RZ/T1 Group

DMA Sample Program

R01AN2598EJ0140
Rev.1.40
Jun. 07, 2018

Introduction

This application note explains a Direct Memory Access (DMA) sample program that supports the RSK RZ/T1 Evaluation Board equipped with RZ/T1 Group MCU.

This program uses the DMA functionality to copy the contents of the transfer source areas 1, 2, and 3 (256 bytes each) in the internal RAM to the transfer destination areas 1, 2, and 3 (256 bytes each) in the internal RAM.

- Uses channel 0 of DMAC0.
- Uses a software request to start DMAC.
- Specifies 8 bits as the transfer data unit.
- Specifies block transfer as the transfer mode.
- Uses DMA transfer completion and DMA error interrupts.

Target Devices

RZ/T1

When applying the sample program covered in this application note to another microcomputer, modify the program according to the specifications for the target microcomputer and conduct an extensive evaluation of the modified program.

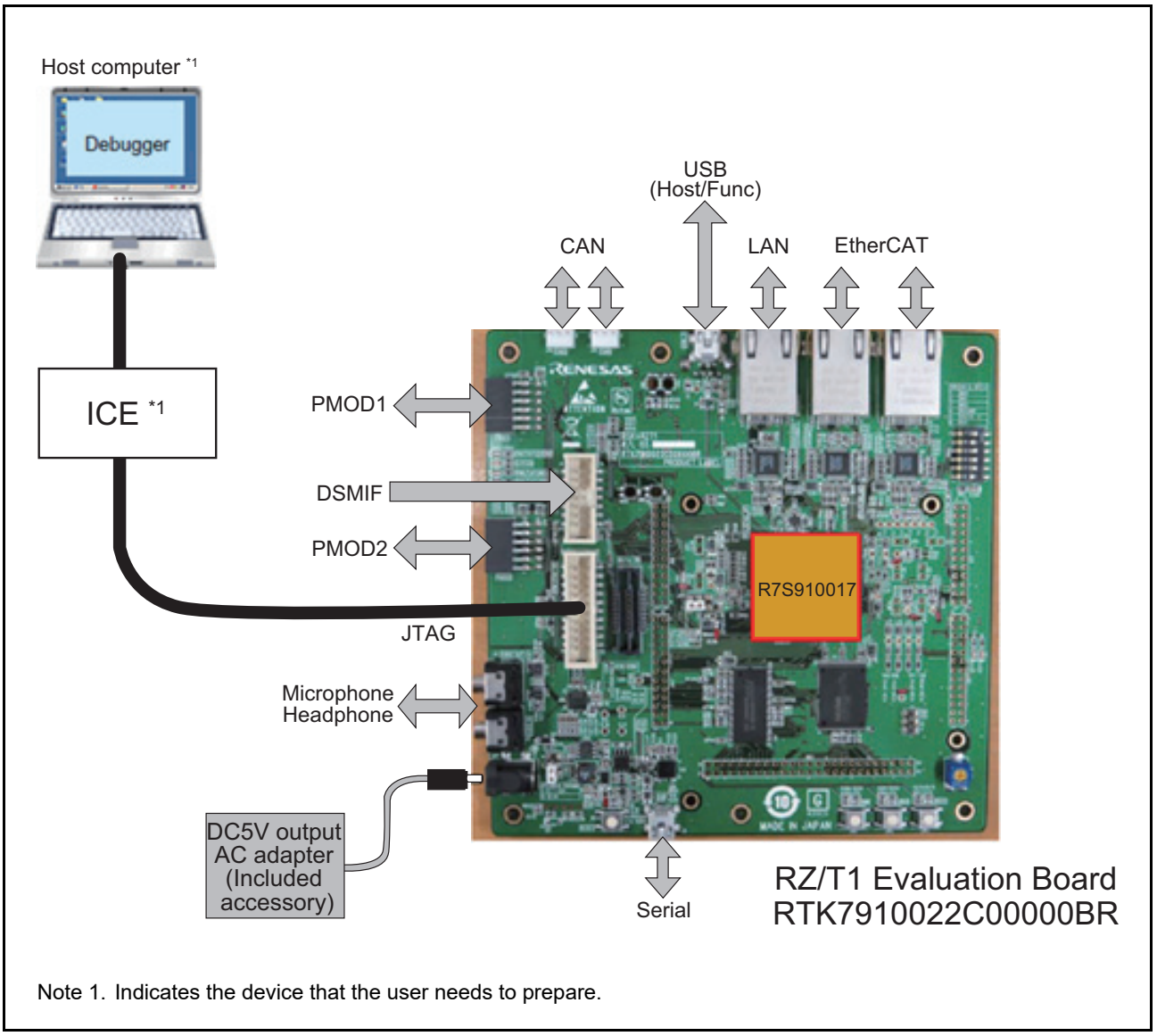
Table of Contents

1. Specifications

Table 1.1 lists the peripheral functions to be used and their applications, and Figure 1.1 shows the operating environment where the sample program is assumed to run.

Table 1.1 Peripheral Functions and Applications

Peripheral Function	Application
RZ/T1 built-in DMA controller (DMACAa)	DMA transfer for channel 0 of DMAC0 (Unit0)
RZ/T1 built-in interrupt controller (ICUA)	Interrupt control • DMA transfer completion by software activation from DMAC0 (vector number 251) • DMA transfer error with DMAC0 (vector number 293)



Note 1. Indicates the device that the user needs to prepare.

Figure 1.1 Operating Environment

2. Operating Environment

The sample code covered in this application note is for the environment below.

Table 2.1 Operating Environment

Item	Description
Microcomputer	RZ/T1 Group
Operating frequency	CPUCLK = 450MHz
Operating voltage	3.3 V
Integrated Development Environment	Manufactured by IAR Systems Embedded Workbench® for Arm Version 8.20.2 Manufactured by Arm DS-5™ 5.26.2 Manufactured by RENESAS e2studio 6.1.0
Operating mode	SPI boot mode 16-bit bus boot mode
Board	RZ/T1 Evaluation Board (RTK7910022C00000BR)
Device (functions to be used on the board)	• NOR flash memory (connected to CS0 and CS1 spaces) Manufacturer: Macronix International Co., Ltd., Model: MX29GL512FLT2I-10Q • SDRAM (connected to CS2 and CS3 spaces) Manufacturer: Integrated Silicon Solution Inc, Model: IS42S16320D-7TL • Serial flash memory Manufacturer: Macronix International Co., Ltd., Model: MX25L51245G

3. Related Application Note

The application note related to this application note is listed below for reference.

- Application Note: RZ/T1 Group Initial Settings (R01AN2554EJ)

Note: For any registers not covered by this application note, use the values specified in Application Note: RZ/T1 Group Initial Settings.

4. Peripheral Functions

The basics of the operating modes, DMA controller (DMACAa), and interrupt controller (ICUA) are described in the RZ/T1 Group User's Manual: Hardware.

5. Hardware

5.1 Hardware Configuration

Because this sample program does not use the DMA functionality for external requests or external interrupts, the DMA pins are not connected to external devices.

5.2 Pins

The DMA sample driver does not use any pins.

6. Software

6.1 Operation Outline

Table 6.1 Operation Outline presents a functional overview of the DMA sample program. Figure 6.1 shows the system block diagram for this program.

Table 6.1 Operation Outline

Function	Outline
Processing outline	• This program uses the DMA functionality to copy the contents of the transfer source areas 1, 2, and 3 (256 bytes each) in the internal RAM to the transfer destination areas 1, 2, and 3 (256 bytes each) in the internal RAM.
DMA setup	• Channel: Channel 0 of DMAC0 • DMAC startup trigger: Software requests • Transfer data unit: 8 bits • Transfer mode: Block transfer • Interrupt request: Transfer completion interrupts for channel 0 of DMAC0, and transfer error interrupts for DMAC0
Interrupt handler setup	• A DMA transfer completion interrupt handler is implemented into the sample program. • The handler counts the number of DMA transfer completions.
Callback function	• A callback function for DMA errors is implemented into the sample program and registered for the DMA driver. • The callback function sets a flag to notify the main processing of an error.
Result check	• Data at the source is compared with data at the destination to verify that the data has been copied.

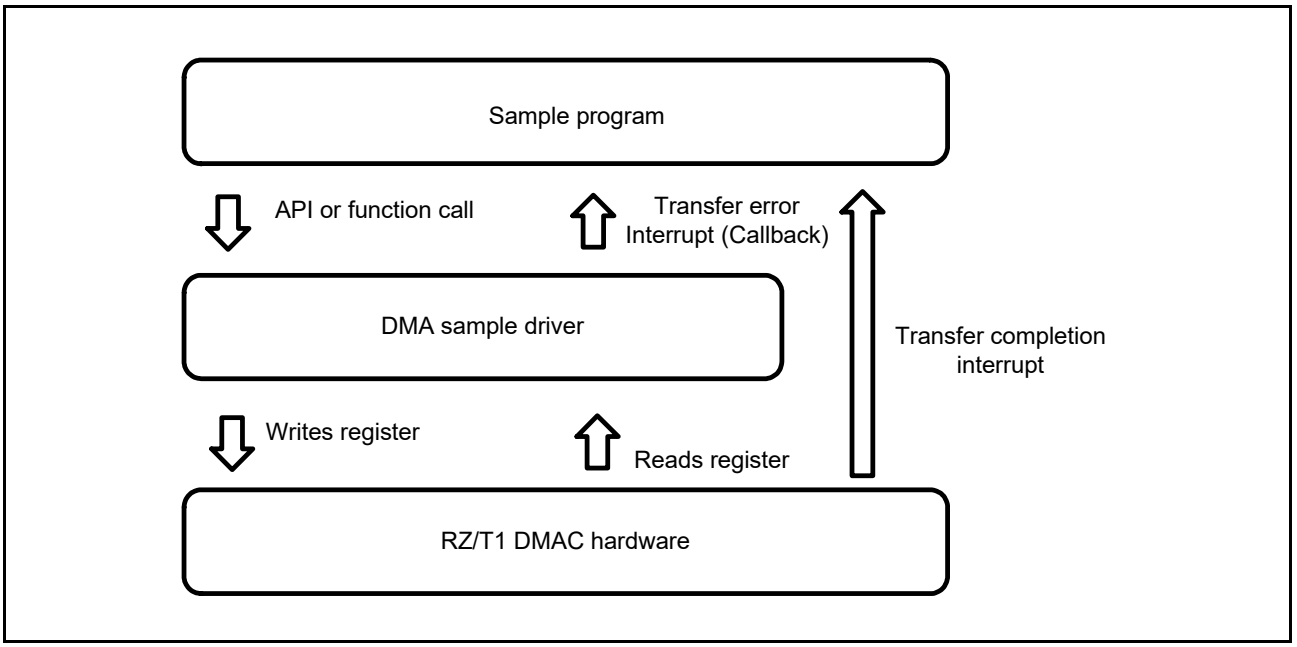


Figure 6.1 System Block Diagram

6.1.1 Project Setup

How to set up projects used in the EWARM development environment is described in the Application Note: RZ/T1 Group Initial Settings.

6.1.2 Preparation

There is no need to prepare for executing this sample program.

6.2 Memory Mapping

Memory mapping for the address spaces in the RZ/T1 Group MCU and the memory in the RZ/T1 Evaluation Board is described in the Application Note: RZ/T1 Group Initial Settings.

6.2.1 Section Allocation for the Sample program

The sections used by the sample program, the section allocation for the sample program in the initial state (load view), and the section allocation for the sample program after the scatter loading function is used (execution view) are described in the Application Note: RZ/T1 Group Initial Settings.

6.2.2 MPU Setup

MPU setup is described in the Application Note: RZ/T1 Group Initial Settings.

6.2.3 Exception Processing Vector Table

Exception processing vector tables are described in the Application Note: RZ/T1 Group Initial Settings.

6.3 Interrupts

The following table lists interrupts for this sample program.

Table 6.2 Interrupts for the Sample Program

Interrupt (Source ID)	Priority	Process Outline
Software activation Unit0 for DMA transfer (DMASRQ0)	3	DMA transfer completion interrupt by software activation for DMA transfer. This process counts the number of DMA transfer completions when all transfers are complete.
Transfer error Unit0 for DMA transfer (DMAERR0)	3	Transfer error for DMA transfer. When a transfer error occurs, this process sets information to notify the main processing of the error.

6.4 Fixed-Width Integer Types

Table 6.3 shows the fixed-width integer types for the sample program.

Table 6.3 Fixed-Width Integer Types for the Sample Program

Symbol	Description
int8_t	8-bit signed integer (defined in the standard library)
int16_t	16-bit signed integer (defined in the standard library)
int32_t	32-bit signed integer (defined in the standard library)
int64_t	64-bit signed integer (defined in the standard library)
uint8_t	8-bit unsigned integer (defined in the standard library)
uint16_t	16-bit unsigned integer (defined in the standard library)
uint32_t	32-bit unsigned integer (defined in the standard library)
uint64_t	64-bit unsigned integer (defined in the standard library)

6.5 Constants/Error Codes

Table 6.4 shows the constants for the DMA sample program, Table 6.5 and Table 6.6 show the constants for the DMA sample driver, and Table 6.7 shows the error codes for the DMA sample driver.

Table 6.4 Constants for the DMA Sample Program

Constant Name	Setting Value	Description
DMA_DATA_LENGTH	0x00000100	Total byte length of DMA transfer data
DMA_DESCRIPTOR_NUM	3	Number of descriptors to be set
DMA_CHITVL_VAL	0x0000	The setting value of the channel interval register Sets the DMA transfer interval.
DMA_DSCITVL_VAL	0x00	The setting value of the descriptor interval register Sets the descriptor read interval.

Table 6.5 Constants for the DMA Sample Driver (Configurable)

Constant Name	Setting Value	Description
DMA_LINK_BUF_NUM	3	Number of descriptor buffers in link mode
DMA_CMNCR_INIT	0x00001010	The initial value of the CMNCR register. Receives a refresh request during DMA burst transfer. Outputs DACK0/1/2 as active-low signals. Outputs TEND0/1/2 as active-low signals.
DMA_DMAC0_DCTRL_A_INIT	0x00000000	The initial value of the DCTRL_A register. DMAC0 channels 0 to 7: Fixed priority mode
DMA_DMAC0_DCTRL_B_INIT	0x00000000	The initial value of the DCTRL_B register. DMAC0 channels 8 to 15: Fixed priority mode
DMA_DMAC1_DCTRL_A_INIT	0x00000000	The initial value of the DCTRL_A register. DMAC1 channels 0 to 7: Fixed priority mode
DMA_DMAC1_DCTRL_B_INIT	0x00000000	The initial value of the DCTRL_B register. DMAC1 channels 8 to 15: Fixed priority mode
DMA_CHITVL_INIT	0x00000000	The initial value of the CHITVL_(ch) (ch=1 to 15) register. Sets the DMA transfer interval (0).
DMA_SCNT_INIT	0x00000000	The initial value of the SCNT_(ch) (ch=1 to 15) register. Sets the continuous-access address space size (0) for the DMA transfer source.
DMA_SSKP_INIT	0x00000000	The initial value of the SSKP_(ch) (ch=1 to 15) register. Sets the skip address space size (0) for the DMA transfer source.
DMA_DCNT_INIT	0x00000000	The initial value of the DCNT_(*1) register. Sets the continuous-access address space size (0) for the DMA transfer destination.
DMA_DSKP_INIT	0x00000000	The initial value of the DSKP_(*1) register. Sets the skip address space size (0) for the DMA transfer destination.
DMA_CFG_PARAM_CHECKING_ENABLE	1	DMA parameter check-enable switch 0: Disable, 1: Enable
DMA_IR_PRIORITY_293_DMAERR_U0	3	Interrupt priority level for vector 293 (DMA Unit0 error)
DMA_IR_PRIORITY_294_DMAERR_U1	3	Interrupt priority level for vector 294 (DMA Unit1 error)

Note 1. 1 to 15

Table 6.6 Constants for the DMA Sample Driver (1 / 2)

Constant Name	Setting Value	Description
DMA_HVA_WRITE_DATA	0x00000000u	Data to be written to HVA
DMA_NUM_UNITS	2	Total number of DMA units
DMA_NUM_CHANNELS	16	Number of channels per DMA unit
DMA_NUM_REG_TBL	11	Number of register tables
DMA_UNIT0	0	DMA Unit 0
DMA_UNIT1	1	DMA Unit 1
DMA_CHANNEL0	0	DMA channel 0
DMA_CHANNEL1	1	DMA channel 1
DMA_CHANNEL2	2	DMA channel 2
DMA_CHANNEL3	3	DMA channel 3
DMA_CHANNEL4	4	DMA channel 4
DMA_CHANNEL5	5	DMA channel 5
DMA_CHANNEL6	6	DMA channel 6
DMA_CHANNEL7	7	DMA channel 7
DMA_CHANNEL8	8	DMA channel 8
DMA_CHANNEL9	9	DMA channel 9
DMA_CHANNEL10	10	DMA channel 10
DMA_CHANNEL11	11	DMA channel 11
DMA_CHANNEL12	12	DMA channel 12
DMA_CHANNEL13	13	DMA channel 13
DMA_CHANNEL14	14	DMA channel 14
DMA_CHANNEL15	15	DMA channel 15
BITSHIFT_4	4	Number of bits to shift: 4 bits
BITSHIFT_16	16	Number of bits to shift: 16 bits
CHANNEL_MASK_4	0x0F	Channel mask: 4 bits
REGISTER_MASK_3	0x7	Register mask: 3 bits
REGISTER_SET_BIT3	0x8	A value for setting bit 3 in the register
REG_INDEX_CHSTAT	0	Table index number of the CHSTAT register
REG_INDEX _CHCTRL	1	Table index number of the CHCTRL register
REG_INDEX _CHCFG	2	Table index number of the CHCFG register
REG_INDEX _CHITVL	3	Table index number of the CHITVL register
REG_INDEX _NXLA	4	Table index number of the NXLA register
REG_INDEX _CRLA	5	Table index number of the CRLA register
REG_INDEX _SCNT	6	Table index number of the SCNT register
REG_INDEX _SSKP	7	Table index number of the SSKP register
REG_INDEX _DCNT	8	Table index number of the DCNT register
REG_INDEX _DSKP	9	Table index number of the DSKP register
REG_INDEX _DMASEL	10	Table index number of the DMASEL register
DATARAM_MIRR_STR_ADR	0x22000000	Starting address of the Data RAM area
DATARAM_MIRR_END_ADR	0x22080000	Ending address of the Data RAM area
DATARAM_MIRR_ADR_OFS	0x02000000	Address offset for the non-cached area and mirrored (cached) area in the Data RAM area
INSTRAM_MIRR_STR_ADR	0x24000000	Starting address of the Instruction RAM area
INSTRAM_MIRR_END_ADR	0x24080000	Ending address of the Instruction RAM area

Table 6.6 Constants for the DMA Sample Driver (2 / 2)

Constant Name	Setting Value	Description
INSTRAM_MIRR_ADR_OFS	0x20000000	Address offset for the non-cached area and mirrored (cached) area in the Instruction RAM area
SPIBSC_MIRR_STR_ADR	0x30000000	Starting address of the SPI multi-I/O bus address space
SPIBSC_MIRR_END_ADR	0x34000000	Ending address of the SPI multi-I/O bus address space
SPIBSC_MIRR_ADR_OFS	0x20000000	Address offset for the non-cached area and mirrored (cached) area in the SPI multi-I/O bus address space
EX_MIRR_STR_ADR	0x40000000	Starting address of the external address space
EX_MIRR_END_ADR	0x58000000	Ending address of the external address space
EX_MIRR_ADR_OFS	0x20000000	Address offset for the non-cached area and mirrored (cached) area in the external address space
DMA_RZT1_VERSION_MAJOR	1	Major Version
DMA_RZT1_VERSION_MINOR	0	Minor Version

Table 6.7 Error Codes for the DMA Sample Driver

Constant Name	Setting Value	Description
DMA_SUCCESS	0	Successful
DMA_ERR_BAD_UNIT	1	Invalid unit number
DMA_ERR_BAD_CHAN	2	Invalid channel number
DMA_ERR_CH_NOT_OPENED	3	Channel not open
DMA_ERR_CH_NOT_CLOSED	4	Cannot close (already open)
DMA_ERR_UNKNOWN_CMD	5	Unknown command
DMA_ERR_INVALID_ARG	6	Invalid parameter settings
DMA_ERR_NULL_PTR	7	Empty parameter (NULL)
DMA_ERR_BUFF_FULL	8	No free space in the descriptor setting buffer

6.6 Structures, Unions, and Enumerated Types

Figure 6.2 shows the structures and unions for the DMA sample driver, and Figure 6.3 shows the enumerated types for the DMA sample driver.

```
/* vector setting */
typedef struct dma_vector_para_s
{
    uint16_t vector_no;                         /* vector no. */
    uint32_t priority;                          /* priority */
    void   (__irq __arm * pcallback_isr)(void); /* pointer to user callback interrupt function. */
} dma_vector_para_t;

/* link desctiptor format. */
typedef struct dma_link_fmt_s
{
    union {
        struct{
            uint16_t        sts         :16;    /* transfer size at the time of DSCFM = 3. */
            uint8_t                     :7;
            uint8_t         d           :1;     /* descriptor status. */
            uint8_t         lv          :1;     /* descriptor valid/invalid status. */
            uint8_t         le          :1;     /* link status of descriptor. */
            uint8_t         wbd         :1;     /* mask of write back. */
            uint8_t                     :1;
            uint8_t         dscfm       :4;     /* format of descriptor */
        } BIT;
        uint32_t LONG;
    } hd;
    uint32_t sa;                                /* source address. */
    uint32_t da;                                /* destination address. */
    uint32_t tb;                                /* transaction byte. */
    uint32_t chcfg;                             /* channel config. */
    uint32_t chitvl;                           /* channel interval. */
    uint32_t ex;                                /* channel extension. */
    uint32_t nla;                               /* next link address. */
} dma_link_fmt_t;

/* DMA handle */
typedef struct dma_handle_s
{
    uint8_t unit;
    uint8_t channel;
    dma_vector_para_t   vector_para;            /* vector parameter. */
    dma_ch_sts_t dma_ch_sts;                    /* channel state */
    void   (*pcallback_err)(void);              /* pointer to user callback function. */
    uint8_t write_index;                        /* write index of link buffer */
    dma_link_fmt_t link_buf[DMA_LINK_BUF_NUM];  /* link buffer */
    dma_mod_skip_t sskip_mode;                  /* source skip mode */
    dma_mod_skip_t dskip_mode;                  /* destionation skip mode */
} dma_handle_t;
```

```
/* link parameter of R_DMA_Control Function. */
typedef struct dma_control_link_para_s
{
    uint32_t sa;                                /* source address. */
    uint32_t da;                                /* destination address. */
    uint32_t tb;                                /* transaction byte. */
    union {
        struct{
            chcfg_sel_t         sel         :3; /* Channel Select */
            chcfg_reqd_t        reqd        :1; /* DMA Activation Request Source Select. */
            chcfg_reqmtd_t      reqmtd      :3; /* Method of Detecting DMA Transfer Request Signals. */
            uint8_t                         :1;
            chcfg_am_t          am          :3; /* ACK Mode. */
            chcfg_drrp_t        drrp        :1; /* Descriptor Reload Enable. */
            chcfg_sds_t         sds         :4; /* Source Data Size. */
            chcfg_dds_t         dds         :4; /* Destination Data Size. */
            chcfg_sad_t         sad         :1; /* Source Address Count Direction. */
            chcfg_dad_t         dad         :1; /* Destination Address Count Direction. */
            chcfg_tm_t          tm          :1; /* Transfer Mode. */
            chcfg_wonly_t       wonly       :1; /* Write-Only Mode. */
            chcfg_dem_t         dem         :1; /* Transfer Completion Interrupt Mask. */
            uint8_t                         :1;
            chcfg_dim_t         dim         :1; /* Descriptor Interrupt Mask. */
            chcfg_sbe_t         sbe         :1; /* Buffer Flush Enable. */
            chcfg_rsel_t        rsel        :1; /* Next Register Select. */
            chcfg_rsw_t         rsw         :1; /* RSEL Reverse. */
            chcfg_ren_t         ren         :1; /* Register Set Enable. */
            chcfg_dms_t         dms         :1; /* DMA Mode Select. */
        } BIT;
        unsigned long LONG;
    } chcfg;
    uint16_t chitvl;                            /* channel interval. */
} dma_control_link_para_t;

/* skip parameter of R_DMA_Control Function. */
typedef struct dma_control_skip_para_s
{
    /* SCNT register */
    uint32_t        scnt;                       /* Source Continuous Access Size. */

    /* SSKP register */
    uint32_t        sskp;                       /* Source Skip Size. */

    /* DCNT register */
    uint32_t        dcnt;                       /* Destination Continuous Access Size. */

    /* DSKP register */
    uint32_t        dskp;                       /* Destination Skip Size. */
} dma_control_skip_para_t;
```

```
/* descriptor interval parameter of R_DMA_Control Function. */
typedef struct dma_control_dscitvl_para_s
{
    /* DSCITVL register */
    uint8_t        dscitvl;                    /* Descriptor Interval. */
} dma_control_dscitvl_para_t;

/* CHSTAT register */
typedef union dma_chstat_reg_u
{
    struct{
        uint8_t         en          :1;     /* DMA Activation Enable. */
        uint8_t         rqst        :1;     /* DMA Transfer Request. */
        uint8_t         tact        :1;     /* DMAC Operating Status. */
        uint8_t         sus         :1;     /* Suspend. */
        uint8_t         er          :1;     /* DMA Error. */
        uint8_t         end         :1;     /* DMA Transfer Completion Interrupt. */
        uint8_t                     :1;
        uint8_t         sr          :1;     /* Next Register Select. */
        uint8_t         dl          :1;     /* Descriptor Load. */
        uint8_t         dw          :1;     /* Descriptor Write Back. */
        uint8_t         der         :1;     /* Descriptor Error. */
        uint8_t         mode        :1;     /* DMA Mode. */
        uint8_t                     :4;
        uint8_t         intm        :1;     /* Interrupt Request Mask. */
        uint8_t         dmarqm      :1;     /* DMA Activation Request Mask. */
        uint8_t         swprq       :1;     /* Forced Ejection Request. */
        uint8_t                     :5;
        uint8_t         dnum        :8;     /* Amount of Data in the Buffer. */
    } BIT;
    unsigned long LONG;
} dma_chstat_reg_t;

/* CHCFG register */
typedef union dma_chcfg_reg_u
{
    /* CHCFG register */
    struct{
        chcfg_sel_t         sel         :3; /* Channel Select */
        chcfg_reqd_t        reqd        :1; /* DMA Activation Request Source Select. */
        chcfg_reqmtd_t      reqmtd      :3; /* Method of Detecting DMA Transfer Request Signals. */
        uint8_t                         :1;
        chcfg_am_t          am          :3; /* ACK Mode. */
        chcfg_drrp_t        drrp        :1; /* Descriptor Reload Enable. */
        chcfg_sds_t         sds         :4; /* Source Data Size. */
        chcfg_dds_t         dds         :4; /* Destination Data Size. */
        chcfg_sad_t         sad         :1; /* Source Address Count Direction. */
        chcfg_dad_t         dad         :1; /* Destination Address Count Direction. */
```

```
        chcfg_tm_t          tm          :1; /* Transfer Mode. */
        chcfg_wonly_t       wonly       :1; /* Write-Only Mode. */
        chcfg_dem_t         dem         :1; /* Transfer Completion Interrupt Mask. */
        uint8_t                         :1;
        chcfg_dim_t         dim         :1; /* Descriptor Interrupt Mask. */
        chcfg_sbe_t         sbe         :1; /* Buffer Flush Enable. */
        chcfg_rsel_t        rsel        :1; /* Next Register Select. */
        chcfg_rsw_t         rsw         :1; /* RSEL Reverse. */
        chcfg_ren_t         ren         :1; /* Register Set Enable. */
        chcfg_dms_t         dms         :1; /* DMA Mode Select. */
    } BIT;
    unsigned long LONG;
} dma_chcfg_reg_t;

/* CHCTRL register */
typedef union dma_chctrl_reg_u
{
    struct{
        chctrl_seten_t      seten       :1; /* DMA Activation Enable. */
        chctrl_clren_t      clren       :1; /* DMA Activation Enable Clear. */
        uint8_t                         :1;
        chctrl_swrst_t      swrst       :1; /* Software Reset. */
        chctrl_clrrq_t      clrrq       :1; /* DMA Transfer Request Clear. */
        chctrl_clrend_t     clrend      :1; /* END Clear. */
        uint8_t                         :1;
        chctrl_clrder_t     clrder      :1; /* DER Clear. */
        chctrl_setsus_t     setsus      :1; /* Suspend Request. */
        chctrl_clrsus_t     clrsus      :1; /* Suspend Clear. */
        uint8_t                         :2;
        chctrl_setren_t     setren      :1; /* REN Set Enable. */
        uint8_t                         :1;
        chctrl_setsswprq_t  setsswprq   :1; /* Software Forced Ejection Request. */
        uint8_t                         :1;
        chctrl_setintm_t    setintm     :1; /* Interrupt Request Mask. */
        chctrl_clrintm_t    clrintm     :1; /* Interrupt Request Mask Clear. */
        chctrl_setdmarqm_t  setdmarqm   :1; /* DMA Activation Request Mask. */
        chctrl_clrdmarqm_t  clrdmarqm   :1; /* DMA Activation Request Mask Clear. */
        uint16_t                        :12;
    } BIT;
    unsigned long LONG;
} dma_chctrl_reg_t;
```

Figure 6.2 Structures and Unions

```
/* DMAC0/1 channel 0-15 */
typedef enum dma_unit_channel_s
{
    DMA_DMAC0_CH0           = (0),  /* Channel 0 of DMAC0. */
    DMA_DMAC0_CH1,                  /* Channel 1 of DMAC0. */
    DMA_DMAC0_CH2,                  /* Channel 2 of DMAC0. */
    DMA_DMAC0_CH3,                  /* Channel 3 of DMAC0. */
    DMA_DMAC0_CH4,                  /* Channel 4 of DMAC0. */
    DMA_DMAC0_CH5,                  /* Channel 5 of DMAC0. */
    DMA_DMAC0_CH6,                  /* Channel 6 of DMAC0. */
    DMA_DMAC0_CH7,                  /* Channel 7 of DMAC0. */
    DMA_DMAC0_CH8,                  /* Channel 8 of DMAC0. */
    DMA_DMAC0_CH9,                  /* Channel 9 of DMAC0. */
    DMA_DMAC0_CH10,                 /* Channel 10 of DMAC0. */
    DMA_DMAC0_CH11,                 /* Channel 11 of DMAC0. */
    DMA_DMAC0_CH12,                 /* Channel 12 of DMAC0. */
    DMA_DMAC0_CH13,                 /* Channel 13 of DMAC0. */
    DMA_DMAC0_CH14,                 /* Channel 14 of DMAC0. */
    DMA_DMAC0_CH15,                 /* Channel 15 of DMAC0. */
    DMA_DMAC1_CH0,                  /* Channel 0 of DMAC1. */
    DMA_DMAC1_CH1,                  /* Channel 1 of DMAC1. */
    DMA_DMAC1_CH2,                  /* Channel 2 of DMAC1. */
    DMA_DMAC1_CH3,                  /* Channel 3 of DMAC1. */
    DMA_DMAC1_CH4,                  /* Channel 4 of DMAC1. */
    DMA_DMAC1_CH5,                  /* Channel 5 of DMAC1. */
    DMA_DMAC1_CH6,                  /* Channel 6 of DMAC1. */
    DMA_DMAC1_CH7,                  /* Channel 7 of DMAC1. */
    DMA_DMAC1_CH8,                  /* Channel 8 of DMAC1. */
    DMA_DMAC1_CH9,                  /* Channel 9 of DMAC1. */
    DMA_DMAC1_CH10,                 /* Channel 10 of DMAC1. */
    DMA_DMAC1_CH11,                 /* Channel 11 of DMAC1. */
    DMA_DMAC1_CH12,                 /* Channel 12 of DMAC1. */
    DMA_DMAC1_CH13,                 /* Channel 13 of DMAC1. */
    DMA_DMAC1_CH14,                 /* Channel 14 of DMAC1. */
    DMA_DMAC1_CH15                  /* Channel 15 of DMAC1. */
} dma_unit_channel_t;

/* DMA API error codes */
typedef enum dma_err_e
{
    DMA_SUCCESS             = (0),  /* success */
    DMA_ERR_BAD_UNIT,               /* Invalid unit number. */
    DMA_ERR_BAD_CHAN,               /* Invalid channel number. */
    DMA_ERR_CH_NOT_OPENED,          /* Channel not yet opened. */
    DMA_ERR_CH_NOT_CLOSED,          /* Channel still open from previous open. */
    DMA_ERR_UNKNOWN_CMD,            /* Control command is not recognized. */
    DMA_ERR_INVALID_ARG,            /* Argument is not valid for parameter. */
    DMA_ERR_NULL_PTR,               /* Received null pointer; missing required argument. */
    DMA_ERR_BUFF_FULL               /* LINK mode buffer full. */
} dma_err_t;
```

```
/* DMA API command codes */
typedef enum dma_cmd_e
{
    DMA_CMD_LINK_SET        = (0),  /* setting of link descriptor. */
    DMA_CMD_SKIP_SET        = (1),  /* setting of skip parameter. */
    DMA_CMD_DSCITVL_SET      = (2),  /* setting of descriptor interval. */
} dma_cmd_t;

/* channel state */
typedef enum
{
    DMA_CH_STS_CLOSE        = (0),  /* channel close state. */
    DMA_CH_STS_OPEN         = (1)   /* channel open state. */
} dma_ch_sts_t;

/* skip mode Select */
typedef enum
{
    DMA_MOD_SKIP_INVALID    = (0),  /* skip mode invalid. */
    DMA_MOD_SKIP_VALID      = (1)   /* skip mode valid. */
} dma_mod_skip_t;

/* DMA Activation Enable (CHCTRL register) */
typedef enum
{
    CHCTRL_SETEN_NOP        = (0),  /* Operation is not affected. */
    CHCTRL_SETEN_EN         = (1)   /* DMA Activation Enable. */
} chctrl_seten_t;

/* DMA Activation Enable Clear (CHCTRL register) */
typedef enum
{
    CHCTRL_CLREN_NOP        = (0),  /* Operation is not affected. */
    CHCTRL_CLREN_STP        = (1)   /* DMA Activation Enable Clear. */
} chctrl_clren_t;

/* Software Reset (CHCTRL register) */
typedef enum
{
    CHCTRL_SWRST_NOP        = (0),  /* Operation is not affected. */
    CHCTRL_SWRST_RST        = (1)   /* DMA Software Reset. */
} chctrl_swrst_t;

/* DMA Transfer Request Clear (CHCTRL register) */
typedef enum
{
    CHCTRL_CLRRQ_NOP        = (0),  /* Operation is not affected. */
    CHCTRL_CLRRQ_CLR        = (1)   /* DMA Transfer Request Clear. */
} chctrl_clrrq_t;

/* END Clear (CHCTRL register) */
typedef enum
{
    CHCTRL_CLREND_NOP       = (0),  /* Operation is not affected. */
    CHCTRL_CLREND_CLR       = (1)   /* DMA END Clear. */
} chctrl_clrend_t;
```

```
/* DER Clear (CHCTRL register) */
typedef enum
{
    CHCTRL_CLRDER_NOP       = (0),  /* Operation is not affected. */
    CHCTRL_CLRDER_CLR       = (1)   /* DMA DER clear. */
} chctrl_clrder_t;

/* Suspend Request (CHCTRL register) */
typedef enum
{
    CHCTRL_SETSUS_NOP       = (0),  /* Operation is not affected. */
    CHCTRL_SETSUS_SUS       = (1)   /* DMA Suspend Request. */
} chctrl_setsus_t;

/* Suspend Clear (CHCTRL register) */
typedef enum
{
    CHCTRL_CLRSUS_NOP       = (0),  /* Operation is not affected. */
    CHCTRL_CLRSUS_SUS       = (1)   /* DMA Suspend Clear. */
} chctrl_clrsus_t;

/* REN Set Enable (CHCTRL register) */
typedef enum
{
    CHCTRL_SETREN_NOP       = (0),  /* Operation is not affected. */
    CHCTRL_SETREN_SET       = (1)   /* REN Set Enable. (Continue to run the DMA transferred.) */
} chctrl_setren_t;

/* Software Forced Ejection Request (CHCTRL register) */
typedef enum
{
    CHCTRL_SETSSWPRQ_NOP    = (0),  /* Operation is not affected. */
    CHCTRL_SETSSWPRQ_OUT    = (1)   /* Software Forced Ejection Request set. */
} chctrl_setsswprq_t;

/* Interrupt Request Mask (CHCTRL register) */
typedef enum
{
    CHCTRL_SETINTM_NOP      = (0),  /* Operation is not affected. */
    CHCTRL_SETINTM_MSK      = (1)   /* Interrupt Request Mask. */
} chctrl_setintm_t;

/* Interrupt Request Mask Clear (CHCTRL register) */
typedef enum
{
    CHCTRL_CLRINTM_NOP      = (0),  /* Operation is not affected. */
    CHCTRL_CLRINTM_CLR      = (1)   /* Interrupt Request Mask Clear. */
} chctrl_clrintm_t;

/* DMA Activation Request Mask (CHCTRL register) */
typedef enum
{
    CHCTRL_SETDMARQM_NOP    = (0),  /* Operation is not affected. */
    CHCTRL_SETDMARQM_MSK    = (1)   /* DMA Activation Request Mask. */
} chctrl_setdmarqm_t;
```

```
/* DMA Activation Request Mask Clear (CHCTRL register) */
typedef enum
{
    CHCTRL_CLRDMARQM_NOP    = (0),  /* Operation is not affected. */
    CHCTRL_CLRDMARQM_CLR    = (1)   /* DMA Activation Request Mask Clear. */
} chctrl_clrdmarqm_t;

/* Channel Select (CHCFG register) */
typedef enum
{
    CHCFG_SEL_CH0           = (0),  /* channel 0 select */
    CHCFG_SEL_CH1           = (1),  /* channel 1 select */
    CHCFG_SEL_CH2           = (2),  /* channel 2 select */
    CHCFG_SEL_CH3           = (3),  /* channel 3 select */
    CHCFG_SEL_CH4           = (4),  /* channel 4 select */
    CHCFG_SEL_CH5           = (5),  /* channel 5 select */
    CHCFG_SEL_CH6           = (6),  /* channel 6 select */
    CHCFG_SEL_CH7           = (7),  /* channel 7 select */
    CHCFG_SEL_CH8           = (0),  /* channel 8 select */
    CHCFG_SEL_CH9           = (1),  /* channel 9 select */
    CHCFG_SEL_CH10          = (2),  /* channel 10 select */
    CHCFG_SEL_CH11          = (3),  /* channel 11 select */
    CHCFG_SEL_CH12          = (4),  /* channel 12 select */
    CHCFG_SEL_CH13          = (5),  /* channel 13 select */
    CHCFG_SEL_CH14          = (6),  /* channel 14 select */
    CHCFG_SEL_CH15          = (7)   /* channel 15 select */
} chcfg_sel_t;

/* DMA Activation Request Source Select (CHCFG register) */
typedef enum
{
    CHCFG_REQD_SRC          = (0),  /* Requested by a transfer source module. */
    CHCFG_REQD_DES          = (1)   /* Requested by a transfer destination module. */
} chcfg_reqd_t;

/* Method of Detecting DMA Transfer Request Signals (CHCFG register) */
typedef enum
{
    CHCFG_REQMTD_INVALID    = (0),  /* detection invalid */
    CHCFG_REQMTD_EDG_FAL    = (1),  /* Detects the falling edge */
    CHCFG_REQMTD_EDG_RIS    = (2),  /* Detects the rising edge */
    CHCFG_REQMTD_LVL_LOW    = (5),  /* Detects the low level */
    CHCFG_REQMTD_LVL_HI     = (6)   /* Detects the high level */
} chcfg_reqmtd_t;

/* ACK Mode (CHCFG register) */
typedef enum
{
    CHCFG_AM_LVLMODE        = (1),  /* Level mode. */
    CHCFG_AM_BUSMODE        = (2),  /* Bus cycle mode. */
    CHCFG_AM_MSK            = (4)   /* Masks DACK/TEND output. */
} chcfg_am_t;
```

```
/* Descriptor Reload Enable (CHCFG register) */
typedef enum
{
    CHCFG_DRRP_END          = (0),  /* stops the operation. */
    CHCFG_DRRP_REP          = (1)   /* Continues reading the same descriptor. */
} chcfg_drrp_t;

/* Source Data Size (CHCFG register) */
typedef enum
{
    CHCFG_SDS_8BIT          = (0),  /* 8bit */
    CHCFG_SDS_16BIT         = (1),  /* 16bit */
    CHCFG_SDS_32BIT         = (2),  /* 32bit */
    CHCFG_SDS_128BIT        = (4),  /* 128bit */
    CHCFG_SDS_256BIT        = (5),  /* 256bit */
    CHCFG_SDS_512BIT        = (6),  /* 512bit */
    CHCFG_SDS_8BIT_SKP      = (8),  /* 8bit (skip mode) */
    CHCFG_SDS_16BIT_SKP     = (9),  /* 16bit (skip mode) */
    CHCFG_SDS_32BIT_SKP     = (10), /* 32bit (skip mode) */
    CHCFG_SDS_128BIT_SKP    = (12), /* 128bit (skip mode) */
    CHCFG_SDS_256BIT_SKP    = (13), /* 256bit (skip mode) */
    CHCFG_SDS_512BIT_SKP    = (14)  /* 512bit (skip mode) */
} chcfg_sds_t;

/* Destination Data Size (CHCFG register) */
typedef enum
{
    CHCFG_DDS_8BIT          = (0),  /* 8bit */
    CHCFG_DDS_16BIT         = (1),  /* 16bit */
    CHCFG_DDS_32BIT         = (2),  /* 32bit */
    CHCFG_DDS_128BIT        = (4),  /* 128bit */
    CHCFG_DDS_256BIT        = (5),  /* 256bit */
    CHCFG_DDS_512BIT        = (6),  /* 512bit */
    CHCFG_DDS_8BIT_SKP      = (8),  /* 8bit (skip mode) */
    CHCFG_DDS_16BIT_SKP     = (9),  /* 16bit (skip mode) */
    CHCFG_DDS_32BIT_SKP     = (10), /* 32bit (skip mode) */
    CHCFG_DDS_128BIT_SKP    = (12), /* 128bit (skip mode) */
    CHCFG_DDS_256BIT_SKP    = (13), /* 256bit (skip mode) */
    CHCFG_DDS_512BIT_SKP    = (14)  /* 512bit (skip mode) */
} chcfg_dds_t;

/* Source Address Count Direction (CHCFG register) */
typedef enum
{
    CHCFG_SAD_INC           = (0),  /* Increment. */
    CHCFG_SAD_FIX           = (1)   /* Fixed. */
} chcfg_sad_t;

/* Destination Address Count Direction (CHCFG register) */
typedef enum
{
    CHCFG_DAD_INC           = (0),  /* Increment. */
    CHCFG_DAD_FIX           = (1)   /* Fixed. */
} chcfg_dad_t;
```

```
/* Transfer Mode (CHCFG register) */
typedef enum
{
    CHCFG_TM_SIN            = (0),  /* Single transfer mode. */
    CHCFG_TM_BLK            = (1)   /* Block transfer mode. */
} chcfg_tm_t;

/* Write-Only Mode (CHCFG register) */
typedef enum
{
    CHCFG_WONLY_INVALID     = (0),  /* write-only mode invalid. */
    CHCFG_WONLY_VALID       = (1)   /* write-only mode valid. */
} chcfg_wonly_t;

/* Transfer Completion Interrupt Mask (CHCFG register) */
typedef enum
{
    CHCFG_DEM_NMSK          = (0),  /* Does not mask. */
    CHCFG_DEM_MSK           = (1)   /* Masks. */
} chcfg_dem_t;

/* Descriptor Interrupt Mask (CHCFG register) */
typedef enum
{
    CHCFG_DIM_NMSK          = (0),  /* Does not mask a DMA transfer completion interrupt. */
    CHCFG_DIM_MSK           = (1)   /* Masks a DMA transfer completion interrupt. */
} chcfg_dim_t;

/* Buffer Flush Enable (CHCFG register) */
typedef enum
{
    CHCFG_SBE_NOUT          = (0),  /* Stops transfer without flushing data in the buffer. */
    CHCFG_SBE_OUT           = (1)   /* Stops transfer after flushing data in the buffer. */
} chcfg_sbe_t;

/* Next Register Select (CHCFG register) */
typedef enum
{
    CHCFG_RSEL_R0           = (0),  /* Executes Next0 Register Set. */
    CHCFG_RSEL_R1           = (1)   /* Executes Next1 Register Set. */
} chcfg_rsel_t;
```

```
/* RSEL Reverse (CHCFG register) */
typedef enum
{
    CHCFG_RSW_NML           = (0),  /* Does not reverse the RSEL bit when DMA transfer completes. */
    CHCFG_RSW_REV           = (1)   /* Reverses the RSEL bit when DMA transfer completes. */
} chcfg_rsw_t;

/* Register Set Enable (CHCFG register) */
typedef enum
{
    CHCFG_REN_NEXE          = (0),  /* Does not perform DMA transfer accordingly. */
    CHCFG_REN_EXE           = (1)   /* Performs DMA transfer accordingly. */
} chcfg_ren_t;

/* DMA Mode Select (CHCFG register) */
typedef enum
{
    CHCFG_DMS_REG           = (0),  /* Register mode. */
    CHCFG_DMS_LNK           = (1)   /* Link mode. */
} chcfg_dms_t;

/* Priority Control Select (DCTRL register) */
typedef enum
{
    DCTRL_PR_FIX            = (0),  /* Fixed priority mode. */
    DCTRL_PR_RNDRBN         = (1)   /* Round-robin mode. */
} dctrl_pr_t;

/* DMA power setting value. */
typedef enum
{   /* Values will be uses as bit flags. */
    DMA_POWER_ON        = (0),      /* power on */
    DMA_POWER_OFF       = (1)       /* power off */
} dma_power_t;

/* interrupt enable/disable parameter. */
typedef enum
{
    DMA_ICU_INTR_DISABLE,           /* disable */
    DMA_ICU_INTR_ENABLE             /* enable */
} dma_icu_intr_para_t;

/* Status of the vector number search */
typedef enum
{
    DMA_SEARCH_CONTINUE,            /* Search continues. */
    DMA_SEARCH_EXIT                 /* Search exit. */
} dma_search_sts_t;
```

Figure 6.3 Enumerated Types

6.7 Global Variables

Table 6.8 lists global variables for the DMA sample program and driver.

Table 6.8 Global Variables

Type	Variable Name	Description	Function
volatile uint8_t	g_dma_cmp_dsc_cnt	Descriptor counter for DMA transfer completion	main dma_dmac0_soft_trg_isr
volatile uint8_t	g_dma_err_flag	DMA error flag	main dma_err_callback
static uint8_t	des_data0[256]	DMA transfer destination area 0	main
static uint8_t	des_data1[256]	DMA transfer destination area 1	main
static uint8_t	des_data2[256]	DMA transfer destination area 2	main
static const uint8_t	src_data0[256]	DMA transfer source area 0	main
static const uint8_t	src_data1[256]	DMA transfer source area 1	main
static const uint8_t	src_data2[256]	DMA transfer source area 2	main
static dma_handle_t	gb_dma_handles[2][16]	Handle information for all DMA channels (for 2 units and 16 channels)	R_DMA_Open R_DMA_Control R_DMA_Close

6.8 Functions

Table 6.9 lists the functions for the DMA sample program, and Table 6.10 lists the functions for the DMA sample driver.

Table 6.9 Functions for the DMA Sample Program

Function Name	Page Number
main	27
dma_err_callback	28
dma_dmac0_soft_trg_isr	29

Table 6.10 Functions for the DMA Sample Driver

Function Name	Page Number
R_DMA_Open	30
R_DMA_Control	33
R_DMA_Close	34
R_DMA_GetVersion	34
R_DMA_SoftTrgCommon	35

6.9 Specifications of Functions

This section presents the specifications of the functions for the sample program and sample driver.

6.9.1 main

main	
Synopsis	Main processing for the sample program
Header	–
Declaration	int main(void)
Description	This function uses the DMA functionality to copy the contents of the transfer source areas src_data0, src_data1, and src_data2 (256 bytes each) to the transfer destination areas des_data0, des_data1, and des_data2 (256 bytes each). Main processing: • Initializing variables and areas • Opening channel 0 of DMAC0 (R_DMA_Open) - Channel: Specifies channel 0 of DMAC0 - DMAC startup trigger: Specifies a DMAC0 software request - Defining DMA transfer completion interrupts • Interrupt handler definition • Vector number settings • Interrupt priority level settings - Registering a callback function for DMA error interrupts • Setting DMA link descriptors and starting transfer (R_DMA_Control) - Specifies the transfer source addresses (src_data0, src_data1, src_data2) - Specifies the transfer destination addresses (des_data0, des_data1, des_data2) - Specifies the number of bytes to transfer (DMA_DATA_LENGTH) - Specifies the transfer request source (CHCFG_REQD_SRC: Transfer source) - Specifies the transfer request detection method (CHCFG_REQMTD_EDG_RIS: Rising edge) - Specifies the ACK mode (CHCFG_AM_BUSMODE: Bus cycle mode) - Specifies the data size at the transfer source (CHCFG_SDS_8BIT: 8 bits) - Specifies the data size at the transfer destination (CHCFG_DDS_8BIT: 8 bits) - Specifies the count direction at the transfer source address (CHCFG_SAD_INC: Incremental) - Specifies the count direction at the transfer destination address (CHCFG_DAD_INC: Incremental) - Specifies the transfer mode (CHCFG_TM_BLK: Block transfer mode) Specifies the write-only mode settings (CHCFG_WONLY_INVALID: Write-only mode disabled) - Specifies the descriptor read interval (1) • Monitoring DMA transfer completion and DMA errors This function monitors DMA transfer completion and DMA error interrupts, and executes close processing when either a DMA transfer completion or DMA error interrupt occurs. • Closing channel 0 of DMAC0
Arguments	None
Return value	0

6.9.2 dma_err_callback

dma_err_callback	
Synopsis	Executing callbacks for DMA errors
Header	–
Declaration	static void dma_err_callback(void)
Description	This function is called from the DMA error handler. It sets an error occurrence flag for notifying the main processing of an error.
Arguments	None
Return values	None

6.9.3 dma_dmac0_soft_trg_isr

dma_dmac0_soft_trg_isr

Synopsis An interrupt handler triggered by a software request source for DMAC0

Header –

Declaration static void dma_dmac0_soft_trg_isr(void)

Description This function performs interrupt handler processing triggered by a software request source for DMAC0.

- Clearing the interrupt edge detection
- Counting the number of descriptor executions
- Terminating the interrupt sequence

Handler processing must be specified in the sample program, as shown in examples 1 and 2. (For details on each request source, see Table 6.12.)

Example 1: Handler triggered by a software request source (vector number = 251)

```
#ifdef __ICCARM__
#pragma type_attribute=__irq __arm
#endif  /* __ICCARM__ */
static void dma_dmac0_soft_trg_isr(void)
{
   /* Clear interrupt edge detection */
   VIC.PIC7.BIT.PIC251 = 1;
   R_DMA_SoftTrgCommon(ICU_VEC_NUM_251);

   g_dma_cmp_dsc_cnt++;

   /* End interrupt sequence */
   VIC.HVA0.LONG = DMA_HVA_WRITE_DATA;
}
```

Example 2: Handler triggered by a serial communication unit 2 source (vector number = 111)

```
#ifdef __ICCARM__
#pragma type_attribute=__irq __arm
#endif  /* __ICCARM__ */
static void dma_scif2_isr(void)
{
   /* Clear interrupt edge detection */
   VIC.PIC3.BIT.PIC111 = 1;

   g_dma_cmp_dsc_cnt++;

   /* End interrupt sequence */
   VIC.HVA0.LONG = DMA_HVA_WRITE_DATA;
}
```

Note: The numbers "251" and "111" in the coding must be changed according to the vector numbers for the interrupt sources.

Note: Call the R_DMA_SoftTrgCommon function only when the vector number for the interrupt source is 251 or 252.

Arguments None

Return values None

6.9.4 R_DMA_Open

R_DMA_Open	
Synopsis	Opening the DMA module
Header	r_dma_rzt1_if.h
Declaration	dma_err_t R_DMA_Open(const dma_unit_channel_t unit_channel, const dma_vector_para_t * const vector_para, void (* const pcallback_err)(void), dma_handle_t * const phandle)
Description	This function opens the DMA module for the specified DMA unit and channel (the first argument), sets the interrupt handler for DMA transfer completion (the second argument), and registers the callback function (the third argument) when a DMA error occurs. It then sets and returns information about the opened DMA module to the handle (the fourth argument). The execution result of the function is returned as a return value. Main processing: • Checking arguments - Checking the range of unit numbers (the first argument) - Checking for null pointers in the interrupt handler for DMA transfer completion (the second argument) - Checking the range of vector numbers (the second argument) - Checking for duplicate vector numbers (the second argument) - Checking for null pointers in the handle (the fourth argument) - Checking if the channel is open If the channel is already open, an error occurs. (The channel cannot open if it is already open. To reopen the channel, first close it with the R_DMA_Close function.) • Setting information - Setting the open state - Setting unit information - Setting channel information - Setting the vector number, interrupt priority level, and handler for DMA transfer completion interrupts - Setting DMA error callbacks - Setting the SKIP mode (disable the SKIP mode) • Releasing the stopped state of the DMA unit 0/1 module (turning OFF the power saving function) • Setting up the DMA functionality - Setting the CMNCR register - Setting the DCTRL_(unit) register (fixed priority mode) Fixed priority is used between channels 0 and 7, and between channels 8 and 15. Round robin is used between channels 0- 7 and channels 8-15. - Setting the NXLA_(ch) register (setting the starting address of the link descriptor area)

- Setting the CHCFG_(ch) register
  - Channel settings
  - Transfer request source (CHCFG_REQD_SRC: Transfer source)
  - Transfer request detection method (CHCFG_REQMTD_EDG_RIS: Rising edge)
  - ACK mode (CHCFG_AM_BUSMODE: Bus cycle mode)
  - Descriptor reread operation (CHCFG_DRRP_REP: Continue reading descriptors)
  - Data size at the transfer source (CHCFG_SDS_8BIT: 8 bits)
  - Data size at the transfer destination (CHCFG_DDS_8BIT: 8 bits)
  - Count direction at the transfer source address (CHCFG_SAD_INC: Incremental)
  - Count direction at the transfer destination address (CHCFG_DAD_INC: Incremental)
  - Transfer mode (CHCFG_TM_BLK: Block transfer mode)
  - Write-only mode (CHCFG_WONLY_INVALID: Disable write-only mode)
  - Transfer completion interrupt mask (CHCFG_DEM_NMSK: Do not mask)
  - Descriptor interrupt mask (CHCFG_DIM_NMSK: Do not mask)
  - Buffer flush settings (CHCFG_SBE_NOUT: Do not flush buffer)
  - Next register (CHCFG_RSEL_R0: Execute Next0 Register Set)
  - RSEL bit reversal
    (CHCFG_RSW_NML: Do not reverse the RSEL bit after DMA transfer is complete)
  - Register setting operation
    (CHCFG_REN_NEXE: Do not execute DMA transfer successively)
  - DMA mode (CHCFG_DMS_LNK: Link mode)
- CHITVL_(ch) register settings (DMA_CHITVL_INIT: 0x00000000)
- SCNT_(ch) register settings (DMA_SCNT_INIT: 0x00000000)
- SSKP_(ch) register settings (DMA_SSKP_INIT: 0x00000000)
- DCNT_(ch) register settings (DMA_DCNT_INIT: 0x00000000)
- DSKP_(ch) register settings (DMA_DSKP_INIT: 0x00000000)

- Initializing a loop-structured descriptor
  - Setting a descriptor header
    - Setting the STS bit to 0
    - Setting the D bit to 0 (no descriptor error)
    - Setting the LV bit to 0 (descriptor disabled)
    - Setting the LE bit to 0 (link continues)
    - Setting the WBD bit to 0 (set 0 back to the LV bit)
    - Setting the DSCFM bit to 1 (descriptor format 1)
  - Setting the next link address
    An address is set to the next link address of each loop-structured descriptor.
- Setting information for the handle (the fourth argument)
- Registering a vector number and interrupt priority level (the second argument) for the ICU and enabling interrupts
- Registering a DMA error interrupt for the ICU and enabling interrupts
- Starting DMA transfer
  - Resetting DMA software
  - Enabling DMA transfer
  - Setting the DMA(unit)SEL(ch) register
    Setting the DMA channel source

Note: (unit) = 0/1, (ch) = 0-15

Arguments	const dma_unit_channel_t unit_channel	Specifies a unit and channel. This argument specifies a DMA unit in the upper four bits and a channel in the lower four bits. (Specified in dma_unit_channel_t in Figure 6.3)
	const dma_vector_para_t * const vector_para	Specifies vector parameters. This argument specifies a vector number (see Table 6.12), interrupt priority level, and handler functions for DMA transfer completion interrupts.
	void (*pcallback_err)(void)	Specifies a DMA error callback. This argument specifies the callback function to be executed when a DMA error interrupt occurs. Specify NULL when no callback function is used.
	dma_handle_t * const phandle	Specifies a pointer to the DMA handle. This argument returns information about a DMA channel that is opened to the area specified by the phandle pointer. The area specified by the phandle pointer must be allocated by processing that calls R_DMA_Open.
Return values	DMA_SUCCESS	Successful
	DMA_ERR_BAD_UNIT	Failed - Invalid unit number
	DMA_ERR_BAD_CHAN	Failed - Invalid channel number
	DMA_ERR_CH_NOT_CLOSED	Failed - Cannot close
	DMA_ERR_INVALID_ARG	Failed - Invalid parameter settings
	DMA_ERR_NULL_PTR	Failed - Empty parameter (NULL)
Remarks	Multiple DMAC channels (the first argument) cannot be allocated to one interrupt source for DMA transfer completion (the vector number specified in vector_para in the second argument). A similar rule also applies to software requests. Only two interrupt sources are allowed for transfer completion interrupts by software requests at the same time, so up to two channels can be used at the same time. This API checks for duplicate uses and returns a DMA_ERR_INVALID_ARG error when a violation is detected.	

6.9.5 R_DMA_Control

R_DMA_Control	
Synopsis	Executing commands for the DMA module
Header	r_dma_rzt1_if.h
Declaration	dma_err_t R_DMA_Control(const dma_handle_t * const phandle, const uint8_t cmd, const void * const param)
Description	This function executes the specified command (the second argument) for the channel specified by the handle (the first argument) and sets the parameter (the third argument). The execution result of the function is returned as a return value. Main processing: • Checking arguments - Checking for null pointers in the specified handle (the first argument) - Checking for null pointers in the specified parameter (the third argument) - Checking if the channel is open If the channel is not open, an error occurs. (Before executing the R_DMA_Control function, open the channel with the R_DMA_Open function.) • Command processing The command specified in the second argument executes each processing. This function handles any invalid command as an unknown command and returns an error. For details on the commands, see Table 6.11.
Arguments	const dma_handle_t * const phandle — Specifies a pointer to the DMA handle. Specify a handle that has been opened by the R_DMA_Open function. const uint8_t cmd — Specifies the command to be executed. (For details, see Table 6.11.) void * const param — Specifies a pointer to the parameter. (For details, see Table 6.11 and Table 6.12.)
Return values	DMA_SUCCESS — Successful DMA_ERR_CH_NOT_OPENED — Failed - Channel not open DMA_ERR_UNKNOWN_CMD — Failed - Unknown command DMA_ERR_INVALID_ARG — Failed - Invalid parameter settings DMA_ERR_NULL_PTR — Failed - Empty parameter (NULL) DMA_ERR_BUFF_FULL — Failed - No free space in the descriptor setting buffer
Remarks	Multiple instances of this API cannot be executed at the same time. This API must be executed exclusively by individual users.

6.9.6 R_DMA_Close

R_DMA_Close		
Synopsis	Closing the DMA module	
Header	r_dma_rzt1_if.h	
Declaration	dma_err_t R_DMA_Close(const dma_handle_t * const phandle)	
Description	This function closes the channel specified by the handle (the first argument). The execution result of the function is returned as a return value. Main processing: • Checking arguments - Checking for null pointers in the specified handle (the first argument) - Checking if the channel is open If the channel is not open, an error occurs. (This function can close only the channels opened by the R_DMA_Open function.) • Causing the ICU to prohibit interrupts to the specified channel (the first argument) • Releasing the open state • Changing the DMA unit 0/1 module to a stopped state (turning ON the power saving function) The power saving function turns ON only when all the channels in the unit are closed. The power saving function does not turn ON even if only one of the channels in the unit is open.	
Arguments	const dma_handle_t * const phandle	Specifies a pointer to the DMA handle. Specify a handle that has been opened by the R_DMA_Open function.
Return values	DMA_SUCCESS DMA_ERR_CH_NOT_OPENED DMA_ERR_NULL_PTR	Successful Failed - Channel not open Failed - Parameter not set (NULL)

6.9.7 R_DMA_GetVersion

R_DMA_GetVersion	
Synopsis	Obtaining version information for the DMA module
Header	r_dma_rzt1_if.h
Declaration	uint32_t R_DMA_GetVersion(void)
Description	This function returns version information for the DMA module as a return value.
Arguments	None
Return values	Version information for the DMA sample driver (32 bits) 16-31bit: Major Version 0-15bit: Minor Version

6.9.8 R_DMA_SoftTrgCommon

R_DMA_SoftTrgCommon		
Synopsis	Issuing a software request source	
Header	r_dma_rzt1_if.h	
Declaration	void R_DMA_SoftTrgCommon(const uint16_t vector_no)	
Description	If there is data to be transferred, this function issues a DMA transfer request by making a call upon the completion of DMA transfer by a software request source (vector number 251 or 252). Note: Note: Make a call from the interrupt handler triggered by the software request source specified in the sample program.	
Arguments	const uint16_t vector_no	Vector number for an interrupt source Specify either ICU_VEC_NUM_251 or ICU_VEC_NUM_252.
Return values	None	

6.10 Flowcharts

6.10.1 main Processing

Figure 6.4 shows the flowchart for main processing.

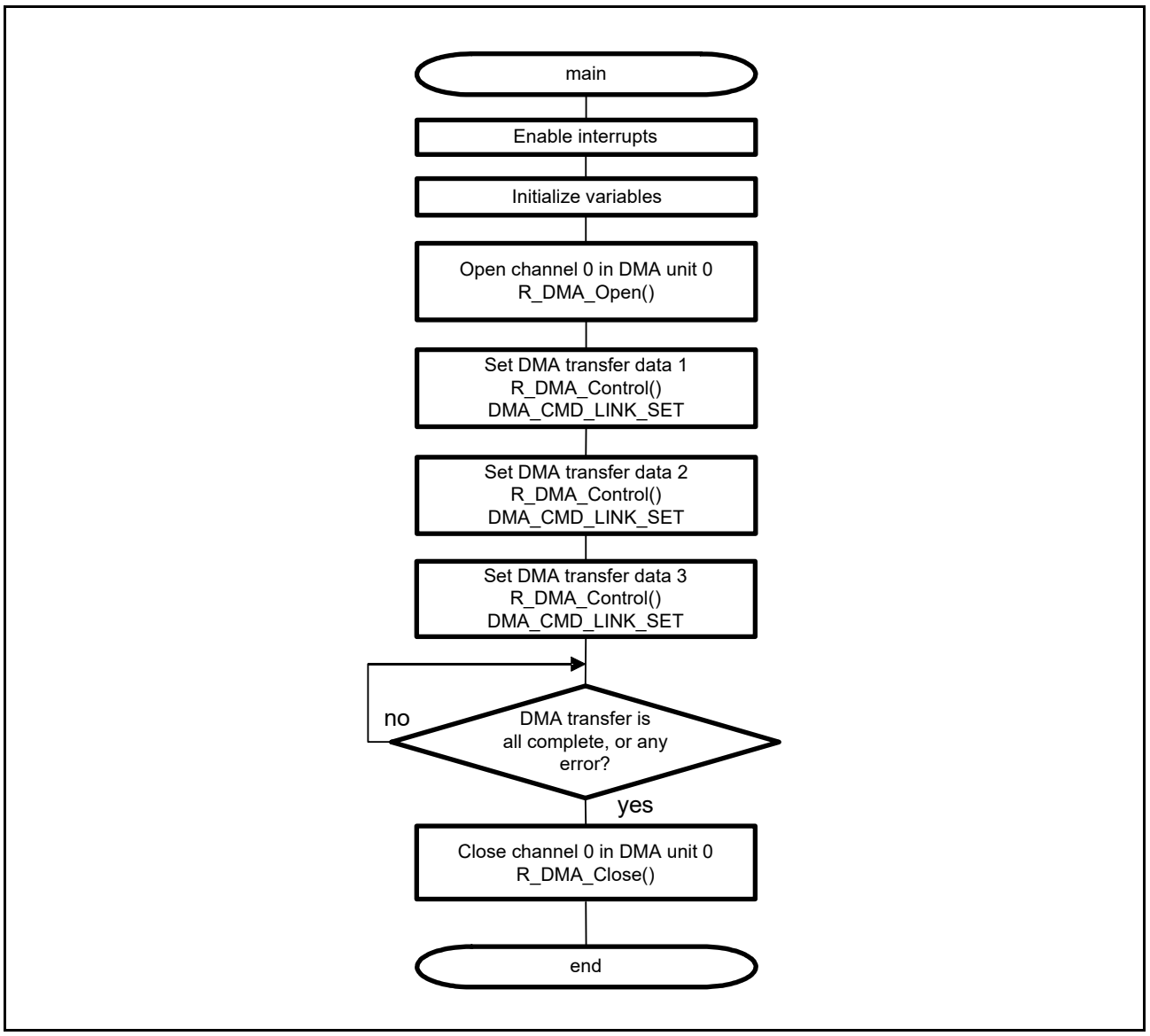


Figure 6.4 main Processing

6.10.2 dma_err_callback Processing

Figure 6.5 shows the flowchart for dma_err_callback processing.

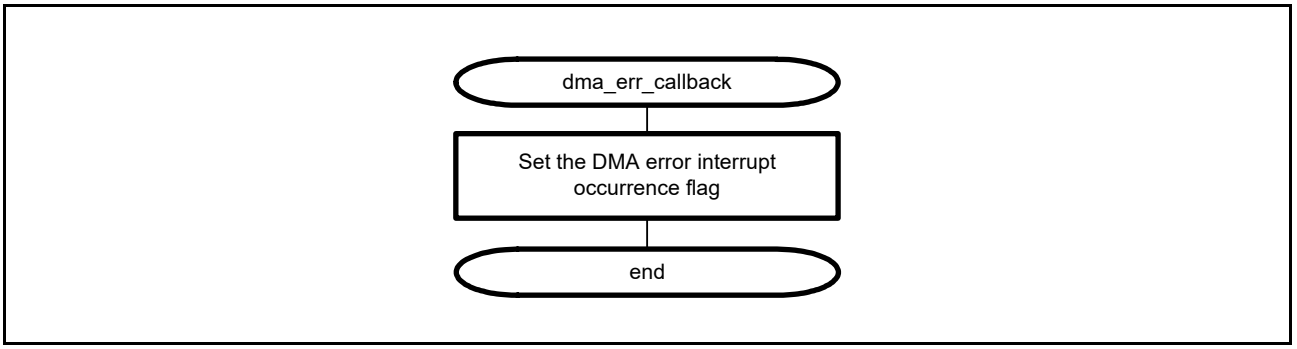


Figure 6.5 dma_err_callback Processing

6.10.3 dma_dmac0_soft_trg_isr Processing

Figure 6.6 shows the flowchart for dma_dmac0_soft_trg_isr processing.

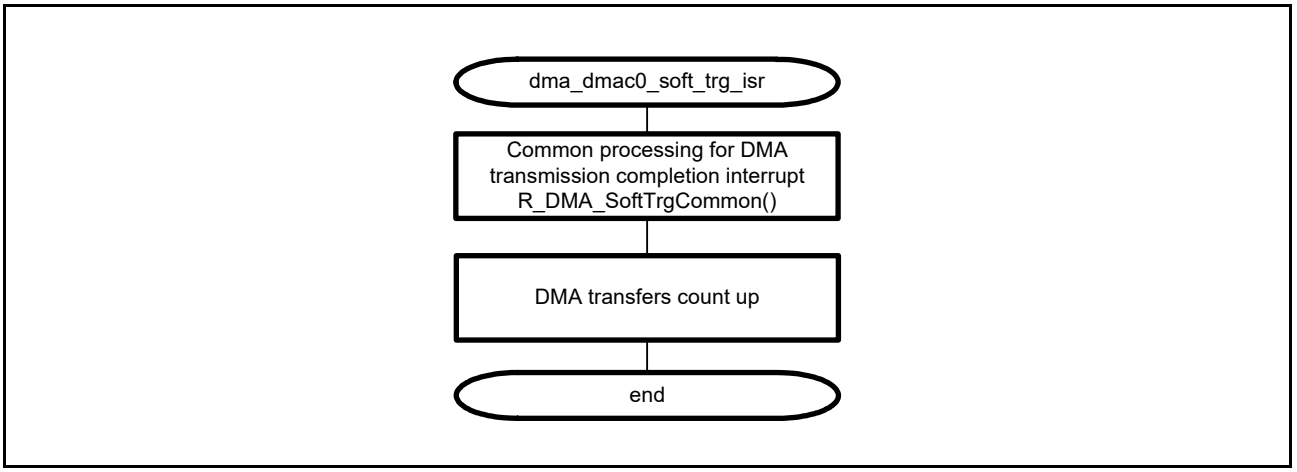


Figure 6.6 dma_dmac0_soft_trg_isr Processing

6.10.4 R_DMA_Open Processing

Figure 6.7 shows the flowchart for R_DMA_Open processing.

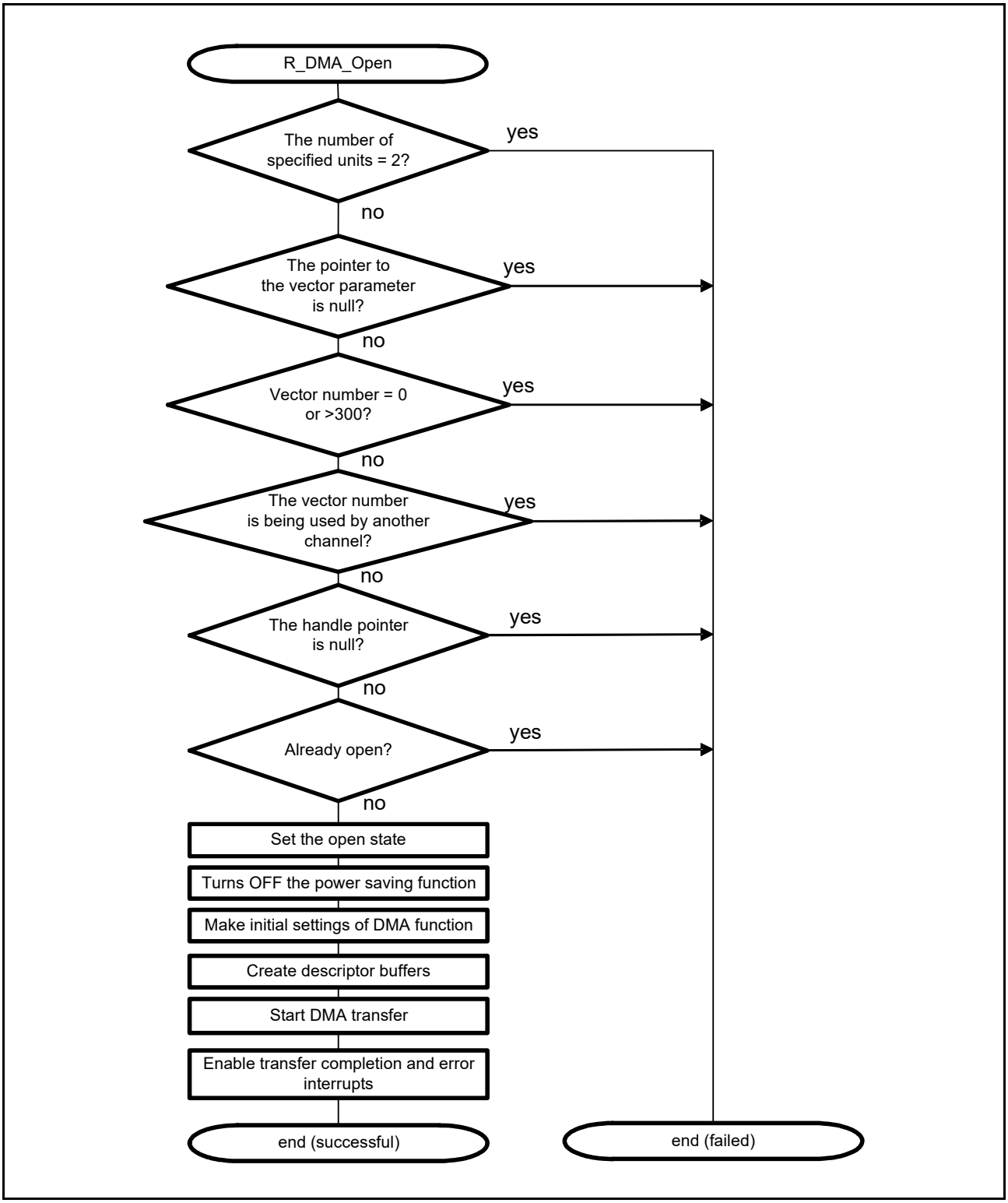


Figure 6.7 R_DMA_Open Processing

6.10.5 R_DMA_Control Processing

Figure 6.8, Figure 6.9, and Figure 6.10 show the flowcharts for R_DMA_Control processing.

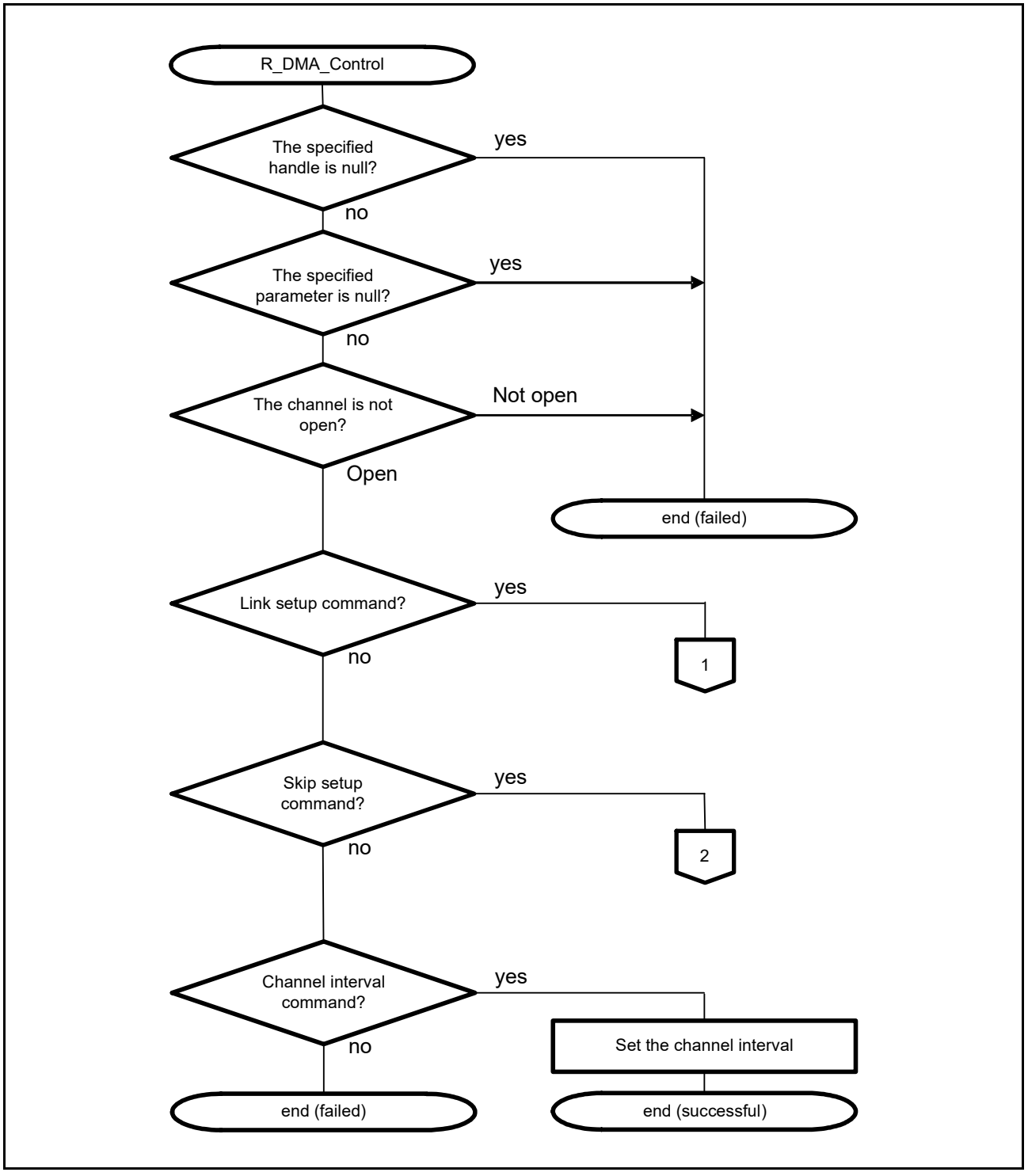


Figure 6.8 R_DMA_Control Processing

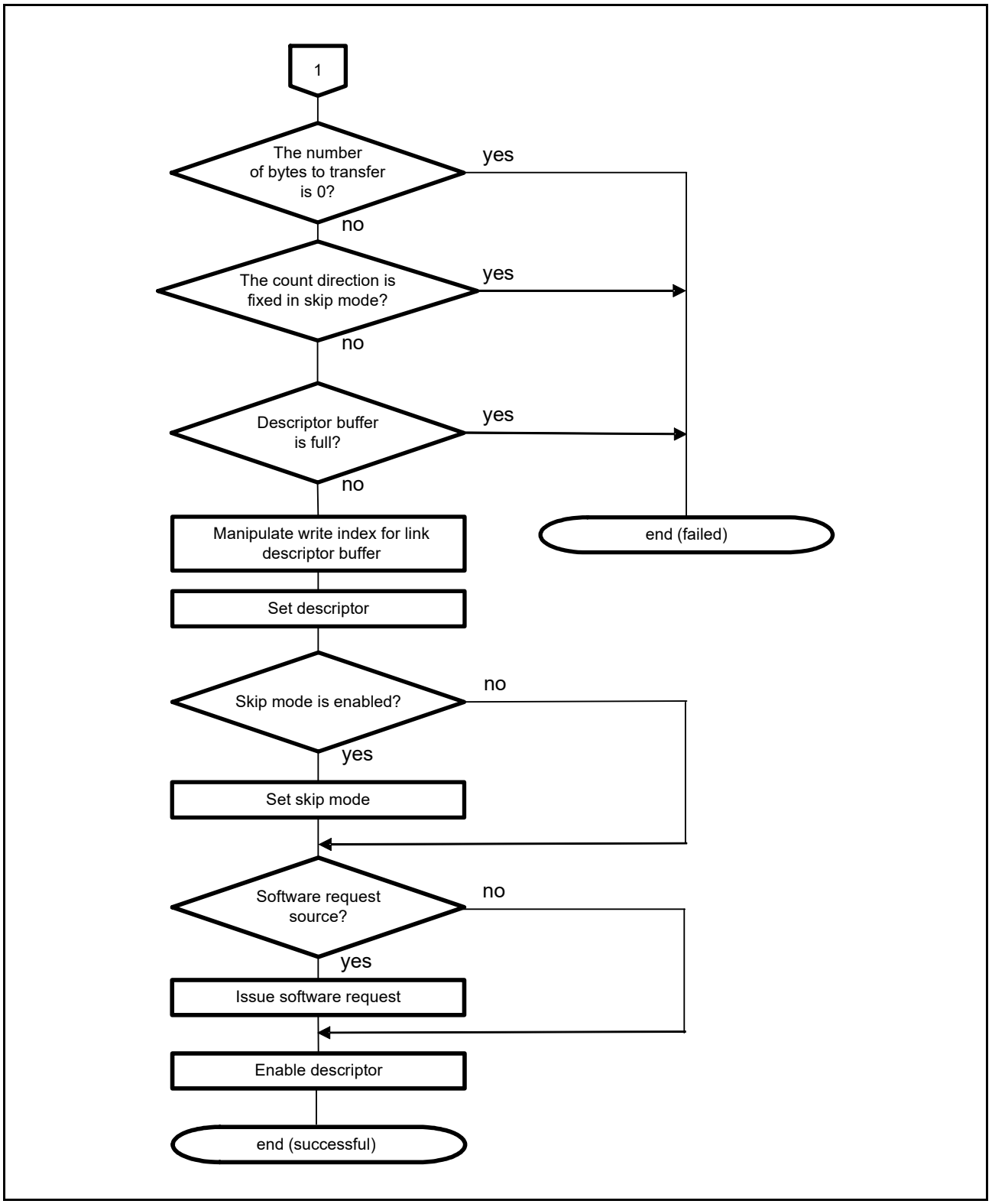


Figure 6.9 R_DMA_Control Processing

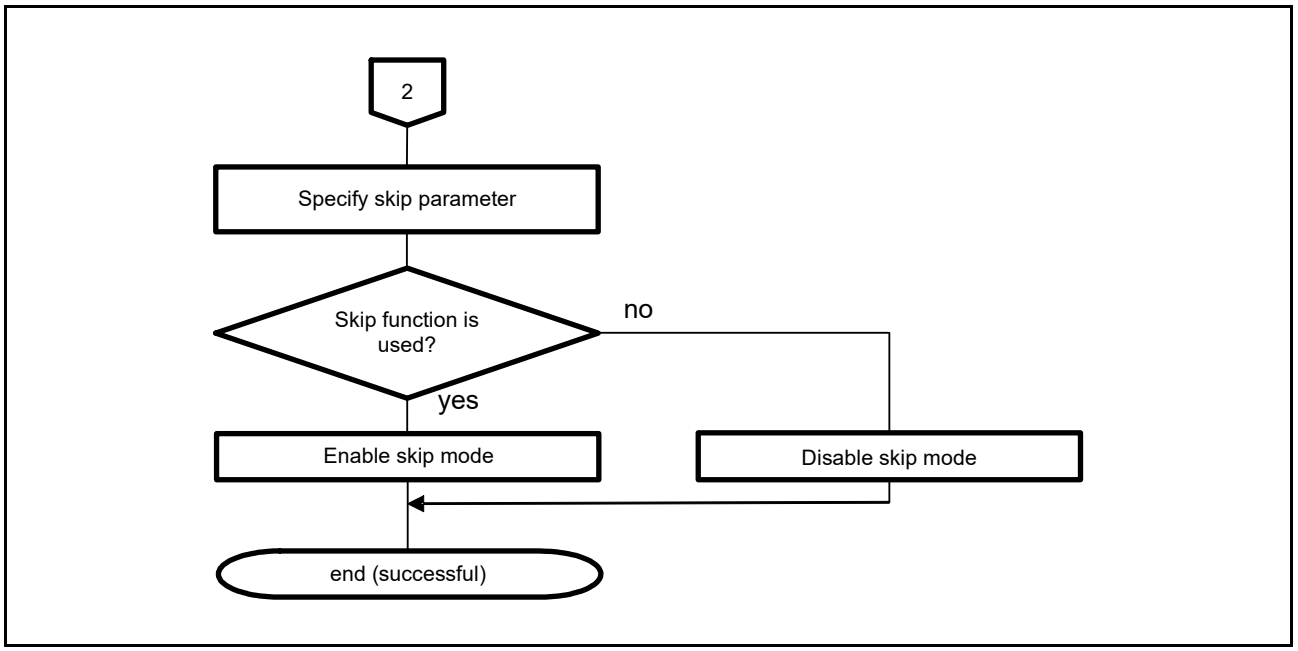


Figure 6.10 R_DMA_Control Processing

6.10.6 R_DMA_Close Processing

Figure 6.11 shows the flowchart for R_DMA_Close processing.

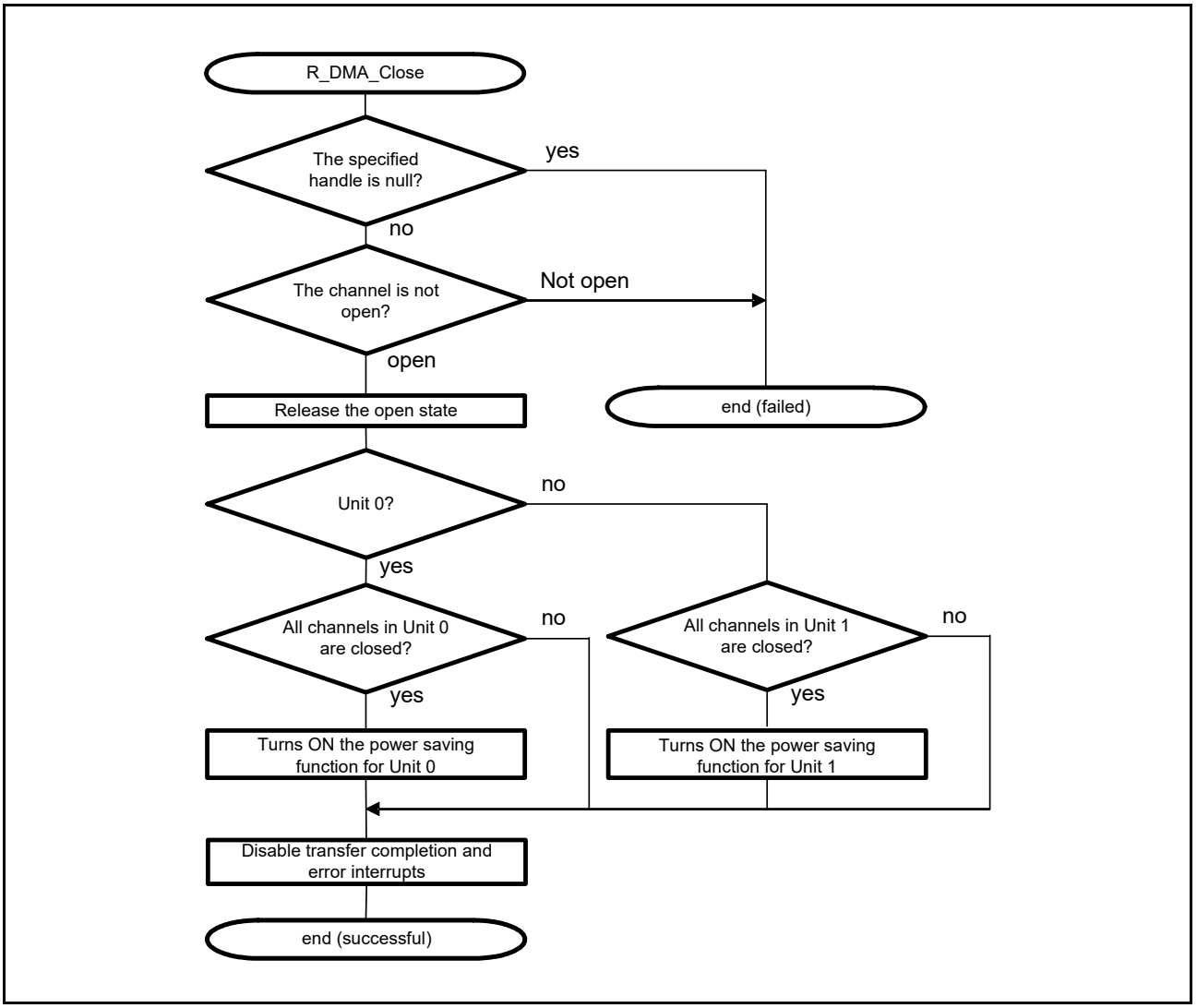


Figure 6.11 R_DMA_Close Processing

6.10.7 R_DMA_GetVersion Processing

Figure 6.12 shows the flowchart for R_DMA_GetVersion processing.

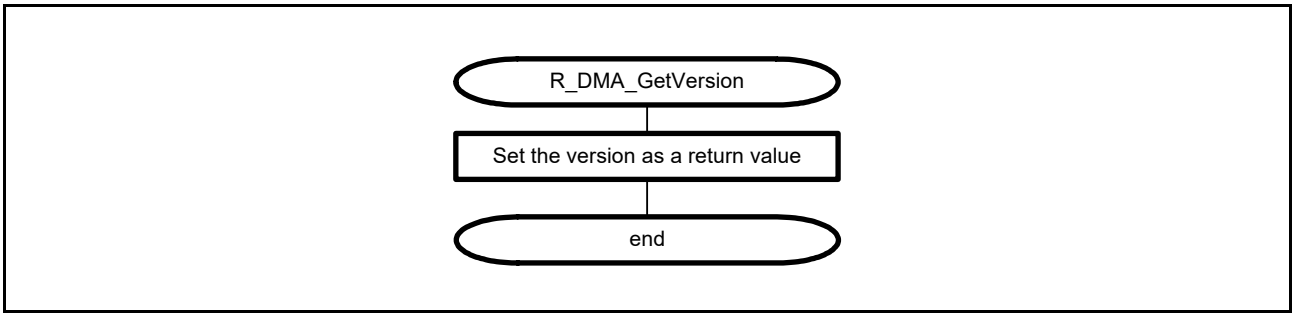


Figure 6.12 DMA_GetVersion Processing

6.10.8 R_DMA_SoftTrgCommon Processing

Figure 6.13 shows the flowchart for R_DMA_SoftTrgCommon processing.

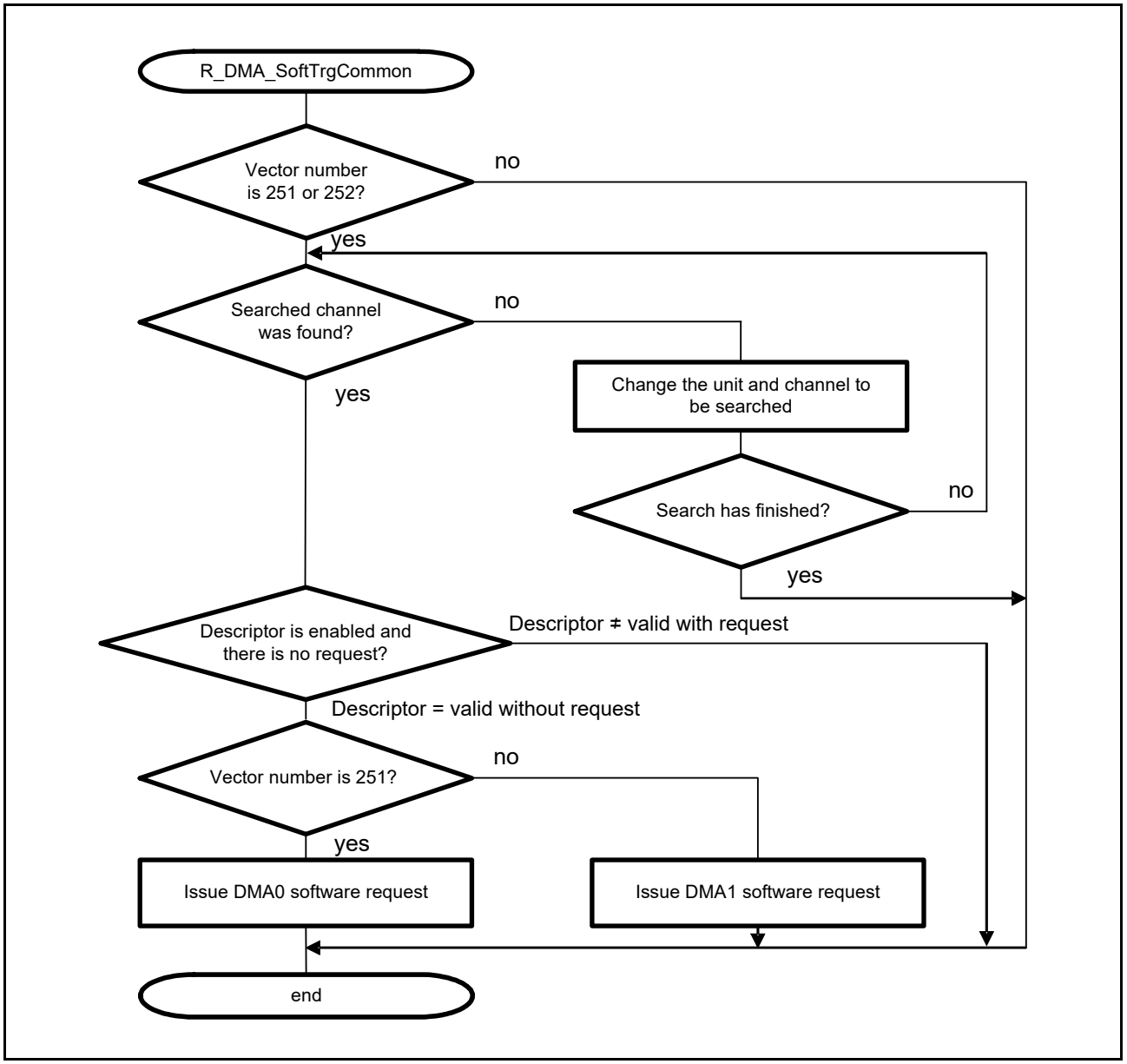


Figure 6.13 R_DMA_SoftTrgCommon Processing

6.11 R_DMA_Control Commands

The following table lists the commands for the R_DMA_Control function.

Table 6.11 R_DMA_Control Commands

Command Name	Description
DMA_CMD_LINK_SET	Sets the link descriptor (transfer parameters) and starts DMA transfer.
DMA_CMD_SKIP_SET	Sets parameters for the skip function.
DMA_CMD_DSCITVL_SET	Sets parameters for descriptor read intervals.

6.11.1 DMA_CMD_LINK_SET

DMA_CMD_LINK_SET		
Synopsis	Setting the link descriptor and starting DMA transfer	
Header	r_dma_rzt1_if.h	
Description	This command sets the link descriptor specified in the third argument for the unit and channel specified by the handle in the first argument, and starts DMA transfer.	
Parameters	dma_control_link_para_t	Parameter structure of the third argument.
	uint32_t sa	Specifies the transfer source address. Note: Note: If write-only mode (wonly= CHCFG_WONLY_VALID) is specified, specify here the data pattern to be transferred. When the data size is 16 or 8 bits, 16 or 8 bits from the right edge are effective.
	uint32_t da	Specifies the transfer destination address.
	uint32_t tb	Specifies the number of bytes to transfer. Note: Note: Do not specify 0. (An error will return.)
	chcfg_reqd_t reqd	Specifies the transfer request source. : CHCFG_REQD_SRC: Specifies the transfer source. : CHCFG_REQD_DES: Specifies the transfer destination.
	chcfg_reqmtd_t reqmtd	Specifies the transfer request detection method. : CHCFG_REQMTD_INVALID: Detection disabled : CHCFG_REQMTD_EDG_FAL: Falling edge : CHCFG_REQMTD_EDG_RIS: Rising edge : CHCFG_REQMTD_LVL_LOW: Low level : CHCFG_REQMTD_LVL_HI: High level Note: In some cases, settings are predetermined for each transfer source. (See Table 6.12.)
	chcfg_am_t am	Specifies the ACK mode. : CHCFG_AM_LVLMODE: Level mode : CHCFG_AM_BUSMODE: Bus cycle mode : CHCFG_AM_MUSK: Mask ACK Note: Settings are predetermined for each transfer source. (See Table 6.12.)
	chcfg_sds_t sds	Specifies the data size at the transfer source. : CHCFG_SDS_8BIT: 8 bits : CHCFG_SDS_16BIT: 16 bits : CHCFG_SDS_32BIT: 32 bits : CHCFG_SDS_128BIT: 128 bits : CHCFG_SDS_256BIT: 256 bits : CHCFG_SDS_512BIT: 512 bits
	chcfg_dds_t dds	Specifies the data size at the transfer destination. : CHCFG_DDS_8BIT: 8 bits : CHCFG_DDS_16BIT: 16 bits : CHCFG_DDS_32BIT: 32 bits : CHCFG_DDS_128BIT: 128 bits : CHCFG_DDS_256BIT: 256 bits : CHCFG_DDS_512BIT: 512 bits

	chcfg_sad_t sad	Specifies the count direction at the transfer source address. : CHCFG_SAD_INC: Incremental : CHCFG_SAD_FIX: Fixed Note: If the SKIP mode is used, do not specify CHCFG_SAD_FIX. (An error will return.) The DMA_CMD_SKIP_SET command can be used to select whether to use the SKIP mode.
	chcfg_dad_t dad	Specifies the count direction at the transfer destination address. : CHCFG_DAD_INC: Incremental : CHCFG_DAD_FIX: Fixed Note: If the SKIP mode is used, do not specify CHCFG_DAD_FIX. (An error will return.) The DMA_CMD_SKIP_SET command can be used to select whether to use the SKIP mode.
	chcfg_tm_t tm	Specifies the transfer mode. : CHCFG_TM_SIN: Single transfer mode : CHCFG_TM_BLK: Block transfer mode
	chcfg_wonly_t wonly	Specifies the write-only mode. : CHCFG_WONLY_INVALID: Disable write-only mode : CHCFG_WONLY_VALID: Enable write-only mode Note: If write-only mode is enabled, specify in "sa" of this parameter structure the data pattern to be transferred.
	uint16_t chitvl	Sets the DMA transfer interval. This parameter releases the bus by inserting an interval after every transfer from the request source specified by reqd at the interval of the specified value x 6.67 ns. Note: Specifiable values are from 0 to 0xFFFF. Note: 6.67ns ≈ 1/ICLK (ICLK=150MHz)
Return values	DMA_SUCCESS	: Successful
	DMA_ERR_NULL_PTR	: Failed - Parameter not specified
	DMA_ERR_CH_NOT_OPENED	: Failed - Channel not open
	DMA_ERR_BUFF_FULL	: Failed - No free space in the descriptor setting buffer
	DMA_ERR_INVALID_ARG	: Failed - Invalid parameter settings

Remarks
This command allocates a link descriptor area in the link buffer area internally allocated by the sample driver each time the command is executed.
The allocated descriptor area is not released until the descriptor has finished being executed. Therefore, if the link buffer area is full of unfinished descriptors and no more new descriptor areas can be allocated, a DMA_ERR_BUFF_FULL error will return.
Change the size of the link buffer area (in DMA_LINK_BUF_NUM) according to the operating conditions.
Increasing the value of DMA_LINK_BUF_NUM by one consumes 1,024 bytes of internal RAM. (Descriptor size 32 bytes x 16 channels x 2 units)

6.11.2 DMA_CMD_SKIP_SET

DMA_CMD_SKIP_SET		
Synopsis	Setting parameters for the DMA transfer skip function	
Header	r_dma_rzt1_if.h	
Description	This command sets the skip function parameters specified in the third argument for the unit and channel specified by the handle in the first argument. This command transfers the size of address space to be continuously transferred that is specified in scnt or dcnt, and then skips the size of address space to be skipped that is specified in sskp or dskp before transfer. If a value other than 0 is specified in each parameter of this command, the skip function will be enabled in the DMA_CMD_LINK_SET command later. If 0 is specified in all the parameters, the skip function will be disabled in the DMA_CMD_LINK_SET command later.	
Parameters	dma_control_skip_para_t	Parameter structure of the third argument
	uint32_t scnt	Specifies the size of address space to be continuously transferred from the transfer source address.
	uint32_t sskp	Specifies the size of address space to be skipped at the transfer source address.
	uint32_t dcnt	Specifies the size of address space to be continuously transferred to the transfer destination address.
	uint32_t dskp	Specifies the size of address space to be skipped at the transfer destination address.
Return values	None	
Remarks	–	

6.11.3 DMA_CMD_DSCITVL_SET

DMA_CMD_DSCITVL_SET		
Synopsis	Setting the descriptor read interval	
Header	r_dma_rzt1_if.h	
Description	This command sets the descriptor read interval specified in the third argument for the unit and channel specified by the handle in the first argument. The channels are divided into four groups that share a setting of the read interval among each. The interval set for a channel by this command applies to all channels within the same group. • Unit 0, channel 0 to 7 • Unit 0, channel 8 to 15 • Unit 1, channel 0 to 7 • Unit 1, channel 8 to 15	
Parameters	dma_control_dscitvl_para_t	Parameter structure of the third argument
	uint8_t dscitvl	Specifies the transfer interval for DMA. The transfer interval is the specified Value x 256 x 6.67ns. Note: Specifiable values are from 0 to 0xFF. Note: 6.67ns ≈ 1/ICLK (ICLK=150MHz)
Return values	None	
Remarks	–	

Table 6.12 Detection Behavior Specification for DMA Transfer Requests (1 / 5)

Transfer Request Source	Request Source	Vector Number	am Setting Value	reqmtd Setting Value
External request	DREQ0	248	CHCFG_AM_LVLMODE: Level mode CHCFG_AM_BUSMODE: Bus cycle mode CHCFG_AM_MUSK: DACK/TEND output mask Note: Specify settings according to the specifications for the DMA transfer request source.	CHCFG_REQMTD_EDG_FAL: Falling edge detected CHCFG_REQMTD_EDG_RIS: Rising edge detected CHCFG_REQMTD_LVL_LOW: Low level detected CHCFG_REQMTD_LVL_HI: High level detected
	DREQ1	249		
	DREQ2	250		
External interrupt	IRQ0	4	CHCFG_AM_BUSMODE: Bus cycle mode	CHCFG_REQMTD_EDG_RIS: Rising edge detected CHCFG_REQMTD_LVL_HI: High level detected Note: Here, only the level or edge is specified. For the level, specify CHCFG_REQMTD_EDG_RIS, and for the edge, specify CHCFG_REQMTD_LVL_HI. The high or low level is actually determined by the IRQCR(ch=1 to 15) register.
	IRQ1	5		
	IRQ2	6		
	IRQ3	7		
	IRQ4	8		
	IRQ5	9		
	IRQ6	10		
	IRQ7	11		
	IRQ8	12		
	IRQ9	13		
	IRQ10	14		
	IRQ11	15		
	IRQ12	16		
	IRQ13	17		
	IRQ14	18		
	IRQ15	19		
CMT Unit0	Compare-Match 0	21		CHCFG_REQMTD_EDG_RIS: Rising edge detected
	Compare-Match 1	22		
CMT Unit1	Compare-Match 0	23		
	Compare-Match 1	24		
CMTW Unit0	Compare-Match	25		
	Input Capture 0	26		
	Input Capture 1	27		
	Output Capture 0	28		
	Output Capture 1	29		
CMTW Unit1	Compare-Match	30		
	Input Capture 0	31		
	Input Capture 1	32		
	Output Capture 0	33		
	Output Capture 1	34		
S12ADC Unit0	AD conversion complete	35		
	Group B conversion complete	36		

Table 6.12 Detection Behavior Specification for DMA Transfer Requests (2 / 5)

Transfer Request Source	Request Source	Vector Number	am Setting Value	reqmtd Setting Value
S12ADC Unit1	AD conversion complete	38	CHCFG_AM_BUSMODE: Bus cycle mode	CHCFG_REQMTD_EDG_RIS: Rising edge detected
	Group B conversion complete	39		
DMAC0	DMAC0 software trigger	251		
DMAC1	DMAC1 software trigger	252		
USB	FuncDMA request 1	43		
	FuncDMA request 1	44		
Ether Switch with IEEE1588	Ether SWITCH interrupt	45		
	Ether SWITCH DLR interrupt	46		
	Ether SWITCH SYNCOUT interrupt	47		
Ether PHY	Ether PHY interrupt 0	48		
	Ether PHY interrupt 1	49		
	Ether PHY interrupt 2	50		
Ether MAC	Ether MAC DMA reception complete	51		
	Ether MAC DMA transmission complete	52		
	Normal interrupt for received frame	53		
MTU3a	TGI7A	59		
	TGI7B	60		
	TGI7C	61		
	TGI7D	62		
	TCIV7	63		
Ether MAC	Access completion interrupt for Ethernet MII Management	64		
	Packet transmission completion interrupt for Ethernet Pause	65		
	Ethernet transmission completion interrupt	66		
Ether CAT slave (only for products equipped with R-IN Engine)	Sync0 interrupt	73		
	Sync1 interrupt	74		
	Ether CAT interrupt	75		
	SOF interrupt	76		
	EOF interrupt	77		

Table 6.12 Detection Behavior Specification for DMA Transfer Requests (3 / 5)

Transfer Request Source	Request Source	Vector Number	am Setting Value	reqmtd Setting Value
RSPI Unit0	Reception buffer full	80	CHCFG_AM_BUSMODE: Bus cycle mode	CHCFG_REQMTD_EDG_RIS: Rising edge detected
	Transmission buffer empty	81		
RSPI Unit1	Reception buffer full	84		
	Transmission buffer empty	85		
RSPI Unit2	Reception buffer full	88		
	Transmission buffer empty	89		
RSPI Unit3	Reception buffer full	92		
	Transmission buffer empty	93		
SCIFA Unit0	Reception buffer full	97		CHCFG_REQMTD_LVL_HI: High level detected
	Transmission buffer empty	98		
SCIFA Unit1	Reception buffer full	101		
	Transmission buffer empty	102		
SCIFA Unit2	Reception buffer full	110		
	Transmission buffer empty	111		
SCIFA Unit3	Reception buffer full	114		
	Transmission buffer empty	115		
SCIFA Unit4	Reception buffer full	118		
	Transmission buffer empty	119		
RIIC Unit0	Data reception complete	122		CHCFG_REQMTD_EDG_RIS: Rising edge detected
	Transmission data empty	123		
RIIC Unit1	Data reception complete	125		
	Transmission data empty	126		
MTU3a	TGIA0	145		
	TGIB0	146		
	TGIC0	147		
	TGID0	148		
	TGIA1	152		
	TGIB1	153		
	TGIA2	156		
	TGIB2	157		
	TGIA3	160		
	TGIB3	161		
	TGIC3	162		
	TGID3	163		
	TGIA4	165		
	TGIB4	166		

Table 6.12 Detection Behavior Specification for DMA Transfer Requests (4 / 5)

Transfer Request Source	Request Source	Vector Number	am Setting Value	reqmtd Setting Value
MTU3a	TGIC4	167	CHCFG_AM_BUSMODE: Bus cycle mode	CHCFG_REQMTD_EDG_RIS: Rising edge detected
	TGID4	168		
	TGIV4	169		
	TGIU5	170		
	TGIV5	171		
	TGIW5	172		
	TGIA8	173		
	TGIB8	174		
	TGIC8	175		
	TGID8	176		
GPT	GTCIA0	178		
	GTCIB0	179		
	GTCIC0	180		
	GTCID0	181		
	GTCIE0	182		
	GTCIF0	183		
	GDTE0	184		
	CTCIV0	185		
	GTCIU0	186		
	GTCIA1	187		
	GTCIB1	188		
	GTCIC1	189		
	GTCID1	190		
	GTCIE1	191		
	GTCIF1	192		
	CDTE1	193		
	GTCIV1	194		
	GTCIU1	195		
	GTCIA2	196		
	GTCIB2	197		
	GTCIC2	198		
	GTCID2	199		
	GTCIE2	200		
	GTCIF2	201		
	CDTE2	202		
	GTCIV2	203		
	GTCIU2	204		
	GTCIA3	205		
	GTCIB3	206		
	GTCIC3	207		
	GTCID3	208		
	GTCIE3	209		
	GTCIF3	210		
	CDTE3	211		
	GTCIV3	212		

Table 6.12 Detection Behavior Specification for DMA Transfer Requests (5 / 5)

Transfer Request Source	Request Source	Vector Number	am Setting Value	reqmtd Setting Value
GPT	GTCIU3	213	CHCFG_AM_BUSMODE: Bus cycle mode	CHCFG_REQMTD_EDG_RIS: Rising edge detected
	ETGIN	214		
	ETGIP	215		
TPUa Unit0	TGI0A	216		
	TGI0B	217		
	TGI1A	221		
	TGI1B	222		
	TGI2A	225		
	TGI2B	226		
	TGI3A	229		
	TGI3B	230		
	TGI4A	234		
	TGI4B	235		
	TGI5A	238		
	TGI5B	239		
ELC	Interrupt request 0	242		
	Interrupt request 1	243		

7. Sample Code

Download the sample code from the Renesas Electronics website.

8. Related Documents

- User's Manuals: Hardware
  RZ/T1 Group User's Manual: Hardware
  (Download the latest edition from the Renesas Electronics website.)

  RZ/T1 Evaluation Board RTK7910022C00000BR User's Manual
  (Download the latest edition from the Renesas Electronics website.)

- Technical Update and Technical News
  (Download the latest information from the Renesas Electronics website.)

- User's Manuals: Development Environment
  For the IAR Embedded Workbench® for Arm, download the user's manual from the IAR website.
  (Download the latest edition from the IAR website.)

Website and Support

Renesas Electronics website

http://www.renesas.com/

Inquiries

http://www.renesas.com/inquiry

Revision History	Application Note: DMA Sample Program

Rev.	Date	Description	
		Page	Summary
0.10	Apr. 02, 2015	—	First Edition issued
1.00	Apr. 10, 2015	—	Only the revision number was changed to be posted on a website.
1.10	Aug. 18, 2015	2. Operating Environment	
		5	Table 2.1 Operating Environment: Description added to Integrated Development Environment
		6. Software	
		10	6.2.4 Required Memory Size: Description and reference added
		10	Table 6.2: Table title and size description were partially amended
		10	Table 6.2 Memory Requirements: Description on the Note and Size, changed
		11	Table 6.3 added
		11	Table 6.4 added
		—	The following changes were made in order to correct the failure where settings of parameters for descriptor interval and channel interval were swapped.
		13	Table 6.7 Constants for the DMA Sample Program Macro DMA_CHITVL_VAL added Value of DMA_DSCITVL_VAL changed (1-> 0)
		16 - 18	Figure 6.2 Structures/Unions Member of dma_link_fmt_t changed from dscitvl to chitvl Member of dma_control_link_para_t changed from dscitvl to chitvl dma_control_chitvl_para_t changed to dma_control_dscitvl_para_t, member changed from chitvl to dscitvl
		21	Figure 6.3 Enumerated Types Member of dma_cmd_t changed from DMA_CMD_CHITVL_SET to DMA_CMD_DSCITVL_SET
		46	Table 6.14 R_DMA_Control Commands DMA_CMD_CHITVL_SET changed to DMA_CMD_DSCITVL_SET
		48	6.11.1 DMA_CMD_LINK_SET Parameter changed from dscitvl to chitvl and relative descriptions changed
		50	6.11.3 DMA_CMD_DSCITVL_SET DMA_CMD_CHITVL_SET replaced with DMA_CMD_DSCITVL_SET and relative descriptions changed
1.20	Dec. 04, 2015	2. Operating Environment	
		5	Table 2.1 Operating Environment: Integrated Development Environment, information partially amended
1.30	Apr. 05, 2017	2. Operating Environment	
		5	Table 2.1 Operating Environment: Integrated Development Environment, modified
		6. Software	
		—	6.2.4 Required Memory Size, deleted
1.40	Jun. 07, 2018	2. Operating Environment	
		5	Table 2.1 Operating Environment: The description on the integrated development environment, modified
		8. Related Documents	
		55	The name of IAR Embedded Workbench, modified

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

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

1. 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 Microprocessing unit or Microcontroller unit products 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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

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

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

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

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

(Rev.4.0-1 November 2017)

RENESAS

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.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

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 Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

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

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

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: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.0