Abstract
This document describes an example of setting items required when using the direct memory access controller (hereinafter called as "DMAC") of the RZ/A1H.

Products
RZ/A1H

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.
Contents

1. Specifications .......................................................................................................................... 4
2. Operation Confirmation Conditions ....................................................................................... 5
3. Reference Application Notes ................................................................................................ 5
4. Peripheral Functions .............................................................................................................. 6
5. Hardware ............................................................................................................................... 7
   5.1 Pins Used .......................................................................................................................... 7
6. Software .................................................................................................................................. 8
   6.1 Operation Overview ............................................................................................................ 8
      6.1.1 Sample Code 1 ........................................................................................................... 8
      6.1.2 Sample Code 2 ........................................................................................................... 9
   6.2 Peripheral Function Settings and Memory Allocation Used in Sample Code .................... 10
      6.2.1 Peripheral Function Settings ..................................................................................... 10
      6.2.2 Section Assignment of Sample Code .......................................................................... 11
   6.3 Interrupts ........................................................................................................................... 14
   6.4 Fixed-Width Integers .......................................................................................................... 14
   6.5 Constants ........................................................................................................................... 15
   6.6 Structure/Union List .......................................................................................................... 16
   6.7 Variables ............................................................................................................................ 17
   6.8 Functions ............................................................................................................................ 19
   6.9 Function Specifications ...................................................................................................... 20
   6.10 Flowcharts ....................................................................................................................... 28
      6.10.1 Main Processing ........................................................................................................ 28
      6.10.2 Sample Code Main Processing ................................................................................ 29
      6.10.3 DMAC Sample Main Function .................................................................................. 30
      6.10.4 Sample Function with Auto-Request as DMA Transfer Request Type .................. 31
      6.10.5 Sample Function with Auto-Request as DMA Transfer Request Type (Continuous Transfer) ................. 33
      6.10.6 DMAC Initial Setting Function when Using Auto-Request as DMA Transfer Request Type ......................... 35
      6.10.7 DMAC Channel 3 Initial Setting Function when Using Auto-Request as DMA Transfer Request Type .......... 36
      6.10.8 DMA Transfer Enable Function ............................................................................. 38
      6.10.9 DMA Transfer Abort Function ............................................................................... 39
      6.10.10 DMA Transfer Information Setting Function ......................................................... 40
      6.10.11 DMA Transfer Information Reload Setting Function ........................................ 41
      6.10.12 DMAC Channel 3 Transfer Enable Function ......................................................... 42
      6.10.13 DMAC Channel 3 Transfer Abort Function .......................................................... 43
      6.10.14 DMAC Channel 3 Setting Function ....................................................................... 44
      6.10.15 DMAC Channel 3 Reload Setting Function ......................................................... 45
      6.10.16 DMAC Channel 3 Transfer interrupt Function ................................................... 46
      6.10.17 Initial Setting Function for Protection Unit Support and Cache Support .................. 47

R01AN1703EJ0101 Rev.1.01
Aug. 28, 2015
6.10.18  Get Function of Setting Information to Protection Unit Support and Cache Support...... 50
6.11  Running the Sample Code............................................................................................................ 55

7.  Notes on Usage of Sample Code........................................................................................................ 56
7.1  Coherency between L1 Caches and External Memory or Large-Capacity On-Chip RAM .... 56
7.2  Notes on DMA Transfer with Speculative Cache Line Fill............................................................. 56

8.  Sample Code..................................................................................................................................... 57

9.  Reference Documents......................................................................................................................... 57
1. Specifications

The DMAC is used to transfer data from the large-capacity on-chip RAM space to the SDRAM space connected to the CS2 space.

The DMA transfers are performed with the large-capacity on-chip RAM space set as the transfer source and the CS2 SDRAM space set as the transfer destination, respectively. The memory management unit (hereinafter called as "MMU") is set to make the large-capacity on-chip RAM space (H'2000 0000 to H'209F FFFF) enable the L1 cache, and the CS2 SDRAM space (H'0800 0000 to H'0BFF FFFF) enable the L1 and L2 caches, respectively.

In this application note, the interrupt controller and the serial communication interface with FIFO are referred to as INTC and SCIF respectively.

Table 1.1 lists the Peripheral Functions and Their Applications.

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAC</td>
<td>Used to transfer data from the large-capacity on-chip RAM space to the SDRAM space allocated to the CS2 space</td>
</tr>
<tr>
<td>INTC</td>
<td>Used for DMAC interrupt control</td>
</tr>
<tr>
<td>SCIF</td>
<td>Used in communication between the SCIF channel 2 and the host PC</td>
</tr>
</tbody>
</table>
2. Operation Confirmation Conditions
The sample code accompanying this application note has been run and confirmed under the conditions below.

Table 2.1 Operation Confirmation Conditions

<table>
<thead>
<tr>
<th>Item</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU used</td>
<td>RZ/A1H</td>
</tr>
<tr>
<td>Operating frequency*</td>
<td>CPU clock (Iφ): 400 MHz</td>
</tr>
<tr>
<td></td>
<td>Image processing clock (Gφ): 266.67 MHz</td>
</tr>
<tr>
<td></td>
<td>Internal bus clock (Bφ): 133.33 MHz</td>
</tr>
<tr>
<td></td>
<td>Peripheral clock 1 (P1φ): 66.67 MHz</td>
</tr>
<tr>
<td></td>
<td>Peripheral clock 0 (P0φ): 33.33 MHz</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>Power supply voltage (I/O): 3.3 V</td>
</tr>
<tr>
<td></td>
<td>Power supply voltage (internal): 1.18 V</td>
</tr>
<tr>
<td>Integrated development</td>
<td>ARM® integrated development environment</td>
</tr>
<tr>
<td>environment</td>
<td>ARM Development Studio 5 (DS-5™) Version 5.16</td>
</tr>
<tr>
<td>C compiler</td>
<td>ARM C/C++ Compiler/Linker/Assembler Ver.5.03 [Build 102]</td>
</tr>
<tr>
<td></td>
<td>Compiler Option (excluding additional directory path)</td>
</tr>
<tr>
<td></td>
<td>-O3 -Os -cpu=Cortex-A9 --littleend --arm --apcs=/interwork</td>
</tr>
<tr>
<td></td>
<td>--no_unaligned_access --fpu=vfpv3_fp16 -g -asm</td>
</tr>
<tr>
<td>Operating mode</td>
<td>Boot mode 0 (CS0 space 16-bit boot)</td>
</tr>
<tr>
<td>Terminal software</td>
<td>Communication speed: 115200bps</td>
</tr>
<tr>
<td>communication settings</td>
<td>Data length: 8 bits</td>
</tr>
<tr>
<td></td>
<td>Parity: None</td>
</tr>
<tr>
<td></td>
<td>Stop bit length: 1 bit</td>
</tr>
<tr>
<td></td>
<td>Flow control: None</td>
</tr>
<tr>
<td>Board used</td>
<td>GENMAI board</td>
</tr>
<tr>
<td></td>
<td>RTK772100BC00000BR (R7S72100 CPU board)</td>
</tr>
<tr>
<td>Device used</td>
<td>NOR flash memory (Connected to CS0 and CS1 spaces)</td>
</tr>
<tr>
<td></td>
<td>Manufacturer: Spansion Inc.</td>
</tr>
<tr>
<td></td>
<td>Product No.: S29GL512S10TFI01</td>
</tr>
<tr>
<td></td>
<td>SDRAM (Connected to CS2 and CS3 spaces)</td>
</tr>
<tr>
<td></td>
<td>Manufacturer: ISSI Inc.</td>
</tr>
<tr>
<td></td>
<td>Part No.: IS42S16320B-75</td>
</tr>
<tr>
<td></td>
<td>Serial interface (9-pin D-Sub connector)</td>
</tr>
<tr>
<td></td>
<td>LED1</td>
</tr>
</tbody>
</table>

Note: * The operating frequency used in clock mode 0 (Clock input of 13.33MHz from EXTAL pin)

3. Reference Application Notes
For additional information associated with this document, refer to the following application notes.

- RZ/A1H Group I/O definition header file <iodefine.h> (R01AN1860EJ)
- RZ/A1H Group Example of Initialization (R01AN1864EJ)
4. Peripheral Functions

This chapter provides supplementary information on the DMA. Refer to the RZ/A1H Group User's Manual (Hardware) for basic information.

The DMAC of the RZ/A1H supports a register mode and a link mode. This chapter describes the register mode used in the sample code.

In the register mode, the DMA transfers are performed using the values specified to the internal registers of the DMAC. Next0 Register Set and Next1 Register Set are provided to set the transfer source address, transfer destination address, and the transfer byte count. The register set should be selected by setting the RSEL bit in the Channel Configuration Register n to execute the DMA transfer. The value of the RSEL bit is automatically inverted when the DAM transfer ends after the RSW bit in the Channel Configuration Register n is set to "1". Therefore, continuous DMA transfers can be performed by using these register sets.

![Figure 4.1 Register Configuration in DMAC Register Mode](image-url)
5. Hardware

5.1 Pins Used

Table 5.1 lists the Pins Used and Their Functions.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>I/O</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>A25 to A1</td>
<td>Output</td>
<td>Address signal output to the NOR flash memory and the SDRAM</td>
</tr>
<tr>
<td>D15 to D0</td>
<td>Input/output</td>
<td>Data signal input/output of the NOR flash memory and the SDRAM</td>
</tr>
<tr>
<td>CS0#</td>
<td>Output</td>
<td>Device select signal output to the NOR flash memory connected to CS0 space</td>
</tr>
<tr>
<td>CS1#</td>
<td>Output</td>
<td>Device select signal output to the NOR flash memory connected to CS1 space</td>
</tr>
<tr>
<td>CS2#</td>
<td>Output</td>
<td>Device select signal output to the SDRAM connected to CS2 space</td>
</tr>
<tr>
<td>CS3#</td>
<td>Output</td>
<td>Device select signal output to the SDRAM connected to CS3 space</td>
</tr>
<tr>
<td>RD#</td>
<td>Output</td>
<td>Read control signal output to the NOR flash memory</td>
</tr>
<tr>
<td>WE0#/DQMLL#</td>
<td>Output</td>
<td>Write enable control signal output to the NOR flash memory and byte select (D7 to D0) control signal output to the SDRAM</td>
</tr>
<tr>
<td>DQMLU#</td>
<td>Output</td>
<td>Byte select (D15 to D8) control signal output to the SDRAM</td>
</tr>
<tr>
<td>RD/WR#</td>
<td>Output</td>
<td>Read control signal or write control signal output to the SDRAM</td>
</tr>
<tr>
<td>CKE</td>
<td>Output</td>
<td>CK enable control signal output to the SDRAM</td>
</tr>
<tr>
<td>CAS#</td>
<td>Output</td>
<td>CAS# control signal output to the SDRAM</td>
</tr>
<tr>
<td>RAS#</td>
<td>Output</td>
<td>RAS# control signal output to the SDRAM</td>
</tr>
<tr>
<td>MD_BOOT1</td>
<td>Input</td>
<td>Selection of boot mode</td>
</tr>
<tr>
<td>MD_BOOT0</td>
<td>Input</td>
<td>MD_BOOT1: &quot;L&quot;, MD_BOOT0: &quot;L&quot; (Set to boot mode 0)</td>
</tr>
<tr>
<td>P4_10</td>
<td>Output</td>
<td>LED ON/OFF</td>
</tr>
<tr>
<td>RxD2</td>
<td>Input</td>
<td>Serial receive data signal</td>
</tr>
<tr>
<td>TxD2</td>
<td>Output</td>
<td>Serial transmit data signal</td>
</tr>
</tbody>
</table>

Note: The symbol # indicates negative logic (or active low).
6. Software

6.1 Operation Overview

The Data located in the large-capacity on-chip RAM space is transferred to the CS2 SDRAM space using channel 3 of the DMAC. The DMA mode is set to register mode, and auto-request (software-activated) transfer requests are used to perform three data transfers (of 4,096 bytes, 1,024 bytes, and 256 bytes, for a total of 5,376 bytes). The transfer source transfer size and transfer destination transfer size are set to 8 bits. Two types of the DMA transfer sample code are provided.

6.1.1 Sample Code 1

The DMAC is activated every time a data transfer ends to perform a total of three DMA transfers. The DMAC transfer-end interrupt handler sets the interrupt notification flag. In sample code 1, the interrupt notification flag is polled after the DMAC is activated to confirm the end of DMA transfer. When the interrupt notification flag is set, the information for the next DMA transfer is specified to activate the DMAC.

Figure 6.1 shows the Operation Sequence of Sample Code 1.

![Operation Sequence of Sample Code 1](image-url)
6.1.2 Sample Code 2

The continuous transfer function of the DMAC is used to perform a total of three sequential DMA transfers by activating the DMAC once. Information for the first and the second DMA transfers should be specified before activating the DMAC. The interrupt handler for end of DMA transfer sets the interrupt notification flag. In the sample code 2, when the first DMA transfer ends after activating the DMAC, the second DMA transfer is continuously performed. The interrupt notification flag is polled to confirm the end of the first DMA transfer and the start of the second DMA transfer. When the interrupt notification flag is set, the information of the third DMA transfer is set in the process of the second DMA transfer.

Figure 6.2 shows the Operation Sequence of Sample Code 2.

![Figure 6.2 Operation Sequence of Sample Code 2](image)

Because the REN bit is set to “1”, DMAC activation by software is not necessary. DMA transfers can be performed continuously.
6.2 Peripheral Function Settings and Memory Allocation Used in Sample Code

6.2.1 Peripheral Function Settings
Table 6.1 lists the settings when the sample code is executed.

<table>
<thead>
<tr>
<th>Module</th>
<th>Settings</th>
</tr>
</thead>
</table>
| DMAC   | Channel 3  
DMA mode: Register mode  
Transfer request type: Auto-request (software-activated)  
Transfer mode: Block transfer  
DMA interrupt requests are used.  
Buffer data is not swept when DMA is halted.  
Transfer source address: Large-capacity on-chip RAM space  
Transfer destination address: SDRAM space  
Transfer address count direction: Incrementing of both transfer source and transfer destination  
Total transfer byte count: Three transfers of 4096, 1024, and 256 bytes, respectively  
Transfer size: 8 bits for both transfer source and transfer destination  
Priority control: Round-robin mode  
In the sample code 2, the continuous transfer function is used. |
6.2.2 Section Assignment of Sample Code

Table 6.2 and Table 6.3 list the Sections Used in the sample code, and Figure 6.3 shows the sample code section assignment in the initial state (load view) and the one after the scatter loading function is used (execution view).

For details about sections and the scatter loading function, refer to "Image structure and generation" in ARM Compiler toolchain: Using the Linker.

<table>
<thead>
<tr>
<th>Area Name</th>
<th>Description</th>
<th>Type</th>
<th>Load Area</th>
<th>Execution Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>VECTOR_TABLE</td>
<td>Exception processing vector table</td>
<td>Code</td>
<td>FLASH</td>
<td>FLASH</td>
</tr>
</tbody>
</table>
| RESET_HANDLER      | Program code area of reset handler processing  
• INITCA9CACHE (L1 cache setting)  
• INIT_TTB (MMU setting)  
• RESET_HANDLER (Reset handler) | Code  | FLASH     | FLASH          |
| CODE_BASIC_SETUP   | Program code area to optimize operating frequency and flash memory                                                                                                                                               | Code  | FLASH     | FLASH          |
| InRoot             | This area consists of the sections located in the root area such as C standard library.                                                                                                                                 | Code and RO Data | RO Data | FLASH          |
| CODE_FPU_INIT      | Program code area for NEON and VFP initializations  
• CODE_FPU_INIT  
• FPU_INIT | Code  | FLASH     | FLASH          |
| CODE_RESET         | Program code area for hardware initialization  
• CODE_RESET (Startup processing)  
• INIT_VBAR (Vector base setting) | Code  | FLASH     | FLASH          |
| CODE_IO_REGRW      | Program code area for read/write functions of I/O register                                                                                                                                                     | Code  | FLASH     | LRAM           |
| CODE               | Program code area for defaults  
All the Code type sections which do not define section names with C source are assigned in this area.                                                                                                        | Code  | FLASH     | FLASH          |
| CONST              | Constant data area for defaults  
All the RO Data type sections which do not define section names with C source are assigned in this area.                                                                                          | RO Data | FLASH     | FLASH          |
### Table 6.3 Sections Used (2/2)

<table>
<thead>
<tr>
<th>Area Name</th>
<th>Description</th>
<th>Type</th>
<th>Load Area</th>
<th>Execution Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>VECTOR_MIRROR_TABLE</td>
<td>Exception processing vector table (Section to transfer data to large-capacity on-chip RAM)</td>
<td>Code</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>CODE_HANDLER_JMPTBL</td>
<td>Program code area for user-defined functions of IRQ interrupt handler</td>
<td>Code</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>CODE_HANDLER</td>
<td>Program code area of IRQ interrupt handler</td>
<td>Code</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td></td>
<td>This area consists of the following sections.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• CODE_HANDLER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• IRQ_FIQ_HANDLER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CODE_CACHE_OPERATION</td>
<td>Program code area for setting the L1 and L2 caches*3</td>
<td>Code</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>DATA_HANDLER_JMPTBL</td>
<td>Registration table data area for user-defined functions of IRQ interrupt handler</td>
<td>RW Data</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td>ARM_LIB_STACK</td>
<td>Application stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>IRQ_STACK</td>
<td>IRQ mode stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>FIQ_STACK</td>
<td>FIQ mode stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>SVC_STACK</td>
<td>Supervisor (SVC) mode stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>ABT_STACK</td>
<td>Abort (ABT) mode stack area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>TTB</td>
<td>MMU translation table area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>ARM_LIB_HEAP</td>
<td>Application heap area</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td>DATA</td>
<td>Data area with initial value for defaults</td>
<td>RW Data</td>
<td>FLASH</td>
<td>LRAM</td>
</tr>
<tr>
<td></td>
<td>All the RW Data type sections which do not define section names with C source are assigned in this area.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSS</td>
<td>Data area without initial value for defaults</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td></td>
<td>All the ZI Data type sections which do not define section names with C source are assigned in this area.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSS_DMAC_SAMPLE_INTERNAL_RAM</td>
<td>Data area without initial value used by DMAC sample code</td>
<td>ZI Data</td>
<td>-</td>
<td>LRAM</td>
</tr>
<tr>
<td></td>
<td>Defined to be assigned to the L1 cache-enabled area.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSS_DMAC_SAMPLE_SDRAM</td>
<td>Data area without initial value used by DMAC sample code (used as DMAC transfer destination area)</td>
<td>ZI Data</td>
<td>-</td>
<td>SDRAM</td>
</tr>
<tr>
<td></td>
<td>Defined to be assigned to the L1 and L2 cache-enabled area.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
1. In the listing of load areas and execution areas in the table, "FLASH", "LRAM", and "SDRAM" indicate NOR flash memory area, large-capacity on-chip RAM area, and CS2 SDRAM area respectively.
2. The section names basically are the same as the area names, but the RESET_HANDLER, InRoot, CODE_FPU_INIT, CODE_RESET, CODE, CONST, CODE_HANDLER, DATA, and BSS areas each comprise multiple sections. For details on areas and sections, refer to the ARM compiler toolchain manual.
3. This section should be placed in the cache-disabled area.
Cache-disabled space in large-capacity on-chip RAM (10MB)

Section assignment (load view)

Memory allocation after scatter loading

Section assignment (execution view)

Transfer cache operation process to cache-disabled area in on-chip RAM

Clear to 0

Initialize data with initial values

Obtain areas for stacks and the like

Initialize data with initial values

Transfer program code that requires high-speed processing to the on-chip RAM

Transfer exception processing vector to the on-chip RAM

Figure 6.3 Section Assignment
6.3 Interrupts
Table 6.4 lists the interrupt Used in Sample Code.

Table 6.4 interrupt Used in Sample Code

<table>
<thead>
<tr>
<th>Interrupt Source (Interrupt ID)</th>
<th>Priority</th>
<th>Processing Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAINT3 (44)</td>
<td>1</td>
<td>Generates an interrupt when a data transfer ends on the DMAC channel 3 and sets the interrupt notification flag.</td>
</tr>
</tbody>
</table>

6.4 Fixed-Width Integers
Table 6.5 lists the Fixed-Width integers Used in Sample Code.

Table 6.5 Fixed-Width integers Used in Sample Code

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char_t</td>
<td>8-bit character</td>
</tr>
<tr>
<td>bool_t</td>
<td>Boolean type, value: true (1) or false (0)</td>
</tr>
<tr>
<td>int_t</td>
<td>High-speed integer, signed 32-bit integer in this sample code</td>
</tr>
<tr>
<td>int8_t</td>
<td>8-bit integer, signed (Defined by standard library)</td>
</tr>
<tr>
<td>int16_t</td>
<td>16-bit integer, signed (Defined by standard library)</td>
</tr>
<tr>
<td>int32_t</td>
<td>32-bit integer, signed (Defined by standard library)</td>
</tr>
<tr>
<td>int64_t</td>
<td>64-bit integer, signed (Defined by standard library)</td>
</tr>
<tr>
<td>uint8_t</td>
<td>8-bit integer, unsigned (Defined by standard library)</td>
</tr>
<tr>
<td>uint16_t</td>
<td>16-bit integer, unsigned (Defined by standard library)</td>
</tr>
<tr>
<td>uint32_t</td>
<td>32-bit integer, unsigned (Defined by standard library)</td>
</tr>
<tr>
<td>uint64_t</td>
<td>64-bit integer, unsigned (Defined by standard library)</td>
</tr>
<tr>
<td>float32_t</td>
<td>32-bit floating point (Defined by standard library when specifying &quot;<strong>ARM_NEON</strong>&quot;)</td>
</tr>
<tr>
<td>float64_t</td>
<td>64-bit floating point (Defined by standard library) (Defined by standard library when specifying &quot;<strong>ARM_NEON</strong>&quot;)</td>
</tr>
<tr>
<td>float128_t</td>
<td>128-bit floating point</td>
</tr>
</tbody>
</table>
### 6.5 Constants

Table 6.6 lists the Constants Used in Sample Code.

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAC_SAMPLE_SINGLE</td>
<td>(0)</td>
<td>Do not use continuous transfer function.</td>
</tr>
<tr>
<td>DMAC_SAMPLE_CONTINUATION</td>
<td>(1)</td>
<td>Use continuous transfer function.</td>
</tr>
<tr>
<td>DMAC_MODE_REGISTER</td>
<td>(0)</td>
<td>Use register mode.</td>
</tr>
<tr>
<td>DMAC_REQ_MODE_SOFT</td>
<td>(2)</td>
<td>Auto-request transfer request</td>
</tr>
<tr>
<td>DMAC_TRANS_SIZE_8</td>
<td>(0)</td>
<td>Transfer size: 8 bits</td>
</tr>
<tr>
<td>DMAC_TRANS_SIZE_16</td>
<td>(1)</td>
<td>Transfer size: 16 bits</td>
</tr>
<tr>
<td>DMAC_TRANS_SIZE_32</td>
<td>(2)</td>
<td>Transfer size: 32 bits</td>
</tr>
<tr>
<td>DMAC_TRANS_SIZE_64</td>
<td>(3)</td>
<td>Transfer size: 64 bits</td>
</tr>
<tr>
<td>DMAC_TRANS_SIZE_128</td>
<td>(4)</td>
<td>Transfer size: 128 bits</td>
</tr>
<tr>
<td>DMAC_TRANS_SIZE_256</td>
<td>(5)</td>
<td>Transfer size: 256 bits</td>
</tr>
<tr>
<td>DMAC_TRANS_SIZE_512</td>
<td>(6)</td>
<td>Transfer size: 512 bits</td>
</tr>
<tr>
<td>DMAC_TRANS_SIZE_1024</td>
<td>(7)</td>
<td>Transfer size: 1,024 bits</td>
</tr>
<tr>
<td>DMAC_TRANS_ADR_INC</td>
<td>(0)</td>
<td>Increment transfer source or transfer destination address.</td>
</tr>
<tr>
<td>DMAC_CH_TOTAL</td>
<td>(16)</td>
<td>Number of DMAC channels</td>
</tr>
<tr>
<td>DMAC_FLG_ON</td>
<td>(1)</td>
<td>Generate DMA transfer-end interrupt</td>
</tr>
<tr>
<td>DMAC_FLG_OFF</td>
<td>(0)</td>
<td>Do not generate DMA transfer-end interrupt</td>
</tr>
<tr>
<td>DMAC_BUFF_SIZE_4096</td>
<td>(4096)</td>
<td>Size of the first DMA transfer: 4096 bytes</td>
</tr>
<tr>
<td>DMAC_BUFF_SIZE_1024</td>
<td>(1024)</td>
<td>Size of the second DMA transfer: 1024 bytes</td>
</tr>
<tr>
<td>DMAC_BUFF_SIZE_256</td>
<td>(256)</td>
<td>Size of the third DMA transfer: 256 bytes</td>
</tr>
<tr>
<td>DMAC_PROT_SECURE_NORMAL</td>
<td>(0)</td>
<td>Definition of protection unit support*: Secure access, data access, normal access</td>
</tr>
<tr>
<td>DMAC_PROT_NONSECURE_NORMAL</td>
<td>(2)</td>
<td>Definition of protection unit support*: Non-secure access, data access, normal access</td>
</tr>
<tr>
<td>DMAC_CACHE_ATTRIB_L2_CACHEABLE</td>
<td>(0xF)</td>
<td>Definition of cache support*: L2 cacheable external memory space access</td>
</tr>
<tr>
<td>DMAC_CACHE_ATTRIB_L2_NON_CACHEABLE</td>
<td>(0x3)</td>
<td>Definition of cache support*: L2 non-cacheable external memory space access</td>
</tr>
<tr>
<td>DMAC_CACHE_ATTRIB_STRONGLY</td>
<td>(0x0)</td>
<td>Definition of cache support*: External memory space access of strongly ordered memory</td>
</tr>
<tr>
<td>DMAC_CACHE_ATTRIB_DEVICE</td>
<td>(0x1)</td>
<td>Definition of cache support*: External memory space access of device memory</td>
</tr>
<tr>
<td>DMAC_CACHE_ATTRIB_INTERNAL_AREA</td>
<td>(0x0)</td>
<td>Definition of cache support*: Internal space access</td>
</tr>
<tr>
<td>DMAC_CACHE_ATTRIB_L2C_DISABLE</td>
<td>(0x0)</td>
<td>Definition of cache support*: L2 cache-disabled space access</td>
</tr>
</tbody>
</table>

Notes:
1. The definition of the protection unit support defines the values to be set in the SPR bits or the DPR bits of the CHEXT_n register.
2. The definition of the cache support defines the values to be set in the SCA bits or the DCA bits of the CHEXT_n register.
### 6.6 Structure/Union List

Figure 6.4 shows the Structure/Union Used in Sample Code.

```c
typedef struct dmac_transinfo
{
    uint32_t src_addr;  /* Transfer source address */
    uint32_t dst_addr;  /* Transfer destination address */
    uint32_t count;   /* Transfer byte count */
    uint32_t src_size;  /* Transfer source data size */
    uint32_t dst_size;  /* Transfer destination data size */
    uint32_t saddr_dir;  /* Transfer source address direction */
    uint32_t daddr_dir;  /* Transfer destination address direction */
} dmac_transinfo_t;

uint32_t buff;  /* DMAC transfer information */

dmac_transinfo_t buff;

dmac_transinfo_t buff;

typedef struct dmac_cache_prot_info
{
    uint32_t src_prot;  /* Source protection */
    uint32_t dst_prot;  /* Destination protection */
    uint32_t src_attrib;  /* Source cache attribute */
    uint32_t dst_attrib;  /* Destination cache attribute */
} dmac_cache_prot_info_t;
```

Figure 6.4 Structure/Union Used in Sample Code
### 6.7 Variables

Table 6.7 lists the static Variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>static volatile</td>
<td>dmac_autoreq_trans_flg[]</td>
<td>DMA transfer-end interrupt notification flag</td>
<td>Sample_DMAC_AutoReq</td>
</tr>
<tr>
<td>uint8_t</td>
<td></td>
<td></td>
<td>Sample_DMAC_AutoReqSingle</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sample_DMAC_AutoReq</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>static uint8_t</td>
<td>dmac_src_data_internalram_1[DMAC_BUFF_SIZE_4096]</td>
<td>Transfer source data buffer of the first DMA transfer</td>
<td>Sample_DMAC_AutoReqSingle</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sample_DMAC_AutoReq</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>static uint8_t</td>
<td>dmac_src_data_internalram_2[DMAC_BUFF_SIZE_1024]</td>
<td>Transfer source data buffer of the second DMA transfer</td>
<td>Sample_DMAC_AutoReqSingle</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sample_DMAC_AutoReq</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>static uint8_t</td>
<td>dmac_src_data_internalram_3[DMAC_BUFF_SIZE_256]</td>
<td>Transfer source data buffer of the third DMA transfer</td>
<td>Sample_DMAC_AutoReqSingle</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sample_DMAC_AutoReq</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>static uint8_t</td>
<td>dmac_dst_data_sdram_1[DMAC_BUFF_SIZE_4096]</td>
<td>Transfer destination data buffer of the first DMA transfer</td>
<td>Sample_DMAC_AutoReqSingle</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sample_DMAC_AutoReq</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>static uint8_t</td>
<td>dmac_dst_data_sdram_2[DMAC_BUFF_SIZE_1024]</td>
<td>Transfer destination data buffer of the second DMA transfer</td>
<td>Sample_DMAC_AutoReqSingle</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sample_DMAC_AutoReq</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>static uint8_t</td>
<td>dmac_dst_data_sdram_3[DMAC_BUFF_SIZE_256]</td>
<td>Transfer destination data buffer of the third DMA transfer</td>
<td>Sample_DMAC_AutoReqSingle</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sample_DMAC_AutoReq</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: dmac_src_data_internalram_1, dmac_src_data_internalram_2, and dmac_src_data_internalram_3 are defined as the BSS_DMAC_SAMPLE_INTERNAL_RAM section and assigned to the large-capacity on-chip RAM space (H'2000 0000 to H'209F FFFF). dmac_dst_data_sdram_1, dmac_dst_data_sdram_2, and dmac_dst_data_sdram_3 are defined as the BSS_DMAC_SAMPLE_SDRAM section and assigned to the CS2 SDRAM space (H'0800 0000 to H'0BFF FFFF). By setting the MMU, the large-capacity on-chip RAM space enables the L1 cache, and the CS2 SDRAM space enables both the L1 and the L2 caches, respectively.
Figure 6.5 shows the Memory Allocation of Transfer Source Transfer Destination Buffers Used in DMA Transfer of the transfer source buffer and the transfer destination buffer used in the DAM transfer.
### 6.8 Functions

Table 6.8 lists the Functions.

#### Table 6.8 Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>Main processing</td>
</tr>
<tr>
<td>Sample_Main</td>
<td>Sample code main processing</td>
</tr>
<tr>
<td>Sample_DMAC_Main</td>
<td>DMAC sample code main processing</td>
</tr>
<tr>
<td>Sample_DMAC_AutoReqSingle</td>
<td>Sample processing which enables three individual DMA transfers when each of the transfer ends using auto-request as the DMA transfer request type</td>
</tr>
<tr>
<td>Sample_DMAC_AutoReq</td>
<td>Sample processing which performs three successive DMA transfers once DMA operation enabled by means of continuous transfer function using auto-request as the DMA transfer request type</td>
</tr>
<tr>
<td>DMAC_AutoReq_Init</td>
<td>DMAC initial settings when using auto-request as the DMA transfer request type</td>
</tr>
<tr>
<td>DMAC3_AutoReqInit</td>
<td>DMAC channel 3 initial settings when using auto-request as the DMA transfer request type</td>
</tr>
<tr>
<td>DMAC_Open</td>
<td>DMA transfer enable</td>
</tr>
<tr>
<td>DMAC3_Open</td>
<td>DMA channel 3 transfer enable</td>
</tr>
<tr>
<td>DMAC_Close</td>
<td>DMA transfer disable</td>
</tr>
<tr>
<td>DMAC3_Close</td>
<td>DMA channel 3 transfer disable</td>
</tr>
<tr>
<td>DMAC_Load_Set</td>
<td>DMA transfer information setting</td>
</tr>
<tr>
<td>DMAC3_Load_Set</td>
<td>DMA channel 3 transfer information setting</td>
</tr>
<tr>
<td>DMAC_Reload_Set</td>
<td>Reload setting for DMA transfer information</td>
</tr>
<tr>
<td>DMAC3_Reload_Set</td>
<td>Reload setting for DMAC channel 3 transfer information</td>
</tr>
<tr>
<td>Sample_DMA3Interrupt</td>
<td>DMAC channel 3 interrupt processing</td>
</tr>
<tr>
<td>DMAC_InitCacheProt</td>
<td>Initial settings for protection unit support and cache support (CHEXT_n register)</td>
</tr>
<tr>
<td>Sample_DMA3_GetMemoryType</td>
<td>Obtaining of the setting information to protection unit support and cache support for the specified address</td>
</tr>
</tbody>
</table>
## 6.9 Function Specifications

The following tables list the sample code function specifications.

### main

<table>
<thead>
<tr>
<th>Outline</th>
<th>Main processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>int_t main(void)</td>
</tr>
<tr>
<td>Description</td>
<td>Displays the sample code program information to the terminal running on the host PC which is connected to the GENMAI board via the serial interface.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>0</td>
</tr>
</tbody>
</table>

### Sample_Main

<table>
<thead>
<tr>
<th>Outline</th>
<th>Sample code main processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>void Sample_Main(void)</td>
</tr>
<tr>
<td>Description</td>
<td>Waits for the character input from the terminal running on the host PC which is connected to the GENMAI board via the serial interface. Activates the DMAC sample code by inputting &quot;DMAC&quot; + &quot;Enter&quot; key from the terminal.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

### Sample_DMAC_Main

<table>
<thead>
<tr>
<th>Outline</th>
<th>DMA sample code processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>int32_t Sample_DMAC_Main(int32_t argc, char_t **argv)</td>
</tr>
<tr>
<td>Description</td>
<td>Waits for the character input from the terminal running on the host PC which is connected to the GENMAI board via the serial interface. When the following commands are input, each DMAC sample code is executed. &quot;1&quot; + &quot;Enter&quot; key: Sample code 1 &quot;2&quot; + &quot;Enter&quot; key: Sample code 2</td>
</tr>
<tr>
<td>Arguments</td>
<td>int32_t argc : Number of command arguments input from the terminal char_t **argv : Pointer to the command input from the terminal</td>
</tr>
<tr>
<td>Return Value</td>
<td>COMMAND_EXIT : Termination of the DMA sample code</td>
</tr>
</tbody>
</table>
Sample_DMAC_AutoReqSingle

Outline
Sample processing which enables three individual DMA transfers when each of the transfers ends using auto-request type

Declaration
int32_t Sample_DMAC_AutoReqSingle(int32_t argc, char_t **argv)

Description
Transfers the large-capacity on-chip RAM data to the SDRAM allocated to the CS2 space in register mode for DMA mode and using auto-request as transfer request type. Sets the next DMA transfer information and enables the DMA transfer when a DMA transfer ends. Activates DMA for each transfer-end so that a total of three DMA transfers are performed to send 4096 bytes, 1024 bytes, and 256 bytes in order.

Arguments
int32_t argc : Number of command arguments input from the terminal
               (Not used in this function)
char_t **argv : Pointer to the command input from the terminal
               (Not used in this function)

Return Value
COMMAND_SUCCESS : Success of DMA sample code processing
COMMAND_ERROR : Failure of DMA sample code processing

Sample_DMAC_AutoReq

Outline
Sample processing which performs three successive DMA transfers using auto-request

Declaration
int32_t Sample_DMAC_AutoReq(int32_t argc, char_t **argv)

Description
Transfers the data in the large-capacity on-chip RAM to the SDRAM allocated to the CS2 space in register mode for DMA mode and using auto-request as transfer request type. Sets the next DMA transfer information before the DMA transfer ends so that successive DMA transfer can be executed after a transfer ends by enabling DMA transfer once. Total of three transfers are performed. Refer to the flowcharts shown in Figure 6.11 and Figure 6.12 for details about the timing to set the DMA transfer information.

Arguments
int32_t argc : Number of command arguments input from the terminal
               (Not used in this function)
char_t **argv : Pointer to the command input from the terminal
               (Not used in this function)

Return Value
COMMAND_SUCCESS : Success of DMA sample code processing
COMMAND_ERROR : Failure of DMA sample code processing
### DMAC_AutoReq_Init

**Outline**
DMAC initial settings when using auto-request as the DMA transfer request type

**Declaration**
```c
int32_t DMAC_AutoReq_Init(uint32_t channel, const dmac_transinfo_t *trans_info,
                          uint32_t dmamode, uint32_t continuation)
```

**Description**
Executes the DMAC initial settings using auto-request as the DMA transfer request type. In the sample code, initial setting for the DMAC channel 3 is executed.

**Arguments**
- `uint32_t channel`: DMA channel
- `const dmac_transinfo_t * trans_info`: Setting information to the DMAC registers
- `uint32_t dmamode`: DMA mode (only DMAC_MODE_REGISTER supported)
- `uint32_t continuation`: Enable continuous transfer after DMA transfer-end

<table>
<thead>
<tr>
<th>DMAC_SAMPLE_CONTINUATION</th>
<th>Execute continuous transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAC_SAMPLE_SINGLE</td>
<td>Do not execute continuous transfer</td>
</tr>
</tbody>
</table>

**Return Value**
- 0: Success of initial setting processing
- -1: Argument error

### DMAC3_AutoReqInit

**Outline**
DMAC channel 3 initial settings when using auto-request as the DMA transfer request type

**Declaration**
```c
void DMAC3_AutoReqInit(const dmac_transinfo_t *trans_info,
                        uint32_t dmamode, uint32_t continuation)
```

**Description**
Sets the DMAC channel 3 to the register mode for DMA mode and the auto-request as transfer request type. Executes the DMAC initial settings using the DMA information specified by the argument *trans_info and the enabled/disabled information for the continuous transfer specified by the argument continuation. Registers the DMAC channel 3 interrupt handler function and sets the interrupt priority level to enable the transfer-end interrupt.

**Arguments**
- `const dmac_transinfo_t * trans_info`: Setting information to the DMAC registers
- `uint32_t dmamode`: DMA mode (only DMAC_MODE_REGISTER supported)
- `uint32_t continuation`: Enable continuous transfer after DMA transfer-end

<table>
<thead>
<tr>
<th>DMAC_SAMPLE_CONTINUATION</th>
<th>Execute continuous transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAC_SAMPLE_SINGLE</td>
<td>Do not execute continuous transfer</td>
</tr>
</tbody>
</table>

**Return Value**
None
### DMAC_Open

**Outline**
DMA transfer enable

**Declaration**
```c
int32_t DMAC_Open(uint32_t channel, uint32_t req)
```

**Description**
Enables DMA transfer specified by the argument channel.

In the sample code, the DMAC channel 3 is enabled for transfer in auto-request mode by specifying `DMAC_REQ_MODE_SOFT` for the argument `req`.

**Arguments**
- `uint32_t channel`: DMAC channel
- `uint32_t req`: DMAC request mode

**Return Value**
- 0 : Success in enabling DMA transfer
- -1 : Argument error or failure in enabling DMA transfer

### DMAC3_Open

**Outline**
DMAC channel 3 transfer enable

**Declaration**
```c
int32_t DMAC3_Open(uint32_t req)
```

**Description**
Enables the DMAC channel 3 transfer. In the sample code, `DMAC_REQ_MODE_SOFT` is specified for the argument `req`, and the transfer is enabled in auto-request mode.

**Arguments**
- `uint32_t req`: DMAC request mode

**Return Value**
- 0 : Success in enabling DMA transfer
- -1 : Failure in enabling DMA transfer (Because DMA in operation)

### DMAC_Close

**Outline**
DMA transfer abort

**Declaration**
```c
int32_t DMAC_Close(uint32_t channel, uint32_t *remain)
```

**Description**
Aborts the DMA transfer specified by the argument channel. In the sample code, the DMAC channel 3 transfer is suspended. Returns the remaining transfer byte count at the time of DMA transfer abort to the argument `*remain`.

**Arguments**
- `uint32_t channel`: DMAC channel
- `uint32_t *remain`: Remaining transfer byte count when the DMA transfer is suspended.

**Return Value**
- 0 : Success of DMA transfer abort
- -1 : Argument error or failure of DMA transfer abort

### DMAC3_Close

**Outline**
DMAC channel 3 transfer abort

**Declaration**
```c
int32_t DMAC3_Close(uint32_t *remain)
```

**Description**
Aborts the DMA transfer specified by the argument channel. Returns the remaining transfer byte count at the time of DMA transfer abort to the argument `*remain`.

**Arguments**
- `uint32_t *remain`: Remaining transfer byte count when the DMA transfer is suspended.

**Return Value**
- 0 : Success of DMA transfer abort
- -1 : Argument error or failure of DMA transfer abort
DMAC_Load_Set

Outline         DMA transfer information setting
Declaration     int32_t DMAC_Load_Set(uint32_t channel, uint32_t src_addr, uint32_t dst_addr,
                                uint32_t count)
Description     Sets the transfer source address, transfer destination address, and total transfer byte count specified by the arguments src_addr, dst_addr, and count respectively to the DMAC channel specified by the argument channel. This function should be called when the DMA transfer on the specified channel is suspended. The DMAC channel 3 is specified in the sample code.
Arguments       unit32_t channel : DMAC channel
                unit32_t src_addr  : Transfer source address
                unit32_t dst_addr  : Transfer destination address
                unit32_t count     : Total transfer byte count
Return Value    0       : Success of DMA setting
                -1       : Argument error or DMA specified by channel is in operation
Note            This function is used to make transfer information settings when the DMA mode is register mode. It does not support making settings when the DMA mode is link mode.

DMAC_Reload_Set

Outline         Reload setting for DMA transfer information
Declaration     int32_t DMAC_Reload_Set(uint32_t channel, uint32_t src_addr, uint32_t dst_addr,
                                uint32_t count)
Description     Sets the transfer source address, transfer destination address, and total transfer byte count specified by the arguments src_addr, dst_addr, and count respectively to the DMAC channel specified by the argument channel as the DMA transfer information for the next continuous transfer. This function should be called while DMA of the specified channel is in operation and when the REN bit is "0" during the continuous transfer operation. The DMAC channel 3 is specified in the sample code.
Arguments       unit32_t channel : DMAC channel
                unit32_t src_addr  : Transfer source address to reload
                unit32_t dst_addr  : Transfer destination address to reload
                unit32_t count     : Total transfer byte count for reload
Return Value    0       : Succeed of DMA reload setting
                -1       : Argument error or DMA specified by channel ends before the reload setting
Note            This function is used to reload settings when the DMA mode is register mode. It does not support reloading settings when the DMA mode is link mode.
DMAC3_Load_Set

Outline  
DMAC channel 3 transfer information setting

Declaration  
int32_t DMAC3_Load_Set(uint32_t src_addr, uint32_t dst_addr, uint32_t count)

Description  
Sets the transfer source address, transfer destination address, and total transfer byte count specified by the arguments src_addr, dst_addr, and count respectively to DMAC channel 3 as the DMA transfer information.
Sets the register set selected by the RSEL bit in the CHCFG_3 register either from Next0 or Next1 register set. This function should be called when the DMA transfer of DMAC channel 3 is aborted.

Arguments  
  unit32_t src_addr : Transfer source address
  unit32_t dst_addr : Transfer destination address
  unit32_t count : Total transfer byte count

Return Value  
  0 : Success in DMA setting
  -1 : DMA of DMAC channel 3 is in operation

Note  
This function is used to reload settings when the DMA mode is register mode. It does not support reloading settings when the DMA mode is link mode. Also, when not using the continuous transfer function, call this function when the value of the RSW bit is "0".

DMAC3_Reload_Set

Outline  
Reload setting for DMAC channel 3 transfer information

Declaration  
int32_t DMAC3_Reload_Set(uint32_t src_addr, uint32_t dst_addr, uint32_t count)

Description  
Sets the transfer source address, transfer destination address, and total transfer byte count specified by the arguments src_addr, dst_addr, and count respectively to the DMAC channel 3 as the DMA transfer information for the next continuous transfer.
Sets the REN bit in the CHCFG_n register to "1" to enable the continuous transfer function.
When the RSW bit of the CHCFG_3 register is set to "1", executes reload setting for the register set which is not selected by the RSEL bit either from Next0 or Next1 register set.
When the RSE bit is set to "0", executes reload setting for the register set selected by the RSEL bit.
This function should be called while DMA of the DMAC channel 3 is in operation and when the REM bit is "0" at the time of continuous transfer operation.

Arguments  
  unit32_t src_addr : Transfer source address to reload
  unit32_t dst_addr : Transfer destination address to reload
  unit32_t count : Total transfer byte count to reload

Return Value  
  0 : Success in DMA reload setting
  -1 : DMA transfer on DMAC channel 3 ends before reload setting

Note  
This function is used to reload settings when the DMA mode is register mode. It does not support reloading settings when the DMA mode is link mode.
Sample_DMAC3_Interrupt

Outline: DMAC channel 3 interrupt processing

Declaration: void Sample_DMAC3_Interrupt(uint32_t int_sense)

Description: This function is an interrupt handler which is executed when the DMAC channel 3 transfer-end interrupt has been accepted.

Notifies that the DMAC channel 3 transfer-end interrupt has been generated.

Arguments:

- uint32_t int_sense: Interrupt detection method (Not used)
  - INTC_LEVEL_SENSITIVE: Level sense
  - INTC_EDGE_TRIGGER: Edge trigger

Return Value: None

DMAC_InitCacheProt

Outline: Initial settings for protection unit support and cache support (CHEXT_n register)

Declaration: int32_t DMAC_InitCacheProt(uint32_t channel, dmac_cache_prot_info_t * info)

Description: Executes the initial setting of the CHEXT_n register of the DMAC channel specified by the argument channel. Sets the SCA bits, DCA bits, and SPR bits according to the contents specified by the argument info respectively. The information of the argument info is obtained by calling the Sample_DMAC_GetMemoryType function before this function is called.

It is necessary to execute the initial setting of the CHEXT_n register before a DMA transfer starts.

Arguments:

- uint32_t channel: DMAC channel
- dmac_cache_prot_info_t * info: Setting information to protection unit support and cache support for the CHEXT_n register

Return Value:

- 0: Success of the CHEXT_n register initial setting processing
- -1: Argument error

Sample_DMAC_GetMemoryType

Outline: Obtaining of the setting information to protection unit support and cache support for the specified address

Declaration: void Sample_DMAC_GetMemoryType(void * addr, uint32_t * protection, uint32_t * cache_attrib)

Description: Obtains the information to be set to the protection unit support and the cache support in the area specified by the argument addr. The obtained contents are set to the arguments *protection and *cache_attrib. In the sample code, the contents of the protection unit support and the cache support, shown in Table 6.9, for the area including the address specified by the argument addr, are obtained.

Arguments:

- void * addr: Address specified as transfer source or transfer destination of DMAC
- uint32_t * protection: Protection unit information
- uint32_t * cache_attrib: Memory area attribute

Return Value: None

Note: In this function, the setting information to the protection unit support and the cache support, for the area including the specified address, is obtained according to the information of the default MMU translation table used in the sample code. When the MMU translation table (set by the file ttb_init.s) is customized, it is necessary to modify this function according to the changed contents of the table.
Table 6.9  Setting Information to Protection Unit Support and Cache Support, Used in Sample Code

<table>
<thead>
<tr>
<th>Address</th>
<th>Values of SPR and DPR bits</th>
<th>Values of SCA and DCA bits</th>
<th>Memory Area Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>H'0000 0000 to H'07FF FFFF (CS0 and CS1 spaces)</td>
<td>B'010</td>
<td>B'1111</td>
<td>• Non-secure / data / normal access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• External space access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2 cache-enabled normal memory</td>
</tr>
<tr>
<td>H'0800 0000 to H'0FFF FFFF (CS2 and CS3 spaces)</td>
<td>B'010</td>
<td>B'1111</td>
<td>• Non-secure / data / normal access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• External space access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2 cache-enabled normal memory</td>
</tr>
<tr>
<td>H'1000 0000 to H'17FF FFFF (CS4 and CS5 spaces)</td>
<td>B'010</td>
<td>B'0000</td>
<td>• Non-secure / data / normal access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• External space access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Strongly ordered memory</td>
</tr>
<tr>
<td>H'1800 0000 to H'1FFF FFFF (SPI multi I/O bus space 1 and 2)</td>
<td>B'010</td>
<td>B'1111</td>
<td>• Non-secure / data / normal access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• External space access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2 cache-enabled normal memory</td>
</tr>
<tr>
<td>H'2000 0000 to H'209F FFFF (Large-capacity on-chip RAM space)</td>
<td>B'010</td>
<td>B'0000</td>
<td>• Non-secure / data / normal access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• External space access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Internal space access</td>
</tr>
<tr>
<td>H'20A0 0000 to H'3FFF FFF (On-chip peripheral module and reserved area)</td>
<td>B'000</td>
<td>B'0000</td>
<td>• Secure / data / normal access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Internal space access</td>
</tr>
<tr>
<td>H'4000 0000 to H'47FF FFFF (CS0 and CS1 mirror spaces)</td>
<td>B'010</td>
<td>B'0011</td>
<td>• Non-secure / data / normal access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• External space access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2 cache-disabled normal memory</td>
</tr>
<tr>
<td>H'4800 0000 to H'4FFF FFFF (CS2 and CS3 mirror spaces)</td>
<td>B'010</td>
<td>B'0011</td>
<td>• Non-secure / data / normal access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• External space access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2 cache-disabled normal memory</td>
</tr>
<tr>
<td>H'5000 0000 to H'57FF FFFF (CS4 and CS5 mirror spaces)</td>
<td>B'010</td>
<td>B'0000</td>
<td>• Non-secure / data / normal access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• External space access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Strongly ordered memory</td>
</tr>
<tr>
<td>H'5800 0000 to H'5FFF FFFF (SPI multi I/O bus mirror space 1 and 2)</td>
<td>B'010</td>
<td>B'0011</td>
<td>• Non-secure / data / normal access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• External space access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2 cache-disabled normal memory</td>
</tr>
<tr>
<td>H'6000 0000 to H'609F FFFF (Large-capacity on-chip RAM mirror space)</td>
<td>B'010</td>
<td>B'0000</td>
<td>• Non-secure / data / normal access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Internal space access</td>
</tr>
<tr>
<td>H'60A0 0000 to H'FFFF FFFF (On-chip peripheral module and reserved area)</td>
<td>B'000</td>
<td>B'0000</td>
<td>• Secure / data / normal access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Internal space access</td>
</tr>
</tbody>
</table>

Notes
1. In the case that the L2 cache is disabled even in the external space access, the value B'0000 is obtained to the argument *cache.attrib when the Sample_DMAC_GetMemoryType function is called. Then the value B'0000 is set in the SCA bits or the DCA bits of the CHEXT_n register when the DMAC_InitCacheProt function is executed.
2. The values set in the SPR[2:0], DPR[2:0], SCA[3:0] and DCA[3:0] bits of the CHEXT_n register are to be output to the ARPROT[2:0], AWPROT[2:0], ARCACHE[3:0] and AWCACHE[3:0] bits, respectively, supported by the AMBA AXI protocol. See AMBA® AXI Protocol Specification from ARM Limited for details.
6.10 Flowcharts

6.10.1 Main Processing
Figure 6.6 shows the flowchart of Main Processing.

Figure 6.6 Main Processing

```
main
  Output to terminal
    printf()
  OSTM0-related settings
  Peripheral function sample
    code startup function
      Sample_Main()
  return (0)
```

- Outputs the sample code version information to the terminal program running on the host PC connected via the serial interface.
- Blinks the LED at 500ms intervals using the OSTM channel 0 interrupt.
  Refer to the RZ/A1H group Example of Initialization application note for details.
- Branches to processing to wait for a command reception from the terminal.
  The sample code for the appropriate peripheral function is executed according to the command received.
6.10.2 Sample Code Main Processing

Figure 6.7 shows the flowchart of Sample Code Main Processing. This function waits for the character input from the terminal software running on the host PC.

The DMAC sample code is executed when "DMAC" + "Enter" key is input.

```
Sample_Main

Output to terminal
printf()

Acquisition of main processing command list
Sample_GetCmdList()

Registration of main processing command list
CommandSetCmdList()

Wait for command input
gets()

Analyze and execute command
CommandExe()

"EXIT" input?

"EXIT" was input

Reacquisition of main processing command list
Sample_GetCmdList()

"EXIT" was not input

Reregistration of main processing command list
CommandSetCmdList()
```

Figure 6.7 Sample Code Main Processing
6.10.3 DMAC Sample Main Function

Figure 6.8 shows the flowchart of DMAC Sample Main Function. This function waits for the character input from the terminal running on the host PC and branches to the DMAC sample code according to the input command.

When "1" + "Enter" key is input, the sample code 1 (which performs three DMA transactions) is executed.

When "2" + "Enter" key is input, the sample code 2 (which performs three successive DMA transactions using the continuous transfer function) is executed.

![Flowchart of DMAC Sample Main Function]

<table>
<thead>
<tr>
<th>Sample_DMCA_Main</th>
<th>Output to terminal printf()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Outputs the DMAC sample code version information to the terminal running on the host PC.</td>
</tr>
<tr>
<td>Acquisition of DMAC sample processing command list Sample_DMCA_GetCmdList()</td>
<td></td>
</tr>
<tr>
<td>Registration of main processing command list CommandSetCmdList()</td>
<td>Registers the DMAC sample processing command list. The menu list for launching the DAMC sample code is registered in the sample code.</td>
</tr>
<tr>
<td>Wait for command input gets()</td>
<td>Waits for command input from the terminal and stores it in the command buffer.</td>
</tr>
<tr>
<td>Analyze and execute command CommandExe()</td>
<td>Analyzes and executes the contents of the command buffer. In the sample code, this function branches to one of the following DMAC sample code processing routines, according to the input command: &quot;1&quot;: Branches to the Sample_DMCA_AutoReqSingle function. &quot;2&quot;: Branches to the Sample_DMCA_AutoReq function. &quot;HELP&quot;: Displays a list of the available commands.</td>
</tr>
</tbody>
</table>

"EXIT" was not input

"EXIT" was input

"EXIT" input?

return (COMMAND_EXIT)
6.10.4 Sample Function with Auto-Request as DMA Transfer Request Type

Figure 6.9 and Figure 6.10 show the flowcharts of Sample Function with Auto-Request as DMA Transfer Request Type.

---

**Flowchart:** Sample Function with Auto-Request as DMA Transfer Request Type

1. **Sample_DMAC_AutoReqSingle**
   - Outputs DMAC setting information to the terminal running on the host PC.
2. **Clear DMA transfer-end interrupt notification flag**
   - Clears the interrupt notification flag set by the interrupt processing when a DMA transfer-end interrupt is generated.
3. **Initialize DMA transfer information variables and structures**
   - The DMA transfer information variables and structures are initialized with the following settings:
     - Continuous execution: Disabled at DMA transfer-end
     - DMA mode: Register mode
     - DMA channel: Channel 3
     - 1st transfer source address: Large-capacity on-chip RAM
     - 1st transfer destination address: SDRAM
     - 1st transfer total byte count: 4096 bytes
     - Transfer source transfer size: 8 bits
     - Transfer destination transfer size: 8 bits
     - Transfer source address count direction: Increment
     - Transfer destination address count direction: Increment
   - Makes initial settings with auto-request as the DMAC transfer request type. In the sample code, initial settings for the DMAC channel 3 are made in register mode for DMA mode, and initial interrupt settings are made.
4. **Enable 1st DMA transfer**
   - Enables DMA transfers.
   - In the sample code, the DMA transfer operation on channel 3 starts in auto-request as the transfer request type.
5. **Wait for DMA transfer-end**
   - Waits for the interrupt notification flag to be set by the DMAC channel 3 transfer-end interrupt processing.
6. **Clear DMA transfer-end interrupt notification flag**
   - Clears the interrupt notification flag set by the interrupt processing.

---

**Figure 6.9 Sample Function with Auto-Request as DMA Transfer Request Type (1/2)**
A

Initialize DMA transfer information variables

Set DMA transfer information for 2nd transfer
DMAC_Load_Set()

Enable 2nd DMA transfer
DMAC_Open()

Wait for DMA transfer-end

Clear DMA transfer-end interrupt notification flag

Initialize DMA transfer information variables

Set DMA transfer information for 3rd transfer
DMAC_Load_Set()

Enable 3rd DMA transfer
DMAC_Open()

Wait for DMA transfer-end

Clear DMA transfer-end interrupt notification flag

Flush all cache lines in L1 data caches
L1_D_CacheFlushAll()

Contents of DMA Transfer Destination Buffer Correct?

Transferred abnormally

return (COMMAND_ERROR)

Transferred normally

return (COMMAND_SUCCESS)

RZ/A1H Group Direct Memory Access Controller Setting Example

return (COMMAND_SUCCESS)

Initialize the DMA transfer information for the second DMA transfer.
2nd transfer source address : Large-capacity on-chip RAM
2nd transfer destination address : SDRAM
2nd transfer total byte count : 1024 bytes

Makes DMA transfer information settings. Specifies the transfer source address, transfer destination address, and total transfer byte count for the DMA transfer.

Enables DMA transfers.

Waits for the interrupt notification flag to be set by the DMAC channel 3 transfer-end interrupt processing.

Clears the interrupt notification flag set by the interrupt processing.

Initialize the DMA transfer information for the third DMA transfer.
3rd transfer source address : Large-capacity on-chip RAM
3rd transfer destination address : SDRAM
3rd transfer total byte count : 256 bytes

Makes DMA transfer information settings. Specifies the transfer source address, transfer destination address, and total transfer byte count for the DMA transfer.

Enables DMA transfers.

Waits for the interrupt notification flag to be set by the DMAC channel 3 transfer-end interrupt processing.

Clears the interrupt notification flag set by the interrupt processing.

All cache lines in the L1 data caches are flushed. In the transfer destination address area of the DMA transfer, the data may have been cached without waiting for the completion of the transfer because of a speculative cache line fill. By flushing the caches again, the CPU can read the data after the completion of the DMA transfer.

Figure 6.10 Sample Function with Auto-Request as DMA Transfer Request Type (2/2)
6.10.5 Sample Function with Auto-Request as DMA Transfer Request Type (Continuous Transfer)

Figure 6.11 and Figure 6.12 show the flowcharts of Sample Function with Auto-Request as DMA Transfer Request Type (Continuous Transfer).

- Outputs DMAC setting information to the terminal running on the host PC.
- Clears the interrupt notification flag set by the interrupt processing when a DMA transfer-end interrupt is generated.
- The DMA transfer information variables and structures are initialized with the following settings:
  - Continuous execution: Disabled at DMA transfer-end
  - DMA mode: Register mode
  - DMA channel: Channel 3
  - 1st transfer source address: Large-capacity on-chip RAM
  - 1st transfer destination address: SDRAM
  - 1st transfer total byte count: 4096 bytes
  - Transfer source transfer size: 8 bits
  - Transfer destination transfer size: 8 bits
  - Transfer source address count direction: Increment
  - Transfer destination address count direction: Increment

- To ensure the coherency between the memory areas, set as the transfer source and the transfer destination of the DMA transfers, and the caches, all cache lines in the L1 data caches are written back and flushed.
- The setting information to the protection unit support and the cache support, for the transfer source and the transfer destination, are obtained.
  - Information of the transfer source (large-capacity on-chip RAM)
    - Information to the protection unit support: Non-secure / data / normal access
    - Information to the cache support: Internal space access
  - Information of the transfer destination (SDRAM)
    - Information to the protection unit support: Non-secure / data / normal access
    - Information to the cache support: External space access
  - L2 cache-enabled normal memory

- Makes initial settings for the SPR[2:0], DPR[2:0], SCA[3:0] and DCA[3:0] bits of the CHEXT_n register using the information obtained as described above.
- Makes initial settings with auto-request as the DMAC transfer request type.
  - In the sample code, initial settings for the DMAC channel 3 are made in register mode for DMA mode, and initial interrupt settings are made.
Initialize DMA transfer information variables

Set DMA transfer information for 2nd transfer DMAC_Load_Set()

Enable DMA transfers DMAC_Open()

Wait for 1st DMA transfer to finish

Clear DMA transfer-end interrupt notification flag

Initialize DMA transfer information variables

Reload settings for 3rd DMA transfer information DMAC_Reload_Set()

Wait for 2nd DMA transfer to finish

Clear DMA transfer-end interrupt notification flag

Wait for 3rd DMA transfer to finish

Clear DMA transfer-end interrupt notification flag

Flush all cache lines in L1 data caches L1_D_CacheFlushAll()

Contents of DMA Transfer Destination Buffer Correct?

Transferred abnormally

Transferred normally

return (COMMAND_SUCCESS)

return (COMMAND_ERROR)

Figure 6.12 Sample Function with Auto-Request as DMA Transfer Request Type (Continuous Transfer) (2/2)
6.10.6 DMAC Initial Setting Function when Using Auto-Request as DMA Transfer Request Type

Figure 6.13 shows the flowchart of DMAC Initial Setting Function when Using Auto-Request as DMA Transfer Request Type.

In the sample code, the DMA transfer-end interrupt handler function is registered, and the interrupt initial settings are executed using the DMAC channel 3.

```
DMAC_AutoReq_Init
argument error in function?

Channel 3?
channel is other than 3?

initial settings for auto-request on DMAC channel 3
DMAC3_AutoReqInit()

return (0)
```

Figure 6.13   DMAC Initial Setting Function when Using Auto-Request as DMA Transfer Request Type
6.10.7 DMAC Channel 3 Initial Setting Function when Using Auto-Request as DMA Transfer Request Type

Figure 6.14 and Figure 6.15 show the flowcharts of DMAC Channel 3 Initial Setting Function when Using Auto-Request as DMA Transfer Request Type.

The API functions are used to register the DMAC channel 3 interrupt function as the INTC handler, to set the interrupt priority level, and to enable the interrupts. Refer to the RZ/A1H group Application Note "Example of Initialization" for more details about the INTC interrupt API functions R_INTC_RegistIntFunc, R_INTC_SetPriority, and R_INTC_Enablet.

![Flowchart of DMAC Channel 3 Initial Setting Function when Using Auto-Request as DMA Transfer Request Type](image)

Figure 6.14 DMAC Channel 3 Initial Setting Function when Using Auto-Request as DMA Transfer Request Type (1/2)
Figure 6.15   DMAC Channel 3 Initial Setting Function when Using Auto-Request as DMA Transfer Request Type (2/2)

Set auto-request information

CHCFG_3 register
TM bit ← 1 : Block transfer mode
AM bit ← B'100 : DMA activation by the STG bit
SEL bit ← B'011 : Specifies channel 3.
LVL bit ← 0
HIEN bit ← 0
LOEN bit ← 0
REQD bit ← 0

Set transfer request source

DMARS1 register
CH3_MID bit ← 0 : Specifies auto-request mode.
CH3_RID bit ← 0

Setting for shared channel

DCTRL_0_7 register
PR bit ← 1 : Specifies round-robin priority control.

Register DMAC channel 3 interrupt handler
R_INTC_RegistIntFunc()

Sets the DMAC channel 3 interrupt priority level.
In the sample code, "1" is set as the interrupt priority level.

Enable DMAC channel 3 interrupt
R_INTC_Enable()

Enables the DMAC channel 3 interrupt.

return

Registers the DMAC channel 3 interrupt handler.
In the sample code, the handler function Sample_DMAC3_Interrupt is registered to notify the DMA transfer-end.

A
6.10.8 DMA Transfer Enable Function

Figure 6.16 shows the flowchart of DMA Transfer Enable Function. In the sample code, the DMA transfer is enabled in auto-request mode using the DMA channel 3.

```
DMAC_Open

Argument error in function?
   Error in specification for channel or error in DMA setting information
   return (-1)

Channel 3?
   Channel is other than 3
   Enable DMAC channel 3 transfer
   DMAC3_Open()

Enables transfers on DMAC channel 3 and starts transfer operation in auto-request mode.
```

Figure 6.16 DMA Transfer Enable Function
6.10.9 DMA Transfer Abort Function

Figure 6.17 shows the flowchart of DMA Transfer Abort Function.

![Flowchart of DMA Transfer Abort Function](image)

- **DMAC_Close**
- **Argument error in function?**
  - Error in specification for channel
    - return (-1)
- **Channel 3?**
  - Channel is other than 3?
    - Stop DMAC channel 3 transfer
      - DMAC3_Close()
        - Stops transfer operation on DMAC channel 3.
        - In the sample code, the remaining transfer byte count at the time of DMA transfer operation stop is returned to the argument *remain.*
    - return (0)

Figure 6.17 DMA Transfer Abort Function
6.10.10 DMA Transfer Information Setting Function

Figure 6.18 shows the flowchart of DMA Transfer Information Setting Function.

```
DMAC_Load_Set

Argument error in function?

Error in specification for channel

return (-1)

Channel 3?

Channel is other than 3

DMAC channel 3 settings
DMAC3_Load_Set()

Sets the transfer source address, transfer destination address, and total transfer byte count for DMAC channel 3.

return (ret)
```

Figure 6.18 DMA Transfer Information Setting Function
6.10.11 DMA Transfer Information Reload Setting Function

Figure 6.19 shows the flowchart of DMA Transfer Information Reload Setting Function.

```
DMAC_Reload_Set

Argument error in function?
    return (-1)

Channel 3?
    Channel is other than 3

Reload Settings for DMAC channel 3
DMAC3_Reload_Set()

Sets the transfer source address, transfer destination address, and total transfer byte count for the next transfer on DMAC channel 3.
```

Figure 6.19 DMA Transfer Information Reload Setting Function
### 6.10.12 DMAC Channel 3 Transfer Enable Function

Figure 6.20 shows the flowchart of DMAC Channel 3 Transfer Enable Function.

**DMAC Channel 3 Transfer Enable Function Flowchart**

- **DMAC3_Open**
  - **Channel 3 DMA stopped?**
    - **Clear DMAC channel 3 status**
      - **CHCTL_3 register**
        - **SWRST bit ← 1**
          : Resets the channel 3 status register.
    - **Enable DMA transfer operation on DMAC channel 3**
      - **CHCTL_3 register**
        - **SETEN bit ← 1**
          : Enables DMA transfer operation on channel 3.
    - **Auto-request?**
      - **Value of argument req is not DMAC_REQ_MODE_SOFT**
        - **Specify software transfer requests for DMAC channel 3**
          - **CHCTL_3 register**
            - **STG bit ← 1**
              : Specifies software transfer requests for channel 3.
  - **return (ret)**
6.10.13 DMAC Channel 3 Transfer Abort Function

Figure 6.21 shows the flowchart of DMAC Channel 3 Transfer Abort Function.

![Flowchart of DMAC Channel 3 Transfer Abort Function](image)

- **DMAC3_Close**
- **Suspend transfer operation on DMAC channel 3**
  - **CHCTL_3 register**
  - SETSUS bit ← 1 : DMA transfer operation on channel 3 is suspended.
- **DMA transfer is not suspended?**
  - **Operation enabled (EN bit = 1)**
  - **Suspended (SUS bit = 1)**
  - **Not suspended (SUS bit = 0)**
- **DMA operation stopped?**
  - **Operation stopped (EN bit = 0)**
  - **Operating (TACT bit = 1)**
  - **DMA transfer operation in stopped state?**
  - **Stopped (TACT bit = 0)**
  - **Operation enabled (EN bit = 1)**
  - **DMA operation stopped?**
- **Obtain remaining transfer byte count**
  - Argument *remain ← CRTB_3 register
  - Returns remaining transfer byte count at the time of DMA transfer operation stop.
- **Release transfer request source**
  - **DMARS1 register**
  - CH3_MID bit ← 0 : Clear transfer request source
  - CH3_RID bit ← 0
- **return**

**Figure 6.21 DMAC Channel 3 Transfer Abort Function**
6.10.14 DMAC Channel 3 Setting Function

Figure 6.22 shows the flowchart of DMAC Channel 3 Setting Function.

![Flowchart of DMAC Channel 3 Setting Function](image)

Figure 6.22 DMAC Channel 3 Setting Function
6.10.15 DMAC Channel 3 Reload Setting Function

Figure 6.23 shows the flowchart of DMAC Channel 3 Reload Setting Function.

![Flowchart of DMAC Channel 3 Reload Setting Function]

- **DMAC3_Reload_Set**
  - DMA operation on channel 3 and continuous transfer operation are both in progress?
    - DMA transfer in progress and one of two consecutive transfers finished (EN bit = 1 and REN bit = 0)
      - Register set 0 selected?
        - Register set 0 is selected (SR bit = 0)
          - Set register set information for selected DMA
          - Obtains the register set information specified by the SR bit in the CHSTAT_3 register.
          - N0SA_3 or N1SA_3 register ← src_addr
          - N0DA_3 or N1DA_3 register ← dst_addr
          - N0TB_3 or N1TB_3 register ← count
          - Sets the transfer source address, transfer destination address, and total transfer byte count for the next transfer on DMAC channel 3.
          - Set continuing execution bit for DMA continuous transfer
            - CHCFG_3 register
              - REN bit ← 1 : Continues execution after DMA transfer-end.
          - return (ret)
      - Register set 1 is selected (SR bit = 1)
        - Set reload information for next DMA transfer to register set 0
          - N0SA_3 or N1SA_3 register ← src_addr
          - N0DA_3 or N1DA_3 register ← dst_addr
          - N0TB_3 or N1TB_3 register ← count
          - return (ret)
    - Operation stopped (EN bit = 0) or continuous transfer not finished (REN bit = 0)

Figure 6.23   DMAC Channel 3 Reload Setting Function
6.10.16 DMAC Channel 3 Transfer interrupt Function

Figure 6.24 shows the flowchart of DMAC Channel 3 Transfer Interrupt Function.

```
Sample_DMAC3_Interrupt

Set DMA transfer-end interrupt notification flag

Sets the flag to notify that the DMA transfer-end interrupt has been generated.

return
```

Figure 6.24 DMAC Channel 3 Transfer Interrupt Function
6.10.17 Initial Setting Function for Protection Unit Support and Cache Support

Figure 6.25 to Figure 6.27 show the flowcharts of the Initial Setting Function for Protection Unit Support and Cache Support.

In the sample code, the setting information to the protection unit support and the cache support, for the area including the specified address, is obtained by calling the Sample_DMAC_GetMemoryType function, before this function is called.

```
DMAC_InitCacheProt

Error in specification for channel or error in CHEXT_n register setting information (info)

return (-1)
```

- **Channel?**
  - channel = 0
  - **Set protection unit support and cache support**
    - CEXT_0 register
      - SPR bits ← info->src_prot : Protection unit support for transfer source
      - SCA bits ← info->src_attrib : Cache support for transfer source
      - DPR bits ← info->dst_prot : Protection unit support for transfer destination
      - DCA bits ← info->dst_attrib : Cache support for transfer destination
  - channel = 1
    - **Set protection unit support and cache support**
      - CEXT_1 register
        - SPR bits ← info->src_prot : Protection unit support for transfer source
        - SCA bits ← info->src_attrib : Cache support for transfer source
        - DPR bits ← info->dst_prot : Protection unit support for transfer destination
        - DCA bits ← info->dst_attrib : Cache support for transfer destination
  - channel = 2
    - **Set protection unit support and cache support**
      - CEXT_2 register
        - SPR bits ← info->src_prot : Protection unit support for transfer source
        - SCA bits ← info->src_attrib : Cache support for transfer source
        - DPR bits ← info->dst_prot : Protection unit support for transfer destination
        - DCA bits ← info->dst_attrib : Cache support for transfer destination
  - channel = 3
    - **Set protection unit support and cache support**
      - CEXT_3 register
        - SPR bits ← info->src_prot : Protection unit support for transfer source
        - SCA bits ← info->src_attrib : Cache support for transfer source
        - DPR bits ← info->dst_prot : Protection unit support for transfer destination
        - DCA bits ← info->dst_attrib : Cache support for transfer destination
  - channel = 4
    - **Set protection unit support and cache support**
      - CEXT_4 register
        - SPR bits ← info->src_prot : Protection unit support for transfer source
        - SCA bits ← info->src_attrib : Cache support for transfer source
        - DPR bits ← info->dst_prot : Protection unit support for transfer destination
        - DCA bits ← info->dst_attrib : Cache support for transfer destination

*Figure 6.25 Initial Setting Function for Protection Unit Support and Cache Support (1/3)*
Set protection unit support and cache support

channel = 5
CHEXT_5 register
SPR bits ← info->src_prot
SCA bits ← info->src_attrib
DPR bits ← info->dst_prot
DCA bits ← info->dst_attrib

channel = 6

Set protection unit support and cache support

channel = 7
CHEXT_6 register
SPR bits ← info->src_prot
SCA bits ← info->src_attrib
DPR bits ← info->dst_prot
DCA bits ← info->dst_attrib

channel = 8

Set protection unit support and cache support

channel = 9
CHEXT_7 register
SPR bits ← info->src_prot
SCA bits ← info->src_attrib
DPR bits ← info->dst_prot
DCA bits ← info->dst_attrib

channel = 10

Set protection unit support and cache support

channel = 11
CHEXT_8 register
SPR bits ← info->src_prot
SCA bits ← info->src_attrib
DPR bits ← info->dst_prot
DCA bits ← info->dst_attrib

Figure 6.26 Initial Setting Function for Protection Unit Support and Cache Support (2/3)
channel = 12

Set protection unit support and cache support

channel = 13

Set protection unit support and cache support

channel = 14

Set protection unit support and cache support

channel = 15

Set protection unit support and cache support

default

return (0)

CEXT_12 register
SPR bits ← info->src_prot
SCA bits ← info->src_attrib
DPR bits ← info->dst_prot
DCA bits ← info->dst_attrib

: Protection unit support for transfer source
: Cache support for transfer source
: Protection unit support for transfer destination
: Cache support for transfer destination

CEXT_13 register
SPR bits ← info->src_prot
SCA bits ← info->src_attrib
DPR bits ← info->dst_prot
DCA bits ← info->dst_attrib

: Protection unit support for transfer source
: Cache support for transfer source
: Protection unit support for transfer destination
: Cache support for transfer destination

CEXT_14 register
SPR bits ← info->src_prot
SCA bits ← info->src_attrib
DPR bits ← info->dst_prot
DCA bits ← info->dst_attrib

: Protection unit support for transfer source
: Cache support for transfer source
: Protection unit support for transfer destination
: Cache support for transfer destination

CEXT_15 register
SPR bits ← info->src_prot
SCA bits ← info->src_attrib
DPR bits ← info->dst_prot
DCA bits ← info->dst_attrib

: Protection unit support for transfer source
: Cache support for transfer source
: Protection unit support for transfer destination
: Cache support for transfer destination

Figure 6.27 Initial Setting Function for Protection Unit Support and Cache Support (3/3)
6.10.18 Get Function of Setting Information to Protection Unit Support and Cache Support

Figure 6.28 to Figure 6.32 show the flowcharts of the Get Function of Setting Information to Protection Unit Support and Cache Support.

In this function, the setting information to the protection unit support and the cache support, for the area including the specified address, is obtained according to the information of the default MMU translation table used in the sample code. When the MMU translation table is customized, it is necessary to modify this function according to the changed contents of the table.

Sample_DMAC_GetMemoryType

L2 cache enabled?

Enabled

Transfer address less than H'0800 0000?

CS0 and CS1 spaces (NOR flash memory)

Obtain setting information to protection unit support and cache support for CS0 and CS1 spaces

Argument *protection ← DMAC_PROT_NONSECURE_NORMAL
Argument *cache_attrib ← DMAC_CACHE_ATTRIB_L2_CACHEABLE

Information to the protection unit support : Non-secure / data / normal access
Information to the cache support : External space access
L2 cache-enabled normal memory

Transfer address less than H'1000 0000?

CS2 and CS3 spaces (SDRAM)

Obtain setting information to protection unit support and cache support for CS2 and CS3 spaces

Argument *protection ← DMAC_PROT_NONSECURE_NORMAL
Argument *cache_attrib ← DMAC_CACHE_ATTRIB_L2_CACHEABLE

Information to the protection unit support : Non-secure / data / normal access
Information to the cache support : External space access
L2 cache-enabled normal memory

Transfer address less than H'1800 0000?

CS4 and CS5 spaces

Obtain setting information to protection unit support and cache support for CS4 and CS5 spaces

Argument *protection ← DMAC_PROT_NONSECURE_NORMAL
Argument *cache_attrib ← DMAC_CACHE_ATTRIB_STRONGLY

Information to the protection unit support : Non-secure / data / normal access
Information to the cache support : External space access
Strongly ordered memory

A B C

Figure 6.28 Get Function of Setting Information to Protection Unit Support and Cache Support (1/5)
Transfer address less than H'20A0 0000?

Large-capacity on-chip RAM space

Obtain setting information to protection unit support and cache support for large-capacity on-chip RAM space

Argument *protection ← DMAC_PROT_NONSECURE_NORMAL
Argument *cache_attrib ← DMAC_CACHE_ATTRIB_INTERNAL_AREA
Information to the protection unit support : Non-secure / data / normal access
Information to the cache support : Internal space access

Transfer address less than H'2000 0000?

On-chip peripheral module and reserved area

Obtain setting information to protection unit support and cache support for on-chip peripheral module and reserved area

Argument *protection ← DMAC_PROT_SECURE_NORMAL
Argument *cache_attrib ← DMAC_CACHE_ATTRIB_INTERNAL_AREA
Information to the protection unit support : Secure / data / normal access
Information to the cache support : Internal space access

Obtain setting information to protection unit support and cache support for SPI multi I/O bus space 1 and 2

Argument *protection ← DMAC_PROT_NONSECURE_NORMAL
Argument *cache_attrib ← DMAC_CACHE_ATTRIB_L2_CACHEABLE
Information to the cache support : External space access
L2 cache-enabled normal memory

Figure 6.29 Get Function of Setting Information to Protection Unit Support and Cache Support (2/5)
Transfer address less than H'5000 0000?

CS0 and CS1 mirror spaces

Obtain setting information to protection unit support and cache support for CS0 and CS1 mirror spaces

Argument "protection ← DMAC_PROT_NONSECURE_NORMAL"
Argument "cache_attrib ← DMAC_CACHE_ATTRIB_L2_NON_CACHEABLE"
Information to the protection unit support : Non-secure / data / normal access
Information to the cache support : External space access
L2 cache-disabled normal memory

Figure 6.30  Get Function of Setting Information to Protection Unit Support and Cache Support (3/5)
Figure 6.31  Get Function of Setting Information to Protection Unit Support and Cache Support (4/5)
Figure 6.32 Get Function of Setting Information to Protection Unit Support and Cache Support (5/5)
6.11 Running the Sample Code

The sample code is operated by entering commands in the terminal software running on a host PC connected to the GENMAI board via the serial interface.

After supplying power to the GENMAI board, the message (1) shown in Figure 6.33 is output. To run the DMAC sample code, input "DMAC" + "Enter" key subsequent to the "SAMPLE>" prompt. When the message (2) is output, input "1" + "Enter" key to run sample code 1 or "2" + "Enter" key to run sample code 2 subsequent to "DMAC SAMPLE>" prompt.

By inputting "HELP" + "Enter" key, the sample code information (3) is displayed. "EXIT" + "Enter" key terminates the DMAC sample code operation.

Ver.X.XX and Ver.Y.YY shown in Figure 6.33 indicate the main processing version of sample code and the DMAC sample code version respectively.

<table>
<thead>
<tr>
<th>Display Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>RZ/A1H CPU Board Sample Program. Ver.X.XX  (1)</td>
</tr>
<tr>
<td>Copyright (C) 2015 Renesas Electronics Corporation. All rights reserved.</td>
</tr>
<tr>
<td>select sample program.</td>
</tr>
<tr>
<td>SAMPLE&gt;</td>
</tr>
<tr>
<td>RZ/A1H DMAC Sample Program. Ver.Y.YY  (2)</td>
</tr>
<tr>
<td>Copyright (C) 2015 Renesas Electronics Corporation. All rights reserved.</td>
</tr>
<tr>
<td>select sample program.</td>
</tr>
<tr>
<td>DMAC SAMPLE&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DMAC SAMPLE&gt; help  (3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 : Single DMA transfer (3 times trigger)</td>
</tr>
<tr>
<td>- Request : Auto request</td>
</tr>
<tr>
<td>- DMA mode : Register mode</td>
</tr>
<tr>
<td>- Transfer : On-chip RAM -&gt; SDRAM</td>
</tr>
<tr>
<td>2 : 3 continuation DMA transfer</td>
</tr>
<tr>
<td>- Request : Auto request</td>
</tr>
<tr>
<td>- DMA mode : Register mode</td>
</tr>
<tr>
<td>- Transfer : On-chip RAM -&gt; SDRAM</td>
</tr>
<tr>
<td>EXIT : Exit DMAC sample</td>
</tr>
<tr>
<td>DMAC SAMPLE&gt;</td>
</tr>
</tbody>
</table>

Figure 6.33 Terminal Display at DMAC Sample Code Startup
7. Notes on Usage of Sample Code

For DMA transfers, when an L1 cache-enabled area is specified as the transfer source or the transfer destination, it is necessary to consider the handling of software to avoid malfunctions in coherencies or due to speculative cache line fills, as described below. Meanwhile, when an L1 cache-disabled area (or an area in other than the normal memory attribute) is specified as the transfer source or the transfer destination, such malfunctions do not occur.

7.1 Coherency between L1 Caches and External Memory or Large-Capacity On-Chip RAM

The DMAC does not access memory areas via L1 caches. Thus, when the CPU and DMAC share and access an area for which the L1 cache is enabled, it is necessary to ensure the coherency between the contents of the L1 cache and the contents of the external memory or large-capacity on-chip RAM, by flushing and writing back the cache with software.

In this sample code, the DMA transfer source buffer is located in the L1 cache-enabled large-capacity on-chip RAM, and be initialized with CPU access. The coherency between the L1 cache and the large-capacity on-chip RAM is ensured, by writing back and flushing the L1 cache before a DMA transfer starts.

For the DMA transfer destination buffer, when the corresponding area is accessed by the CPU and cached to the L1 cache before a DMA transfer starts, it is necessary to ensure the coherency as well as the transfer source buffer.

7.2 Notes on DMA Transfer with Speculative Cache Line Fill

The Cortex-A9, the CPU core in the RZ/A1H, supports speculation. When the DMA transfer destination buffer is located in a cache-enabled area, the access by the CPU to the transfer destination buffer, which should occur after the completion of the DMA transfer, may occur before the completion because of the speculation. Consequently, the data before the completion of the transfer may be read to the cache. This phenomenon is named a speculative cache line fill.

In this sample code, the DMA transfer destination buffer is located in the L1 cache-enabled CS2 SDRAM space. To avoid malfunctions due to speculative cache line fills, the L1 data cache is flushed after the completion of the DMA transfer. With this measure, even when a speculative cache line fill occurs, the CPU can read the correct data in the transfer destination buffer after the completion of the DMA transfer by flushing the L1 data cache once.

In addition, in this sample code, after all cache lines in the L1 data caches are flushed, the processing of this sample code is ended with only the confirmation of the data in the transfer destination buffer. When the processing of software continues after flushing the cache lines, the data in anything such as work memory, if cached, would be deleted by the flushing. Consequently, the processing would not be able to continue normally. Therefore the user should consider the measures, such as the installation of the operation for caches to flush only the cache lines to which the DMA transfer destination buffer is cached.
8. Sample Code
Sample code can be downloaded from the Renesas Electronics website.

9. Reference Documents
User's Manual: Hardware
RZ/A1H Group User's Manual: Hardware
The latest version can be downloaded from the Renesas Electronics website.

The latest version can be downloaded from the Renesas Electronics website.

The latest version can be downloaded from the ARM website.

ARM Generic Interrupt Controller Architecture Specification Architecture version 1.0
The latest version can be downloaded from the ARM website.

ARM Cortex™-A9 (Revision: r3p0) Technical Reference Manual
The latest version can be downloaded from the ARM website.

ARM CoreLink™ Level 2 Cache Controller L2C-310 (Revision: r3p2) Technical Reference Manual
The latest version can be downloaded from the ARM website.

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

ARM Software Development Tools (ARM Compiler toolchain, ARM DS-5 etc) can be downloaded from the ARM website.
The latest version can be downloaded from the ARM website.
Website and Support

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

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

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

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rev.1.00</td>
<td>Jul. 11, 2014</td>
<td>-</td>
<td>First edition issued</td>
</tr>
<tr>
<td>Rev.1.01</td>
<td>Aug. 28, 2015</td>
<td></td>
<td>Reflected the update contents of &quot;RZ/A1H Group Example of Initialization Rev.1.01&quot; application note.</td>
</tr>
</tbody>
</table>

- **P11** Table 6.2 Sections Used (1/2)
  - Changed the execution area for the CODE_IO_REGRW section in the table from "FLASH" to "LRAM".

- **P12** Table 6.3 Sections Used (2/2)
  - Added the CODE_CACHE_OPERATION section to the table.
  - Added precautions about the CODE_CACHE_OPERATION section to Note 3.

- **P13** Figure 6.3 Section Assignment
  - Provided information due to the CODE_IO_REGRW section arrangement being changed and the CODE_CACHE_OPERATION section being added.

- **P4** 1. Specifications
  - Changed the areas set as the transfer source and the transfer destination of the DMA transfers from cache-disabled areas to cache-enabled ones.

- **P12** Table 6.3 Sections Used (2/2)
  - Changed the BSS_DMAC_SAMPLE_INTERNAL_RAM section assignment from the L1 cache-disabled area to the L1 cache-enabled one.
  - Changed the BSS_DMAC_SAMPLE_SDRAM section assignment from the L1 cache-disabled area to the L1 and L2 cache-enabled one.

- **P15** Table 6.6 Constants Used in Sample Code
  - Added the definitions of the settings for the protection unit support and the cache support.

- **P16** Figure 6.4 Structure/Union Used in Sample Code
  - Added the structure dmac_cache_prot_info in which the setting information to the protection unit support and the cache support is stored.

- **P17** Table 6.7 Static Variables
  - In note, changed the BSS_DMAC_SAMPLE_INTERNAL_RAM section assignment from the L1 cache-disabled large-capacity on-chip RAM space to the L1 cache-enabled one.
  - Similarly, changed the BSS_DMAC_SAMPLE_SDRAM section assignment from the L1 cache-disabled CS2 SDRAM to the L1 and L2 cache-enabled one.

- **P18** Figure 6.5 Memory Allocation of Transfer Source Transfer Destination Buffers Used in DMA Transfer
  - Changed both the BSS_DMAC_SAMPLE_INTERNAL_RAM section assignment and the BSS_DMAC_SAMPLE_SDRAM section assignment from the cache-disabled area to the cache-enabled one.

- **P19** Table 6.8 Functions
  - Added the Initial Setting Function (DMAC_InitCacheProt) and the Get Function of the setting information (Sample_DMAD_GetMemoryType) to the protection unit support and the cache support.
6.9 Function Specifications
Added the DMAC_InitCacheProt function and the Sample_DMAM_GetMemoryType function.

P27 Added Table 6.9, Setting Information to Protection Unit Support and Cache Support, Used in Sample Code
The table describes the setting information to the protection unit support and the cache support when the RZ/A1H internal space and external space are specified as the transfer source and the transfer destination of the DMA transfers, in the sample code.

P31, P32 6.10.4 Sample Function with Auto-Request as DMA Transfer Request Type
Added the operations for caches and the processing of the protection unit support and the cache support to the flowcharts, according to the change of the areas set as the transfer source and the transfer destination of the DMA transfers from cache-disabled areas to cache-enabled ones.

P33, P34 6.10.5 Sample Function with Auto-Request as DMA Transfer Request Type (Continuous Transfer)
Added the operations for caches and the processing of the protection unit support and the cache support to the flowcharts, according to the change of the areas set as the transfer source and the transfer destination of the DMA transfers from cache-disabled areas to cache-enabled ones.

P47 to P49 Added 6.10.17, Initial Setting Function for Protection Unit Support and Cache Support
The section describes the flowcharts of the Initial Setting Function for the protection unit support and the cache support.

P50 to P54 Added 6.10.18, Get Function of Setting Information to Protection Unit Support and Cache Support
The section describes the flowcharts of the Get Function of the setting information to the protection unit support and the cache support.

P56 Added 7., Notes on Usage of Sample Code
The section describes the notes on coherencies and speculative cache line fills when cache-enabled areas are set as the transfer source and the transfer destination of the DMA transfers.
General Precautions in the Handling of MPU/MCU Products

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

1. 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 LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on
   The state of the product is undefined at the moment 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 moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

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

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

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

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

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

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

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

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

   "High-Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crim systems; and safety equipment etc.

Renesas Electronics products are neither intended nor 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 damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.

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

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

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

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

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

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

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

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

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

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

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

Renesas Electronics America Inc.
2901 South Boulevard, Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6500, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Mead, Milford Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-583-100, Fax: +44-1628-583-900

Renesas Electronics (Shanghai) Co., Ltd.
Arcadia Building 10, 40472 Dusseldorf, Germany
Tel: +49-211-6503-300, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Jupiter Plaza, No.27, Zhongshan Lu, Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Hong Kong) Limited
Unit 301, Tower A, Central Tower, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-8998, Fax: +86-21-2226-9999

Renesas Electronics Malaysia Sdn Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9300, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-45268700, Fax: +91-80-45268777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-258-3131, Fax: +82-2-258-3141

© 2015 Renesas Electronics Corporation. All rights reserved.