SH7786 Group

SH7786 DMAC Data Transfer Example

R01AN0807EJ0100
Rev.1.00
Jan 20, 2012

Introduction

This document presents a sample program that demonstrates data transfers using the HPB-DMAC and the SH7786's direct memory access controller units 0 and 1 (DMAC0/1).

Target Device

SH7786

Contents

1. Introduction

1.1 Specifications

This application note presents a sample program that demonstrates both the use of direct memory access controller units 0 and 1 (DMAC0/1) to transfer data between internal memory and external memory and the use of the HPB-DMAC to transfer data between a peripheral module and external memory.

1.2 Functions Used

- Direct memory access controller 0 (DMAC0, channels 0 and 4)
- Direct memory access controller 1 (DMAC1, channels 0 and 2)
- HPB-DMAC
- DDR3-SDRAM interface (DBSC3)
- Internal memory (OL memory)
- Built-in FIFO serial communications interface (SCIF channel 0)

1.3 Applicable Conditions

Evaluation board	AP-AH4AD-0A *1 manufactured by AlphaProject Co., Ltd. External memory Area 0: NOR flash memory: 16 MB S29GL128P90TFIR20 (Spansion Inc.) Areas 2 to 5: DDR3-SDRAM: 256 MB 2 × MT41J64M16LA-187E (Micron Technology)
MCU	SH7786
Operating frequencies	Internal clock: 533 MHz SuperHyway clock: 267 MHz Peripheral clock: 44 MHz DDR3 clock: 533 MHz External bus clock: 89 MHz
Area 0 bus width	16 bits (MD4 = low, MD5 = high, MD6 = low)
Clock operating mode	Clock mode 3 (MD0 = high, MD1 = high, MD2 = low, MD3 = low)
Endian mode	Little endian (MD8 = high)
Addressing mode	29-bit addressing mode (MD10 = low)
Tool chain	Super-H RISC engine Standard Toolchain Ver 9.3.2.0
Compiler options	Other than the include specifications for the High-performance Embedded Workshop, the default settings are used. -cpu=sh4a -endian=little -include="$(PROJDIR)\inc\drv","$(PROJDIR)\inc" -object="$(CONFIGDIR)\$(FILELEAF).obj" -debug -gbr=auto -chgincpath -errorpath -global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1 –nologo
Assembler options	cpu=sh4a -endian=little -round=zero -denormalize=off -include="$(PROJDIR)\inc" -include="$(PROJDIR)\inc\drv" –debug -object="$(CONFIGDIR)\$(FILELEAF).obj" -literal=pool,branch,jump,return -nolist -nologo -chgincpath -errorpath

Note: 1. For detailed information on using the AP-SH4AD-0A, refer to *AP-SH4AD-0A Hardware Manual.*

Table 1.1 lists the section allocations used in this sample program.

Table 1.1 Section Allocations

Section	Section Usage	Area	Allocation Address (Virtual Address)	
INTHandler	Exception/interrupt handler	ROM	0x00000800	P0 area (Can be cached, MMU address conversion not possible)
VECTTBL	Reset vector table Interrupt vector table	ROM		
INTTBL	Interrupt mask table	ROM		
PIntPRG	Interrupt function	ROM		
PResetPRG	Reset program	ROM	0x00002000	
P	Program area	ROM	0x00004000	
C	Constant area	ROM		
C$BSEC	Uninitialized data area address structure	ROM		
C$DSEC	Initialized data area address structure	ROM		
D	Initialized data	ROM		
B	Uninitialized data area	RAM	0x0DF00000	
R	Initialized data area	RAM		
S	Stack area	RAM	0x0DFF0000	
RSTHandler	Reset handler	ROM	0xA0000000	P2 area (Can not be cached, MMU address conversion not possible)

1.4 Related Application Notes

The reference program used in this document has been verified to operate under the settings and conditions described in the SH7786 Group Application Note SH7786 Initial Settings Example (R01AN0519EJ0101).

The SCIF asynchronous mode initial settings are taken from the SH7786 Group Application Note - SH7786 PCI Express Controller (PCIEC) Initial Settings Example (R01AN557EJ0100).

Those application notes should be consulted in conjunction with this application note.

2. DMAC0 Memory Transfer Example

2.1 Application Example

This sample program transfers data between internal RAM and external memory using DMAC0 channels 0 and 4. OL memory is used as the internal memory and DDR3-SDRAM as the external memory. This data transfer is performed in cycle-stealing mode, and normal mode and intermittent mode 16/32 are used. Also, auto-request is used as the DMA transfer request. The transfer method is selected by key input to a serial console using the built-in FIFO serial communications interface (SCIF channel 0).

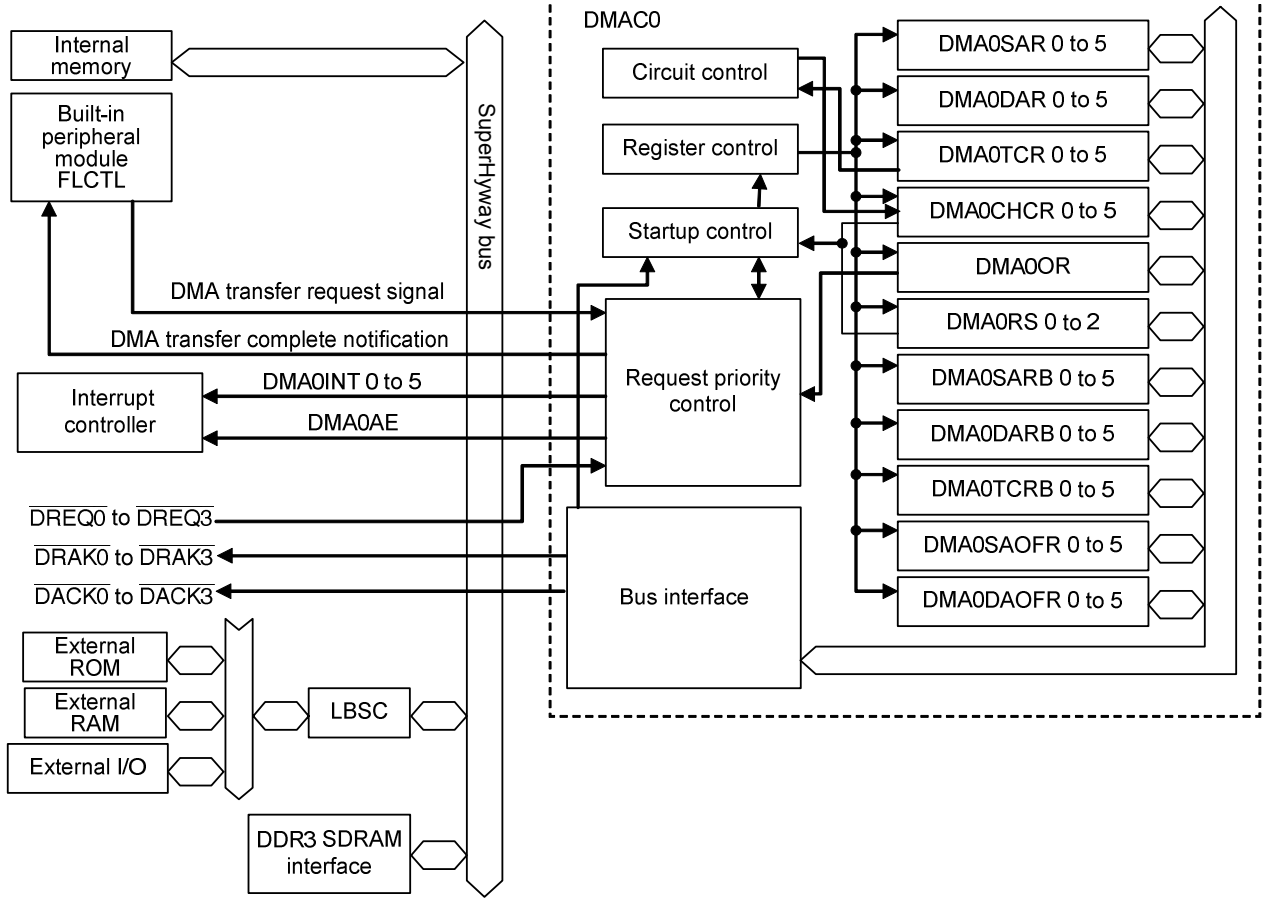
2.1.1 Functions Used and Operation Overview

When a DMA transfer request occurs, DMAC0 starts a transfer according to the determined channel priority and terminates the transfer when the transfer termination conditions are met. There are three modes for transfer requests: auto-request, external request, and built-in peripheral module request. There are two bus modes: burst mode and cycle-stealing mode. Either normal mode or intermittent mode can be selected for cycle-stealing mode.

Table 2.1 lists the DMAC0 module features and their descriptions and figure 2.1 presents a conceptual overview of this module.

Table 2.1 DMAC0 Features

Item	Description
Number of channels	• 6 channels (channels 0 to 5) — Channels 0 to 3 can accept external requests
Address space	• Architecture supports 4 GB
Transfer data length	• Byte, word (2 bytes), longword (4 bytes), 16 bytes, and 32 bytes
Maximum transfer count	• 16,777,216 transfers
Addressing modes	• Dual addressing modes
Transfer requests	• One of three types may be selected: external request (channels 0 to 3), built-in peripheral module request, and auto-request. • Only the FLCTL module can issue built-in peripheral module requests.
Bus modes	• Cycle-stealing mode (normal mode and intermittent mode 16/32) • Burst mode (In external request mode, burst mode can only be set when PCMCIA ATA supplementary mode is enabled.)
Data transfers	• Repeat mode • Reload mode • Multidimensional mode — Multidimensional transfers, scatter transfers, gather transfers, and stride transfers
Priorities	• Fixed channel priority mode • Round robin mode
Interrupt requests	• Interrupt requests can be issued to the CPU at data transfer half end, data transfer complete, and if an address error occurs.
External request detection	• Either low level/high level detection or rising edge/falling edge detection can be selected for the DREQ input.
Transfer complete notification signal	• DACK can be independently set to the active level.



Abbreviations:

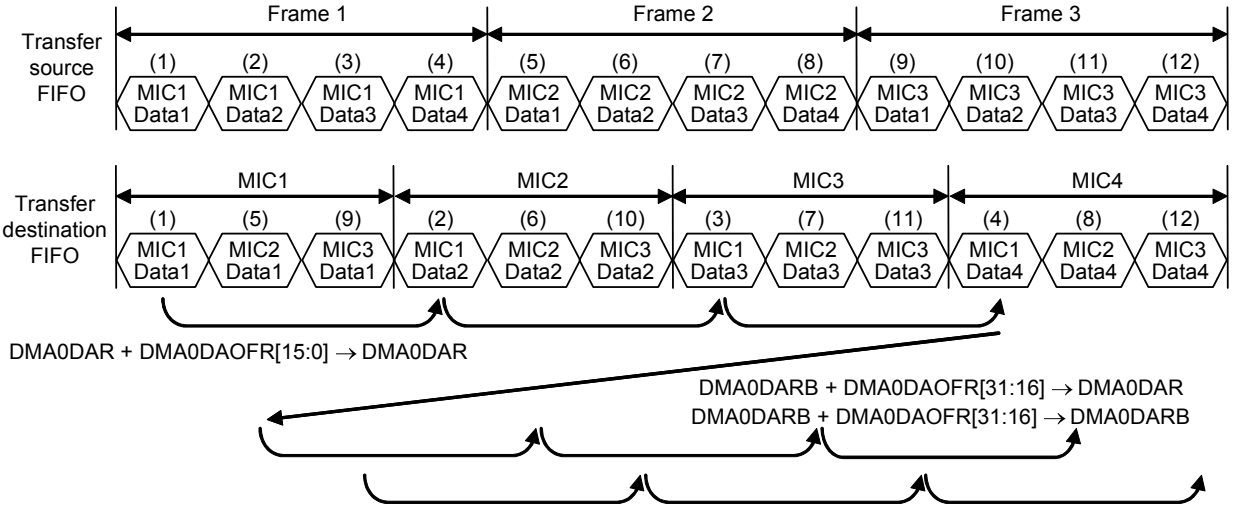
DMA0SAR 0 to 5: DMA0 source address register
DMA0SARB 0 to 5: DMA0 source address register B
DMA0DAR 0 to 5: DMA0 destination address register
DMA0DARB 0 to 5: DMA0 destination address register B
DMA0TCR 0 to 5: DMA0 transfer count register
DMA0TCRB 0 to 5: DMA0 transfer count register B
DMA0SAOFR 0 to 5: DMA0 source address offset register
DMA0DAOFR 0 to 5: DMA0 destination address offset register
DMA0CHCR 0 to 5: DMA0 channel control register
DMA0OR: DMA0 operation register
DMA0RS 0 to 2: DMA0 extension resource register
DMA0INT 0 to 5: DMA0 transfer complete/half end interrupt request
DMA0AE: Common channel address error interrupt request for all channels

Figure 2.1 DMAC0 Conceptual Overview

2.1.2 Transfer Methods

The DMAC0 module provides the following modes for data transfers: normal mode, repeat mode, remote mode, and multidimensional mode. Furthermore, multidimensional mode supports multidimensional, scatter, gather, and stride transfers. The following figures present sample settings for various transfer operations.

The audio data stored in transfer source FIFO memory shown below can be reordered and transferred using a multidimensional transfer.

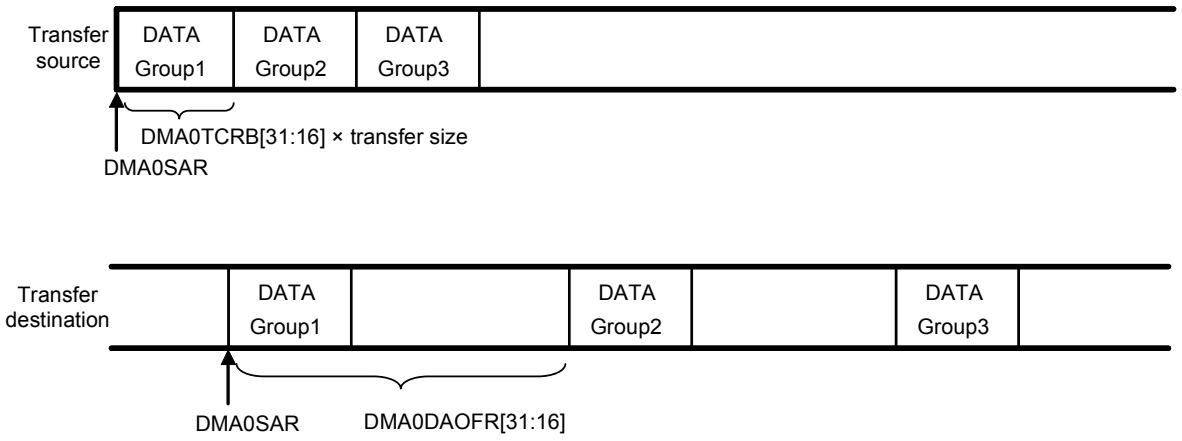


(1) to (12): DMA transfer sequence

• DMAC0 settings
DMA0SAR: Specifies an arbitrary memory address (a FIFO is assumed)
DMA0DAR: Specifies an arbitrary memory address
DMA0TCR = H'0000000C (12 transfer operations)
DMA0TCRB = H'00040004 (Updates DMA0DARB + DMA0DAOFR[31:16] → DMA0DAR and DMA0DARB + DMA0DAOFR[31:16] → DMA0DARB four times each)
DMA0DAOFR = H'00020006
DMA0CHCR: Set as follows
RPT[3:0] = B'1110: Multidimensional mode with the DMA0DAR address updated by the values specified in DMA0DAOFR[15:0] and DMA0DAOFR[31:16].
TS[2:0] = B'001: Word transfers
SM[1:0] = B'01: DMA0SAR is updated for each transfer size.
DE = B'1: Transfers enabled
Other fields are set according to RS and other usage conditions.
DME is set to 1 according to the DMA0OR CMS and PR usage conditions.
• The DMA0SAR and DMA0DAR addresses are updated as follows according to the above settings.
1st: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR
2nd: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR + H'0006
3rd: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR + H'000C
4th: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR + H'0012
5th: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR + H'0002
6th: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR + H'0008
7th: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR + H'000E
8th: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR + H'0014
9th: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR + H'0004
10th: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR + H'000A
11th: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR + H'0010
12th: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR + H'0016

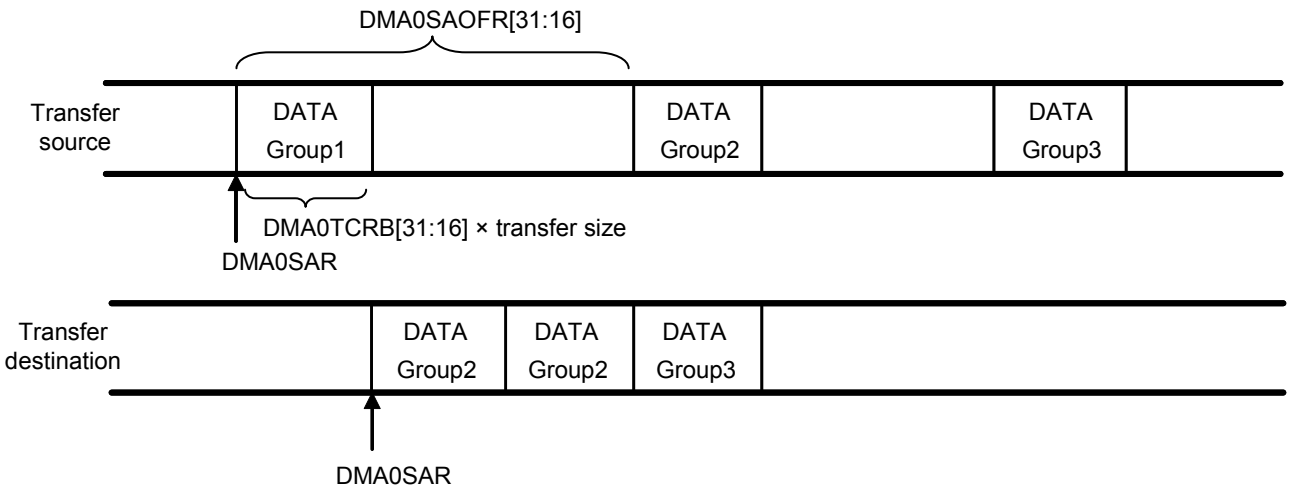
Figure 2.2 Multidimensional Transfer Example

Note: See section 2.1.5 (1), Multidimensional Mode Multidimensional Transfers, for more information on multidimensional transfers.



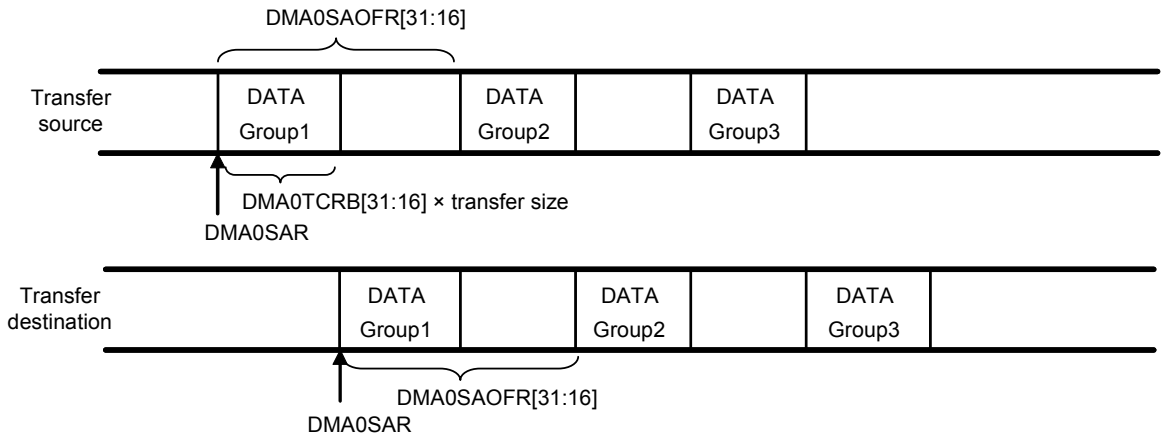
• DMAC0 settings
DMA0SAR: Specifies an arbitrary memory address
DMA0DAR: Specifies an arbitrary memory address
DMA0TCR = H'0000000C (12 transfer operations)
DMA0TCRB = H'00040004 (Updates DMA0DARB + DMA0DAOFR[31:16] → DMA0DAR and DMA0DARB + DMA0DAOFR[31:16] → DMA0DARB four times each)
DMA0DAOFR = H'02000004 (DMA0DAOFR[15:0] must be set to the same value as the transfer size.)
DMA0CHCR: Set as follows
RPT[3:0] = B'1110: Multidimensional mode with the DMA0DAR address updated by the values specified in DMA0DAOFR[15:0] and DMA0DAOFR[31:16].
TS[2:0] = B'010: Longword transfers
SM[1:0] = B'01: DMA0SAR is incremented.
DE = B'1: Transfers enabled
Other fields are set according to RS and other usage conditions.
DME is set to 1 according to the DMA0OR CMS and PR usage conditions.
• The DMA0SAR and DMA0DAR addresses are updated as follows according to the above settings.
1st: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR
2nd: Transfer source address = DMA0SAR + H'0004, transfer destination address = DMA0DAR + H'0004
3rd: Transfer source address = DMA0SAR + H'0008, transfer destination address = DMA0DAR + H'0008
4th: Transfer source address = DMA0SAR + H'000C, transfer destination address = DMA0DAR + H'000C
5th: Transfer source address = DMA0SAR + H'0010, transfer destination address = DMA0DAR + H'0200
6th: Transfer source address = DMA0SAR + H'0014, transfer destination address = DMA0DAR + H'0204
7th: Transfer source address = DMA0SAR + H'0018, transfer destination address = DMA0DAR+ H'0208
8th: Transfer source address = DMA0SAR + H'001C, transfer destination address = DMA0DAR + H'020C
9th: Transfer source address = DMA0SAR + H'0020, transfer destination address = DMA0DAR + H'0400
10th: Transfer source address = DMA0SAR + H'0024, transfer destination address = DMA0DAR + H'0404
11th: Transfer source address = DMA0SAR + H'0028, transfer destination address = DMA0DAR + H'0408
12th: Transfer source address = DMA0SAR + H'002C, transfer destination address = DMA0DAR + H'040C

Figure 2.3 Scatter Transfer Example



• DMAC0 settings
DMA0SAR: Specifies an arbitrary memory address
DMA0DAR: Specifies an arbitrary memory address
DMA0TCR = H'0000000C (12 transfer operations)
DMA0TCRB = H'00040004 (Updates DMA0DARB + DMA0DAOFR[31:16] → DMA0DAR and DMA0DARB + DMA0DAOFR[31:16] → DMA0DARB four times each)
DMA0DAOFR = H'02000004 (DMA0DAOFR[15:0] must be set to the same value as the transfer size.)
DMA0CHCR: Set as follows
RPT[3:0] = B'1110: Multidimensional mode with the DMA0DAR address updated by the values specified in DMA0DAOFR[15:0] and DMA0DAOFR[31:16].
TS[2:0] = B'010: Longword transfers
SM[1:0] = B'01: DMA0SAR is incremented.
DE = B'1: Transfers enabled
Other fields are set according to RS and other usage conditions.
DME is set to 1 according to the DMA0OR CMS and PR usage conditions.
• The DMA0SAR and DMA0DAR addresses are updated as follows according to the above settings.
1st: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR
2nd: Transfer source address = DMA0SAR + H'0004, transfer destination address = DMA0DAR + H'0004
3rd: Transfer source address = DMA0SAR + H'0008, transfer destination address = DMA0DAR + H'0008
4th: Transfer source address = DMA0SAR + H'000C, transfer destination address = DMA0DAR + H'000C
5th: Transfer source address = DMA0SAR + H'0010, transfer destination address = DMA0DAR + H'0200
6th: Transfer source address = DMA0SAR + H'0014, transfer destination address = DMA0DAR + H'0204
7th: Transfer source address = DMA0SAR + H'0018, transfer destination address = DMA0DAR+ H'0208
8th: Transfer source address = DMA0SAR + H'001C, transfer destination address = DMA0DAR + H'020C
9th: Transfer source address = DMA0SAR + H'0020, transfer destination address = DMA0DAR + H'0400
10th: Transfer source address = DMA0SAR + H'0024, transfer destination address = DMA0DAR + H'0404
11th: Transfer source address = DMA0SAR + H'0028, transfer destination address = DMA0DAR + H'0408
12th: Transfer source address = DMA0SAR + H'002C, transfer destination address = DMA0DAR + H'040C

Figure 2.4 Gather Transfer Example



• DMAC0 settings
DMA0SAR: Specifies an arbitrary memory address
DMA0DAR: Specifies an arbitrary memory address
DMA0TCR = H'0000000C (12 transfer operations)
DMA0TCRB = H'00040004 (Updates DMA0DARB + DMA0DAOFR[31:16] → DMA0SAR,
DMA0SARB + DMA0SAOFR[31:16] → DMA0SARB,
DMA0DARB + DMA0DAOFR[31:16] → DMA0DAR,
DMA0DARB + DMA0DAOFR[31:16] → DMA0DARB four times each)
DMA0SAOFR = H'0100_0004 (DMA0SAOFR[15:0] must be set to the same value as the transfer size.)
DMA0DAOFR = H'0200_0004 (DMA0DAOFR[15:0] must be set to the same value as the transfer size.)
DMA0CHCR: Set as follows
RPT[3:0] = B'1101: Multidimensional mode with the DMA0SAR address updated by the values specified in DMA0SAOFR[15:0] and DMA0SAOFR[31:16] and with the DMA0DAR address updated by the values specified in DMA0DAOFR[15:0] and DMA0DAOFR[31:16].
TS[2:0] = B'010: Longword transfers
DE = B'1: Transfers enabled
Other fields are set according to RS and other usage conditions.
DME is set to 1 according to the DMA0OR CMS and PR usage conditions.
• The DMA0SAR and DMA0DAR addresses are updated as follows according to the above settings.
1st: Transfer source address = DMA0SAR, transfer destination address = DMA0DAR
2nd: Transfer source address = DMA0SAR + H'0004, transfer destination address = DMA0DAR + H'0004
3rd: Transfer source address = DMA0SAR + H'0008, transfer destination address = DMA0DAR + H'0008
4th: Transfer source address = DMA0SAR + H'000C, transfer destination address = DMA0DAR + H'000C
5th: Transfer source address = DMA0SAR + H'0100, transfer destination address = DMA0DAR + H'0200
6th: Transfer source address = DMA0SAR + H'0104, transfer destination address = DMA0DAR + H'0204
7th: Transfer source address = DMA0SAR + H'0108, transfer destination address = DMA0DAR+ H'0208
8th: Transfer source address = DMA0SAR + H'010C, transfer destination address = DMA0DAR + H'020C
9th: Transfer source address = DMA0SAR + H'0200, transfer destination address = DMA0DAR + H'0400
10th: Transfer source address = DMA0SAR + H'0204, transfer destination address = DMA0DAR + H'0404
11th: Transfer source address = DMA0SAR + H'0208, transfer destination address = DMA0DAR + H'0408
12th: Transfer source address = DMA0SAR + H'020C, transfer destination address = DMA0DAR + H'040C

Figure 2.5 Stride Transfer Example

2.1.3 Sample Program

The sample program uses auto-request mode to activate DMAC0 channel 0 or channel 4 to perform bidirectional data transfers between internal RAM and external memory in cycle-stealing mode. Since cycle-stealing mode is used, the DMAC releases the bus to the CPU after each individual data transfer operation.

Here, it is also possible to select whether cache coherence with external memory is guaranteed during the DMA transfers by performing flush/purge operations. When the flush/purge operations are controlled in software and a flush/purge is not performed, it is possible for inconsistencies to occur between the transfer source data and transfer destination data. See section 7, Coherence Control Between Cache and External Memory, for details.

Table 2.2 lists the specifications for this program.

Table 2.2 Sample Program Specifications

Item	Specification
Channels used	• Channel 0 • Channel 4
Memory	• OL memory (internal memory) • DDR3-SDRAM (external memory)
Transfer direction	• OL memory → DDR3-SDRAM • DDR3-SDRAM → OL memory
Transfer data amounts	• Channel 0: 4-byte units • Channel 4: 1-byte units
Transfer data size	• Channel 0: Longword (4 bytes), 16 bytes, or 32 bytes • Channel 4: Byte, word (2 bytes), longword (4 bytes), 16 bytes, or 32 bytes
Number of transfers	• Calculated from the transfer data size
Transfer requests	• Auto-request
Bus mode	• Cycle-stealing mode — Normal mode — Intermittent mode 16 — Intermittent mode 32
Data transfers	• Normal mode (continuous transfers) • Repeat mode • Reload mode • Multidimensional mode — Multidimensional transfer — Scatter transfer — Gather transfer — Stride transfer
Priority	• Fixed channel priority mode
Interrupt requests	• An interrupt is issued to the CPU on transfer complete or if an address error occurs.
Coherence control between cache and external memory	• Copy back mode • Operand cache and secondary cache enabled • Cache flush/purge operations controlled in software (On (controlled)/off (not controlled) can be selected from a menu.) Note: If cache coherence is not controlled in copy back mode, it is possible for the operand cache and external memory to become inconsistent. See section 7, Coherence Control Between Cache and External Memory, for details.

2.1.4 Sample Program Register Settings

The following tables list the functions of the registers used by the sample program.

Table 2.3 DMAC0 Register Settings (Settings common to both channels)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA1 operation register (DMA1OR)	H'FE80 8060	R/W	16	H'0001	• Cycle-stealing mode selected — At initialization CMS = B'00: normal mode DMA1E = B'1: DMA transfers enabled for all channels
				H'2001	• Cycle-stealing mode selected CMS = B'10: Intermittent mode 16 DMA1E = B'1: DMA transfers enabled for all channels
				H'3001	• Cycle-stealing mode selected CMS = B'11: Intermittent mode 64 DMA1E = B'1: DMA transfers enabled for all channels
				H'0000	• Cycle-stealing mode selected — At initialization CMS = B'00: Normal mode DMA1E = B'0: DMA transfers disabled for all channels
				H'2000	• Cycle-stealing mode selected CMS = B'10: Intermittent mode 16 DMA1E = B'0: DMA transfers disabled for all channels
				H'3000	• Cycle-stealing mode selected CMS = B'11: Intermittent mode 64 DMA1E = B'0: DMA transfers disabled for all channels

Note: Registers not used by this program and bits that are not set remain set to their initial values.

Table 2.4 DMAC0 Register Settings 1 (Channel 0)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA0 source address register 0 (DMA0SAR0)	H'FE80 8020	R/W	32	H'1400 E000	• Specifies the start address for the transfer source When OL memory is specified (DMA0DAR0 specifies DDR3-SDRAM)
				H'0800 0000	• Specifies the start address for the transfer source When DDR3-SDRAM is specified (DMA0DAR0 specifies OL memory)
DMA0 destination address register 0 (DMA0DAR0)	H'FE80 8024	R/W	32	H'1400 E000	• Specifies the start address for the transfer destination When OL memory is specified (DMA0SAR0 specifies DDR3-SDRAM)
				H'0800 0000	• Specifies the start address for the transfer destination When DDR3-SDRAM is specified (DMA0SAR0 specifies OL memory)
DMA0 transfer count register 0 (DMA0TCR0)	H'FE80 8028	R/W	32	H'0000 0064	• Sets the transfer count 100 times (when the transfer size is 1 byte) Note: For normal, reload, and stride mode transfers
				H'0000 0032	• Sets the transfer count 50 times (when the transfer size is 2 bytes) Note: For normal, reload, and stride mode transfers
				H'0000 0020	• Sets the transfer count 32 times (when the transfer size is 4 bytes) Note: For normal, reload, and stride mode transfers
				H'0000 0008	• Sets the transfer count 8 times (when the transfer size is 16 bytes) Note: For normal, reload, and stride mode transfers
				H'0000 0004	• Sets the transfer count 4 times (when the transfer size is 32 bytes) Note: For normal, reload, and stride mode transfers
				H'0000 0032	• Sets the transfer count 50 times (when the transfer size is 1 byte) Note: For repeat mode, scatter, and gather transfers (Scatter and gather transfers are multidimensional mode transfers)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA0 transfer count register 0 (DMA0TCR0)	H'FE80 8028	R/W	32	H'0000 0016	• Sets the transfer count 32 times (when the transfer size is 2 bytes) Note: For repeat mode, scatter, and gather transfers (Scatter and gather transfers are multidimensional mode transfers)
				H'0000 0010	• Sets the transfer count 16 times (when the transfer size is 4 bytes) Note: For repeat mode, scatter, and gather transfers (Scatter and gather transfers are multidimensional mode transfers)
				H'0000 0004	• Sets the transfer count 4 times (when the transfer size is 16 bytes) Note: For repeat mode, scatter, and gather transfers (Scatter and gather transfers are multidimensional mode transfers)
				H'0000 0002	• Sets the transfer count 2 times (when the transfer size is 32 bytes) Note: For repeat mode, scatter, and gather transfers (Scatter and gather transfers are multidimensional mode transfers)
				H'0000 000C	• Sets the transfer count 12 times (all transfer sizes) For multidimensional transfers

Table 2.5 DMAC0 Register Settings 2 (Channel 0)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA0 source address register B0 (DMA0SARB0)	H'FE80 8120	R/W	32	H'1400 E032	• Address reset into DMA0DAR0 When OL memory is specified When the transfer size is 1 or 2 bytes in repeat mode
				H'1400 E040	• Address reset into DMA0DAR0 When OL memory is specified When the transfer size is 4 to 32 bytes in repeat mode
DMA0 destination address register B0 (DMA0DARB0)	H'FE80 8124	R/W	32	H'0800 0032	• Address reset into DMA0DAR0 When DDR3-SDRAM is specified When the transfer size is 1 or 2 bytes in repeat mode
				H'0800 0040	• Address reset into DMA0DAR0 When DDR3-SDRAM is specified When the transfer size is 4 to 32 bytes in repeat mode
DMA0 transfer count register B0 (DMA0TCRB0)	H'FE80 8128	R/W	32	H'0001 0001	• Reload mode, scatter, and gather transfers Bits [31:16]: Specifies the transfer count reloaded into bits [15:0] Bits [15:0]: Transfer count counter
				H'0002 0002	• Stride transfer Bits [31:16]: Specifies the transfer count reloaded into bits [15:0] Bits [15:0]: Transfer count counter
				H'0004 0004	• Multidimensional transfer Bits [31:16]: Specifies the transfer count reloaded into bits [15:0] Bits [15:0]: Transfer count counter
DMA0 source address offset register 0 (DMA0SAOFR0)	H'FE80 8220	R/W	32	H'0002 0002	• Stride and gather transfers (transfer size: 1 byte) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0004 0004	• Stride and gather transfers (transfer size: 2 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0004 0004	• Stride and gather transfers (transfer size: 4 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA0 source address offset register 0 (DMA0SAOFR0)	H'FE80 8220	R/W	32	H'0020 0020	• Stride and gather transfers (transfer size: 16 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0040 0040	• Stride and gather transfers (transfer size: 32 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer

Table 2.6 DMAC0 Register Settings 3 (Channel 0)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA0 destination address offset register 0 (DMA0DAOFR0)	H'FE80 8224	R/W	32	H'0002 0002	• Stride and scatter transfers (transfer size: 1 byte) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0004 0004	• Stride and scatter transfers (transfer size: 2 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0004 0004	• Stride and scatter transfers (transfer size: 4 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0020 0020	• Stride and scatter transfers (transfer size: 16 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0040 0040	• Stride and scatter transfers (transfer size: 32 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0001 0003	• Multidimensional transfer (transfer size: 1 byte) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA0 destination address offset register 0 (DMA0DAOFR0)	H'FE80 8224	R/W	32	H'0002 0006	• Multidimensional transfer (transfer size: 2 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0004 0010	• Multidimensional transfer (transfer size: 4 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0010 0030	• Multidimensional transfer (transfer size: 16 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0020 0060	• Multidimensional transfer (transfer size: 32 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer

Table 2.7 DMAC0 Channel Control Register 0 Settings 1 (channel 0)

		Operational Specification		
Register (Abbreviation)	**Address**	**Bit Name**	**Set Value**	**Description**
DMA0 channel control register 0 (DMA0CHCR0)	H'FE80 802C	RPT[3:0] (bits 28 to 25)	H'0	Normal mode
			H'3	Repeat mode
			H'7	Reload mode
			H'D	• Multidimensional mode Stride transfer Note: SAR is updated by SAOFR, DAR is updated by DAOFR
			H'E	• Multidimensional mode Multidimensional transfer Scatter transfer Note: DAR is updated by DAOFR
			H'F	• Multidimensional mode Gather transfer Note: SAR is updated by DASAR
		TS[2:0] (bits 20, 4 and 3)	H'0	• DMA transfer size specification Byte unit
			H'1	• DMA transfer size specification Word units
			H'2	• DMA transfer size specification Longword units
			H'3	• DMA transfer size specification 16-byte units
			H'4	• DMA transfer size specification 32-byte units
		DM (bits15 and 14)	H'1	• Destination address mode Destination address incremented
		SM (bits 13 and 12)	H'1	• Source address mode Source address incremented
		RS (bits 11 to 6)	H'8	• Resource select Built-in peripheral module request
		IE (bit 2)	H'1	• Interrupt enable At initialization: enabled
			H'0	• Interrupt enable At interrupt handling: disabled
		TE (bit 1)	H'0	• Transfer end flag Note: Set to H'1 at the start of the last transfer
		DE (bit 0)	H'0	• DMA enable At initialization and transfer complete
			H'1	• DMA enable At transfer start

Table 2.8 DMAC0 Register Settings 1 (Channel 4)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA0 source address register 4 (DMA0SAR4)	H'FE80 8070	R/W	32	H'1400 E000	• Specifies the start address for the transfer source When OL memory is specified (DMA0DAR4 specifies DDR3-SDRAM)
				H'0800 0000	• Specifies the start address for the transfer source When DDR3-SDRAM is specified (DMA0DAR4 specifies OL memory)
DMA0 destination address register 4 (DMA0DAR4)	H'FE80 8074	R/W	32	H'1400 E000	• Specifies the start address for the transfer destination When OL memory is specified (DMA0SAR4 specifies DDR3-SDRAM)
				H'0800 0000	• Specifies the start address for the transfer destination When DDR3-SDRAM is specified (DMA0SAR4 specifies OL memory)
DMA0 transfer count register 4 (DMA0TCR4)	H'FE80 8078	R/W	32	H'0000 0064	• Sets the transfer count 100 times (when the transfer size is 1 byte) Note: For normal, reload, and stride mode transfers
				H'0000 0032	• Sets the transfer count 50 times (when the transfer size is 2 bytes) Note: For normal, reload, and stride mode transfers
				H'0000 0020	• Sets the transfer count 32 times (when the transfer size is 4 bytes) Note: For normal, reload, and stride mode transfers
				H'0000 0008	• Sets the transfer count 8 times (when the transfer size is 16 bytes) Note: For normal, reload, and stride mode transfers
				H'0000 0004	• Sets the transfer count 4 times (when the transfer size is 32 bytes) Note: For normal, reload, and stride mode transfers
				H'0000 0032	• Sets the transfer count 50 times (when the transfer size is 1 byte) Note: For repeat mode, scatter, and gather transfers (Scatter and gather transfers are multidimensional mode transfers)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA0 transfer count register 4 (DMA0TCR4)	H'FE80 8078	R/W	32	H'0000 0016	• Sets the transfer count 32 times (when the transfer size is 2 bytes) Note: For repeat mode, scatter, and gather transfers (Scatter and gather transfers are multidimensional mode transfers)
				H'0000 0010	• Sets the transfer count 16 times (when the transfer size is 4 bytes) Note: For repeat mode, scatter, and gather transfers (Scatter and gather transfers are multidimensional mode transfers)
				H'0000 0004	• Sets the transfer count 4 times (when the transfer size is 16 bytes) Note: For repeat mode, scatter, and gather transfers (Scatter and gather transfers are multidimensional mode transfers)
				H'0000 0002	• Sets the transfer count 2 times (when the transfer size is 32 bytes) Note: For repeat mode, scatter, and gather transfers (Scatter and gather transfers are multidimensional mode transfers)
				H'0000 000C	• Sets the transfer count 12 times (all transfer sizes) For multidimensional transfers

Table 2.9 DMAC0 Register Settings 2 (Channel 4)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA0 source address register B4 (DMA0SARB4)	H'FE80 8170	R/W	32	H'1400 E032	• Address reset into DMA0DAR0 When OL memory is specified When the transfer size is 1 or 2 bytes in repeat mode
				H'1400 E040	• Address reset into DMA0DAR0 When OL memory is specified When the transfer size is 4 to 32 bytes in repeat mode
DMA0 destination address register B4 (DMA0DARB4)	H'FE80 8174	R/W	32	H'0800 0032	• Address reset into DMA0DAR0 When DDR3-SDRAM is specified When the transfer size is 1 or 2 bytes in repeat mode
				H'0800 0040	• Address reset into DMA0DAR0 When DDR3-SDRAM is specified When the transfer size is 4 to 32 bytes in repeat mode
DMA0 transfer count register B4 (DMA0TCRB4)	H'FE80 8178	R/W	32	H'0001 0001	• Reload mode, scatter, and gather transfers Bits [31:16]: Specifies the transfer count reloaded into bits [15:0] Bits [15:0]: Transfer count counter
				H'0002 0002	• Stride transfer Bits [31:16]: Specifies the transfer count reloaded into bits [15:0] Bits [15:0]: Transfer count counter
				H'0004 0004	• Multidimensional transfer Bits [31:16]: Specifies the transfer count reloaded into bits [15:0] Bits [15:0]: Transfer count counter
DMA0 source address offset register 4 (DMA0SAOFR4)	H'FE80 8270	R/W	32	H'0002 0002	• Stride and gather transfers (transfer size: 1 byte) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0004 0004	• Stride and gather transfers (transfer size: 2 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0004 0004	• Stride and gather transfers (transfer size: 4 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA0 source address offset register 4 (DMA0SAOFR4)	H'FE80 8270	R/W	32	H'0020 0020	• Stride and gather transfers (transfer size: 16 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0040 0040	• Stride and gather transfers (transfer size: 32 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer

Table 2.10 DMAC0 Register Settings 3 (Channel 4)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA0 destination address offset register 4 (DMA0DAOFR4)	H'FE80 8274	R/W	32	H'0002 0002	• Stride and scatter transfers (transfer size: 1 byte) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0004 0004	• Stride and scatter transfers (transfer size: 2 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0004 0004	• Stride and scatter transfers (transfer size: 4 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0020 0020	• Stride and scatter transfers (transfer size: 16 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0040 0040	• Stride and scatter transfers (transfer size: 32 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0001 0003	• Multidimensional transfer (transfer size: 1 byte) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0002 0006	• Multidimensional transfer (transfer size: 2 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0004 0010	• Multidimensional transfer (transfer size: 4 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA0 destination address offset register 4 (DMA0DAOFR4)	H'FE80 8274	R/W	32	H'0010 0030	• Multidimensional transfer (transfer size: 16 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer
				H'0020 0060	• Multidimensional transfer (transfer size: 32 bytes) Bits [31:16]: Sets the address offset that is reloaded Bits [15:0]: Sets the address increment on each transfer

Table 2.11 DMAC0 Channel Control Register 4 Settings 1 (channel 4)

		Operational Specification		
Register (Abbreviation)	**Address**	**Bit Name**	**Set Value**	**Description**
DMA0 channel control register 4 (DMA0CHCR4)	H'FE80 807C	RPT[3:0] (bits 28 to 25)	H'0	Normal mode
			H'3	Repeat mode
			H'7	Reload mode
			H'D	• Multidimensional mode Stride transfer Note: SAR is updated by SAOFR, DAR is updated by DAOFR
			H'E	• Multidimensional mode Multidimensional transfer Scatter transfer Note: DAR is updated by DAOFR
			H'F	• Multidimensional mode Gather transfer Note: SAR is updated by DASAR
		TS[2:0] (bits 20, 4 and 3)	H'0	• DMA transfer size specification Byte unit
			H'1	• DMA transfer size specification Word units
			H'2	• DMA transfer size specification Longword units
			H'3	• DMA transfer size specification 16-byte units
			H'4	• DMA transfer size specification 32-byte units
		DM (bits 15 and 14)	H'1	• Destination address mode Destination address incremented
		SM (bits13 and 12)	H'1	• Source address mode Source address incremented
		RS (bits 11 to 6)	H'8	• Resource select Built-in peripheral module request
		IE (bit 2)	H'1	• Interrupt enable At initialization: enabled
			H'0	• Interrupt enable At interrupt handling: disabled
		TE (bit 1)	H'0	• Transfer end flag Note: Set to H'1 at the start of the last transfer
		DE (bit 0)	H'0	• DMA enable At initialization and transfer complete
			H'1	• DMA enable At transfer start

Note: Registers not used by this program and bits that are not set remain set to their initial values.

2.1.5 Notes on Program Creation

Keep the following points in mind when creating programs that use DMAC0.

(1) Multidimensional Mode Multidimensional Transfers

While multidimensional transfers allow the data to be reordered, they cannot reorder in higher dimensions. These transfers should be seen performing an X/Y conversion on data that is seen as being a 2-dimensional matrix.

The following figure presents an example of operation when data is seen as a 2-dimensional matrix based on the multidimensional transfer operation example presented in figure 15.10 in the SH7786 Group Hardware Manual.

• DMAC0 multidimensional mode multidimensional transfer (in 2-byte units)
Transfer count: 12 operations (24 bytes)
Register Settings

DMA0SAR	H'E500 E000
DMA0DAR	H'0800 0000
DMA0TCR	H'0000 000C (12 transfers)
DMA0TCRB	H'0004 0004 Updates by transferring DMA0DARB + DMA0DAOFR[31:16] → DMA0DAR and DMA0DARB + DMA0DAOFR[31:16] → DMA0DARB four times each.
DMA0DAFOR	H'0002 0006 RPT[3:0] = B'1110: Multidimensional mode with the DMA0DAR address updated by the values specified in DMA0DAOFR[15:0] and DMA0DAOFR[31:16]. TS[2:0] = B'001: Word transfers
DMAOCHCR	SM[1:0] = B'01: DMA0SAR is incremented each time data of the transfer size is transferred. DE = B'1: Transfers enabled Other fields are set according to RS and other usage conditions. DME is set to 1 according to the DMA0OR CMS and PR usage conditions.

Source address:

H'E500 E000 - H'E5000 E016

	H'E500 E000	H'E500 E002	H'E500 E004	H'E500 E006	H'E500 E008	H'E500 E00A	H'E500 E00C	H'E500 E00E
+00	0001	0203	0405	0607	0809	0a0b	0c0d	0e0f
+10	1011	1213	1415	1617				

Destination address:

H'0800 0000 - H'08000 0016

	H'0800 0000	H'0800 0002	H'0800 0004	H'0800 0006	H'0800 0008	H'0800 000A	H'0800 000C	H'0800 000E
+00	0001	0809	1011	0203	0a0b	1213	0405	0c0d
+10	1415	0607	0e0f	1617				

• Two-dimensional matrix

Source address:

0001	0203	0405	0607
0809	0a0b	0c0d	0e0f
1011	1213	1415	1617

4x3 → 3x4

Destination address:

0001	0809	1011
0203	0a0b	1213
0405	0c0d	1415
0607	0e0f	1617

Figure 2.6 Multidimensional Transfer Operation Example

See the sample program for details on the coding.

3. DMAC1 Inter-Memory Transfer Example

3.1 Application Example

This application note's sample program transfers data between internal RAM and external memory (bidirectionally) using direct memory access controller 1 (DMAC1) channels 0 and 2. OL memory is used as the internal memory and DDR3-SDRAM as the external memory. The transfer method is selected from a serial console using the built-in FIFO serial communications interface (SCIF channel 0).

3.1.1 Functions Used and Operation Overview

When a DMA transfer request occurs, DMAC1 starts a transfer according to the determined channel priority and terminates the transfer when the transfer termination conditions are met. Continuous area transfers, stride transfers, and gather/scatter transfers are possible between resources on the SuperHyway bus.

Table 3.1 lists the DMAC1 module features and their descriptions and figure 3.1 presents a conceptual overview of this module.

Table 3.1 DMAC1 Features

Item	Description
Number of channels	• 4 channels (channels 0 to 3)
Address space	• Supports up to a 32-bit address space
Transfer data size	• Channels 0 and 1: 4-byte units • Channels 2 and 3: 1-byte units
Transfer data length	• Channels 0 and 1: 4, 8, 16, or 32 bytes (Note: Transfers must fall on a 32-byte boundary when the transfer source or destination is L memory, L2C memory, or LBSC.) • Channels 2 and 3: 1, 2, 4, 8, 16, or 32 bytes
Addressing modes	• Dual addressing modes
Priority	• Fixed channel priority
Interrupt requests	• DMA transfer complete, transfer source transfer error, and transfer destination transfer error interrupts can be generated for each channel. (corresponding to each channel)
Data transfers	• Channels 0 and 1: Continuous area transfers, stride transfers, and gather/scatter transfers are possible between resources on the SuperHyway bus. • Channels 2 and 3: Continuous area transfers are possible between resources on the SuperHyway bus.
Command chain	• Channels 0 and 1: Multiple data transfers can be executed continuously according to data transfer instructions set for specified addresses. • Channels 2 and 3: Command chains are not supported.

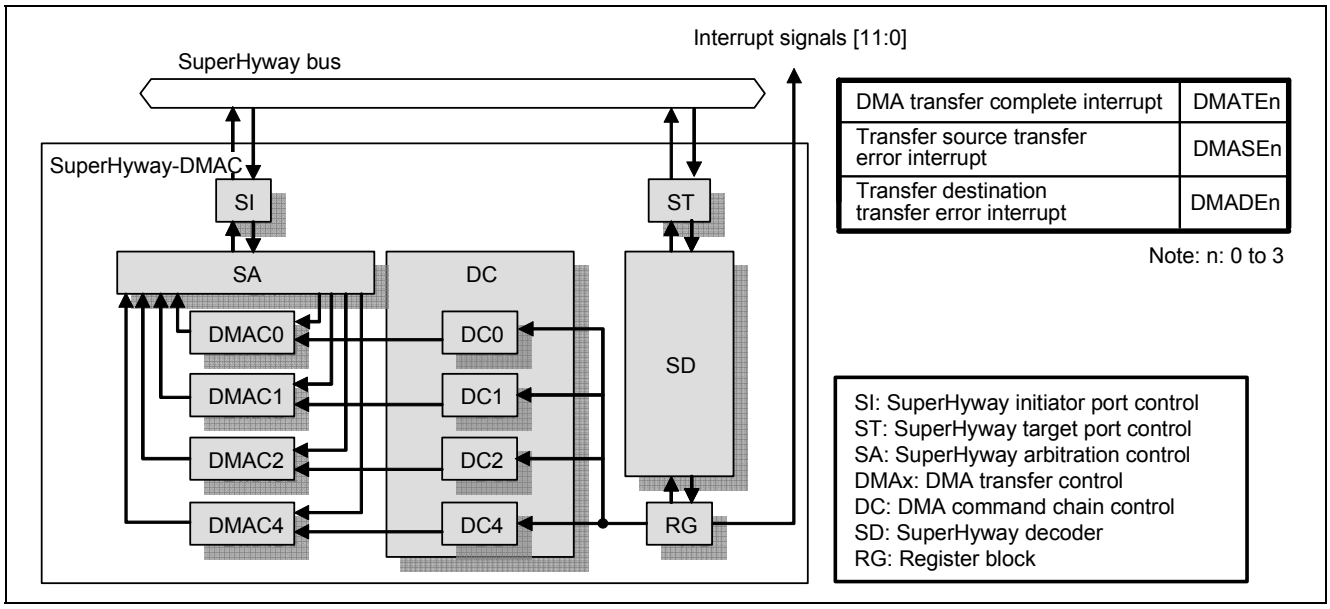


Figure 3.1 DMAC1 Conceptual Overview

3.1.2 Transfer Methods

The DMAC1 module provides continuous area transfers, stride transfers, and gather/scatter transfers. Only channels 0 and 1 support stride transfers and gather/scatter transfers. Channels 0 and 1 also support transfers using a command chain. This application note uses the channel 0 command chain function for stride and gather scatter transfers and channel 4 for continuous transfers.

The following figures present the stride transfer, gather/scatter transfer, and command chain operations.

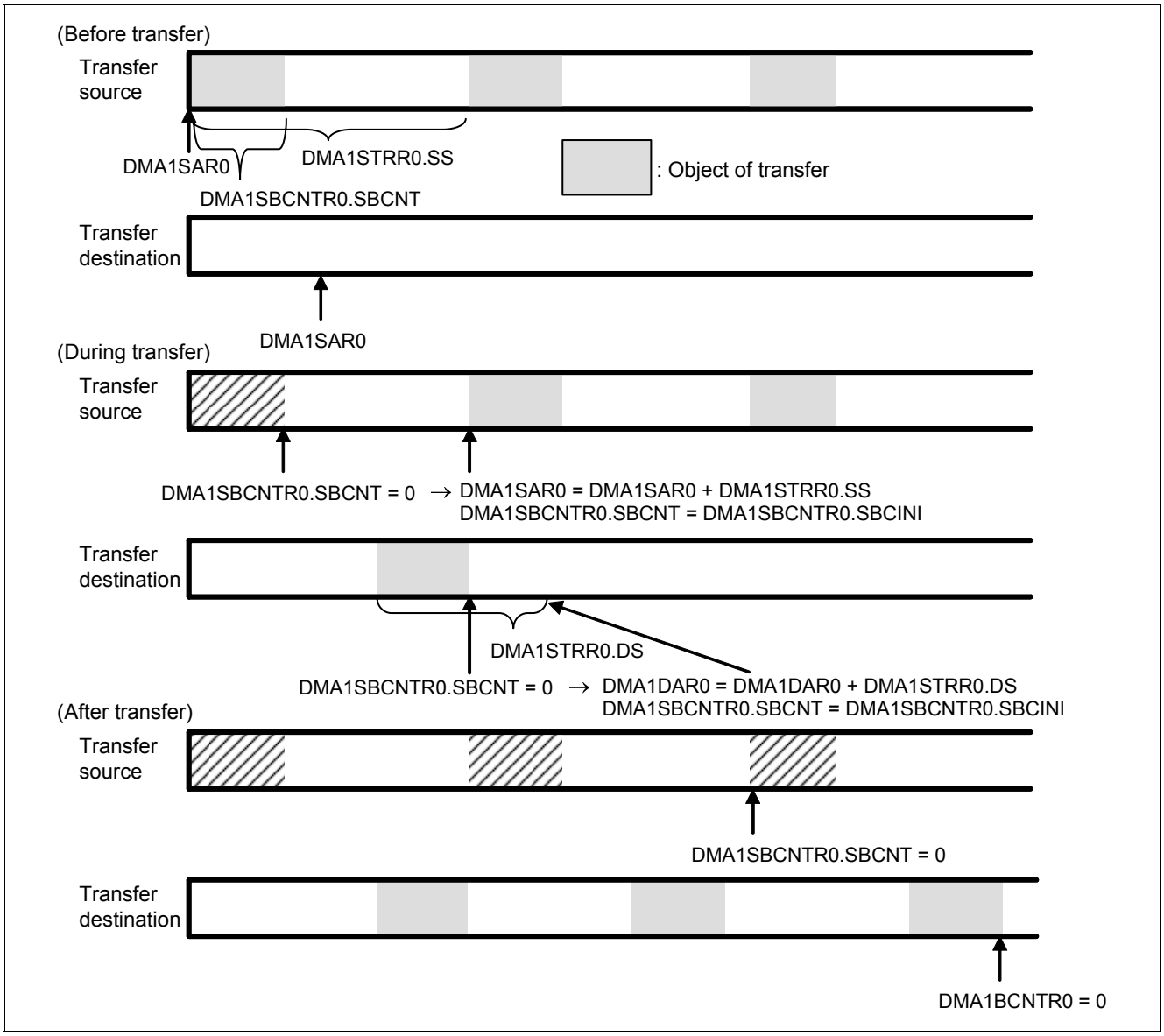


Figure 3.2 Stride Transfer

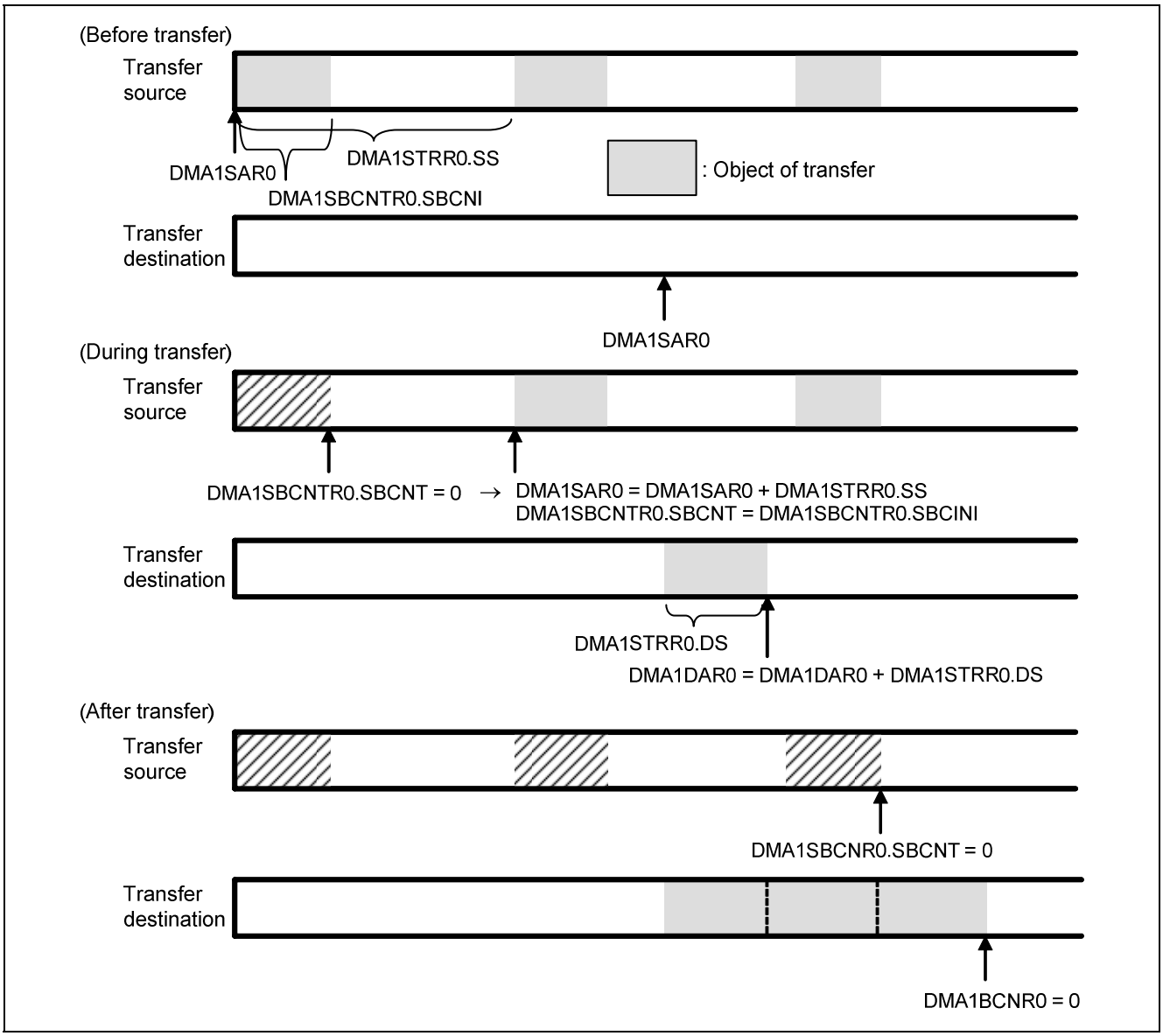


Figure 3.3 Gather Transfer

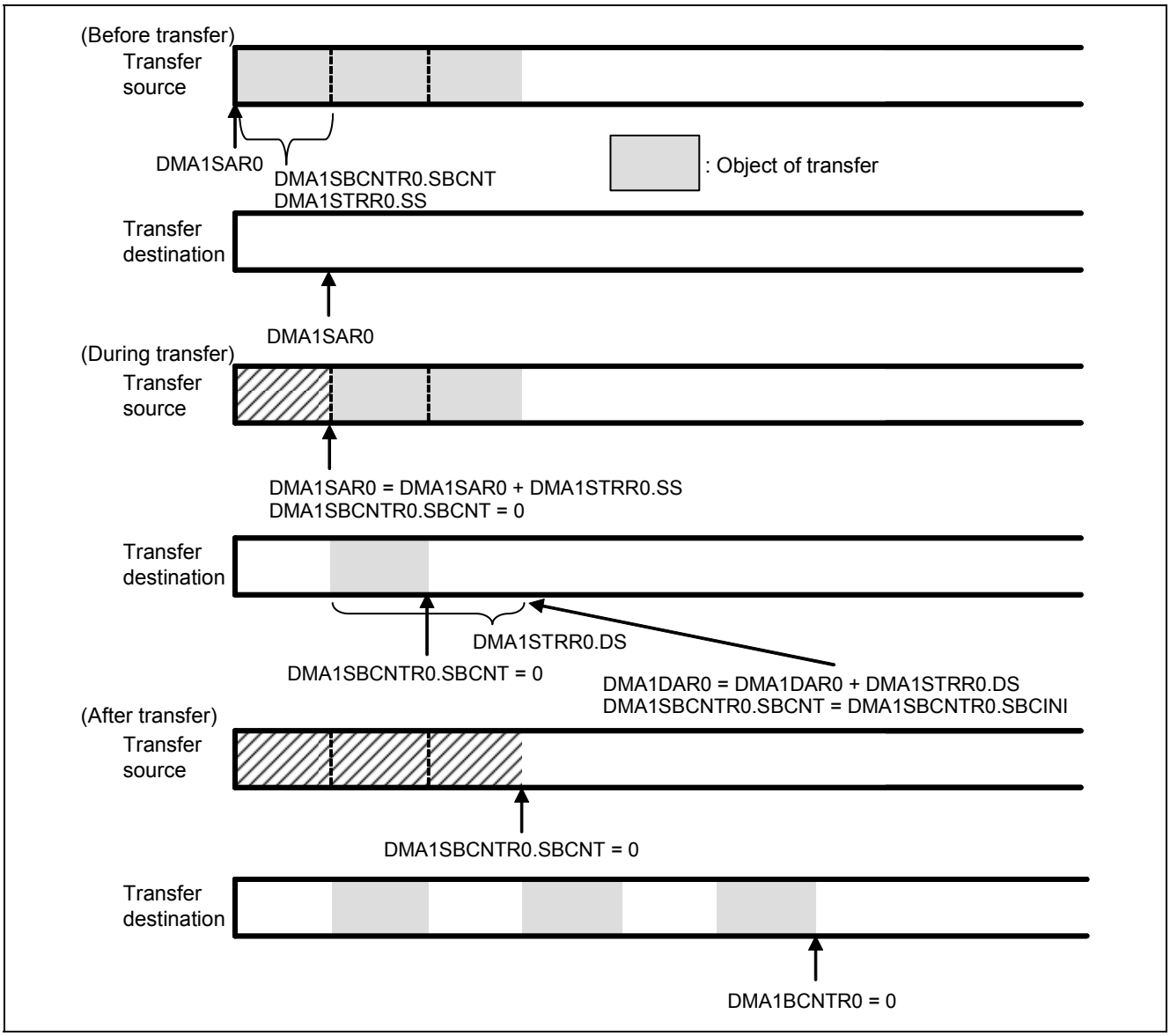


Figure 3.4 Scatter Transfer

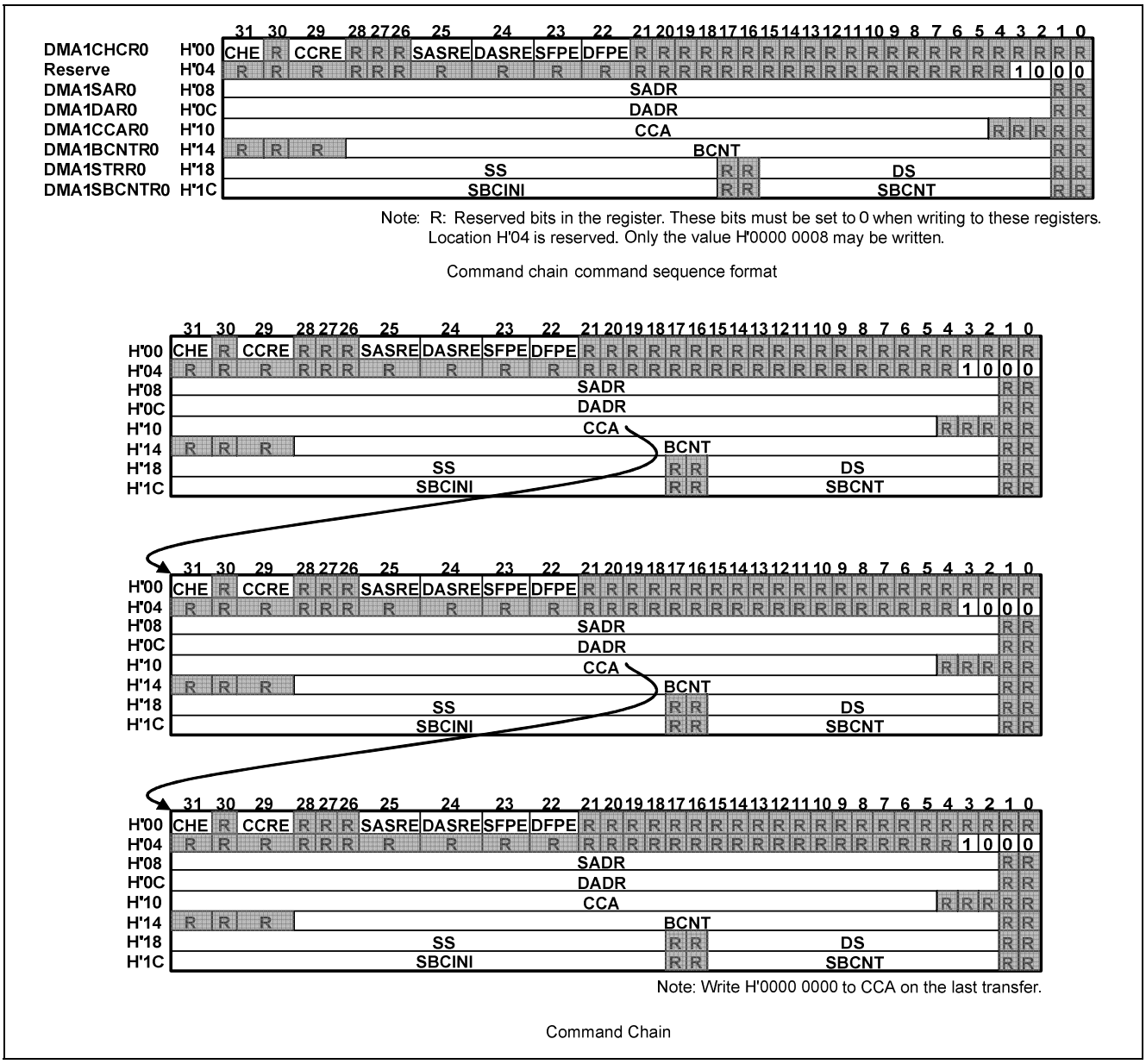


Figure 3.5 Command Chain Operation

3.1.3 Sample Program

The sample program starts DMAC1 channel 0 or 2 and performs bidirectional data transfers between internal RAM and external memory. Note that channel 2 does not have the hardware functions required for command chain operation.

Here, it is also possible to select whether cache coherence with external memory is guaranteed during the DMA transfers by performing flush/purge operations. When the flush/purge operations are controlled in software and a flush/purge is not performed, it is possible for inconsistencies to occur between the transfer source data and transfer destination data. See section 7, Coherence Control Between Cache and External Memory, for details.

Table 3.2 lists the specifications for this program.

Table3.2 Sample Program Specifications

Item	Specification
Channels used	• Channel 0 • Channel 2
Memory	• OL memory (internal memory) • DDR3-SDRAM (external memory)
Transfer direction	• OL memory → DDR3-SDRAM • DDR3-SDRAM → OL memory
Transfer data length	• Channel 0: 4-byte units • Channel 2: 1-byte units
Transfer data size	• Channel 0: 32 bytes • Channel 4: 1 / 2 / 4 / 8 / 32 bytes
Number of transfers	• Channel 0: 4 times • Channel 2: Calculated from the transfer data size
Address mode	• Dual addressing modes
Data transfers	• Channel 0 — Continuous area transfer — Scatter transfer — Gather transfer — Stride transfer Two transfers are performed using a command chain • Channel 2 — Continuous area transfer
Priority	• Fixed channel priority mode
Command chain	• Channel 0: Supported • Channel 2: No hardware functions
Interrupt requests	• An interrupt is issued to the CPU on transfer complete or if an address error occurs.
Coherence control between cache and external memory	• Copy back mode • Operand cache and secondary cache enabled • Cache flush/purge operations controlled in software (On (controlled)/off (not controlled) can be selected from a menu.) Note: If cache coherence is not controlled in copy back mode, it is possible for the operand cache and external memory to become inconsistent. See section 7, Coherence Control Between Cache and External Memory, for details.

3.1.4 Sample Program Register Settings

The following tables list the functions of the registers used by the sample program.

Two channel 0 transfers are performed using a command chain.

Table 3.3 DMAC1 Register Settings (Settings common to both channels)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA operation register (DMA1OR)	H'FEA0 0010	R/W	32	H'8000 0000	• DMA start/stop — During initialization: DMA1E = 1, DMA start.
				H'0000 0000	• DMA start/stop — At DMA transfer complete: DMA1E = 0, DMA stop.

Table 3.4 DMAC1 Channel 0 Register Initial Settings

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA1 source address register 0 (DMA1SAR0)	H'FEA0 0020	R/W	32	*1	• Specifies the transfer source start address
DMA1 destination address register 0 (DMA1DAR0)	H'FEA0 0028	R/W	32	*1	• Specifies the transfer destination start address
DMA1 byte count register 0 (DMA1BCNTR0)	H'FEA0 0030	R/W	32	*1	• Specifies the transfer byte count
DMA1 stride count register 0 (DMA1SBCNTR0)	H'FEA0 0034	R/W	32	*1	• Initial value setting for the data transfer byte count transferred as a single unit during stride and gather/scatter transfers. — Initial stride counter bits [32:16] = SBCINI — Stride counter bits [15:0] = SBCNT Note: The specified address is in 4-byte units.
DMA1 stride register 0 (DMA1STRR0)	H'FEA0 0038	R/W	32	*1	• Specifies the stride width for the transfer source address bits [32:16] = SS • Specifies the stride width for the transfer destination address bits [15:0] = DS Note: The specified addresses are all in 4-byte units.
DMA1 command chain register 0 (DMA1CCAR0)	H'FEA0 0040	R/W	32	H'E500 E100	• Specifies the command sequence address for the first command chain (command chain 1) Note: CCA must be set to H'0000 0000 for the last command sequence.

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA1 channel control register 0 (DMA1CHCR0)	H'FEA0 0048	R/W	32	H'A000 0000	• Specifies the DMA transfer enabled/disabled state, the command chain enabled/disabled state, and the transfer source/transfer destination address stride register enable state. CHE (bit 31) = H'1: DMA transfers enabled CCRE (bit 29) = H'1: Command chain enabled
DMA1 channel status register 0 (DMA1CHSR0)	H'FEA0 004C	R/(W)	32	*1	• Indicates the states of the following: — Transfer source transfer error interrupt — Transfer destination transfer error interrupt — Transfer source transfer error flag — Transfer destination transfer error flag — DMA transfer complete interrupt — DMA transfer complete flag

Note: 1. Registers other than DMA1CCAR0 and DMA1CHCR0 are set by command chain command sequence formats. See section 3.1.5 (1) Command Chains, for details on command chain command sequence formats.

The following tables list the register setting values set in the command chain addresses (H'E500 E100 to H'E500 E11C).

Table 3.5 DMAC1 Register Settings 1 (Channel 0 command chain address 1)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA1 source address register 0 (DMA1SAR0)	H'FEA0 0020	R/W	32	H'1400 E000	• Specifies the start address for the transfer source When OL memory is specified (DMA1DAR0 specifies DDR3)
				H'0800 0000	• Specifies the start address for the transfer source When DDR3-SDRAM is specified (DMA1DAR0 specifies OL memory)
DMA1 destination address register 0 (DMA1DAR0)	H'FEA0 0028	R/W	32	H'1400 E000	• Specifies the start address for the transfer destination When OL memory is specified (DMA1SAR0 specifies DDR3)
				H'0800 0000	• Specifies the start address for the transfer destination When DDR3-SDRAM is specified (DMA1SAR0 specifies OL memory)
DMA1 byte count register 0 (DMA1BCNTR0)	H'FEA0 0030	R/W	32	H'0000 0040	• Specifies the transfer byte count. 64 bytes Note: The transfer size is 4-byte units.
DMA1 stride count register 0 (DMA1SBCNTR0)	H'FEA0 0034	R/W	32	H'0000 0000	• Continuous area transfer SBCINI = 0, SBCNT = 0
				H'0004 0004	• The transfer size for stride and scatter/gather transfers is 4 bytes. SBCINI = 4, SBCNT = 4
				H'0008 0008	• The transfer size for stride and scatter/gather transfers is 8 bytes. SBCINI = 8, SBCNT = 8
				H'0010 0010	• The transfer size for stride and scatter/gather transfers is 16 bytes. SBCINI = 16, SBCNT = 16
				H'0020 0020	• The transfer size for stride and scatter/gather transfers is 32 bytes. SBCINI = 32, SBCNT = 32

Table 3.6 DMAC1 Register Settings 2 (Channel 0 command chain address 1)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA1 stride register 0 (DMA1STRR0)	H'FEA0 0038	R/W	32	H'0000 0000	• Continuous area transfer SS = 0, DS = 0
				H'0008 0008	• The transfer size for stride transfer is 4 bytes. SS = 8, DS = 8
				H'0004 0008	• The transfer size for scatter transfer is 4 bytes. SS = 4, DS = 8
				H'0008 0004	• The transfer size for gather transfer is 4 bytes. SS = 8, DS = 4
				H'0010 0010	• The transfer size for stride transfer is 8 bytes. SS = 16, DS = 16
				H'0008 0010	• The transfer size for scatter transfer is 8 bytes. SS = 8, DS = 16
				H'0010 0008	• The transfer size for gather transfer is 8 bytes. SS = 16, DS = 8
				H'0020 0020	• The transfer size for stride transfer is 16 bytes. SS = 32, DS = 32
				H'0010 0020	• The transfer size for scatter transfer is 16 bytes. SS = 16, DS = 32
				H'0020 0010	• The transfer size for gather transfer is 16 bytes. SS = 32, DS = 16
				H'0040 0040	• The transfer size for stride transfer is 32 bytes. SS = 64, DS = 64
				H'0020 0040	• The transfer size for scatter transfer is 32 bytes. SS = 32, DS = 64
				H'0040 0020	• The transfer size for gather transfer is 32 bytes. SS = 64, DS = 32
DMA1 command chain register 0 (DMA1CCAR0)	H'FEA0 0040	R/W	32	H'E500 E120	• Specifies the address for the next command chain command sequence (command chain 2)
DMA1 channel control register 0 (DMA1CHCR0)	H'FEA0 0048	R/W	32	H'A000 0000	• Continuous area transfer CHE = 1, CCRE = 1
				H'A300 0000	• The transfer size for stride and scatter/gather transfers is 4 bytes. CHE = 1, CCRE = 1, SARE = 1, DARE = 1
DMA1 channel status register 0 (DMA1CHSR0)	H'FEA0 004C	R/(W)	32	H'0000 0000	• The interrupts are not used.

The following tables list the register setting values set in the command chain addresses 2 (H'E500 E120 to H'E500 E13C).

Table 3.7 DMAC1 Register Settings 1 (Channel 0 command chain address 2)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA1 source address register 0 (DMA1SAR0)	H'1EA0_00020	R/W	32	H'1400 E000	• Specifies the start address for the transfer source When OL memory is specified (DMA1DAR0 specifies DDR3)
				H'0800 0000	• Specifies the start address for the transfer source When DDR3-SDRAM is specified (DMA1DAR0 specifies OL memory)
DMA1 destination address register 0 (DMA1DAR0)	H'1EA0_0028	R/W	32	H'1400 E000	• Specifies the start address for the transfer destination When OL memory is specified (DMA1SAR0 specifies DDR3)
				H'0800 0000	• Specifies the start address for the transfer destination When DDR3-SDRAM is specified (DMA1SAR0 specifies OL memory)
DMA1 byte count register 0 (DMA1BCNTR0)	H'1EA0_0030	R/W	32	H'0000 0040	• Specifies the transfer byte count. 64 bytes Note: The transfer size is 4-byte units.
DMA1 stride count register 0 (DMA1SBCNTR0)	H'1EA0_0034	R/W	32	H'0000 0000	• Continuous area transfer SBCINI = 0, SBCNT = 0
				H'0004 0004	• The transfer size for stride and scatter/gather transfers is 4 bytes. SBCINI = 4, SBCNT = 4
				H'0008 0008	• The transfer size for stride and scatter/gather transfers is 8 bytes. SBCINI = 8, SBCNT = 8
				H'0010 0010	• The transfer size for stride and scatter/gather transfers is 16 bytes. SBCINI = 16, SBCNT = 16
				H'0020 0020	• The transfer size for stride and scatter/gather transfers is 32 bytes. SBCINI = 32, SBCNT = 32

Table 3.8 DMAC1 Register Settings 2 (Channel 0 command chain address 2)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA1 stride register 0 (DMA1STRR0)	H'1EA0_0038	R/W	32	H'0000 0000	• Continuous area transfer SS = 0, DS = 0
				H'0008 0008	• The transfer size for stride transfer is 4 bytes. SS = 8, DS = 8
				H'0004 0008	• The transfer size for scatter transfer is 4 bytes. SS = 4, DS = 8
				H'0008 0004	• The transfer size for gather transfer is 4 bytes. SS = 8, DS = 4
				H'0010 0010	• The transfer size for stride transfer is 8 bytes. SS = 16, DS = 16
				H'0008 0010	• The transfer size for scatter transfer is 8 bytes. SS = 8, DS = 16
				H'0010 0008	• The transfer size for gather transfer is 8 bytes. SS = 16, DS = 8
				H'0020 0020	• The transfer size for stride transfer is 16 bytes. SS = 32, DS = 32
				H'0010 0020	• The transfer size for scatter transfer is 16 bytes. SS = 16, DS = 32
				H'0020 0010	• The transfer size for gather transfer is 16 bytes. SS = 32, DS = 16
				H'0040 0040	• The transfer size for stride transfer is 32 bytes. SS = 64, DS = 64
				H'0020 0040	• The transfer size for scatter transfer is 32 bytes. SS = 32, DS = 64
				H'0040 0020	• The transfer size for gather transfer is 32 bytes. SS = 64, DS = 32
DMA1 command chain register 0 (DMA1CCAR0)	H'1EA0_0040	R/W	32	H'0000 0000	• Specifies the address for the next command chain command sequence (command chain 2)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA1 channel control register 0 (DMA1CHCR0)	H'1EA0_0048	R/W	32	H'A000 0000	• Continuous area transfer CHE = 1, CCRE = 1
				H'A300 0000	• The transfer size for stride and scatter/gather transfers is 4 bytes. CHE = 1, CCRE = 1, SARE = 1, DARE = 1
DMA1 channel status register 0 (DMA1CHSR0)	H'1EA0_004C	R/(W)	32	H'0000 0000	• The interrupts are not used.

Table 3.9 DMAC1 Register Settings (Channel 2)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA1 source address register 2 (DMA1SAR2)	H'1EA0_00220	R/W	32	H'1400 E040	• Specifies the start address for the transfer source When OL memory is specified (DMA1DAR2 specifies DDR3)
				H'0800 0000	• Specifies the start address for the transfer source When DDR3-SDRAM is specified (DMA1DAR2 specifies OL memory)
DMA1 destination address register 2 (DMA1DAR2)	H'1EA0_0228	R/W	32	H'1400 E040	• Specifies the start address for the transfer destination When OL memory is specified (DMA1SAR2 specifies DDR3)
				H'0800 0000	• Specifies the start address for the transfer destination When DDR3-SDRAM is specified (DMA1SAR2 specifies OL memory)
DMA1 byte count register 2 (DMA1BCNTR2)	H'1EA0_0230	R/W	32	H'0000 0064	• Specifies the transfer byte count When the transfer size is 2 bytes or less, 100 bytes are transferred.
				H'0000 0080	• Specifies the transfer byte count — When the transfer size is 4 bytes or less, 128 bytes are transferred.

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA1 channel control register 2 (DMA1CHCR2)	H'1EA0_0248	R/W	32	H'0000 0000	• In the initial settings: CHE = 0
				H'8000 0000	• At DMA transfer start: CHE = 1
				H'0000 0000	• At DMA transfer complete or cancel: CHE = 0
DMA1 channel status register 2 (DMA1CHSR2)	H'1EA0_024C	R/(W)	32	H'0000 0000	• In the initial settings: TE = 0
				H'0000 0001	• At transfer complete: TE = 1 (automatically set to 1 at transfer complete)
DMA1 source transfer size register 2 (DMA1STRS2)	H'1EA0_0260	R/W	32	H'0000 0000	• Transfer source DMA transfer size — Byte unit
				H'0000 0001	• Transfer source DMA transfer size — Word units
				H'0000 0002	• Transfer source DMA transfer size — Longword units
				H'0000 0003	• Transfer source DMA transfer size — 8-byte units
				H'0000 0005	• Transfer source DMA transfer size — 32-byte units
DMA1 destination transfer size register 2 (DMA1DTRS2)	H'1EA0_0270	R/W	32	H'0000 0000	• Transfer source DMA transfer size — Byte unit
				H'0000 0001	• Transfer source DMA transfer size — Word units
				H'0000 0002	• Transfer source DMA transfer size — Longword units
				H'0000 0003	• Transfer source DMA transfer size — 8-byte units
				H'0000 0005	• Transfer source DMA transfer size — 32-byte units

Note: Registers not used by this program and bits that are not set remain set to their initial values.

3.1.5 Programming Notes

Keep the following points in mind when creating programs that use DMAC1.

(1) Command Chain

Figure 3.6 shows the command chain command sequence format as presented in figure 16.6 in the SH7786 Group Hardware Manual.

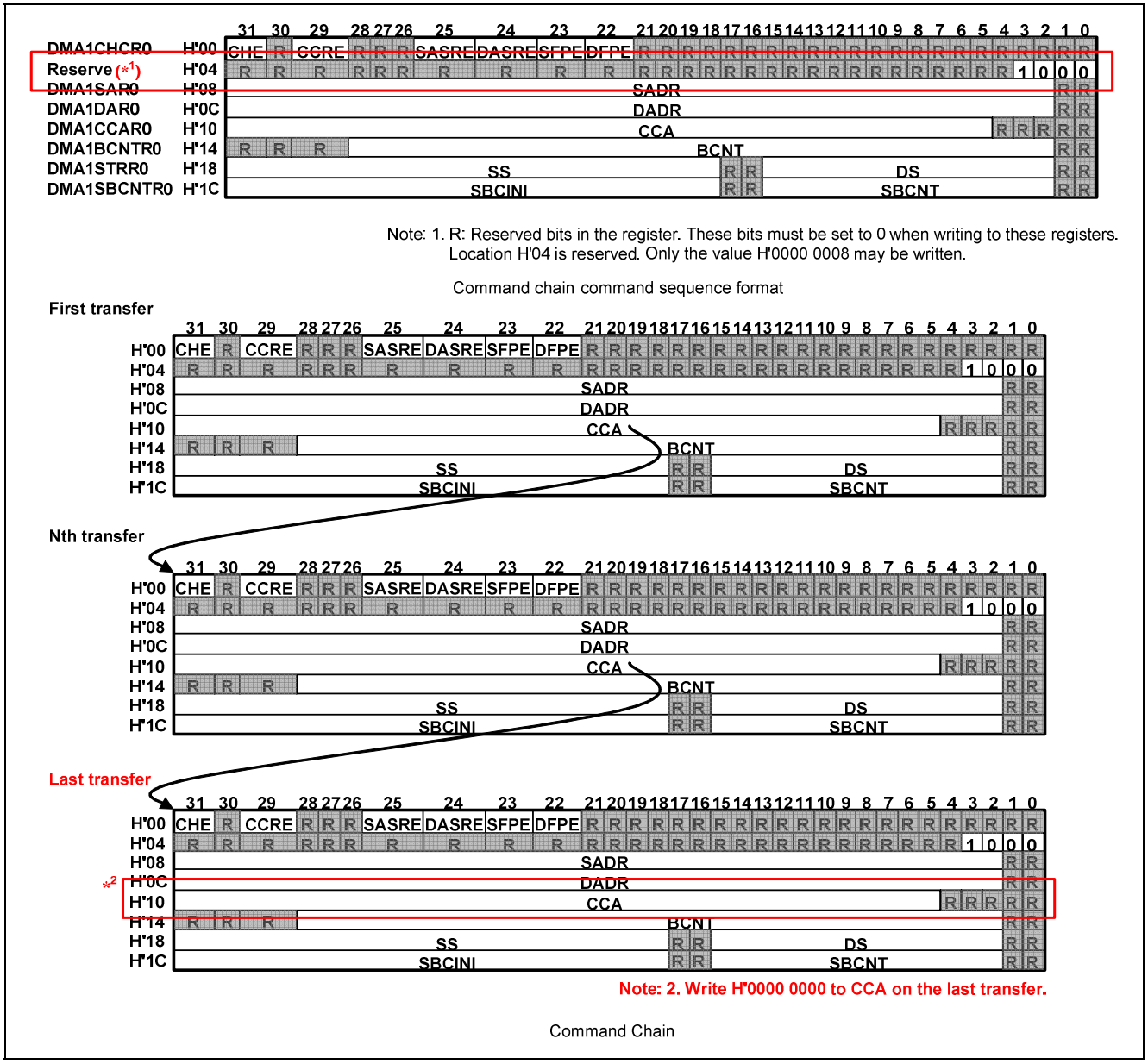


Figure 3.6 Command Chain Command Sequence Format

Notes: 1. Set address H'04 in the command chain command sequence to H'00000008.
2. CCA at H'10 in the last command sequence in the command chain must be set to H'00000000.
3. To terminate a command chain transfer, set the CCRE bit in the channel control register to 0 (disabled) at data transfer command completion.

4. HPB-DMAC Data Transfer Example

4.1 Application Example

This application note's sample program transfers data from a peripheral module to external memory, and from external memory to a peripheral module. Continuous transfer mode is used for these data transfers. Auto-request is used as the DMA transfer request. The transfer method is selected from a serial console using the built-in FIFO serial communications interface (SCIF channel 0).

4.1.1 Functions Used and Operation Overview

When an HPB-DMAC transfer request occurs, a transfer is started according to the determined channel priority and terminates the transfer when the transfer termination conditions are met. There are three transfer request modes: peripheral request, auto-request, and timer request.

Table 4.1 lists the HPB-DMAC module features and their descriptions and figure 4.1 presents a conceptual overview of this module.

Table 4.1 HPB-DMAC Features

Item	Description
Number of channels	• 14 channels (channels 00 to 13) — Channels 00 to 06: Any of SCIF0 to 5 and HSPI may be selected. — Channels 07 to 11: Any of SSI0 to 3, HAC0/1, SD0-1, and SD1-1 may be selected. — Channels 12 and 13: Either USB-FUNC0 or 1 may be selected.
Address space	• Physical address space
Transfer directions	• Peripheral module to memory (SuperHyway bus) • Memory (SuperHyway bus) to peripheral module
Transfer data length	• Peripheral module: 1, 2, or 4 bytes • Memory (SuperHyway bus): Set with the DMA control register
Transfer burst length	• 1 or 8 (Only channels 10 to 13 support a burst length of 8)
Maximum transfer count	• 16 M (16,777,216 transfers)
Addressing modes	• Dual addressing modes
Transfer requests	• Peripheral request, auto-request, and timer request
Transfer mode	• Simple transfer mode, continuous transfer mode
Transfer complete interrupt	• An interrupt is generated after the specified transfer count completes for one DMA information unit.

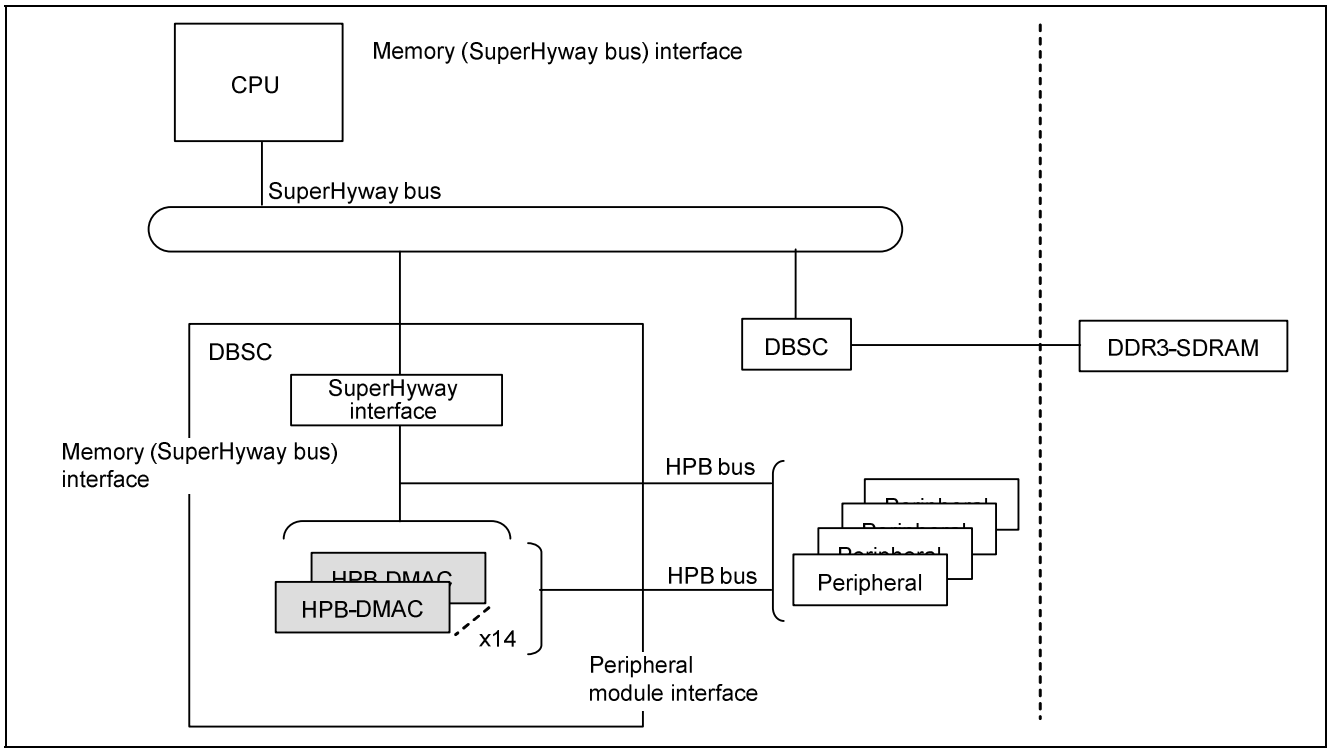


Figure 4.1 HPB-DMAC Conceptual Overview

4.1.2 Transfer Methods

The HPB-DMAC provides the following data transfer modes: simple transfer mode and continuous transfer mode.

- Simple transfer mode: the transfer completes when the number of transfers specified with the DMA transfer count register have completed.
- Continuous transfer mode: For all channels, if there is a next DMA transfer request (DNXT) when the number of transfers specified with the DMA transfer count register have completed, the HPB-DMAC acquires the next DMA information and performs that DMA transfer. If there is no next DMA transfer request (DNXT), it continues to wait until the next DMA transfer request is set up. The DMA command register (DCMDR) DQEND bit is used to terminate continuous transfer mode.

For detailed information on continuous transfer mode, see section 17.5.2, DMA Continuous Transfer Operation, in the SH7786 Group Hardware Manual.

4.1.3 Sample Program

The sample program performs bidirectional data transfers between a peripheral module and external memory. SCIF0 is used as the peripheral module and ASCII code data input to a serial console is DMA transferred to DDR3-SDRAM. Key data is echoed back from DDR3-SDRAM to SCIF0 at each key input using DMA transfers. This key input consists of 8 characters.

Here, it is also possible to select whether cache coherence with external memory is guaranteed during the DMA transfers by performing flush/purge operations. When the flush/purge operations are controlled in software and a flush/purge is not performed, it is possible for inconsistencies to occur between the transfer source data and transfer destination data. See section 7, Coherence Control Between Cache and External Memory, for details.

Table 4.2 lists the specifications for this program.

Table 4.2 Sample Program Specifications

Item	Specification
Channels used	• HPB DMAC0 (SCIF0)
Memory	• DDR3-SDRAM (external memory)
Transfer direction	• SCIF0 (peripheral module) → DDR3-SDRAM • DDR3-SDRAM → SCIF0 (peripheral module)
Transfer data length	• SCIF0: 1 byte (ordinary ASCII characters) • DDR3-SDRAM: 1 byte (PKMD is disabled)
Transfer burst length	• 1
Number of transfers	• 8 times (Ordinary ASCII characters are entered with 8 key strokes)
Address modes	• Dual address mode — DMA information 0 — DMA information 1
Data transfers	• Simple transfer • Consecutive transfer
Priority	• H'8 (default)
Transfer requests	• SCIF0 → DDR3-SDRAM: Peripheral module request • DDR3-SDRAM → SCIF0: Auto-request
Interrupt requests	• Generated when the number of transfers specified in a simple DMA information unit complete.
Coherence control between cache and external memory	• Copy back mode • Operand cache and secondary cache enabled • Cache flush/purge operations controlled in software (On (controlled)/off (not controlled) can be selected from a menu.) Note: If cache coherence is not controlled in copy back mode, it is possible for the operand cache and external memory to become inconsistent. See section 7, Coherence Control Between Cache and External Memory, for details.

4.1.4 Sample Program Register Settings

The following tables list the functions of the registers used by the sample program.

When a simple transfer is performed, only information plane 0 is used. When a continuous transfer is performed information planes 0 and 1 are used once each to perform the transfer.

Table 4.3 HPB-DMAC Register Settings (common to all channels)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA transfer complete interrupt display clear register (DINTCR)	H'FFC0 8810	R/WC1	32	H'0000 0001	• Clears the interrupt — At interrupt handling: DTEC0 = 1.
DMA transfer complete interrupt display enable register (DINTMR)	H'FFC0 8814	R	32	H'0000 0001	• An internal due to DMA transfer completion is output — At initialization: DTEM0 = 1.

Table 4.4 HPB-DMAC Register Settings (channel 0)

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA source address register 0 (DSAR0)	H'FFC0 8000	R/W	32	H'0800 0000	• Plane 0 transfer source start address Transfer direction: DDR3-SDRAM → SCIF SDRAM address specification
				H'1FEA 000C	• Plane 0 transfer source start address Transfer direction: SCIF → DDR3-SDRAM SCFTDR0 address specification
DMA destination address register 0 (DDAR0)	H'FFC08004	R/W	32	H'0800 0000	• Plane 0 transfer source start address Transfer direction: DDR3-SDRAM → SCIF SDRAM address specification
				H'1FEA 000C	• Plane 0 transfer source start address Transfer direction: SCIF → DDR3-SDRAM SCFTDR0 address specification

Register (Abbreviation)	Address	R/W	Size	Set Value	Operational Specification
DMA transfer count register 0 (DTCR0)	H'FFC08008	R/W	32	H'0000 0020	• Transfer count setting When a simple transfer is performed One byte × 64 times
				H'0000 0010	• Transfer count setting When a continuous transfer is performed One byte × 32 times
				H'0000 0008	• Transfer count setting When a simple transfer is performed One byte × 8 times
				H'0000 0004	• Transfer count setting When a continuous transfer is performed One byte × 4 times
DMA source address register 1 (DSAR1)	H'FFC0800C	R/W	32	H'0800 0010	• Plane 1 transfer source start address Transfer direction: DDR3-SDRAM → SCIF SDRAM address specification
				H'1FEA 000C	• Plane 1 transfer source start address Transfer direction: SCIF → DDR3-SDRAM SCFTDR0 specification
DMA destination address register 1 (DDAR1)	H'FFC08010	R/W	32	H'0800 0010	• Plane 1 transfer source start address Transfer direction: DDR3-SDRAM → SCIF SDRAM address specification
				H'1FEA 000C	• Plane 1 transfer source start address Transfer direction: SCIF → DDR3-SDRAM SCFTDR0 specification
DMA transfer count register 1 (DTCR1)	H'FFC08014	R/W	32	H'0000 0010	• Transfer count setting Continuous transfers only One byte × 32 times
				H'0000 0004	• Transfer count setting Continuous transfers only One byte × 4 times

Table 4.5 HPB-DMAC DMA Control Registers

Register (Abbreviation)	Address	Operational Specification: Bit Name	Set Value	Description
DMA control register (DCR)	H'FFC08028	CT (bit 18)	H'0	• Simple transfer (Continuous DMA transfers are not performed.)
			H'1	• Continuous transfer
		ACMD (bit 17)	H'0	• Simple transfer (Automatic continuous transfers are not performed.)
			H'1	• Automatic continuous transfer
		DIP (bit 16)	H'0	• The plane 1 DMA information page is used continuously.
			H'1	• The plane 2 DMA information page is used alternately.
		SMDL (bit 13)	H'0	• Continuous transfer When the transfer direction is DDR3-SDRAM → SCIF, memory is set as the transfer source module.
			H'1	• Continuous transfer When the transfer direction is SCIF → DDR3-SDRAM, peripheral (SCIF) is set as the transfer source module.
		SDRMD (bit[11:10])	H'1	• Continuous transfer The transfer source DMA request mode is set to auto-request.
		DMDL (bit 5)	H'0	• Continuous transfer When the transfer direction is DDR3-SDRAM → SCIF, memory is set as the transfer source module.
			H'1	• Continuous transfer When the transfer direction is SCIF → DDR3-SDRAM, peripheral (SCIF) is set as the transfer source module.

Table 4.6 HPB-DMAC DMA Command Register Settings

Register (Abbreviation)	Address	Operational Specification: Bit Name	Set Value	Description
DMA command register (DCMDR)	H'FFC0802C	DQEND (bit 2)	H'1	• At interrupt handling: DMA continuous transfer mode is terminated.
		DNXT (bit 1)	H'1	• At interrupt handling: The next DMA transfer is requested.
		DMEN (bit 0)	H'1	• During transfer processing: The DMA is started

Note: Registers not used by this program and bits that are not set remain set to their initial values.

4.1.5 Notes on Program Creation

Keep the following points in mind when creating programs that use the HPB-DMAC.

(1) DCR Register SWMD Bit

When the PKMD bit is invalid, the SWMD bit will also be invalid. In this case the access data size to DDR3-SDRAM from HPB-DMAC0 to 13 will be in 1-byte units for all channels.

Table 4.7 Memory SuperHyway Bus Access Size

		SuperHyway bus access size		
PKMD Bit in DCR Register	SWMD Bit in DCR Register	HPB-DMAC0 to HPB-DMAC6	HPB-DMAC7 to HPB-DMAC11	HPB-DMAC12, HPB-DMAC13
1	0	8 bytes	16 bytes	32 bytes
0	Invalid	1 byte		

(2) DSAR0/1 and DDAR0/1 Register Address Boundaries

The address boundary when the address is set as in note 1 in the SH7786 Hardware Manual is memory (DDR3-SDRAM) is similar to the situation described above in section 4.1.5 (1), namely, when the DCR register PKMD bit is invalid, the SWMD bit will also be invalid. In this case the address boundary will be a 4-byte boundary for all channels.

Also, the addresses following the start addresses set with the DSAR0/1 and DDAR0/1 registers are accessed in 1-byte units. (See section 4.1.5 (1))

Table 4.8 DSAR0/1 and DDAR0/1 Address Boundaries

		Address Boundary		
PKMD Bit in DCR Register	SWMD Bit in DCR Register	HPB-DMAC0 to HPB-DMAC6	HPB-DMAC7 to HPB-DMAC11	HPB-DMAC12, HPB-DMAC13
1	0	8-byte boundary	16-byte boundary	32-byte boundary
1	1	4-byte boundary		
0	Invalid	4-byte boundary		

(3) Automatic Continuous Transfers

When transferring data from DDR3-SDRAM to one of HPB-DMAC0 to 13 using automatic continuous transfers, there are cases where unexpected data may be transferred. This is because, when the size of the data being transferred is small, the next DMA transfer cycle may start between the point the CPU accepts notification of the transfer complete interrupt and it stops the next DMA request. Thus it is necessary to consider the data size when using automatic continuous transfers. The minimum data size is 32 bytes when using the AlphaProject AP-AH4AD-0A memory and transferring data from DDR3-SDRAM to an SCIF module using DMA transfers.

5. Sample Program Processing Sequences

This sample allows DMAC0, DMAC1, or HPB-DMAC to be selected from a menu displayed on a serial console and that module's operation to be confirmed. The following flowcharts show the common processing procedures and the DMAC processing procedures.

Note: This sample program's specifications stipulate that it allows various transfer methods to be selected from a menu selection screen output to a serial console and the corresponding DMA transfer is performed. See the Sample Program Register Settings section for each DMAC for details on the detailed setting values for the various registers.

5.1 Common Processing Procedures

The following figures show the flowcharts for the common processing procedures.

5.1.1 Main (main())

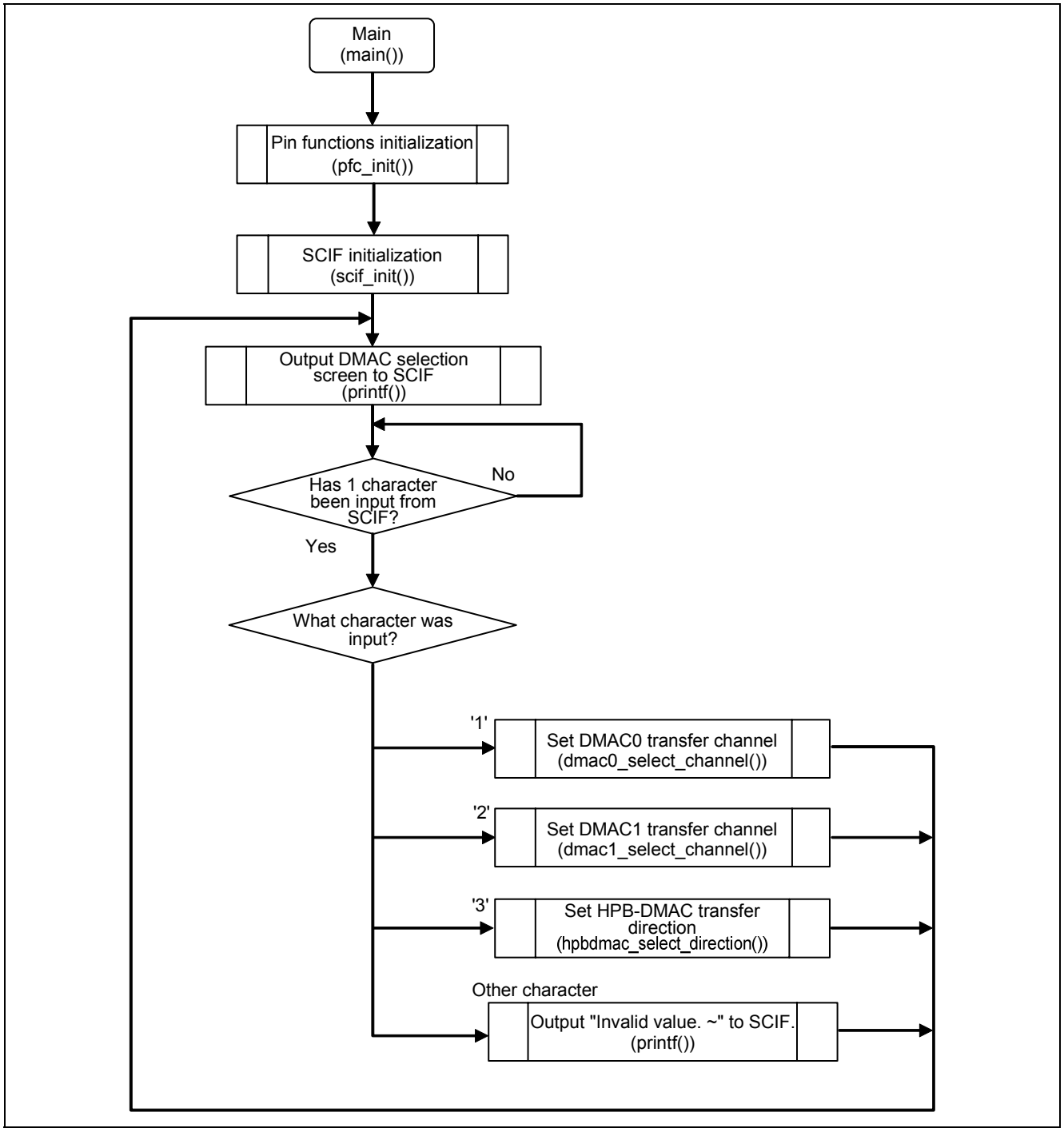


Figure 5.1 main() Flowchart

5.1.2 Pin Function Initialization (pfc_init())

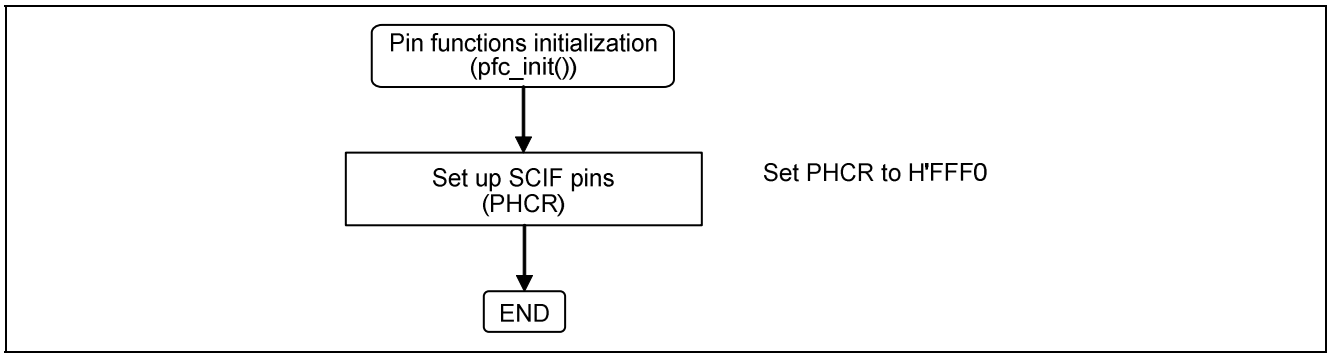


Figure 5.2 Pin Function Initialization Flowchart

5.1.3 Transfer Source/Transfer Destination Initialization (memory_init())

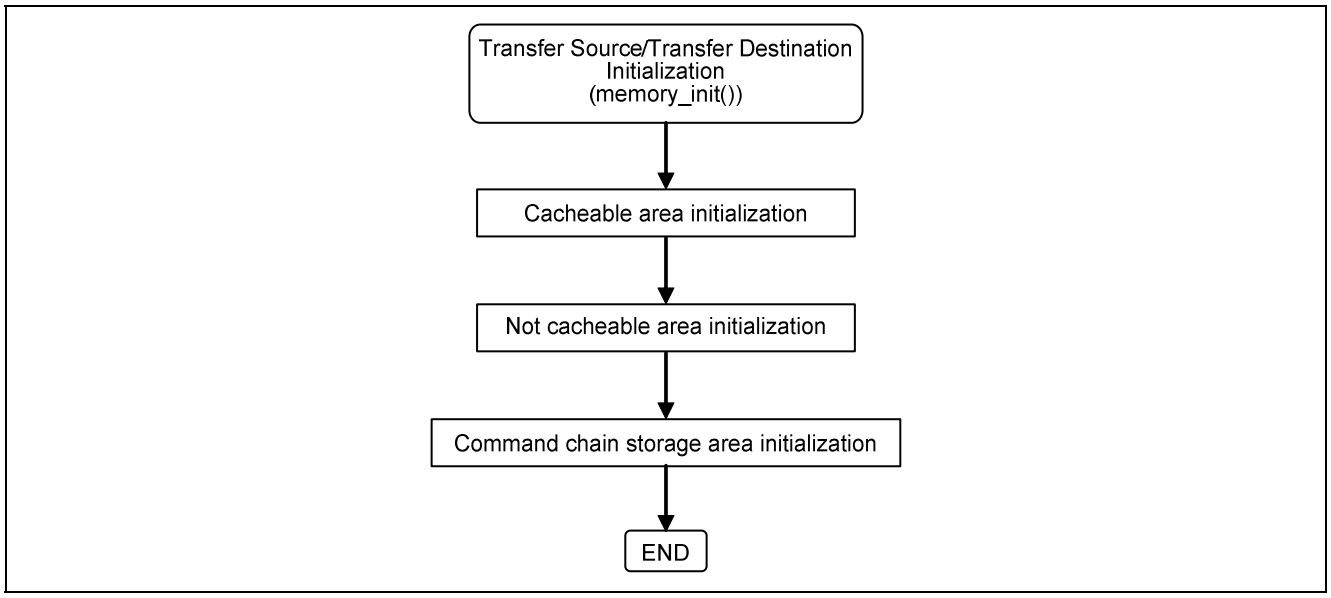


Figure 5.3 Transfer Source/Transfer Destination Initialization Flowchart

5.1.4 Transfer Result Data Display (print_result(), print_result_multi(), print_result_hpb())

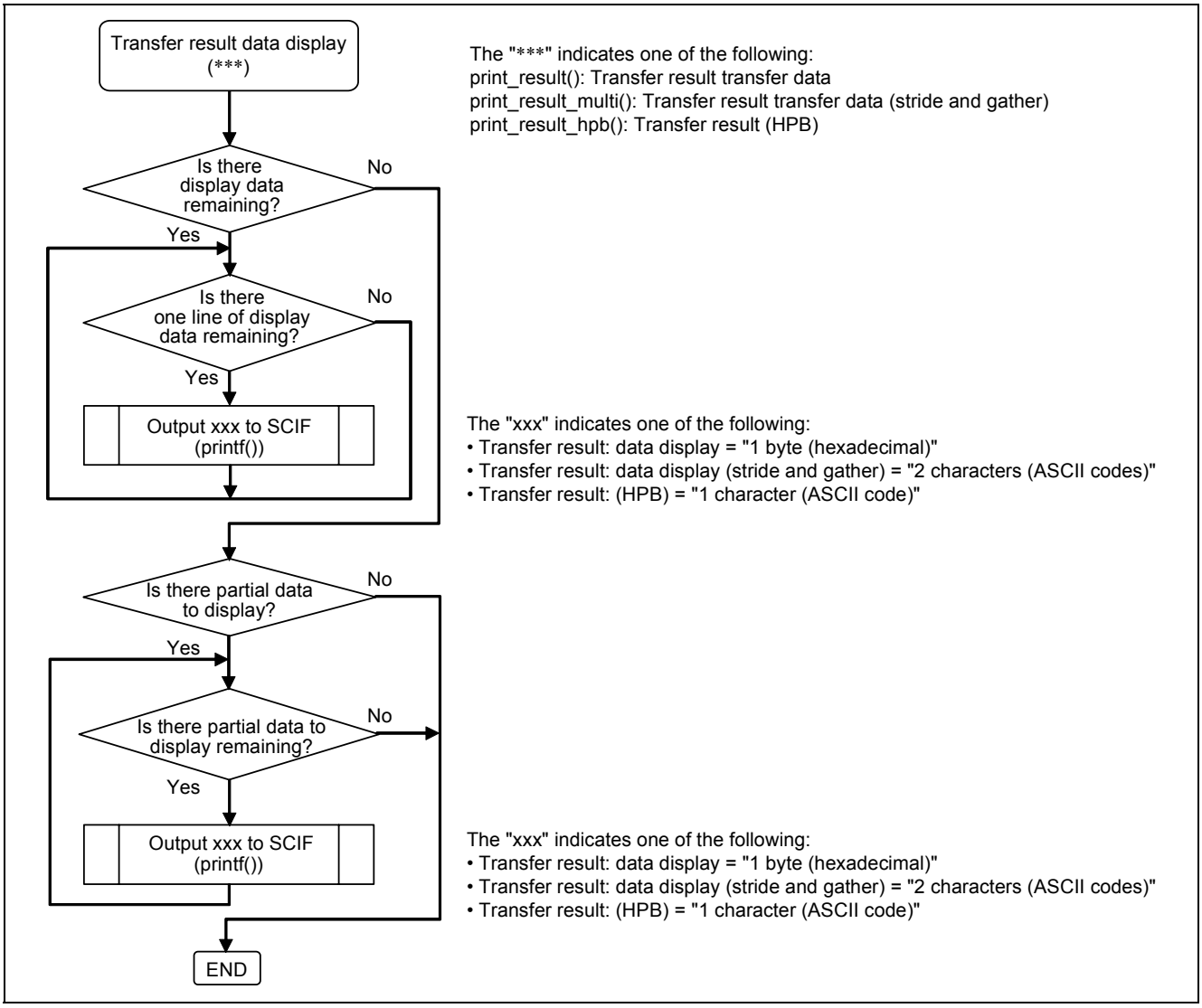


Figure 5.4 Transfer Result Data Display Flowchart

5.1.5 SCIF Initialization (scif_init())

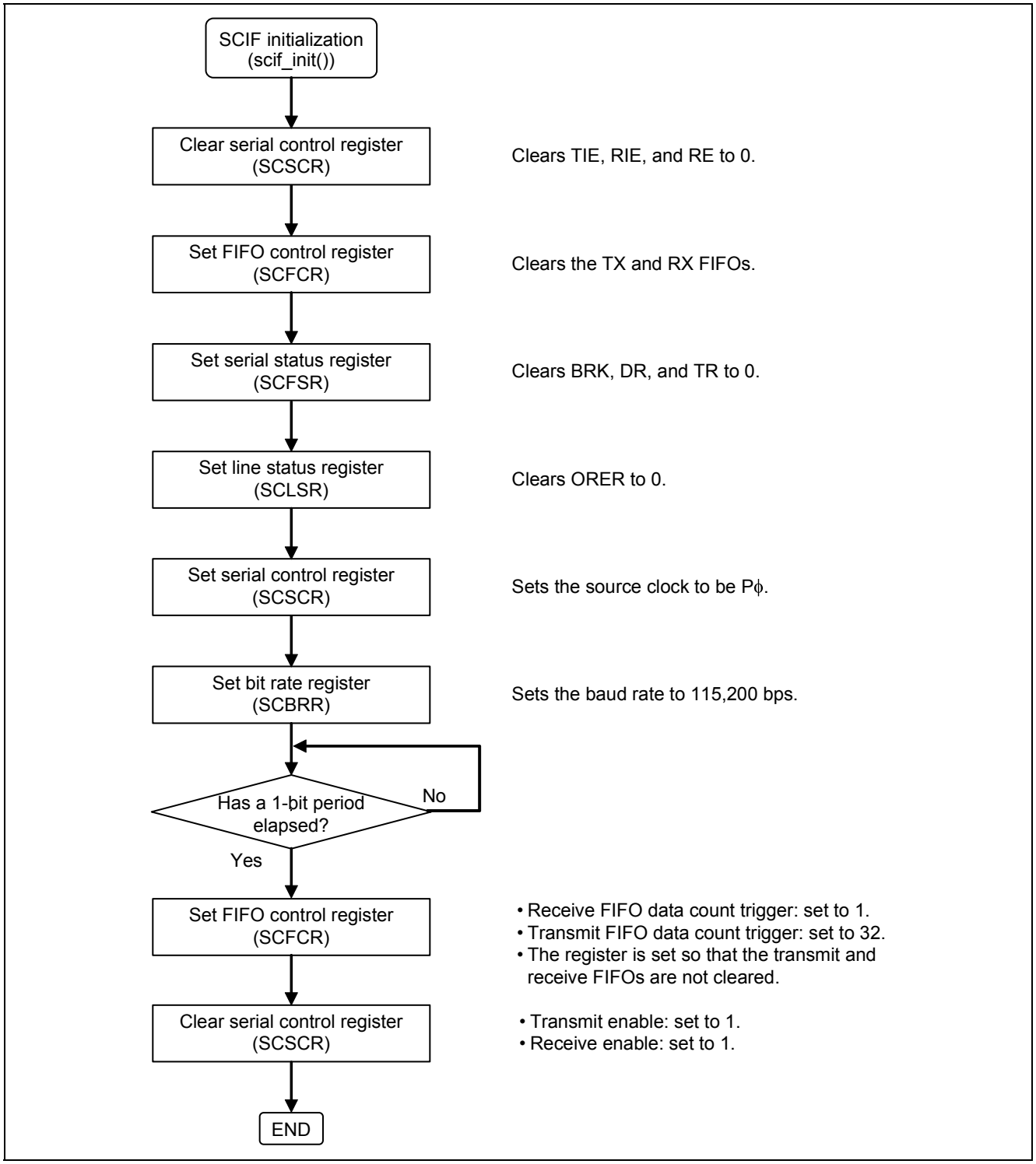


Figure 5.5 SCIF Initialization Flowchart

5.1.6 SCIF Transmit Data (scif_transmit_data())

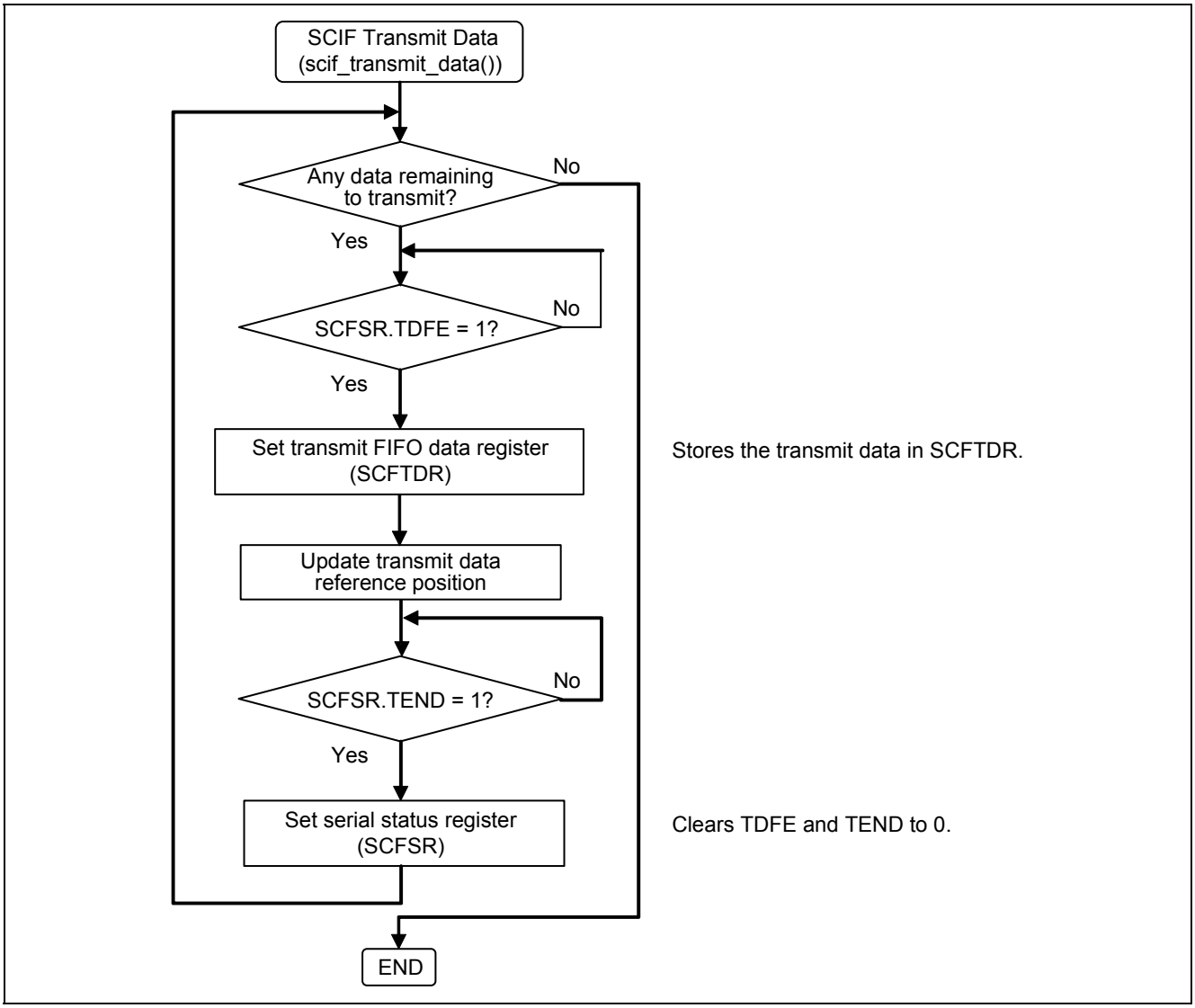


Figure 5.6 SCIF Transmit Data Flowchart

5.1.7 SCIF Transmit Data Byte (scif_transmit_data_byte())

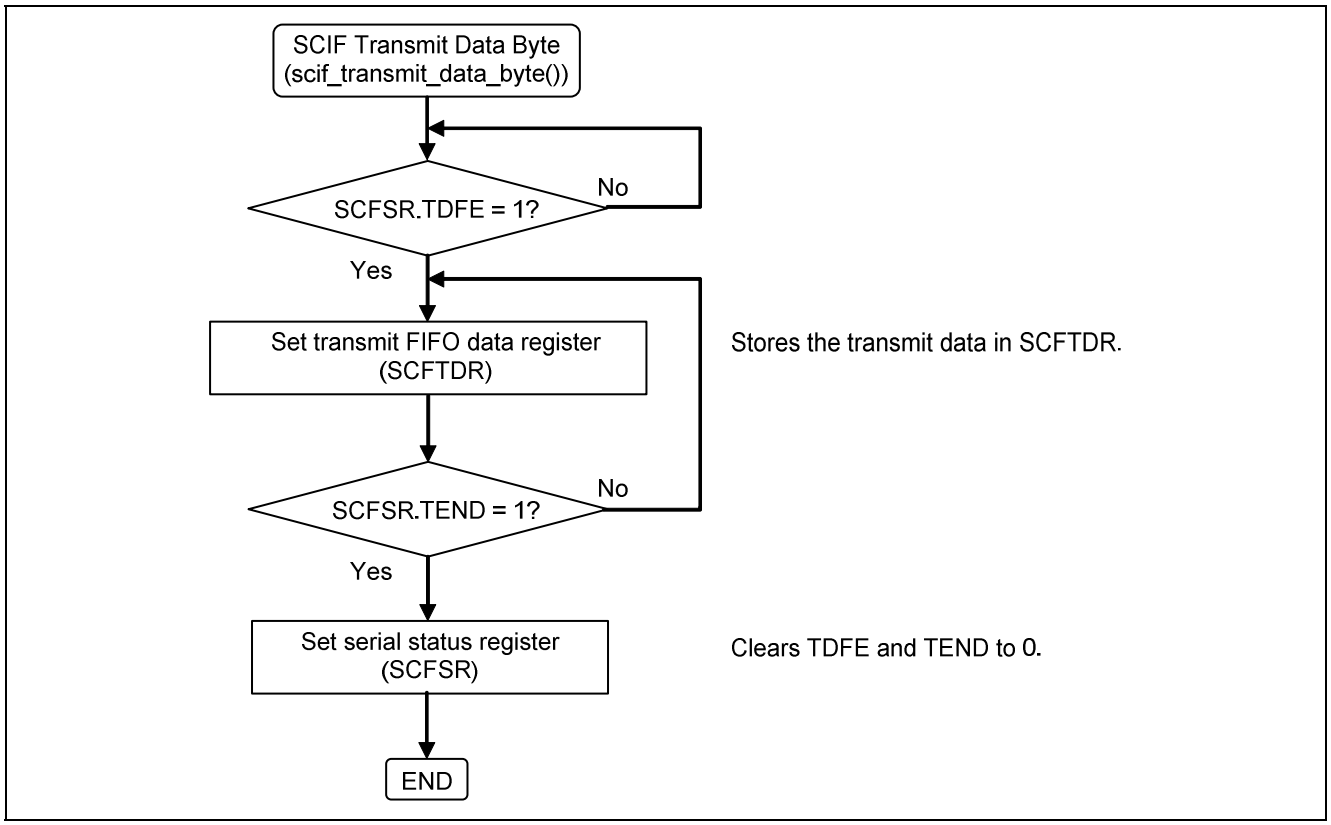
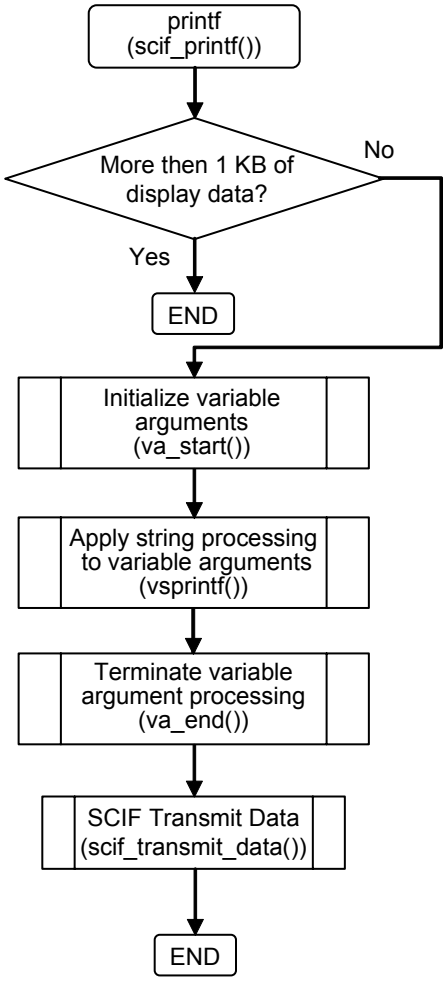


Figure 5.7 SCIF Transmit Data Byte Flowchart

5.1.8 SCIF printf (scif_printf())



Note: The vastart(), vsprintf(), and va_end() functions are library functions.
For details on these functions, see the User's Manual for the SuperH™ RISC Engine C/C++ Compiler, Assembler, and Optimizing Linkage Editor Compiler Package, version 9.0.1.

Figure 5.8 SCIF printf Flowchart

5.1.9 SCIF Receive Data Byte (scif_receive_data_byte())

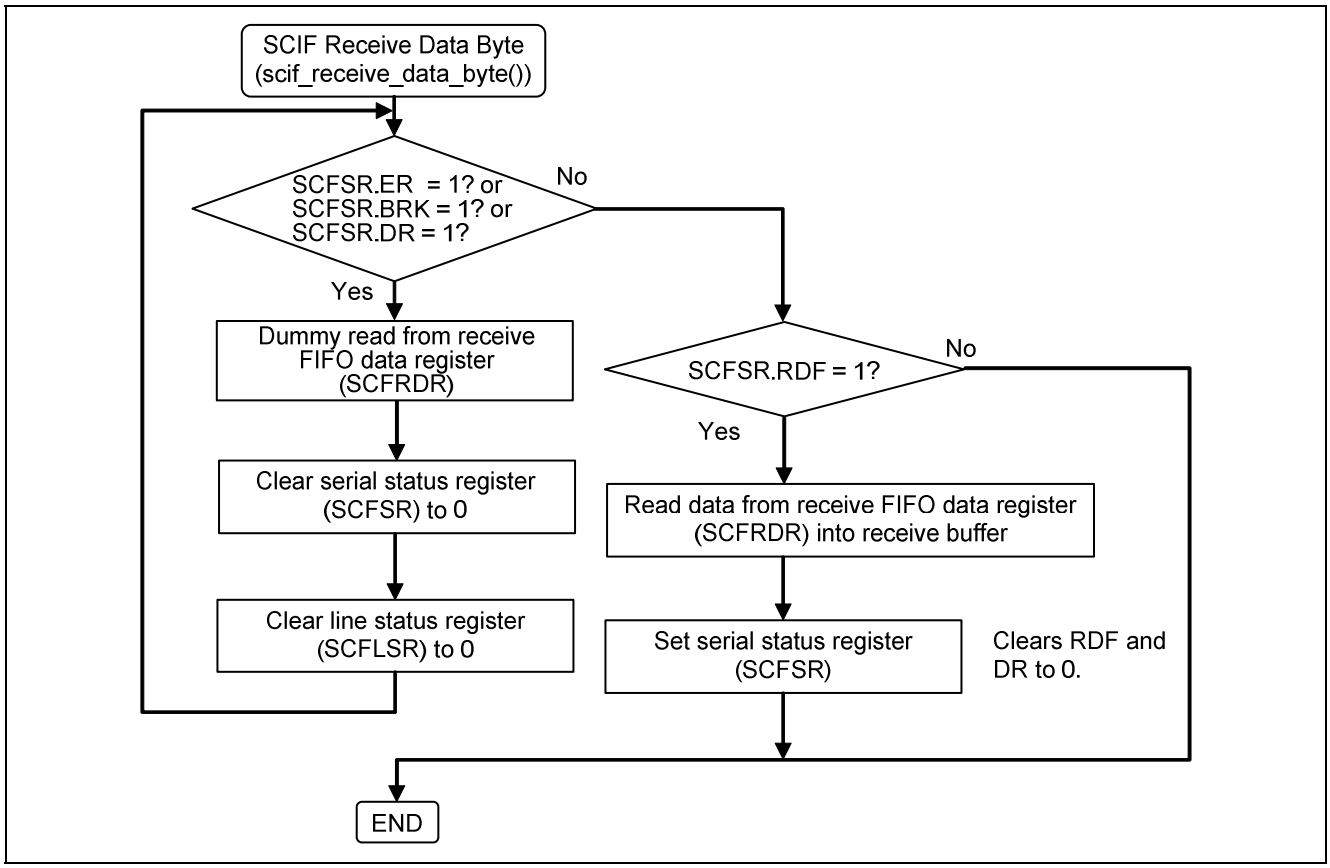


Figure 5.9 SCIF Receive Data Byte Flowchart

5.2 DMAC0 Processing Procedures

The following figures show the flowcharts for the DMAC0 processing procedures.

5.2.1 DMAC0 Select Channel (dmac0_select_channel())

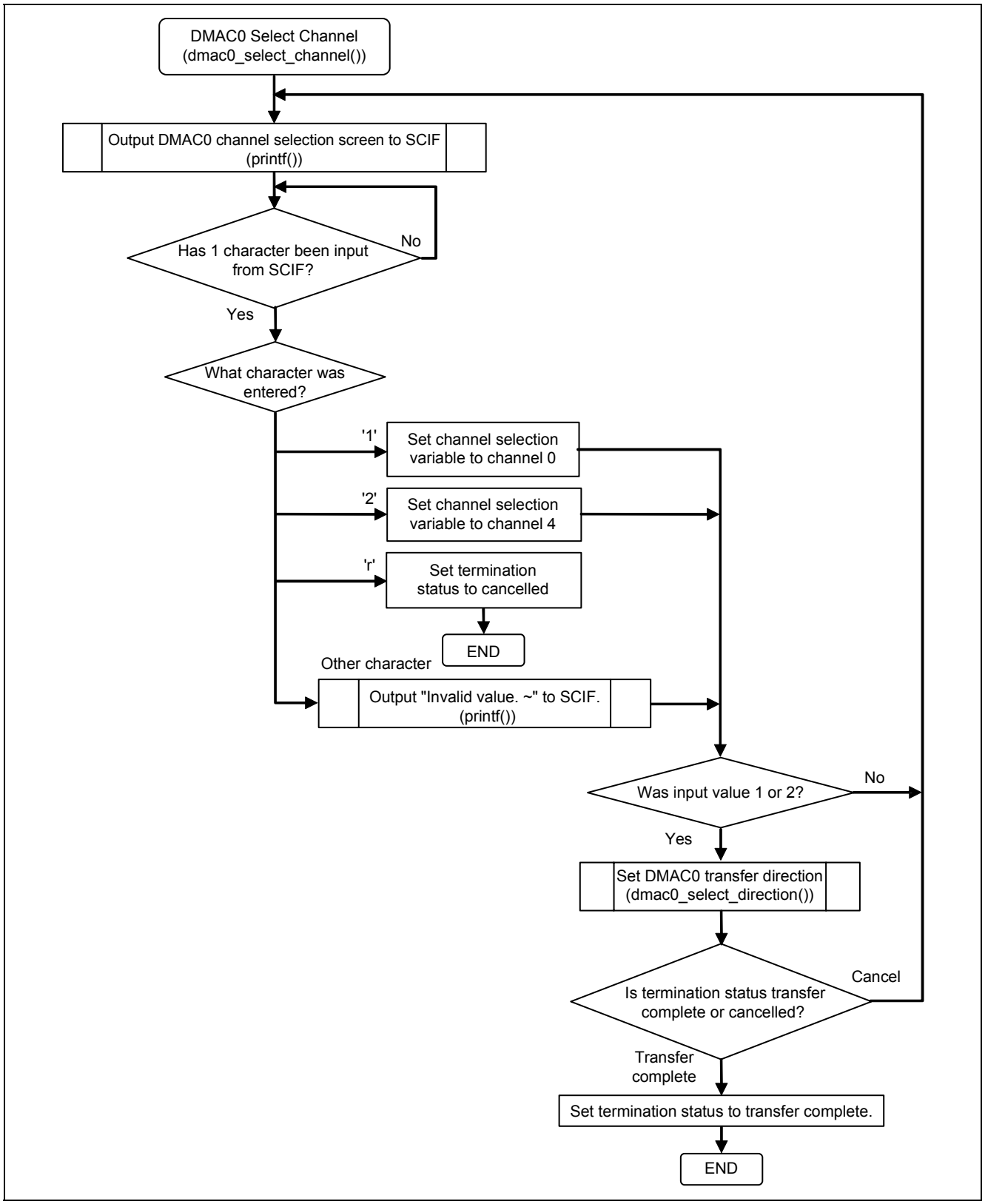


Figure 5.10 DMAC0 Select Channel Flowchart

5.2.2 DMAC0 Select Direction (dmac0_select_direction())

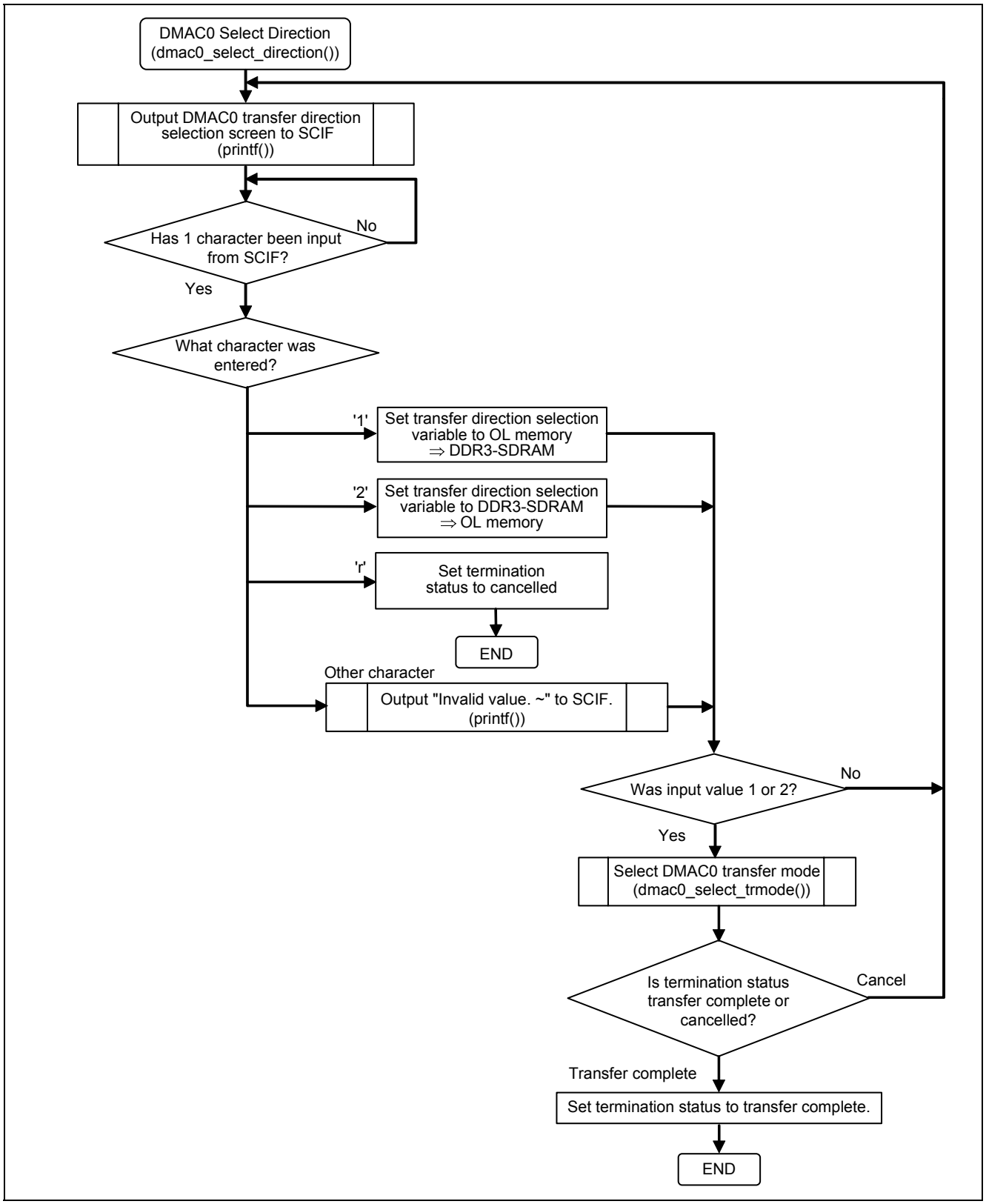


Figure 5.11 DMAC0 Select Direction Flowchart

5.2.3 DMAC0 Select Transfer Mode (dmac0_select_tmode())

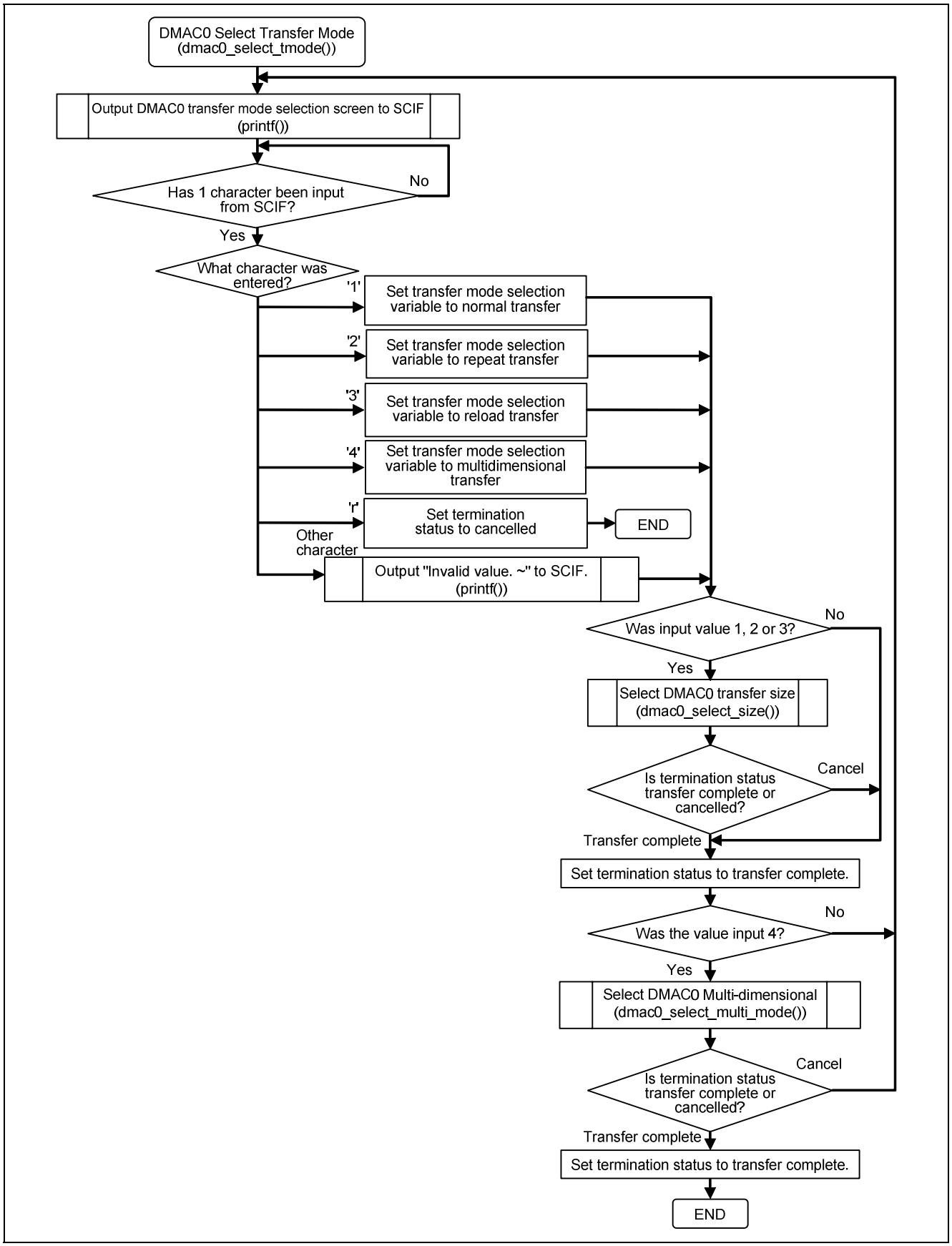


Figure 5.12 DMAC0 Transfer Mode Selection Flowchart

5.2.4 DMAC0 Select Multidimensional Mode (dmac0_select_multi-mode())

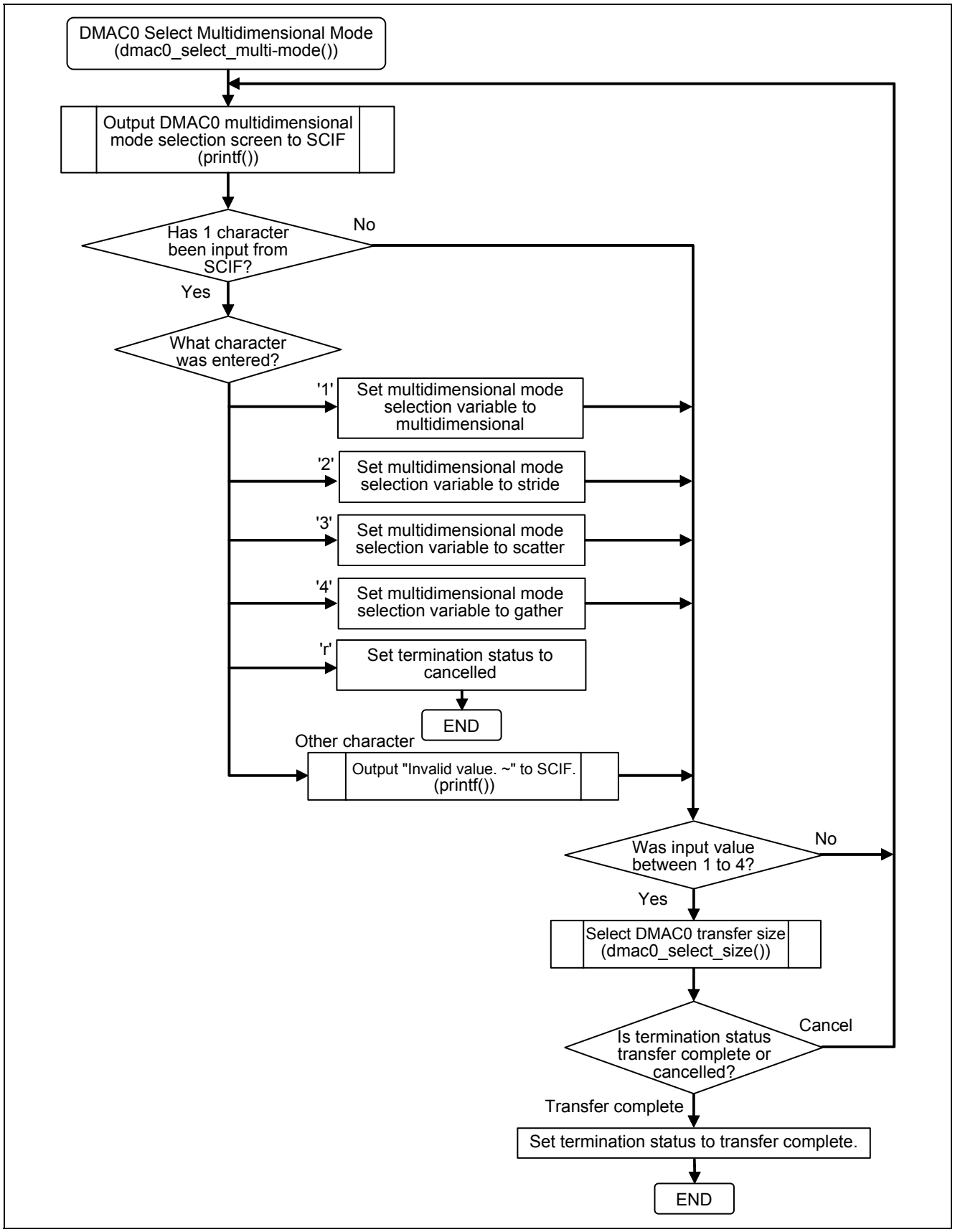


Figure 5.13 DMAC0 Select Multidimensional Mode Selection Flowchart

5.2.5 DMAC0 Select Transfer Size (dmac0_select_size())

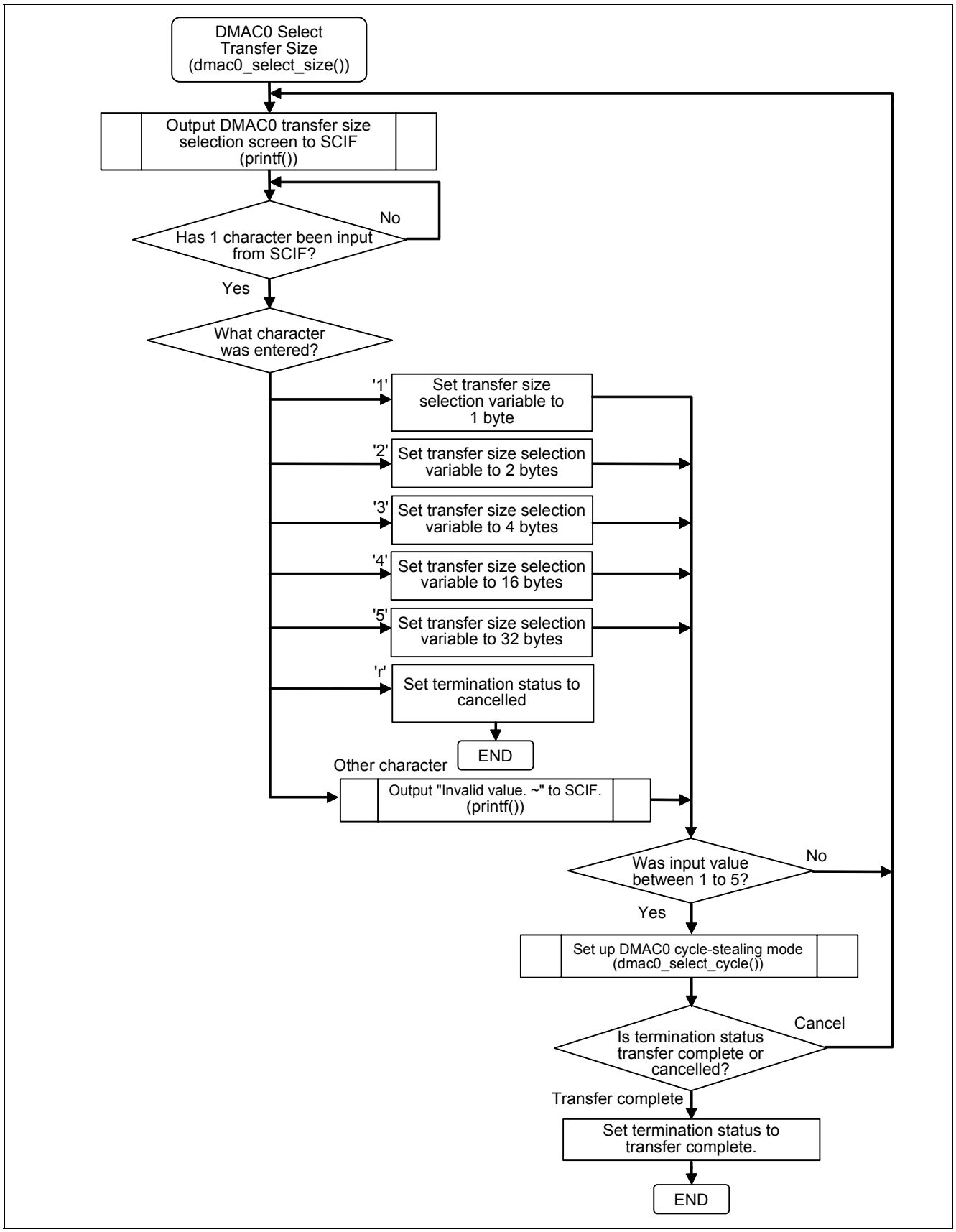


Figure 5.14 DMAC0 Transfer Size Selection Flowchart

5.2.6 DMAC0 Select Cycle-Stealing Mode Control (dmac0_select_cycle())

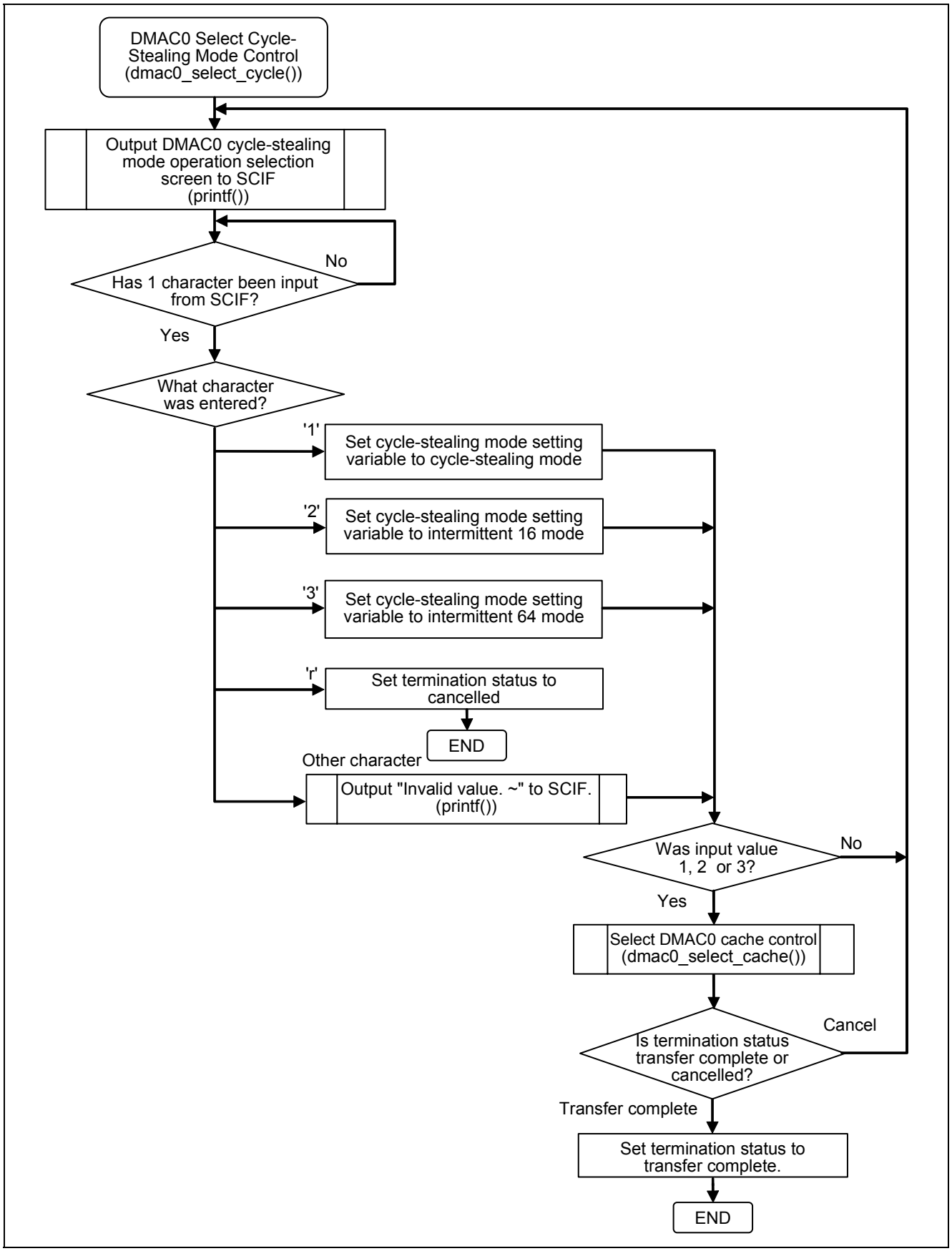


Figure 5.15 DMAC0 Cycle-Stealing Mode Control Selection Flowchart

5.2.7 DMAC0 Select Cache Control (dmac0_select_cache())

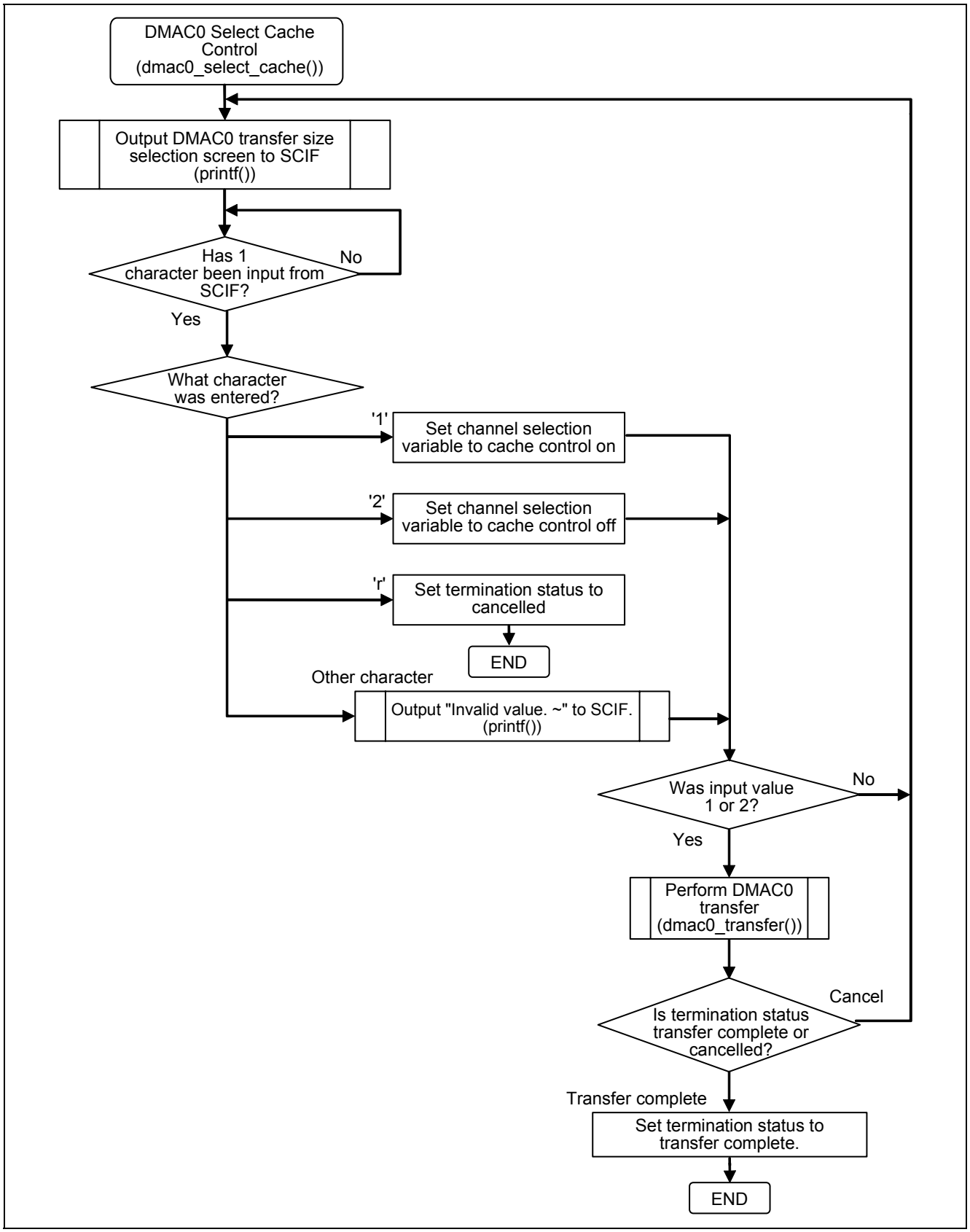


Figure 5.16 Cache Control Selection Flowchart

5.2.8 DMAC0 Transfer (dmac0_transfer())

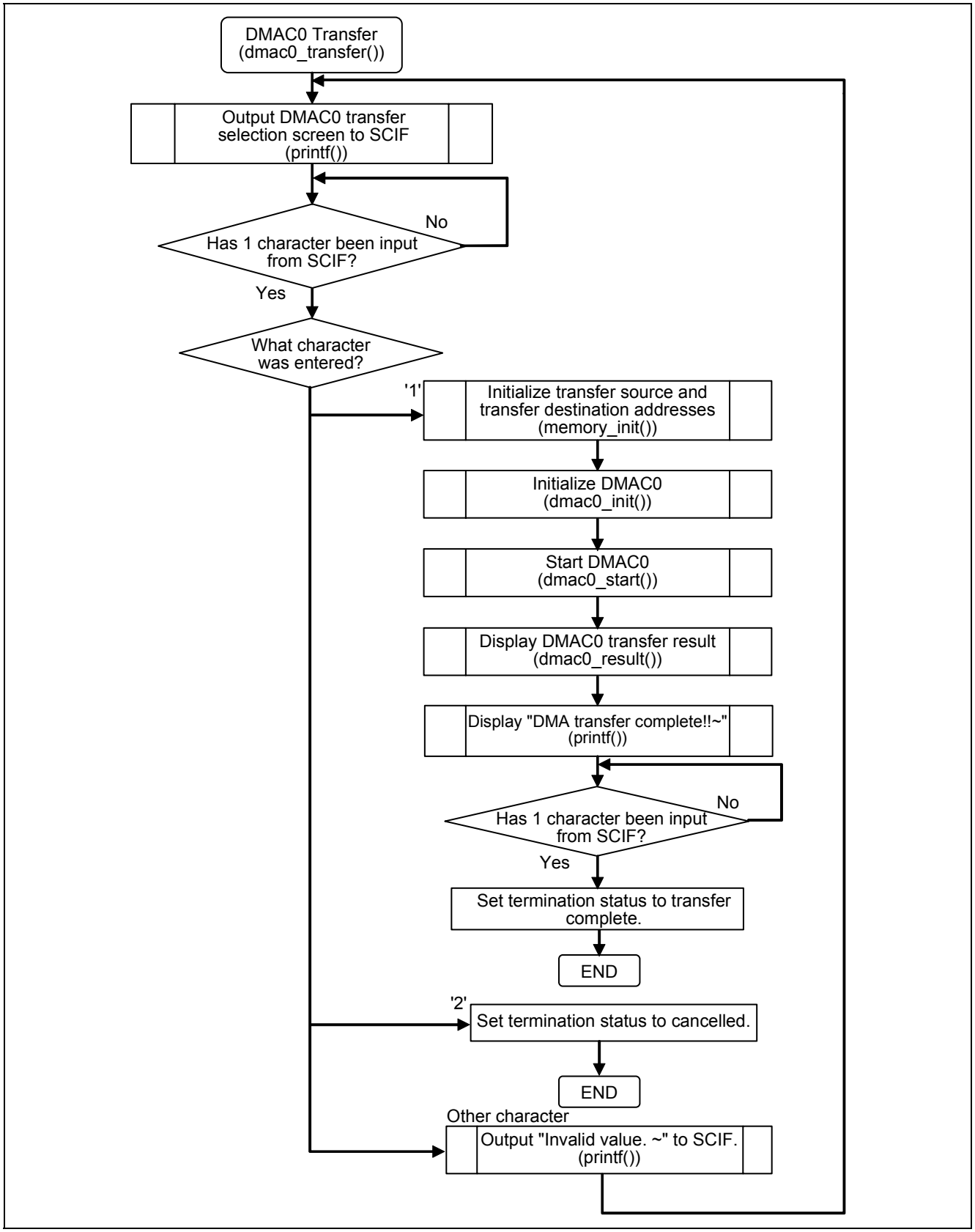


Figure 5.17 DMA Transfer Flowchart

5.2.9 DMAC0 Initialization (dmac0_init())

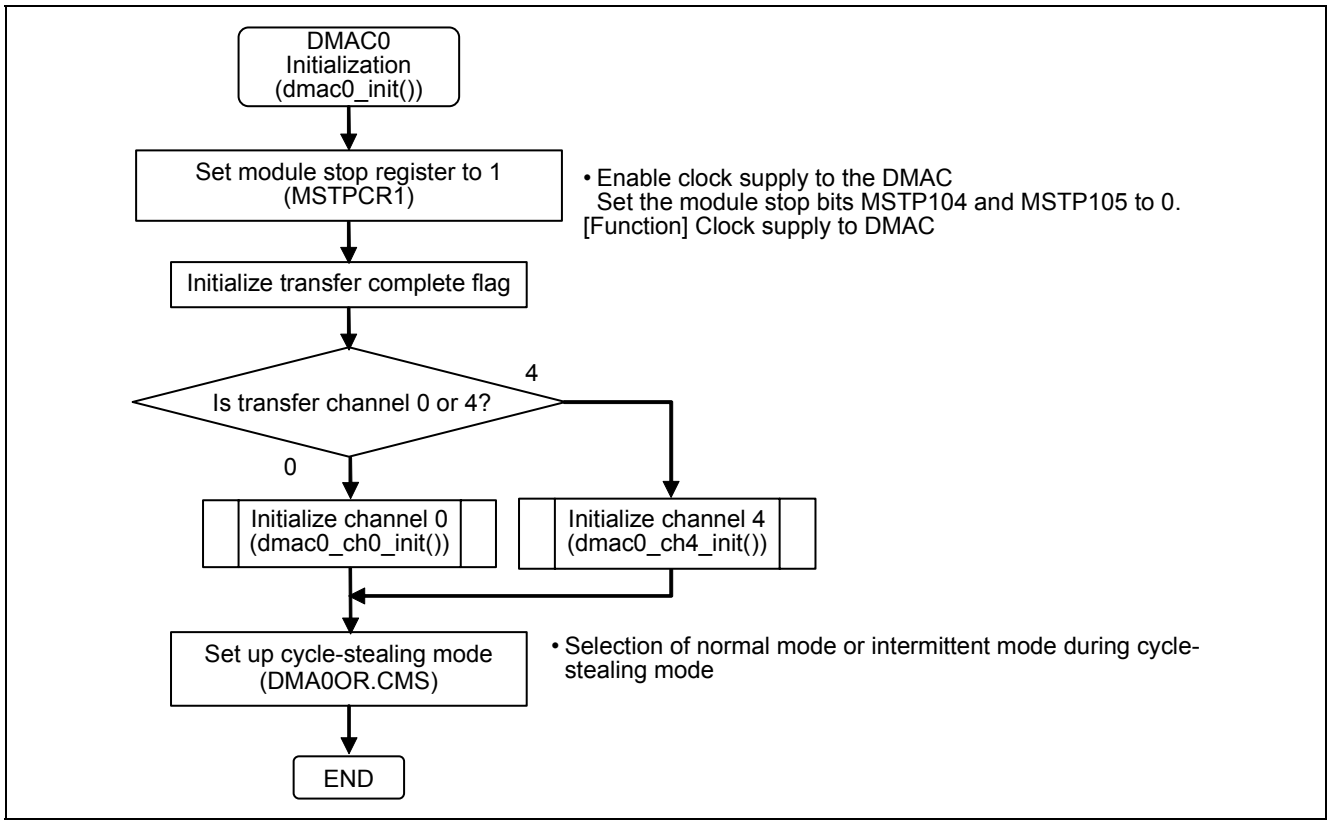


Figure 5.18 DMAC0 Initialization Flowchart

5.2.10 DMAC0 Channel 0 and 4 Initialization (dmac0_ch0_init(), dmac0_ch4_init())

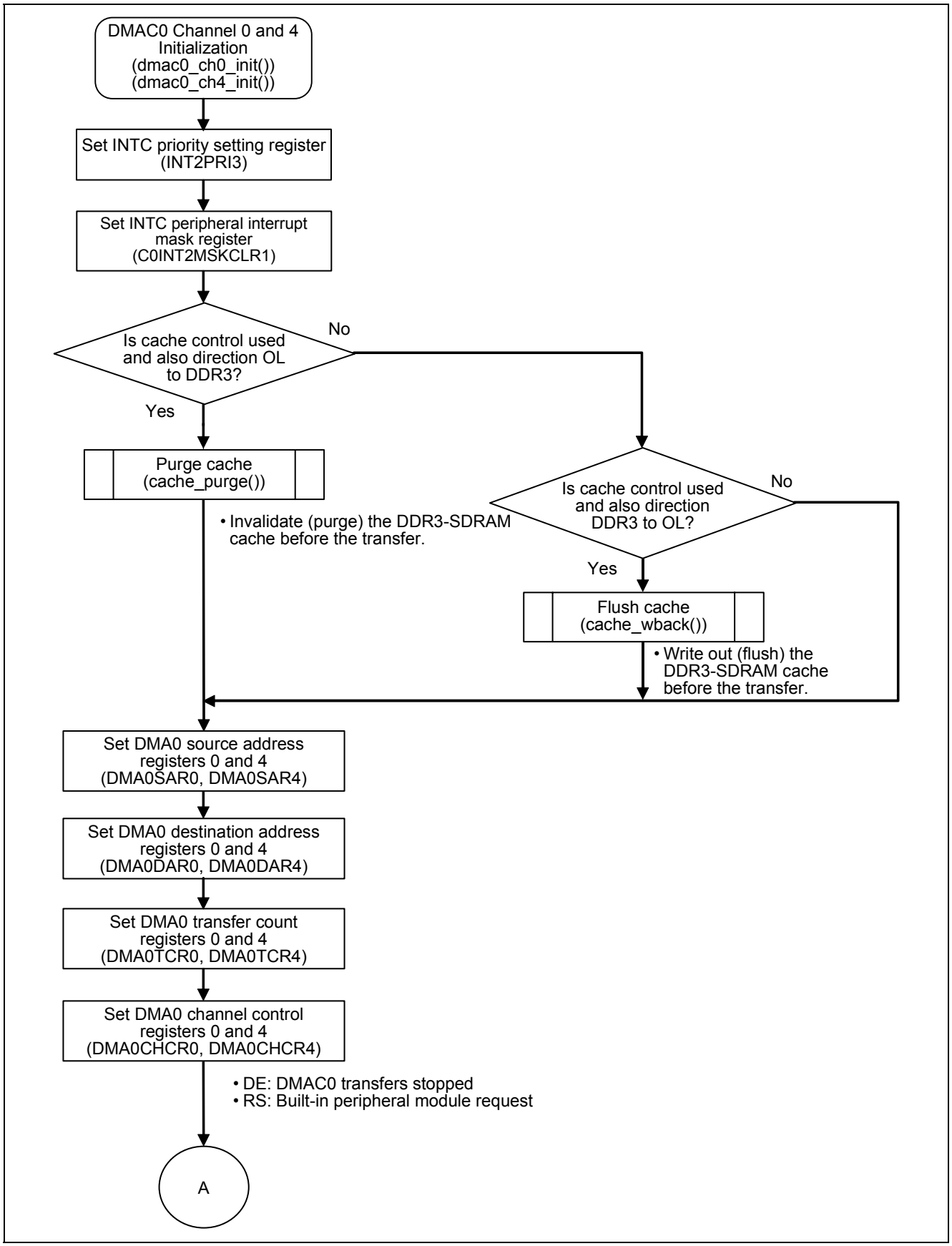


Figure 5.19 DMAC0 Channel 0 and 4 Initialization Flowchart 1

5.2.11 DMAC0 Channel 0 and 4 Initialization 2 (dmac0_ch0_init(), dmac0_ch4_init())

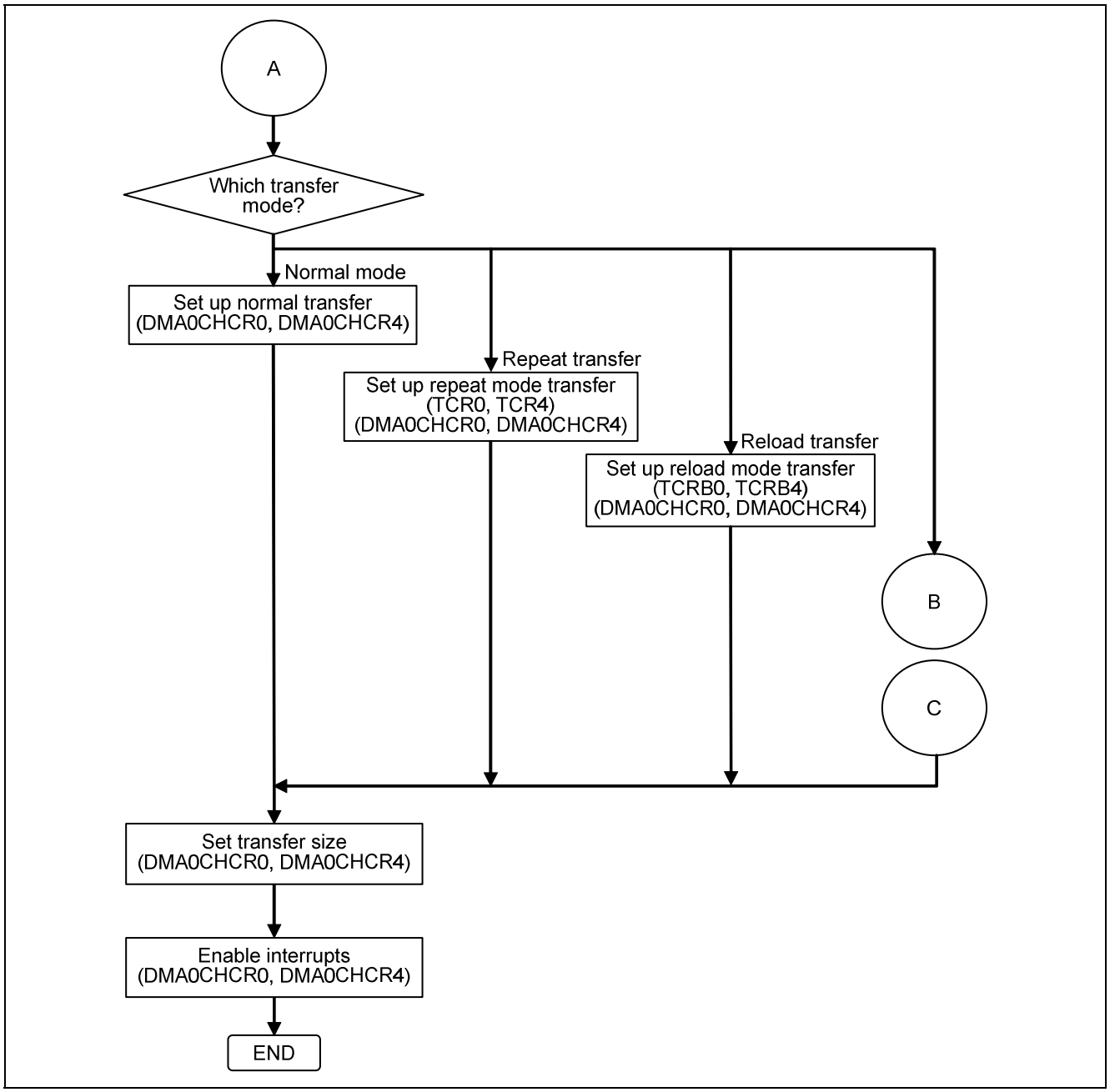


Figure 5.20 DMAC0 Channel 0 and 4 Initialization Flowchart 2

5.2.12 DMAC0 Channel 0 and 4 Initialization 3 (dmac0_ch0_init(), dmac0_ch4_init())

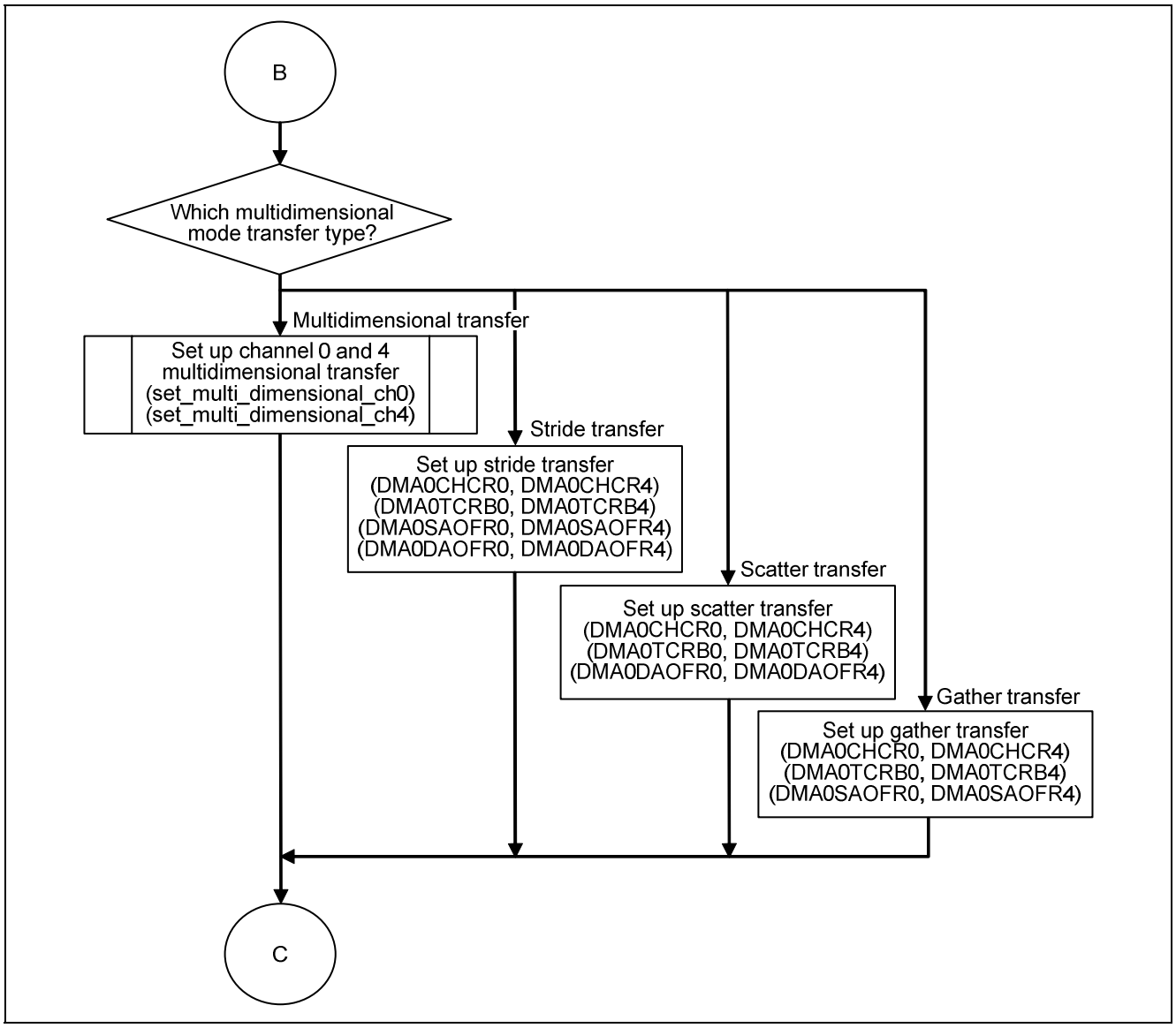


Figure 5.21 DMAC0 Channel 0 and 4 Initialization Flowchart 3

5.2.13 DMAC0 Channel 0 and 4 Multidimensional Initialization (set_multi_dimensional_ch0(), set_multi_dimensional_ch4())

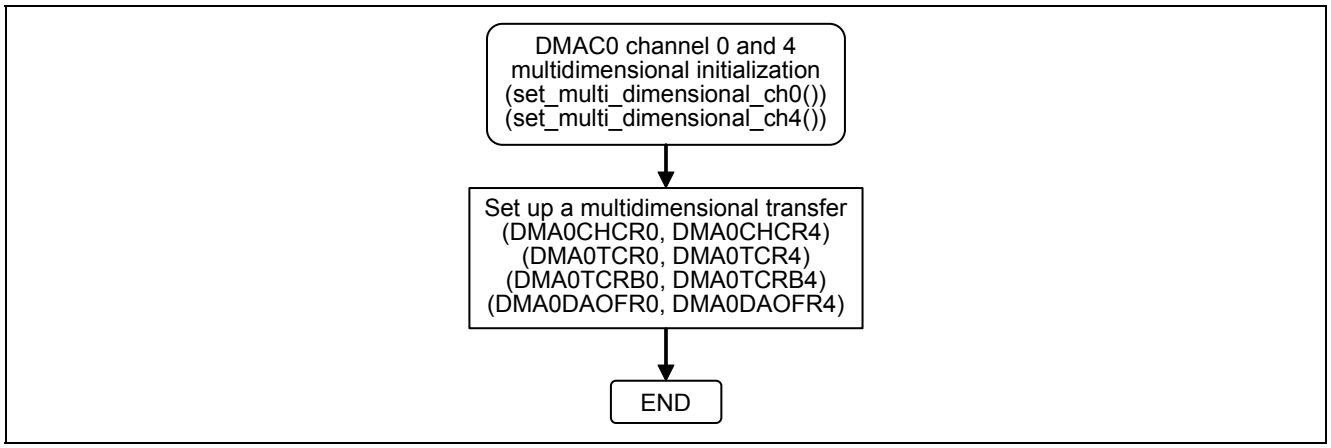


Figure 5.22 DMAC0 Channel 0 and 4 Multidimensional Initialization Flowchart

5.2.14 DMAC0 Start (dmac0_start())

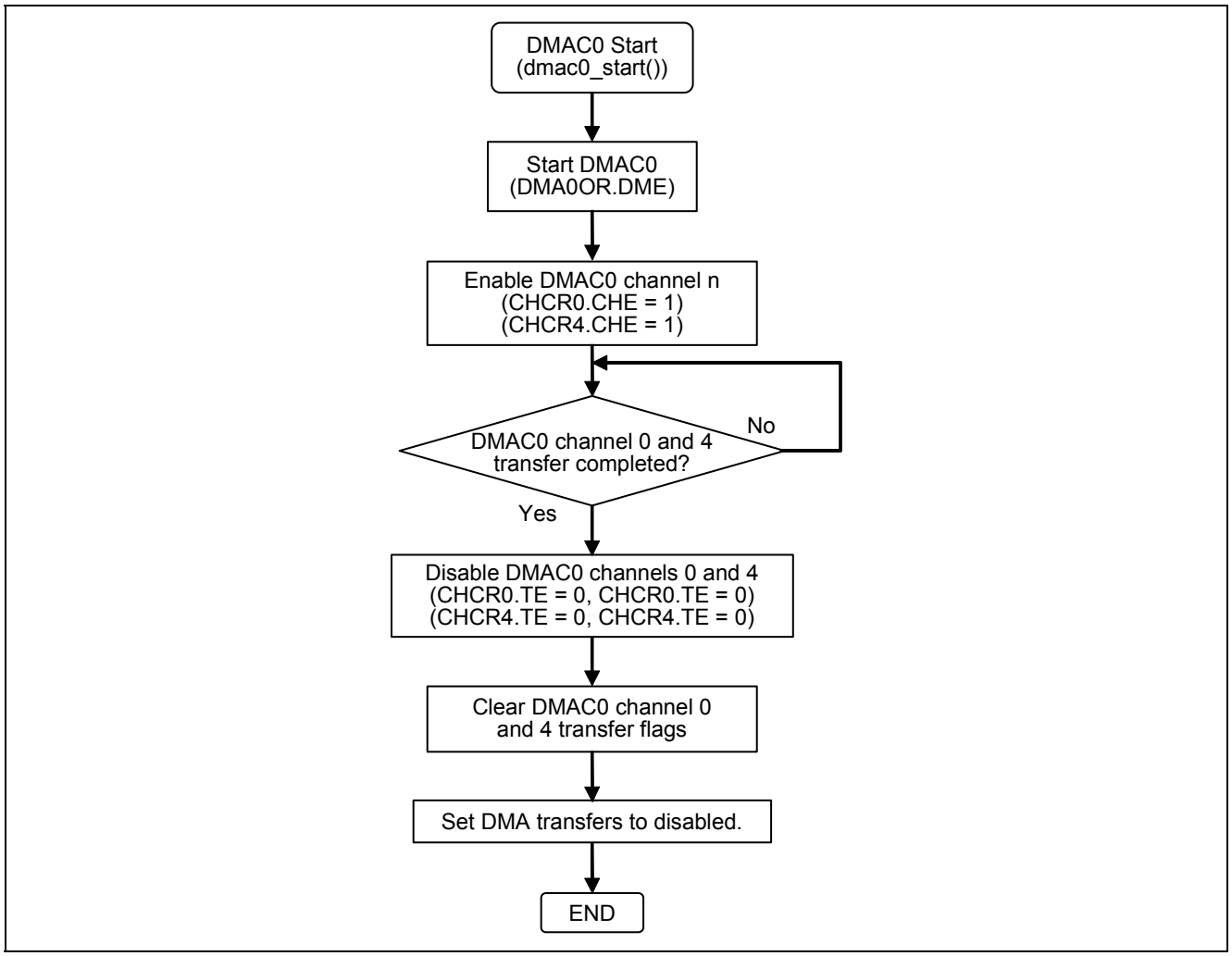


Figure 5.23 DMAC0 Start Flowchart

5.2.15 DMAC0 Transfer Result Display (dmac0_result())

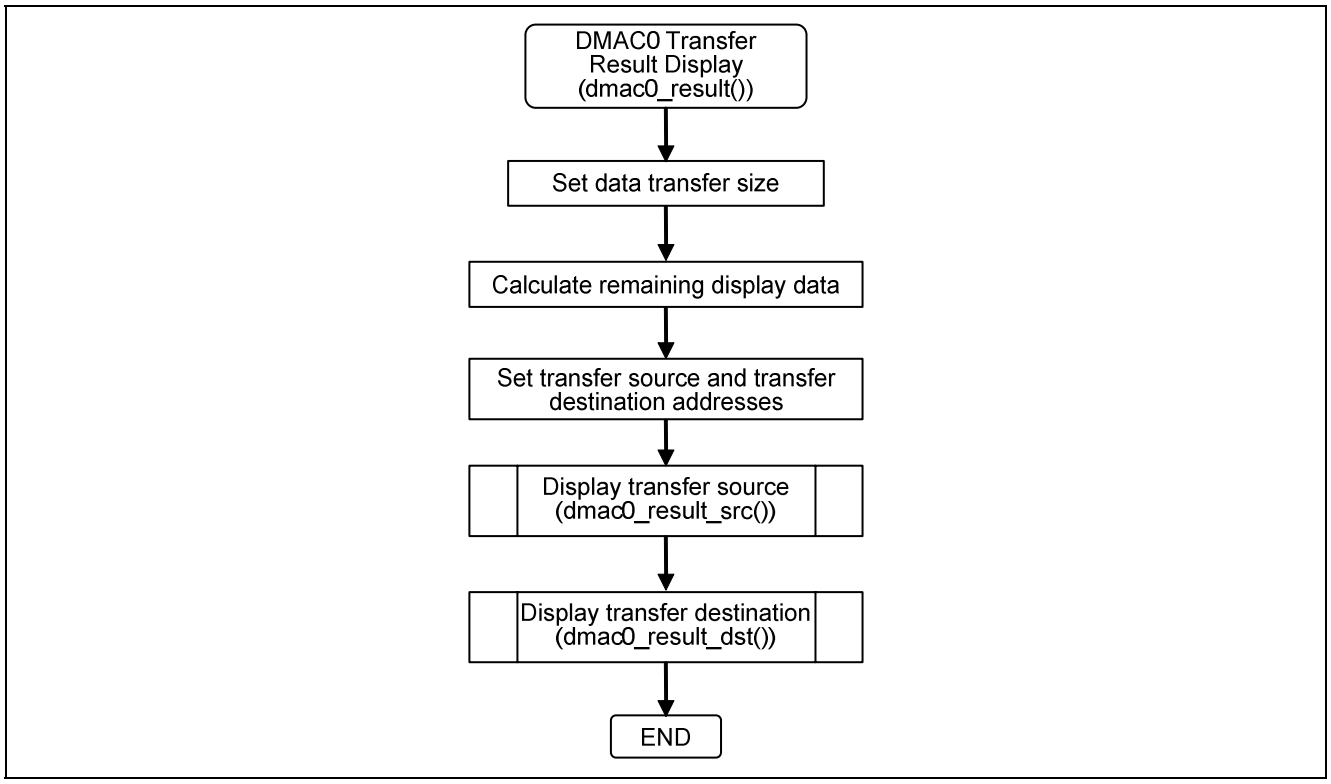


Figure 5.24 DMAC0 Transfer Result Display Flowchart

5.2.16 DMAC0 Transfer Source Display 1 (dmac0_result_src())

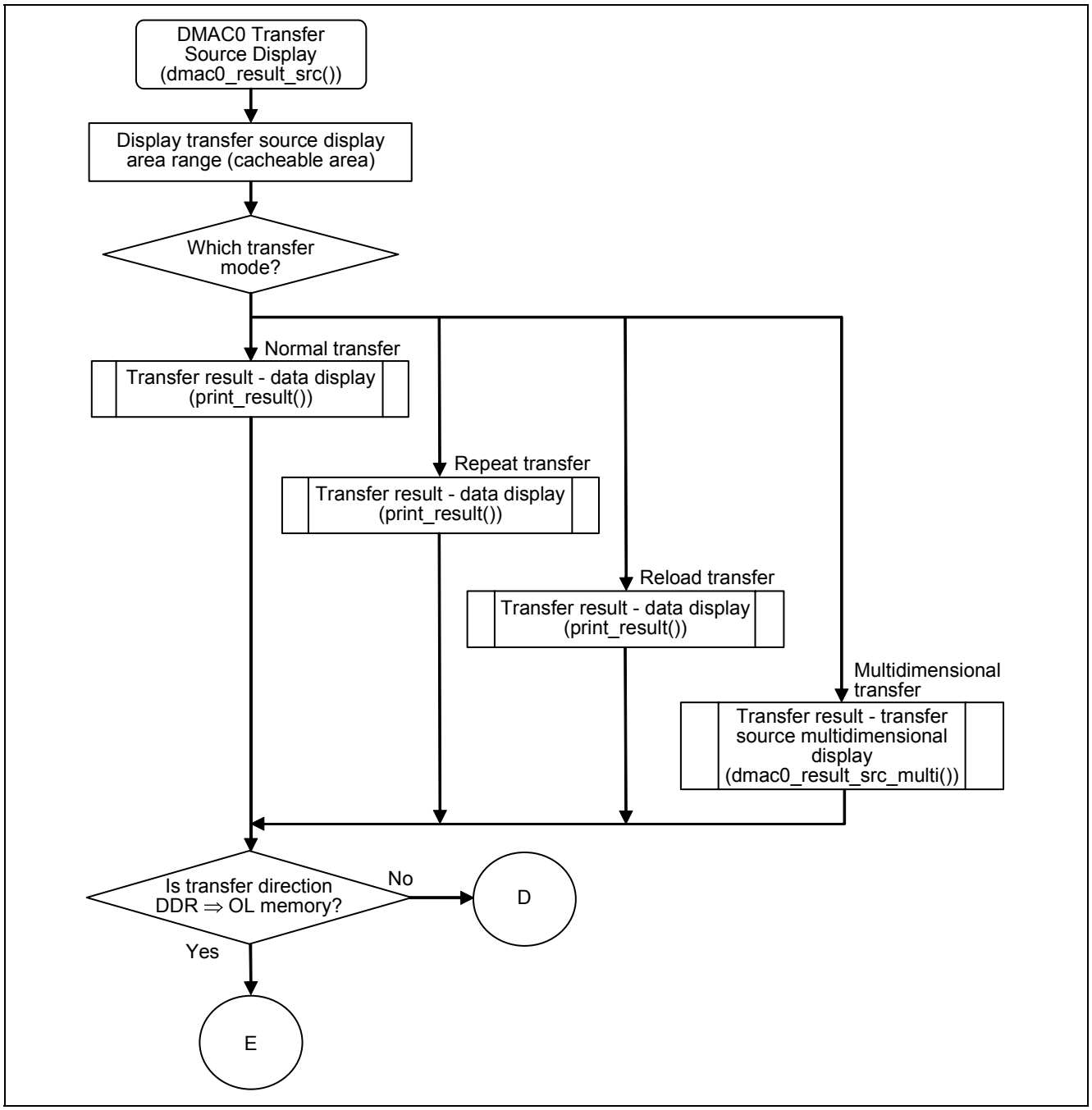


Figure 5.25 DMAC0 Transfer Source Display 1 Flowchart

5.2.17 DMAC0 Transfer Source Display 2 (dmac0_result_src())

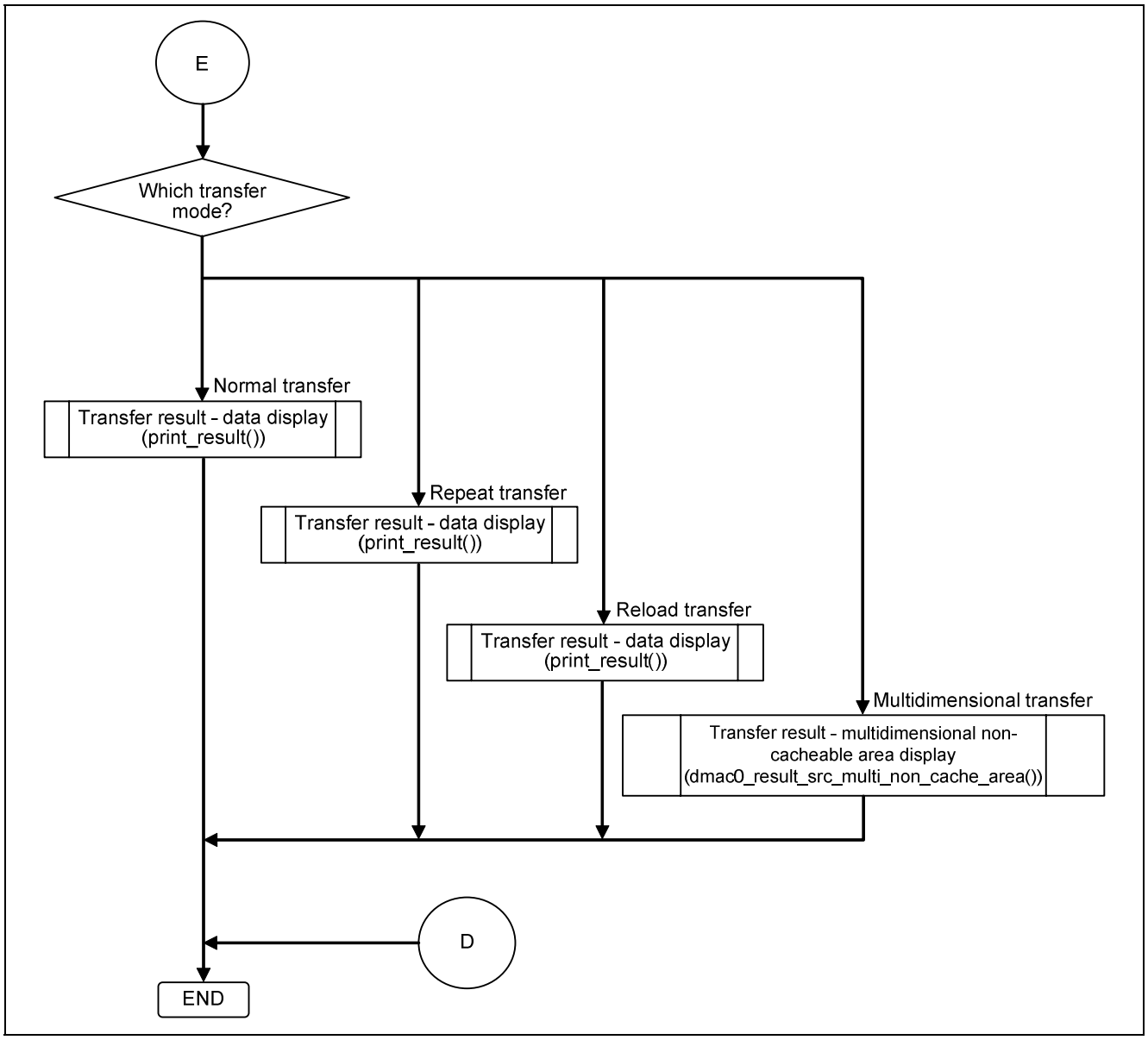


Figure 5.26 DMAC0 Transfer Source Display Flowchart 2

5.2.18 Transfer Result Multidimensional Non-Cacheable Area Display (dmac0_result_src_multi_non_cache_area())

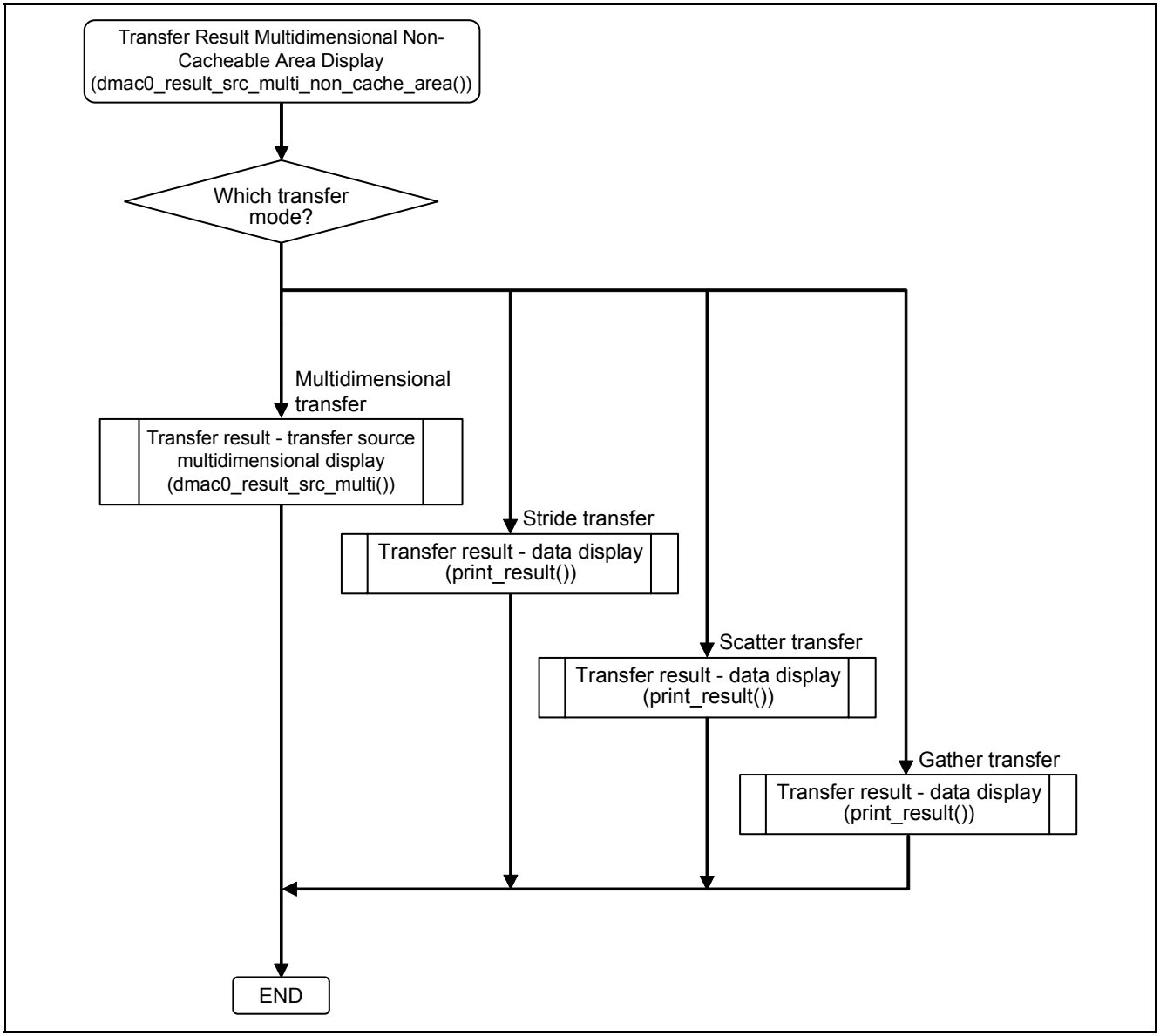


Figure 5.27 Transfer Result Multidimensional Non-Cacheable Area Display Flowchart

5.2.19 Transfer Result Transfer Source Multidimensional Display (dmac0_result_src_multi())

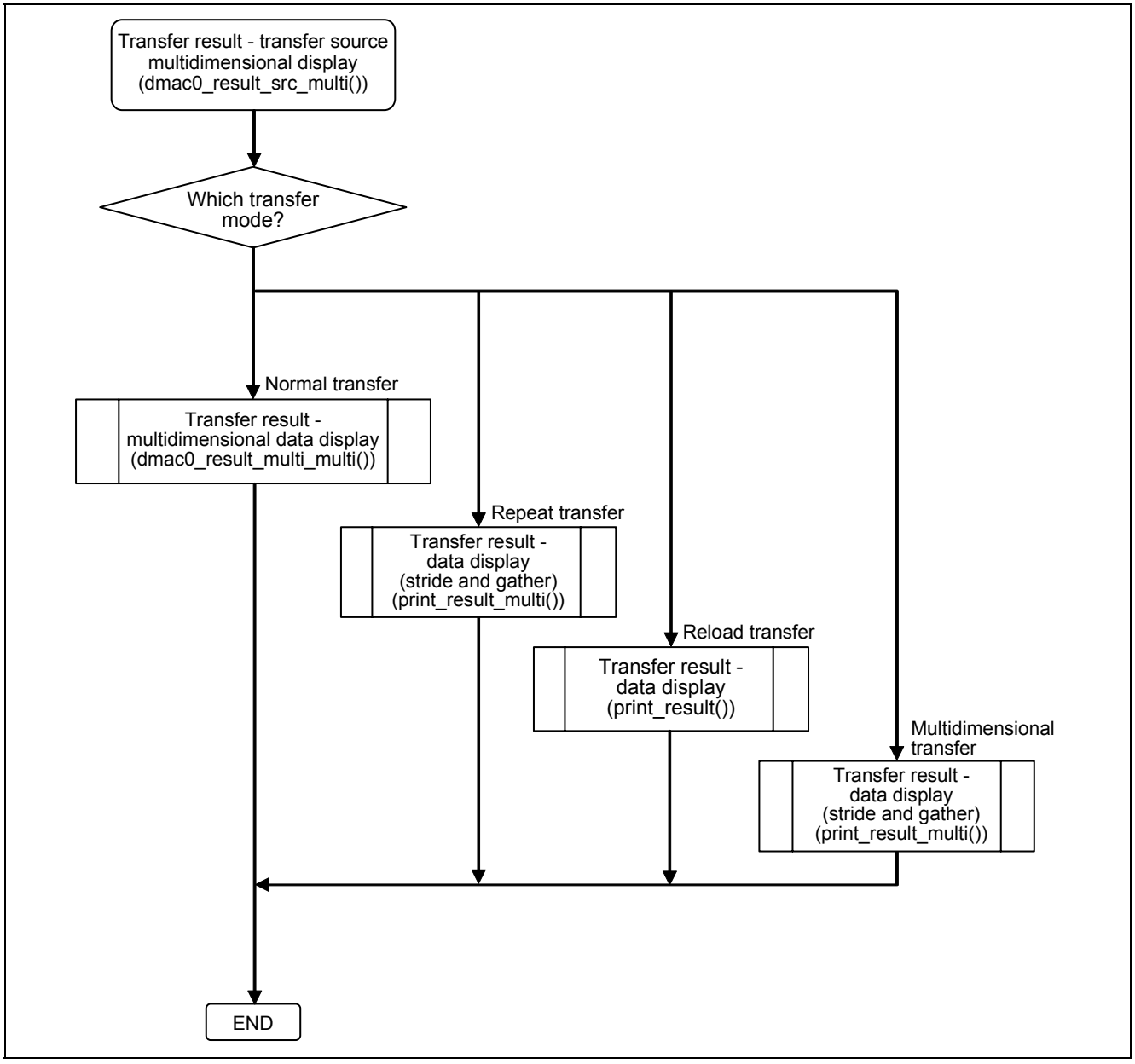


Figure 5.28 Transfer Result Transfer Source Multidimensional Display Flowchart

5.2.20 DMAC0 Transfer Destination Display (dmac0_result_dst())

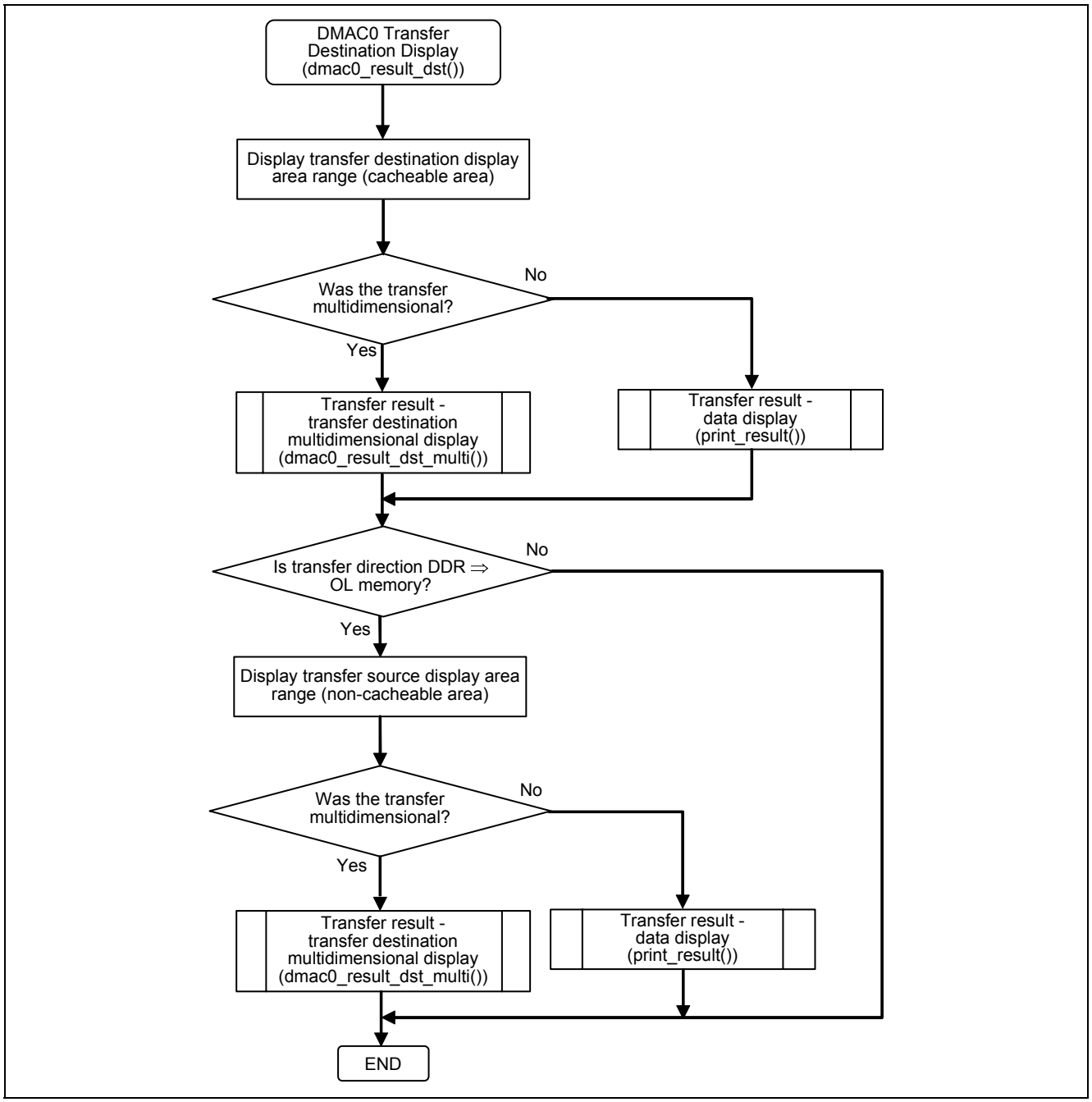


Figure 5.29 DMAC0 Transfer Destination Display Flowchart

5.2.21 Transfer Result Transfer Destination Multidimensional Display (dmac0_result_dst_multi())

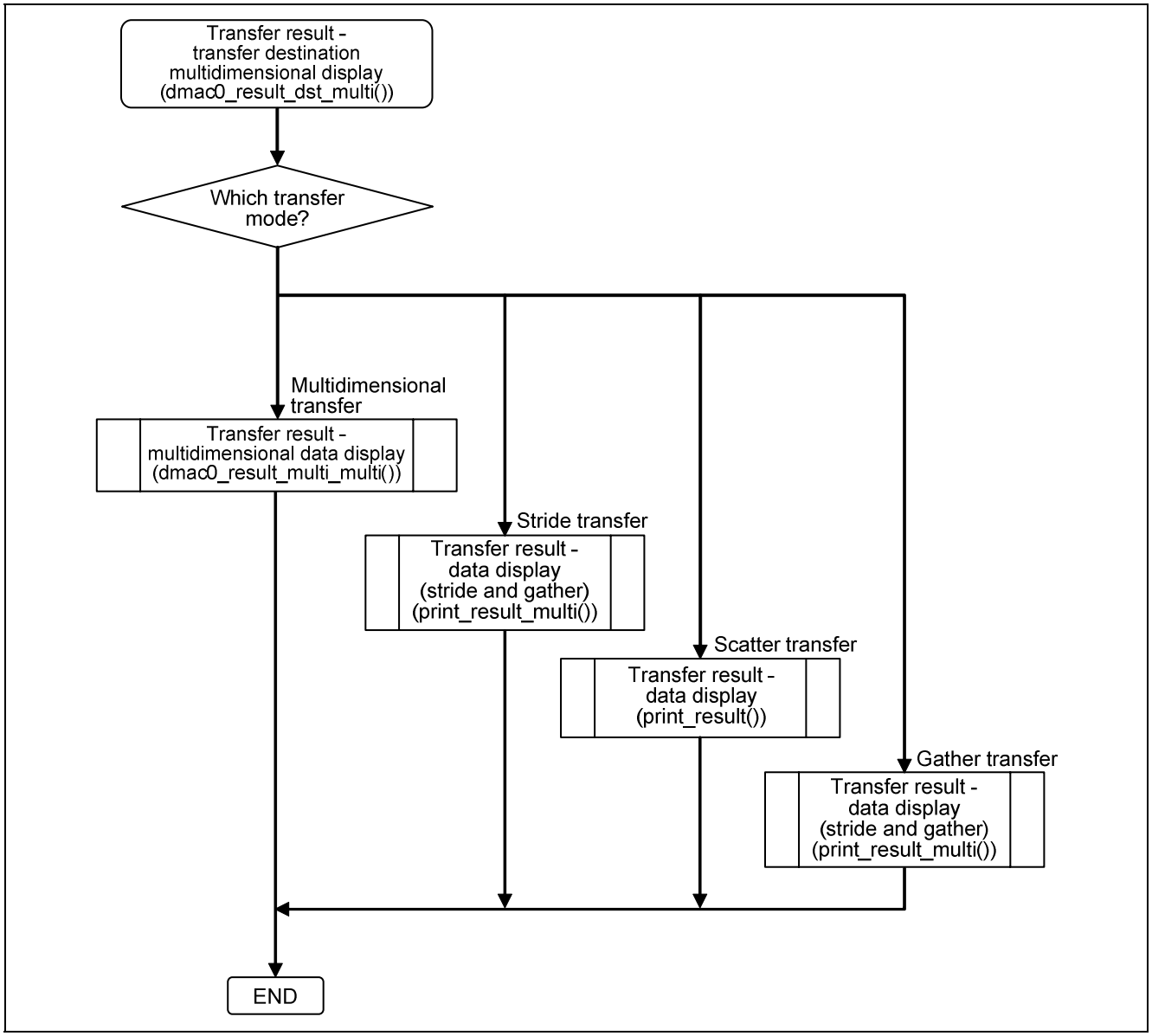


Figure 5.30 Transfer Result Transfer Destination Multidimensional Display Flowchart

5.2.22 Transfer Result Multidimensional Data Display (dmac0_result_multi_multi())

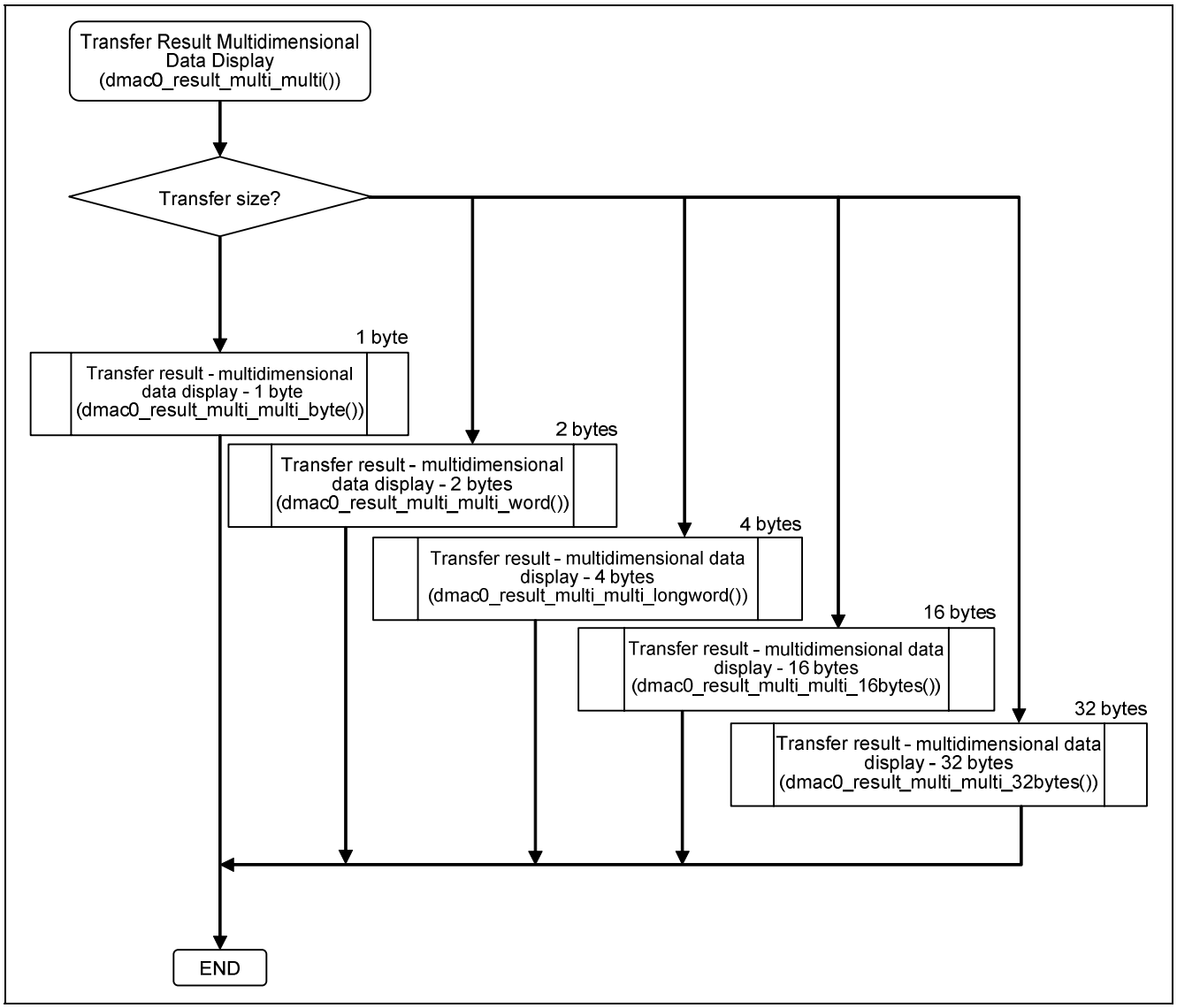


Figure 5.31 Transfer Result Multidimensional Data Display Flowchart

5.2.23 Transfer Result Multidimensional Data Display n Bytes (dmac0_result_multi_multi_n(), n = byte, word, longword, 16bytes, or 32bytes)

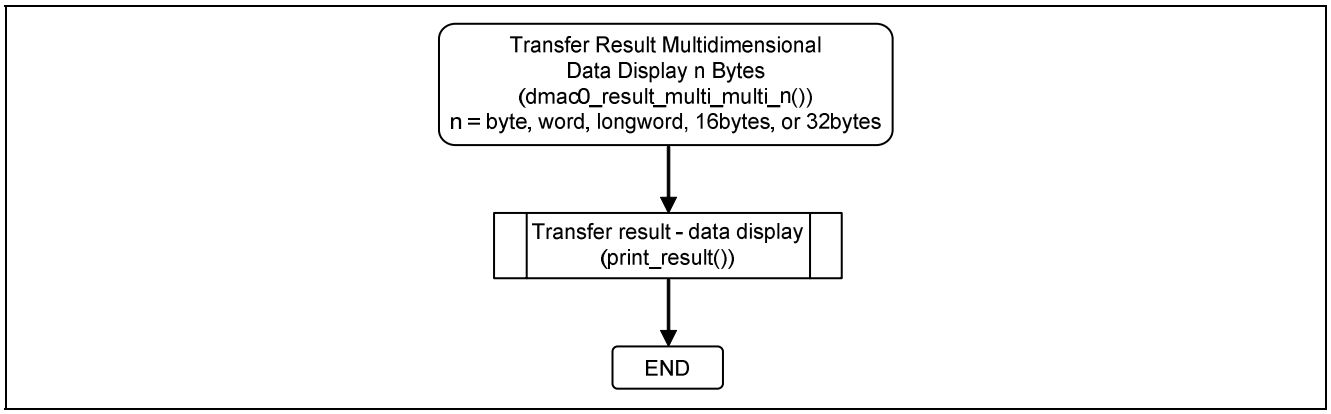


Figure 5.32 Transfer Result Multidimensional Data Display n Bytes Flowchart

5.2.24 DMAC0 Interrupt Handlers Channels 0 and 4 (INT_DAM0INT0, INT_DAM0INT4)

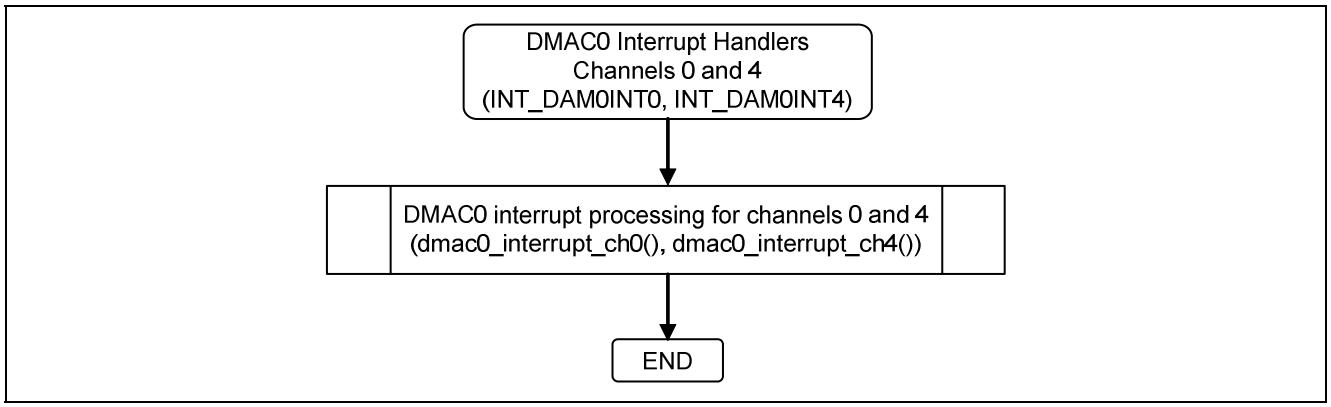


Figure 5.33 DMAC0 Interrupt Handlers Channels 0 and 4 Flowchart

5.2.25 DMAC0 Interrupt Processing Channels 0 and 4 (dmac0_interrupt_ch0(), dmac0_interrupt_ch4())

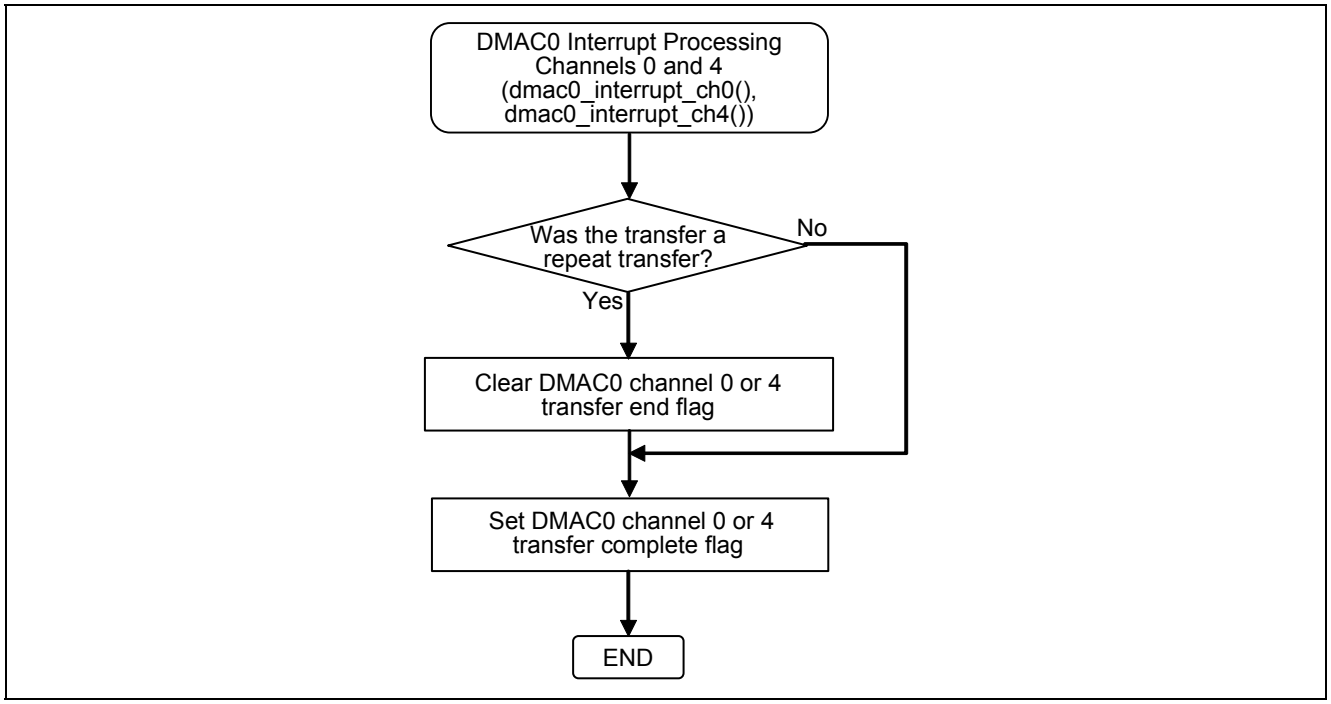


Figure 5.34 DMAC0 Interrupt Processing Channels 0 and 4 Flowchart

5.3 DMAC1 Processing Procedures

The following figures show the flowcharts for the DMAC1 processing procedures.

5.3.1 DMAC1 Select Channel (dmac1_select_channel())

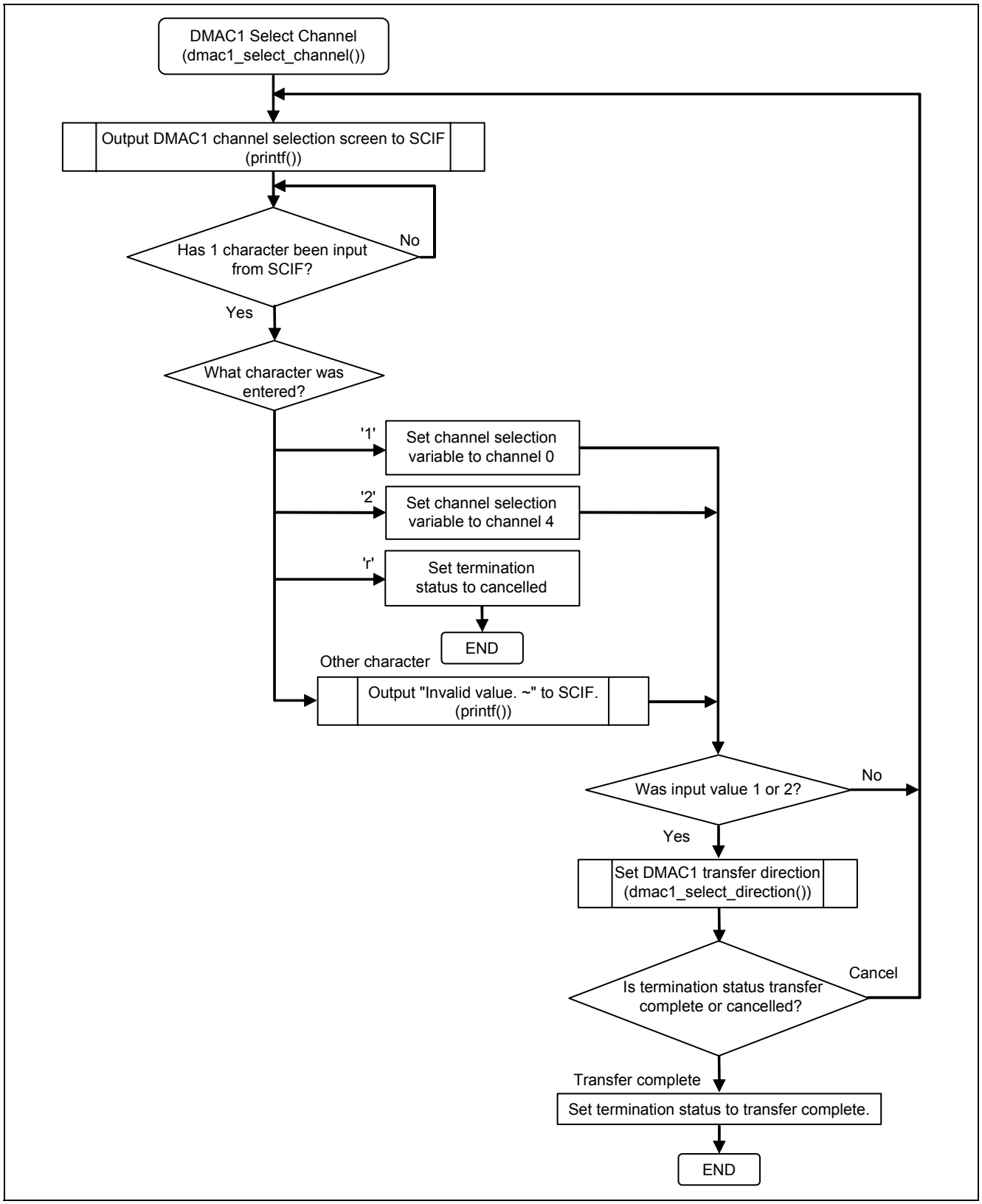


Figure 5.35 DMAC1 Select Channel Flowchart

5.3.2 DMAC1 Select Direction (dmac1_select_direction())

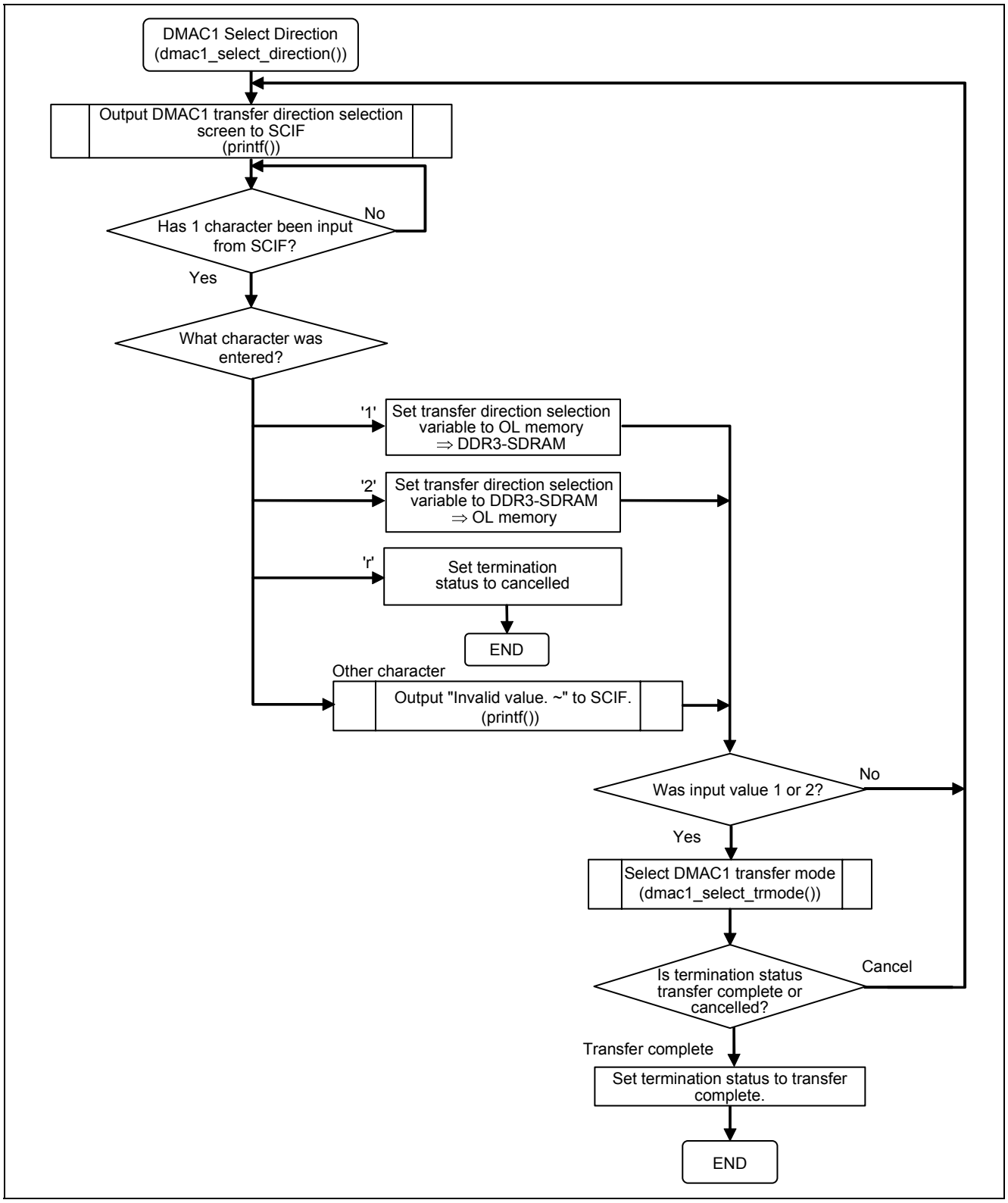


Figure 5.36 DMAC1 Select Direction Flowchart

5.3.3 DMAC1 Select Transfer Mode (dmac1_select_direction())

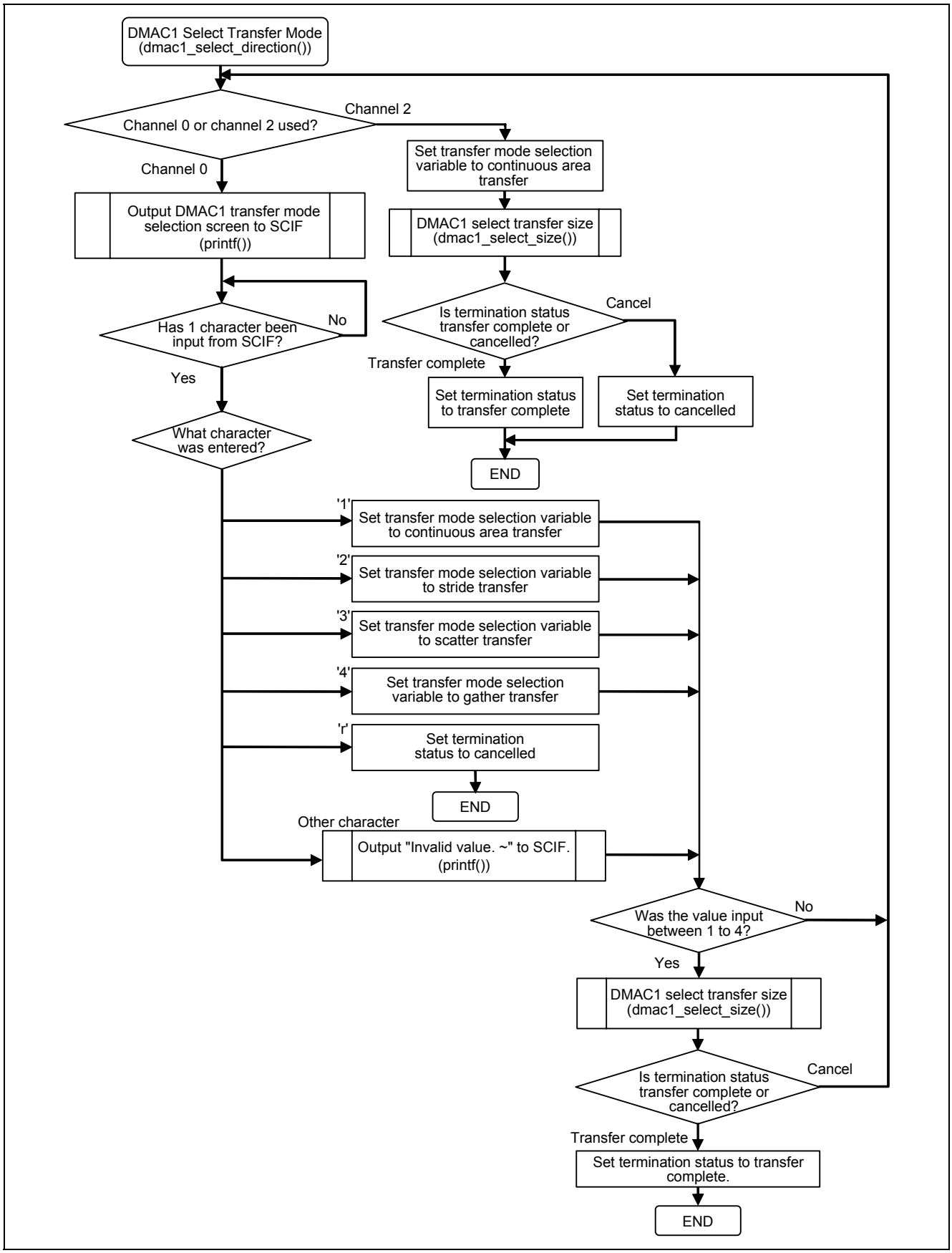


Figure 5.37 DMAC1 Transfer Mode Selection Flowchart

5.3.4 DMAC1 Select Transfer Size (dmac1_select_size())

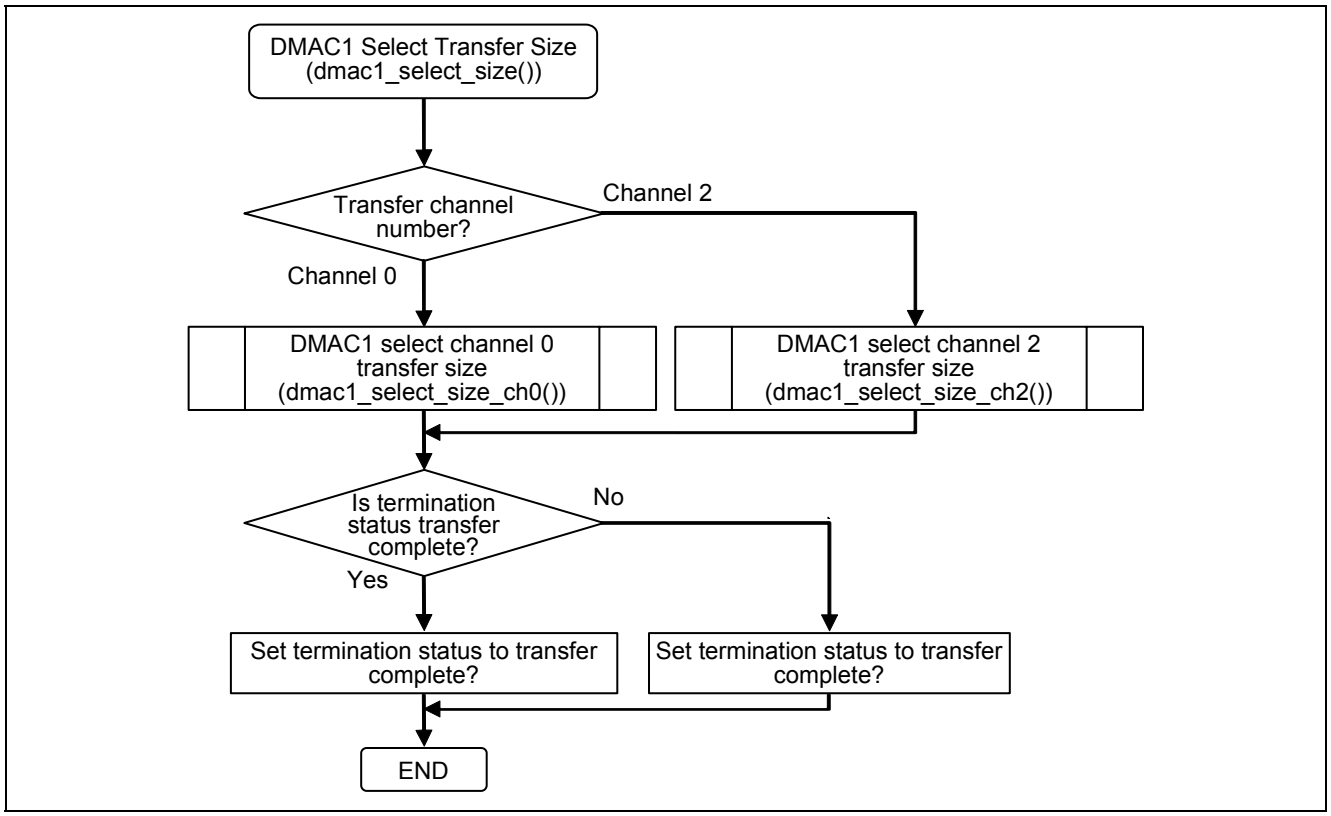


Figure 5.38 DMAC1 Transfer Size Selection Flowchart

5.3.5 DMAC1 Select Transfer Size Channel 0 (dmac1_select_size_ch0())

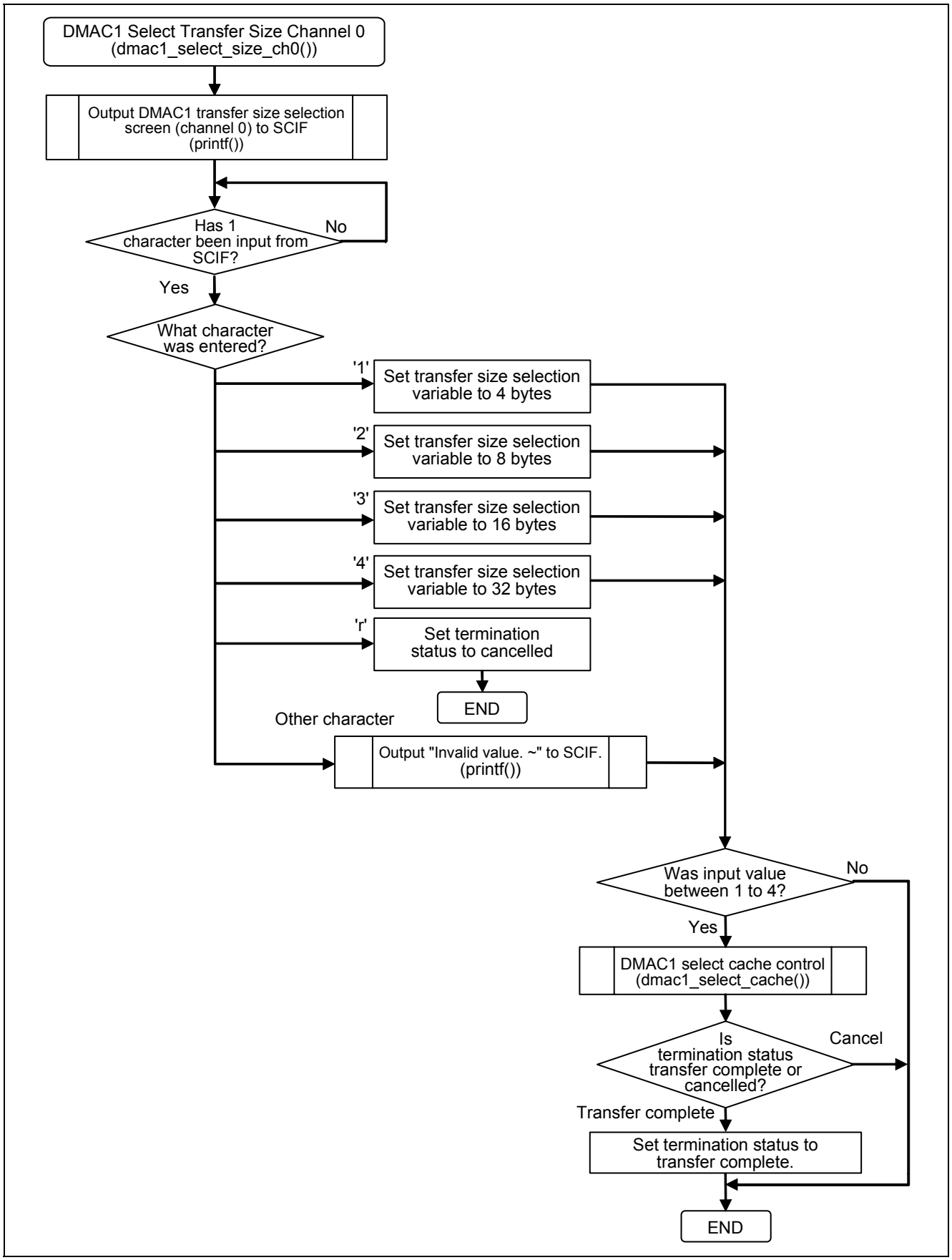


Figure 5.39 DMAC1 Transfer Size Selection Channel 0 Flowchart

5.3.6 DMAC1 Select Transfer Size Channel 2 (dmac1_select_size_ch2())

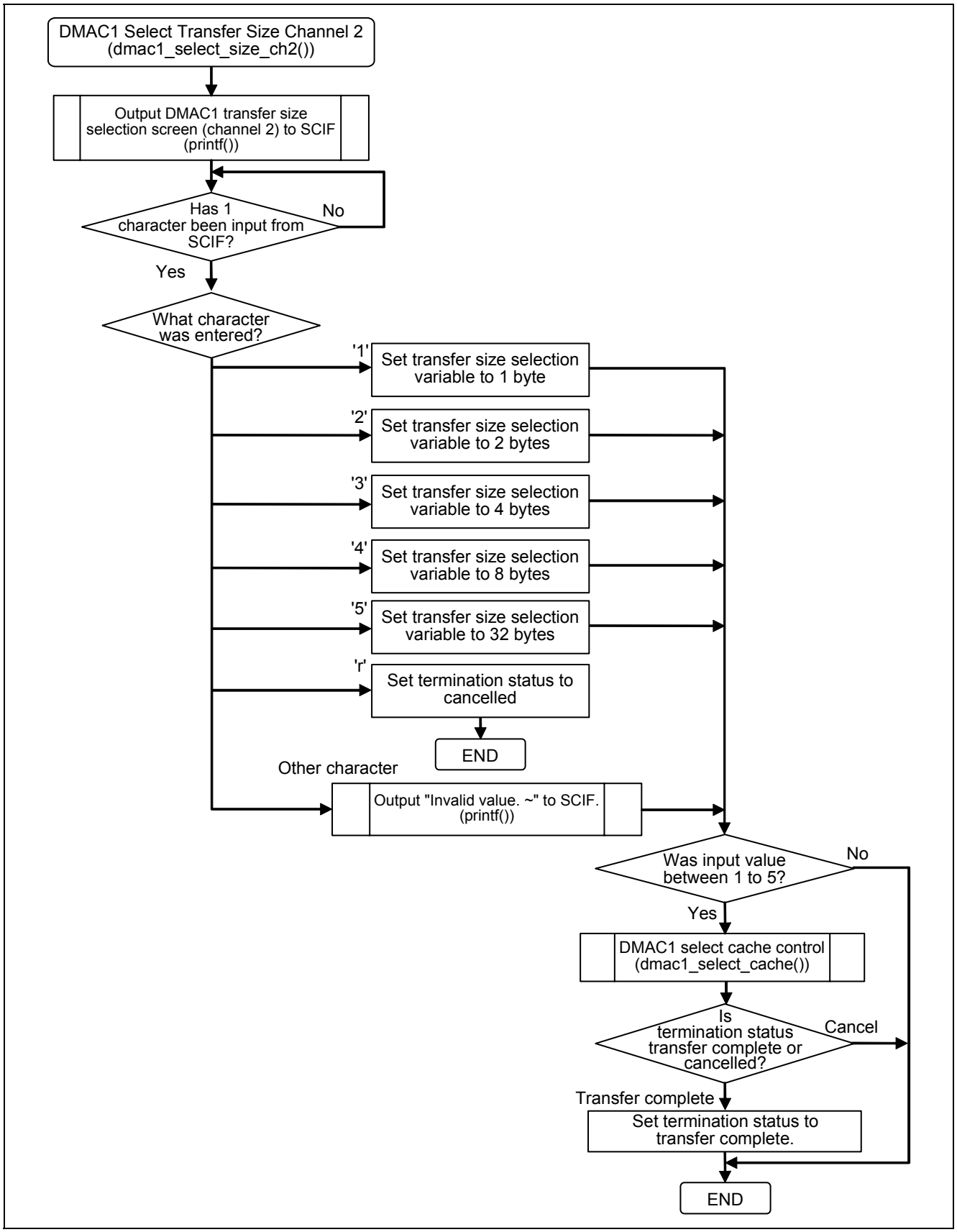


Figure 5.40 DMAC1 Transfer Size Selection Channel 2 Flowchart

5.3.7 DMAC1 Select Cache Control (dmac1_select_cache())

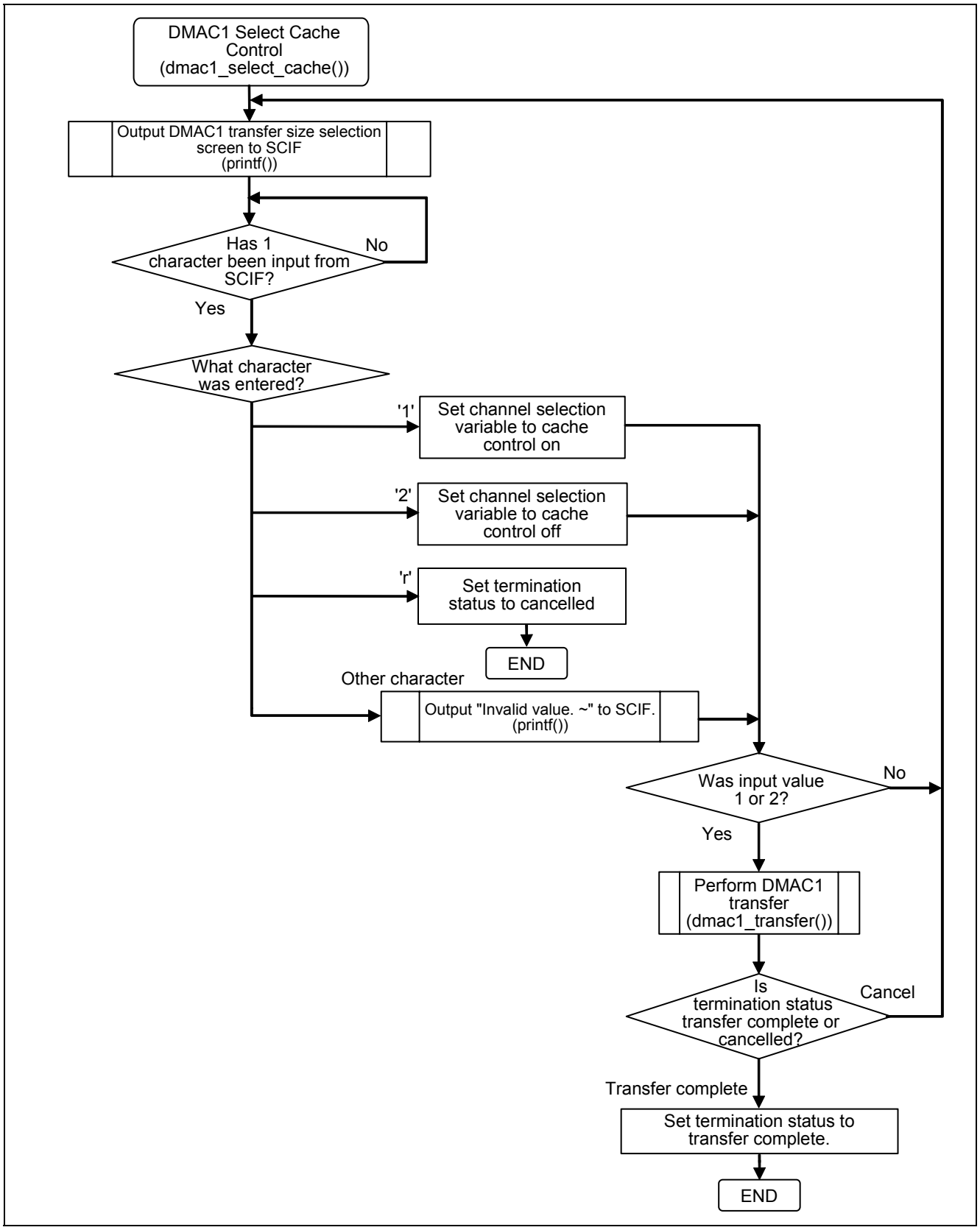


Figure 5.41 DMAC1 Cache Control Selection Flowchart

5.3.8 DMAC1 Transfer (dmac1_transfer())

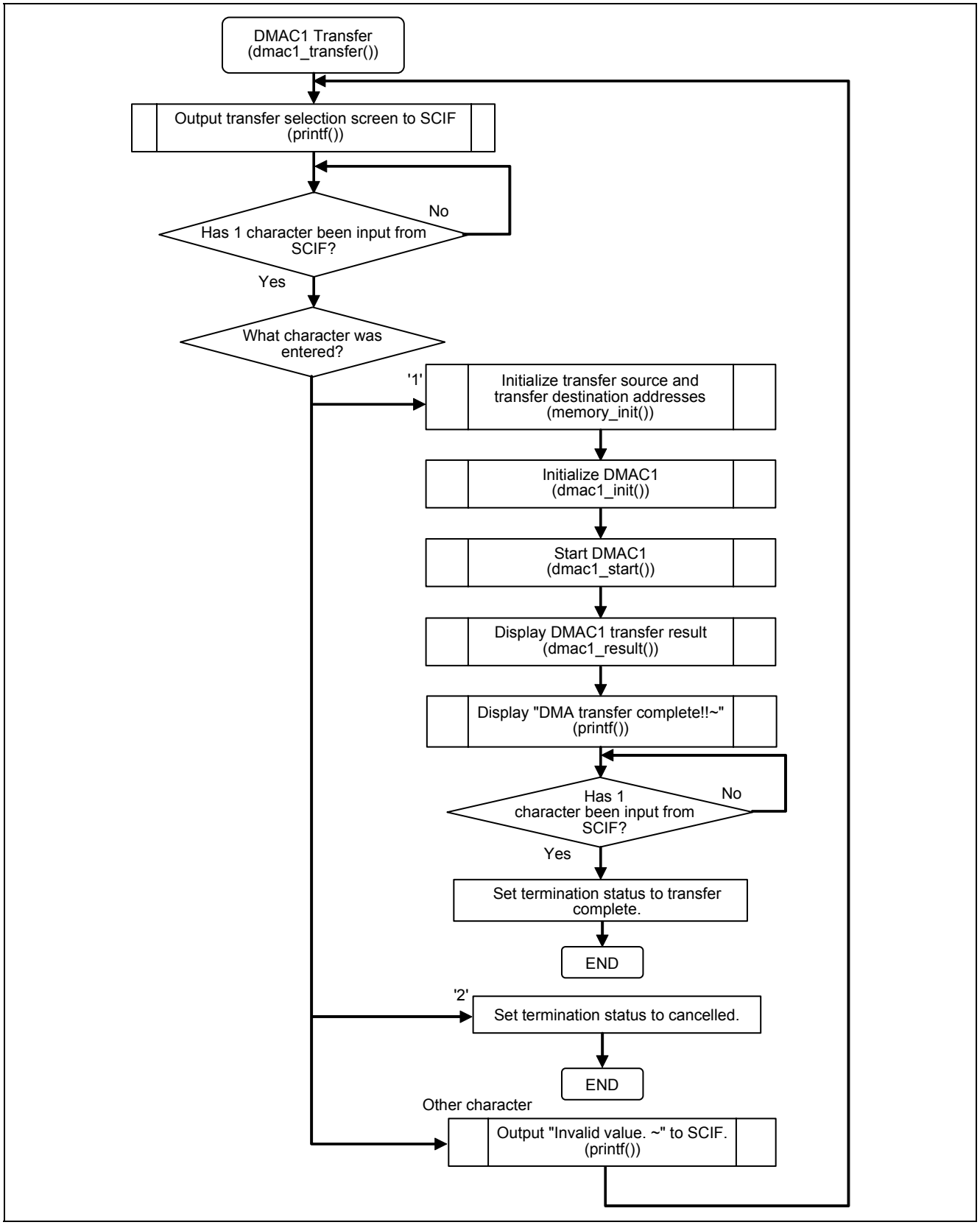


Figure 5.42 DMAC1 Transfer Flowchart

5.3.9 DMAC1 Initialization (dmac1_init())

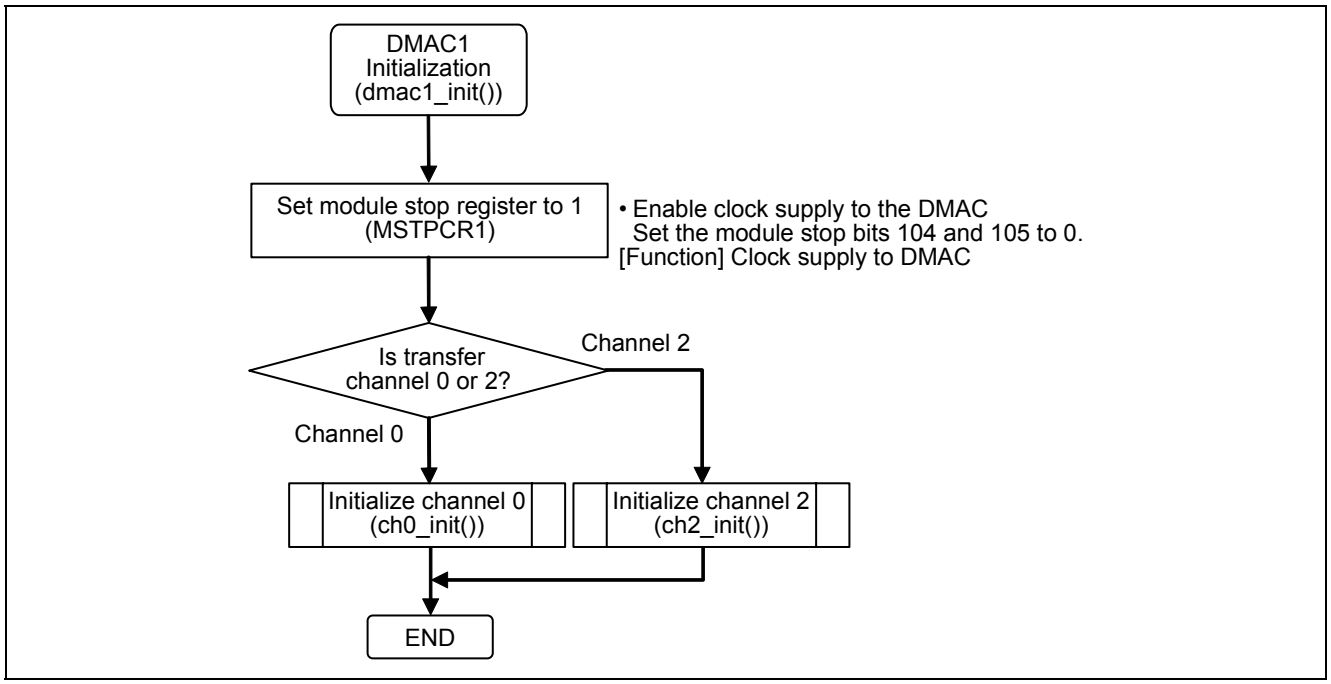


Figure 5.43 DMAC1 Initialization Flowchart

5.3.10 DMAC1 Channel 0 Initialization (dmac1_ch0_init())

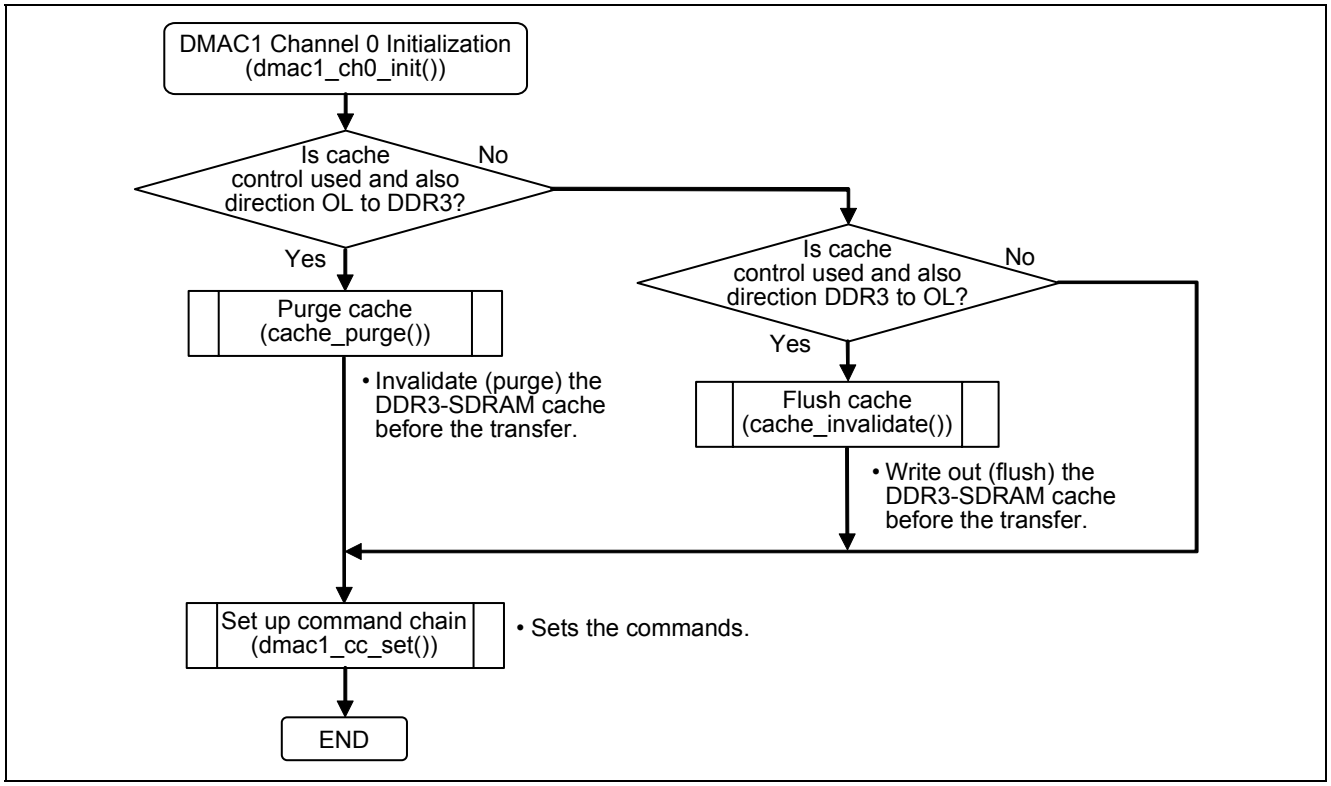


Figure 5.44 DMAC1 Channel 0 Initialization Flowchart

5.3.11 DMAC1 Channel 2 Initialization (dmac1_ch2_init())

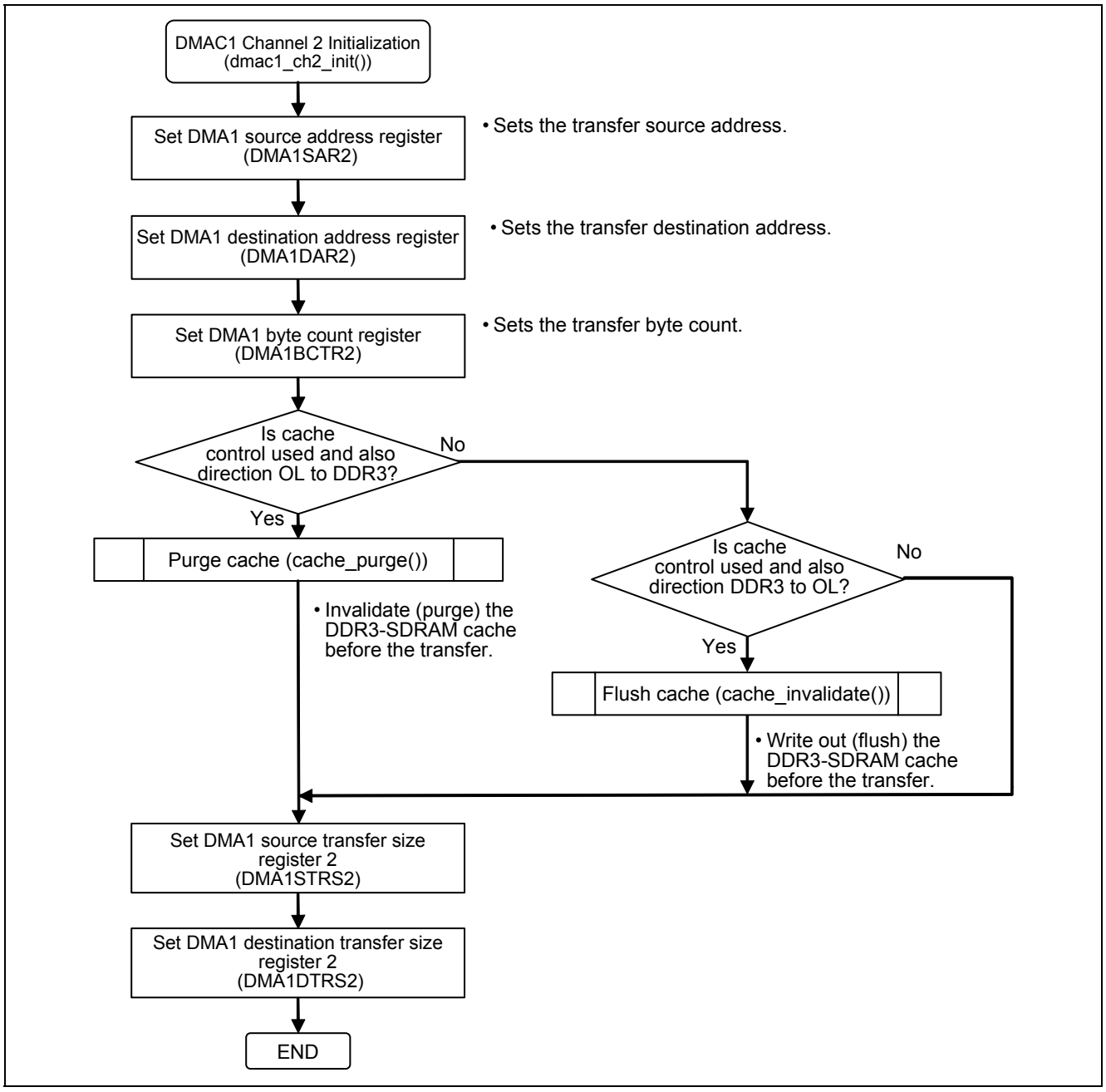


Figure 5.45 DMAC1 Channel 2 Initialization Flowchart

5.3.12 Command Chain Setup (dmac1_cc_set())

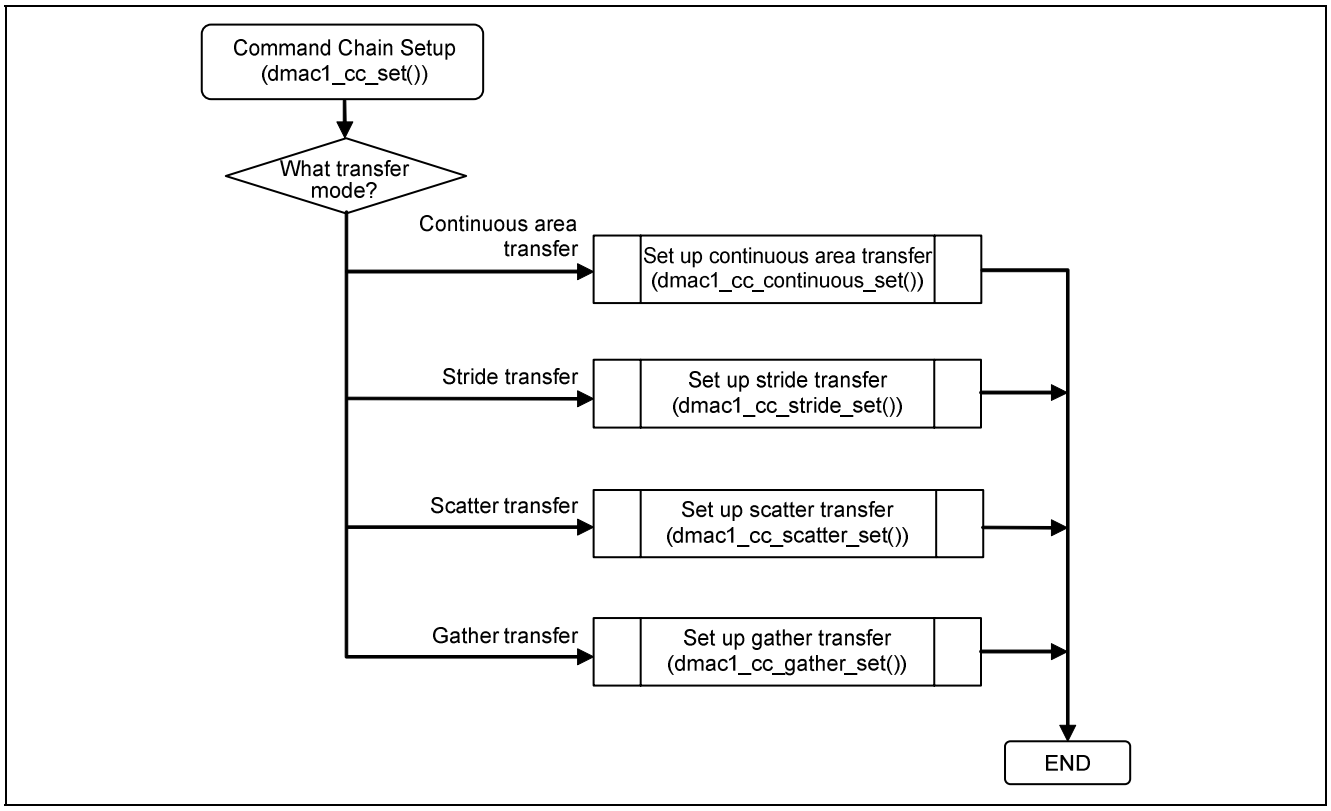
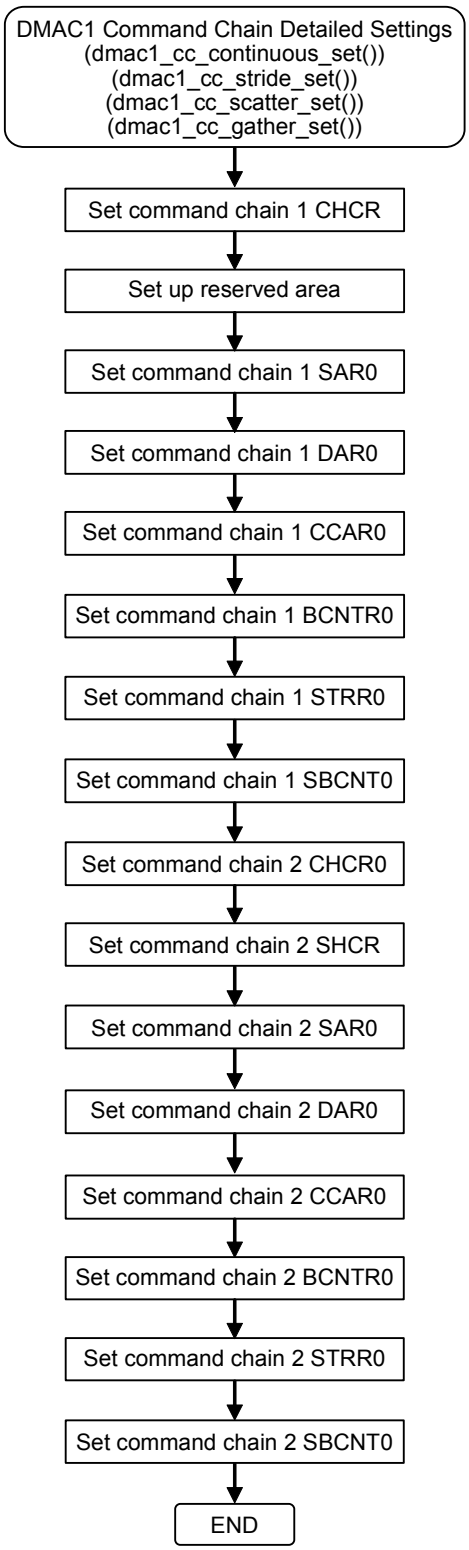


Figure 5.46 Command Chain Setup Flowchart

5.3.13 DMAC1 Command Chain Detailed Settings (dmac1_cc_continuous_set(), dmac1_cc_stride_set(), dmac1_cc_scatter_set(), dmac1_cc_gather_set())



Note: Since the specifications of this sample program stipulate that a menu selection screen is output to a serial console and that a DMA transfer is performed according to the transfer method selected from that menu, the various register values will differ depending on that menu selection. See section 3.1.4, Sample Program Register Settings, for details on the register setting values.

Figure 5.47 Command Chain Detailed Settings Flowchart

5.3.14 DMAC1 Start (dmac1_start())

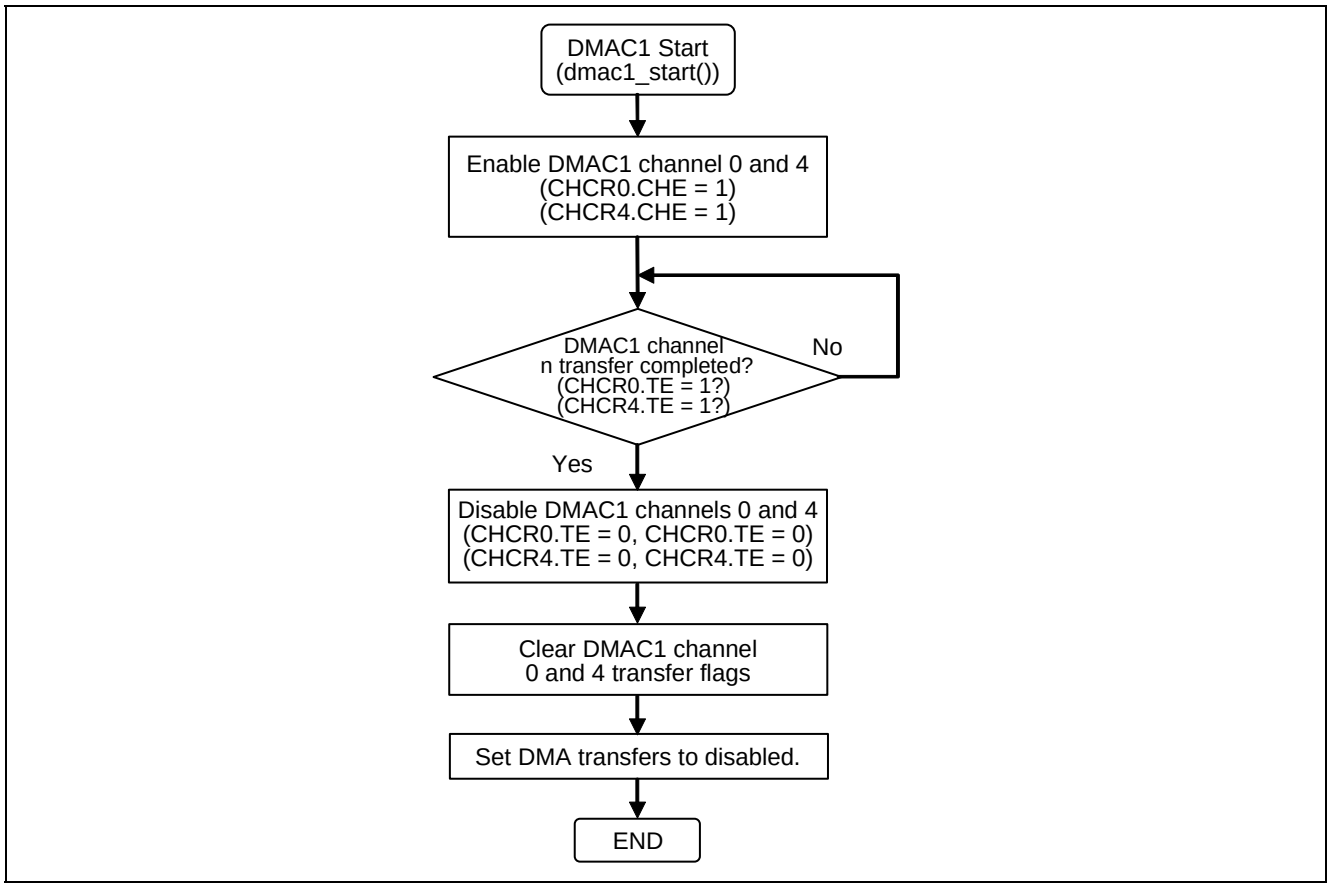


Figure 5.48 DMAC1 Start Flowchart

5.3.15 DMAC1 Transfer Result Display

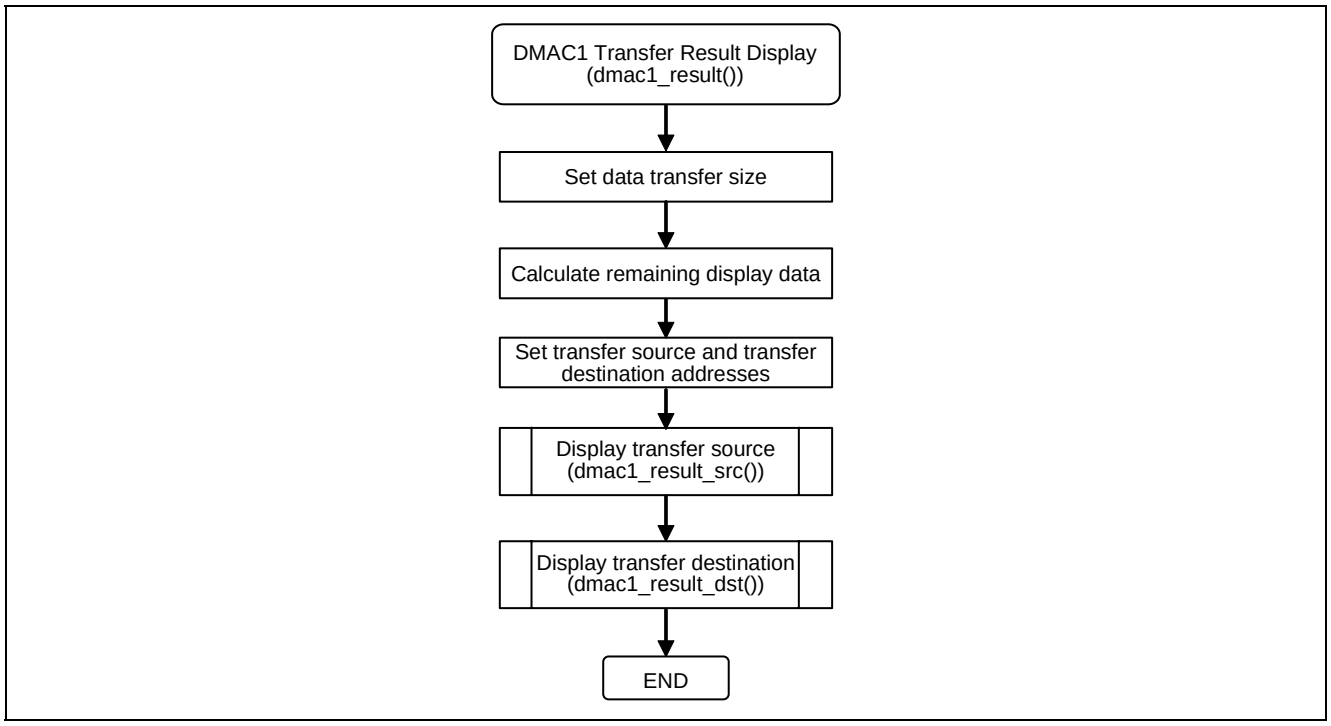


Figure 5.49 DMAC1 Transfer Result Display Flowchart

5.3.16 DMAC1 Transfer Source Display (dmac1_result_src())

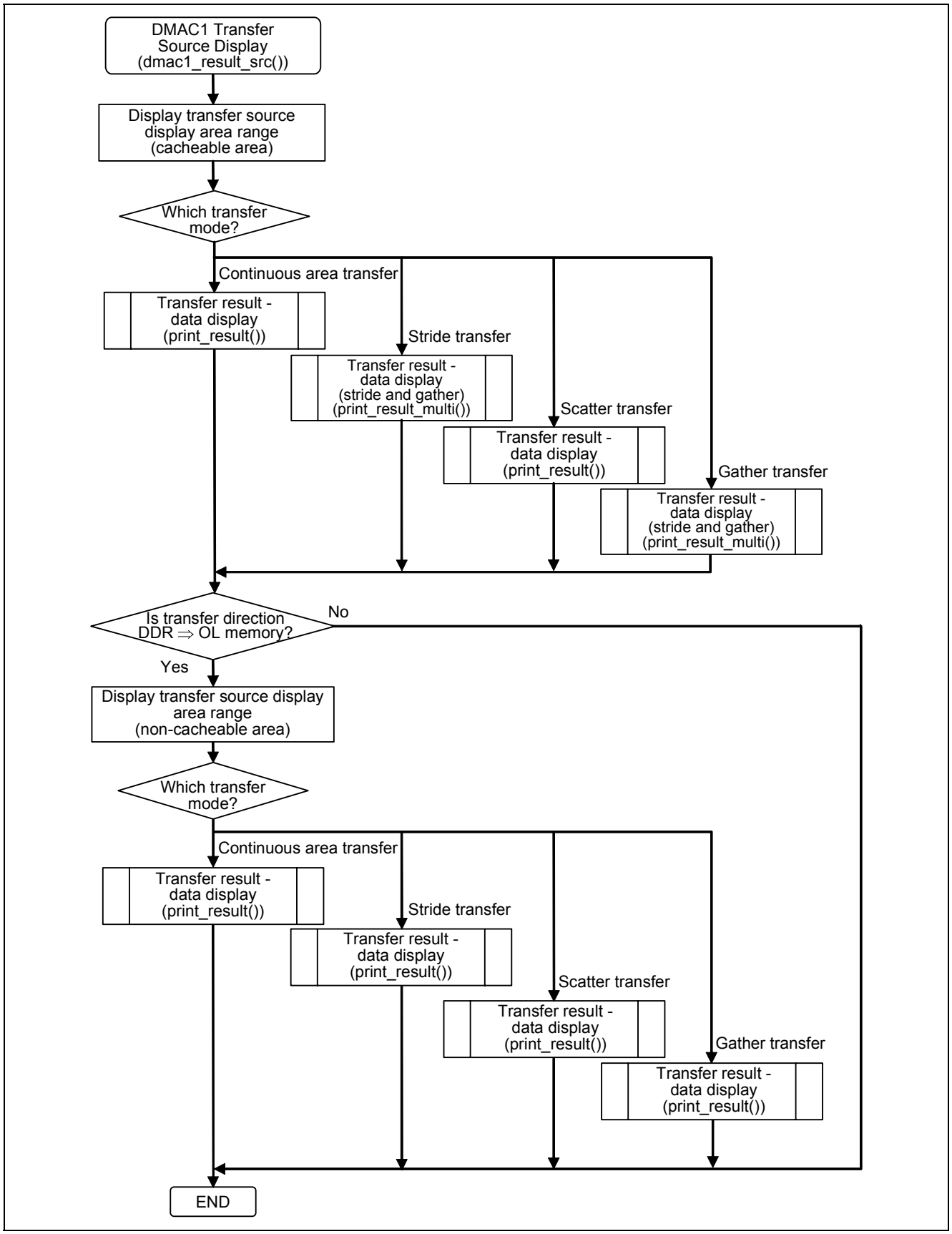


Figure 5.50 DMAC1 Transfer Source Display Flowchart

5.3.17 DMAC1 Transfer Destination Display (dmac1_result_dst())

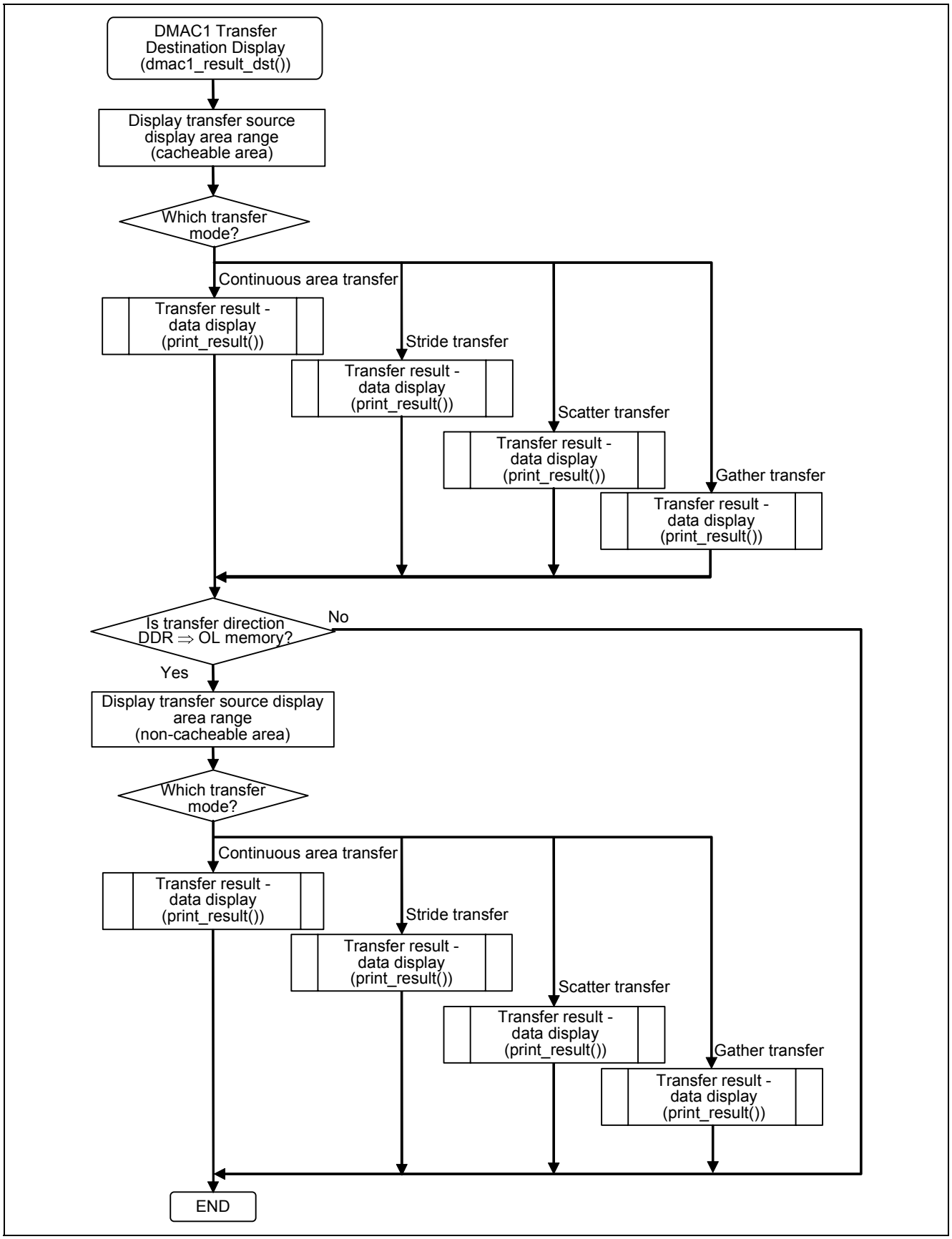


Figure 5.51 DMAC1 Transfer Destination Display Flowchart

5.4 HPB-DMAC Processing Procedures

The following figures show the flowcharts for the HPB-DMAC processing procedures.

5.4.1 HPB-DMAC Select Direction (hpbdmac_select_direction())

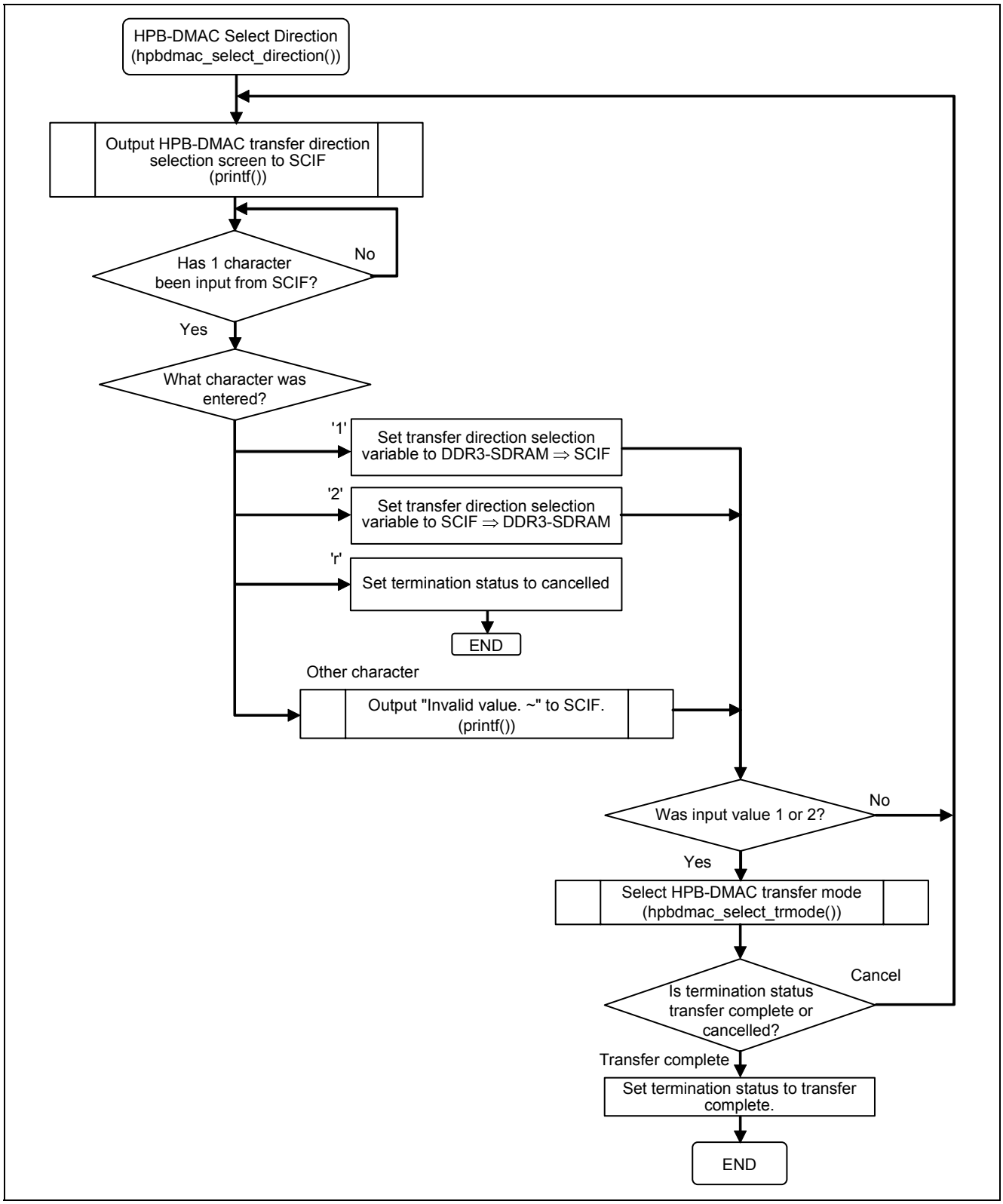


Figure 5.52 HPB-DMAC Select Direction Flowchart

5.4.2 HPB-DMAC Select Transfer Mode (hpbdmac_select_tmode())

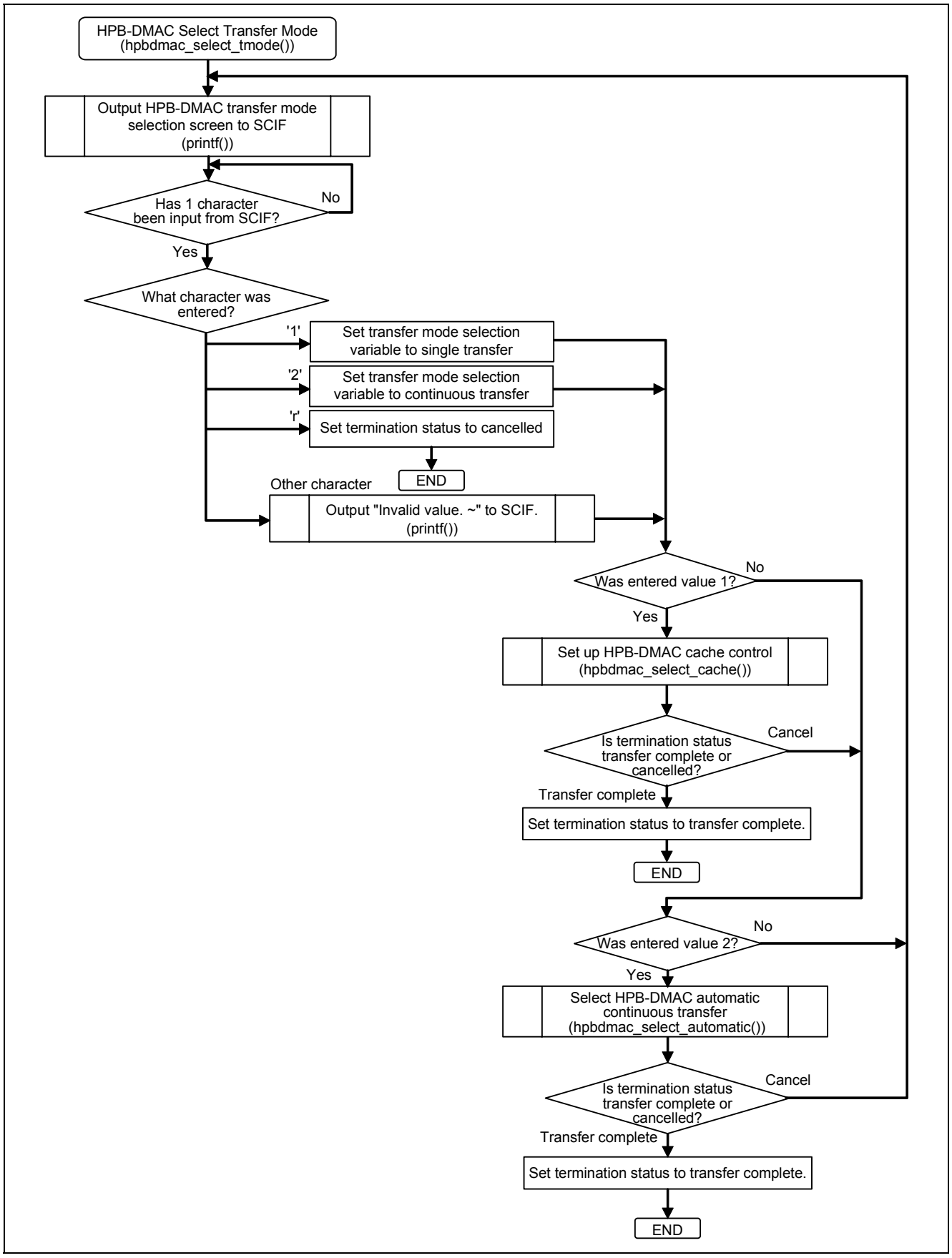


Figure 5.53 HPB-DMAC Transfer Mode Selection Flowchart

5.4.3 HPB-DMAC Automatic Continuous Transfer Setup (hpbdmac_select_automatic())

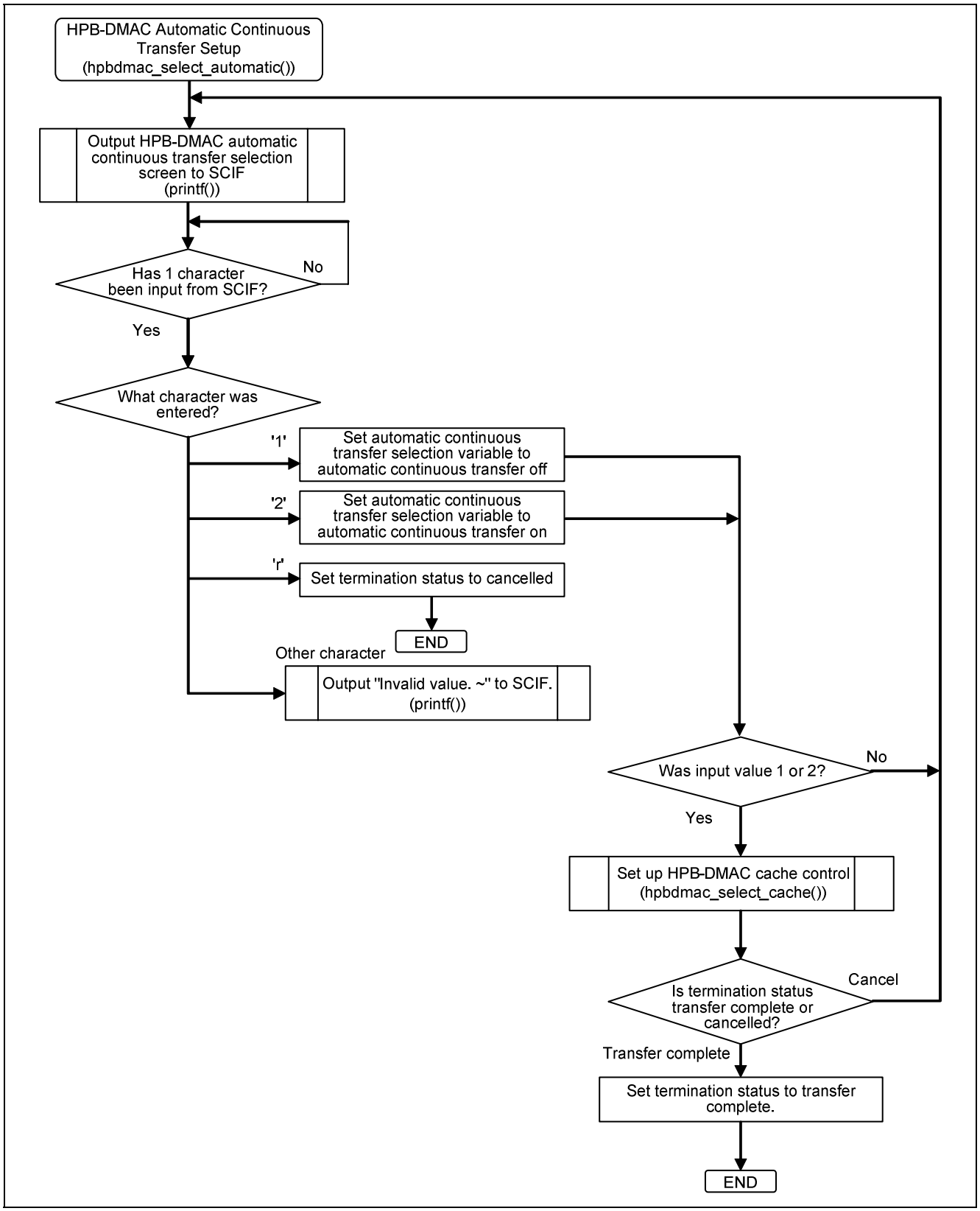


Figure 5.54 HPB-DMAC Automatic Continuous Transfer Setup Flowchart

5.4.4 HPB-DMAC Select Cache Control (hpbdmac_select_cache())

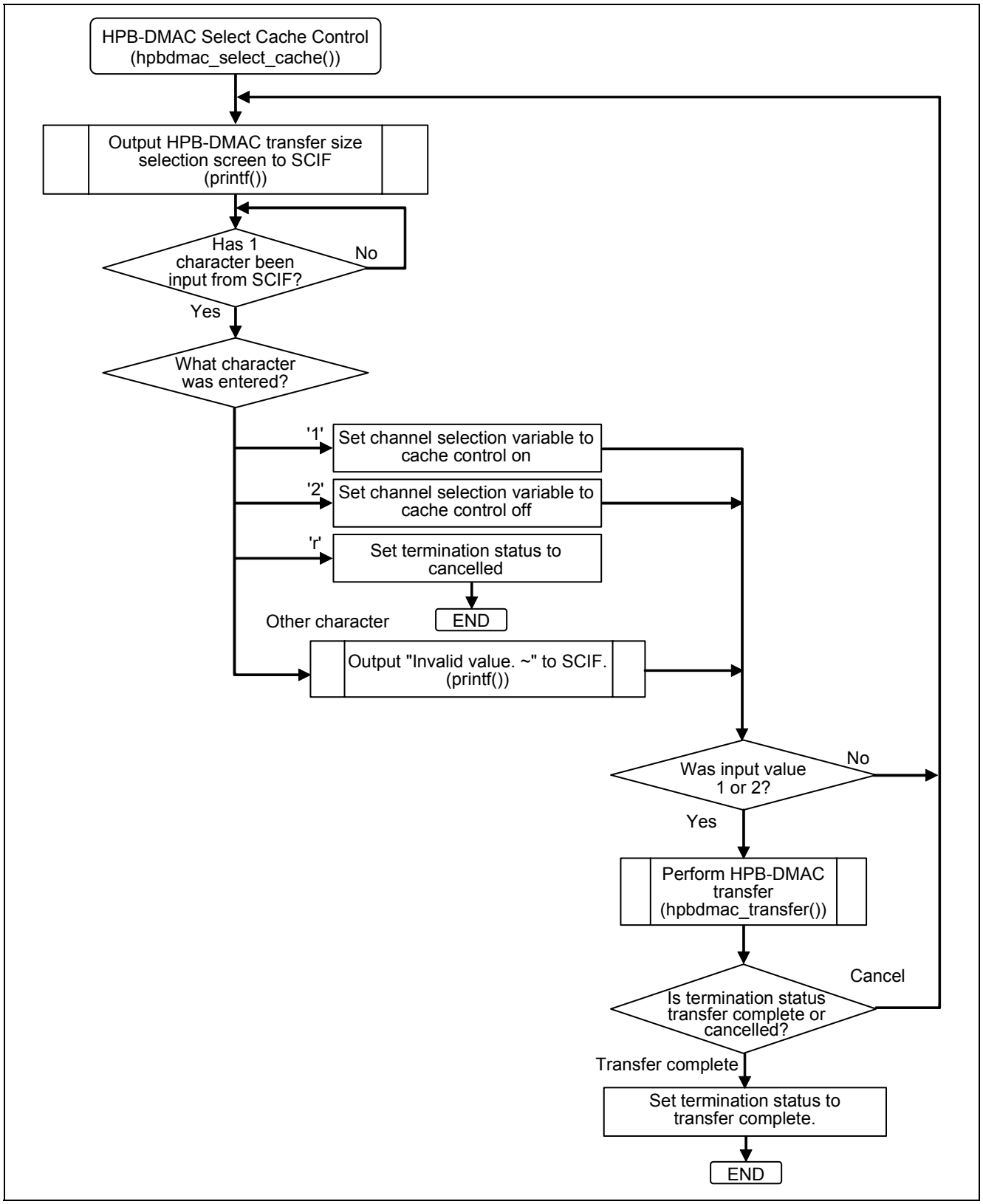


Figure 5.55 HPB-DMAC Cache Control Selection Flowchart

5.4.5 HPB-DMAC Transfer (hpbdmac_transfer())

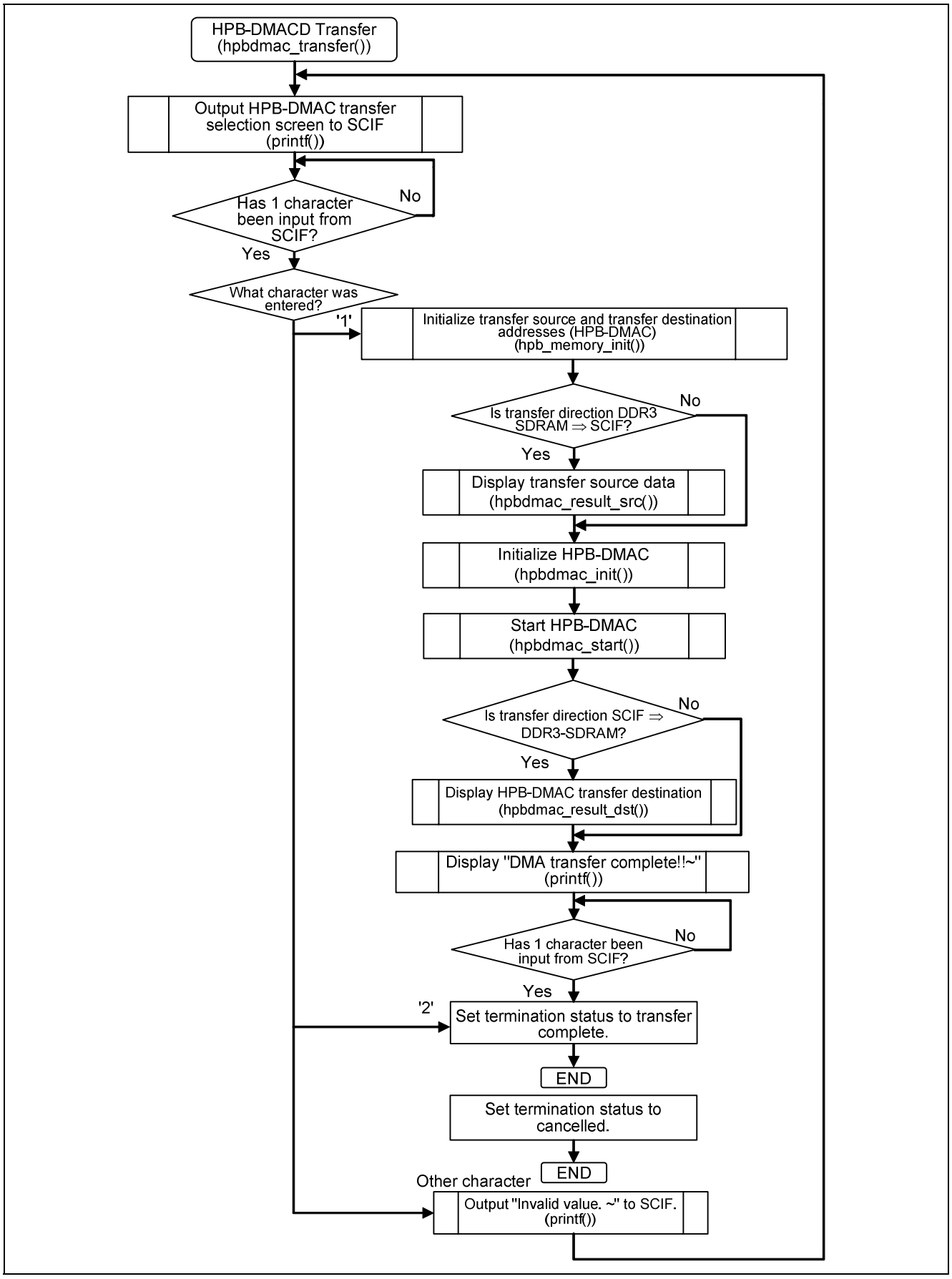


Figure 5.56 HPB-DMAC Transfer Flowchart

5.4.6 Transfer Source and Transfer Destination Initialization (HPB-DMAC) (hpb_memory_init())

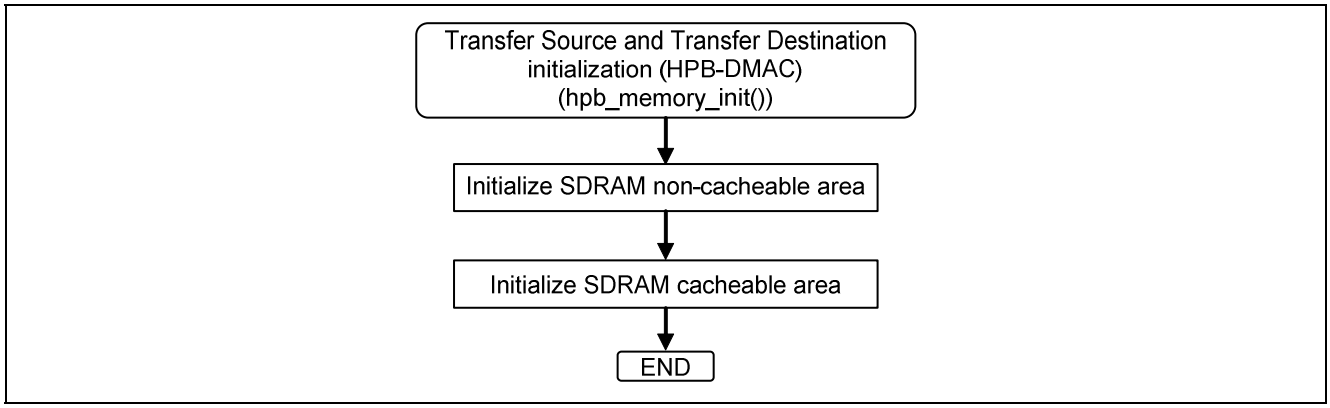


Figure 5.57 Transfer Source and Transfer Destination Initialization (HPB-DMAC) Flowchart

5.4.7 HPB-DMAC Transfer Source Data Display (hpbdmac_result_src())

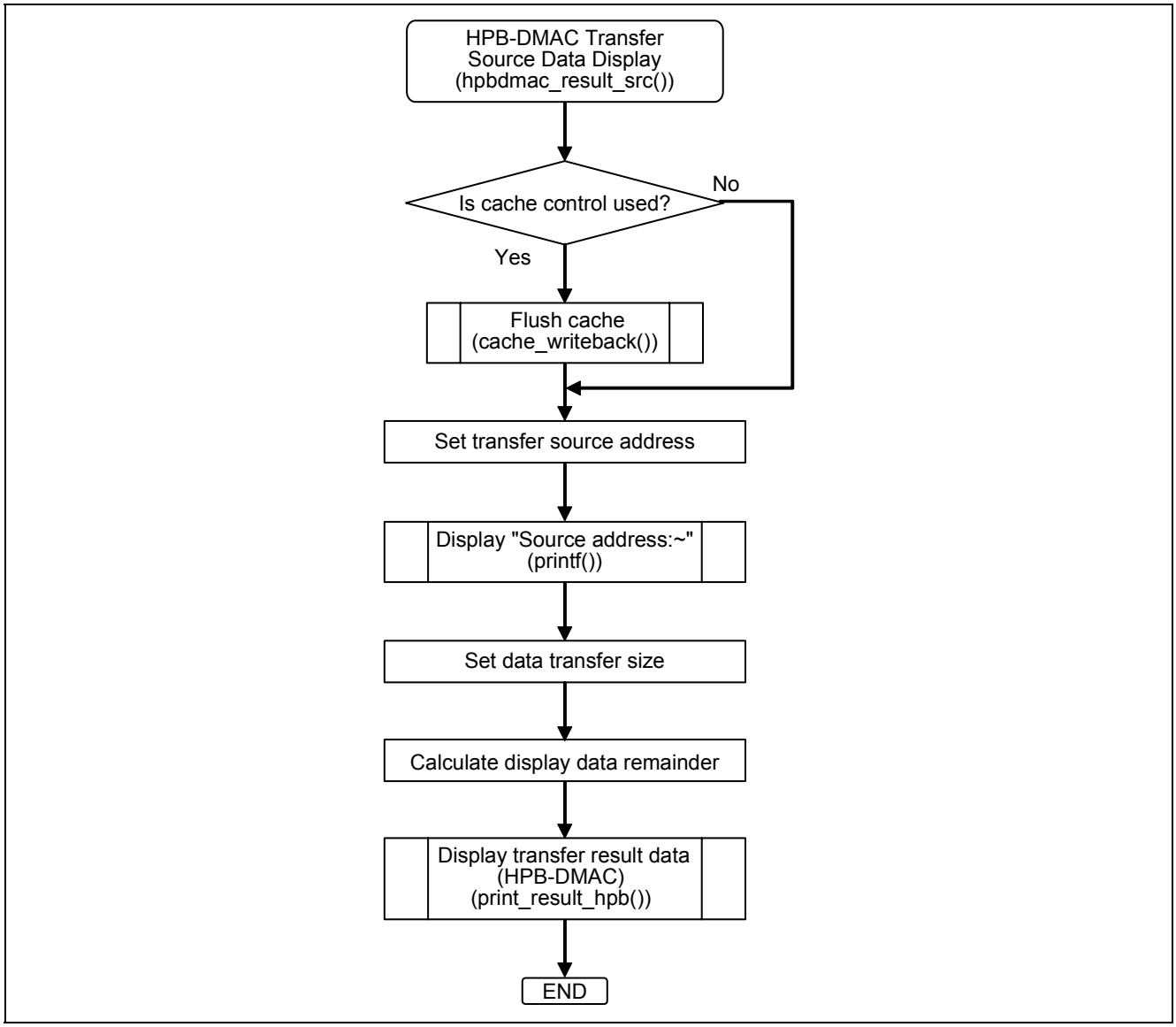


Figure 5.58 HPB-DMAC Transfer Source Data Display Flowchart

5.4.8 HPB-DMAC Transfer Destination Data Display (hpbdmac_result_src())

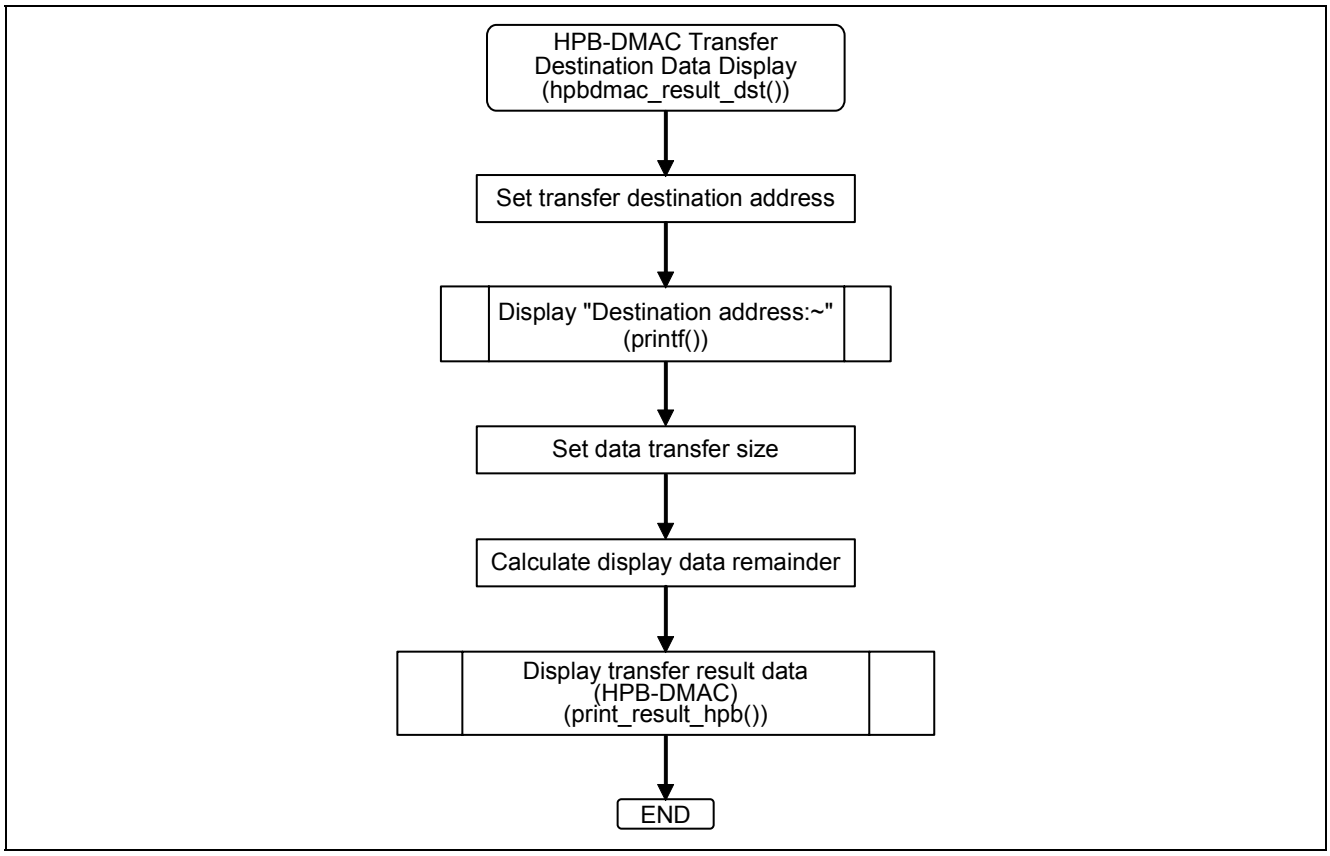


Figure 5.59 HPB-DMAC Transfer Destination Data Display Flowchart

5.4.9 HPB-DMAC Initialization (hpbdmac_init())

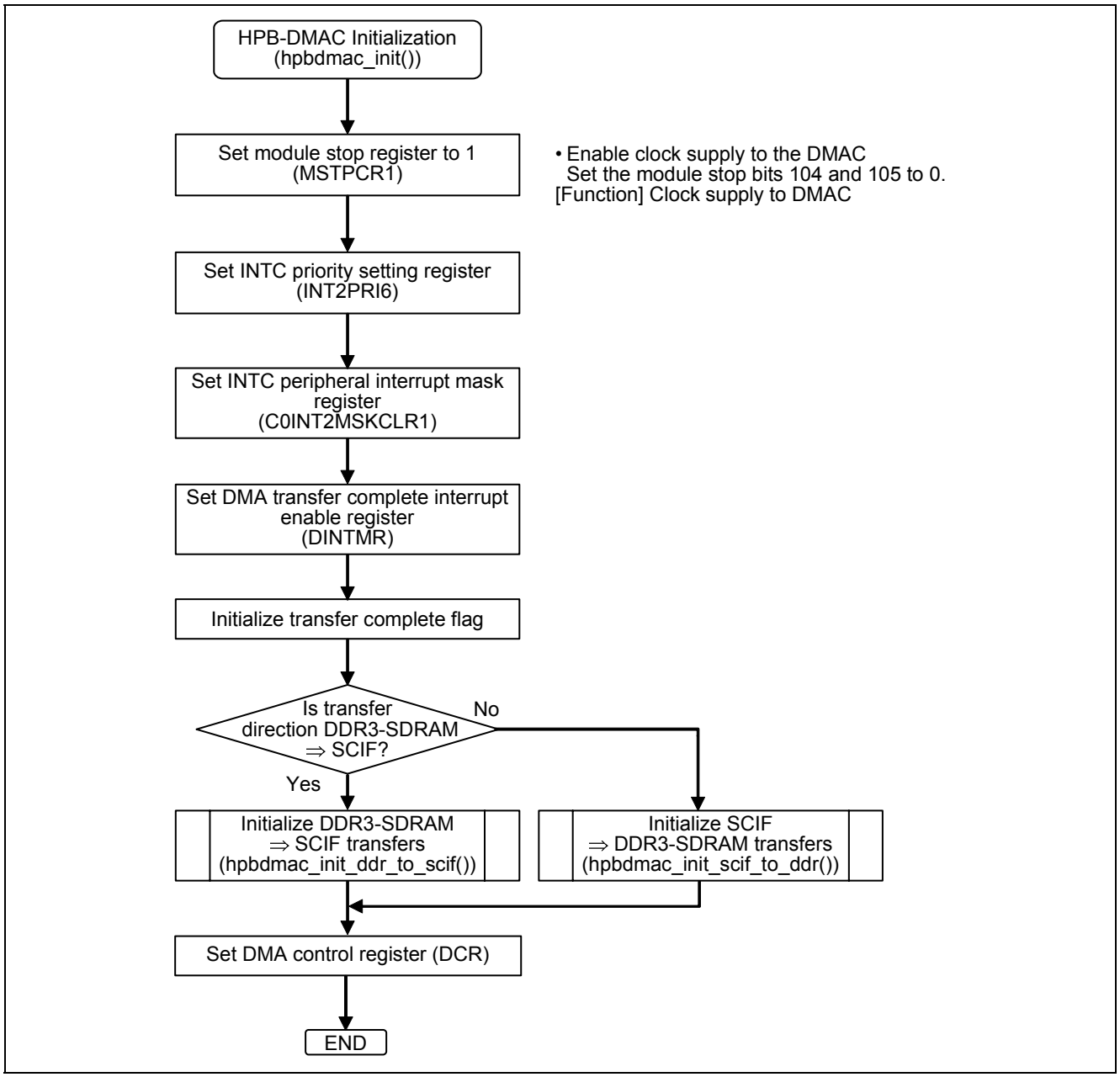


Figure 5.60 HPB-DMAC Initialization Flowchart

5.4.10 HPB-DMAC DDR3-SDRAM ⇔ SCIF Initialization (hpbdmac_init_ddr_to_scif(), hpbdmac_init_scif_to_ddr())

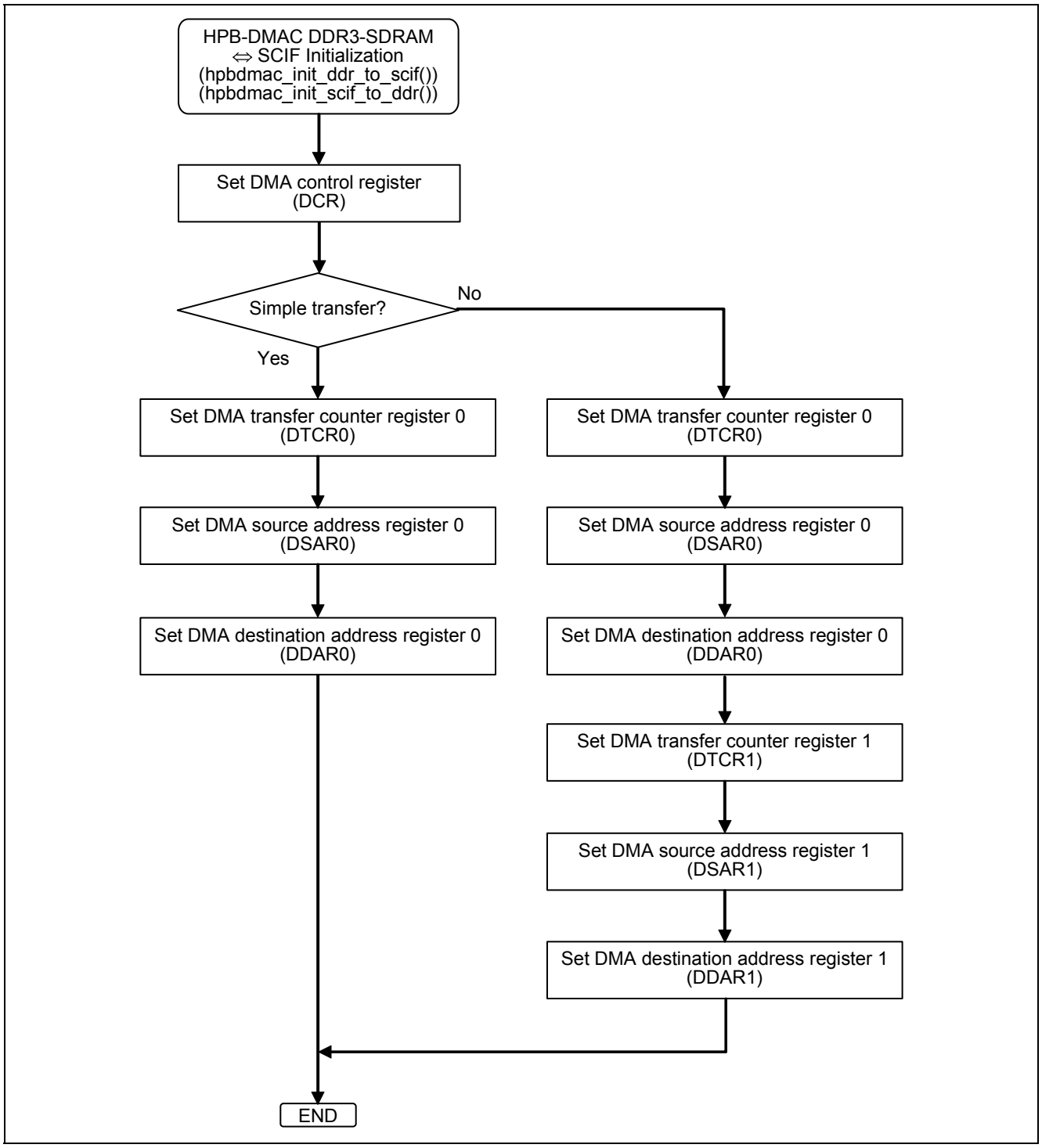


Figure 5.61 HPB-DMAC DDR3-SDRAM ⇔ SCIF Initialization Flowchart

5.4.11 HPB-DMAC Start (hpbdmac_start())

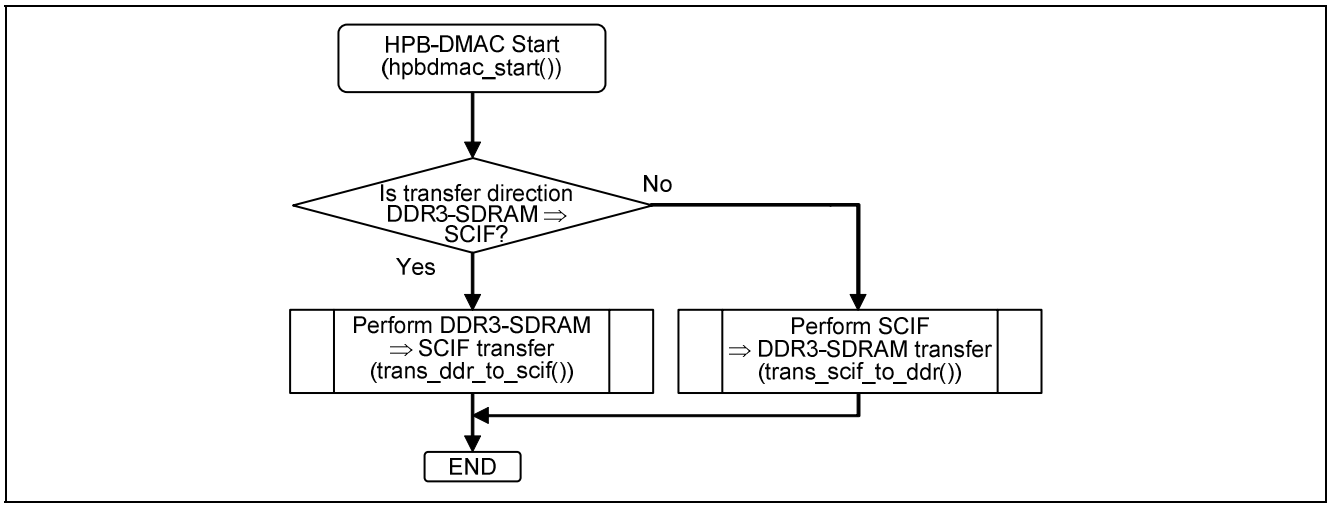


Figure 5.62 HPB-DMAC Start Flowchart

5.4.12 DDR3-SDRAM ⇒ SCIF Transfer (trans_ddr_to_scif())

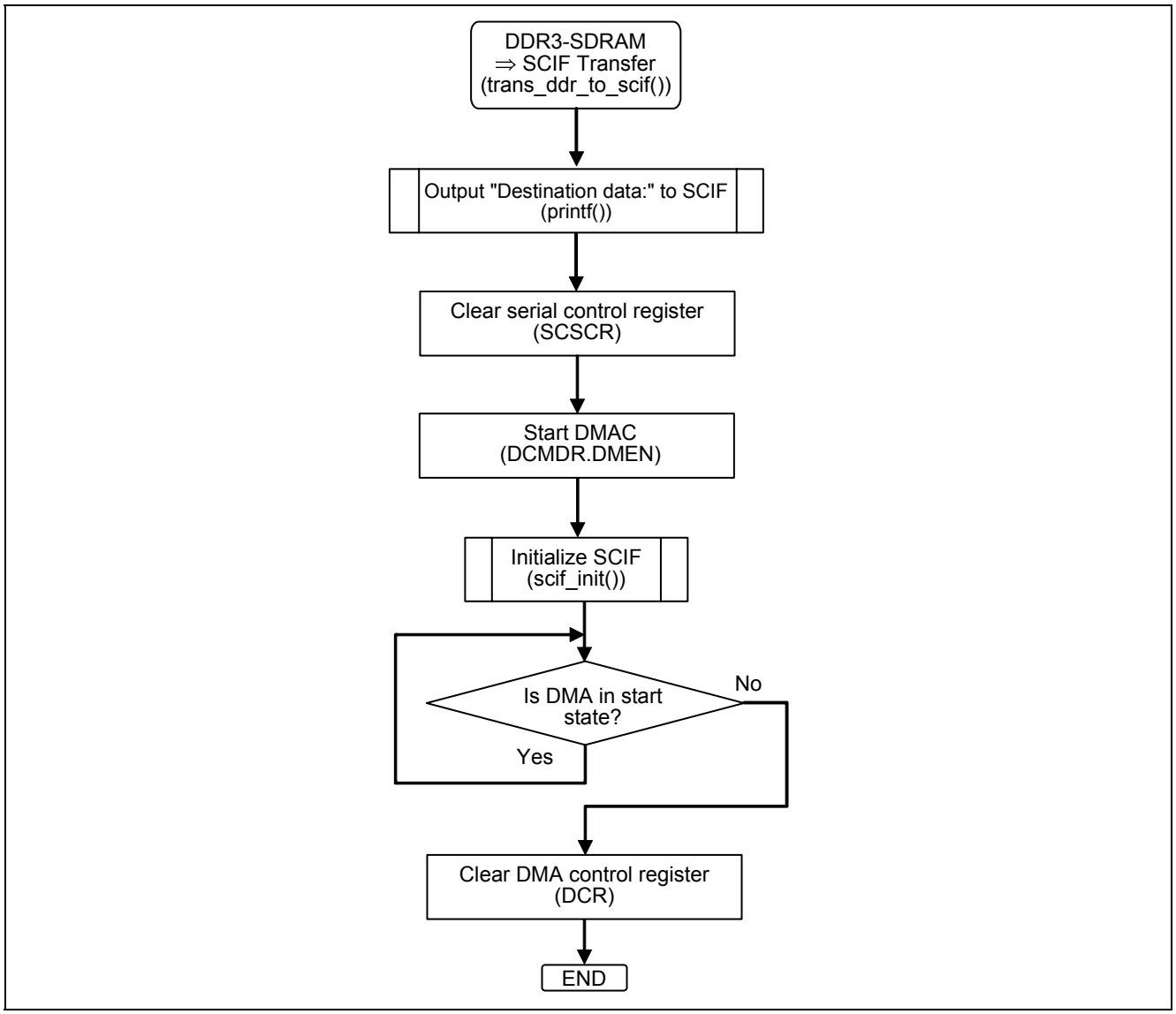


Figure 5.63 DDR3-SDRAM ⇒ SCIF Transfer Flowchart

5.4.13 SCIF ⇒ DDR3-SDRAM Transfer (trans_scif_to_ddr())

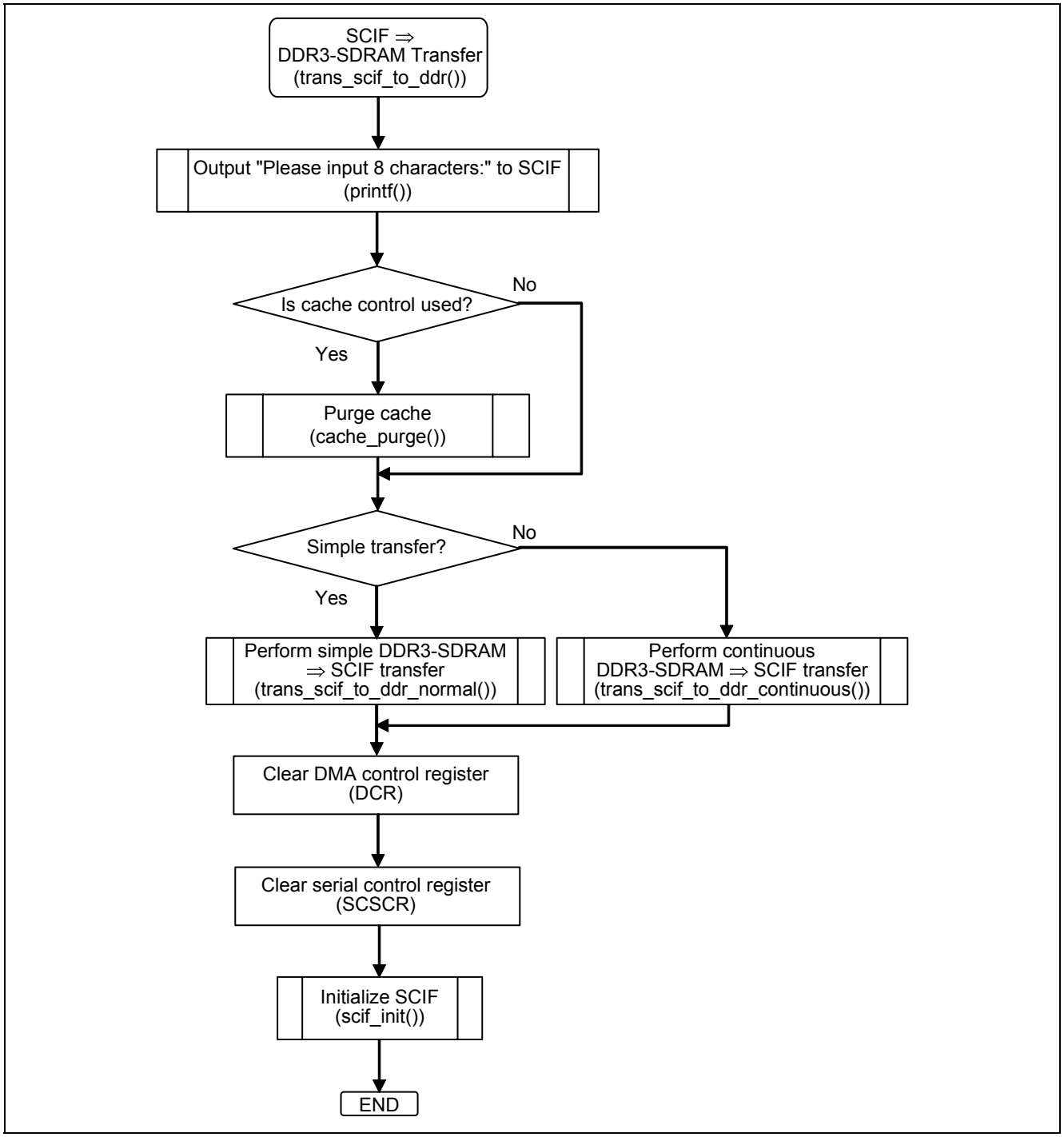


Figure 5.64 SCIF ⇒ DDR3-SDRAM Transfer Flowchart

5.4.14 Simple SCIF ⇒ DDR3-SDRAM Transfer (trans_scif_to_ddr_normal())

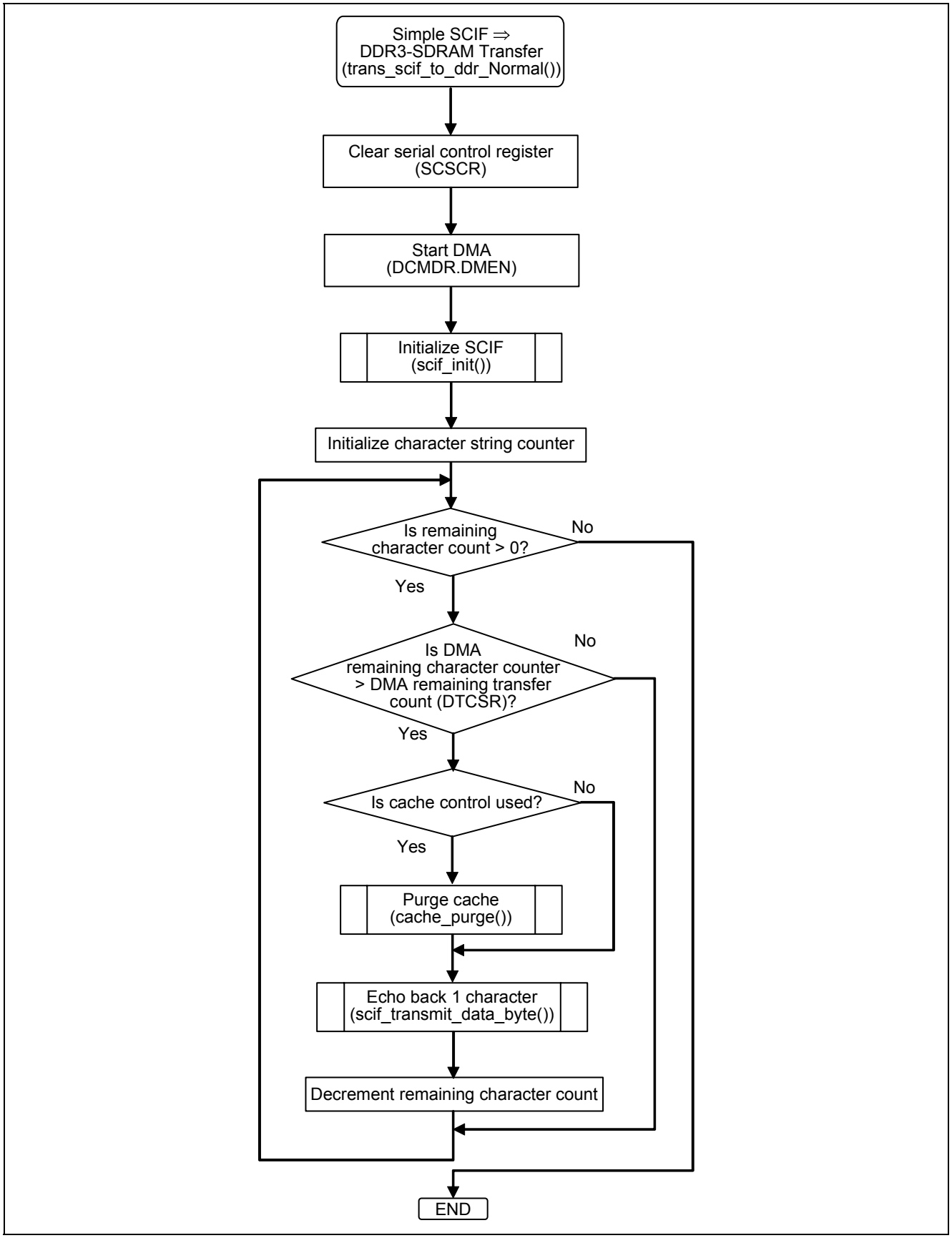


Figure 5.65 Simple SCIF ⇒ DDR3-SDRAM Transfer Flowchart

5.4.15 SCIF ⇒ DDR3-SDRAM Continuous Transfer (trans_scif_to_ddr_continuous())

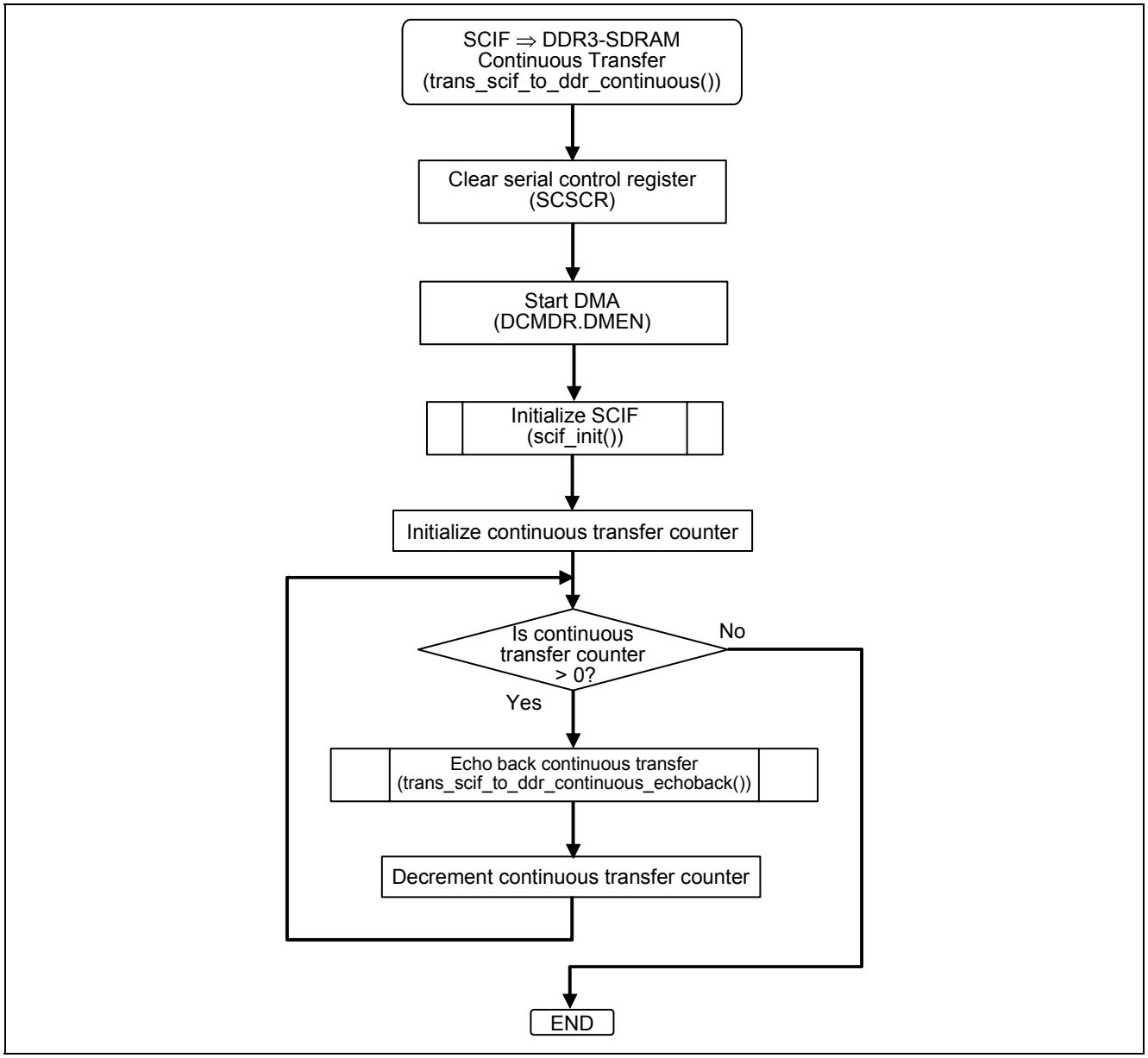


Figure 5.66 SCIF ⇒ DDR3-SDRAM Continuous Transfer Flowchart

5.4.16 Continuous Transfer Echo Back (trans_scif_to_ddr_continuous_echoback())

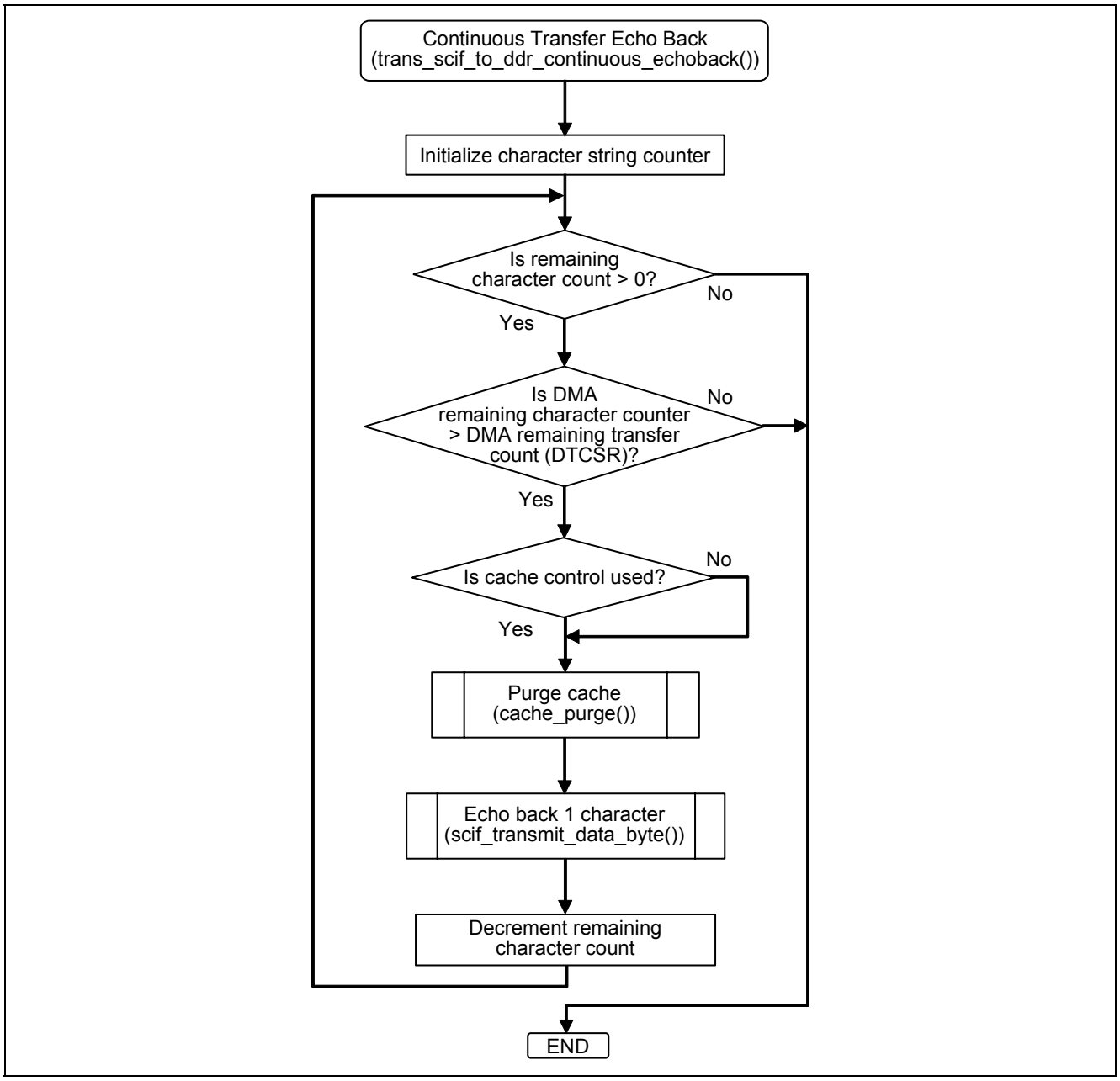


Figure 5.67 Continuous Transfer Echo Back Flowchart

5.4.17 HPB-DMAC Interrupt Handler (hpbdmac_interrupt())

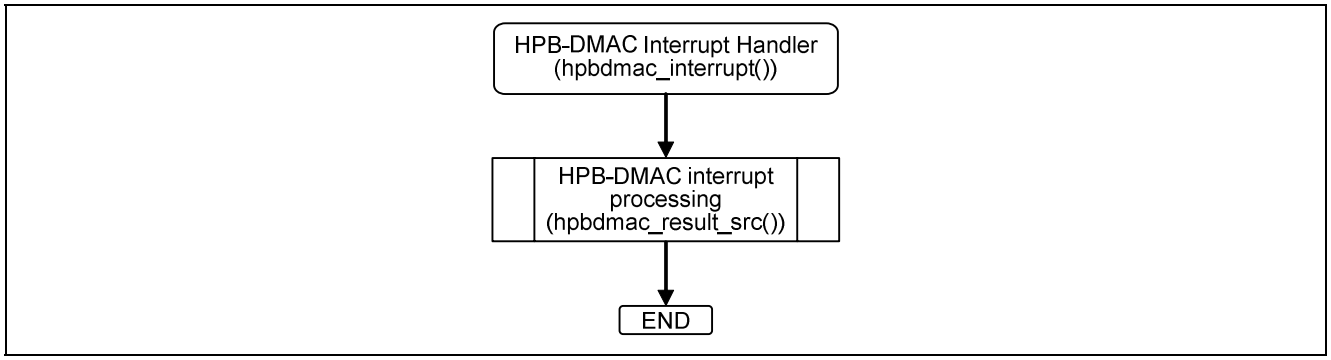


Figure 5.68 HPB-DMAC Interrupt Handler Flowchart

5.4.18 HPB-DMAC Interrupt Processing (hpbdmac_result_src())

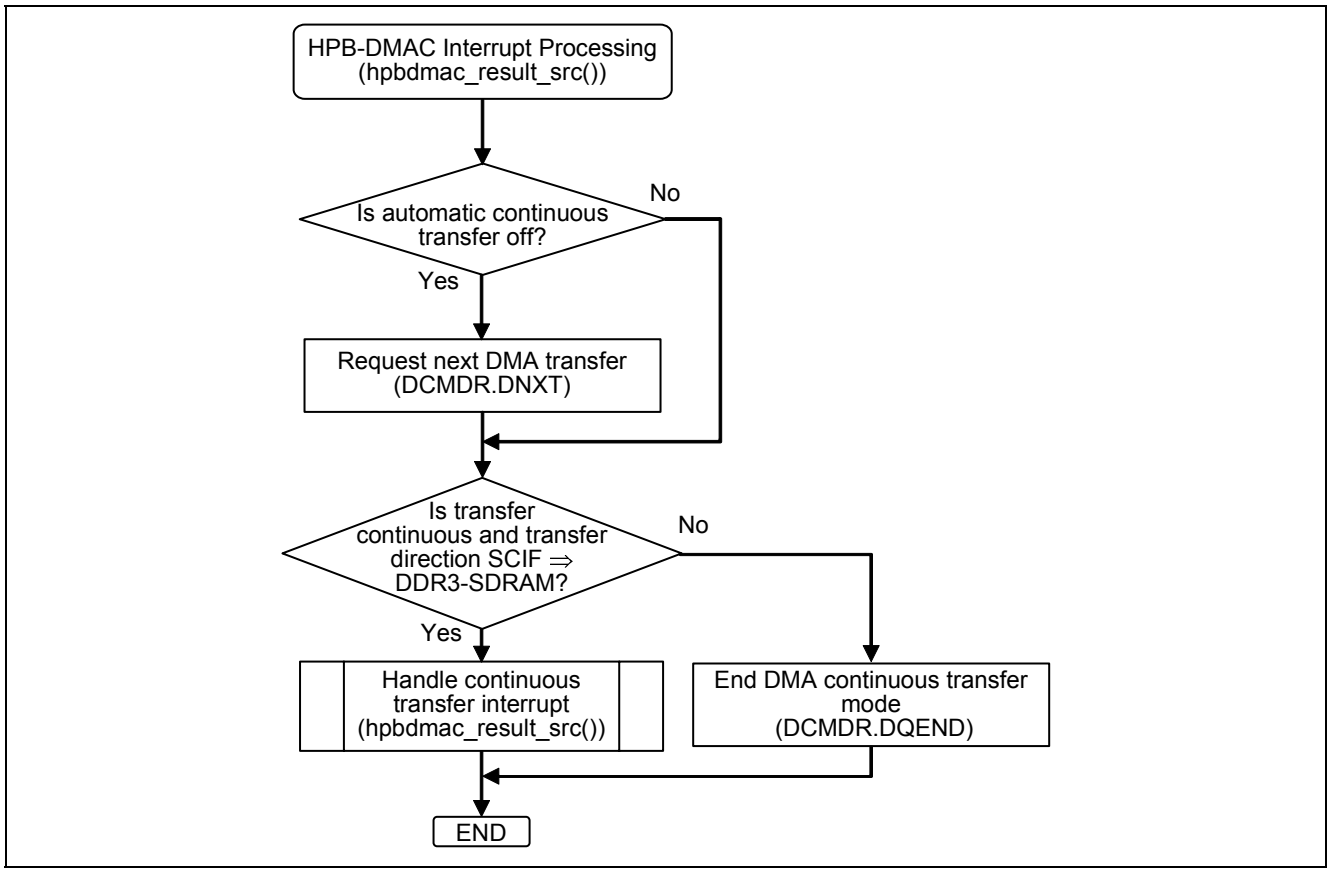


Figure 5.69 HPB-DMAC Interrupt Processing Flowchart

5.4.19 Handle Continuous Transfer Interrupt (hpbdmac_result_src())

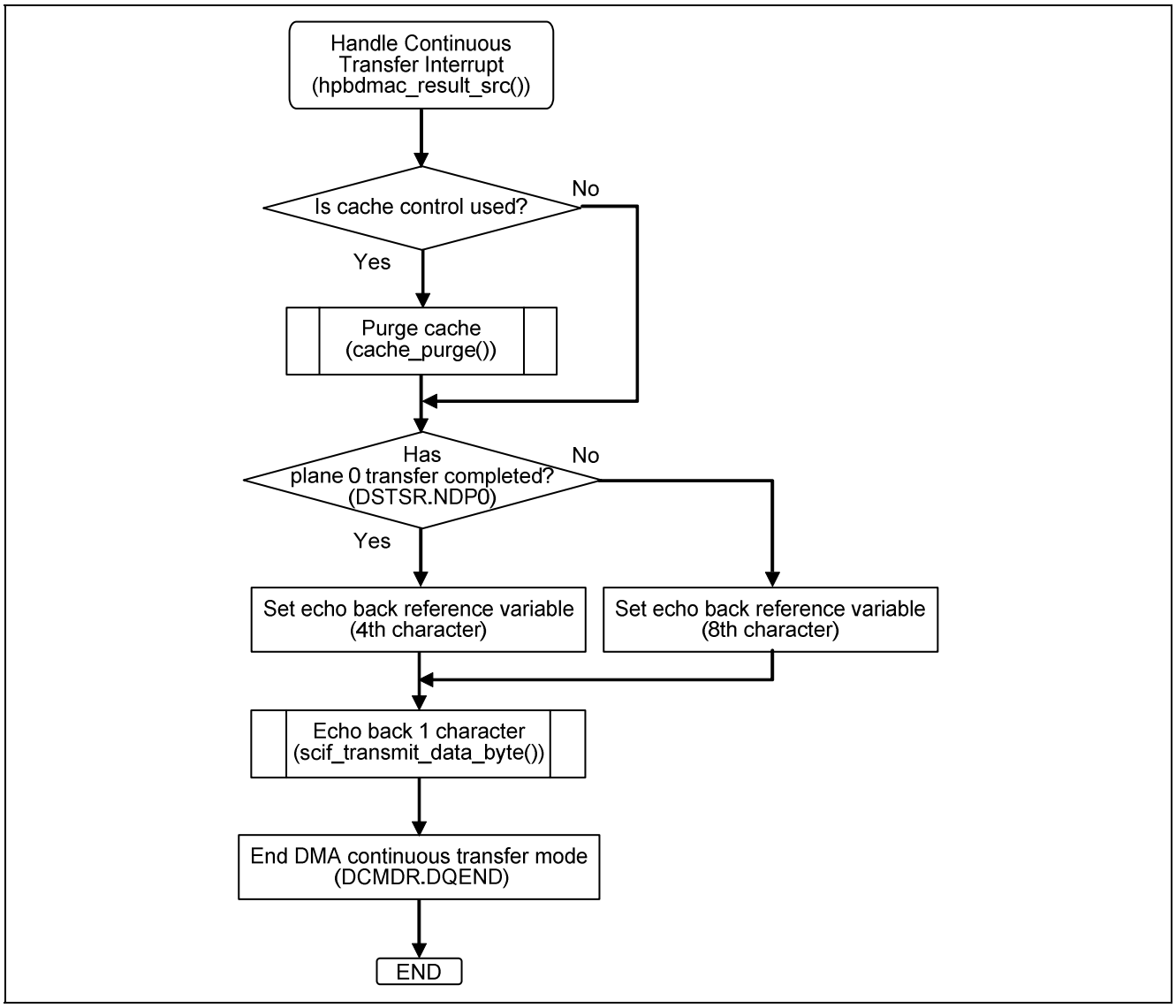


Figure 5.70 Handle Continuous Transfer Interrupt Flowchart

6. Sample Program

The sample program code is shown below.

6.1 Sample Program Listing: sh7786_DMAC_sample.c

```
001 /***********************************************************************
002 ;* DISCLAIMER
003 ;
004 ;* This software is supplied by Renesas Electronics Corporation. and is only
005 ;* intended for use with Renesas products. No other uses are authorized.
006 ;
007 ;* This software is owned by Renesas Electronics Corporation. and is protected under
008 ;* all applicable laws, including copyright laws.
009 ;
010 ;* THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES
011 ;* REGARDING THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY,
012 ;* INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR
013 ;* A PARTICULAR PURPOSE AND NON-INFRINGEMENT.  ALL SUCH WARRANTIES ARE
014 ;* EXPRESSLY DISCLAIMED.
015 ;
016 ;* TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
017 ;* ELECTRONICS CORPORATION. NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
018 ;* FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
019 ;* FOR ANY REASON RELATED TO THE THIS SOFTWARE, EVEN IF RENESAS OR ITS
020 ;* AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
021 ;
022 ;* Renesas reserves the right, without notice, to make changes to this
023 ;* software and to discontinue the availability of this software.
024 ;* By using this software, you agree to the additional terms and
025 ;* conditions found by accessing the following link:
026 ;* http://www.renesas.com/disclaimer
027 ;***********************************************************************/
028 /* Copyright (C) 2011. Renesas Electronics Corporation., All Rights Reserved.*/
029 /*""FILE COMMENT""*********** Technical reference data ****************
030 ;* System Name  : SH7786 DMAC Sample Program
031 ;* File Name    : sh7786_DMAC_sample.c
032 ;* Abstract     : Main Program
033 ;* Version      : Ver 1.00
034 ;* Device       : SH7786
035 ;* Tool-Chain   : High-performance Embedded Workshop (Version 4.09.00.007)
036 ;*              : C/C++ Compiler Package for SuperH Family (V.9.3.2.0)
037 ;* OS           : None
038 ;* H/W Platform : SH-4A Board P/N:AP-SH4AD-0A (Manufacturer:ALPHA PROJECT)
039 ;* Description  : Main routine and common functions
040 ;* Operation    :
041 ;* Limitation   :
042 ;*              :
043 ;***********************************************************************
044 ;* History      : 26.Aug.2011 Ver. 1.00 First Release
045 ;*""FILE COMMENT END""*************************************************/
046
047 #include "config.h"
048 #include "dmac.h"
049
050 /* ==== Function declaration ==== */
051 void pfc_init(void);
```

```
052 void memory_init(char dir);
053 void print_result(int, struct result, unsigned char *);
054 void print_result_hpb(char, struct result, unsigned char *);
055 void print_result_multi(char, struct result, unsigned char *);
056
057 /*""FUNC COMMENT""*****************************************************
058 * ID                :
059 * Outline           : DMAC sample program main
060 * Declaration       : void main(void)
061 * Description       : select the DMAC
062 * Argument          : none
063 * Return Value      : none
064 * Calling Functions :
065 *""FUNC COMMENT END""*************************************************/
066 void main(void)
067 {
068   char KeyBuff;
069
070   /* pin function init */
071   pfc_init();
072   /* SCIF init */
073   scif_init();
074
075   while (1) {
076      /* DMAC output selection screen */
077      printf("[DMAC operating sample plogram]\n\r");
078      printf("- DMAC select\n\r");
079      printf("  1. DMAC0\n\r");
080      printf("  2. DMAC1\n\r");
081      printf("  3. HPB-DMAC\n\r");
082      printf("  Select No:");
083
084      /* clear key buffer */
085      KeyBuff = 0;
086
087      /* waiting for input from SCIF */
088      while( scif_recive_data_byte( &KeyBuff ) != 0)
089      ;
090
091      printf("\n\r");
092
093      /* judgment of input characters */
094      switch (KeyBuff) {
095         case '1' :
096            /* DMAC0 channel selection */
097            dmac0_select_channel();
098            break;
099         case '2' :
100            /* DMAC1 channel selection */
101            dmac1_select_channel();
102            break;
103         case '3' :
104            /* HPB-DMAC direction selection */
105            hpbdmac_select_direction();
106            break;
107         default :
108            /* selecting an invalid value */
```

```
109             printf("Invalid value. Please selected 1 to 3.\n\r");
110             break;
111         }
112     }
113 }
114 /*""FUNC COMMENT""*****************************************************
115 * ID                :
116 * Outline           : pin function init
117 * Declaration       : void pfc_init(void)
118 * Description       : set PH0 and PH1 to SCIF mode
119 * Argument          : none
120 * Return Value      : none
121 * Calling Functions :
122 *""FUNC COMMENT END""*************************************************/
123 void pfc_init(void)
124 {
125   /* set PH0 and PH1 to SCIF mode */
126   GPIO.PHCR.WORD = 0xFFF0;
127 }
128
129 /*""FUNC COMMENT""*****************************************************
130 * ID                :
131 * Outline           : memory Initialization
132 * Include           :
133 * Declaration       : void memory_init(char dir)
134 * Description       : DDR3SDRAM and OL memory initialization
135 * Argument          : char dir:Transfer direction
136 * Return Value      : none
137 * Calling Functions :
138 *""FUNC COMMENT END""*************************************************/
139 void memory_init(char dir)
140 {
141   int i;
142   unsigned char *src,*dst,*p1,*p2;
143
144   p1 = D_DMAC_SDRAM_VLADR;            /* DDR3SDRAM P1 Area Address set */
145   p2 = (unsigned char *)(p1 + 0x20000000);   /* DDR3SDRAM P2 Area Address set */
146
147   for(i = 0; i < 384; i++)
148   {
149      *p1++ = 0x55;                    /* Initialization of the P1 area */
150      *p2++ = 0x55;                    /* Initialization of the P2 area */
151   }
152
153   if (dir == OL_TO_DDR) {
154      src = D_DMAC_OL_VLADR;              /* OL memory Address set */
155      dst = D_DMAC_SDRAM_VLADR;           /* DDR3SDRAM Address set */
156   } else {
157      src = D_DMAC_SDRAM_VLADR;           /* DDR3SDRAM Address set */
158      dst = D_DMAC_OL_VLADR;              /* OL memory Address set */
159   }
160
161   for(i = 0; i < 256; i++)
162   {
163      *src++ = i;       /* Initialization of the source address(0x00 to 0x80) */
164      *dst++ = 0x55;    /* Initialization of the destination address */
165   }
```

```
166
167   for(i = 0; i < 128; i++)
168   {
169      *src++ = i;        /* Initialization of the source address(0x00 to 0xFF) */
170      *dst++ = 0x55;     /* Initialization of the destination address */
171   }
172
173   p1 = COMMAND_CHAIN_VLADDR_1;      /* set the address of the command chain 1 */
174   for (i = 0;i < 32 ; i++)
175      *p1++ = 0x55;                  /* Initialization of the command chain 1 */
176
177   p1 = COMMAND_CHAIN_VLADDR_2;      /* set the address of the command chain 2 */
178   for (i = 0;i < 32 ; i++)
179      *p1++ = 0x55;                  /* Initialization of the commandc hain 2 */
180 }
181
182 /*""FUNC COMMENT""*****************************************************
183 * ID                :
184 * Outline           : show the transfer results (Hexadecimal)
185 * Declaration: void print_result(int size, struct result result_data, unsigned char *adr)
186 * Description       : transfer results of DMAC into the SCIF
187 * Argument          : int tr_size           : transfer size
188 *             : struct result result_data: structure for transfer results
189 *             : unsigned char * adr    : address for the showing
190 * Return Value      : none
191 * Calling Functions :
192 *""FUNC COMMENT END""*************************************************/
193 void print_result(int tr_size, struct result result_data, unsigned char *adr)
194 {
195   int i,j;
196
197   /* loop line */
198   for(i = 0; i < result_data.trans_size / NEWLINE; i++) {
199      printf("\n\r");
200      printf("  ");
201      /* loop column */
202      for (j = 0; j < NEWLINE;j++ ){
203         /* Space is output at intervals of tr_size */
204         if ((j % tr_size) == 0)
205            printf("  ");
206         /* Transfer Result(Hexadecimal) */
207         printf("%02X", *(adr + (i * NEWLINE + j)));
208      }
209   }
210
211   /* Is there a fraction? */
212   if (result_data.fraction != 0) {
213      printf("\n\r");
214      printf("  ");
215      for (j = 0; j < result_data.fraction;j++ ){
216         /* Space is output at intervals of tr_size */
217         if ((j % tr_size) == 0)
218            printf("  ");
219         /* Transfer Result(Hexadecimal) */
220         printf("%02X", *(adr + (i * NEWLINE + j)));
221      }
222   }
```

```
223 }
224
225 /*""FUNC COMMENT""*****************************************************
226 * ID                :
227 * Outline           : show the transfer results (ASCII)
228 * Declaration: void print_result_hpb(char tr_size, struct result result_data, char *adr)
229 * Description       : transfer results of DMAC into the SCIF
230 * Argument          : int tr_size           : transfer size
231 *             : struct result result_data: structure for transfer results
232 *             : unsigned char * adr    : address for the showing
233 * Return Value      : none
234 * Calling Functions :
235 *""FUNC COMMENT END""*************************************************/
236 void print_result_hpb(char tr_size, struct result result_data, unsigned char *adr)
237 {
238   int i,j;
239
240   /* loop line */
241   for(i = 0; i < result_data.trans_size / NEWLINE; i++) {
242      printf("\n\r");
243      printf("  ");
244      /* loop column */
245      for (j = 0; j < NEWLINE;j++ ){
246         /* Space is output at intervals of tr_size */
247         if ((j % tr_size) == 0)
248            printf("  ");
249         /* Transfer Result(ASCII) */
250         printf("%c", *(adr + (i * NEWLINE + j)));
251      }
252   }
253
254   /* Is there a fraction? */
255   if (result_data.fraction != 0) {
256      printf("\n\r");
257      printf("  ");
258      for (j = 0; j < result_data.fraction;j++ ){
259         /* Space is output at intervals of tr_size */
260         if ((j % tr_size) == 0)
261            printf("  ");
262         /* Transfer Result(ASCII) */
263         printf("%c", *(adr + (i * NEWLINE + j)));
264      }
265   }
266 }
267
268 /*""FUNC COMMENT""*****************************************************
269 * ID                :
270 * Outline           : show the transfer results (ASCII 2 characters)
271 * Declaration: void print_result_multi(char tr_size, struct result result_data,char *str)
272 * Description       : transfer results of DMAC into the SCIF
273 * Argument          : int tr_size           : transfer size
274 *             : struct result result_data: structure for transfer results
275 *             : unsigned char * adr    : address for the showing
276 * Return Value      : none
277 * Calling Functions :
278 *""FUNC COMMENT END""*************************************************/
279 void print_result_multi(char tr_size, struct result result_data, unsigned char *str)
```

```
{
  int i,j;

  /* loop line */
  for(i = 0; i < (result_data.trans_size / NEWLINE)* 2; i += 2) {
     printf("\n\r");
     printf("  ");
     /* loop column */
     for (j = 0; j < NEWLINE * 2;j += 2){
        /* Space is output at intervals of tr_size */
        if ((j % (tr_size * 2)) == 0)
           printf("  ");
        /* Transfer Result(ASCII 2 characters) */
        printf("%c%c", str[i * NEWLINE + j],str[i * NEWLINE + j + 1]);
     }
  }

  /* Is there a fraction? */
  if (result_data.fraction != 0) {
     printf("\n\r");
     printf("  ");
     for (j = 0; j < result_data.fraction * 2;j += 2){
        /* Space is output at intervals of tr_size */
        if ((j % (tr_size * 2)) == 0)
           printf("  ");
        /* Transfer Result(ASCII 2 characters) */
        printf("%c%c", str[i * NEWLINE + j],str[i * NEWLINE + j + 1]);
     }
  }
}
```

6.2 Sample Program Listing: scif.c

```
001 /***********************************************************************
002 ;* DISCLAIMER
003 ;
004 ;* This software is supplied by Renesas Electronics Corporation. and is only
005 ;* intended for use with Renesas products. No other uses are authorized.
006 ;
007 ;* This software is owned by Renesas Electronics Corporation. and is
008 ;* protected under all applicable laws, including copyright laws.
009 ;
010 ;* THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES
011 ;* REGARDING THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY,
012 ;* INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR
013 ;* A PARTICULAR PURPOSE AND NON-INFRINGEMENT.  ALL SUCH WARRANTIES ARE
014 ;* EXPRESSLY DISCLAIMED.
015 ;
016 ;* TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
017 ;* ELECTRONICS CORPORATION. NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
018 ;* FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
019 ;* FOR ANY REASON RELATED TO THE THIS SOFTWARE, EVEN IF RENESAS OR ITS
020 ;* AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
021 ;
022 ;* Renesas reserves the right, without notice, to make changes to this
023 ;* software and to discontinue the availability of this software.
024 ;* By using this software, you agree to the additional terms and
025 ;* conditions found by accessing the following link:
026 ;* http://www.renesas.com/disclaimer
027 ;***********************************************************************/
028 /* Copyright (C) 2011. Renesas Electronics Corporation., All Rights Reserved.*/
029 /*""FILE COMMENT""************ Technical reference data ****************
030 ;* System Name  : SH7786 DMAC Sample Program
031 ;* File Name    : scif.c
032 ;* Abstract     : process of SCIF
033 ;* Version      : Ver 1.00
034 ;* Device       : SH7786
035 ;* Tool-Chain   : High-performance Embedded Workshop (Version 4.09.00.007)
036 ;*              : C/C++ Compiler Package for SuperH Family (V.9.3.2.0)
037 ;* OS           : None
038 ;* H/W Platform : SH-4A Board P/N:AP-SH4AD-0A (Manufacturer:ALPHA PROJECT)
039 ;* Description  : Main routine and common functions
040 ;* Operation    :
041 ;* Limitation   :
042 ;*              :
043 ;***********************************************************************
044 ;* History      : 26.Aug.2011 Ver. 1.00 First Release
045 ;*""FILE COMMENT END""*************************************************/
046
047 #include"scif.h"
048
049 /*""FUNC COMMENT""*******************************************************
050 * ID                :
051 * Outline           : Sample Program Main
052 *                   :
053 * Include           :
054 * Declaration       : int delay( int cnt )
055 * Description       : Software weight
056 *                   : A part for the count of "cnt" and a "for" are repeated.
```

```
057 *                   :
058 *                   :
059 *                   :
060 *                   :
061 * Limitation        :
062 *                   :
063 * Argument          : cnt
064 * Return Value      : none
065 * Calling Functions :
066 *""FUNC COMMENT END""**************************************************/
067 void delay( int cnt )
068 {
069   int i;
070   for(i=0;i<cnt;i++);
071 }
072
073 /*""FUNC COMMENT""**************************************************
074 * ID                :
075 * Outline           : Sample Program Main
076 *                   :
077 * Include           :
078 * Declaration       : int scif_init(void)
079 * Description       : The initialization of SCIF
080 *                   :
081 *                   :
082 *                   :
083 *                   :
084 *                   :
085 * Limitation        :
086 *                   :
087 * Argument          : none
088 * Return Value      : -1:Baud rate clock count error
089 * Calling Functions :
090 *""FUNC COMMENT END""**************************************************/
091 int scif_init(void)
092 {
093   unsigned short data;
094   int t = -1, cnt = 0;
095
096   SCIF.SCSCR.WORD = 0x0000;   /* TIE, RIE, TE, RE Clear */
097
098   SCIF.SCFCR.BIT.TFCL = 1;/* Tx FIFO Clear */
099   SCIF.SCFCR.BIT.RFCL = 1;/* Rx FIFO Clear */
100
101   SCIF.SCFSR.WORD = 0x0000;   /* BRK, DR, TR Clear */
102   SCIF.SCLSR.BIT.ORER = 0;/* ORER Clear */
103
104 #if defined(CONFIG_SCIF_CLK_EXTERNAL)
105   SCIF.SCSCR.BIT.CKE = 2; /* Clock source:SCK */
106 #elif  defined(CONFIG_SCIF_CLK_PCLK)
107   SCIF.SCSCR.BIT.CKE = 0; /*  Clock source:PCLK */
108   t = SCBRR_VALUE(CONFIG_BPS, CONFIG_SCIF_CLK_PCLK);
109 #endif /* CONFIG_SCIF_CLK */
110
111   if(t > 0) {
112      while(t >= 256) {
113         cnt++;
114         t >> 2;
```

```
115      }
116      if(cnt > 3)
117         return -1;
118
119      SCIF.SCSMR.BIT.CKS = cnt;
120      SCIF.SCBRR = t;
121   }
122   delay(1000);
123
124   SCIF.SCFCR.BIT.RTRG = 0;
125   SCIF.SCFCR.BIT.TTRG = 0;
126   SCIF.SCFCR.BIT.TFCL = 1;/* Tx FIFO Clear */
127   SCIF.SCFCR.BIT.RFCL = 1;/* Rx FIFO Clear */
128
129   SCIF.SCFCR.BIT.TFCL = 0;/* Tx FIFO Not Clear */
130   SCIF.SCFCR.BIT.RFCL = 0;/* Rx FIFO Not Clear */
131   SCIF.SCSCR.BIT.TE  = 1;
132   SCIF.SCSCR.BIT.RE  = 1;
133   return 0;
134 }
135
136 /*""FUNC COMMENT""*****************************************************
137 * ID                :
138 * Outline           : Sample Program Main
139 *                   :
140 * Include           :
141 * Declaration       : void scif_transmit_data( char *Data )
142 * Description       : A transmission of two or more byte data of SCIF.
143 *                   :
144 *                   :
145 *                   :
146 *                   :
147 *                   :
148 * Limitation        :
149 *                   :
150 * Argument          : *Data:A send data is stored.
151 * Return Value      : none
152 * Calling Functions :
153 *""FUNC COMMENT END""*************************************************/
154 void scif_transmit_data( char*Data )
155 {
156   while( *Data )
157   {
158      while(!(SCIF.SCFSR.BIT.TDFE));  /* Weight is carried out until the write of a send data will be in an authorized state. */
159      SCIF.SCFTDR = *Data;              /* A set of a send data */
160      Data++;
161      while(!(SCIF.SCFSR.BIT.TEND));/* Waiting for the quit of transmitting*/
162      SCIF.SCFSR.BIT.TDFE = 0;
163      SCIF.SCFSR.BIT.TEND = 0;
164   }
165 }
166
167 /*""FUNC COMMENT""*****************************************************
168 * ID                :
169 * Outline           : Sample Program Main
170 *                   : (PCIe)
171 * Include           :
172 * Declaration       : void scif_transmit_byte_data( char *Data )
```

```
173 * Description       : A transmission of the single byte data of SCIF
174 *                   :
175 *                   :
176 *                   :
177 *                   :
178 *                   :
179 * Limitation        :
180 *                   :
181 * Argument          : *Data:A send data is stored.
182 * Return Value      : none
183 * Calling Functions :
184 *""FUNC COMMENT END""***************************************************/
185 void scif_transmit_data_byte( char *Data )
186 {
187       while(!(SCIF.SCFSR.BIT.TDFE));    /* Weight is carried out until the write of a send data will be in an authorized state. */
188       SCIF.SCFTDR = *Data;              /* A set of a send data */
189       while(!(SCIF.SCFSR.BIT.TEND));/* Waiting for the quit of transmitting */
190       SCIF.SCFSR.BIT.TDFE = 0;
191       SCIF.SCFSR.BIT.TEND = 0;
192 }
193
194 /*""FUNC COMMENT""*****************************************************
195 * ID                :
196 * Outline           : Sample Program Main
197 *                   : (PCIe)
198 * Include           :
199 * Declaration       : void sci_printf(char* str, ...)
200 * Description       : A text with a format is outputted.
201 *                   :
202 *                   :
203 *                   :
204 *                   :
205 *                   :
206 * Limitation        :
207 *                   :
208 * Argument          : *Data:A send data is stored.
209 * Return Value      : none
210 * Calling Functions :
211 *""FUNC COMMENT END""***************************************************/
212 /**********************************/
213 /*   A text with a format is outputted         */
214 /**********************************/
215 #define PRINTF_SIZE 1024
216 static char printf_str[PRINTF_SIZE];
217
218 void scif_printf(char* str, ...)
219 {
220   va_list args;
221   size_t size;
222
223   size = strlen(str);
224
225   if( size > PRINTF_SIZE ) {
226      return;
227   }
228
229   va_start(args, str);
230   vsprintf(printf_str, str, args);
```

```
231   va_end(args);
232
233   scif_transmit_data(printf_str);
234 }
235
236
237 /*""FUNC COMMENT""*****************************************************
238 * ID                :
239 * Outline           : Sample Program Main
240 *                   :
241 * Include           :
242 * Declaration       : char scif_recive_data_byte( char   *Data )
243 * Description       : A data reception of SCIF
244 *                   :
245 *                   :
246 *                   :
247 *                   :
248 *                   :
249 * Limitation        :
250 *                   :
251 * Argument          : *Data:A receive data is stored.
252 * Return Value      : -1:A receive data error
253 * Calling Functions :
254 *""FUNC COMMENT END""*************************************************/
255 char scif_recive_data_byte( char*Data )
256 {
257   unsigned char  ReadData, i = 0;
258   char  ret_cd = 0;
259
260   for(;;)
261   {
262      if(( SCIF.SCFSR.BIT.ER  ) ||
263         ( SCIF.SCFSR.BIT.BRK ) ||
264         ( SCIF.SCFSR.BIT.DR  ))    /* An error occurs? */
265      {
266         ReadData = SCIF.SCFRDR; /* Read of a data dummy */
267         ret_cd = -1;/* A set of a reception error */
268         SCIF.SCFSR.WORD &= 0x0000; /* A clear of an error  */
269         SCIF.SCLSR.WORD &= 0x0000;
270      }
271      else if( SCIF.SCFSR.BIT.RDF ) /* A data was received? */
272      {
273         *Data = SCIF.SCFRDR; /* A data is acquired*/
274         SCIF.SCFSR.BIT.RDF = 0; /* A clear of a receive data sign */
275         SCIF.SCFSR.BIT.DR  = 0; /* A clear of a receive data sign */
276         scif_transmit_data_byte( Data );
277         break;/* A processing is completed. */
278      }
279   }
280   return( ret_cd );
281 }
282
```

6.3 Sample Program Listing: cachecontrol.c

```
001 /***********************************************************************
002 ;* DISCLAIMER
003 ;
004 ;* This software is supplied by Renesas Electronics Corporation. and is only
005 ;* intended for use with Renesas products. No other uses are authorized.
006 ;
007 ;* This software is owned by Renesas Electronics Corporation. and is protected under
008 ;* all applicable laws, including copyright laws.
009 ;
010 ;* THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES
011 ;* REGARDING THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY,
012 ;* INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR
013 ;* A PARTICULAR PURPOSE AND NON-INFRINGEMENT.  ALL SUCH WARRANTIES ARE
014 ;* EXPRESSLY DISCLAIMED.
015 ;
016 ;* TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
017 ;* ELECTRONICS CORPORATION. NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
018 ;* FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
019 ;* FOR ANY REASON RELATED TO THE THIS SOFTWARE, EVEN IF RENESAS OR ITS
020 ;* AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
021 ;
022 ;* Renesas reserves the right, without notice, to make changes to this
023 ;* software and to discontinue the availability of this software.
024 ;* By using this software, you agree to the additional terms and
025 ;* conditions found by accessing the following link:
026 ;* http://www.renesas.com/disclaimer
027 ;***********************************************************************/
028 /* Copyright (C) 2011. Renesas Electronics Corporation., All Rights Reserved.*/
029 /*""FILE COMMENT""*********** Technical reference data ****************
030 ;* System Name  : SH7786 DMAC Sample Program
031 ;* File Name    : cachcontrol.c
032 ;* Abstract     : Caching control
033 ;* Version      : Ver 1.00
034 ;* Device       : SH7786
035 ;* Tool-Chain   : High-performance Embedded Workshop (Version 4.09.00.007)
036 ;*              : C/C++ Compiler Package for SuperH Family (V.9.3.2.0)
037 ;* OS           : None
038 ;* H/W Platform : SH-4A Board P/N:AP-SH4AD-0A (Manufacturer:ALPHA PROJECT)
039 ;* Description  : Main routine and common functions
040 ;* Operation    :
041 ;* Limitation   :
042 ;*              :
043 ;***********************************************************************
044 ;* History      : 26.Aug.2011 Ver. 1.00 First Release
045 ;*""FILE COMMENT END""*************************************************/
046
047 #include <machine.h>
048
049 #define CACHE_LINE_SIZE 32
050
051 /*""FUNC COMMENT""*****************************************************
052 * ID               :
053 * Outline          : Controls the cache
054 * Declaration      : void cache_writeback(void *start, int size)
055 * Description      : Performed to flush the cache
056 * Argument         : void *start: Start address of flush
```

```
057 *             : int size    : The amount for flush
058 * Return Value      :
059 * Calling Functions :
060 *""FUNC COMMENT END""***************************************************/
061 void cache_writeback(void *start, int size)
062 {
063   unsigned long v;
064   unsigned long begin, end;
065
066   /* Setting for address boundary */
067   begin = (unsigned long)start & ~(CACHE_LINE_SIZE-1);
068   end = ((unsigned long)start + size + CACHE_LINE_SIZE - 1) &
069         ~(CACHE_LINE_SIZE - 1);
070
071   /* Flushing the cache */
072   for (v = begin; v < end; v += CACHE_LINE_SIZE)
073      ocbwb((void *)v);
074
075 }
076
077 /*""FUNC COMMENT""*****************************************************
078 * ID                :
079 * Outline           : Controls the cache
080 * Declaration       : void cache_purge(void *start, int size)
081 * Description       : Make the cache purge
082 * Argument          : void *start: Start address of invalidate
083 *             : int size    : The amount for invalidate
084 * Return Value      :
085 * Calling Functions :
086 *""FUNC COMMENT END""***************************************************/
087 void cache_purge(void *start, int size)
088 {
089   unsigned long v;
090   unsigned long begin, end;
091
092   /* Setting for address boundary */
093   begin = (unsigned long)start & ~(CACHE_LINE_SIZE-1);
094   end = ((unsigned long)start + size + CACHE_LINE_SIZE - 1) &
095         ~(CACHE_LINE_SIZE - 1);
096
097   /* Invalidation the cache */
098   for (v = begin; v < end; v += CACHE_LINE_SIZE)
099      ocbp((void *)v);
100
101 }
```

6.4 Sample Program Listing: dmac0.c

```
0001 /***********************************************************************
0002 ;* DISCLAIMER
0003 ;
0004 ;* This software is supplied by Renesas Electronics Corporation. and is only
0005 ;* intended for use with Renesas products. No other uses are authorized.
0006 ;
0007 ;* This software is owned by Renesas Electronics Corporation. and is protected under
0008 ;* all applicable laws, including copyright laws.
0009 ;
0010 ;* THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES
0011 ;* REGARDING THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY,
0012 ;* INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
0013 ;* PARTICULAR PURPOSE AND NON-INFRINGEMENT.  ALL SUCH WARRANTIES ARE EXPRESSLY
0014 ;* DISCLAIMED.
0015 ;
0016 ;* TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
0017 ;* ELECTRONICS CORPORATION. NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
0018 ;* FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
0019 ;* FOR ANY REASON RELATED TO THE THIS SOFTWARE, EVEN IF RENESAS OR ITS
0020 ;* AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
0021 ;
0022 ;* Renesas reserves the right, without notice, to make changes to this
0023 ;* software and to discontinue the availability of this software.
0024 ;* By using this software, you agree to the additional terms and
0025 ;* conditions found by accessing the following link:
0026 ;* http://www.renesas.com/disclaimer
0027 ;***********************************************************************/
0028 /* Copyright (C) 2011. Renesas Electronics Corporation., All Rights Reserved.*/
0029 /*""FILE COMMENT""************ Technical reference data ****************
0030 ;* System Name  : SH7786 DMAC Sample Program
0031 ;* File Name    : dmac0.c
0032 ;* Abstract     : DMAC0 is transfer process
0033 ;* Version      : Ver 1.00
0034 ;* Device       : SH7786
0035 ;* Tool-Chain   : High-performance Embedded Workshop (Version 4.09.00.007)
0036 ;*              : C/C++ Compiler Package for SuperH Family (V.9.3.2.0)
0037 ;* OS           : None
0038 ;* H/W Platform : SH-4A Board P/N:AP-SH4AD-0A (Manufacturer:ALPHA PROJECT)
0039 ;* Description  : Main routine and common functions
0040 ;* Operation    :
0041 ;* Limitation   :
0042 ;*              :
0043 ;***********************************************************************
0044 ;* History      : 26.Aug.2011 Ver. 1.00 First Release
0045 ;*""FILE COMMENT END""*************************************************/
0046
0047 #include "config.h"
0048 #include "dmac0.h"
0049
0050 int int_flg;          /* Interrupt Flag */
0051 struct DMAC_0 dmac0;     /* Structure for storing configuration */
0052
0053 /*""FUNC COMMENT""*****************************************************
0054 * ID               :
0055 * Outline          : DMAC0 channel select
0056 * Declaration      : void dmac0_select_channel( void )
```

```
0057 * Description       : DMAC0 chanel select
0058 * Argument          : none
0059 * Return Value      : none
0060 * Calling Functions :
0061 *""FUNC COMMENT END""***************************************************/
0062 void dmac0_select_channel( void )
0063 {
0064  int ret;
0065  char KeyBuff;
0066
0067  /* Structure clear */
0068  dmac0_data_clear();
0069
0070  while(1){
0071     /* showing DMAC0 channel selection screen */
0072     printf("[DMAC0]\n\r");
0073     printf("  - Chanel Select\n\r");
0074     printf("  1. Chanel 0\n\r");
0075     printf("  2. Chanel 4\n\r");
0076     printf("  r. Return to previous menu\n\r");
0077     printf("  Select No:");
0078     /* clear key buffer */
0079     KeyBuff = 0;
0080
0081     /* waiting for input from SCIF */
0082     while( scif_recive_data_byte( &KeyBuff ) != 0)
0083     ;
0084
0085     printf("\n\r");
0086
0087     /* judgment of input characters */
0088     switch (KeyBuff) {
0089        /* channel 0 selected */
0090        case '1' :
0091           dmac0.ch = 0;
0092           break;
0093        /* channel 4 selected */
0094        case '2' :
0095           dmac0.ch = 4;
0096           break;
0097        /* previous menu selected */
0098        case 'r' :
0099              return;
0100        /* selecting an invalid value */
0101        default :
0102           printf("Invalid value. Please selected 1 to 2 or r.\n\r");
0103           break;
0104     }
0105
0106     /* in the case of inputting the value from '1' to '2' */
0107     if (KeyBuff >= '1' && KeyBuff <= '2') {
0108        /* DMAC0 direction to be selected */
0109        ret = dmac0_select_direction();
0110        if (ret == 0)
0111           return;
0112     }
0113  }
0114
```

```
0115 }
0116
0117 /*""FUNC COMMENT""************************************************
0118 * ID                :
0119 * Outline           : DMAC0 direction select
0120 * Declaration       : int dmac0_select_direction( void )
0121 * Description       : DMAC0 direction select
0122 * Argument          : none
0123 * Return Value      : 0:transfer end,1:canceled menu
0124 * Calling Functions :
0125 *""FUNC COMMENT END""*************************************************/
0126 int dmac0_select_direction( void )
0127 {
0128  int ret;
0129  char KeyBuff;
0130
0131  while (1){
0132     /* showing DMAC0 direction selection screen */
0133     printf("[DMAC0-ch%d]\n\r",dmac0.ch);
0134     printf("  - Direction Select\n\r");
0135     printf("  1. OL memory to DDR3-SDRAM\n\r");
0136     printf("  2. DDR3-SDRAM to OL memory\n\r");
0137     printf("  r. Return to previous menu\n\r");
0138     printf("  Select No:");
0139     /* clear key buffer */
0140     KeyBuff = 0;
0141
0142     /* waiting for input from SCIF */
0143     while( scif_recive_data_byte( &KeyBuff ) != 0)
0144     ;
0145
0146     printf("\n\r");
0147
0148     /* judgment of input characters */
0149     switch (KeyBuff) {
0150        /* OL memory to DDR3SDRAM selected */
0151        case '1' :
0152           dmac0.dir = OL_TO_DDR;
0153           break;
0154        /* DDR3SDRAM to OL memory selected */
0155        case '2' :
0156           dmac0.dir = DDR_TO_OL;
0157           break;
0158        /* previous menu selected */
0159        case 'r' :
0160           return 1;
0161        /* selecting an invalid value */
0162        default :
0163           printf("Invalid value. Please selected 1 to 2 or r.\n\r");
0164           break;
0165     }
0166
0167     /* in the case of inputting the value from '1' to '2' */
0168     if (KeyBuff >= '1' && KeyBuff <= '2') {
0169           /* DMAC0 transfer mode to be selected */
0170        ret = dmac0_select_trmode();
0171        if (ret == 0)
0172           return 0;
```

```
0173     }
0174  }
0175 }
0176
0177 /*""FUNC COMMENT""************************************************
0178 * ID                :
0179 * Outline           : DMAC0 trans mode select
0180 * Declaration       : int dmac0_select_trmode( void )
0181 * Description       : DMAC0 trans mode select
0182 * Argument          : none
0183 * Return Value      : 0:transfer end,1:canceled menu
0184 * Calling Functions :
0185 *""FUNC COMMENT END""*********************************************/
0186 int dmac0_select_trmode( void )
0187 {
0188  int ret;
0189  char KeyBuff;
0190
0191  while (1){
0192     /* showing DMAC0 transfer mode selection screen */
0193     printf("[DMAC0-ch%d-%s]\n\r",dmac0.ch,dmac0_dir_str[dmac0.dir -1]);
0194     printf("  - Transfer mode select\n\r");
0195     printf("  1. Normal\n\r");
0196     printf("  2. Repeat\n\r");
0197     printf("  3. Reload\n\r");
0198     printf("  4. Multi-dimensional mode\n\r");
0199     printf("  r. Return to previous menu\n\r");
0200     printf("  Select No:");
0201     /* clear key buffer */
0202     KeyBuff = 0;
0203
0204     /* waiting for input from SCIF */
0205     while( scif_recive_data_byte( &KeyBuff ) != 0)
0206     ;
0207
0208     printf("\n\r");
0209
0210     /* judgment of input characters */
0211     switch (KeyBuff) {
0212        /* normal mode selected */
0213        case '1' :
0214           dmac0.mode = TRANSFER_MODE_NORMAL;
0215           break;
0216        /* repeat mode selected */
0217        case '2' :
0218           dmac0.mode = TRANSFER_MODE_REPEAT;
0219           break;
0220        /* reload mode selected */
0221        case '3' :
0222           dmac0.mode = TRANSFER_MODE_RELOAD;
0223           break;
0224        /* multi-dimensional mode selected */
0225        case '4' :
0226           dmac0.mode = TRANSFER_MODE_MULTI;
0227           break;
0228        /* previous menu selected */
0229        case 'r' :
0230           return 1;
```

```
0231        /* selecting an invalid value */
0232        default :
0233           printf("Invalid value. Please selected 1 to 4 or r.\n\r");
0234           break;
0235     }
0236
0237     /* in the case of inputting the value from '1' to '3' */
0238     if (KeyBuff >= '1' && KeyBuff <= '3') {
0239        /* DMAC0 transfer size to be selected */
0240        ret = dmac0_select_size();
0241        if (ret == 0)
0242           return 0;
0243     }
0244
0245     /* in the case of inputting the value of '4' */
0246     if (KeyBuff == '4') {
0247        /* DMAC0 multi-dimensional transfer mode to be selected */
0248        ret = dmac0_select_multi_mode();
0249        if (ret == 0)
0250           return 0;
0251     }
0252  }
0253 }
0254
0255 /*""FUNC COMMENT""*****************************************************
0256 * ID                :
0257 * Outline           : DMAC0 multi-dimensional transfer mode select
0258 * Declaration       : int dmac0_select_multi_mode( void )
0259 * Description       : DMAC0 multi-dimensional transfer mode select
0260 * Argument          : none
0261 * Return Value      : 0:transfer end,1:canceled menu
0262 * Calling Functions :
0263 *""FUNC COMMENT END""*************************************************/
0264 int dmac0_select_multi_mode( void )
0265 {
0266  int ret;
0267  char KeyBuff;
0268
0269  while (1){
0270     /* showing DMAC0 multi-dimensional mode selection screen */
0271     printf("[DMAC0-ch%d-%s-%s]\n\r",
0272        dmac0.ch,dmac0_dir_str[dmac0.dir -1],dmac0_mode[dmac0.mode -1]);
0273     printf("  - Multi-dimensional mode select\n\r");
0274     printf("  1. Multi-dimensional transfer\n\r");
0275     printf("  2. Stride transfer\n\r");
0276     printf("  3. Scatter transfer\n\r");
0277     printf("  4. Gather transfer\n\r");
0278     printf("  r. Return to previous menu\n\r");
0279     printf("  Select No:");
0280     /* clear key buffer */
0281     KeyBuff = 0;
0282
0283     /* waiting for input from SCIF */
0284     while( scif_recive_data_byte( &KeyBuff ) != 0)
0285     ;
0286
0287     printf("\n\r");
0288
```

```
0289     /* judgment of input characters */
0290     switch (KeyBuff) {
0291        /* multi-dimensional transfer selected */
0292        case '1' :
0293           dmac0.multi_mode = TRANSFER_MULTI_MULTI;
0294           break;
0295        /* stride transfer selected */
0296        case '2' :
0297           dmac0.multi_mode = TRANSFER_MULTI_STRIDE;
0298           break;
0299        /* scatter transfer selected */
0300        case '3' :
0301           dmac0.multi_mode = TRANSFER_MULTI_SCATTER;
0302           break;
0303        /* gather transfer selected */
0304        case '4' :
0305           dmac0.multi_mode = TRANSFER_MULTI_GATHER;
0306           break;
0307        /* previous menu selected */
0308        case 'r' :
0309           return 1;
0310        /* selecting an invalid value */
0311        default :
0312           printf("Invalid value. Please selected 1 to 4 or r.\n\r");
0313           break;
0314     }
0315
0316     /* in the case of inputting the value from '1' to '4' */
0317     if (KeyBuff >= '1' && KeyBuff <= '4') {
0318        /* DMAC0 transfer size to be selected */
0319        ret = dmac0_select_size();
0320        if (ret == 0)
0321           return 0;
0322     }
0323  }
0324 }
0325
0326 /*""FUNC COMMENT""*****************************************************
0327 * ID                :
0328 * Outline           : DMAC0 transfer size select
0329 * Declaration       : int dmac0_select_size( void )
0330 * Description       : DMAC0 transfer size select
0331 * Argument          : none
0332 * Return Value      : 0:transfer end,1:canceled menu
0333 * Calling Functions :
0334 *""FUNC COMMENT END""*************************************************/
0335 int dmac0_select_size( void )
0336 {
0337  int ret;
0338  char KeyBuff;
0339
0340  while (1){
0341     /* showing DMAC0 transfer size selection screen */
0342     if (dmac0.mode != TRANSFER_MODE_MULTI)
0343        printf("[DMAC0-ch%d-%s-%s]\n\r",
0344           dmac0.ch,dmac0_dir_str[dmac0.dir -1],dmac0_mode[dmac0.mode -1]);
0345     else
0346        printf("[DMAC0-ch%d-%s-%s-%s]\n\r",
```

```
0347              dmac0.ch,dmac0_dir_str[dmac0.dir -1],
0348              dmac0_mode[dmac0.mode -1],dmac0_multi_mode[dmac0.multi_mode -1]);
0349
0350     printf("  - Transfer data size select\n\r");
0351     printf("  1. Byte\n\r");
0352     printf("  2. Words\n\r");
0353     printf("  3. Long Words\n\r");
0354     printf("  4. 16bytes\n\r");
0355     printf("  5. 32bytes\n\r");
0356     printf("  r. Return to previous menu\n\n\r");
0357     printf("  Select No:");
0358     /* clear key buffer */
0359     KeyBuff = 0;
0360
0361     /* waiting for input from SCIF */
0362     while( scif_recive_data_byte( &KeyBuff ) != 0)
0363     ;
0364
0365     printf("\n\r");
0366
0367     /* judgment of input characters */
0368     switch (KeyBuff) {
0369        /* transfer size is byte selected */
0370        case '1' :
0371           dmac0.size = TRANSFER_SIZE_BYTE;
0372           break;
0373        /* transfer size is word selected */
0374        case '2' :
0375           dmac0.size = TRANSFER_SIZE_WORD;
0376           break;
0377        /* transfer size is long word selected */
0378        case '3' :
0379           dmac0.size = TRANSFER_SIZE_LONG_WORD;
0380           break;
0381        /* transfer size is 16bytes selected */
0382        case '4' :
0383           dmac0.size = TRANSFER_SIZE_16BYTES;
0384           break;
0385        /* transfer size is 32bytes selected */
0386        case '5' :
0387           dmac0.size = TRANSFER_SIZE_32BYTES;
0388           break;
0389        /* previous menu selected */
0390        case 'r' :
0391           return 1;
0392        /* selecting an invalid value */
0393        default :
0394           printf("Invalid value. Please selected 1 to 5 or r.\n\r");
0395           break;
0396     }
0397
0398     /* in the case of inputting the value from '1' to '5' */
0399     if (KeyBuff >= '1' && KeyBuff <= '5') {
0400        /* DMAC0 cycle steal mode to be selected */
0401        ret = dmac0_select_cycle();
0402        if (ret == 0)
0403           return 0;
0404     }
```

```
0405  }
0406 }
0407
0408 /*""FUNC COMMENT""************************************************
0409 * ID                :
0410 * Outline           : DMAC0 cycle steal mode select
0411 * Declaration       : int dmac0_select_cycle( void )
0412 * Description       : DMAC0 cycle steal mode select
0413 * Argument          : none
0414 * Return Value      : 0:transfer end,1:canceled menu
0415 * Calling Functions :
0416 *""FUNC COMMENT END""*************************************************/
0417 int dmac0_select_cycle( void )
0418 {
0419  int ret;
0420  char KeyBuff;
0421
0422  while (1){
0423     /* showing DMAC0 cycle steal mode selection screen */
0424     if (dmac0.mode != TRANSFER_SIZE_32BYTES)
0425        printf("[DMAC0-ch%d-%s-%s-%dbyte(s)]\n\r",
0426              dmac0.ch, dmac0_dir_str[dmac0.dir -1], dmac0_mode[dmac0.mode -1],
0427              dmac0.size);
0428     else
0429        printf("[DMAC0-ch%d-%s-%s-%s-%dbyte(s)]\n\r",
0430              dmac0.ch,dmac0_dir_str[dmac0.dir -1],
0431              dmac0_mode[dmac0.mode -1],dmac0_multi_mode[dmac0.multi_mode -1],
0432              dmac0.size);
0433
0434     printf("  - Cycle steal mode select\n\r");
0435     printf("  1. Normal\n\r");
0436     printf("  2. Intermittent mode 16\n\r");
0437     printf("  3. Intermittent mode 64\n\r");
0438     printf("  r. Return to previous menu\n\r");
0439     printf("  Select No:");
0440     /* clear key buffer */
0441     KeyBuff = 0;
0442
0443     /* waiting for input from SCIF */
0444     while( scif_recive_data_byte( &KeyBuff ) != 0)
0445     ;
0446     printf("\n\r");
0447
0448     /* judgment of input characters */
0449     switch (KeyBuff) {
0450        /* normal mode selected */
0451        case '1' :
0452           dmac0.cycle = CYCLE_STEALMODE_NORMAL;
0453           break;
0454        /* intermittent mode 16 selected */
0455        case '2' :
0456           dmac0.cycle = CYCLE_STEALMODE_16;
0457           break;
0458        /* intermittent mode 64 selected */
0459        case '3' :
0460           dmac0.cycle = CYCLE_STEALMODE_64;
0461           break;
0462        /* previous menu selected */
```

```
0463        case 'r' :
0464           return 1;
0465        /* selecting an invalid value */
0466        default :
0467           printf("Invalid value. Please selected 1 to 3 or r.\n\r");
0468           break;
0469     }
0470
0471     /* in the case of inputting the value from '1' to '3' */
0472     if (KeyBuff >= '1' && KeyBuff <= '3') {
0473        /* DMAC0 cache control to be selected */
0474        ret = dmac0_select_cache();
0475        if (ret == 0)
0476           return 0;
0477     }
0478  }
0479
0480 }
0481
0482 /*""FUNC COMMENT""*****************************************************
0483 * ID                :
0484 * Outline           : DMAC0 cache control select
0485 * Declaration       : int dmac0_select_cache( void )
0486 * Description       : DMAC0 cache control select
0487 * Argument          : none
0488 * Return Value      : 0:transfer end,1:canceled menu
0489 * Calling Functions :
0490 *""FUNC COMMENT END""*************************************************/
0491 int dmac0_select_cache( void )
0492 {
0493  int ret;
0494  char KeyBuff;
0495
0496  while (1){
0497     /* showing DMAC0 cache control selection screen */
0498     if (dmac0.mode != TRANSFER_SIZE_32BYTES)
0499        printf("[DMAC0-ch%d-%s-%s-%dbyte(s)-%s]\n\r",
0500              dmac0.ch, dmac0_dir_str[dmac0.dir -1], dmac0_mode[dmac0.mode -1],
0501              dmac0.size,cycle_str[dmac0.cycle -1]);
0502     else
0503        printf("[DMAC0-ch%d-%s-%s-%s-%dbyte(s)-%s]\n\r",
0504              dmac0.ch,dmac0_dir_str[dmac0.dir -1],
0505              dmac0_mode[dmac0.mode -1],dmac0_multi_mode[dmac0.multi_mode -1],
0506              dmac0.size,cycle_str[dmac0.cycle -1]);
0507
0508     printf("  - Cache Select\n\r");
0509     printf("  1. Flush/Purge\n\r");
0510     printf("  2. No Flush/Purge\n\r");
0511     printf("  r. Return to previous menu\n\r");
0512     printf("  Select No:");
0513
0514     /* clear key buffer */
0515     KeyBuff = 0;
0516
0517     /* waiting for input from SCIF */
0518     while( scif_recive_data_byte( &KeyBuff ) != 0)
0519     ;
0520
```

```
0521     printf("\n\r");
0522
0523     /* judgment of input characters */
0524     switch (KeyBuff) {
0525        /* cache control ON selected */
0526        case '1' :
0527           dmac0.cache = SELECT_CACHE_ON;
0528           break;
0529        /* cache control OFF selected */
0530        case '2' :
0531           dmac0.cache = SELECT_CACHE_OFF;
0532           break;
0533        /* previous menu selected */
0534        case 'r' :
0535           return 1;
0536        /* selecting an invalid value */
0537        default :
0538           printf("Invalid value. Please selected 1 to 2 or r.\n\r");
0539           break;
0540     }
0541
0542     /* in the case of inputting the value from '1' to '2' */
0543     if (KeyBuff >= '1' && KeyBuff <= '2') {
0544        /* DMAC0 transfer to be selected */
0545        ret = dmac0_transfer();
0546        if (ret == 0)
0547           return 0;
0548     }
0549  }
0550 }
0551
0552
0553 /*""FUNC COMMENT""*****************************************************
0554 * ID                :
0555 * Outline           : DMAC0 transfer process
0556 * Declaration       : int dmac0_transfer( void )
0557 * Description       : DMAC0 transfer process
0558 * Argument          : none
0559 * Return Value      : 0:transfer end,1:canceled menu
0560 * Calling Functions :
0561 *""FUNC COMMENT END""*************************************************/
0562 int dmac0_transfer( void )
0563 {
0564  int ret;
0565  char KeyBuff;
0566  char *ol_address,sdram_address;
0567
0568  while (1){
0569     /* showing DMAC0 transfer process selection screen */
0570     if (dmac0.mode != TRANSFER_SIZE_32BYTES)
0571        printf("[DMAC0-ch%d-%s-%s-%dbyte(s)-%s-%s]\n\r",
0572              dmac0.ch, dmac0_dir_str[dmac0.dir -1], dmac0_mode[dmac0.mode -1],
0573              dmac0.size,cycle_str[dmac0.cycle -1],cache_str[dmac0.cache -1]);
0574     else
0575        printf
0576        ("[DMAC0-ch%d-%s-%s-%s-%dbyte(s)-%s-%s]\n\r",
0577              dmac0.ch,dmac0_dir_str[dmac0.dir -1],
0578              dmac0_mode[dmac0.mode -1],dmac0_multi_mode[dmac0.multi_mode -1],
```

```
0579              dmac0.size,cycle_str[dmac0.cycle -1],cache_str[dmac0.cache -1]);
0580
0581     printf("  - Do you start DMA transfer?\n\r");
0582     printf("  1. Yes\n\r");
0583     printf("  r. Return to previous menu\n\r");
0584     printf("  Select No:");
0585     /* clear key buffer */
0586     KeyBuff = 0;
0587
0588     /* waiting for input from SCIF */
0589     while( scif_recive_data_byte( &KeyBuff ) != 0)
0590     ;
0591
0592     printf("\n\r");
0593
0594     /* judgment of input characters */
0595     switch (KeyBuff) {
0596        /* transfer start selected */
0597        case '1' :
0598           memory_init(dmac0.dir);          /* memory initialization */
0599           dmac0_init();                    /* DMAC0 initialization */
0600           dmac0_start();                /* DMAC0 transfer start */
0601           dmac0_result();               /* showing DMAC0 transfer result */
0602
0603           printf("DMA transfer compleate!!\n\r");
0604           while (1) {
0605              printf("Please hit any key.\n\r");
0606              /* clear key buffer */
0607              KeyBuff = 0;
0608
0609              /* wait for one character */
0610              while( scif_recive_data_byte( &KeyBuff ) != 0)
0611              ;
0612
0613              printf("\n\r");
0614              return 0;
0615           }
0616           break;
0617        /* previous menu selected */
0618        case 'r' :
0619           return 1;
0620        /* selecting an invalid value */
0621        default :
0622           printf("Invalid value. Please selected 1 or r.\n\r");
0623           break;
0624     }
0625  }
0626 }
0627
0628 /*""FUNC COMMENT""*****************************************************
0629 * ID                :
0630 * Outline           : clear of data storage structure
0631 * Declaration       : void dmac0_data_clear( void )
0632 * Description       : clear of data storage structure
0633 * Argument          : none
0634 * Return Value      : none
0635 * Calling Functions :
0636 *""FUNC COMMENT END""*************************************************/
```

```
0637 void dmac0_data_clear( void )
0638 {
0639  dmac0.ch = 0xFF;
0640  dmac0.dir = 0;
0641  dmac0.mode = 0;
0642  dmac0.multi_mode = 0;
0643  dmac0.size = 0;
0644  dmac0.cycle = 0;
0645  dmac0.cache = 0;
0646 }
0647
0648 /*""FUNC COMMENT""*****************************************************
0649 * ID                :
0650 * Outline           : Initialization of DMAC0
0651 * Declaration       : void dmac0_init( void )
0652 * Description       : Initialization of DMAC0
0653 * Argument          : none
0654 * Return Value      : none
0655 * Calling Functions :
0656 *""FUNC COMMENT END""*************************************************/
0657 void dmac0_init( void )
0658 {
0659  volatile int i;
0660
0661  /* Stop the clock supply to DAMC */
0662  CPG.MSTPCR1.BIT.MSTP104 = 1;
0663  CPG.MSTPCR1.BIT.MSTP105 = 1;
0664
0665  /* wait for DMAC stop */
0666  for (i = 0; i < 10000; i++)
0667  ;
0668
0669  /* Start the clock supply to DAMC */
0670  CPG.MSTPCR1.BIT.MSTP104 = 0;
0671  CPG.MSTPCR1.BIT.MSTP105 = 0;
0672
0673  /* wait for DMAC start */
0674  for (i = 0; i < 10000; i++)
0675  ;
0676
0677  /* interrupt flag clear */
0678  int_flg = 0;
0679
0680  /* transfer is channel 0 or channel 4 ? */
0681  if (dmac0.ch ==0) {
0682     dmac0_ch0_init();                    /* DMAC0 channel 0 init */
0683  } else {
0684     dmac0_ch4_init();                    /* DMAC0 channel 4 init */
0685  }
0686
0687  /* Cycle steal mode setteings */
0688  switch (dmac0.cycle) {
0689     case CYCLE_STEALMODE_NORMAL:
0690        DMAC0.DMA0OR.BIT.CMS = 0x00;      /* set the normal mode */
0691        break;
0692     case CYCLE_STEALMODE_16:
0693        DMAC0.DMA0OR.BIT.CMS = 0x02;      /* set the intermittent mode 16 */
0694        break;
```

```
0695     case CYCLE_STEALMODE_64:
0696        DMAC0.DMA0OR.BIT.CMS = 0x03;         /* set the intermittent mode 64 */
0697        break;
0698  }
0699 }
0700
0701 /*""FUNC COMMENT""*****************************************************
0702 * ID                :
0703 * Outline           : Initialization of DMAC0
0704 * Declaration       : void dmac0_ch0_init( void )
0705 * Description       : Initialization of DMAC0 channel0
0706 * Argument          : none
0707 * Return Value      : none
0708 * Calling Functions :
0709 *""FUNC COMMENT END""*************************************************/
0710 void dmac0_ch0_init( void )
0711 {
0712  INTC.INT2PRI3.BIT.DMAC00 = 3;              /* Set interrupt priority DMAC0 */
0713  INTC.C0INT2MSKCLR1.BIT._DMAC00 = 1;        /* DMAC0 interrupt mask clear */
0714
0715  /* cache control */
0716  if (dmac0.cache == SELECT_CACHE_ON && dmac0.dir == OL_TO_DDR)
0717     /* Cache Invalidation */
0718     cache_invalidate((void *)D_DMAC_SDRAM_VLADR, TRANSFER_SIZE_MULTI_MULTI_32BYTES);
0719  else if (dmac0.cache == SELECT_CACHE_ON && dmac0.dir == DDR_TO_OL)
0720     /* Writeback cache data to DDR3SDRAM */
0721     cache_writeback((void *)D_DMAC_SDRAM_VLADR, TRANSFER_SIZE_MULTI_MULTI_32BYTES);
0722
0723
0724  DMAC0.CHCR0.LONG = 0x40000000;                    /* CHCR0 clear */
0725
0726  /* Settings DMAC0 SAR0 and DAR0  */
0727  if (dmac0.dir == OL_TO_DDR) {
0728     DMAC0.DAR0 = (unsigned long)D_DMAC_SDRAM_ADR;   /* DDR3SDRAM to set a transfer destination address */
0729     DMAC0.SAR0 = (unsigned long)D_DMAC_OL_ADR;      /* OL memory to set a transfer soruce address */
0730  } else {
0731     DMAC0.DAR0 = (unsigned long)D_DMAC_OL_ADR;      /* OL memory to set a transfer destination address */
0732     DMAC0.SAR0 = (unsigned long)D_DMAC_SDRAM_ADR;   /* DDR3SDRAM to set a transfer soruce address */
0733  }
0734
0735  /* Setting the amount of data transferred */
0736  if (dmac0.size > TRANSFER_SIZE_WORD)
0737     DMAC0.TCR0 = D_DMAC_TRANS_SIZE / dmac0.size;
0738  else
0739     DMAC0.TCR0 = D_DMAC_TRANS_SIZE_LITTLE / dmac0.size;
0740
0741  DMAC0.CHCR0.BIT.DE = 0x00;           /* DMA transfer disabled */
0742  DMAC0.CHCR0.BIT.RS = 0x04;           /* transfer request source Auto-request */
0743
0744  /* Transfer mode */
0745  switch (dmac0.mode) {
0746     case TRANSFER_MODE_NORMAL:
0747        DMAC0.CHCR0.BIT.RPT = 0x00;    /* normal mode */
0748        DMAC0.CHCR0.BIT.DM = 0x01;        /* Destination address is incremented */
0749        DMAC0.CHCR0.BIT.SM = 0x01;        /* Source address is incremented */
0750        break;
0751     case TRANSFER_MODE_REPEAT:
0752        DMAC0.TCR0 = DMAC0.TCR0 / 2;
```

```
0753        DMAC0.DARB0 = DMAC0.DAR0 + DMAC0.TCR0 * dmac0.size; /* set a repeat address */
0754        DMAC0.CHCR0.BIT.RPT = 0x03;    /* repeat mode */
0755        DMAC0.CHCR0.BIT.SM = 0x01;        /* Destination address is incremented */
0756        DMAC0.CHCR0.BIT.DM = 0x01;        /* Source address is incremented */
0757        break;
0758     case TRANSFER_MODE_RELOAD:
0759        DMAC0.CHCR0.BIT.RPT = 0x07;    /* reload mode */
0760        DMAC0.TCRB0 = (1 << 16) | 1;      /* set the reload counter */
0761        DMAC0.CHCR0.BIT.DM = 0x01;        /* Destination address is incremented */
0762        DMAC0.CHCR0.BIT.SM = 0x01;        /* Source address is incremented */
0763        break;
0764     case TRANSFER_MODE_MULTI:
0765        switch (dmac0.multi_mode) {
0766           case TRANSFER_MULTI_MULTI:
0767              /* multi-dimensional transfer settings */
0768              set_multi_dimensional_ch0();
0769              break;
0770           case TRANSFER_MULTI_STRIDE:
0771              DMAC0.CHCR0.BIT.RPT = 0x0D;       /* multi-dimensional mode */
0772              DMAC0.TCRB0 = 0x00020002;            /* set the reload counter */
0773              DMAC0.SAOFR0 = (dmac0.size * 2  << 16) | dmac0.size * 2; /* setting SAOFR */
0774              DMAC0.DAOFR0 = (dmac0.size * 2  << 16) | dmac0.size * 2; /* setting DAOFR */
0775              break;
0776           case TRANSFER_MULTI_SCATTER:
0777              DMAC0.TCR0 /= 2;
0778              DMAC0.CHCR0.BIT.RPT = 0x0E;        /* multi-dimensional mode */
0779              DMAC0.TCRB0 = 0x00010001;             /* set the reload counter */
0780              DMAC0.CHCR0.BIT.SM = 0x01;      /* Source address is incremented */
0781              DMAC0.DAOFR0 = (dmac0.size * 2  << 16) | dmac0.size * 2; /* setting DAOFR */
0782              break;
0783           case TRANSFER_MULTI_GATHER:
0784              DMAC0.TCR0 /= 2;
0785              DMAC0.CHCR0.BIT.RPT = 0x0F;          /* multi-dimensional mode */
0786              DMAC0.TCRB0 = 0x00010001;            /* set the reload counter */
0787              DMAC0.CHCR0.BIT.DM = 0x01; /* Destination address is incremented */
0788              DMAC0.SAOFR0 = (dmac0.size * 2  << 16) | dmac0.size * 2; /* setting SAOFR */
0789              break;
0790        }
0791  }
0792
0793  /* setting transfer Size */
0794  switch (dmac0.size) {
0795     case TRANSFER_SIZE_BYTE:
0796        DMAC0.CHCR0.BIT.TS2 = 0x00;            /* Byte units */
0797        DMAC0.CHCR0.BIT.TS = 0x00;
0798        break;
0799     case TRANSFER_SIZE_WORD:
0800        DMAC0.CHCR0.BIT.TS2 = 0x00;            /* Word units */
0801        DMAC0.CHCR0.BIT.TS = 0x01;
0802        break;
0803     case TRANSFER_SIZE_LONG_WORD:
0804        DMAC0.CHCR0.BIT.TS2 = 0x00;            /* Long Word units */
0805        DMAC0.CHCR0.BIT.TS = 0x2;
0806        break;
0807     case TRANSFER_SIZE_16BYTES:
0808        DMAC0.CHCR0.BIT.TS2 = 0x00;            /* 16bytes units */
0809        DMAC0.CHCR0.BIT.TS = 0x3;
0810        break;
```

```
0811     case TRANSFER_SIZE_32BYTES:
0812        DMAC0.CHCR0.BIT.TS2 = 0x01;             /* 32bytes units */
0813        DMAC0.CHCR0.BIT.TS = 0x00;
0814        break;
0815  }
0816
0817  DMAC0.CHCR0.BIT.IE = 1;                       /* set a interrupt enable */
0818 }
0819
0820 /*""FUNC COMMENT""*****************************************************
0821 * ID                :
0822 * Outline           : Initialization of DMAC0
0823 * Declaration       : void dmac0_ch4_init( void )
0824 * Description       : Initialization of DMAC0 channel4
0825 * Argument          : none
0826 * Return Value      : none
0827 * Calling Functions :
0828 *""FUNC COMMENT END""*************************************************/
0829 void dmac0_ch4_init( void )
0830 {
0831  INTC.INT2PRI4.BIT.DMAC04 = 3;                 /* Set interrpt priority DMAC0 */
0832  INTC.C0INT2MSKCLR1.BIT._DMAC04 = 1;           /* DMAC0 interrupt mask clear */
0833
0834  /* cache control */
0835  if (dmac0.cache == SELECT_CACHE_ON && dmac0.dir == OL_TO_DDR) {
0836     /* Cache Invalidation */
0837     cache_invalidate((void *)D_DMAC_SDRAM_VLADR, TRANSFER_SIZE_MULTI_MULTI_32BYTES);
0838  } else if (dmac0.cache == SELECT_CACHE_ON && dmac0.dir == DDR_TO_OL)
0839     /* Writeback cache data to DDR3SDRAM */
0840     cache_writeback((void *)D_DMAC_SDRAM_VLADR, TRANSFER_SIZE_MULTI_MULTI_32BYTES);
0841
0842  DMAC0.CHCR0.LONG = 0x40000000;                    /* CHCR0 clear */
0843
0844  /* Settings DMAC0 SAR0 and DAR0  */
0845  if (dmac0.dir == OL_TO_DDR) {
0846     DMAC0.DAR4 = (unsigned long)D_DMAC_SDRAM_ADR;      /* DDR3SDRAM to set a transfer destination address */
0847     DMAC0.SAR4 = (unsigned long)D_DMAC_OL_ADR;         /* OL memory to set a transfer soruce address */
0848  } else {
0849     DMAC0.DAR4 = (unsigned long)D_DMAC_OL_ADR;         /* OL memory to set a transfer destination address */
0850     DMAC0.SAR4 = (unsigned long)D_DMAC_SDRAM_ADR;      /* DDR3SDRAM to set a transfer soruce address */
0851  }
0852
0853  /* Setting the amount of data transferred */
0854  if (dmac0.size > TRANSFER_SIZE_WORD)
0855     DMAC0.TCR4 = D_DMAC_TRANS_SIZE / dmac0.size;
0856  else
0857     DMAC0.TCR4 = D_DMAC_TRANS_SIZE_LITTLE / dmac0.size;
0858
0859  DMAC0.CHCR4.BIT.DE = 0x00;                        /* DMA transfer disabled */
0860  DMAC0.CHCR4.BIT.RS = 0x04;            /* transfer request source Auto-request */
0861
0862  /* Transfer mode */
0863  switch (dmac0.mode) {
0864     case TRANSFER_MODE_NORMAL:
0865        DMAC0.CHCR4.BIT.RPT = 0x00;             /* normal mode */
0866        DMAC0.CHCR4.BIT.DM = 0x01;      /* Destination address is incremented */
0867        DMAC0.CHCR4.BIT.SM = 0x01;      /* Source address is incremented */
0868        break;
```

```
0869     case TRANSFER_MODE_REPEAT:
0870        DMAC0.TCR4 = DMAC0.TCR0 / 2;
0871        DMAC0.DARB4 = DMAC0.DAR0 + DMAC0.TCR0 * dmac0.size; /* set a repeat address */
0872        DMAC0.CHCR4.BIT.RPT = 0x03;       /* repeat mode */
0873        DMAC0.CHCR4.BIT.SM = 0x01;        /* Destination address is incremented */
0874        DMAC0.CHCR4.BIT.DM = 0x01;        /* Source address is incremented */
0875        break;
0876     case TRANSFER_MODE_RELOAD:
0877        DMAC0.CHCR4.BIT.RPT = 0x07;       /* reload mode */
0878        DMAC0.TCRB4 = (1 << 16) | 1;      /* set the reload counter */
0879        DMAC0.CHCR4.BIT.DM = 0x01;        /* Destination address is incremented */
0880        DMAC0.CHCR4.BIT.SM = 0x01;        /* Source address is incremented */
0881        break;
0882     case TRANSFER_MODE_MULTI:
0883        switch (dmac0.multi_mode) {
0884           case TRANSFER_MULTI_MULTI:
0885              /* multi-dimensional transfer settings */
0886              set_multi_dimensional_ch4();
0887              break;
0888           case TRANSFER_MULTI_STRIDE:
0889              DMAC0.CHCR4.BIT.RPT = 0x0D;    /* multi-dimensional mode */
0890              DMAC0.TCRB4 = 0x00020002;      /* set the reload counter */
0891              DMAC0.SAOFR4 = (dmac0.size * 2  << 16) | dmac0.size * 2; /* setting SAOFR */
0892              DMAC0.DAOFR4 = (dmac0.size * 2  << 16) | dmac0.size * 2; /* setting DAOFR */
0893              break;
0894           case TRANSFER_MULTI_SCATTER:
0895              DMAC0.TCR4 /= 2;
0896              DMAC0.CHCR4.BIT.RPT = 0x0E;          /* multi-dimensional mode */
0897              DMAC0.TCRB4 = 0x00010001;            /* set the reload counter */
0898              DMAC0.CHCR4.BIT.SM = 0x01;                    /* Source address is incremented */
0899              DMAC0.DAOFR4 = (dmac0.size * 2  << 16) | dmac0.size * 2; /* setting DAOFR */
0900              break;
0901           case TRANSFER_MULTI_GATHER:
0902              DMAC0.TCR4 /= 2;
0903              DMAC0.CHCR4.BIT.RPT = 0x0F;          /* multi-dimensional mode */
0904              DMAC0.TCRB4 = 0x00010001;            /* set the reload counter */
0905              DMAC0.CHCR4.BIT.DM = 0x01;           /* Destination address is incremented */
0906              DMAC0.SAOFR4 = (dmac0.size * 2  << 16) | dmac0.size * 2; /* setting SAOFR */
0907              break;
0908        }
0909  }
0910
0911  /* setting transfer Size */
0912  switch (dmac0.size) {
0913     case TRANSFER_SIZE_BYTE:
0914        DMAC0.CHCR4.BIT.TS2 = 0x00;                  /* Byte units */
0915        DMAC0.CHCR4.BIT.TS = 0x00;
0916        break;
0917     case TRANSFER_SIZE_WORD:
0918        DMAC0.CHCR4.BIT.TS2 = 0x00;                  /* Word units */
0919        DMAC0.CHCR4.BIT.TS = 0x01;
0920        break;
0921     case TRANSFER_SIZE_LONG_WORD:
0922        DMAC0.CHCR4.BIT.TS2 = 0x00;                  /* Long Word units */
0923        DMAC0.CHCR4.BIT.TS = 0x2;
0924        break;
0925     case TRANSFER_SIZE_16BYTES:
0926        DMAC0.CHCR4.BIT.TS2 = 0x00;                  /* 16bytes units */
```

```
0927        DMAC0.CHCR4.BIT.TS = 0x3;
0928        break;
0929     case TRANSFER_SIZE_32BYTES:
0930        DMAC0.CHCR4.BIT.TS2 = 0x01;                  /* 32bytes units */
0931        DMAC0.CHCR4.BIT.TS = 0x00;
0932        break;
0933  }
0934  DMAC0.CHCR4.BIT.IE = 1;                          /* set a interrupt enable */
0935 }
0936
0937 /*""FUNC COMMENT""*****************************************************
0938 * ID                :
0939 * Outline           : Initialization of DMAC0
0940 * Declaration       : void set_multi_dimensional_ch0( void )
0941 * Description       : Initialization of multi-dimensional mode for DMAC0 channel0.
0942 * Argument          : none
0943 * Return Value      : none
0944 * Calling Functions :
0945 *""FUNC COMMENT END""*************************************************/
0946 void set_multi_dimensional_ch0( void )
0947 {
0948  DMAC0.TCR0 = 0x0C;                      /* set the transfer count 12 */
0949  DMAC0.CHCR0.BIT.RPT = 0x0E;                /* multi-dimensional mode */
0950  DMAC0.CHCR0.BIT.SM = 0x01;                 /* Source address is incremented */
0951  DMAC0.TCRB0 =  0x00040004;                 /* setting reload counter */
0952  switch (dmac0.size) {
0953     case TRANSFER_SIZE_BYTE:
0954        DMAC0.DAOFR0 = 0x00010003;           /* setting DAOFR */
0955        break;
0956     case TRANSFER_SIZE_WORD:
0957        DMAC0.DAOFR0 = 0x00020006;           /* setting DAOFR */
0958        break;
0959     case TRANSFER_SIZE_LONG_WORD:
0960        DMAC0.DAOFR0 = 0x00040010;           /* setting DAOFR */
0961        break;
0962     case TRANSFER_SIZE_16BYTES:
0963        DMAC0.DAOFR0 = 0x00100030;           /* setting DAOFR */
0964        break;
0965     case TRANSFER_SIZE_32BYTES:
0966        DMAC0.DAOFR0 = 0x00200060;           /* setting DAOFR */
0967        break;
0968  }
0969 }
0970
0971 /*""FUNC COMMENT""*****************************************************
0972 * ID                :
0973 * Outline           : Initialization of DMAC0
0974 * Declaration       : void set_multi_dimensional_ch4( void )
0975 * Description       : Initialization of multi-dimensional mode for DMAC0 channel4
0976 * Argument          : none
0977 * Return Value      : none
0978 * Calling Functions :
0979 *""FUNC COMMENT END""*************************************************/
0980 void set_multi_dimensional_ch4( void )
0981 {
0982  DMAC0.TCR4 = 0x0C;                      /* set the transfer count 12 */
0983  DMAC0.CHCR4.BIT.RPT = 0x0E;             /* multi-dimensional mode */
0984  DMAC0.CHCR4.BIT.SM = 0x01;                 /* Source address is incremented */
```

```
0985  DMAC0.TCRB4 =  0x00040004;                /* setting reload counter */
0986  switch (dmac0.size) {
0987     case TRANSFER_SIZE_BYTE:
0988        DMAC0.DAOFR4 = 0x00010003;          /* setting DAOFR */
0989        break;
0990     case TRANSFER_SIZE_WORD:
0991        DMAC0.DAOFR4 = 0x00020006;          /* setting DAOFR */
0992        break;
0993     case TRANSFER_SIZE_LONG_WORD:
0994        DMAC0.DAOFR4 = 0x00040010;          /* setting DAOFR */
0995        break;
0996     case TRANSFER_SIZE_16BYTES:
0997        DMAC0.DAOFR4 = 0x00100030;          /* setting DAOFR */
0998        break;
0999     case TRANSFER_SIZE_32BYTES:
1000        DMAC0.DAOFR4 = 0x00200060;          /* setting DAOFR */
1001        break;
1002  }
1003 }
1004
1005 /*""FUNC COMMENT""************************************************
1006 * ID                :
1007 * Outline           : DMAC0 starting transfer
1008 * Declaration       : void dmac0_start( void )
1009 * Description       : DMAC0 starting transfer
1010 * Argument          : none
1011 * Return Value      : none
1012 * Calling Functions :
1013 *""FUNC COMMENT END""*********************************************/
1014 void dmac0_start( void )
1015 {
1016  unsigned long dumyy;
1017
1018  DMAC0.DMA0OR.BIT.DME = 0x01;      /* DMA transfers on all channels are enabled */
1019  if (dmac0.ch == 0) {
1020     DMAC0.CHCR0.BIT.DE = 0x01;     /* DMA channel 0 transfer enabled */
1021  } else {
1022     DMAC0.CHCR4.BIT.DE = 0x01;     /* DMA channel 4 transfer enabled */
1023  }
1024
1025  /* Waiting for the transfer end */
1026  while ( int_flg != 1)
1027  ;
1028
1029  /* process for after the transfer */
1030  if (dmac0.ch == 0) {
1031     DMAC0.CHCR0.BIT.DE = 0;              /* DMA channel 0 transfer disabled */
1032     dummy = DMAC0.CHCR0.BIT.TE;       /* dummy lead for TE clear */
1033     DMAC0.CHCR0.BIT.TE = 0;              /* transfer end flag clear */
1034  } else {
1035     DMAC0.CHCR4.BIT.DE = 0;              /* DMA channel 4 transfer disabled */
1036     dummy = DMAC0.CHCR4.BIT.TE;       /* dummy lead for TE clear */
1037     DMAC0.CHCR4.BIT.TE = 0;              /* transfer end flag clear */
1038  }
1039
1040  DMAC0.DMA0OR.BIT.DME = 0x00;   /* DMA transfers on all channels are disabled */
1041 }
1042
```

```
1043 /*""FUNC COMMENT""*****************************************************
1044 * ID                :
1045 * Outline           : the transfer results show DMAC0
1046 * Declaration       : void dmac0_result( void )
1047 * Description       : the transfer results show DMAC0
1048 * Argument          : none
1049 * Return Value      : none
1050 * Calling Functions :
1051 *""FUNC COMMENT END""*************************************************/
1052 void dmac0_result( void )
1053 {
1054  struct result result_data,tmp;
1055
1056  /* Displaying the transfer settings */
1057  if (dmac0.mode != TRANSFER_MODE_MULTI)
1058     printf("[DMAC0-ch%d-%s-%s-%dbyte(s)-%s-%s]\n\r",
1059           dmac0.ch, dmac0_dir_str[dmac0.dir -1], dmac0_mode[dmac0.mode -1],
1060           dmac0.size,cycle_str[dmac0.cycle -1],cache_str[dmac0.cache -1]);
1061  else
1062     printf("[DMAC0-ch%d-%s-%s-%s-%dbyte(s)-%s-%s]\n\r",
1063           dmac0.ch,dmac0_dir_str[dmac0.dir -1],
1064           dmac0_mode[dmac0.mode -1],dmac0_multi_mode[dmac0.multi_mode -1],
1065           dmac0.size,cycle_str[dmac0.cycle -1],cache_str[dmac0.cache -1]);
1066
1067  /* Set the size of data transferred to the structure */
1068  if (dmac0.size > TRANSFER_SIZE_WORD) {
1069     result_data.trans_size = D_DMAC_TRANS_SIZE;
1070  } else {
1071     result_data.trans_size = D_DMAC_TRANS_SIZE_LITTLE;
1072  }
1073
1074  if (dmac0.dir == OL_TO_DDR) {
1075     result_data.src = D_DMAC_OL_VLADR;     /* Set the source of address structure */
1076     result_data.dst = D_DMAC_SDRAM_VLADR;  /* Set the destination address structure */
1077  } else {
1078     result_data.src = D_DMAC_SDRAM_VLADR;  /* Set the source of address structure */
1079     result_data.dst = D_DMAC_OL_VLADR;     /* Set the destination address structure */
1080  }
1081
1082  /* Set the fraction of the data transferred to the structure */
1083  result_data.fraction = result_data.trans_size % NEWLINE;
1084
1085  /* showing transfer results  of DMAC0 (source) */
1086  dmac0_result_src(result_data);
1087
1088  /* showing transfer results  of DMAC0 (destination) */
1089  dmac0_result_dst(result_data);
1090
1091  printf("\n\r\n\r");
1092 }
1093
1094 /*""FUNC COMMENT""*****************************************************
1095 * ID                :
1096 * Outline           : the transfer results show DMAC0
1097 * Declaration       : void dmac0_result_src(struct result result_data)
1098 * Description       : showing transfer results  of DMAC0 (source)
1099 * Argument          : struct result result_data  : structure for transfer results
1100 * Return Value      : none
```

```
1101 * Calling Functions :
1102 *""FUNC COMMENT END""************************************************/
1103 void dmac0_result_src(struct result result_data)
1104 {
1105  struct result tmp;
1106  tmp = result_data;
1107
1108  /* Displaying range of source */
1109  printf("  Source address:\n\r");
1110  if (dmac0.mode == TRANSFER_MODE_REPEAT |
1111     dmac0.multi_mode == TRANSFER_MULTI_SCATTER)
1112     printf("  H'%x - H'%x\n\r",(unsigned long)result_data.src,
1113                              (unsigned long)result_data.src + (result_data.trans_size / 2) -1);
1114  else if (dmac0.mode == TRANSFER_MODE_RELOAD)
1115     printf("  H'%x - H'%x\n\r",(unsigned long)result_data.src,
1116                              (unsigned long)result_data.src + dmac0.size -1);
1117  else if (dmac0.multi_mode != TRANSFER_MULTI_MULTI)
1118     printf("  H'%x - H'%x\n\r",(unsigned long)result_data.src,
1119                              (unsigned long)result_data.src + result_data.trans_size -1);
1120
1121  /* showing source data */
1122  switch (dmac0.mode) {
1123     case TRANSFER_MODE_NORMAL:
1124        /* showing normal mode */
1125        print_result(dmac0.size,tmp,tmp.src);
1126        break;
1127     case TRANSFER_MODE_REPEAT:
1128        tmp.trans_size /= 2;
1129        tmp.fraction /= 2;
1130        /* showing repeat mode */
1131        print_result(dmac0.size,tmp,tmp.src);
1132        break;
1133     case TRANSFER_MODE_RELOAD:
1134        if (dmac0.size > TRANSFER_SIZE_LONG_WORD) {
1135           tmp.trans_size = dmac0.size;
1136           tmp.fraction = 0;
1137        } else {
1138           tmp.trans_size = 0;
1139           tmp.fraction = dmac0.size;
1140        }
1141        /* showing reload mode */
1142        print_result(dmac0.size,tmp,tmp.src);
1143        break;
1144     case TRANSFER_MODE_MULTI:
1145        /* showing multi-dimensional mode */
1146        dmac0_result_src_multi(tmp);
1147        break;
1148  }
1149
1150  printf("\n\r\n\r");
1151
1152  tmp = result_data;
1153
1154  /* showing the source data of non cache area */
1155  if (dmac0.dir == DDR_TO_OL) {
1156     /* setting the non cacheing area address */
1157     tmp.src =  (unsigned char *)(tmp.src + 0x20000000);
1158     printf("  NON-Cachessing Area Source address:\n\r");
```

```
1159
1160     /* Displaying range of source */
1161     if (dmac0.mode == TRANSFER_MODE_REPEAT |
1162        dmac0.multi_mode == TRANSFER_MULTI_SCATTER)
1163        printf("  H'%x - H'%x\n\r",(unsigned long)tmp.src,
1164                                 (unsigned long)tmp.src + (tmp.trans_size / 2) -1);
1165     else if (dmac0.mode == TRANSFER_MODE_RELOAD)
1166        printf("  H'%x - H'%x\n\r",(unsigned long)tmp.src,
1167                                 (unsigned long)tmp.src + dmac0.size -1);
1168     else if (dmac0.multi_mode != TRANSFER_MULTI_MULTI)
1169        printf("  H'%x - H'%x\n\r",(unsigned long)tmp.src,
1170                                 (unsigned long)tmp.src + tmp.trans_size -1);
1171
1172     /* showing source data */
1173     switch (dmac0.mode) {
1174        case TRANSFER_MODE_NORMAL:
1175           /* showing normal mode */
1176           print_result(dmac0.size,tmp,tmp.src);
1177           break;
1178        case TRANSFER_MODE_REPEAT:
1179           tmp.trans_size /= 2;
1180           tmp.fraction /= 2;
1181           /* showing repeat mode */
1182           print_result(dmac0.size,tmp,tmp.src);
1183           break;
1184        case TRANSFER_MODE_RELOAD:
1185           if (dmac0.size > TRANSFER_SIZE_LONG_WORD) {
1186              tmp.trans_size = dmac0.size;
1187              tmp.fraction = 0;
1188           } else {
1189              tmp.trans_size = 0;
1190              tmp.fraction = dmac0.size;
1191           }
1192           /* showing reload mode */
1193           print_result(dmac0.size,tmp,tmp.src);
1194           break;
1195        case TRANSFER_MODE_MULTI:
1196           /* showing multi-dimensional mode */
1197           dmac0_result_src_multi_non_cache_area(tmp);
1198           break;
1199     }
1200  printf("\n\r\n\r");
1201  }
1202 }
1203
1204 /*""FUNC COMMENT""*****************************************************
1205 * ID                :
1206 * Outline           : the transfer results show DMAC0
1207 * Declaration       : void dmac0_result_src_multi( struct result tmp )
1208 * Description       : showing multi-dimensional mode result(source)
1209 * Argument          : struct result tmp  : structure for transfer results
1210 * Return Value      : none
1211 * Calling Functions :
1212 *""FUNC COMMENT END""*************************************************/
1213 void dmac0_result_src_multi(struct result tmp)
1214 {
1215  int multi_mode;
1216
```

```
1217  /* select the offset of the string for display multi-dimensional transfer */
1218  switch (dmac0.size) {
1219     case TRANSFER_SIZE_BYTE:
1220        multi_mode = 0;
1221        break;
1222     case TRANSFER_SIZE_WORD:
1223        multi_mode = 1;
1224        break;
1225     case TRANSFER_SIZE_LONG_WORD:
1226        multi_mode = 2;
1227        break;
1228     case TRANSFER_SIZE_16BYTES:
1229        multi_mode = 3;
1230        break;
1231     case TRANSFER_SIZE_32BYTES:
1232        multi_mode = 4;
1233        break;
1234  }
1235
1236  switch (dmac0.multi_mode) {
1237     case TRANSFER_MULTI_MULTI:
1238        /* showing multi-dimensional mode result */
1239        dmac0_result_multi_multi(tmp.src);
1240        break;
1241     case TRANSFER_MULTI_SCATTER:
1242        tmp.trans_size /= 2;
1243        tmp.fraction /= 2;
1244        /* Results show the transfer */
1245        print_result(dmac0.size,tmp,tmp.src);
1246        break;
1247     case TRANSFER_MULTI_STRIDE:
1248     case TRANSFER_MULTI_GATHER:
1249        /* Predefined text display */
1250        print_result_multi(dmac0.size,tmp,gather[multi_mode]);
1251        break;
1252  }
1253 }
1254
1255 /*""FUNC COMMENT""*****************************************************
1256 * ID                :
1257 * Outline           : the transfer results show DMAC0
1258 * Declaration    : void dmac0_result_src_multi_non_cache_area( struct result tmp )
1259 * Description    : showing non caching area multi-dimensional mode result(source)
1260 * Argument          : struct result tmp  : structure for transfer results
1261 * Return Value      : none
1262 * Calling Functions :
1263 *""FUNC COMMENT END""*************************************************/
1264 void dmac0_result_src_multi_non_cache_area(struct result tmp)
1265 {
1266  switch (dmac0.multi_mode) {
1267     case TRANSFER_MULTI_MULTI:
1268        /* result for multi-dimensional transfer */
1269        dmac0_result_src_multi(tmp);
1270        break;
1271     case TRANSFER_MULTI_GATHER:
1272     case TRANSFER_MULTI_STRIDE:
1273        /* result for gather & stride transfer */
1274        print_result(dmac0.size,tmp,tmp.src);
```

```
1275        break;
1276     case TRANSFER_MULTI_SCATTER:
1277        tmp.trans_size /= 2;
1278        tmp.fraction /= 2;
1279        /* result for scatter transfer */
1280        print_result(dmac0.size,tmp,tmp.src);
1281        break;
1282  }
1283 }
1284
1285 /*""FUNC COMMENT""*****************************************************
1286 * ID                :
1287 * Outline           : the transfer results show DMAC0
1288 * Declaration       : void dmac0_result_dst( struct result result_data )
1289 * Description       : showing transfer results of DMAC0 (destination)
1290 * Argument          : struct result result_data  : structure for transfer results
1291 * Return Value      : none
1292 * Calling Functions :
1293 *""FUNC COMMENT END""*************************************************/
1294 void dmac0_result_dst( struct result result_data )
1295 {
1296  printf("  Destination address:\n\r");
1297
1298  if (dmac0.multi_mode == TRANSFER_MULTI_MULTI) {
1299     /* If transfer is multi-dimensional,displayed in a dedicated processing */
1300     dmac0_result_multi_multi(result_data.dst);
1301  } else {
1302     /* Displaying range of distination */
1303     printf("  H'%x - H'%x\n\r",(unsigned long)result_data.dst,
1304                           (unsigned long)result_data.dst + result_data.trans_size -1);
1305     /* Display of the destination data */
1306     print_result(dmac0.size,result_data,result_data.dst);
1307  }
1308
1309  /* showing for non cache area destination data */
1310  if (dmac0.dir == OL_TO_DDR) {
1311     printf("\n\r\n\r");
1312     /* setting the non cacheing area address */
1313     result_data.dst =  (unsigned char *)(result_data.dst + 0x20000000);
1314
1315     /* Displaying range of destination */
1316     printf("  NON-Cachessing Area Destination address:\n\r");
1317     if (dmac0.multi_mode == TRANSFER_MULTI_MULTI) {
1318        /* If transfer is multi-dimensional,displayed in a dedicated processing */
1319        dmac0_result_multi_multi(result_data.dst);
1320     } else {
1321        printf("  H'%x - H'%x\n\r",(unsigned long)result_data.dst,
1322                              (unsigned long)result_data.dst + result_data.trans_size -1);
1323        /* Display of the destination data */
1324        print_result(dmac0.size,result_data,result_data.dst);
1325     }
1326  }
1327 }
1328
1329 /*""FUNC COMMENT""*****************************************************
1330 * ID                :
1331 * Outline           : the transfer results show DMAC0
1332 * Declaration       : void dmac0_result_multi_multi( char *adr )
```

```
1333 * Description       : showing multi-dimensional mode result
1334 * Argument          : char *adt:The start address of display data
1335 * Return Value      : none
1336 * Calling Functions :
1337 *""FUNC COMMENT END""**************************************************/
1338 void dmac0_result_multi_multi(unsigned char *adr)
1339 {
1340  switch (dmac0.size) {
1341     case TRANSFER_SIZE_BYTE:
1342        dmac0_result_multi_multi_byte(adr);
1343        break;
1344     case TRANSFER_SIZE_WORD:
1345        dmac0_result_multi_multi_word(adr);
1346        break;
1347     case TRANSFER_SIZE_LONG_WORD:
1348        dmac0_result_multi_multi_longword(adr);
1349        break;
1350     case TRANSFER_SIZE_16BYTES:
1351        dmac0_result_multi_multi_16bytes(adr);
1352        break;
1353     case TRANSFER_SIZE_32BYTES:
1354        dmac0_result_multi_multi_32bytes(adr);
1355        break;
1356  }
1357 }
1358
1359 /*""FUNC COMMENT""*****************************************************
1360 * ID                :
1361 * Outline           : the transfer results show DMAC0
1362 * Declaration       : void dmac0_result_multi_multi_byte(char *adr)
1363 * Description       : showing multi-dimensional mode result(byte)
1364 * Argument          : char *adr:The start address of display data
1365 * Return Value      : none
1366 * Calling Functions :
1367 *""FUNC COMMENT END""**************************************************/
1368 void dmac0_result_multi_multi_byte(unsigned char *adr)
1369 {
1370  struct result result_data;
1371
1372  /* Displaying range */
1373  printf("  H'%x - H'%x\n\r",(unsigned long)adr,
1374                       (unsigned long)adr + TRANSFER_SIZE_MULTI_MULTI_BYTE -1);
1375
1376  result_data.trans_size = 0;
1377  result_data.fraction = TRANSFER_SIZE_MULTI_MULTI_BYTE;
1378
1379  /* display of the data */
1380  print_result(TRANSFER_SIZE_BYTE,result_data,adr);
1381 }
1382
1383 /*""FUNC COMMENT""*****************************************************
1384 * ID                :
1385 * Outline           : the transfer results show DMAC0
1386 * Declaration       : void dmac0_result_multi_multi_word(char *adr)
1387 * Description       : showing multi-dimensional mode result(word)
1388 * Argument          : char *adr:The start address of display data
1389 * Return Value      : none
1390 * Calling Functions :
```

```
1391 *""FUNC COMMENT END""************************************************/
1392 void dmac0_result_multi_multi_word(unsigned char *adr)
1393 {
1394  struct result result_data;
1395
1396  /* Displaying range */
1397  printf("  H'%x - H'%x\n\r",(unsigned long)adr,
1398                       (unsigned long)adr + TRANSFER_SIZE_MULTI_MULTI_WORD -1);
1399  result_data.trans_size = 0;
1400  result_data.fraction = TRANSFER_SIZE_MULTI_MULTI_WORD;
1401
1402  /* display of the data */
1403  print_result(TRANSFER_SIZE_WORD,result_data,adr);
1404 }
1405
1406 /*""FUNC COMMENT""*****************************************************
1407 * ID                :
1408 * Outline           : the transfer results show DMAC0
1409 * Declaration       : dmac0_result_multi_multi_longword(char *adr)
1410 * Description       : showing multi-dimensional mode result(long word)
1411 * Argument          : char *adr:The start address of display data
1412 * Return Value      : none
1413 * Calling Functions :
1414 *""FUNC COMMENT END""************************************************/
1415 void dmac0_result_multi_multi_longword(unsigned char *adr)
1416 {
1417  struct result result_data;
1418
1419  /* Displaying range */
1420  printf("  H'%x - H'%x\n\r",(unsigned long)adr,
1421                       (unsigned long)adr + TRANSFER_SIZE_MULTI_MULTI_LONG_WORD -1);
1422  result_data.trans_size = TRANSFER_SIZE_MULTI_MULTI_LONG_WORD;
1423  result_data.fraction = 0;
1424
1425  /* display of the data */
1426  print_result(TRANSFER_SIZE_LONG_WORD,result_data,adr);
1427 }
1428
1429 /*""FUNC COMMENT""*****************************************************
1430 * ID                :
1431 * Outline           : the transfer results show DMAC0
1432 * Declaration       : void dmac0_result_multi_multi_16bytes(char *adr)
1433 * Description       : showing multi-dimensional mode result(16 bytes)
1434 * Argument          : char *adr:The start address of display data
1435 * Return Value      : none
1436 * Calling Functions :
1437 *""FUNC COMMENT END""************************************************/
1438 void dmac0_result_multi_multi_16bytes(unsigned char *adr)
1439 {
1440  struct result result_data;
1441
1442  /* Displaying range */
1443  printf("  H'%x - H'%x\n\r",(unsigned long)adr,
1444                       (unsigned long)adr + TRANSFER_SIZE_MULTI_MULTI_16BYTES -1);
1445  result_data.trans_size = TRANSFER_SIZE_MULTI_MULTI_16BYTES;
1446  result_data.fraction = TRANSFER_SIZE_MULTI_MULTI_16BYTES % NEWLINE;
1447
1448  /* display of the data */
```

```
 print_result(TRANSFER_SIZE_16BYTES,result_data,adr);
}

/*""FUNC COMMENT""************************************************
* ID                :
* Outline           : the transfer results show DMAC0
* Declaration       : void dmac0_result_multi_multi_32bytes(char *adr)
* Description       : showing multi-dimensional mode result(32 bytes)
* Argument          : char *adr:The start address of display data
* Return Value      : none
* Calling Functions :
*""FUNC COMMENT END""*********************************************/
void dmac0_result_multi_multi_32bytes(unsigned char *adr)
{
 struct result result_data;

 /* Displaying range */
 printf("  H'%x - H'%x\n\r",(unsigned long)adr,
                      (unsigned long)adr + TRANSFER_SIZE_MULTI_MULTI_32BYTES -1);
 result_data.trans_size = TRANSFER_SIZE_MULTI_MULTI_32BYTES;
 result_data.fraction = TRANSFER_SIZE_MULTI_MULTI_32BYTES % NEWLINE;

 /* display of the data */
 print_result(TRANSFER_SIZE_32BYTES,result_data,adr);
}

/*""FUNC COMMENT""************************************************
* ID                :
* Outline           : Interrupt handling DMAC0
* Declaration       : void dmac0_interrupt_ch0( void )
* Description       : Interrupt handling DMAC0(channel 0)
* Argument          : none
* Return Value      : none
* Calling Functions :
*""FUNC COMMENT END""*********************************************/
void dmac0_interrupt_ch0( void )
{
 int tmp;

 /* "Repeat transfer" the case of transferring again */
 if (DMAC0.CHCR0.BIT.TE == 1 && DMAC0.CHCR0.BIT.RPT == 3) {
    tmp = DMAC0.CHCR0.BIT.TE;        /* dummy read  for TE clear */
    DMAC0.CHCR0.BIT.TE = 0x00;       /* tranfer end flag clear */
    DMAC0.CHCR0.BIT.IE = 0;          /* interrupt flag clear */
    int_flg = 1;                  /* set flag for transfer end interrupt */
 } else if (DMAC0.CHCR0.BIT.TE == 1 ) {
    DMAC0.CHCR0.BIT.IE = 0;          /* interrupt flag clear */
    int_flg = 1;                  /* set flag for transfer end interrupt */
 }
}

/*""FUNC COMMENT""************************************************
* ID                :
* Outline           : Interrupt handling DMAC0
* Declaration       : void dmac0_interrupt_ch4( void )
* Description       : Interrupt handling DMAC0(channel 4)
* Argument          : none
* Return Value      : none
```

```
1507 * Calling Functions :
1508 *""FUNC COMMENT END""***************************************************/
1509 void dmac0_interrupt_ch4( void )
1510 {
1511  int tmp;
1512
1513  /* "Repeat transfer" the case of transferring again */
1514  if (DMAC0.CHCR4.BIT.TE == 1 && DMAC0.CHCR4.BIT.RPT == 3) {
1515     tmp = DMAC0.CHCR4.BIT.TE;         /* dummy read  for TE clear */
1516     DMAC0.CHCR4.BIT.TE = 0x00;        /* tranfer end flag clear */
1517     DMAC0.CHCR4.BIT.IE = 0;           /* interrupt flag clear */
1518     int_flg = 1;                   /* set flag for transfer end interrupt */
1519  } else if (DMAC0.CHCR4.BIT.TE == 1 ) {
1520     DMAC0.CHCR4.BIT.IE = 0;           /* interrupt flag clear */
1521     int_flg = 1;                   /* set flag for transfer end interrupt */
1522  }
1523 }
```

6.5 Sample Program Listing: dmac1.c

```
0001 /***********************************************************************
0002 ;* DISCLAIMER
0003 ;
0004 ;* This software is supplied by Renesas Electronics Corporation. and is only
0005 ;* intended for use with Renesas products. No other uses are authorized.
0006 ;
0007 ;* This software is owned by Renesas Electronics Corporation. and is protected under
0008 ;* all applicable laws, including copyright laws.
0009 ;
0010 ;* THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES
0011 ;* REGARDING THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY,
0012 ;* INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
0013 ;* PARTICULAR PURPOSE AND NON-INFRINGEMENT.  ALL SUCH WARRANTIES ARE EXPRESSLY
0014 ;* DISCLAIMED.
0015 ;
0016 ;* TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
0017 ;* ELECTRONICS CORPORATION. NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
0018 ;* FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
0019 ;* FOR ANY REASON RELATED TO THE THIS SOFTWARE, EVEN IF RENESAS OR ITS
0020 ;* AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
0021 ;
0022 ;* Renesas reserves the right, without notice, to make changes to this
0023 ;* software and to discontinue the availability of this software.
0024 ;* By using this software, you agree to the additional terms and
0025 ;* conditions found by accessing the following link:
0026 ;* http://www.renesas.com/disclaimer
0027 ;***********************************************************************/
0028 /* Copyright (C) 2011. Renesas Electronics Corporation., All Rights Reserved.*/
0029 /*""FILE COMMENT""************ Technical reference data ****************
0030 ;* System Name  : SH7786 DMAC Sample Program
0031 ;* File Name    : dmac1.c
0032 ;* Abstract     : DMAC1 is transfer process
0033 ;* Version      : Ver 1.00
0034 ;* Device       : SH7786
0035 ;* Tool-Chain   : High-performance Embedded Workshop (Version 4.09.00.007)
0036 ;*              : C/C++ Compiler Package for SuperH Family (V.9.3.2.0)
0037 ;* OS           : None
0038 ;* H/W Platform : SH-4A Board P/N:AP-SH4AD-0A (Manufacturer:ALPHA PROJECT)
0039 ;* Description  : Main routine and common functions
0040 ;* Operation    :
0041 ;* Limitation   :
0042 ;*              :
0043 ;***********************************************************************
0044 ;* History      : 26.Aug.2011 Ver. 1.00 First Release
0045 ;*""FILE COMMENT END""*************************************************/
0046
0047 #include "config.h"
0048 #include "dmac1.h"
0049
0050 struct DMAC_1 dmac1;           /* Structure for storing configuration */
0051
0052 /*""FUNC COMMENT""*****************************************************
0053 * ID                :
0054 * Outline           : DMAC1 channel select
0055 * Declaration       : void dmac1_select_channel( void )
0056 * Description       : DMAC1 channel select
```

```
0057 * Argument           : none
0058 * Return Value       : none
0059 * Calling Functions :
0060 *""FUNC COMMENT END""***************************************************/
0061 void dmac1_select_channel(void)
0062 {
0063  int ret;
0064  char KeyBuff;
0065
0066  /* Structure clear */
0067  dmac1_data_clear();
0068
0069  do {
0070     /* showing DMAC1 channel selection screen */
0071     printf("[DMAC1]\n\r");
0072     printf("  - Chanel Select\n\r");
0073     printf("  1. Chanel 0\n\r");
0074     printf("  2. Chanel 2\n\r");
0075     printf("  r. Return to previous menu\n\n\r");
0076     printf("  Select No:");
0077     /* clear key buffer */
0078     KeyBuff = 0;
0079
0080     /* waiting for input from SCIF */
0081     while( scif_recive_data_byte( &KeyBuff ) != 0)
0082     ;
0083
0084     printf("\n\r");
0085
0086     /* judgment of input characters */
0087     switch (KeyBuff) {
0088        case '1' :
0089           /* channel 0 selected */
0090           dmac1.ch = 0;
0091           break;
0092        case '2' :
0093           /* channel 2 selected */
0094           dmac1.ch = 2;
0095           break;
0096        case 'r' :
0097           /* previous menu selected */
0098           return;
0099        default :
0100           /* selecting an invalid value */
0101           printf("Invalid value. Please selected 1 to 2 or r.\n\r");
0102           break;
0103     }
0104
0105     /* in the case of inputting the value from '1' to '2' */
0106     if (KeyBuff >= '1' && KeyBuff <= '2') {
0107        /* DMAC1 direction to be selected */
0108        ret = dmac1_select_direction();
0109        if (ret == 0)
0110           return;
0111     }
0112  } while(1);
0113
0114 }
```

```
0115
0116
0117 /*""FUNC COMMENT""************************************************
0118 * ID                :
0119 * Outline           : DMAC1 direction select
0120 * Declaration       : int dmac1_select_direction( void )
0121 * Description       : DMAC1 direction select
0122 * Argument          : none
0123 * Return Value      : 0:transfer end,1:canceled menu
0124 * Calling Functions :
0125 *""FUNC COMMENT END""************************************************/
0126 int dmac1_select_direction( void )
0127 {
0128  int ret;
0129  char KeyBuff;
0130
0131  do{
0132     /* showing DMAC1 direction selection screen */
0133     printf("[DMAC1-ch%d]\n\r",dmac1.ch);
0134     printf("  - Direction Select\n\r");
0135     printf("  1. OL memory to DDR3-SDRAM\n\r");
0136     printf("  2. DDR3-SDRAM to OL memory\n\r");
0137     printf("  r. Return to previous menu\n\r");
0138     printf("  Select No:");
0139     /* clear key buffer */
0140     KeyBuff = 0;
0141
0142     /* waiting for input from SCIF */
0143     while( scif_recive_data_byte( &KeyBuff ) != 0)
0144     ;
0145     printf("\n\r");
0146
0147     /* judgment of input characters */
0148     switch (KeyBuff) {
0149        case '1' :
0150           /* OL memory to DDR3SDRAM selected */
0151           dmac1.dir = OL_TO_DDR;
0152           break;
0153        case '2' :
0154           /* DDR3SDRAM to OL memory selected */
0155           dmac1.dir = DDR_TO_OL;
0156           break;
0157        case 'r' :
0158           /* previous menu selected */
0159           return 1;
0160        default :
0161           /* selecting an invalid value */
0162           printf("Invalid value. Please selected 1 to 2 or r.\n\r");
0163           break;
0164     }
0165
0166     /* in the case of inputting the value from '1' to '2' */
0167     if (KeyBuff >= '1' && KeyBuff <= '2') {
0168        /* DMAC1 transfer mode to be selected */
0169        ret = dmac1_select_trmode();
0170        if (ret == 0)
0171           return 0;
0172     }
```

```
0173  }while(1);
0174 }
0175
0176 /*""FUNC COMMENT""*****************************************************
0177 * ID                :
0178 * Outline           : DMAC1 trans mode select
0179 * Declaration       : int dmac1_select_trmode( void )
0180 * Description       : DMAC1 trans mode select
0181 * Argument          : none
0182 * Return Value      : 0:transfer end,1:canceled menu
0183 * Calling Functions :
0184 *""FUNC COMMENT END""*************************************************/
0185 int dmac1_select_trmode( void )
0186 {
0187  int ret;
0188  char KeyBuff;
0189
0190  if (dmac1.ch == 0) {
0191     while (1){
0192        /* showing DMAC1 transfer mode selection screen */
0193        printf("[DMAC1-ch%d-%s]\n\r",dmac1.ch,dmac1_dir_str[dmac1.dir -1]);
0194        printf("  - Transfer mode select\n\r");
0195        printf("  1. Continuous\n\r");
0196        printf("  2. Stride mode\n\r");
0197        printf("  3. Scatter mode\n\r");
0198        printf("  4. Gather mode\n\r");
0199        printf("  r. Return to previous menu\n\r");
0200        printf("  Select No:");
0201        /* clear key buffer */
0202        KeyBuff = 0;
0203
0204        /* waiting for input from SCIF */
0205        while( scif_recive_data_byte( &KeyBuff ) != 0)
0206        ;
0207
0208        printf("\n\r");
0209
0210        /* judgment of input characters */
0211        switch (KeyBuff) {
0212           case '1' :
0213              /* continuous transfer selected */
0214              dmac1.mode = TRANSFER_MODE_CONTINUOUS;
0215              break;
0216           case '2' :
0217              /* stride transfer selected */
0218              dmac1.mode = TRANSFER_MODE_STRIDE;
0219              break;
0220           case '3' :
0221              /* scatter transfer selected */
0222              dmac1.mode = TRANSFER_MODE_SCATTER;
0223              break;
0224           case '4' :
0225              /* gather transfer selected */
0226              dmac1.mode = TRANSFER_MODE_GATHER;
0227              break;
0228           case 'r' :
0229              /* previous menu selected */
0230              return 1;
```

```
0231            default :
0232               /* selecting an invalid value */
0233               printf("Invalid value. Please selected 1 to 4 or r.\n\r");
0234               break;
0235         }
0236
0237         /* in the case of inputting the value from '1' to '4' */
0238         if (KeyBuff >= '1' && KeyBuff <= '4') {
0239            /* DMAC1 transfer size to be selected */
0240            ret = dmac1_select_size();
0241            if (ret == 0)
0242               return 0;
0243         }
0244      }
0245   } else {
0246      /* continuous transfer only can be executed when channel 2 is selected */
0247      dmac1.mode = TRANSFER_MODE_CONTINUOUS;
0248      /* DMAC1 transfer size to be selected */
0249      ret = dmac1_select_size();
0250      if (ret == 0)
0251         return 0;
0252      else
0253         return 1;
0254   }
0255 }
0256
0257 /*""FUNC COMMENT""*****************************************************
0258 * ID                :
0259 * Outline           : DMAC1 transfer size select
0260 * Declaration       : int dmac1_select_size( void )
0261 * Description       : DMAC1 transfer size select
0262 * Argument          : none
0263 * Return Value      : 0:transfer end,1:canceled menu
0264 * Calling Functions :
0265 *""FUNC COMMENT END""*************************************************/
0266 int dmac1_select_size( void )
0267 {
0268  int ret;
0269
0270  while (1){
0271     if (dmac1.ch == 0) {
0272        /* DMAC1 transfer size select (channel 0) */
0273        ret = dmac1_select_size_ch0();
0274
0275     } else {
0276        /* DMAC1 transfer size select (channel 2) */
0277        ret = dmac1_select_size_ch2();
0278     }
0279
0280     if (ret == 0)
0281        return 0;
0282     else
0283        return 1;
0284  }
0285 }
0286
0287 /*""FUNC COMMENT""*****************************************************
0288 * ID                :
```

```
0289 * Outline           : DMAC1 transfer size select
0290 * Declaration       : int dmac1_select_size_ch0( void )
0291 * Description       : DMAC1 transfer size select (channel 0)
0292 * Argument          : none
0293 * Return Value      : 0:transfer end,1:canceled menu
0294 * Calling Functions :
0295 *""FUNC COMMENT END""***************************************************/
0296 int dmac1_select_size_ch0( void )
0297 {
0298  int ret;
0299  char KeyBuff;
0300
0301  /* showing DMAC1 transfer size selection screen (channel 0) */
0302  printf("[DMAC1-ch%d-%s-%s]\n\r",
0303        dmac1.ch,dmac1_dir_str[dmac1.dir -1],mode_str[dmac1.mode -1]);
0304  printf("  - Transfer data size select\n\r");
0305  printf("  1. Long Words\n\r");
0306  printf("  2. 8bytes\n\r");
0307  printf("  3. 16bytes\n\r");
0308  printf("  4. 32bytes\n\r");
0309  printf("  r. Return to previous menu\n\r");
0310  printf("  Select No:");
0311
0312  /* clear key buffer */
0313  KeyBuff = 0;
0314
0315  /* waiting for input from SCIF */
0316  while( scif_recive_data_byte( &KeyBuff ) != 0)
0317  ;
0318
0319  printf("\n\r");
0320
0321  /* judgment of input characters */
0322  switch (KeyBuff) {
0323     /* transfer size is long word selected */
0324     case '1' :
0325        dmac1.size = TRANSFER_SIZE_LONG_WORD;
0326        break;
0327     /* transfer size is 8bytes selected */
0328     case '2' :
0329        dmac1.size = TRANSFER_SIZE_8BYTES;
0330        break;
0331     /* transfer size is 16bytes selected */
0332     case '3' :
0333        dmac1.size = TRANSFER_SIZE_16BYTES;
0334        break;
0335     /* transfer size is 32bytes selected */
0336     case '4' :
0337        dmac1.size = TRANSFER_SIZE_32BYTES;
0338        break;
0339     /* previous menu selected */
0340     case 'r' :
0341        return 1;
0342     /* selecting an invalid value */
0343     default :
0344        printf("Invalid value. Please selected 1 to 4 or r.\n\r");
0345        break;
0346  }
```

```
0347
0348  /* in the case of inputting the value from '1' to '4' */
0349  if (KeyBuff >= '1' && KeyBuff <= '4') {
0350     /* DMAC1 cache control to be selected */
0351     ret = dmac1_select_cache();
0352     if (ret == 0)
0353        return 0;
0354  }
0355
0356 }
0357
0358 /*""FUNC COMMENT""*****************************************************
0359 * ID                :
0360 * Outline           : DMAC1 transfer size select
0361 * Declaration       : int dmac1_select_size_ch2( void )
0362 * Description       : DMAC1 transfer size select (channel 2)
0363 * Argument          : none
0364 * Return Value      : 0:transfer end,1:canceled menu
0365 * Calling Functions :
0366 *""FUNC COMMENT END""*************************************************/
0367 int dmac1_select_size_ch2( void )
0368 {
0369  int ret;
0370  char KeyBuff;
0371
0372  /* showing DMAC1 transfer size selection screen (channel 2) */
0373  printf("[DMAC1-ch%d-%s-%s]\n\r",
0374        dmac1.ch,dmac1_dir_str[dmac1.dir -1],mode_str[dmac1.mode -1]);
0375  printf("  - Transfer data size select\n\r");
0376  printf("  1. Byte\n\r");
0377  printf("  2. Words\n\r");
0378  printf("  3. Long Words\n\r");
0379  printf("  4. 8bytes\n\r");
0380  printf("  5. 32bytes\n\r");
0381  printf("  r. Return to previous menu\n\r");
0382  printf("  Select No:");
0383
0384  /* clear key buffer */
0385  KeyBuff = 0;
0386  while( scif_recive_data_byte( &KeyBuff ) != 0)
0387  ;
0388
0389  printf("\n\r");
0390
0391  /* waiting for input from SCIF */
0392  switch (KeyBuff) {
0393     /* transfer size is byte selected */
0394     case '1' :
0395        dmac1.size = TRANSFER_SIZE_BYTE;
0396        break;
0397     /* transfer size is word selected */
0398     case '2' :
0399        dmac1.size = TRANSFER_SIZE_BYTE;
0400        break;
0401     /* transfer size is long word selected */
0402     case '3' :
0403        dmac1.size = TRANSFER_SIZE_LONG_WORD;
0404        break;
```

```
0405     /* transfer size is 8bytes selected */
0406     case '4' :
0407        dmac1.size = TRANSFER_SIZE_8BYTES;
0408        break;
0409     /* transfer size is 32bytes selected */
0410     case '5' :
0411        dmac1.size = TRANSFER_SIZE_32BYTES;
0412        break;
0413     /* previous menu selected */
0414     case 'r' :
0415        return 1;
0416     /* selecting an invalid value */
0417     default :
0418        printf("Invalid value. Please selected 1 to 5 or r.\n\r");
0419        break;
0420  }
0421
0422  /* in the case of inputting the value from '1' to '5' */
0423  if (KeyBuff >= '1' && KeyBuff <= '5') {
0424     /* DMAC1 cache control to be selected */
0425     ret = dmac1_select_cache();
0426     if (ret == 0)
0427        return 0;
0428  }
0429
0430 }
0431 /*""FUNC COMMENT""*****************************************************
0432 * ID                :
0433 * Outline           : DMAC1 cache control select
0434 * Declaration       : int dmac1_select_cache( void )
0435 * Description       : DMAC1 cache control select
0436 * Argument          : none
0437 * Return Value      : 0:transfer end,1:canceled menu
0438 * Calling Functions :
0439 *""FUNC COMMENT END""*************************************************/
0440 int dmac1_select_cache( void )
0441 {
0442  int ret;
0443  char KeyBuff;
0444
0445  while (1){
0446     /* showing DMAC1 cache control selection screen */
0447     printf("[DMAC1-ch%d-%s-%s-%dbyte(s)]\n\r",
0448           dmac1.ch, dmac1_dir_str[dmac1.dir -1], mode_str[dmac1.mode -1],
0449           dmac1.size);
0450
0451     printf("  - Cache Select\n\r");
0452     printf("  1. Flush/Purge\n\r");
0453     printf("  2. No Flush/Purge\n\r");
0454     printf("  r. Return to previous menu\n\r");
0455     printf("  Select No:");
0456     /* clear key buffer */
0457     KeyBuff = 0;
0458
0459     /* waiting for input from SCIF */
0460     while( scif_recive_data_byte( &KeyBuff ) != 0)
0461     ;
0462
```

```
0463     printf("\n\r");
0464
0465     /* judgment of input characters */
0466     switch (KeyBuff) {
0467        /* cache control ON selected */
0468        case '1' :
0469           dmac1.cache = SELECT_CACHE_ON;
0470           break;
0471        /* cache control OFF selected */
0472        case '2' :
0473           dmac1.cache = SELECT_CACHE_OFF;
0474           break;
0475        /* previous menu selected */
0476        case 'r' :
0477           return 1;
0478        /* selecting an invalid value */
0479        default :
0480           printf("Invalid value. Please selected 1 to 2 or r.\n\r");
0481           break;
0482     }
0483
0484     /* in the case of inputting the value from '1' to '2' */
0485     if (KeyBuff >= '1' && KeyBuff <= '2') {
0486        /* DAMC0 transfer to be selected */
0487        ret = dmac1_transfer();
0488        if (ret == 0)
0489           return 0;
0490     }
0491  }
0492 }
0493
0494 /*""FUNC COMMENT""*****************************************************
0495 * ID                :
0496 * Outline           : DMAC1 transfer process
0497 * Declaration       : int dmac1_transfer( void )
0498 * Description       : DMAC1 transfer process
0499 * Argument          : none
0500 * Return Value      : 0:transfer end,1:canceled menu
0501 * Calling Functions :
0502 *""FUNC COMMENT END""*************************************************/
0503 int dmac1_transfer( void )
0504 {
0505  int ret;
0506  char KeyBuff;
0507
0508  do{
0509     /* showing DMAC1 transfer process selection screen */
0510     printf("[DMAC1-ch%d-%s-%s-%dbyte(s)-%s]\n\r",
0511           dmac1.ch, dmac1_dir_str[dmac1.dir -1], mode_str[dmac1.mode -1],
0512           dmac1.size,dmac1_cache_str[dmac1.cache -1]);
0513
0514     printf("  - Do you start DMA transfer?\n\r");
0515     printf("  1. Yes\n\r");
0516     printf("  r. Return to previous menu\n\r");
0517     printf("  Select No:");
0518     /* clear key buffer */
0519     KeyBuff = 0;
0520
```

```
0521     /* waiting for input from SCIF */
0522     while( scif_recive_data_byte( &KeyBuff ) != 0)
0523     ;
0524
0525     printf("\n\r");
0526
0527     /* judgment of input characters */
0528     switch (KeyBuff) {
0529        /* transfer start selected */
0530        case '1' :
0531           memory_init(dmac1.dir);          /* memory Initialization */
0532           dmac1_init();                    /* DMAC1 Initialization */
0533           dmac1_start();                /* DMAC1 transfer start */
0534           dmac1_result();               /* showing DMAC1 transfer result */
0535
0536           printf("DMA transfer compleate!!\n\r");
0537           while (1) {
0538              printf("Please hit any key.\n\r");
0539              /* clear key buffer */
0540              KeyBuff = 0;
0541
0542              /* wait for one character */
0543              while( scif_recive_data_byte( &KeyBuff ) != 0)
0544              ;
0545
0546              printf("\n\r");
0547              return 0;
0548           }
0549           break;
0550
0551        /* previous menu selected */
0552        case 'r' :
0553           return 1;
0554        default :
0555           /* selecting an invalid value */
0556           printf("Invalid value. Please selected 1 or r.\n\r");
0557           break;
0558     }
0559
0560  }while(1);
0561 }
0562
0563
0564 /*""FUNC COMMENT""*****************************************************
0565 * ID                :
0566 * Outline           : clear of data storage structure
0567 * Declaration       : void dmac1_data_clear( void )
0568 * Description       : clear of data storage structure
0569 * Argument          : none
0570 * Return Value      : none
0571 * Calling Functions :
0572 *""FUNC COMMENT END""*************************************************/
0573 void dmac1_data_clear( void )
0574 {
0575  dmac1.ch = 0xFF;
0576  dmac1.dir = 0;
0577  dmac1.mode = 0;
0578  dmac1.size = 0;
```

```
0579  dmac1.cache = 0;
0580 }
0581
0582
0583 /*""FUNC COMMENT""*****************************************************
0584 * ID                :
0585 * Outline           : Initialization of DMAC1
0586 * Declaration       : void dmac1_init( void )
0587 * Description       : Initialization of DMAC1
0588 * Argument          : none
0589 * Return Value      : none
0590 * Calling Functions :
0591 *""FUNC COMMENT END""************************************************/
0592 void dmac1_init( void )
0593 {
0594  volatile int i;
0595
0596  /* Stop the clock supply to DAMC */
0597  CPG.MSTPCR1.BIT.MSTP104 = 1;
0598  CPG.MSTPCR1.BIT.MSTP105 = 1;
0599
0600  /* wait for DMAC stop */
0601  for (i = 0; i < 10000; i++)
0602  ;
0603
0604  /* Start the clock supply to DAMC */
0605  CPG.MSTPCR1.BIT.MSTP104 = 0;
0606  CPG.MSTPCR1.BIT.MSTP105 = 0;
0607
0608  /* wait for DMAC start */
0609  for (i = 0; i < 10000; i++)
0610  ;
0611
0612  /* transfer is channel 0 or channel 2 ? */
0613  if (dmac1.ch == 0)
0614     dmac1_ch0_init();                /* DMAC1 cahnnel 0 init */
0615   else
0616     dmac1_ch2_init();                /* DMAC1 cahnnel 2 init */
0617 }
0618
0619 /*""FUNC COMMENT""*****************************************************
0620 * ID                :
0621 * Outline           : Initialization of DMAC1
0622 * Declaration       : void dmac1_ch0_init( void )
0623 * Description       : Initialization of DMAC1 channel 0
0624 * Argument          : none
0625 * Return Value      : none
0626 * Calling Functions :
0627 *""FUNC COMMENT END""************************************************/
0628 void dmac1_ch0_init( void )
0629 {
0630
0631  /* cache control */
0632  if (dmac1.cache == SELECT_CACHE_ON && dmac1.dir == OL_TO_DDR)
0633     /* Cache Invalidation */
0634     cache_invalidate((void *)D_DMAC_SDRAM_VLADR, D_DMAC_TRANS_SIZE);
0635  else if (dmac1.cache == SELECT_CACHE_ON && dmac1.dir == DDR_TO_OL)
0636     /* Writeback cache data to DDR3SDRAM */
```

```
0637     cache_writeback((void *)D_DMAC_SDRAM_VLADR, D_DMAC_TRANS_SIZE);
0638
0639  /* Settings DMAC1 SAR0 and DAR0 */
0640  if (dmac1.dir == OL_TO_DDR) {
0641     DMAC1.DAR0 = (unsigned long)D_DMAC_SDRAM_ADR;     /* DDR3SDRAM to set a transfer destination address */
0642     DMAC1.SAR0 = (unsigned long)D_DMAC_OL_ADR;           /* OL memory to set a transfer soruce address */
0643  } else {
0644     DMAC1.DAR0 = (unsigned long)D_DMAC_OL_ADR;         /* OL memory to set a transfer destination address */
0645     DMAC1.SAR0 = (unsigned long)D_DMAC_SDRAM_ADR;      /* DDR3SDRAM to set a transfer soruce address */
0646  }
0647
0648  /* Set command chain */
0649  dmac1_cc_set();
0650 }
0651
0652 /*""FUNC COMMENT""*****************************************************
0653 * ID                :
0654 * Outline           : Initialization of DMAC1
0655 * Declaration       : void dmac1_ch2_init( void )
0656 * Description       : Initialization of DMAC1 channel 2
0657 * Argument          : none
0658 * Return Value      : none
0659 * Calling Functions :
0660 *""FUNC COMMENT END""*************************************************/
0661 void dmac1_ch2_init( void )
0662 {
0663  /* Settings DMAC1 SAR2 and DAR2 */
0664  if (dmac1.dir == DDR_TO_OL) {
0665     DMAC1.SAR2 = (unsigned long)D_DMAC_SDRAM_ADR;     /* DDR3SDRAM to set a transfer destination address */
0666     DMAC1.DAR2 = (unsigned long)D_DMAC_OL_ADR;           /* OL memory to set a transfer soruce address */
0667  } else {
0668     DMAC1.SAR2 = (unsigned long)D_DMAC_OL_ADR;         /* OL memory to set a transfer destination address */
0669     DMAC1.DAR2 = (unsigned long)D_DMAC_SDRAM_ADR;      /* DDR3SDRAM to set a transfer soruce address */
0670  }
0671
0672  /* Setting the amount of data transferred */
0673  if (dmac1.size <= TRANSFER_SIZE_WORD) {
0674     DMAC1.DMA1BCNTR2.BIT.BCNT = D_DMAC_TRANS_SIZE_LITTLE;
0675  } else {
0676     DMAC1.DMA1BCNTR2.BIT.BCNT = D_DMAC_TRANS_SIZE;
0677  }
0678
0679  /* cache control */
0680  if (dmac1.cache == SELECT_CACHE_ON && dmac1.dir == DDR_TO_OL)
0681     /* Cache Invalidation */
0682     cache_writeback((void *)D_DMAC_SDRAM_VLADR, DMAC1.DMA1BCNTR2.BIT.BCNT);
0683  else if (dmac1.cache == SELECT_CACHE_ON && dmac1.dir == OL_TO_DDR)
0684     /* Writeback cache data to DDR3SDRAM */
0685     cache_invalidate((void *)D_DMAC_SDRAM_VLADR, DMAC1.DMA1BCNTR2.BIT.BCNT);
0686
0687  /* setting transfer Size */
0688  switch (dmac1.size) {
0689     case TRANSFER_SIZE_BYTE:
0690        DMAC1.DMA1STRS2.BIT.STRS = 0x00;                /* Byte units */
0691        DMAC1.DMA1DTRS2.BIT.DTRS = 0x00;
0692        break;
0693     case TRANSFER_SIZE_WORD:
0694        DMAC1.DMA1STRS2.BIT.STRS = 0x01;                /* Word units */
```

```
0695        DMAC1.DMA1DTRS2.BIT.DTRS = 0x01;
0696        break;
0697     case TRANSFER_SIZE_LONG_WORD:
0698        DMAC1.DMA1STRS2.BIT.STRS = 0x02;                  /* Long word units */
0699        DMAC1.DMA1DTRS2.BIT.DTRS = 0x02;
0700        break;
0701     case TRANSFER_SIZE_8BYTES:
0702        DMAC1.DMA1STRS2.BIT.STRS = 0x03;                     /* 8bytes units */
0703        DMAC1.DMA1DTRS2.BIT.DTRS = 0x03;
0704        break;
0705     case TRANSFER_SIZE_32BYTES:
0706        DMAC1.DMA1STRS2.BIT.STRS = 0x05;                  /* 32bytes units */
0707        DMAC1.DMA1DTRS2.BIT.DTRS = 0x05;
0708        break;
0709  }
0710 }
0711
0712 /*""FUNC COMMENT""*****************************************************
0713 * ID                :
0714 * Outline           : Initialization of DMAC1
0715 * Declaration       : void dmac1_cc_set( void )
0716 * Description       : Set command chain
0717 * Argument          : none
0718 * Return Value      : none
0719 * Calling Functions :
0720 *""FUNC COMMENT END""************************************************/
0721 void dmac1_cc_set( void )
0722 {
0723  DMAC1.DMA1CHCR0.BIT.CCRE = 1;                       /* command chain enable */
0724  DMAC1.DMA1CCAR0.BIT.CCA = (unsigned long)COMMAND_CHAIN_ADDR_1 >> 5;    /* set addres of command chain area */
0725
0726  switch (dmac1.mode) {
0727        /* CONTINUOUS transfer */
0728     case TRANSFER_MODE_CONTINUOUS:
0729        dmac1_cc_cotinuous_set();
0730        break;
0731        /* STRIDE transfer */
0732     case TRANSFER_MODE_STRIDE:
0733        dmac1_cc_stride_set();
0734        break;
0735        /* SCATTER transfer */
0736     case TRANSFER_MODE_SCATTER:
0737        dmac1_cc_scatter_set();
0738        break;
0739        /* GATHER transfer */
0740     case TRANSFER_MODE_GATHER:
0741        dmac1_cc_gather_set();
0742        break;
0743  }
0744 }
0745
0746 /*""FUNC COMMENT""*****************************************************
0747 * ID                :
0748 * Outline           : Initialization of DMAC1
0749 * Declaration       : void dmac1_cc_cotinuous_set( void )
0750 * Description       : Set command chain (cotinuous)
0751 * Argument          : none
0752 * Return Value      : none
```

```
0753 * Calling Functions :
0754 *""FUNC COMMENT END""***************************************************/
0755 void dmac1_cc_cotinuous_set( void )
0756 {
0757  unsigned long *ccadr1,*ccadr2;
0758
0759  ccadr1 = COMMAND_CHAIN_VLADDR_1;
0760  ccadr2 = COMMAND_CHAIN_VLADDR_2;
0761
0762  /* set for command chain 1 */
0763  *ccadr1 = 0xA0000000;               /* set the CHCR0, channel 0 and command chain enable */
0764  *(ccadr1 + 0x01) = 0x00000008;                /* reserve area */
0765  *(ccadr1 + 0x02) = (unsigned long)DMAC1.SAR0;    /* set the SAR0 */
0766  *(ccadr1 + 0x03) = (unsigned long)DMAC1.DAR0;    /* set the DAR0 */
0767  *(ccadr1 + 0x04) = (unsigned long)COMMAND_CHAIN_ADDR_2;/* set the CCAR for next command chain address */
0768  *(ccadr1 + 0x05) = D_DMAC_TRANS_SIZE / 2;/* set the BCNTR0 amount of data transferred */
0769  *(ccadr1 + 0x06) = 0x00;                  /* set the STRR0 without stride */
0770  *(ccadr1 + 0x07) = 0x00;                  /* set the SBCNTR0 without stride */
0771
0772  /* set for command chain 2 */
0773  *ccadr2 = 0x80000000;               /* set the CHCR0, command chain enable  */
0774  *(ccadr2 + 0x01) = 0x00000008;                    /* reserve area */
0775  *(ccadr2 + 0x02) = (unsigned long)DMAC1.SAR0;    /* set the SAR0 */
0776  *(ccadr2 + 0x03) = (unsigned long)DMAC1.DAR0 + D_DMAC_TRANS_SIZE / 2;   /* set the DAR0 */
0777  *(ccadr2 + 0x04) = 0x00;             /* set the CCAR since the command chain to finish, set to 0  */
0778  *(ccadr2 + 0x05) = D_DMAC_TRANS_SIZE / 2;/* set the BCNTR0 amount of data transferred */
0779  *(ccadr2 + 0x06) = 0x00;                  /* set the STRR0 without stride */
0780  *(ccadr2 + 0x07) = 0x00;                  /* set the SBCNTR0 without stride */
0781 }
0782
0783 /*""FUNC COMMENT""*****************************************************
0784 * ID                :
0785 * Outline           : Initialization of DMAC1
0786 * Declaration       : void dmac1_cc_stride_set( void )
0787 * Description       : Set command chain (stride)
0788 * Argument          : none
0789 * Return Value      : none
0790 * Calling Functions :
0791 *""FUNC COMMENT END""***************************************************/
0792 void dmac1_cc_stride_set( void )
0793 {
0794  unsigned long *ccadr1,*ccadr2;
0795
0796  ccadr1 = COMMAND_CHAIN_VLADDR_1;
0797  ccadr2 = COMMAND_CHAIN_VLADDR_2;
0798
0799  /* set for command chain 1 */
0800  *ccadr1 = 0xA3000000;     /* set the CHCR0, channel 0 and command chain enable and
0801                                                          source stride enable */
0802  *(ccadr1 + 0x01) = 0x00000008;                          /* reserve area */
0803  *(ccadr1 + 0x02) = (unsigned long)DMAC1.SAR0;           /* set the SAR0 */
0804  *(ccadr1 + 0x03) = (unsigned long)DMAC1.DAR0;           /* set the DAR0 */
0805  *(ccadr1 + 0x04) = (unsigned long)COMMAND_CHAIN_ADDR_2;/* set the CCAR for next command chain address */
0806  *(ccadr1 + 0x05) = D_DMAC_TRANS_SIZE / 2;       /* set the BCNTR0 amount of data transferred */
0807  *(ccadr1 + 0x06) = (dmac1.size / 2) << 18 | (dmac1.size / 2 << 2);  /* set the STRR0 */
0808  *(ccadr1 + 0x07) = (dmac1.size / 4) << 18 | (dmac1.size / 4 << 2);  /* set the SBCNTR0 */
0809
0810  /* set for command chain 2 */
```

```
0811  *ccadr2 = 0x83000000;    /* set the CHCR0, command chain enable and
0812                                                    source stride enable */
0813  *(ccadr2 + 0x01) = 0x00000008;                       /* reserve area */
0814  *(ccadr2 + 0x02) = (unsigned long)DMAC1.SAR0 ;       /* set the SAR0 */
0815  *(ccadr2 + 0x03) = (unsigned long)DMAC1.DAR0 + D_DMAC_TRANS_SIZE / 2;   /* set the DAR0 */
0816  *(ccadr2 + 0x04) = 0x00;    /* set the CCAR since the command chain to finish, set to 0 */
0817  *(ccadr2 + 0x05) = D_DMAC_TRANS_SIZE / 2;       /* set the BCNTR0 amount of data transferred */
0818  *(ccadr2 + 0x06) = (dmac1.size / 2) << 18 | (dmac1.size / 2 << 2);      /* set the STRR0 */
0819  *(ccadr2 + 0x07) = (dmac1.size / 4) << 18 | (dmac1.size / 4 << 2);      /* set the SBCNTR0 */
0820 }
0821
0822 /*""FUNC COMMENT""*****************************************************
0823 * ID                :
0824 * Outline           : Initialization of DMAC1
0825 * Declaration       : void dmac1_cc_scatter_set( void )
0826 * Description       : Set command chain (scatter)
0827 * Argument          : none
0828 * Return Value      : none
0829 * Calling Functions :
0830 *""FUNC COMMENT END""*************************************************/
0831 void dmac1_cc_scatter_set( void )
0832 {
0833  unsigned long *ccadr1,*ccadr2;
0834
0835  ccadr1 = COMMAND_CHAIN_VLADDR_1;
0836  ccadr2 = COMMAND_CHAIN_VLADDR_2;
0837
0838  /* set for command chain 1 */
0839  *ccadr1 = 0xA3000000;    /* set the CHCR0, channel 0 and command chain enable and
0840                                                    source stride enable */
0841  *(ccadr1 + 0x01) = 0x00000008;                       /* reserve area */
0842  *(ccadr1 + 0x02) = (unsigned long)DMAC1.SAR0;        /* set the SAR0 */
0843  *(ccadr1 + 0x03) = (unsigned long)DMAC1.DAR0;        /* set the DAR0 */
0844  *(ccadr1 + 0x04) = (unsigned long)COMMAND_CHAIN_ADDR_2; /* set the CCAR for next command chain address */
0845  *(ccadr1 + 0x05) = D_DMAC_TRANS_SIZE / 4;/* set the BCNTR0 amount of data transferred */
0846  *(ccadr1 + 0x06) = (dmac1.size / 4 << 18) | dmac1.size / 2 << 2;/* set the STRR0 */
0847  *(ccadr1 + 0x07) = (dmac1.size / 4 << 18) | dmac1.size / 4 << 2;/* set the SBCNTR0 */
0848
0849  /* set for command chain 2 */
0850  *ccadr2 = 0x83000000;          /* set the CHCR0, command chain enable and
0851                                                    source stride enable */
0852  *(ccadr2 + 0x01) = 0x00000008;                       /* reserve area */
0853  *(ccadr2 + 0x02) = (unsigned long)DMAC1.SAR0;        /* set the SAR0 */
0854  *(ccadr2 + 0x03) = (unsigned long)DMAC1.DAR0 + D_DMAC_TRANS_SIZE / 2;   /* set the DAR0 */
0855  *(ccadr2 + 0x04) = 0x00;                /* set the CCAR since the command chain to finish, set to 0 */
0856  *(ccadr2 + 0x05) = D_DMAC_TRANS_SIZE / 4;   /* set the BCNTR0 amount of data transferred */
0857  *(ccadr2 + 0x06) = (dmac1.size / 4 << 18) | dmac1.size / 2 << 2;/* set the STRR0 */
0858  *(ccadr2 + 0x07) = (dmac1.size / 4 << 18) | dmac1.size / 4 << 2;/* set the SBCNTR0 */
0859 }
0860
0861 /*""FUNC COMMENT""*****************************************************
0862 * ID                :
0863 * Outline           : Initialization of DMAC1
0864 * Declaration       : void dmac1_cc_gather_set( void )
0865 * Description       : Set command chain (gather)
0866 * Argument          : none
0867 * Return Value      : none
0868 * Calling Functions :
```

```
0869 *""FUNC COMMENT END""***************************************************/
0870 void dmac1_cc_gather_set( void )
0871 {
0872  unsigned long *ccadr1,*ccadr2;
0873
0874  ccadr1 = COMMAND_CHAIN_VLADDR_1;
0875  ccadr2 = COMMAND_CHAIN_VLADDR_2;
0876
0877  /* set for command chain 1 */
0878  *ccadr1 = 0xA3000000;     /* set the CHCR0, channel 0 and command chain enable and
0879                                                        source stride enable */
0880  *(ccadr1 + 0x01) = 0x00000008;                              /* reserve area */
0881  *(ccadr1 + 0x02) = (unsigned long)DMAC1.SAR0;               /* set the SAR0 */
0882  *(ccadr1 + 0x03) = (unsigned long)DMAC1.DAR0;               /* set the DAR0 */
0883  *(ccadr1 + 0x04) = (unsigned long)COMMAND_CHAIN_ADDR_2;     /* set the CCAR for next command chain address */
0884  *(ccadr1 + 0x05) = D_DMAC_TRANS_SIZE / 2;/* set the BCNTR0 amount of data transferred */
0885  *(ccadr1 + 0x06) = (dmac1.size / 2 << 18) | dmac1.size / 4 << 2;/* set the STRR0 */
0886  *(ccadr1 + 0x07) = (dmac1.size / 4 << 18) | dmac1.size / 4 << 2;/* set the SBCNTR0 */
0887
0888  /* set for command chain 2 */
0889  *ccadr2 = 0x83000000;               /* set the CHCR0, command chain enable and
0890                                                        source stride enable */
0891  *(ccadr2 + 0x01) = 0x00000008;                              /* reserve area */
0892  *(ccadr2 + 0x02) = (unsigned long)DMAC1.SAR0;               /* set the SAR0 */
0893  *(ccadr2 + 0x03) = (unsigned long)DMAC1.DAR0 + D_DMAC_TRANS_SIZE / 2;   /* set the DAR0 */
0894  *(ccadr2 + 0x04) = 0x00;       /* set the CCAR since the command chain to finish, set to 0 */
0895  *(ccadr2 + 0x05) = D_DMAC_TRANS_SIZE / 2;      /* set the BCNTR0 amount of data transferred */
0896  *(ccadr2 + 0x06) = (dmac1.size / 2 << 18) | dmac1.size / 4 << 2;    /* set the STRR0 */
0897  *(ccadr2 + 0x07) = (dmac1.size / 4 << 18)| (dmac1.size / 4 << 2);      /* set the SBCNTR0 */
0898 }
0899
0900 /*""FUNC COMMENT""*************************************************
0901 * ID                :
0902 * Outline           : DMAC1 starting transfer
0903 * Declaration       : void dmac1_start( void )
0904 * Description       : DMAC1 starting transfer
0905 * Argument          : none
0906 * Return Value      : none
0907 * Calling Functions :
0908 *""FUNC COMMENT END""***************************************************/
0909 void dmac1_start( void )
0910 {
0911  DMAC1.DMA1OR.BIT.DMA1E = 0x01;          /* Enables DMA transfers */
0912
0913  if (dmac1.ch == 0) {
0914     DMAC1.DMA1CHCR0.BIT.CHE = 0x01;      /* DMAC1 channel 0 enable */
0915     while (DMAC1.DMA1CHSR0.BIT.TE != 1)  /* Waiting for the transfer end */
0916     ;
0917  } else {
0918     DMAC1.DMA1CHCR2.BIT.CHE = 0x01;      /* DMAC1 channel 2 enable */
0919     while (DMAC1.DMA1CHSR2.BIT.TE != 1)  /* Waiting for the transfer end */
0920     ;
0921  }
0922
0923  /* process for  after the transfer */
0924  if (dmac1.ch == 0) {
0925     DMAC1.DMA1CHCR0.BIT.CHE = 0x00;      /* DMA channel 0 transfer disebled */
0926     DMAC1.DMA1CHSR0.BIT.TE = 0x1;        /* tranfer end flag clear */
```

```
0927
0928  } else {
0929     DMAC1.DMA1CHCR2.BIT.CHE = 0x00;      /* DMA channel 2 transfer disebled */
0930     DMAC1.DMA1CHSR2.BIT.TE = 0x1;        /* tranfer end flag clear */
0931
0932  }
0933
0934  DMAC1.DMA1OR.BIT.DMA1E = 0x00;          /* disables DMA transfers */
0935 }
0936
0937
0938 /*""FUNC COMMENT""*****************************************************
0939 * ID                :
0940 * Outline           : the transfer results show DMAC1
0941 * Declaration       : void dmac1_result( void )
0942 * Description       : the transfer results show DMAC1
0943 * Argument          : none
0944 * Return Value      : none
0945 * Calling Functions :
0946 *""FUNC COMMENT END""*************************************************/
0947 void dmac1_result( void )
0948 {
0949  struct result result_data;
0950
0951  /* Displaying the transfer settings */
0952  printf("[DMAC1-ch%d-%s-%s-%dbyte(s)-%s]\n\r",
0953        dmac1.ch, dmac1_dir_str[dmac1.dir -1], mode_str[dmac1.mode -1],
0954        dmac1.size,dmac1_cache_str[dmac1.cache -1]);
0955
0956  /* Set the size of data transferred to the structure */
0957  if (dmac1.size > TRANSFER_SIZE_WORD) {
0958     result_data.trans_size = D_DMAC_TRANS_SIZE;
0959  } else {
0960     result_data.trans_size = D_DMAC_TRANS_SIZE_LITTLE;
0961  }
0962
0963  if (dmac1.dir == OL_TO_DDR) {
0964     result_data.src = D_DMAC_OL_VLADR;       /* Set the source of address structure */
0965     result_data.dst = D_DMAC_SDRAM_VLADR;    /* Set the destination address structure */
0966  } else {
0967     result_data.src = D_DMAC_SDRAM_VLADR;    /* Set the source of address structure */
0968     result_data.dst = D_DMAC_OL_VLADR;       /* Set the destination address structure */
0969  }
0970
0971  /* Set the fraction of the data transferred to the structure */
0972  result_data.fraction = result_data.trans_size % NEWLINE;
0973
0974  /* showing transfer results  of DMAC1 (source) */
0975  dmac1_result_src(result_data);
0976
0977  /* showing transfer results  of DMAC1 (destination) */
0978  dmac1_result_dst(result_data);
0979
0980  printf("\n\r\n\r");
0981 }
0982
0983 /*""FUNC COMMENT""*****************************************************
0984 * ID                :
```

```
0985 * Outline           : the transfer results show DMAC1
0986 * Declaration       : void dmac1_result_src(struct result result_data)
0987 * Description       : showing transfer results  of DMAC1 (source)
0988 * Argument          : struct result result_data  : structure for transfer results
0989 * Return Value      : none
0990 * Calling Functions :
0991 *""FUNC COMMENT END""**************************************************/
0992 void dmac1_result_src(struct result result_data)
0993 {
0994  struct result tmp;
0995  int mode;
0996
0997  tmp = result_data;
0998
0999  /* Displaying range of source */
1000  printf("  Source address:\n\r");
1001  if (dmac1.mode == TRANSFER_MODE_SCATTER )
1002  printf("  H'%x - H'%x\n\r",(unsigned long)result_data.src,
1003                           (unsigned long)result_data.src + (result_data.trans_size -1) / 4);
1004  else if(dmac1.mode != TRANSFER_MODE_GATHER && dmac1.ch == 0)
1005     printf("  H'%x - H'%x\n\r",(unsigned long)result_data.src,
1006                           (unsigned long)result_data.src + (result_data.trans_size -1) / 2);
1007  else
1008     printf("  H'%x - H'%x\n\r",(unsigned long)result_data.src,
1009                           (unsigned long)result_data.src + result_data.trans_size -1);
1010
1011  /* Offset of the string for display of the stride and Gather transfer */
1012  switch (dmac1.size) {
1013     case TRANSFER_SIZE_BYTE:
1014        mode = 0;
1015        break;
1016     case TRANSFER_SIZE_WORD:
1017        mode = 1;
1018        break;
1019     case TRANSFER_SIZE_LONG_WORD:
1020        mode = 2;
1021        break;
1022     case TRANSFER_SIZE_8BYTES:
1023        mode = 3;
1024        break;
1025     case TRANSFER_SIZE_16BYTES:
1026        mode = 4;
1027        break;
1028     case TRANSFER_SIZE_32BYTES:
1029        mode = 5;
1030        break;
1031  }
1032
1033  switch (dmac1.mode) {
1034     case TRANSFER_MODE_CONTINUOUS:
1035        if (dmac1.ch == 0) {
1036           tmp.trans_size /= 2;
1037           tmp.fraction /= 2;
1038        }
1039
1040        /* Results show the transfer */
1041        print_result(dmac1.size,tmp,result_data.src);
1042        break;
```

```
1043     case TRANSFER_MODE_STRIDE:
1044        tmp.trans_size /= 2;
1045        tmp.fraction /= 2;
1046
1047        /* Predefined text display */
1048        print_result_multi(dmac1.size,tmp,dmac1_stride[mode]);
1049        break;
1050     case TRANSFER_MODE_GATHER:
1051        /* Predefined text display */
1052        print_result_multi(dmac1.size,tmp,dmac1_stride[mode]);
1053        break;
1054     case TRANSFER_MODE_SCATTER:
1055        tmp.trans_size /= 4;
1056        tmp.fraction /= 4;
1057
1058        /* Predefined text display */
1059        print_result(dmac1.size,tmp,result_data.src);
1060  }
1061
1062  printf("\n\r\n\r");
1063
1064  /* showing for non cache area source data */
1065  if (dmac1.dir == DDR_TO_OL) {
1066     /* setting the non cacheing area address */
1067     tmp = result_data;
1068     tmp.src =  (unsigned char *)(tmp.src + 0x20000000);
1069
1070     printf("  NON-Cachessing Area Source address:\n\r");
1071
1072     /* Displaying range of source */
1073     if (dmac1.mode == TRANSFER_MODE_SCATTER )
1074     printf("  H'%x - H'%x\n\r",(unsigned long)tmp.src,
1075                              (unsigned long)tmp.src + (tmp.trans_size -1) / 4);
1076     else if(dmac1.mode != TRANSFER_MODE_GATHER && dmac1.ch == 0)
1077        printf("  H'%x - H'%x\n\r",(unsigned long)tmp.src,
1078                              (unsigned long)tmp.src + (tmp.trans_size -1) / 2);
1079     else
1080        printf("  H'%x - H'%x\n\r",(unsigned long)tmp.src,
1081                              (unsigned long)tmp.src + tmp.trans_size -1);
1082
1083     switch (dmac1.mode) {
1084        case TRANSFER_MODE_CONTINUOUS:
1085           if (dmac1.ch == 0) {
1086              tmp.trans_size /= 2;
1087              tmp.fraction /= 2;
1088           }
1089
1090           /* result for continuous mode */
1091           print_result(dmac1.size,tmp,tmp.src);
1092           break;
1093           case TRANSFER_MODE_STRIDE:
1094              tmp.trans_size /= 2;
1095              tmp.fraction /= 2;
1096
1097              /* result for stride mode */
1098              print_result(dmac1.size,tmp,tmp.src);
1099              break;
1100           case TRANSFER_MODE_SCATTER:
```

```
1101              tmp.trans_size /= 4;
1102              tmp.fraction /= 4;
1103
1104              /* result for scatter mode */
1105              print_result(dmac1.size,tmp,tmp.src);
1106              break;
1107           case TRANSFER_MODE_GATHER:
1108              /* result for gather mode */
1109              print_result(dmac1.size,tmp,tmp.src);
1110              break;
1111
1112     }
1113     printf("\n\r\n\r");
1114  }
1115 }
1116
1117 /*""FUNC COMMENT""*****************************************************
1118 * ID                :
1119 * Outline           : the transfer results show DMAC1
1120 * Declaration       : void dmac1_result_dst(struct result result_data)
1121 * Description       : showing transfer results  of DMAC1 (distination)
1122 * Argument          : struct result result_data  : structure for transfer results
1123 * Return Value      : none
1124 * Calling Functions :
1125 *""FUNC COMMENT END""*************************************************/
1126 void dmac1_result_dst(struct result result_data)
1127 {
1128  struct result tmp;
1129  tmp = result_data;
1130
1131  /* Displaying range of distination */
1132  printf("  Destination address:\n\r");
1133  printf("  H'%x - H'%x\n\r",(unsigned long)tmp.dst,
1134                       (unsigned long)tmp.dst + tmp.trans_size -1);
1135
1136  /* display of the destination data */
1137  print_result(dmac1.size,tmp,tmp.dst);
1138
1139  /* showing for non cache area destination data */
1140  if (dmac1.dir == OL_TO_DDR) {
1141     printf("\n\r\n\r");
1142     /* setting the non cacheing area address */
1143     tmp.dst =  (unsigned char *)(tmp.dst + 0x20000000);
1144
1145     /* Displaying range of distination */
1146     printf("  NON-Cachessing Area Destination address:\n\r");
1147     printf("  H'%x - H'%x\n\r",(unsigned long)tmp.dst,
1148                          (unsigned long)tmp.dst + tmp.trans_size -1);
1149
1150     /* display of the destination data */
1151     print_result(dmac1.size,tmp,tmp.dst);
1152  }
1153 }
```

6.6 Sample Program Listing: hpbdamc.c

```
001 /***********************************************************************
002 ;* DISCLAIMER
003 ;
004 ;* This software is supplied by Renesas Electronics Corporation. and is only
005 ;* intended for use with Renesas products. No other uses are authorized.
006 ;
007 ;* This software is owned by Renesas Electronics Corporation. and is protected under
008 ;* all applicable laws, including copyright laws.
009 ;
010 ;* THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES
011 ;* REGARDING THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY,
012 ;* INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
013 ;* PARTICULAR PURPOSE AND NON-INFRINGEMENT.  ALL SUCH WARRANTIES ARE EXPRESSLY
014 ;* DISCLAIMED.
015 ;
016 ;* TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
017 ;* ELECTRONICS CORPORATION. NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
018 ;* FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
019 ;* FOR ANY REASON RELATED TO THE THIS SOFTWARE, EVEN IF RENESAS OR ITS
020 ;* AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
021 ;
022 ;* Renesas reserves the right, without notice, to make changes to this
023 ;* software and to discontinue the availability of this software.
024 ;* By using this software, you agree to the additional terms and
025 ;* conditions found by accessing the following link:
026 ;* http://www.renesas.com/disclaimer
027 ;***********************************************************************/
028 /* Copyright (C) 2011. Renesas Electronics Corporation., All Rights Reserved.*/
029 /*""FILE COMMENT""************ Technical reference data ****************
030 ;* System Name  : SH7786 DMAC Sample Program
031 ;* File Name    : hpbdmac.c
032 ;* Abstract     : HPB-DMAC is transfer process
033 ;* Version      : Ver 1.00
034 ;* Device       : SH7786
035 ;* Tool-Chain   : High-performance Embedded Workshop (Version 4.09.00.007)
036 ;*              : C/C++ Compiler Package for SuperH Family (V.9.3.2.0)
037 ;* OS           : None
038 ;* H/W Platform : SH-4A Board P/N:AP-SH4AD-0A (Manufacturer:ALPHA PROJECT)
039 ;* Description  : Main routine and common functions
040 ;* Operation    :
041 ;* Limitation   :
042 ;*              :
043 ;***********************************************************************
044 ;* History      : 26.Aug.2011 Ver. 1.00 First Release
045 ;*""FILE COMMENT END""*************************************************/
046
047 #include "config.h"
048 #include "hpbdmac.h"
049
050 struct HPB_DMAC hpbdmac;        /* Structure for storing configuration */
051 int rem_cnt;                  /* counter of remain characters */
052
053 /*""FUNC COMMENT""*****************************************************
054 * ID               :
055 * Outline          : HPB-DMAC direction select
056 * Declaration      : void hpbdmac_select_direction( void )
```

```
057 * Description       : HPB-DMAC direction select
058 * Argument          : none
059 * Return Value      : none
060 * Calling Functions :
061 *""FUNC COMMENT END""***************************************************/
062 void hpbdmac_select_direction( void )
063 {
064   int ret;
065   char KeyBuff;
066
067   /* Structure clear */
068   hpbdmac_data_clear();
069
070   while (1){
071      /* showing HPB-DMAC direction selection screen */
072      printf("[HPB]\n\r");
073      printf("  - Direction Select\n\r");
074      printf("  1. DDR3-SDRAM to SCIF\n\r");
075      printf("  2. SCIF to DDR3-SDRAM\n\r");
076      printf("  r. Return to previous menu\n\r");
077      printf("  Select No:");
078      /* clear key buffer */
079      KeyBuff = 0;
080
081      /* waiting for input from SCIF */
082      while( scif_recive_data_byte( &KeyBuff ) != 0)
083      ;
084
085      printf("\n\r");
086
087      /* judgment of input characters */
088      switch (KeyBuff) {
089         /* DDR3SDRAM to SCIF selected */
090         case '1' :
091            hpbdmac.dir = DDR_TO_SCIF;
092            break;
093         /* SCIF to DDR3SDRAM selected */
094         case '2' :
095            hpbdmac.dir = SCIF_TO_DDR;
096            break;
097         /* previous menu selected */
098         case 'r' :
099            return;
100         /* selecting an invalid value */
101         default :
102            printf("Invalid value. Please selected 1 to 2 or r.\n\r");
103            break;
104      }
105
106      /* in the case of inputting the value from '1' to '2' */
107      if (KeyBuff >= '1' && KeyBuff <= '2') {
108         /* HPB-DMAC transfer mode to be selected */
109         ret = hpbdmac_select_trmode();
110         if (ret == 0)
111            return;
112      }
113   }
114 }
```

```
115
116 /*""FUNC COMMENT""*****************************************************
117 * ID                :
118 * Outline           : HPB-DMAC trans mode select
119 * Declaration       : int hpbdmac_select_trmode( void )
120 * Description       : HPB-DMAC trans mode select
121 * Argument          : none
122 * Return Value      : 0:transfer end,1:canceled menu
123 * Calling Functions :
124 *""FUNC COMMENT END""*************************************************/
125 int hpbdmac_select_trmode( void )
126 {
127   int ret;
128   char KeyBuff;
129
130   while (1){
131      /* showing HPB-DMAC transfer mode selection screen */
132      printf("[HPB-%s]\n\r",hpbdmac_dir_str[hpbdmac.dir -1]);
133      printf("  - Transfer mode select\n\r");
134      printf("  1. Single\n\r");
135      printf("  2. Continuous DMA transfer mode\n\r");
136      printf("  r. Return to previous menu\n\r");
137      printf("  Select No:");
138      /* clear key buffer */
139      KeyBuff = 0;
140
141      /* waiting for input from SCIF */
142      while( scif_recive_data_byte( &KeyBuff ) != 0)
143      ;
144
145      printf("\n\r");
146
147      /* judgment of input characters */
148      switch (KeyBuff) {
149         /* single transfer selected */
150         case '1' :
151            hpbdmac.mode = TRANSFER_MODE_NORMAL;
152            break;
153         /* continuous transfer selected */
154         case '2' :
155            hpbdmac.mode = TRANSFER_MODE_CONTINUOUS;
156            break;
157         /* previous menu selected */
158         case 'r' :
159            return 1;
160         /* selecting an invalid value */
161         default :
162            printf("Invalid value. Please selected 1 to 2 or r.\n\r");
163            break;
164      }
165
166      /* in the case of inputting the value of '1' */
167      if (KeyBuff == '1' ) {
168         /* HPB-DMAC cache control to be selected */
169         ret = hpbdmac_select_cache();
170         if (ret == 0)
171            return 0;
172      }
```

```
173
174      /* in the case of inputting the value of '2' */
175      if (KeyBuff == '2' ) {
176         /* HPB-DMAC Automatic continuous transfer to be selected */
177         ret = hpbdmac_select_automatic();
178         if (ret == 0)
179            return 0;
180      }
181
182   }
183 }
184
185 /*""FUNC COMMENT""*****************************************************
186 * ID                :
187 * Outline           : HPB-DMAC Automatic continuous transfer select
188 * Declaration       : int hpbdmac_select_automatic( void )
189 * Description       : HPB-DMAC Automatic continuous transfer select
190 * Argument          : none
191 * Return Value      : 0:transfer end,1:canceled menu
192 * Calling Functions :
193 *""FUNC COMMENT END""*************************************************/
194 int hpbdmac_select_automatic( void )
195 {
196   int ret;
197   char KeyBuff;
198
199   while (1){
200      /* showing HPB-DMAC Automatic continuous transfer selection screen */
201      printf("[HPB-%s-Continuous-1byte]\n\r",
202            hpbdmac_dir_str[hpbdmac.dir -1]);
203      printf("  - DMA infomation select\n\r");
204      printf("  1. Non-automatic continuous\n\r");
205      printf("  2. Automatic continuous\n\r");
206      printf("  r. Return to previous menu\n\n\r");
207      printf("  Select No:");
208      /* clear key buffer */
209      KeyBuff = 0;
210
211      /* waiting for input from SCIF */
212      while( scif_recive_data_byte( &KeyBuff ) != 0)
213      ;
214
215      printf("\n\r");
216
217      /* judgment of input characters */
218      switch (KeyBuff) {
219         /* non automatic continuous transfer selected */
220         case '1' :
221            hpbdmac.automatic = TRANSFER_NON_AUTOMATIC;
222            break;
223         case '2' :
224         /* automatic continuous transfer selected */
225            hpbdmac.automatic = TRANSFER_AUTOMATIC;
226            break;
227         /* previous menu selected */
228         case 'r' :
229            return 1;
230         /* selecting an invalid value */
```

```
231         default :
232            printf("Invalid value. Please selected 1 to 2 or r.\n\r");
233            break;
234      }
235
236      /* in the case of inputting the value from '1' to '2' */
237      if (KeyBuff >= '1' && KeyBuff <= '2') {
238         /* HPB-DMAC cache control to be selected */
239         ret = hpbdmac_select_cache();
240         if (ret == 0)
241            return 0;
242      }
243   }
244 }
245
246 /*""FUNC COMMENT""*****************************************************
247 * ID                :
248 * Outline           : HPB-DMAC cache control select
249 * Declaration       : int hpbdmac_select_cache( void )
250 * Description       : HPB-DMAC cache control select
251 * Argument          : none
252 * Return Value      : 0:transfer end,1:canceled menu
253 * Calling Functions :
254 *""FUNC COMMENT END""*************************************************/
255 int hpbdmac_select_cache( void )
256 {
257   int ret,i,tmp;
258   char KeyBuff;
259
260   while (1){
261      /* showing HPB-DMAC cache control selection screen */
262      if (hpbdmac.mode == TRANSFER_MODE_NORMAL)
263         printf("[HPB-%s-Single-1byte]\n\r",
264               hpbdmac_dir_str[hpbdmac.dir -1],hpbdmac.size);
265      else
266         printf("[HPBDMAC-%s-Continuous-1byte-%s]\n\r",
267            hpbdmac_dir_str[hpbdmac.dir -1],
268            hpbdmac_automatic_str[hpbdmac.automatic -1]);
269
270      printf("  - Cache Select\n\r");
271      printf("  1. Flush/Purge\n\r");
272      printf("  2. No Flush/Purge\n\r");
273      printf("  r. Return to previous menu\n\r");
274      printf("  Select No:");
275      /* clear key buffer */
276      KeyBuff = 0;
277
278      /* waiting for input from SCIF */
279      while( scif_recive_data_byte( &KeyBuff ) != 0)
280      ;
281
282      printf("\n\r");
283
284      /* judgment of input characters */
285      switch (KeyBuff) {
286         /* cache control ON selected */
287         case '1' :
288            hpbdmac.cache = SELECT_CACHE_ON;
```

```
289            break;
290         /* cache control OFF selected */
291         case '2' :
292            hpbdmac.cache = SELECT_CACHE_OFF;
293            break;
294         /* previous menu selected */
295         case 'r' :
296            return 1;
297         /* selecting an invalid value */
298         default :
299            printf("Invalid value. Please selected 1 to 2 or r.\n\r");
300            break;
301      }
302
303      /* in the case of inputting the value from '1' to '2' */
304      if (KeyBuff >= '1' && KeyBuff <= '2') {
305         /* HPB-DMAC transfer to be selected */
306         ret = hpbdmac_transfer();
307         if (ret == 0)
308            return 0;
309      }
310   }
311 }
312
313 /*""FUNC COMMENT""*****************************************************
314 * ID                :
315 * Outline           : HPB-DMAC transfer process
316 * Declaration       : int hpbdmac_transfer( void )
317 * Description       : HPB-DMAC transfer process
318 * Argument          : none
319 * Return Value      : 0:transfer end,1:canceled menu
320 * Calling Functions :
321 *""FUNC COMMENT END""*************************************************/
322 int hpbdmac_transfer( void )
323 {
324   int ret;
325   char KeyBuff;
326
327   while (1){
328      /* showing HPB-DMAC transfer process selection screen */
329      if (hpbdmac.mode == TRANSFER_MODE_NORMAL)
330         printf("[HPB-%s-Single-1byte-%s]\n\r",
331               hpbdmac_dir_str[hpbdmac.dir -1],
332               hpbdmac_cache_str[hpbdmac.cache -1]);
333      else
334         printf("[HPBDMAC-%s-Continuous-1byte-%s-%s]\n\r",
335            hpbdmac_dir_str[hpbdmac.dir -1],
336            hpbdmac_automatic_str[hpbdmac.automatic -1],
337            hpbdmac_cache_str[hpbdmac.cache -1]);
338
339      printf("  - Do you start DMA transfer?\n\r");
340      printf("  1. Yes\n\r");
341      printf("  r. Return to previous menu\n\r");
342      printf("  Select No:");
343      /* clear key buffer */
344      KeyBuff = 0;
345
346      /* waiting for input from SCIF */
```

```
347       while( scif_recive_data_byte( &KeyBuff ) != 0)
348       ;
349
350       printf("\n\r");
351
352       /* judgment of input characters */
353       switch (KeyBuff) {
354          /* transfer start selected */
355          case '1' :
356             hpb_memory_init();                    /* memory initialization */
357             if ( hpbdmac.dir == DDR_TO_SCIF ) {
358                /* show the HPB-DMAC source */
359                hpbdmac_result_src();
360             }
361
362             hpbdmac_init();                       /* HPB-DMAC initialization */
363             hpbdmac_start();                      /* HPB-DMAC transfer start */
364
365             if ( hpbdmac.dir == SCIF_TO_DDR ) {
366                /* show the HPB-DMAC distination */
367                hpbdmac_result_dst();
368             }
369
370             printf("The resulting data is the ASCII code.\n\r");
371             printf("DMA transfer compleate!!\n\r");
372             while (1) {
373                printf("Please hit any key.\n\r");
374                /* clear key buffer */
375                KeyBuff = 0;
376
377                /* wait for one character */
378                while( scif_recive_data_byte( &KeyBuff ) != 0)
379                ;
380
381                printf("\n\r");
382                return 0;
383             }
384             break;
385          /* previous menu selected */
386          case 'r' :
387             return 1;
388          /* selecting an invalid value */
389          default :
390             printf("Invalid value. Please selected 1 or r.\n\r");
391             break;
392       }
393    }
394 }
395
396 /*""FUNC COMMENT""*****************************************************
397 * ID                :
398 * Outline           : clear of data storage structures
399 * Declaration       : void hpbdmac_data_clear( void )
400 * Description       : clear of data storage structures
401 * Argument          : none
402 * Return Value      : none
403 * Calling Functions :
404 *""FUNC COMMENT END""*************************************************/
```

```
405 void hpbdmac_data_clear( void )
406 {
407   hpbdmac.dir = 0;
408   hpbdmac.mode = 0;
409   hpbdmac.size = 0;
410   hpbdmac.automatic = 0;
411   hpbdmac.cache = 0;
412 }
413
414 /*""FUNC COMMENT""*****************************************************
415 * ID                :
416 * Outline           : DDR3SDRAM initialization
417 * Declaration       : void hpb_memory_init( void )
418 * Description       : DDR3SDRAM initialization
419 * Argument          : none
420 * Return Value      : none
421 * Calling Functions :
422 *""FUNC COMMENT END""*************************************************/
423 void hpb_memory_init( void )
424 {
425   int i;
426   unsigned char *p1,*p2;
427
428   p1 = D_DMAC_SDRAM_VLADR;                /* DDR3SDRAM P1 Area Address set */
429   p2 = (unsigned char *)(p1 + 0x20000000);      /* DDR3SDRAM P2 Area Address set */
430
431   if (hpbdmac.dir == DDR_TO_SCIF) {
432      for(i=0; i < 64; i++)
433      {
434         *p1++ = 0x55;               /* Initialization DDR3SDRAM P1 Area at 0x55 */
435         *p2++ = 0x55;               /* Initialization DDR3SDRAM P2 Area at 0x55 */
436      }
437
438      p1 = D_DMAC_SDRAM_VLADR;             /* DDR3SDRAM P1 Area Address set */
439
440      for(i=0; i < 10; i++)
441      {
442         *p1++ = 0x30 + i;                 /* set the value from '0' to '9' */
443      }
444      for(i=0; i < 25; i++)
445      {
446         *p1++ = 0x41 + i;                 /* set the value from 'a' to 'z' */
447      }
448      for(i=0; i < 29; i++)
449      {
450         *p1++ = 0x61 + i;                 /* set the value from 'A' to 'Z' */
451      }
452   } else {
453      for(i=0; i < 8; i++)
454      {
455         *p1++ = 0x55;                        /* set the value from 0x55 */
456      }
457   }
458
459 }
460
461 /*""FUNC COMMENT""*****************************************************
462 * ID                :
```

```
463 * Outline           : the transfer results show HPB-DMAC
464 * Declaration       : void hpbdmac_result_src( void )
465 * Description       : showing transfer results of HPB-DMAC (source)
466 * Argument          : none
467 * Return Value      : none
468 * Calling Functions :
469 *""FUNC COMMENT END""***************************************************/
470 void hpbdmac_result_src( void )
471 {
472   struct result result_data;
473
474   result_data.src = D_DMAC_SDRAM_VLADR;
475
476   /* Displaying range of source */
477   printf("  Source address:\n\r");
478   printf("  H'%x - H'%x\n\r",(unsigned long)result_data.src,
479                         (unsigned long)result_data.src + D_DMAC_TRANS_SIZE_HPB_DDR_TO_SCIF -1);
480
481   /* Set the size of data transferred to the structure */
482   result_data.trans_size = D_DMAC_TRANS_SIZE_HPB_DDR_TO_SCIF;
483   result_data.fraction = 0;
484
485   /* showing transfer results of HPB-DMAC (source) */
486   print_result_hpb(1,result_data,result_data.src);
487
488   printf("\n\r\n\r");
489
490   /* cache control */
491   if (hpbdmac.cache == SELECT_CACHE_ON)
492      /* Writeback cache data to DDR3SDRAM */
493      cache_writeback(D_DMAC_SDRAM_VLADR, D_DMAC_TRANS_SIZE_HPB_DDR_TO_SCIF);
494
495   /* showing for non cache area source data */
496   result_data.src = (char *)(result_data.src + 0x20000000);
497
498   /* Displaying range of source */
499   printf("  NON-Cachessing Area Source address:\n\r");
500   printf("  H'%x - H'%x\n\r",(unsigned long)result_data.src,
501                         (unsigned long)result_data.src + D_DMAC_TRANS_SIZE_HPB_DDR_TO_SCIF -1);
502
503   /* showing transfer results of HPB-DMAC (source) */
504   print_result_hpb(1,result_data,result_data.src);
505   printf("\n\r");
506 }
507
508 /*""FUNC COMMENT""*****************************************************
509 * ID                :
510 * Outline           : the transfer results show HPB-DMAC
511 * Declaration       : void hpbdmac_result_dst( void )
512 * Description       : showing transfer results of HPB-DMAC (destination)
513 * Argument          : none
514 * Return Value      : none
515 * Calling Functions :
516 *""FUNC COMMENT END""***************************************************/
517 void hpbdmac_result_dst( void )
518 {
519   struct result result_data;
520
```

```
521   result_data.dst = D_DMAC_SDRAM_VLADR;
522
523   /* showing HPB-DMAC transfer process selection screen */
524   if (hpbdmac.mode == TRANSFER_MODE_NORMAL)
525      printf("[HPB-%s-Single-1byte-%s]\n\r",
526            hpbdmac_dir_str[hpbdmac.dir -1],
527            hpbdmac_cache_str[hpbdmac.cache -1]);
528   else
529      printf("[HPBDMAC-%s-Continuous-1byte-%s-%s]\n\r",
530            hpbdmac_dir_str[hpbdmac.dir -1],
531            hpbdmac_automatic_str[hpbdmac.automatic -1],
532            hpbdmac_cache_str[hpbdmac.cache -1]);
533
534   /* Displaying range of destination */
535   result_data.trans_size = D_DMAC_TRANS_SIZE_HPB;
536   printf("  Destination address:\n\r");
537   printf("  H'%x - H'%x\n\r",(unsigned long)result_data.dst,
538                        (unsigned long)result_data.dst + D_DMAC_TRANS_SIZE_HPB -1);
539
540   /* Set the size of data transferred to the structure */
541   result_data.trans_size = 0;
542   result_data.fraction = D_DMAC_TRANS_SIZE_HPB;
543
544   /* showing transfer results of HPB-DMAC (destination) */
545   print_result_hpb(1,result_data,result_data.dst);
546
547   printf("\n\r\n\r");
548
549   /* showing for non cache area destination data */
550   result_data.dst = (char *)(result_data.dst + 0x20000000);
551   printf("  NON-Cachessing Area Destination address:\n\r");
552
553   /* Displaying range of destination */
554   printf("  H'%x - H'%x\n\r",(unsigned long)result_data.dst,
555                        (unsigned long)result_data.dst + D_DMAC_TRANS_SIZE_HPB -1);
556
557   /* showing transfer results of HPB-DMAC (destination) */
558   print_result_hpb(1,result_data,result_data.dst);
559
560   printf("\n\r\n\r");
561 }
562
563 /*""FUNC COMMENT""*****************************************************
564 * ID                :
565 * Outline           : Initialization of HPB-DMAC
566 * Declaration       : void hpbdmac_init( void )
567 * Description       : Initialization of HPB-DMAC
568 * Argument          : none
569 * Return Value      : none
570 * Calling Functions :
571 *""FUNC COMMENT END""*************************************************/
572 void hpbdmac_init( void )
573 {
574   volatile int i;
575
576   /* Stop the clock supply to DAMC */
577   CPG.MSTPCR1.BIT.MSTP104 = 1;
578   CPG.MSTPCR1.BIT.MSTP105 = 1;
```

```
579
580   /* wait for DMAC stop */
581   for (i = 0; i < 10000; i++)
582   ;
583
584   /* Start the clock supply to DAMC */
585   CPG.MSTPCR1.BIT.MSTP104 = 0;
586   CPG.MSTPCR1.BIT.MSTP105 = 0;
587
588   /* wait for DMAC start */
589   for (i = 0; i < 10000; i++)
590   ;
591
592   INTC.INT2PRI6.BIT.HPB0 = 0x3;     /* Set interrupt priority HPB-DMAC */
593   INTC.C0INT2MSKCLR1.BIT._HPB0 = 1;    /* HPB-DMAC interrupt mask clear */
594
595   HPBDMAC.DINTMR.BIT.DTEM0 = 1;     /* HPB-DMAC interput enable */
596
597   rem_cnt = 0;                /* clear of remain characters counter */
598
599   /* transfer is DDR to SCIF or SCIF to DDR ? */
600   if (hpbdmac.dir == DDR_TO_SCIF) {
601      hpbdmac_init_ddr_to_scif();     /* Initialization of HPBDMAC, for DDR3SDRAM to SCIF */
602   } else {
603      hpbdmac_init_scif_to_ddr();     /* Initialization of HPBDMAC, for SCIF to DDR3SDRAM */
604   }
605
606   /* Single or Continuous */
607   if (hpbdmac.mode == TRANSFER_MODE_NORMAL) {
608      HPBDMAC0.DCR.BIT.CT = 0;    /* Does not transfer in continuous mode */
609      HPBDMAC0.DCR.BIT.DIP = 0;      /* Uses one DMA information set repeatedly */
610   } else {
611      HPBDMAC0.DCR.BIT.CT = 1;    /* Transfers in continuous mode */
612      HPBDMAC0.DCR.BIT.DIP = 1;      /* Uses two DMA information sets alternately */
613   }
614
615   /* Non-automatic or Automatic */
616   if (hpbdmac.automatic == TRANSFER_AUTOMATIC)
617      HPBDMAC0.DCR.BIT.ACMD = 1;     /* Transfers in automatic continuous mode */
618   else
619      HPBDMAC0.DCR.BIT.ACMD = 0;      /* Does not transfer in automatic continuous mode */
620 }
621
622 /*""FUNC COMMENT""*****************************************************
623 * ID                :
624 * Outline           : Initialization of HPB-DMAC
625 * Declaration       : void hpbdmac_init_ddr_to_scif( void )
626 * Description       : Initialization of the transfer to the SCIF from DDR3SDRAM
627 * Argument          : none
628 * Return Value      : none
629 * Calling Functions :
630 *""FUNC COMMENT END""*************************************************/
631 void hpbdmac_init_ddr_to_scif( void )
632 {
633   HPBDMAC0.DCR.BIT.SDRMD = 1;       /* set the Auto-request */
634   HPBDMAC0.DCR.BIT.DMDL = 1;           /* transfer destination module is Peripheral (SCIF) */
635   HPBDMAC0.DCR.BIT.SMDL = 0;           /* transfer soruce module is memory */
636   HPBDMAC0.DSAR0 = (unsigned long)D_DMAC_SDRAM_ADR;      /* To set the source address of the information area 0 */
```

```
637    HPBDMAC0.DDAR0 = (unsigned long)D_DMAC_SCFRDR0_ADR;    /* To set the destination address of the information area 0  */
638
639    if (hpbdmac.mode == TRANSFER_MODE_NORMAL ) {
640        HPBDMAC0.DTCR0.LONG = D_DMAC_TRANS_SIZE_HPB_DDR_TO_SCIF;    /* Setting the amount of data transferred information area 0 */
641    } else {
642        HPBDMAC0.DTCR0.LONG = D_DMAC_TRANS_SIZE_HPB_DDR_TO_SCIF_HALF;  /* Setting the amount of data transferred information area 0  */
643        HPBDMAC0.DTCR1.LONG = D_DMAC_TRANS_SIZE_HPB_DDR_TO_SCIF_HALF;  /* Setting the amount of data transferred information area 1  */
644        HPBDMAC0.DSAR1 = (unsigned long)D_DMAC_SDRAM_ADR +
645                        D_DMAC_TRANS_SIZE_HPB_DDR_TO_SCIF_HALF;       /* To set the source address of the information area 1  */
646        HPBDMAC0.DDAR1 = (unsigned long)D_DMAC_SCFRDR0_ADR;           /* To set the destination address of the information area 1  */
647    }
648 }
649
650 /*""FUNC COMMENT""*****************************************************
651 * ID                :
652 * Outline           : Initialization of HPB-DMAC
653 * Declaration       : void hpbdmac_init_scif_to_ddr( void )
654 * Description       : Initialization of the transfer to the DDR3SDRAM from SCIF
655 * Argument          : none
656 * Return Value      : none
657 * Calling Functions :
658 *""FUNC COMMENT END""*************************************************/
659 void hpbdmac_init_scif_to_ddr( void )
660 {
661    HPBDMAC0.DCR.BIT.SDRMD = 0 ;      /* set the Module request (SCIF interrupt) */
662    HPBDMAC0.DCR.BIT.DMDL = 0;        /* transfer destination module is memory */
663    HPBDMAC0.DCR.BIT.SMDL = 1;     /* transfer soruce module is Peripheral (SCIF) */
664    HPBDMAC0.DSAR0 = (unsigned long)D_DMAC_SCFTDR0_ADR;        /* To set the source address of the information area 0  */
665    HPBDMAC0.DDAR0 = (unsigned long)D_DMAC_SDRAM_ADR;          /* To set the destination address of the information area 0  */
666
667    if (hpbdmac.mode == TRANSFER_MODE_NORMAL )
668        HPBDMAC0.DTCR0.LONG = D_DMAC_TRANS_SIZE_HPB;      /* Setting the amount of data transferred information area 0  */
669    else {
670        HPBDMAC0.DTCR0.LONG = D_DMAC_TRANS_SIZE_HALF_HPB;            /* Setting the amount of data transferred information area 0  */
671        HPBDMAC0.DTCR1.LONG = D_DMAC_TRANS_SIZE_HALF_HPB;            /* Setting the amount of data transferred information area 1 */
672        HPBDMAC0.DSAR1 = (unsigned long)D_DMAC_SCFTDR0_ADR;              /* To set the source address of the information area 1  */
673        HPBDMAC0.DDAR1 = (unsigned long)D_DMAC_SDRAM_ADR +
674                    D_DMAC_TRANS_SIZE_HALF_HPB;                  /* To set the destination address of the information area 0 */
675    }
676 }
677
678 /*""FUNC COMMENT""*****************************************************
679 * ID                :
680 * Outline           : HPB-DMAC starting transfer
681 * Declaration       : void hpbdmac_start( void )
682 * Description       : HPB-DMAC starting transfer
683 * Argument          : none
684 * Return Value      : none
685 * Calling Functions :
686 *""FUNC COMMENT END""*************************************************/
687 void hpbdmac_start( void )
688 {
689
690    if (hpbdmac.dir == DDR_TO_SCIF) {
691       /* DDR3SDRAM to SCIF transfer */
692       trans_ddr_to_scif();
693    } else {
694       /* SCIF to DDR3SDRAM transfer */
```

```
695       trans_scif_to_ddr();
696     }
697
698     printf("\n\r");
699   }
700
701   /*""FUNC COMMENT""*****************************************************
702   * ID                :
703   * Outline           : HPB-DMAC starting transfer
704   * Declaration       : void trans_ddr_to_scif( void )
705   * Description       : DDR3SDRAM to SCIF transfer
706   * Argument          : none
707   * Return Value      : none
708   * Calling Functions :
709   *""FUNC COMMENT END""*************************************************/
710   void trans_ddr_to_scif( void )
711   {
712     printf("\n\r");
713     printf("  Destination data :\n\r");
714     printf("  ");
715
716     SCIF0.SCSCR.WORD = 0x0000;          /* TIE, RIE, TE, RE clear */
717     HPBDMAC0.DCMDR.BIT.DMEN = 1;        /* Activates DMA transfer */
718
719     /* SCIF init */
720     scif_init();
721
722     /* wait for DMA transfer has been completed */
723     while (HPBDMAC0.DSTSR.BIT.DMSTS == 1)
724     ;
725
726     HPBDMAC0.DCR.LONG = 0x00000000;       /* DCR clear */
727     printf("\n\r");
728
729   }
730
731   /*""FUNC COMMENT""*****************************************************
732   * ID                :
733   * Outline           : HPB-DMAC starting transfer
734   * Declaration       : void trans_scif_to_ddr( void )
735   * Description       : SCIF to DDR3SDRAM transfer
736   * Argument          : none
737   * Return Value      : none
738   * Calling Functions :
739   *""FUNC COMMENT END""*************************************************/
740   void trans_scif_to_ddr( void )
741   {
742     printf("\n\r");
743     printf("Please input 8 characters:");
744
745     /* cache control */
746     if (hpbdmac.cache == SELECT_CACHE_ON)
747       /* Cache Invalidation */
748       cache_purge((void *)D_DMAC_SDRAM_VLADR, D_DMAC_TRANS_SIZE_HPB);
749
750     if (hpbdmac.mode == TRANSFER_MODE_NORMAL) {
751       /* single transfer */
752       trans_scif_to_ddr_normal();
```

```
753   } else {
754      /* continuous transfer */
755      trans_scif_to_ddr_continuous();
756   }
757
758   HPBDMAC0.DCR.LONG = 0x00000000;            /* DCR clear */
759   SCIF0.SCSCR.WORD = 0x0000;              /* TIE, RIE, TE, RE Clear */
760   scif_init();                       /* SCIF init */
761   printf("\n\r\n\r");
762 }
763
764 /*""FUNC COMMENT""*****************************************************
765 * ID                :
766 * Outline           : HPB-DMAC starting transfer
767 * Declaration       : void trans_scif_to_ddr_normal( void )
768 * Description       : SCIF to DDR3SDRAM transfer (normal mode)
769 * Argument          : none
770 * Return Value      : none
771 * Calling Functions :
772 *""FUNC COMMENT END""*************************************************/
773 void trans_scif_to_ddr_normal( void )
774 {
775   int cnt;
776   char *tmp;
777
778   tmp = D_DMAC_SDRAM_VLADR;
779   tmp = (char *)(tmp + 0x20000000);        /* Set the start address of a point echo back */
780
781   SCIF0.SCSCR.WORD = 0x0000;              /* TIE, RIE, TE, RE Clear */
782   HPBDMAC0.DCMDR.BIT.DMEN = 1;            /* Activates DMA transfer */
783
784   /* SCIF init */
785   scif_init();
786
787   cnt = D_DMAC_TRANS_SIZE_HPB;            /* Set the counter for echo back */
788
789   while(cnt > 0) {
790      /* Characters are entered from SCIF? */
791      if (cnt > HPBDMAC0.DTCSR.BIT.DTCS) {
792         if (hpbdmac.cache == SELECT_CACHE_ON)
793            /* Cache Invalidation */
794            cache_purge((void *)D_DMAC_SDRAM_VLADR, D_DMAC_TRANS_SIZE_HPB);
795
796         /*  Echo back the one character */
797         scif_transmit_data_byte(tmp);
798         cnt -= 1;
799         tmp++;
800      }
801   }
802 }
803
804 /*""FUNC COMMENT""*****************************************************
805 * ID                :
806 * Outline           : HPB-DMAC starting transfer
807 * Declaration       : void trans_scif_to_ddr_continuous( void )
808 * Description       : SCIF to DDR3SDRAM transfer (continuous mode)
809 * Argument          : none
810 * Return Value      : none
```

```
811 * Calling Functions :
812 *""FUNC COMMENT END""***************************************************/
813 void trans_scif_to_ddr_continuous( void )
814 {
815   int i;
816   char *tmp;
817
818   tmp = D_DMAC_SDRAM_VLADR;
819   tmp = (char *)(tmp + 0x20000000);    /* Set the start address of a point echo back */
820
821   SCIF0.SCSCR.WORD = 0x0000;           /* TIE, RIE, TE, RE Clear */
822   HPBDMAC0.DCMDR.BIT.DMEN = 1;         /* Activates DMA transfer */
823
824   /* SCIF init */
825   scif_init();
826
827   i = 2;                          /* set the continuous counter */
828
829   while (i > 0) {
830      /* Echo back for continuous transfer */
831      trans_scif_to_ddr_continuous_echoback(tmp);
832      tmp += D_DMAC_TRANS_SIZE_HALF_HPB;
833      i--;
834   }
835 }
836
837 /*""FUNC COMMENT""*****************************************************
838 * ID                :
839 * Outline           : HPB-DMAC starting transfer
840 * Declaration       : void trans_scif_to_ddr_continuous_echoback(char *ddr_adr)
841 * Description       : SCIF to DDR3SDRAM transfer (continuous mode)
842 * Argument          : char *ddr_adr :Address to be echo back
843 * Return Value      : none
844 * Calling Functions :
845 *""FUNC COMMENT END""***************************************************/
846 void trans_scif_to_ddr_continuous_echoback(char *ddr_adr)
847 {
848   rem_cnt = D_DMAC_TRANS_SIZE_HALF_HPB;           /* Initialization of remain characters counter */
849
850   while(rem_cnt > 0) {
851      /* Characters are entered from SCIF? */
852      if (rem_cnt > HPBDMAC0.DTCSR.BIT.DTCS) {
853         if (hpbdmac.cache == SELECT_CACHE_ON)
854            /* Cache Invalidation */
855            cache_purge((void *)D_DMAC_SDRAM_VLADR, D_DMAC_TRANS_SIZE_HPB);
856
857         /*  Echo back the one character */
858         scif_transmit_data_byte(ddr_adr);
859         rem_cnt -= 1;
860         ddr_adr++;
861      }
862   }
863 }
864
865 /*""FUNC COMMENT""*****************************************************
866 * ID                :
867 * Outline           : Interrupt handling HPB-DMAC
868 * Declaration       : void hpbdmac_interrupt(void)
```

```
869 * Description       : Interrupt handling HPB-DMAC
870 * Argument          : none
871 * Return Value      : none
872 * Calling Functions :
873 *""FUNC COMMENT END""**************************************************/
874 void hpbdmac_interrupt(void)
875 {
876   if (hpbdmac.automatic == TRANSFER_NON_AUTOMATIC)
877      HPBDMAC0.DCMDR.BIT.DNXT = 1;                  /* Requests the next DMA transfer */
878
879   if ( hpbdmac.mode == TRANSFER_MODE_CONTINUOUS &&
880        hpbdmac.dir == SCIF_TO_DDR) {
881      hpbdmac_interrupt_continuous();
882   } else {
883      HPBDMAC0.DCMDR.BIT.DQEND = 1;           /* Terminates continuous DMA transfer mode */
884   }
885
886   HPBDMAC.DINTCR.BIT.DTEC0 = 1;              /* DMA Transfer End Interrupt Status clear */
887 }
888
889 /*""FUNC COMMENT""*****************************************************
890 * ID                :
891 * Outline           : Interrupt handling HPB-DMAC
892 * Declaration       : void hpbdmac_interrupt_continuous(void)
893 * Description       : Interrupt handling HPB-DMAC (continuous)
894 * Argument          : none
895 * Return Value      : none
896 * Calling Functions :
897 *""FUNC COMMENT END""**************************************************/
898 void hpbdmac_interrupt_continuous( void )
899 {
900   char *tmp;
901
902   tmp = D_DMAC_SDRAM_VLADR;
903   tmp = (char *)(tmp + 0x20000000); /* Set the start address of a point echo back */
904
905   if (hpbdmac.cache == SELECT_CACHE_ON)
906      /* Cache Invalidation */
907      cache_purge((void *)D_DMAC_SDRAM_VLADR, D_DMAC_TRANS_SIZE_HPB);
908
909   /* transfer completed for information area 0? */
910   if (HPBDMAC0.DSTSR.BIT.NDP0 == 1) {
911      HPBDMAC0.DCMDR.BIT.DQEND = 1;           /* Terminates continuous DMA transfer mode */
912      tmp += 3;                               /* set of Echo back position */
913   } else {
914      tmp += 7;                               /* set of Echo back position */
915   }
916
917   /*  Echo back the one character */
918   scif_transmit_data_byte(tmp);
919   rem_cnt -= 1;
920 }
```

7. Control of Coherence Between Cache and External Memory

The SH7786 cache supports two write techniques: copy back and write through. This sample program enables both the operand cache and the secondary cache, and uses copy-back mode.

In copy-back mode, external memory is not written when a cache hit occurs. Therefore, there are cases when the operand cache and external memory do not match. When an external device (such as a DMAC) accesses a cached area, if the software does not perform either the write back processing or a cache invalidate (flush or purge) operation, coherence between the operand cache and external memory will not be guaranteed. Thus care is required. This sample program makes it possible to verify that the DMA transfer result differs depending on whether or not a flush or purge operation is performed.

Table 7 Coherence Control

Cache control	Function
Flush	If the operand cache is enabled, this operation searches the operand cache and if there is an entry for which a cache hit occurred, it writes that entry to external memory. The entry for which there was a hit is not invalidated. No operation is performed if a cache miss occurred or if the entry was not dirty.
Purge	If the operand cache is enabled, this operation searches the operand cache and invalidates entries for which there was a hit. If the line that was invalidated was dirty, that line is written to external memory. No operation is performed if a cache miss occurred.

Note that since write-through mode always performs a write back to external memory, coherence is guaranteed.

We recommend referring to section 8.5, Cache Manipulation Instructions, in the SH7786 Hardware Manual, which also describes these control methods.

8. Reference Documents

- Software Manual
  SH4-A Software Manual (REJ09B0090)
  (The latest version can be downloaded from the Renesas Electronics Web site.)

- Hardware Manual
  SH7786 Group Hardware Manual (REJ09B0533)
  (The latest version can be downloaded from the Renesas Electronics Web site.)

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jan.20.12	—	First edition issued

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 manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
   Handle unused pins in accord 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 one with a different type number, confirm that the change will not lead to problems.
   — The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. 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.
5. When exporting the 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. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. 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.
6. 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.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. 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 categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. 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 an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
   - "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.
   - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
   - "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. 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 them 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 system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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

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.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

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

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, 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 Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.1