use trace-cmd, edit the given file in the way shown below and rebuild the file system.

```
[Add the following line to build/conf/local.conf.]

CORE_IMAGE_EXTRA_INSTALL += "trace-cmd"
```

**Figure A3-2 Adding trace-cmd**

(3) Installing KernelShark
If you intend to run KernelShark on the Linux host PC, use apt-get or your preferred commands to install KernelShark on the Linux host PC.
A4. Building an Environment for taskset

To use taskset, edit the given file in the way shown below and rebuild the file system.

```
[Add the following line to build/conf/local.conf.]

CORE_IMAGE_EXTRA_INSTALL += "util-linux"
```

Figure A4-1 Adding taskset
## Revision History

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>June, 2018</td>
<td></td>
<td>First edition issued</td>
</tr>
<tr>
<td>1.01</td>
<td>August, 2019</td>
<td>1</td>
<td>Purpose, Target Readers and Target Device updated (&quot;R-Car M3-W+&quot;)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>Assigning CPU resources to the groups updated. (&quot;R-Car M3-W+&quot;)</td>
</tr>
</tbody>
</table>
General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit 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.

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

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

3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the 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.

5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, 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.

6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between \( V_{IL}(\text{Max.}) \) and \( V_{IH}(\text{Min.}) \) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between \( V_{IL}(\text{Max.}) \) and \( V_{IH}(\text{Min.}) \).

7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of 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.
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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving 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, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.

3. 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 shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.

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

*Standard*: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

*High Quality*: Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.

8. Please contact a Renesas Electronics sales office for details as to environment matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technologies shall 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 shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.

11. This document shall not be reprinted, 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.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rv.4.0-1  November 2017)