R-Car S4 Series

Guide to Accelerating Data Transfer through the PCIe Interfaces of the R-Car S4

Introduction

The R-Car S4 incorporates a PCIe controller which supports two PCIe serial interface channels, each with two lanes, conformant with the PCI Express 4.0 standard, with both providing endpoint and root complex functionalities. Attempting to create software in a Linux environment without taking bottlenecks for the transfer processing into account may lead to not even half of the transfer capability of the hardware being brought out. This application note describes how to create software which brings out the high transfer capability of the hardware by describing countermeasures for bottlenecks in the transfer processing.

Target Board

R-Car S4 evaluation board RTP8A779F0ASKB0SC1S or RTP8A779F0ASKB0SB0S
We simply refer to it as “evaluation board” in this document.
Contents

1. Overview ......................................................................................................................... 4
   1.1 Purpose ......................................................................................................................... 4

2. Contents of the Sample Program Package ........................................................................ 5

3. Environment for Confirming Operation ............................................................................ 7

4. Overview of the Software ................................................................................................. 8

5. Configuration of Software ............................................................................................... 10
   5.1 Application (for Data Transmission) .............................................................................. 10
   5.2 PCIe Data Transmitting Device Driver ........................................................................ 10
   5.3 PCIe EP Driver ............................................................................................................. 11
   5.4 Application (for Data Reception) .................................................................................. 11
   5.5 PCIe Data Receiving Device Driver ............................................................................. 11
   5.6 PCIe Channels and Device Trees .................................................................................. 11

6. Details of the Software .................................................................................................... 12
   6.1 Policy for Accelerating the PCIe Data Transfer Rate ...................................................... 12
   6.1.1 Eliminating Copying of the Contents of Memory ....................................................... 13
   6.2 Ways to Accelerate the PCIe Data Transfer Rate .......................................................... 15
   6.2.1 Accelerating the Detection of PCI Commands by the Endpoint Device .................... 16
   6.2.2 Allocating the Memory Area for Transfer .................................................................. 17
   6.2.2.1 kmalloc Function ................................................................................................. 17
   6.2.2.2 dma_alloc_coherent Function ........................................................................... 17
   6.2.2.3 memremap Function ............................................................................................ 18
   6.2.3 Elimination of Copying the Contents of Memory between the Device Driver and Application ................................................................. 19
   6.2.4 Cache Control by the Application .......................................................................... 21
   6.2.5 Management of Buffers for Use in the Shared Memory .............................................. 22
   6.2.6 Running Elements of Data Processing by the Application in Parallel ........................ 23

7. How to Build from the Source Code .............................................................................. 25
   7.1 Building the Basic Linux .............................................................................................. 26
   7.2 Building the Linux for Use with the Root Complex ...................................................... 27
   7.3 Building the Linux for Use with the Endpoint ............................................................... 28
   7.4 Building the SDK ......................................................................................................... 29
   7.5 Building and Installing the Application Programs ........................................................ 30

8. Executing the Sample Program ....................................................................................... 31
   8.1 Setting up the Terminal Software ............................................................................... 31
   8.2 Setting up U-Boot ......................................................................................................... 31
   8.3 Execution ..................................................................................................................... 32
   8.4 Results for the Performance in Transfer ....................................................................... 33
   8.5 Specifications of the pcpiperf Command ..................................................................... 34
   8.6 Operations of the Transmitting (EP) Device Driver ...................................................... 35
The following registered trademarks and trademarks of third parties, as well as company names, system names, and product names mentioned in this document are generally registered trademarks or trademarks of their respective companies. ™ mark and ® mark for companies trademark or registered trademark are omitted in this document.

* Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. (https://www.arm.com/ja/company/policies/trademarks)
1. Overview

1.1 Purpose

Creating application software involving data transfer conformant with the PCI Express 4.0 standard in a Linux environment requires taking bottlenecks that lower the performance of processing in terms of the device driver, application, and their interfaces into account. Attempting to create software without taking the bottlenecks into account may lead to not even half of the transfer capability of the hardware being brought out. This application note is intended to help users in developing software for high-speed data transfer through PCIe interfaces by giving examples of countermeasures for accelerating transfer between two R-Car S4 evaluation boards connected with each other through an OCuLink cable to bring out the high transfer capability of the hardware. The following figure is an overview of the sample system.

![Figure 1 Overview of the Sample System](image-url)
2. Contents of the Sample Program Package

This section describes the contents of the sample program used in this application note. Download it from the following page on the Renesas Official Web site.

Renesas Official Web site

The following shows the software package required for the software configuration described in section 4, Overview of the Software. It includes the modules framed by red rectangles in the following figure.

![Sample Program Package for PCIe Data Transfer between Two R-Car S4 Devices](image)

The following is a list of modules the package contains.

<table>
<thead>
<tr>
<th>Board</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation board [1]</td>
<td>Application</td>
<td>Application software for both transmitting and receiving data</td>
</tr>
</tbody>
</table>
| Evaluation board [1]                 | PCIe data transmitting device driver | Device driver for data transmission
|                                      |                               | Generation of this driver is based on the pci-epf-test.c file which comes with the Linux kernel. |
| Evaluation board [1]                 | Patches for the OS            | Acceleration patches for use with the PCIe endpoint                        |
| Evaluation board [2]                 | Application                   | Application software for both transmitting and receiving data               |
| Evaluation board [2]                 | PCIe data receiving device driver | Device driver for data reception
|                                      |                               | Generation of this driver is based on the pci_endpoint_test.c file which comes with the Linux kernel. |
| Evaluation board [2]                 | Patches for the OS            | Patches for use with the PCIe root complex                                  |
The folders of this package are structured as follows.

<table>
<thead>
<tr>
<th>app/</th>
</tr>
</thead>
<tbody>
<tr>
<td>pciperf/ Application on evaluation boards</td>
</tr>
<tr>
<td>(common to evaluation boards 1 and 2)</td>
</tr>
<tr>
<td>os</td>
</tr>
<tr>
<td>meta-patch-pcie Patches for the PCIe drivers</td>
</tr>
<tr>
<td>meta-patch-pcie-ep Patch for the PCIe endpoint settings</td>
</tr>
<tr>
<td>meta-pcikmod Device drivers for PCIe transmission and reception</td>
</tr>
</tbody>
</table>

Figure 3  Structure of Folders in the Package
3. Environment for Confirming Operation

The following lists the items of the operating environment in which operation of the sample program was confirmed.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (CA55)</td>
<td>Arm Cortex-A55 (CA55) incorporated in the R-Car S4. Operating frequency: 1.2 GHz</td>
</tr>
<tr>
<td>Evaluation board</td>
<td>Two R-Car S4 evaluation boards</td>
</tr>
<tr>
<td>OCuLink cable</td>
<td>CBL-SAST-0818 from Super Micro Computer, Inc.</td>
</tr>
<tr>
<td>Boot loader</td>
<td>ICUMX Loader Rev.0.16.0</td>
</tr>
<tr>
<td>Development environment</td>
<td>OS: Ubuntu 20.04 LTS (64 bits)</td>
</tr>
<tr>
<td></td>
<td>CA55 compiler: gcc version 9.3.0</td>
</tr>
<tr>
<td>Linux</td>
<td>RCar XOS SDK-Gateway V3.9</td>
</tr>
</tbody>
</table>

The following shows the connections between the evaluation boards and between the respective boards and PCs for the use of serial terminals.

The following shows the settings of SW1, SW2, and switches on the switch board. These settings are common to both boards.
4. Overview of the Software

In creating an application for data transfer which runs under Linux, it will generally include the procedures described below (see the figure below).

- Using the device driver to dynamically allocate the DMA transfer area
- Transferring data to the allocated area through DMA transfer
- Copying the contents of memory between the device driver in the kernel space and the application program in the user space

![Read and Write Processing between the Device Driver and Application](image)

Figure 6  Read and Write Processing between the Device Driver and Application

With this procedure\(^1\), however, accelerating transfer by hardware may lower the performance of the system as a whole due to the times taken by software processing, such as for copying the contents of memory, dynamically allocating buffers, and access by the CPU to the non-cacheable area. Specifically, with two lanes conformant with the PCI Express 4.0 standard, transfer by hardware at high speed (32 GT/s) may create a problem by leading to the causes of lowering performance described above. In the sample program used with this application note, the memory area for use with DMA transfer is shared between the device driver and application, which eliminates the need for processing to copy the contents of memory. In addition, data caching is enabled for the memory area for use with DMA transfer so that the application program can apply cache control. The sample program also handles the reception processing and memory operation, including cache control for the received data, in parallel by using different threads for these types of processing, which enables the further acceleration of performance. The figure on the following page gives an overview of the operations of the applications.

\(^1\) Procedure described in the *R-Car S4 Series Guide to Transferring Data through PCIe Interfaces in the R-Car S4* application note
Figure 7  Overview of Application Operations
5. Configuration of Software

The software for data transmission consists of the application, PCIe data transmitting device driver, and Linux. On the other side, the software for data reception consists of the application, PCIe data receiving device driver, and Linux. The following shows the configuration of the software layers.

![Software Configuration Diagram](image)

**Figure 8**  Configuration of Software

### 5.1 Application (for Data Transmission)

After Linux on the data transmitting side has started up, it needs to make the settings for the PCIe to function as an endpoint device, such as the vendor ID, device ID, and PCIe channel to be used, before activating the application program (pciperf). In this sample program, the ep_start.sh script can be used to make these settings for the PCIe to function as an endpoint device (see section 8.3, Execution). The application for data transmission uses the API functions for use in management of the shared memory and control over transmission which are provided by the PCIe data transmitting device driver to transfer data to the data receiving evaluation board.

### 5.2 PCIe Data Transmitting Device Driver

The PCIe data transmitting device driver (sendpci.ko) operates as a PCI function driver which allows the exclusive use of the given device for data transmission as an PCI endpoint device. The PCIe data transmitting device driver is incorporated in the kernel by executing the ep_start.sh script. The driver provides the API functions for use in management of the shared memory and control over transmission to the application. In addition, the driver needs to respond to the PCI commands which are irregularly transmitted by the root complex, so always monitors for PCI commands by using an internally activated kernel thread. The driver generates /dev/sendpci0 as an interface with the application.
5.3 PCIe EP Driver

The PCIe EP driver provides the standard API functions for the PCIe endpoint device incorporated in the Linux kernel and controls PCI resources with the use of sysfs, the interface for access by the user process to the information on the kernel and device driver. The PCIe EP driver consists of the following three components and provides the API functions for the host function driver.

1) API functions for use in control over the PCIe controller for the endpoint such as interrupt control
2) API functions for use in control over the endpoint functionality such as control over the PCI configuration
3) Control over the PCI resources with the use of sysfs such as setting the vendor ID and device ID

5.4 Application (for Data Reception)

The application program for data reception (pciperf) repeats the reception of data being transferred through PCIe transfer and the disabling of caches and outputs the reception rate to the console window at a regular interval. The program also checks the contents of the transferred data according to the specified option.

5.5 PCIe Data Receiving Device Driver

The PCIe data receiving device driver provides the API functions for use in management of the shared memory and control over reception to the application. The PCIe data receiving device driver issues PCI transfer request commands to the data transmitting (endpoint) device in response to transfer requests from the application. The driver generates /dev/receivepci0 as an interface with the application.

5.6 PCIe Channels and Device Trees

The R-Car S4 incorporates a PCIe controller which supports the PCIe interface with two lanes × two channels. This sample program only uses channel 0 of the two PCIe channels. Using channel 1 requires changes to the contents of the endpoint and root complex device trees. Note that the numbers of the PCIe channels to be used have no relation to those of the device driver file names (/dev/sendpci0 and /dev/receivepci0). The numbers of the device driver files are always set to 0.
6. Details of the Software

For details on the basic processing for PCIe data transfer, refer to the *R-Car S4 Series Guide to Transferring Data through PCIe Interfaces in the R-Car S4* application note under References in this application note. The subsections below and on the following pages give detailed descriptions of PCIe data transfer in terms of acceleration of the transfer rate.

6.1 Policy for Accelerating the PCIe Data Transfer Rate

The most important point in raising the PCIe data transfer rate is as far as possible to suppress the time lags from completion of a PCIe data transfer by hardware until the next data transfer starts. To start with, the various types of processing required for PCIe data transfer are listed in the table below.

<table>
<thead>
<tr>
<th>Table 3</th>
<th>Various Types of Processing Required for PCIe Data Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCIe endpoint side</td>
<td>PCIe root complex side</td>
</tr>
<tr>
<td>&lt;1&gt; Allocating the buffer for transmission</td>
<td></td>
</tr>
<tr>
<td>&lt;2&gt; Data processing</td>
<td></td>
</tr>
<tr>
<td>&lt;3&gt; Copying data to the DMA buffer</td>
<td>&lt;a&gt; Allocating the buffer for reception</td>
</tr>
<tr>
<td>&lt;4&gt; Waiting for the PCI command and starting transfer</td>
<td>↔ &lt;b&gt; Issuing the PCI data transfer request command</td>
</tr>
<tr>
<td></td>
<td>↔ &lt;c&gt; Waiting for the data transfer completion interrupt</td>
</tr>
<tr>
<td>&lt;5&gt; Waiting for completion of data transfer</td>
<td></td>
</tr>
<tr>
<td>&lt;6&gt; Freeing the buffer for transmission</td>
<td>&lt;d&gt; Copying data from the DMA buffer to the buffer for reception</td>
</tr>
<tr>
<td></td>
<td>&lt;e&gt; Data processing</td>
</tr>
<tr>
<td></td>
<td>&lt;f&gt; Freeing the buffer for reception</td>
</tr>
</tbody>
</table>

Note that “↔” in the table indicates that synchronization between the processing on the endpoint and root complex sides is required.

PCIe transfer of 2 Mbytes of data in two lanes at 16 GT/s requires 650 μs (the theoretical value). As this is with the use of DMA transfer by hardware, the load on the CPU is not heavy. Therefore, steps to make various types of processing in the table other than <4>, <5>, <b>, and <c> run in parallel during transfer can be expected to accelerate the transfer rate.
6.1.1 Eliminating Copying of the Contents of Memory

Making the two types of processing to copy the contents of memory, <3> Copying data to the DMA buffer and <3> Copying data from the DMA buffer to the buffer for reception in Table 3 run in parallel, requires the loss of synchronization of these types of processing and transfer processing. This might lead to the processing becoming more complicated. In addition, as using software to copy 2 Mbytes of data in memory takes approximately 1 ms (the measured value), the data transfer time becomes shorter than the time to copy the contents of memory. Accordingly, copying the contents of memory in parallel is not applicable as a countermeasure. For these reasons, we stopped the use of the read and write functions, which would generally proceed between the device driver and application, and employ the use of the shared memory instead of copying the contents of memory. The figure below shows a schematic view of the two types of procedure, that is, with the use of the read and write functions and with the use of the shared memory, respectively, in data transfer between the kernel and user process.

![Figure 9 Schematic View of the Two Types of Procedure for Data Transfer between the Kernel and User Process](image-url)
For the procedure with the use of the shared memory, mapping the buffer area allocated in the kernel space to the user space enables direct handling of the buffer by the user process, which eliminates the need to copy the contents of memory. The figure below shows a schematic view of the processing times when the reception processing incorporates use of the read function and when copying of the contents of memory is eliminated. Note that the results in the figure are on the assumptions that detecting the transfer request command on the endpoint side takes no time and data are always ready for transmission on the endpoint side in response to a transfer request. The conditions for transfer are as follows.

- PCIe transfer with two lanes, each at 16 GT/s
- Amount of data in one round of DMA transfer: 2 Mbytes
- Period of copying the contents of memory: Measured value
- Period of DMA transfer: Theoretical value

![Schematic view of the processing times when the reception processing incorporates use of the read function](image1)

![Schematic view of the processing times when the reception processing incorporates use of the shared memory](image2)

**Figure 10** Schematic View of the Processing Times for the Procedures with Use of the read Function and with Shared Memory

As shown in the figure above, simply eliminating copying of the contents of memory can be expected to considerably improve the transfer rate.
6.2 Ways to Accelerate the PCIe Data Transfer Rate

Although not mentioned in the previous subsection, the PCIe endpoint device runs software processing specifically to monitor PCI commands. As the root complex device uses a PCI command to request a transfer, quick detection of the PCI command by the endpoint device is also an important point in raising the transfer rate. The six approaches to acceleration based on the policy described in the previous subsection, and which are applied in this sample program, are listed below.

1. Accelerating the detection of PCI commands by the endpoint device
2. Allocating the memory area for transfer
3. Elimination of copying the contents of memory between the device driver and application
4. Cache control by the application
5. Management of buffers for use in the shared memory
6. Running elements of data processing by the application in parallel
6.2.1 Accelerating the Detection of PCI Commands by the Endpoint Device

The PCIe endpoint device needs to respond to each PCI command specified by the PCIe root complex device. Though some PCIe endpoint devices, such as graphics cards and network cards, can detect a PCI command at high speed with the use of a hardware circuit, an endpoint device that is virtualized in software must always monitor PCI commands. This sample program uses the device driver to activate a kernel thread and monitors PCI commands through a loop process within the thread. Such a case requires ensuring that the load of the loop process does not affect the other processes. One way to suppress the effects on the other processes is to use a sleep-system function in time.c of the kernel to place the loop process in the sleep state at a regular interval. This method, however, incurs a time lag during the period over which the process is in the sleep state. This sample program uses the schedule function in core.c, which handles the scheduling of the kernel, to give higher priority to the detection of PCI commands. The schedule function gives higher priority to another task to be processed and handles the detection of a PCI command when the task schedule is empty. This enables reducing the time lag of detection while suppressing the effects on the other processes. The listing below shows an example of detecting a PCI command (excerpt from sendpci.c).

Note that this processing is only for the endpoint side, that is, the processing is not required on the receiving (root complex) side.

Note that using the schedule function is effective in detecting a PCI command at high speed because this sample program only handles the transfer processing. If another process with a heavy load exists, return from the schedule function is delayed, which may make the transfer rate unstable. In addition, as this approach has the CPU continually operating, the load on the CPU tends to become heavier than that when the sleep-system function is used. If these factors create a problem, replacing the schedule function with the usleep_range function may improve the stability of the rate and lower the CPU load factor. The usleep_range function controls the wait time by using the high-precision timer functionality employed in Linux kernel 2.6 and later versions. The timers are referred to as hrtimers (high-resolution kernel timers). Though this sample program uses the schedule function by default, dynamically switching to the usleep_range function is possible (see section 8.6, Operations of the Transmitting (EP) Device Driver). The usleep_range function has two parameters, the minimum and maximum wait times. Specify these times in μs units. As the values of these parameters depend on the state of the system, adjust them in an experimental way. The top or vmstat command can be used to confirm the load on the system.

```
static int epf_kthread(void *arg)
{
    u32 command;
    struct pci_epf_priv *priv = (struct pci_epf_priv *)arg;
    enum pci_barno sendpci_reg_bar = priv->sendpci_reg_bar;
    volatile struct sendpci_reg *reg = priv->reg[sendpci_reg_bar];

    while(!kthread_should_stop()) {
        command = reg->command;
        if (command & (COMMAND_WRITE|COMMAND_RAISE_MSI_IRQ|COMMAND_RAISE_MSIX_IRQ)) {
            sendpci_cmd_handler(priv);
        } else {
            schedule();
        }
    }
    return 0;
}
```

The sendpci_cmd_handler function actually handles the transmission processing in response to detection of the COMMAND_WRITE PCI transfer command.

Or the `usleep_range` function

The `usleep_range` function controls the wait time by using the high-precision timer functionality employed in Linux kernel 2.6 and later versions. The timers are referred to as hrtimers (high-resolution kernel timers). Though this sample program uses the schedule function by default, dynamically switching to the `usleep_range` function is possible (see section 8.6, Operations of the Transmitting (EP) Device Driver). The `usleep_range` function has two parameters, the minimum and maximum wait times. Specify these times in μs units. As the values of these parameters depend on the state of the system, adjust them in an experimental way. The top or vmstat command can be used to confirm the load on the system.
6.2.2 Allocating the Memory Area for Transfer

The Linux kernel has multiple functions for allocating the memory area for use with DMA transfer. Each of the functions is described below and on the following page.

6.2.2.1 kmalloc Function

The kmalloc function has been part of the Linux kernel for a long time and guarantees the allocation of contiguous physical memory areas. Note that repeatedly allocating and freeing contiguous memory areas may lead to failure to allocate the areas. This function is suitable for allocating relatively small memory areas.

6.2.2.2 dma_alloc_coherent Function

The dma_alloc_coherent function dynamically allocates memory areas from the CMA (contiguous memory allocator) memory area controlled by Linux. As the name implies, this function also allocates contiguous physical memory areas, so is usable in allocating memory for DMA transfer, frame buffers, etc. Using the dma_free_coherent function to free an allocated memory area allows use of the given area by another process. In such cases, settings related to the CMA memory in the device tree and of the CMA driver in the kernel may be required before using a freed area.
6.2.2.3 memremap Function

The memremap function maps a memory area that is not under the control of the OS so that it is accessible from the kernel space. The function can also make the setting to enable or disable cache control for the given memory area. When the physical address of the area to be used as a DMA buffer is known in advance, mapping the area to the kernel space is relatively easy. Therefore, the memremap function is used in this sample program. The listing below shows an example of code for using the probe function of the device driver to allocate a buffer for use in data transfer (excerpt from sendpci.c).

```c
static int sendpci_probe(struct pci_epf *epf)
{
    struct pci_epf_priv *priv;

    priv = devm_kzalloc(dev, sizeof(*priv), GFP_KERNEL);
    if (!priv)
        return -ENOMEM;

    priv->dev = &epf->dev;
    epf->header = &sendpci_header;
    priv->epf = epf;
    priv->phys_addr = PHYS_BUFADR;
    priv->phys_size = sizeof(struct bufctl_t);
    priv->bufctl = (struct bufctl_t *)memremap(
        priv->phys_addr, priv->phys_size, MEMREMAP_WB);

    priv->phys_addr: Fixed to 0x4000_0000
    priv->phys_size: 2 Mbytes × 4 + α (for use in management)
    MEMREMAP_WB: Enabling caching
```

Registration of the endpoint device driver with the kernel automatically generates a virtual directory in sysfs. The sendpci_probe function is called when the directory which represents the PCI function is created in the virtual directory. The concrete operation is as follows.

```
# modprobe sendpci
# mkdir /sys/kernel/config/pci_ep/functions/sendpci/func1
```

However, although the probe function of the receiving (root complex) device driver handles the processing with the use of the memremap function in the same way as on the endpoint side, in the case of the root complex side, systemd-udevd, a user-resident process, automatically incorporates an appropriate device driver based on the PCI device information. The probe function is called at this time, so the operation for the endpoint side described above is not required.
6.2.3 Elimination of Copying the Contents of Memory between the Device Driver and Application

Enabling operations of the memory area for transfer from the user space which was allocated in the previous subsection allows the elimination of copying the contents of memory. This requires mapping the memory area for transfer allocated by the device driver to the user virtual space. The `remap_pfn_range` function of the Linux kernel is used to handle the mapping. The listing below shows an example of code to handle mapping to the user space (excerpt from `sendpci.c`).

```c
static int sendpci_mmap(struct file *file, struct vm_area_struct *vma)
{
    struct pci_epf_priv *priv = file->private_data;
    unsigned long pfn_start = (priv->phys_addr >> PAGE_SHIFT) + vma->vm_pgoff;
    unsigned long size = vma->vm_end - vma->vm_start;

    if (file->f_flags & O_SYNC) {
        vma->vm_page_prot = pgprot_noncached(vma->vm_page_prot);
    }

    remap_pfn_range(vma, vma->vm_start, pfn_start, size, vma->vm_page_prot);
    return 0;
}
```

The `sendpci_mmap` function is called when the `mmap` function is called by the application. The virtual address information for the size specified in the `mmap` function of the application is set in the `vma` argument. Note that though the virtual address information will have been set in the `vma` argument on calling the `sendpci_mmap` function, the virtual and actual addresses have not yet been mapped. Accordingly, the `remap_pfn_range` function must be used to map these addresses to the user space. The variables and constants that this procedure includes are listed in the table below.

<table>
<thead>
<tr>
<th>Variable or Defined Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vma-&gt;vm_start</td>
<td>Virtual address where the user space starts</td>
</tr>
<tr>
<td>vma-&gt;vm_end</td>
<td>Virtual address where the user space ends</td>
</tr>
<tr>
<td>vma-&gt;pgoff</td>
<td>Offset to the start address (in pages)</td>
</tr>
<tr>
<td></td>
<td>Offset specified with the <code>mmap</code> function of the application (in pages)</td>
</tr>
<tr>
<td>vma-&gt;vm_page_prot</td>
<td>Specifying the attribute for use in mapping</td>
</tr>
<tr>
<td>priv-&gt;phys_addr</td>
<td>Physical address where the memory for use in transfer starts</td>
</tr>
<tr>
<td></td>
<td>0x4000_0000 for this sample program</td>
</tr>
<tr>
<td>pfn_start</td>
<td>Frame number of the page where the physical address range starts</td>
</tr>
<tr>
<td></td>
<td>The page frame number is the result of right-shifting an address value by the number of places specified as <code>PAGE_SHIFT</code>, that is, that of converting the address value into page units. The page frame number can be used to obtain the physical address where the page starts through the kernel's internal page table. A page table is provided for the kernel space and per process in the user space. When the process context is switched, the virtual space is switched by switching within the page table. In addition, the exception level is switched from EL0 (user) to EL1 (OS) in response to a mode transition from user to kernel due to a system call or for some other reason. This switches the TTBR (translate table base register), which indicates the page table, from the TTBR for EL0 to the TTBR for EL1, thus automatically switching to the kernel space. Note that the MMU handles conversion between the virtual and physical addresses at high speed. Moreover, the continuity of the physical addresses in the virtual address range that was mapped to the kernel space by using the <code>memremap</code> function, as described in the previous subsection, is guaranteed. Accordingly, contiguous ranges of physical pages are</td>
</tr>
<tr>
<td>size</td>
<td>Size of memory to be mapped</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>PAGE_SHIFT</td>
<td>The page size is 4 Kbytes, so the number of places for shifting is 12.</td>
</tr>
</tbody>
</table>

mapped to the user space in order, starting with the value specified in `pfn_start`. 
6.2.4 Cache Control by the Application

Handling cached memory areas and then synchronizing the contents of the memory and cache may lead to better performance than handling non-cached memory areas. The following are the measured results for processing times depending on cache control when the memset function is used to clear an 8-Mbyte memory area.

<table>
<thead>
<tr>
<th>Status</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>With the cache disabled</td>
<td>12.37 ms</td>
</tr>
<tr>
<td>With the cache enabled</td>
<td>3.99 ms</td>
</tr>
</tbody>
</table>

Measurement was with the clock_gettime function of R-Car S4 Linux.

The cache synchronization processing for the 8-Mbyte memory area with the cache enabled requires approximately 0.4 ms. Even if this time is added to the result of measurement with the cache enabled, the two measured results indicate that cache control by the application is more efficient.

The Linux kernel does not have a function that allows control over caching for a specific memory area by a device driver. Therefore, creating a function for use in control over caching in the device drivers for both transmission and reception is required so that the ioctl function can be used for the control in user mode.

Note that uiomem ([https://github.com/ikwzm/uiomem](https://github.com/ikwzm/uiomem)) was used for reference in setting up the cache control processing. The listings below are for the cache control processing.

### Flushing the cache (excerpt from sendpci.c)

```c
static inline void aarch64_flush(void* start, size_t size)
{
    u64 vaddr = (u64)start;
    u64 __end = (u64)start + size;
    u64 cache_line_mask = cl_size - 1;
    vaddr &= ~cache_line_mask;
    while (vaddr < __end) { 
        asm volatile("dc cvac, %0" : : "r"(vaddr) : );
        vaddr += cl_size;
    }
    asm volatile("dsb sy" : : : );
}
```

cl_size: Cache line size = 64

### Disabling the cache (excerpt from receivepci.c)

```c
static inline void aarch64_invalidate(void* start, size_t size)
{
    u64 vaddr = (u64)start;
    u64 __end = (u64)start + size;
    u64 cache_line_mask = cl_size - 1;
    if (__end & cache_line_mask) != 0) {
        __end &= ~cache_line_mask;
        asm volatile("dc civac, %0" : : "r"(__end) : );
    }
    if ((vaddr & cache_line_mask) != 0) {
        vaddr &= ~cache_line_mask;
        asm volatile("dc civac, %0" : : "r"(vaddr) : );
    }
    while (vaddr < __end) { 
        asm volatile("dc ivac, %0" : : "r"(vaddr) : );
        vaddr += cl_size;
    }
    asm volatile("dsb sy" : : : );
}
```

cl_size: Cache line size = 64
6.2.5 Management of Buffers for Use in the Shared Memory

Securing multiple buffers in units of the data transfer size enables operation in parallel between the device driver and application or between application threads. In addition, if a buffer where the data processing has been completed is prepared in advance on the endpoint device, it can start the transfer processing immediately upon detecting a PCI transfer request command from the root complex side. The figure below is a schematic view of the management of the buffers for use in the shared memory.

![Diagram of parallelization and buffer management](image)

**Figure 11 Schematic View of the Parallelization of Processing through the Use of the Shared Memory**

On the endpoint side, processing in parallel can be realized by using different buffers for a thread for the data processing, which operates in the user space, and a thread for the transmission processing by the device diver, which operates in the kernel space (see section 6.2.1, Accelerating the Detection of PCI Commands by the Endpoint Device). When one round of transfer processing is completed, the individual buffers enter the states indicated by 90-degree rotations within the circle in the figure above and handling proceeds accordingly. Switching the buffers requires synchronization processing. To synchronize the threads in the user and kernel spaces, the wait_event_interruptible and wake_up_interruptible functions of the kernel are used. As a process in the user space is not capable of directly calling these functions, the synchronization processing proceeds via the ioctl function.

On the root complex side, processing in parallel can be realized by preparing threads for the reception processing and the data processing, both of which run in the user space, and using different buffers for these threads. When one round of transfer processing is completed, the individual buffers enter the states indicated by 90-degree rotations within the circle in the same way as on the endpoint side. To synchronize the threads in the user space, the pthread_mutex- and pthread_cond-system functions are used (refer to the fifo.c file that comes with this application note). Note that the figure above shows an example on the assumption that the buffers on the endpoint side are full so that data can be processed immediately after completion of transfer on the root complex side because sufficient spare processing capacity is available. To simplify the management of the buffers on the endpoint side, an idle buffer is included in the illustration in spite of the buffers being full.
6.2.6 Running Elements of Data Processing by the Application in Parallel

Accelerating PCIe data transfer requires taking the time for processing data into account. The application module on the receiving side employs the procedure described in section 6.2.5, Management of Buffers for Use in the Shared Memory to run the reception processing and the data processing in parallel. As this sample program does not run data processing, only the cache control processing is run in parallel with the reception processing. Disabling the cache for a 2-Mbyte memory area takes approximately 100 μs. The time for PCIe transfer with two lanes, each at 16 GT/s, is approximately 650 μs. Accordingly, even running only the cache control processing in parallel will raise the rate by over 10%. The figure below is a schematic view of the processing for PCIe transfer in 2-Mbyte units, with two lanes, and each at 15 GT/s.

![Schematic View of the Times for Sequential Processing and Parallel Processing](image)

The figure above is a schematic view of the processing on the receiving (root complex) side. The receiving device waits for the transfer completion interrupt for approximately 650 μs after the PCIe transfer request command having been set. The parallel data processing thread handles no processing after controlling the cache until the next transfer has been completed, so can be said to be available as a spare thread.
The listings below show examples of the threads for handling the reception and the received data.

**Processing of reception (excerpt from pciperf.c)**

```c
static void do_recv(void)
{
    int fd;
    uint32_t *buf = NULL;
    pthread_t tid;

    fd = open ("/dev/receivepci0", O_RDWR); // Open the device driver.
    pcieif_init(fd); // Initialize the driver's wrapper function.
    fifo_init(); // Initialize a buffer for use in communications between threads.
    // Activate a thread for handling received data.
    pthread_create(&tid, NULL, thread_recv_dataproc, NULL);

    while(!g_terminated) {
        pcieif_receive((void**)&buf); // Wait for reception of PCIe data.
        fifo_put(buf); // Transfer the contents of the buffer to the thread for handling data.
    }

    pthread_join(tid, NULL);
    close(fd);
}
```

**Processing of the received data (excerpt from pciperf.c)**

```c
static void *thread_recv_dataproc(void *param)
{
    while(!g_terminated) {
        void *ptr;

        fifo_get(&ptr); // Receive the data from the thread for data reception.
        pcieif_cache_invalidate(ptr, DATA_SIZE); // Control the cache.

        /* The data processing is to be written here. */
        fifo_release(); // Free the buffer for use in communications between threads.
        pcieif_release_recvbuff(); // Use the pcieif_receive function to free the secured buffer.
    }

    return NULL;
}
```

The descriptions in this subsection are applicable to the processing on the PCIe receiving (root complex) side. On the other hand, parallel processing is not employed on the PCIe transmitting (endpoint) side. The kernel thread for the transmitting device driver handles the transmission processing asynchronously, which is equivalent to parallel processing. Accordingly, having cache control by the user application run in parallel only has a small effect. This is why parallel processing is not employed on the PCIe transmitting (endpoint) side.
7. **How to Build from the Source Code**

Build the executable module under Ubuntu by following the procedure below.

- **Procedure for building the executable module**
  1. Build the basic Linux.
  2. Build the Linux for use with the root complex.
  3. Build the Linux for use with the endpoint.
  4. Build the SDK.
  5. Build and install the application programs.
7.1 Building the Basic Linux

Use build_yocto.sh included in version 3.9 of the software development kit (SDK) provided by Renesas to build the Linux OS.

Environment:

<table>
<thead>
<tr>
<th>Host PC</th>
<th>Recommended OS: Ubuntu 20.04 LTS (64 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The host must be connected to the Internet.</td>
</tr>
<tr>
<td></td>
<td>Free disk space: At least 100 Gbytes</td>
</tr>
</tbody>
</table>

It can be deployed in any directory; the following shows an example of deployment in the 

$HOME/rcars4sdk39 directory.

$ mkdir $HOME/rcars4sdk39

$ cd $HOME/rcars4sdk39

$ sudo apt-get update

$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath socat cpio python3 python3-pip python3-pexpect xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libstdc++10-dev pylint3 xterm libarchive-zip-perl

# Copy the build_yocto.sh file to the current directory.

$ ./build_yocto.sh spider gateway

# Building may take several hours. When building of the basic Linux is completed normally,
# the kernel image, root file system, and device tree will have been generated in the

Building of the basic Linux is completed at this point
7.2 Building the Linux for Use with the Root Complex

The Linux for the R-Car S4 is generated by following the procedure described in section 7.1, Building the Basic Linux. After that, rebuild the Linux for use with the root complex from the basic Linux, which involves incorporating the configuration of the PCIe root complex and the device driver. The procedure for rebuilding is continued from section 7.1, Building the Basic Linux.

$ cd $HOME/rcars4sdk39

Deploy the patches in the RCarS4_PCIe_FastTransfer_Sample.zip file of the sample software package.
$ unzip [...]/RCarS4_PCIe_FastTransfer_Sample.zip 'os/*'
$ mv os/* .
$ rmdir os/

$ source poky/oe-init-build-env build-spider-gateway

$ vi conf/bblayers.conf

Set the BBLAYERS variables by editing the text file as follows.

```
BBLAYERS ?= " 
	$(TOPDIR)/../poky/meta 
	$(TOPDIR)/../poky/meta-poky 
	$(TOPDIR)/../poky/meta-yocto-bsp 
	$(TOPDIR)/../meta-renesas/meta-rcar-gateway 
	$(TOPDIR)/../meta-openembedded/meta-oe 
	$(TOPDIR)/../meta-openembedded/meta-python 
	$(TOPDIR)/../meta-openembedded/meta-networking 
	$(TOPDIR)/../meta-patch-pcie 
	$(TOPDIR)/../meta-pci-kmod 
"
```

$ vi conf/local.conf

Add the following line at the end of local.conf.

```
IMAGE_INSTALL_append = " kernel-module-receivepci"
```

Note: Insert a space before kernel-module-receivepci.

$ bitbake rcar-image-gateway

The building of the Linux for use with the PCIe root complex is completed at this point.

After the successful completion, the root file system will have been generated in the following file.
tmp/deploy/images/spider/rcar-image-gateway-spider.tar.bz2

Deploy the generated root file system to an eMMC on the R-Car S4 evaluation board for data reception (R-Car S4 evaluation board [2] in Figure 7, Overview of Application Operations) and then activate Linux.
7.3 Building the Linux for Use with the Endpoint.

The following shows the procedure for building the Linux for use with the PCIe endpoint from the Linux for use with the PCIe root complex. The procedure for building is continued from section 7.2, Building the Linux for Use with the Root Complex.

$ vi conf/bblayers.conf

Set the BBLAYERS variables by editing the text file as follows.

```
BBLAYERS ?= " \\
    ${TOPDIR}/../poky/meta \\
    ${TOPDIR}/../poky/meta-poky \\
    ${TOPDIR}/../poky/meta-yocto-bsp \\
    ${TOPDIR}/../meta-renesas/meta-rcar-gateway \\
    ${TOPDIR}/../meta-openembedded/meta-oe \\
    ${TOPDIR}/../meta-openembedded/meta-python \\
    ${TOPDIR}/../meta-openembedded/meta-networking \\
    ${TOPDIR}/../meta-patch-pcie \\
    ${TOPDIR}/../meta-pciisol \\
    ${TOPDIR}/../meta-pci-mod \\
    ${TOPDIR}/../meta-pci-ep \\
    ${TOPDIR}/../meta-pci-ep \\
" \\
```

Insert one blank line.

$ vi conf/local.conf

Modify the end of local.conf.

```
# IMAGE_INSTALL_append = " kernel-module-receivepci"
IMAGE_INSTALL_append = " kernel-module-sendpci"
IMAGE_INSTALL_append = " kernel-module-sendpci-dev"
```

Note: Insert a space before kernel-module-....

$ bitbake rcar-image-gateway

The building of the Linux for use with the PCIe endpoint is completed at this point.

After the successful completion, the root file system will have been generated in the following file.

tmp/deploy/images/spider/rcar-image-gateway-spider.tar.bz2

Deploy the generated root file system to an eMMC on the R-Car S4 evaluation board for data transmission (R-Car S4 evaluation board [1] in Figure 7, Overview of Application Operations) and then activate Linux.
7.4 Building the SDK

Build the Yocto SDK from the Linux for use with the PCIe endpoint. As the code of the application program refers to the header file (sendpci.h) of the device driver for the PCIe endpoint, the SDK must be built from the corresponding build of Linux. The procedure for building is continued from section 7.3, Building the Linux for Use with the Endpoint.

$ cd $HOME/rcars4sdk39

$ source poky/oe-init-build-env build-spider-gateway

$ bitbake rcar-image-gateway -c populate_sdk

# Building may take several hours.

# Install the built SDK.
$ sudo ./tmp/deploy/sdk/poky-glibc-x86_64-rcar-image-gateway-aarch64-spider-toolchain-3.1.11.sh

# A message “Enter target directory for SDK”, which requires entering the destination for installation of the SDK, appears during the installation process.
# If the default directory setting /opt/poky/3.1.11 does not create a problem, press the Enter key without changing it.

Building of the SDK is completed at this point, as is the preparation for compiling the application software which runs under Linux on the R-Car S4.
7.5 Building and Installing the Application Programs

Use the SDK generated in section 7.4, Building the SDK to generate the application programs. The following describes the procedures for generating and installing the application programs.

Deploy the applications in the RCarS4_PCIe_FastTransfer_Sample.zip file of the sample software package. They can be deployed in any directory; the following shows an example of deployment in the

$HOME/rcars4app directory.

$ mkdir $HOME/rcars4app
$ unzip {...}/RCarS4_PCIe_FastTransfer_Sample.zip ‘app/*’
$ mv app/* .
$ rmdir app/
$ ls -F
pciperf/ ; The pciperf directory is output.

Set up the SDK environment.

Specify the directory specified as the destination for installation of the SDK in section 7.4, Building the SDK.

The following command is with the default setting.

$ source /opt/poky/3.1.11/environment-setup-aarch64-poky-linux

Building and installing the transmission application

$ cd pciperf
$ make

After successful completion of the process, the pciperf file will have been generated in the build/bin/ directory.

Copy the generated pciperf file to /home/root/ on both evaluation boards (evaluation boards [1] and [2] in Figure 7, Overview of Application Operations).

The following is an example with scp in use.

$ cd build/bin
$ scp pciperf root@[IP address of the R-Car S4 evaluation board [1]]:
$ scp pciperf root@[IP address of the R-Car S4 evaluation board [2]]:

Building and installing of the application programs have been completed at this point.
8. Executing the Sample Program

8.1 Setting up the Terminal Software

Use the TeraTerm terminal software to operate the Linux console.

The following lists the settings of the serial port.

<table>
<thead>
<tr>
<th>Item</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit rate</td>
<td>1,843,200 bps</td>
</tr>
<tr>
<td>Data</td>
<td>8 bits</td>
</tr>
<tr>
<td>Parity</td>
<td>None</td>
</tr>
<tr>
<td>Stop bit</td>
<td>1 bit</td>
</tr>
<tr>
<td>Flow control</td>
<td>None</td>
</tr>
</tbody>
</table>

8.2 Setting up U-Boot

As this sample program uses the memory area starting at 0x4000_0000 for PCIe data transfer, the amount of memory under Linux control on each of the two evaluation boards is limited to 1 Gbyte. A boot parameter of the Linux kernel is used to make the setting to limit the memory area under Linux control. Setting the boot parameter proceeds in the U-Boot mode. The procedure for setting the boot parameter is given below.

Connect the CN20 connector on the evaluation board with the PC through a USB serial cable. For the settings of the serial port, see Table 5, Serial Port Settings. To enter the U-Boot mode, press the Enter key repeatedly in the TeraTerm window immediately after turning on the power supply to the R-Car S4 evaluation board.

After entry to the U-Boot, the ‘=>’ prompt will appear.

Modify the bootargs environment variable for U-Boot. Note that the name of the variable may differ depending on the environment: use the correct variable name for your environment.

=> editenv bootargs

After entry to the edit mode, add the following text at the end of the line. As shown below, insert a space before this “parameter = setting”.

␣ mem=1G

=> saveenv Saves the edited settings in the flash ROM.
=> run bootcmd Activates Linux.

Linux will be activated.

After logging in as the root user, the output of “mem=1G” at the end of the line produced in response to

# cat /proc/cmdline

will indicate the successful completion of these operations.

The memory area under Linux control is now limited to 1 Gbyte.
8.3 Execution

Set up the environment as described in section 3, Environment for Confirming Operation.

Supply power to the R-Car S4 evaluation board for data transmission (evaluation board [1] in Figure 7, Overview of Application Operations) and activate Linux. Do not supply power to evaluation board [2] at this point in time.

Log in as the root user through the terminal.

```
# ep_start.sh  ; Execute the script.
```

When the operation has proceeded normally, the following messages will be output.

```
sendpci_init: finished
sendpci_probe: finished
sendpci_bind: finished
```

Supply power to the R-Car S4 evaluation board for data reception (evaluation board [2] in Figure 7, Overview of Application Operations) and activate Linux.

Log in as the root user through the terminal.

```
# lspci  ; Confirm that the following line is output.
01:00.0 Unassigned class [ff00]: Renesas Technology Corp. Device 1234
```

```
# ./pciperf -rm  ; Execute the program for testing the performance in PCIe transfer.
```

Through the terminal window of the R-Car S4 evaluation board for data transmission (evaluation board [1] in Figure 7, Overview of Application Operations)

```
# ./pciperf -sm  ; Execute the program for testing the performance in PCIe transfer.
```

In the case of normal operation, data transfer will start and data will be output in the following format to the terminal windows of both evaluation boards at a regular interval.

```
<Cumulative times of transfer>: <Transfer rate> MB/s (<Transfer rate> Gbps)
```

Note that the measured transfer rates are the averages of results from 1000 rounds of transfer.

Entering ctrl+c stops the program.
8.4 Results for the Performance in Transfer

The results of executing the `pciperf` command are given below.

Arguments of the `pciperf` command (for PCIe data transfer with the use of the read and write functions)

On the receiving (root complex) side: `-r`

On the transmitting (endpoint) side: `-s`

Example of output:

5000: 1141 MB/s (9.13 Gbps)
6000: 1138 MB/s (9.11 Gbps)
7000: 1138 MB/s (9.10 Gbps)
8000: 1138 MB/s (9.10 Gbps)

Arguments of the `pciperf` command (for PCIe data transfer with the use of the shared memory)

On the receiving (root complex) side: `-rm`

On the transmitting (endpoint) side: `-sm`

Example of output:

5000: 2770 MB/s (22.16 Gbps)
6000: 2696 MB/s (21.57 Gbps)
7000: 2823 MB/s (22.58 Gbps)
8000: 2799 MB/s (22.39 Gbps)

Summary of results

<table>
<thead>
<tr>
<th></th>
<th>GB/s</th>
<th>Gbps</th>
<th>Efficiency of Transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCIe with two lanes, each at 16 GT/s (theoretical value)</td>
<td>3.2</td>
<td>25.6</td>
<td>–</td>
</tr>
<tr>
<td>Without the &quot;m&quot; option for <code>pciperf</code></td>
<td>1.14</td>
<td>9.1</td>
<td>36%</td>
</tr>
<tr>
<td>With the &quot;m&quot; option for <code>pciperf</code></td>
<td>2.77</td>
<td>22.2</td>
<td>87%</td>
</tr>
</tbody>
</table>

Note that the application does not apply parallelization of the processing to the data processing on the receiving side in PCIe data transfer with the use of the read and write functions. In addition, the output results are not uniform and will sometimes change depending on the environment where the command is being executed.
8.5 Specifications of the pciperf Command

Format

pciperf [option]

Description

The pciperf command is a program for measuring the capability in PCIe data transfer between the two R-Car S4 evaluation boards connected with each other through an OCuLink cable. The amount of data in one round of transfer is two Mbytes. The program always controls the cache for data being transferred, but only proceeds with data processing if the data-checking option [-c] is specified.

-s: Data transmission mode
Specify this option on the endpoint side.

-r: Data reception mode
Specify this option on the root complex side.

-c: Check the data.
When the “c” option is specified, the command checks each 2 Mbytes of transfer data. In the data transmission mode, it sets the number of PCIe data transfers – 1 in the 32 bits from which the data area starts and then the value of these bits + 1 in the next 32 bits. After this, it repeats setting the value of the given bits + 1 in the following 32 bits (an example of data starting from the tenth transfer: 0x0000_0009, 0x0000_000a, 0x0000_000b, …). In the data reception mode, the “c” option selects comparison of the received values with those that were set in the transmission mode. On detection of a non-match, it outputs the information on the non-match. This software processing is applied across the whole areas, which lowers the performance in transfer. When data checking is to be applied, specify the “c” option on both the root complex and endpoint sides. When the “c” option is not specified, the command does not proceed with data checking but only proceeds with cache control for data transfer.

-m: Transfer with the use of the shared memory
When the “m” option is specified, the command uses the shared memory for transfer instead of the read and write functions. In the data transmission mode, it flushes the cache for data to be transferred before the transmission processing. In data reception mode, it disables the cache after receiving data. When the “m” option is not specified, transfer proceeds with the use of the read and write functions, which entails copying of the contents of memory.
8.6 Operations of the Transmitting (EP) Device Driver

A function to be used in waiting within the loop processing in the monitoring of PCI commands by the transmitting (EP) device driver is directly specifiable. The virtual files of the transmitting (EP) device driver are described in the table below.

Table 6 Files in the /sys/kernel/sendpci/ Directory

<table>
<thead>
<tr>
<th>File Name</th>
<th>Range of Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>usleep</td>
<td>0 or 1</td>
<td>Either of the following functions is used in the loop processing. Default setting = 0. 0: schedule; 1: usleep_range</td>
</tr>
<tr>
<td>usleep_min</td>
<td>From 1 (μs)</td>
<td>The minimum wait time to be specified as the first argument of the usleep_range function. Default setting = 10. Note that this is only effective when usleep = 1. As the schedule function when usleep = 0 has no arguments, any setting is ignored.</td>
</tr>
<tr>
<td>usleep_max</td>
<td>From 1 (μs)</td>
<td>The maximum wait time to be specified as the second argument of the usleep_range function. Default setting = 100. Note that this is only effective when usleep = 1. As the schedule function when usleep = 0 has no arguments, any setting is ignored.</td>
</tr>
</tbody>
</table>

Referring to the parameter (an example of referring to a value for usleep)

```
# cat /sys/kernel/sendpci/usleep
0
```

Setting the parameter (an example of using the usleep_range function)

```
# echo 1 > /sys/kernel/sendpci/usleep
```

All parameters are reflected in the operation immediately after they have been set.
References

For the following document, obtain the latest version from the Renesas Web site.

- R-Car S4 Series User’s Manual: Hardware

For the following documents, obtain SDK 3.9 from the R-Car Market Place or your local Renesas Electronics sales representative.

- R-Car S4 Series Linux Interface Specification Yocto recipe Start-Up Guide
- R-Car V3U/S4 Series Linux Interface Specification Device Driver PCIEC

Obtain the following document from the R-Car Market Place or your local Renesas Electronics sales representative.

- R-Car S4 Series Guide to Transferring Data through PCIe Interfaces in the R-Car S4 Application Note
### Revision History

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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. Renesas Electronics Corporation shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a product that uses a Renesas Electronics product. RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.

4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.

5. 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.

6. 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.

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. Although Renesas Electronics products are subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, 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 and impractical, you are responsible for evaluating 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. 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 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. You are responsible for ensuring that your use or sale 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.

Notice

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

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.