To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

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

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

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.

Application Note

32

SH7046 Series On-Chip Peripheral Functions — DTC Volume

Application Note

Renesas 32-Bit RISC Microcomputer

SuperH^{TM} RISC engine Family/ SH7046 Series

Renesas Electronics
www.renesas.com

Rev.1.00 2003.11

Renesas 32-Bit RISC Microcomputer
SuperH™ RISC engine Family/SH7046 Series

SH7046 Series On-Chip Peripheral Functions — DTC Volume —

Application Note

RENESAS

REJ05B0275-0100O

Cautions

Keep safety first in your circuit designs!

1. Renesas Technology Corp. puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage. Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corp. product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corp. or a third party.
2. Renesas Technology Corp. assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors. Renesas Technology Corp. assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corp. by various means, including the Renesas Technology Corp. Semiconductor home page (http://www.renesas.com).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corp. semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corp. is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
   Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corp. for further details on these materials or the products contained therein.

Preface

The SH7046F, SH7148, SH7047F, and SH7049 are high-performance microcomputers with a 32-bit SH-2 CPU core that uses a RISC (Reduced Instruction Set Computer) type instruction set, and comprehensive on-chip peripheral functions.

On-chip peripherals include a CPU, ROM, RAM, a 16-bit multifunction timer pulse unit (MTU), serial communication interface (SCI), port output enable (POE), data transfer controller (DTC), and motor management timer (MMT), enabling these microcomputers to be used for a wide range of applications covering small to large-scale systems.

This Application Note includes sample tasks that use the SH7046 Series' on-chip peripheral functions, which we hope users will find useful as reference material in carrying out software design.

Although the operation of the task programs in this Application Note has been checked, operation should be confirmed again before any of these programs are actually used.

Contents

Section 1 Using the SH7046 Series Application Note

1.1 Organization of Application Note

This Application Note consists of two parts, as shown in figure 1.1.

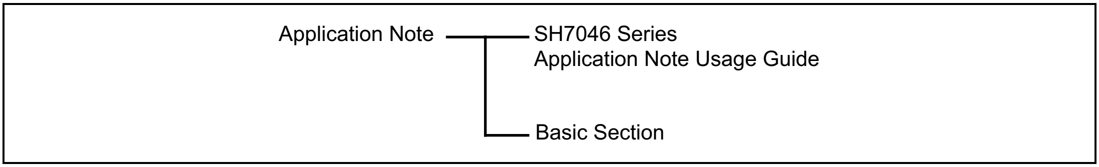


Figure 1.1 Organization of Application Note

(1) SH7046 Series Application Note Usage Guide
Explains how to use the SH7046 Series Application Note.

(2) On-Chip Peripheral Functions — DTC Volume
Mainly illustrates the use of the DTC among the SH7046 Series' on-chip peripheral functions, based on sample tasks.

1.2 Organization

The layout shown in figure 1.2 is employed to describe the use of on-chip peripheral functions.

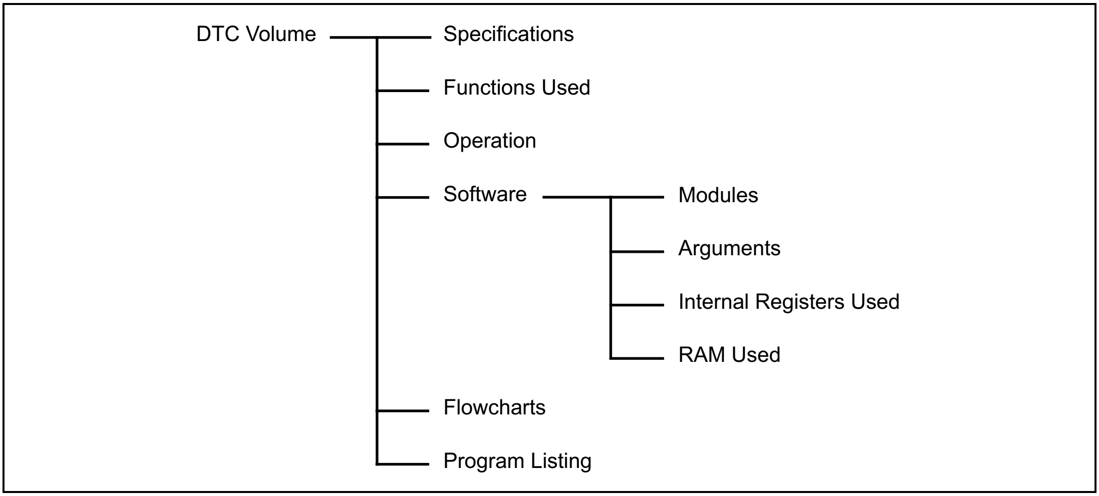


Figure 1.2 Organization

(1) Specifications

Describes the system specifications for the sample task.

(2) Functions Used

Describes the features of the peripheral function(s) used in the sample task, and peripheral function assignment.

(3) Operation

Describes the operation of the sample task, using a timing chart.

(4) Software

(a) Modules

Describes the software modules used in the operation of the sample task.

(b) Arguments

Describes the input arguments needed to execute the modules, and the output arguments after execution.

(c) Internal Registers Used

Describes the peripheral function internal registers (timer control registers, serial mode registers, etc.) set by the modules.

(d) RAM Used

Describes the labels and functions of RAM used by the modules.

(5) Flowcharts

Describes the software that executes the sample task, using flowcharts.

(6) Program Listing

Shows a program listing of the software that executes the sample task.

Section 2 On-Chip Peripheral Functions — DTC Volume

2.1 Data Transfer Using DTC Normal Mode (CMT, DTC)

Data Transfer Using DTC Normal Mode (CMT, DTC)	**Functions Used: CMT, DTC**

Specifications

(1) The data transfer controller (DTC) is activated by a compare match timer (CMT) compare match interrupt, and performs data transfer from on-chip RAM to on-chip RAM, as shown in figure 2.1.

(2) Normal mode is used for DTC data transfer, with 3-byte transfer performed as shown in figure 2.2.

(3) The DTC transfer conditions are shown in table 2.1.

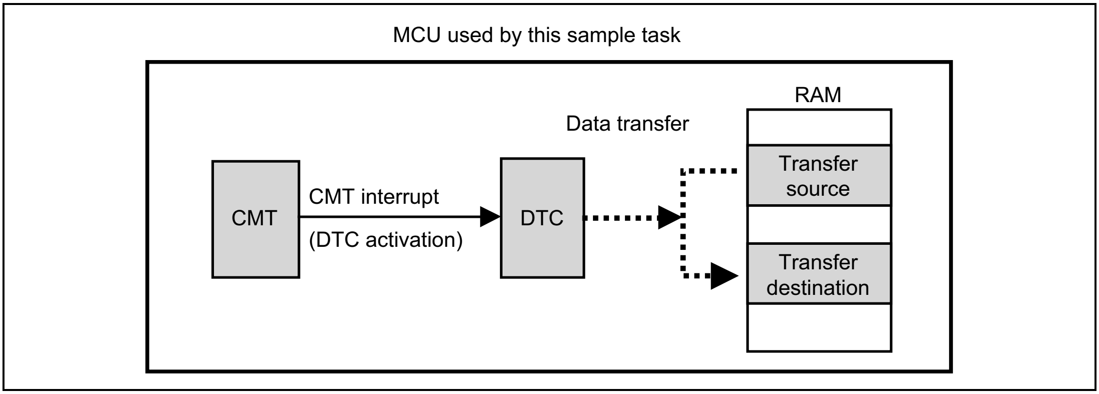


Figure 2.1 Data Transfer Using DTC

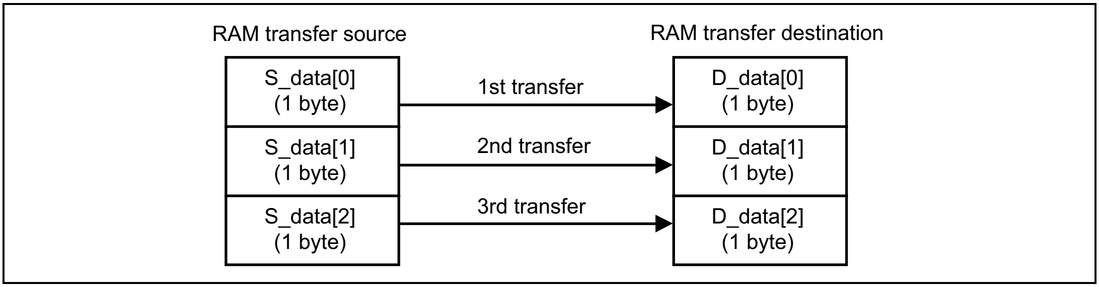


Figure 2.2 Data Transfer Using DTC Normal Mode

Table 2.1 DTC Transfer Conditions

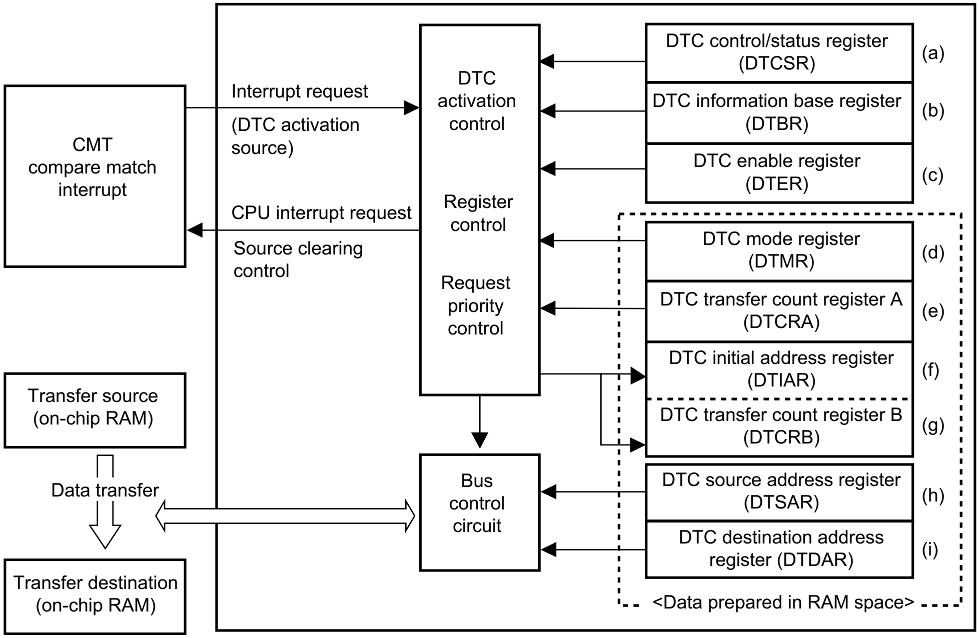
Condition	Description
Transfer mode	Normal mode
Number of transfers	3
Transfer data size	Byte transfer
Transfer source	On-chip RAM
Transfer destination	On-chip RAM
Transfer source address	Transfer source address incremented after transfer
Transfer destination address	Transfer destination address incremented after transfer
Activation source	Activated by CMT ch0 compare match interrupt (CMI0)
Interrupt handling	Interrupt to CPU enabled only at end of specified data transfer

Functions Used

(1) A block diagram of the DTC is shown below. Of the three DTC transfer modes — normal mode, repeat mode, and block transfer mode —this sample task uses normal mode to perform data transfer. Data transfer is performed from on-chip RAM to on-chip RAM, using a CMT compare match interrupt as the DTC activation source. The block diagram is explained below.

- The DTC mode register (DTMR) is a 16-bit register that controls the DTC's operating mode.
- The DTC source address register (DTSAR) is a 32-bit register that specifies the transfer source address of data to be transferred by the DTC. An even address should be specified in the case of word size transfer, and an address that is a multiple of 4 in the case of longword transfer.
- The DTC destination address register (DTDAR) is a 32-bit register that specifies the transfer destination address of data to be transferred by the DTC. An even address should be specified in the case of word size transfer, and an address that is a multiple of 4 in the case of longword transfer.
- The DTC initial address register (DTIAR) is a 32-bit register that specifies the transfer source/transfer destination initial address in repeat mode. In repeat mode, when the DTS bit is 1, specify the initial transfer source address in the repeat area, and when the DTS bit is 0, specify the initial transfer destination address in the repeat area.
- DTC transfer count register A (DTCRA) is a 16-bit register that specifies the number of transfers in DTC data transfer. In normal mode, DTCRA functions as a transfer counter (1 to 65,536). In repeat mode, upper 8-bit DTCRAH holds the number of transfers, and lower 8-bit DTCRAL functions as an 8-bit transfer counter. In block transfer mode, DTCRA functions as a 16-bit transfer counter.
- DTC transfer count register B (DTCRB) is a 16-bit register that specifies the block length in block transfer mode.
- The DTC enable register (DTER) is used to select the interrupt source that activates the DTC, and comprises registers DTEA through DTEF.
- The DTC control/status register (DTCSR) is a 16-bit register that sets enabling/disabling of DTC activation by software, and sets a software activation DTC vector address. DTCSR also shows the DTC transfer status.
- The DTC information base register (DTBR) is a readable/writable 16-bit register that specifies the upper 16 bits of the memory address that stores DTC transfer information. Word or longword access must be used for DTBR. If byte access is used, the register contents will be undefined in the case of a write, and an undefined value will be returned in the case of a read.
- Information of six registers — the DTC mode register (DTMR), DTC source address register (DTSAR), DTC destination address register (DTDAR), DTC initial address register (DTIAR), DTC transfer count register A (DTCRA), and DTC transfer count register B (DTCRB) — cannot be accessed directly from the CPU. When a DTC activation

source occurs, the relevant register information is transferred to these registers from information of an arbitrary set of registers located in on-chip RAM and DTC transfer is performed, and when transfer ends, the contents of these registers are returned to RAM. Therefore, register information should be prepared in arbitrary on-chip RAM in the user program.



Notes:
(a) Performs enabling/disabling of DTC activation by software, and software activation DTC vector address setting.
(b) Performs specification of the upper 16 bits of the memory address that stores DTC transfer information.
(c) Selects the interrupt source that activates the DTC; comprises six registers, DTEA through DTEF.
(d) Performs DTC operating mode setting.
(e) Specifies the number of transfers in DTC data transfer.
(f) In repeat mode, specifies the transfer source/transfer destination initial address in repeat mode. Not used in normal mode. In block transfer mode, functions as the DTCRB register.
(g) In block transfer mode, specifies the block length. Not used in normal mode. In repeat mode, functions as the DTIAR register.
(h) Specifies the transfer source address of data to be transferred by the DTC.
(i) Specifies the transfer destination address of data to be transferred by the DTC.

Figure 2.3 DTC Block Diagram

(2) A block diagram of CMT ch0 is shown is the figure below. In this task, DTC data transfer is performed using a CMT ch0 compare match interrupt as the activation source. The block diagram is explained below.

- The compare match timer start register (CMSTR) is a 16-bit register that is used to set whether the channel 0 and 1 counters (CMCNT) are operated or stopped.
- Compare match timer control/status register 0 (CMCSR_0) is a 16-bit register that performs compare match generation indication, interrupt enabling/disabling, and selection of the clock used for counting up.
- Compare match timer counter 0 (CMCNT_0) is a 16-bit register used as an up-counter for generating an interrupt request.
- Compare match timer constant register 0 (CMCOR_0) is a 16-bit register used to set the CMCNT compare match period.

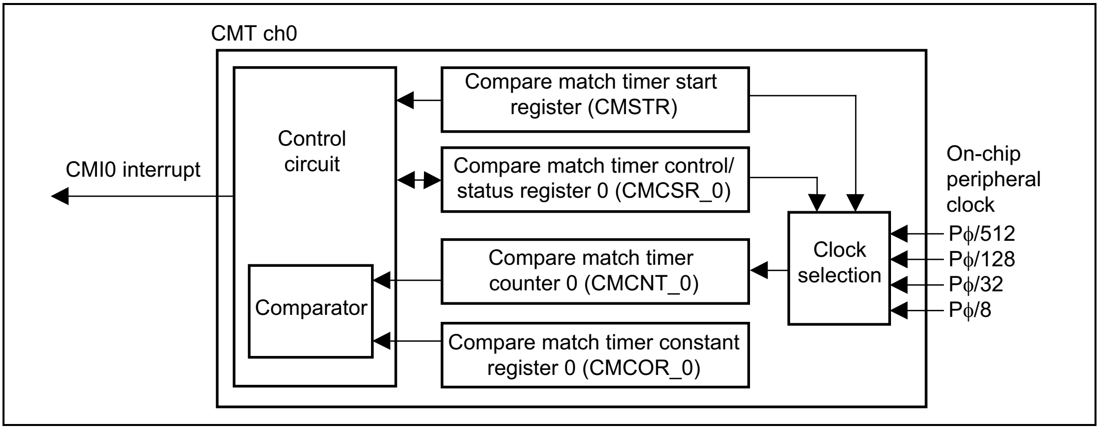

CMT ch0
CMI0 interrupt
Control circuit
Comparator
Compare match timer start register (CMSTR)
Compare match timer control/ status register 0 (CMCSR_0)
Compare match timer counter 0 (CMCNT_0)
Compare match timer constant register 0 (CMCOR_0)
Clock selection
On-chip peripheral clock
Pϕ/512
Pϕ/128
Pϕ/32
Pϕ/8

Figure 2.4 CMT Block Diagram

(3) Table 2.2 shows the function assignments used in this sample task.

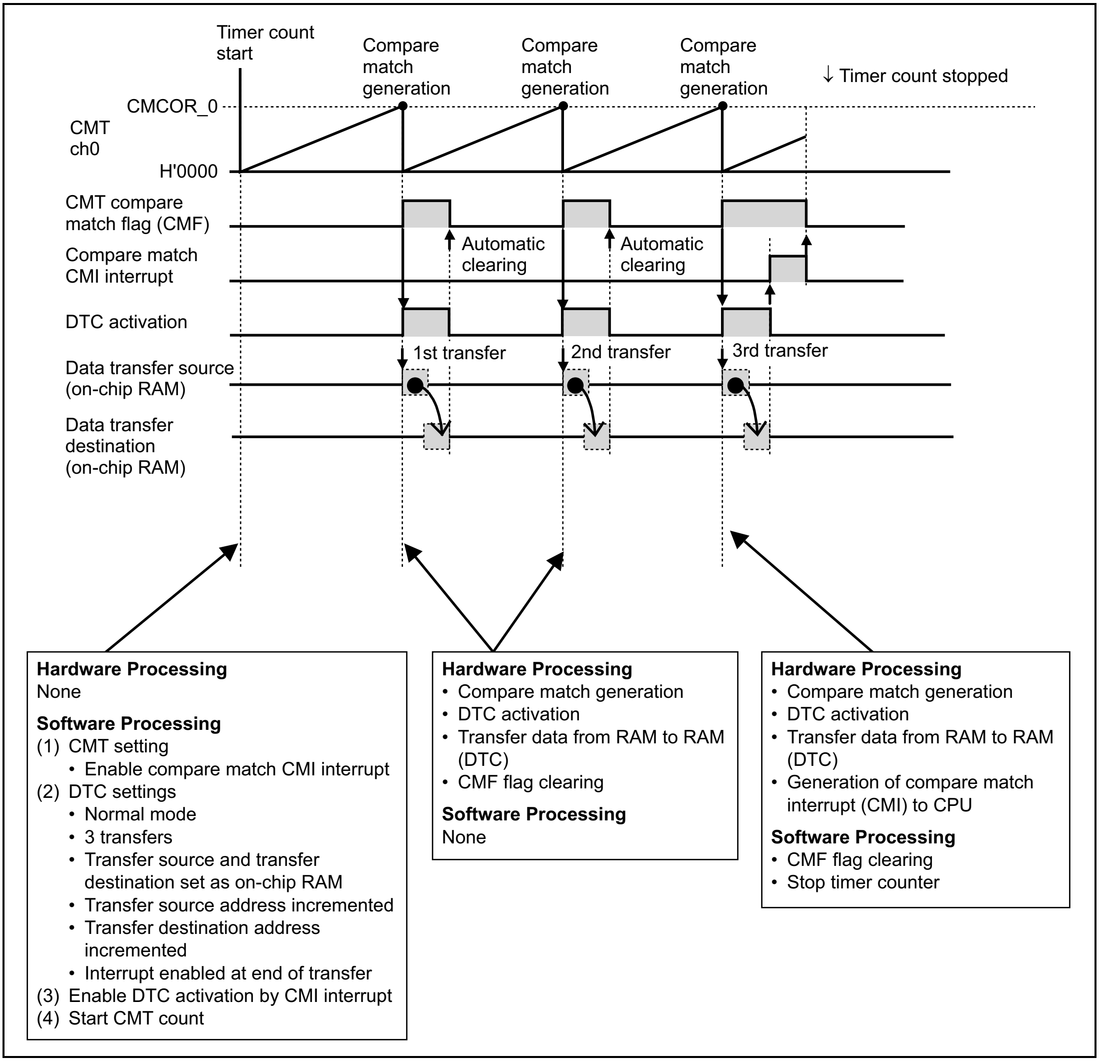
Table 2.2 Function Assignments

Function	Type	Function Assignment
DTMR	DTC	Sets DTC to normal mode
DTCRA	DTC	Setting of number of transfers
DTSAR	DTC	Transfer source address setting
DTDAR	DTC	Transfer destination address setting
DTBR	DTC	Setting of DTC vector upper 16 bits
DTER	DTC	Enables DTC activation by CMT ch0 CMI interrupt
CMSTR	CMT	CMT count start
CMCSR_0	CMT ch0	Count clock selection, interrupt control
CMCNT_0	CMT ch0	Counter
CMCOR_0	CMT ch0	Period setting

Operation

(1) The principles of operation of this sample task are shown in the figure below.

Data transfer from on-chip RAM to on-chip RAM is performed by the DTC by means of hardware and software processing as shown in the figure.



Hardware Processing
None

Software Processing
(1) CMT setting
- Enable compare match CMI interrupt

(2) DTC settings
- Normal mode
- 3 transfers
- Transfer source and transfer destination set as on-chip RAM
- Transfer source address incremented
- Transfer destination address incremented
- Interrupt enabled at end of transfer

(3) Enable DTC activation by CMI interrupt
(4) Start CMT count

Hardware Processing
- Compare match generation
- DTC activation
- Transfer data from RAM to RAM (DTC)
- CMF flag clearing

Software Processing
None

Hardware Processing
- Compare match generation
- DTC activation
- Transfer data from RAM to RAM (DTC)
- Generation of compare match interrupt (CMI) to CPU

Software Processing
- CMF flag clearing
- Stop timer counter

Figure 2.5 Principles of Operation

(2) The principles of operation of DTC activation are shown in the figure below. When executing DTC transfer, the following settings should be made before an activation source occurs.

- Make DTC register information settings and place DTC register information in RAM.
- Set the lower 16 bits of the start address (32 bits) of DTC register information in the DTC vector table.
- Set the lower 16 bits of the start address (32 bits) of DTC register information in the DTC information base register.

The DTC is activated by the following processing.

- A DTC activation source interrupt is generated.
- The lower 16 bits of the start address of DTC register information are read from the address corresponding to the DTC vector table activation source.
- The upper 16 bits of the start address of DTC register information are read from the DTC information base register (DTMR).
- The 32-bit start address of DTC register information is generated from the read start address lower 16 bits and upper 16 bits.
- The start of DTC register information is read sequentially from the DTC register information start address, and data transfer is performed.

In this task, a CMT compare match interrupt is used as the DTC activation source.

The following table shows the register information configuration in normal transfer mode.

Table 2.3 DTC Register Information (Normal Mode)

Setting Address	Register Name	Data Length
RF	DTC mode register (DTMR)	Word (2 bytes)
RF+2	DTC transfer count register A (DTCRA)	Word (2 bytes)
RF+8	DTC source address register (DTSAR)	Longword (4 bytes)
RF+12	DTC destination address register (DTDAR)	Longword (4 bytes)

RF: DTC register information start address (in on-chip RAM)

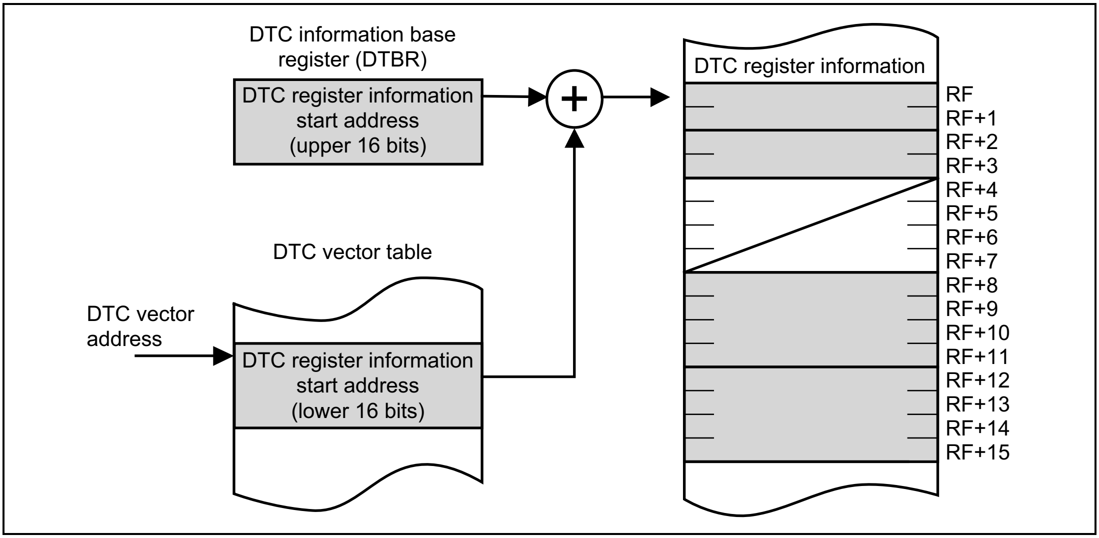

DTC information base register (DTBR)
DTC register information start address (upper 16 bits)
+
DTC register information
RF
RF+1
RF+2
RF+3
RF+4
RF+5
RF+6
RF+7
RF+8
RF+9
RF+10
RF+11
RF+12
RF+13
RF+14
RF+15
DTC vector table
DTC vector address
DTC register information start address (lower 16 bits)

Figure 2.6 Correspondence between DTC Vector Address and Transfer Information

Software

(1) Modules

The following table shows the modules used by this sample task.

Table 2.4 Modules

Module Name	Label	Function
Main routine	main	CMT timer setting, DTC initialization, timer start
CMI0 interrupt	cmt0_cmi0_dtc	CMT ch0 compare match interrupt (CMI0). Interrupt generation at end of specified number of DTC transfers

(2) Arguments

The following table shows the arguments used by this sample task.

Table 2.5 Arguments

Argument	Function	Module Name	Data Length	Input/ Output
S_data [0] to [2]	DTC transfer source transfer data storage	Main routine	1 byte	Output
D_data [0] to [2]	DTC transfer destination transfer data storage	Main routine	1 byte	Input

(3) Internal Registers Used

The following table shows the internal registers used by this sample task.

Table 2.6 Internal Registers Used

Register Name		Function	Address	Set Value
	Bits		Bits	
P_STBY.MSTCR1	MSTP25 MSTP24	Module standby control register 1 DTC module standby control bits: When MSTP25 = MSTP24 = 0, module standby release Same value is set for MSTP25 and MSTP24	H'FFFF861C Bit 9 Bit 8	B'00
P_STBY.MSTCR2	MSTP12	Module standby control register 2 CMT module standby control bit: When MSTP12 = 0, module standby release	H'FFFF861E Bit 12	0
P_INTC.IPRG	CMT0	Interrupt priority register G (IPRG) CMT0 CMI0 interrupt priority level setting: When CMT0 = b'1010 (10), CMI0 interrupt is set to priority level 10	H'FFFF8354 Bits 7 to 4	10
P_CMT.CMSTR		Compare match timer start register (CMTSR) 16-bit register that selects CMCNT operation/stoppage	H'FFFF83D0	H'0001
	STR1	Counter start 1: When STR1 = b'0, TCNT_1 count operation is stopped	Bit 1	
	STR0	Counter start 0: When STR0 = b'1, TCNT_0 counts	Bit 0	

Register Name		Function	Address	Set Value
	Bits		Bits	
P_CMT.CMCSR_0		Compare match timer control/status register 0 (CMCSR_0) Compare match generation indication, interrupt setting, timer clock setting	H'FFFF83D2	H'0043
	CMF	Compare match flag: CMF is set to 1 when CMCNT and CMCOR values match	Bit 7	
	CMIE	Compare match interrupt enable: When CMIE = 1, compare match interrupt (CMI) is enabled	Bit 6	
	CKS1 CKS0	CMCNT counter clock selection: When CKS[1:0] = b'11, count is performed using internal clock Pϕ/512	Bit 1 Bit 0	
P_CMT.CMCNT_0		Compare match timer counter 0 (CMCNT_0) 16-bit register used as up-counter for generating interrupt requests	H'FFFF83D4	H'0000
P_CMT.CMCOR_0		Compare match timer constant register 0 (CMCOR_0) 16-bit register used to set CMCNT compare match period When CMCOR_0 = H'1e84, 100 ms compare match period is used (Pϕ/512 count, Pϕ = 40 MHz)	H'FFFF83D6	H'1e84

Register Name		Function	Address	Set Value
	Bits		Bits	
DTC_N.DTMR		DTC mode register (DTMR) DTC operating mode control setting	Located in on-chip RAM	H'a000
	SM1 SM0	Source address mode: When SM[1:0] = b'10, DTSAR is incremented after transfer	Bit 15 Bit 14	
	DM1 DM0	Destination address mode: When DM[1:0] = b'10, DTDAR is incremented after transfer	Bit 13 Bit 12	
	MD1 MD0	DTC transfer mode: When MD[1:0] = b'00, normal mode	Bit 11 Bit 10	
	SZ1 SZ0	DTC data transfer size: When SZ[1:0] = b'00, byte (1-byte) transfer	Bit 9 Bit 8	
	DTS	DTC transfer mode select: When DTS = b'0, destination side is block area	Bit 7	
	CHNE	DTC chain transfer enable: When CHNE = b'0, chain transfer is cleared	Bit 6	
	DISEL	DTC interrupt select: When DISEL = b'0, interrupt request to CPU is generated only at end of specified data transfer	Bit 5	
	NMIM	DTC NMI mode: When NMIM = b'0, DTC transfer is suspended by NMI	Bit 4	
DTC_N.DTCRA		DTC transfer count register A (DTCRA) Specifies number of transfers in DTC data transfer Set to 3 transfers	Located in on-chip RAM	H'0003
DTC_N.DTSAR		DTC source address register (DTSAR) 32-bit register that specifies transfer source address of data to be transferred by DTC	Located in on-chip RAM	&S_data[0];
DTC_N.DTDAR		DTC destination address register (DTDAR) 32-bit register that specifies transfer destination address of data to be transferred by DTC	Located in on-chip RAM	&D_data[0];

Register Name		Function	Address	Set Value
	Bits		Bits	
P_DTC.DTBR		DTC information base register (DTBR) Specifies upper 16 bits of memory address that stores DTC transfer information	H'FFFF8708	0xFFFF
P_DTC.DTED	CMI0	DTC enable register D (DTED) When set to 1, corresponding interrupt source is selected as DTC activation source: When CMI0 (DTED5) = b'1, CMT0 CMI0 interrupt is activation source	H'FFFF8703 Bit 5	1

(4) RAM Used

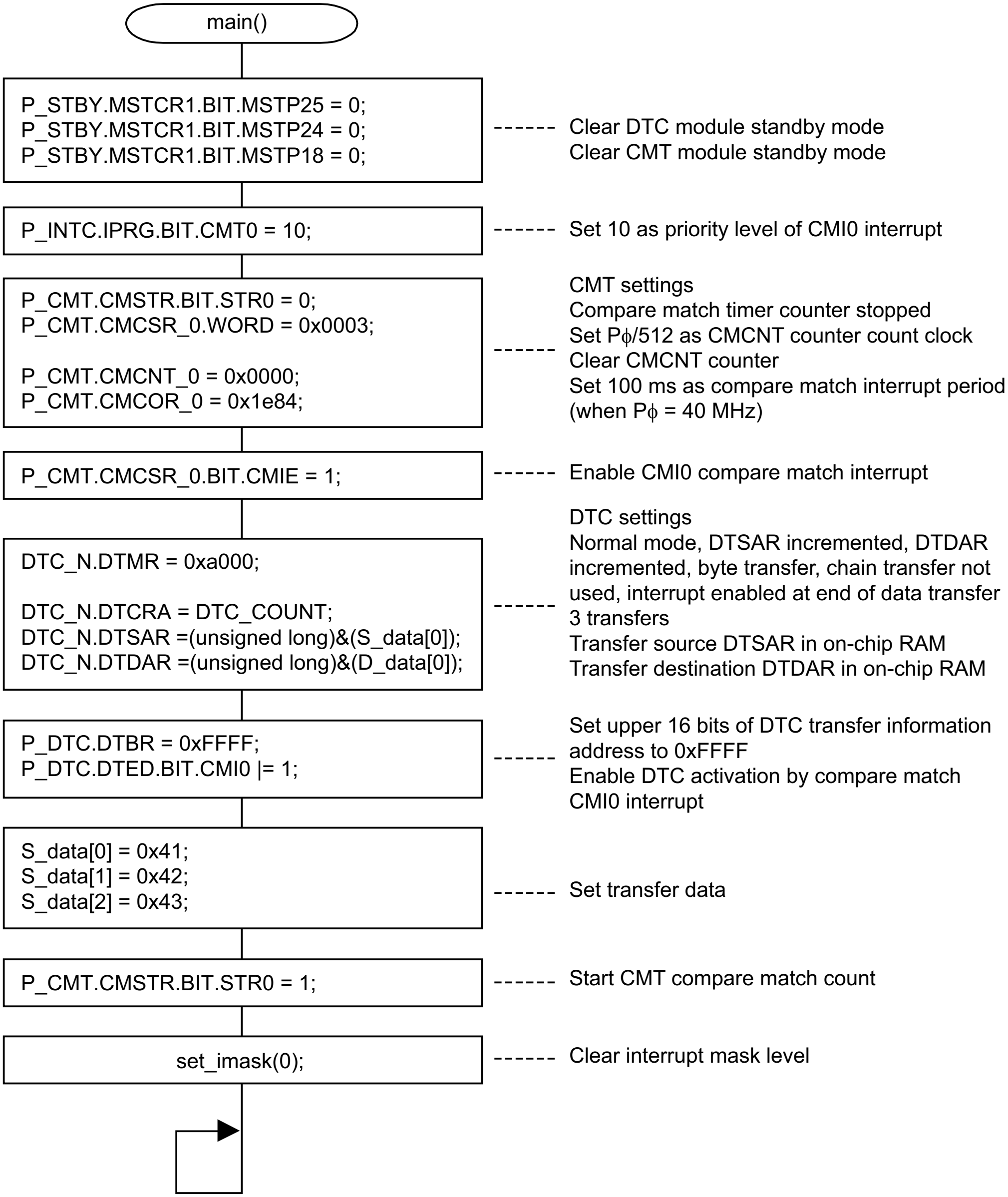
The following table shows the RAM used by this sample task.

Table 2.7 RAM Used

Label	Function	Address	Module Using RAM
S_data	DTC transfer data storage Array storing 3-byte data	On-chip RAM	Main routine
D_data	Data storage after DTC data transfer Array storing 3-byte data	On-chip RAM	Main routine

Flowcharts

(a) Main processing



```
P_STBY.MSTCR1.BIT.MSTP25 = 0;
P_STBY.MSTCR1.BIT.MSTP24 = 0;
P_STBY.MSTCR1.BIT.MSTP18 = 0;
```

Clear DTC module standby mode
Clear CMT module standby mode

```
P_INTC.IPRG.BIT.CMT0 = 10;
```

Set 10 as priority level of CMI0 interrupt

```
P_CMT.CMSTR.BIT.STR0 = 0;
P_CMT.CMCSR_0.WORD = 0x0003;

P_CMT.CMCNT_0 = 0x0000;
P_CMT.CMCOR_0 = 0x1e84;
```

CMT settings
Compare match timer counter stopped
Set Pϕ/512 as CMCNT counter count clock
Clear CMCNT counter
Set 100 ms as compare match interrupt period (when Pϕ = 40 MHz)

```
P_CMT.CMCSR_0.BIT.CMIE = 1;
```

Enable CMI0 compare match interrupt

```
DTC_N.DTMR = 0xa000;

DTC_N.DTCRA = DTC_COUNT;
DTC_N.DTSAR =(unsigned long)&(S_data[0]);
DTC_N.DTDAR =(unsigned long)&(D_data[0]);
```

DTC settings
Normal mode, DTSAR incremented, DTDAR incremented, byte transfer, chain transfer not used, interrupt enabled at end of data transfer
3 transfers
Transfer source DTSAR in on-chip RAM
Transfer destination DTDAR in on-chip RAM

```
P_DTC.DTBR = 0xFFFF;
P_DTC.DTED.BIT.CMI0 |= 1;
```

Set upper 16 bits of DTC transfer information address to 0xFFFF
Enable DTC activation by compare match CMI0 interrupt

```
S_data[0] = 0x41;
S_data[1] = 0x42;
S_data[2] = 0x43;
```

Set transfer data

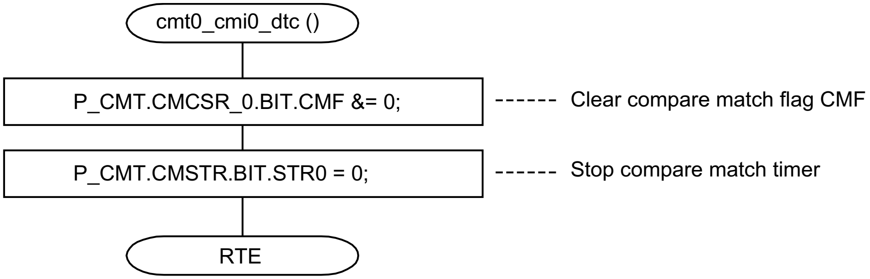
```
P_CMT.CMSTR.BIT.STR0 = 1;
```

Start CMT compare match count

```
set_imask(0);
```

Clear interrupt mask level

(b) Compare match interrupt handling


cmt0_cmi0_dtc ()
P_CMT.CMCSR_0.BIT.CMF &= 0;
------ Clear compare match flag CMF
P_CMT.CMSTR.BIT.STR0 = 0;
------ Stop compare match timer
RTE

Program Listing

```
/*****************************************************/
/* SH7046F Series -SH7047- Application Note            */
/* Data transfer Controller(DTC)                       */
/*     Normal  mode                                    */
/* Function                                            */
/* :Data transfer Controller(DTC)                      */
/* :Compare Match Timer(CMT ch0)                       */
/*                                                     */
/* External input clock      :10MHz                    */
/* Internal CPU clock        :40MHz                    */
/* Internal peripheral clock :40MHz                    */
/*                                                     */
/* Written   :   2002/3/1  Rev.1.0                     */
/*****************************************************/


#include "iodefine_7047v13.1.h"
#include <machine.h>



/*------------ Symbol Definition --------------------------------------------*/
struct st_dtc_normal{         /* DTC Normal  Mode information                 */
    unsigned short DTMR;      /* DTC Mode Register                            */
    unsigned short DTCRA;     /* Transfer counter                             */
    unsigned short dummy1;    /* Reserved                                     */
    unsigned short dummy2;    /* Reserved                                     */
    unsigned long DTSAR;      /* source address register                      */
    unsigned long DTDAR;      /* destination address register                 */
};

#define DTC_COUNT   3                                  /* DTC Transmit count        */
#define DTC_N (*(volatile struct st_dtc_normal*)0xFFFFE000)
                                                       /* DTC information address   */


/*------------ Function Definition ------------------------------------------*/
void main(void);
void cmt0_cmi0_dtc(void);


/*------------ RAM allocation Definition ------------------------------------*/
unsigned char S_data[DTC_COUNT];     /* source buffer memory                        */
unsigned char D_data[DTC_COUNT];     /* destination buffer memory                   */



/*********************************************************/
/* main Program                                           */
/*********************************************************/
void main( void )
{
    /* Set standby mode   */
    P_STBY.MSTCR1.BIT.MSTP25 = 0;             /* Disable DTC standby mode       */
    P_STBY.MSTCR1.BIT.MSTP24 = 0;             /* Disable DTC standby mode       */
    P_STBY.MSTCR2.BIT.MSTP12 = 0;             /* Disable CMT standby mode       */

    /* Set interrupt priority level (0 to 15) */
    P_INTC.IPRG.BIT.CMT0 = 10;                /* CMT0 CMI0 interrupt level 10   */




    /* Initialize CMT0 for Interval timer  */
    P_CMT.CMSTR.BIT.STR0 = 0;     /* timer count stop                           */
    P_CMT.CMCSR_0.WORD = 0x0003;
    /* CMF=0;                     clear compare match flag                      */
    /* CMIE=0;                    compare match interrupt disable               */
    /* CKS[1:0]=b'11;             clock = peripheral clock(Pϕ)/512              */
    P_CMT.CMCNT_0 = 0x0000;       /* timer counter clear                        */
    P_CMT.CMCOR_0 = 0x1e84;       /* 100ms@Pϕ=40MHz                             */
    P_CMT.CMCSR_0.BIT.CMIE = 1;   /* compare match interrupt enable             */



    /* DTC information  */
    DTC_N.DTMR = 0xa000;                  /*                                        */
                    /* SM[1:0]=b'10;  DTSAR is incremented                      */
                    /* DM[1:0]=b'10;  DTDAR is incremented                      */
                    /* MD[1:0]=b'00;  Normal transfer mode                      */
                    /* SZ[1:0]=b'00;  byte-size transfer                        */
                    /* DTS=0;         destination is block area (not used)      */
                    /* CHNE=0;        Chain transfer is disable                 */
                    /* DISEL=0;       Interrupt->transfer ends                  */
                    /* NMIM=0;        NMI->Terminate DTC transfer               */
    DTC_N.DTCRA = DTC_COUNT;          /* DTC transfer Count                     */
    DTC_N.DTSAR =(unsigned long)&(S_data[0]); /* set source address             */
    DTC_N.DTDAR =(unsigned long)&(D_data[0]); /* set destination address        */

    P_DTC.DTBR = 0xFFFF;              /* DTC information base register          */
    /* DTC transmit enable  */
    P_DTC.DTED.BIT.CMI0 |= 1;         /* interrupt sources CMT ch0(CMI0)        */


    /* set transmit data   */
    S_data[0] = 0x41;
    S_data[1] = 0x42;
    S_data[2] = 0x43;

    P_CMT.CMSTR.BIT.STR0 = 1;         /* CMT0 timer count start                 */

    set_imask(0);                     /* clear interrupt mask level             */
    while(1);
}
```

```
/*************************************************************/
/* CMT0 Interrupt                                            */
/* Interval interrupt                                        */
/*************************************************************/
#pragma interrupt(cmt0_cmi0_dtc)
void cmt0_cmi0_dtc(void)
{

    P_CMT.CMCSR_0.BIT.CMF &= 0;       /* Clear CMT0 compare match flag           */

    P_CMT.CMSTR.BIT.STR0 = 0;         /* CMT0 timer count stop                   */

}
```

2.2 Data Transfer Using DTC Repeat Mode (CMT, DTC)

Data Transfer Using DTC Repeat Mode (CMT, DTC)	**Functions Used: CMT, DTC**

Specifications

(1) The data transfer controller (DTC) is activated by a compare match timer (CMT) compare match interrupt, and performs data transfer from on-chip RAM to on-chip RAM, as shown in figure 2.7.

(2) Repeat mode is used for DTC data transfer, with transfer source 3-byte data repeatedly transferred to a fixed area in on-chip RAM as shown in figure 2.8.

(3) The DTC transfer conditions are shown in table 2.8.

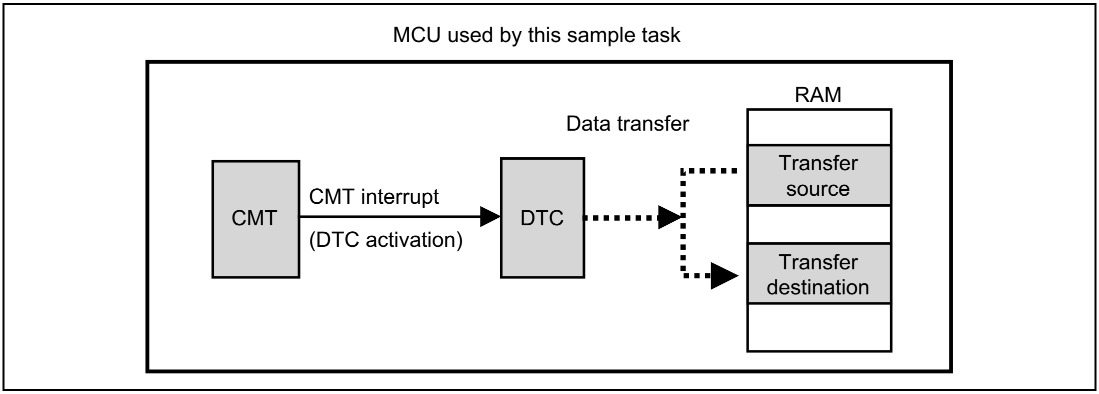


Figure 2.7 Data Transfer Using DTC

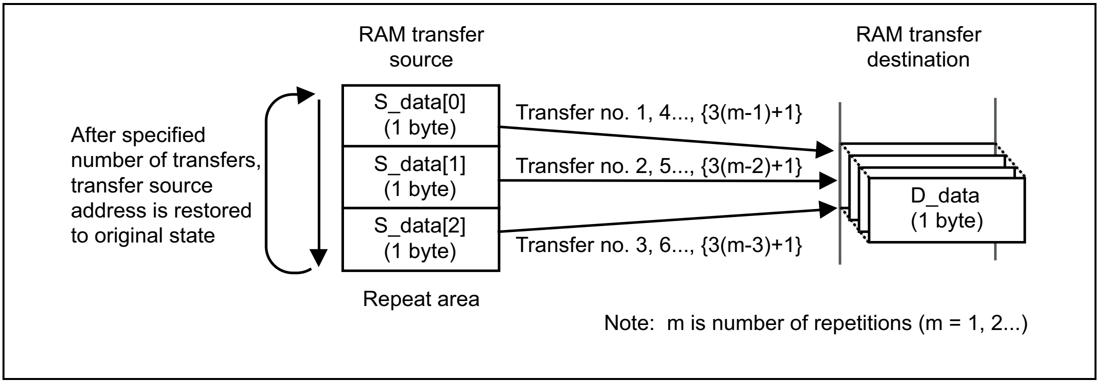

RAM transfer source
RAM transfer destination
After specified number of transfers, transfer source address is restored to original state
S_data[0] (1 byte)
S_data[1] (1 byte)
S_data[2] (1 byte)
Transfer no. 1, 4..., {3(m-1)+1}
Transfer no. 2, 5..., {3(m-2)+1}
Transfer no. 3, 6..., {3(m-3)+1}
D_data (1 byte)
Repeat area
Note: m is number of repetitions (m = 1, 2...)

Figure 2.8 Data Transfer Using DTC Repeat Mode

Table 2.8 DTC Transfer Conditions

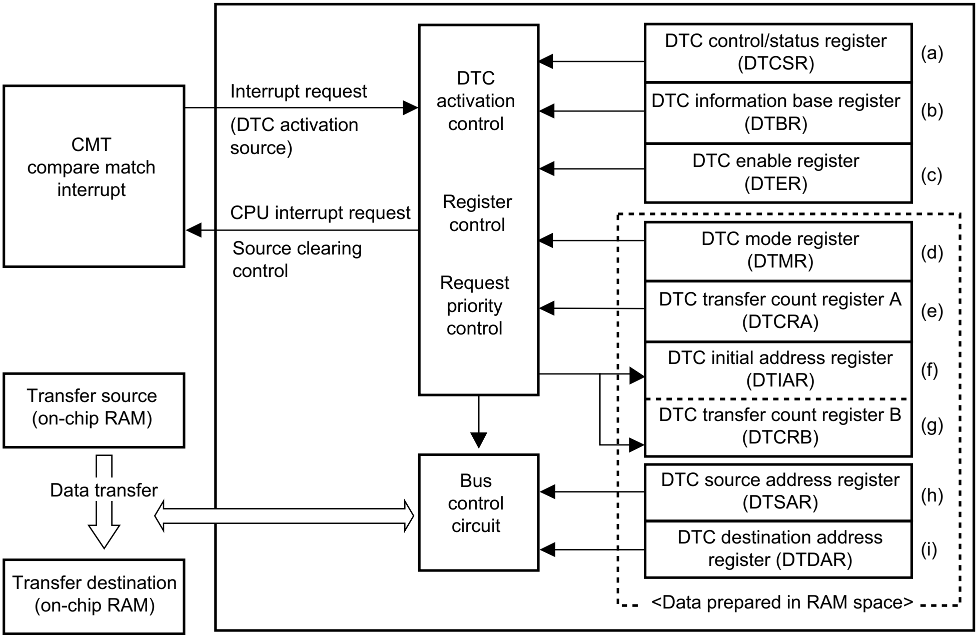
Condition	Description
Transfer mode	Repeat mode, source side (transfer source) is repeat area
Number of transfers	3
Transfer data size	Byte transfer
Transfer source	On-chip RAM (repeat area)
Transfer destination	On-chip RAM
Transfer source address	Transfer source address incremented after transfer
Transfer destination address	Transfer destination address fixed
Activation source	Activated by CMT ch0 compare match interrupt (CMI0)
Interrupt handling	Interrupt to CPU enabled every DTC transfer

Functions Used

(1) A block diagram of the DTC is shown below. Of the three DTC transfer modes — normal mode, repeat mode, and block transfer mode —this sample task uses normal mode to perform data transfer. Data transfer is performed from on-chip RAM to on-chip RAM, using a CMT compare match interrupt as the DTC activation source. The block diagram is explained below.

- The DTC mode register (DTMR) is a 16-bit register that controls the DTC's operating mode.
- The DTC source address register (DTSAR) is a 32-bit register that specifies the transfer source address of data to be transferred by the DTC. An even address should be specified in the case of word size transfer, and an address that is a multiple of 4 in the case of longword transfer.
- The DTC destination address register (DTDAR) is a 32-bit register that specifies the transfer destination address of data to be transferred by the DTC. An even address should be specified in the case of word size transfer, and an address that is a multiple of 4 in the case of longword transfer.
- The DTC initial address register (DTIAR) is a 32-bit register that specifies the transfer source/transfer destination initial address in repeat mode. In repeat mode, when the DTS bit is 1, specify the initial transfer source address in the repeat area, and when the DTS bit is 0, specify the initial transfer destination address in the repeat area.
- DTC transfer count register A (DTCRA) is a 16-bit register that specifies the number of transfers in DTC data transfer. In normal mode, DTCRA functions as a transfer counter (1 to 65,536). In repeat mode, upper 8-bit DTCRAH holds the number of transfers, and lower 8-bit DTCRAL functions as an 8-bit transfer counter. In block transfer mode, DTCRA functions as a 16-bit transfer counter.
- DTC transfer count register B (DTCRB) is a 16-bit register that specifies the block length in block transfer mode.
- The DTC enable register (DTER) is used to select the interrupt source that activates the DTC, and comprises registers DTEA through DTEF.
- The DTC control/status register (DTCSR) is a 16-bit register that sets enabling/disabling of DTC activation by software, and sets a software activation DTC vector address. DTCSR also shows the DTC transfer status.
- The DTC information base register (DTBR) is a readable/writable 16-bit register that specifies the upper 16 bits of the memory address that stores DTC transfer information. Word or longword access must be used for DTBR. If byte access is used, the register contents will be undefined in the case of a write, and an undefined value will be returned in the case of a read.
- Information of six registers — the DTC mode register (DTMR), DTC source address register (DTSAR), DTC destination address register (DTDAR), DTC initial address register (DTIAR), DTC transfer count register A (DTCRA), and DTC transfer count register B (DTCRB) — cannot be accessed directly from the CPU. When a DTC activation

source occurs, the relevant register information is transferred to these registers from information of an arbitrary set of registers located in on-chip RAM and DTC transfer is performed, and when transfer ends, the contents of these registers are returned to RAM. Therefore, register information should be prepared in arbitrary on-chip RAM in the user program.



Notes:

(a) Performs enabling/disabling of DTC activation by software, and software activation DTC vector address setting.

(b) Performs specification of the upper 16 bits of the memory address that stores DTC transfer information.

(c) Selects the interrupt source that activates the DTC; comprises six registers, DTEA through DTEF.

(d) Performs DTC operating mode setting.

(e) Specifies the number of transfers in DTC data transfer.

(f) In repeat mode, specifies the transfer source/transfer destination initial address in repeat mode. Not used in normal mode. In block transfer mode, functions as the DTCRB register.

(g) In block transfer mode, specifies the block length. Not used in normal mode. In repeat mode, functions as the DTIAR register.

(h) Specifies the transfer source address of data to be transferred by the DTC.

(i) Specifies the transfer destination address of data to be transferred by the DTC.

Figure 2.9 DTC Block Diagram

(2) A block diagram of CMT ch0 is shown is the figure below. In this task, DTC data transfer is performed using a CMT ch0 compare match interrupt as the activation source. The block diagram is explained below.

- The compare match timer start register (CMSTR) is a 16-bit register that is used to set whether the channel 0 and 1 counters (CMCNT) are operated or stopped.
- Compare match timer control/status register 0 (CMCSR_0) is a 16-bit register that performs compare match generation indication, interrupt enabling/disabling, and selection of the clock used for counting up.
- Compare match timer counter 0 (CMCNT_0) is a 16-bit register used as an up-counter for generating an interrupt request.
- Compare match timer constant register 0 (CMCOR_0) is a 16-bit register used to set the CMCNT compare match period.

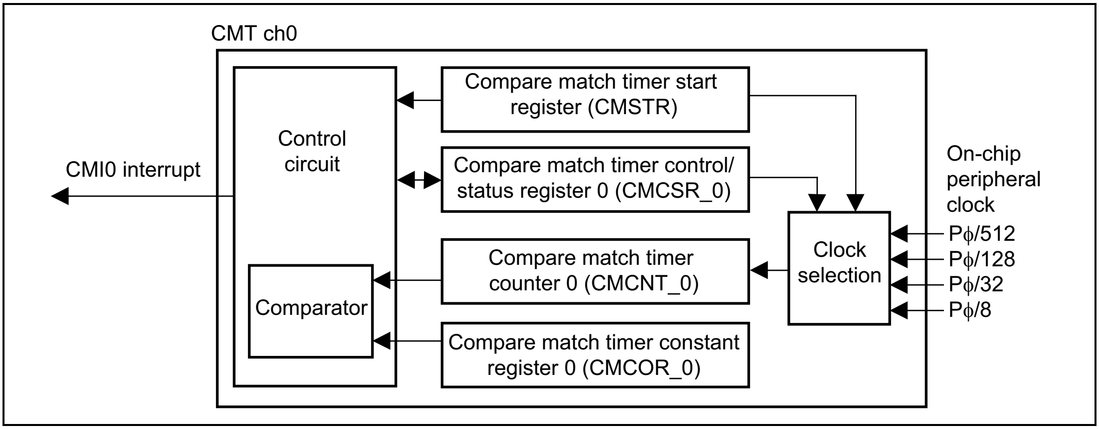

CMT ch0
CMI0 interrupt
Control circuit
Comparator
Compare match timer start register (CMSTR)
Compare match timer control/ status register 0 (CMCSR_0)
Compare match timer counter 0 (CMCNT_0)
Compare match timer constant register 0 (CMCOR_0)
Clock selection
On-chip peripheral clock
Pϕ/512
Pϕ/128
Pϕ/32
Pϕ/8

Figure 2.10 CMT Block Diagram

(4) Table 2.9 shows the function assignments used in this sample task.

Table 2.9 Function Assignments

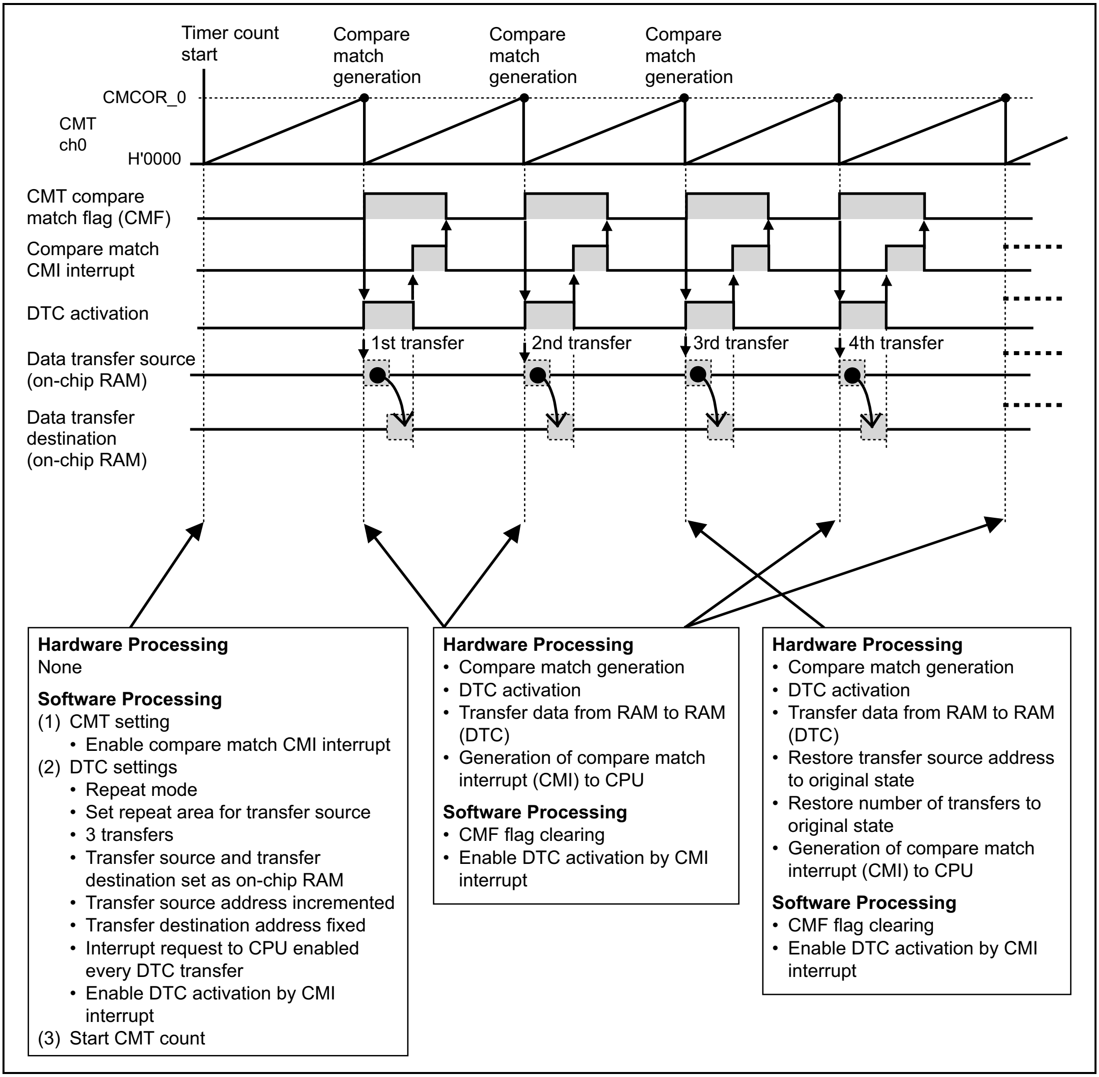
Function	Type	Function Assignment
DTMR	DTC	Sets DTC to repeat mode, with source side (transfer source) as repeat area
DTCRA	DTC	Setting of number of transfers
DTIAR	DTC	Sets transfer source (repeat area) initial address
DTSAR	DTC	Transfer source address setting
DTDAR	DTC	Transfer destination address setting
DTBR	DTC	Setting of DTC vector upper 16 bits
DTER	DTC	Enables DTC activation by CMT ch0 CMI interrupt
CMSTR	CMT	CMT count start
CMCSR_0	CMT ch0	Count clock selection, interrupt control
CMCNT_0	CMT ch0	Counter
CMCOR_0	CMT ch0	Period setting

Operation

(1) The principles of operation of this sample task are shown in the figure below.

Data transfer from on-chip RAM to on-chip RAM is performed by the DTC by means of hardware and software processing as shown in the figure.

This sample task uses repeat mode.



Hardware Processing
None

Software Processing
(1) CMT setting
- Enable compare match CMI interrupt

(2) DTC settings
- Repeat mode
- Set repeat area for transfer source
- 3 transfers
- Transfer source and transfer destination set as on-chip RAM
- Transfer source address incremented
- Transfer destination address fixed
- Interrupt request to CPU enabled every DTC transfer
- Enable DTC activation by CMI interrupt

(3) Start CMT count

Hardware Processing
- Compare match generation
- DTC activation
- Transfer data from RAM to RAM (DTC)
- Generation of compare match interrupt (CMI) to CPU

Software Processing
- CMF flag clearing
- Enable DTC activation by CMI interrupt

Hardware Processing
- Compare match generation
- DTC activation
- Transfer data from RAM to RAM (DTC)
- Restore transfer source address to original state
- Restore number of transfers to original state
- Generation of compare match interrupt (CMI) to CPU

Software Processing
- CMF flag clearing
- Enable DTC activation by CMI interrupt

Figure 2.11 Principles of Operation

(2) The principles of operation of DTC activation are shown in the figure below. When executing DTC transfer, the following settings should be made before an activation source occurs.

- Make DTC register information settings and place DTC register information in RAM.
- Set the lower 16 bits of the start address (32 bits) of DTC register information in the DTC vector table.
- Set the lower 16 bits of the start address (32 bits) of DTC register information in the DTC information base register.

The DTC is activated by the following processing.

- A DTC activation source interrupt is generated.
- The lower 16 bits of the start address of DTC register information are read from the address corresponding to the DTC vector table activation source.
- The upper 16 bits of the start address of DTC register information are read from the DTC information base register (DTMR).
- The 32-bit start address of DTC register information is generated from the read start address lower 16 bits and upper 16 bits.
- The start of DTC register information is read sequentially from the DTC register information start address, and data transfer is performed.

In this task, a CMT compare match interrupt is used as the DTC activation source.

The following table shows the register information configuration in repeat transfer mode.

Table 2.10 DTC Register Information (Repeat Mode)

Setting Address	Register Name	Data Length
RF	DTC mode register (DTMR)	Word (2 bytes)
RF+2	DTC transfer count register AH (DTCRAH)	Byte (1 byte)
RF+3	DTC transfer count register AL (DTCRAL)	Byte (1 byte)
RF+4	DTC initial address register (DTIAR)	Longword (4 bytes)
RF+8	DTC source address register (DTSAR)	Longword (4 bytes)
RF+12	DTC destination address register (DTDAR)	Longword (4 bytes)

RF: DTC register information start address (in on-chip RAM)

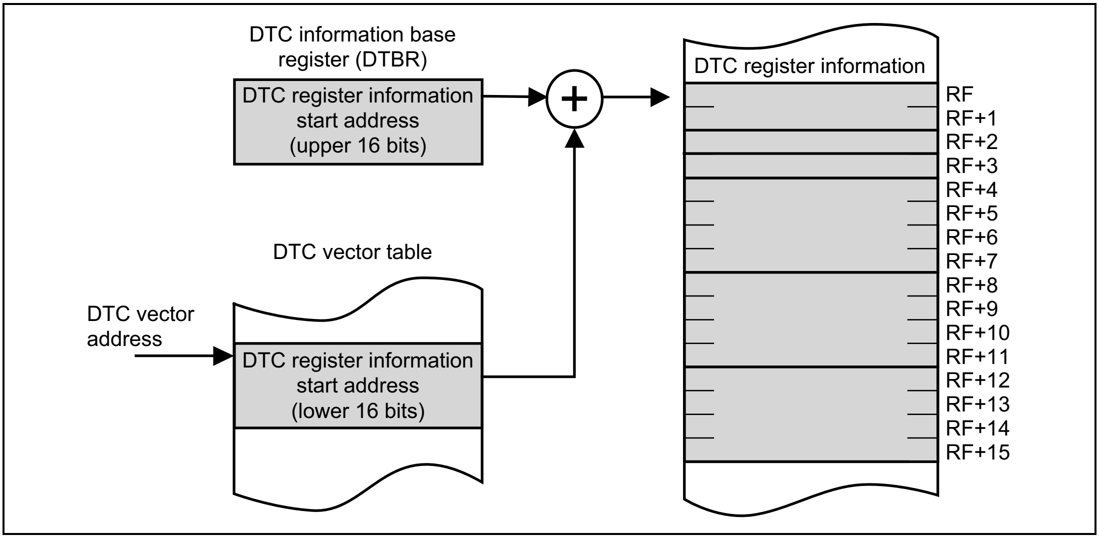

DTC information base register (DTBR)
DTC register information start address (upper 16 bits)
DTC vector table
DTC vector address
DTC register information start address (lower 16 bits)
DTC register information
RF
RF+1
RF+2
RF+3
RF+4
RF+5
RF+6
RF+7
RF+8
RF+9
RF+10
RF+11
RF+12
RF+13
RF+14
RF+15

Figure 2.12 Correspondence between DTC Vector Address and Transfer Information

Software

(1) Modules

The following table shows the modules used by this sample task.

Table 2.11 Modules

Module Name	Label	Function
Main routine	main	CMT timer setting, DTC initialization, timer start
CMI0 interrupt	cmt0_cmi0_dtc	CMT ch0 compare match interrupt (CMI0). Interrupt generated every DTC transfer

(2) Arguments

The following table shows the arguments used by this sample task.

Table 2.12 Arguments

Argument	Function	Module Name	Data Length	Input/ Output
S_data [0] to [2]	DTC transfer source transfer data storage	Main routine	1 byte	Output
D_data	DTC transfer destination transfer data storage	Main routine	1 byte	Input

(3) Internal Registers Used

The following table shows the internal registers used by this sample task.

Table 2.6 Internal Registers Used

Register Name		Function	Address	Set Value
	Bits		Bits	
P_STBY.MSTCR1	MSTP25 MSTP24	Module standby control register 1 DTC module standby control bits: When MSTP25 = MSTP24 = 0, module standby release Same value is set for MSTP25 and MSTP24	H'FFFF861C Bit 9 Bit 8	B'00
P_STBY.MSTCR2	MSTP12	Module standby control register 2 CMT module standby control bit: When MSTP12 = 0, module standby release	H'FFFF861E Bit 12	0
P_INTC.IPRG	CMT0	Interrupt priority register G (IPRG) CMT0 CMI0 interrupt priority level setting: When CMT0 = b'1010 (10), CMI0 interrupt is set to priority level 10	H'FFFF8354 Bits 7 to 4	10
P_CMT.CMSTR		Compare match timer start register (CMTSR) 16-bit register that selects CMCNT operation/stoppage	H'FFFF83D0	H'0001
	STR1	Counter start 1: When STR1 = b'0, TCNT_1 count operation is stopped	Bit 1	
	STR0	Counter start 0: When STR0 = b'1, TCNT_0 counts	Bit 0	

Register Name		Function	Address	Set Value
	Bits		Bits	
P_CMT.CMCSR_0		Compare match timer control/status register 0 (CMCSR_0) Compare match generation indication, interrupt setting, timer clock setting	H'FFFF83D2	H'0043
	CMF	Compare match flag: CMF is set to 1 when CMCNT and CMCOR values match	Bit 7	
	CMIE	Compare match interrupt enable: When CMIE = 1, compare match interrupt (CMI) is enabled	Bit 6	
	CKS1 CKS0	CMCNT counter clock selection: When CKS[1:0] = b'11, count is performed using internal clock Pφ/512	Bit 1 Bit 0	
P_CMT.CMCNT_0		Compare match timer counter 0 (CMCNT_0) 16-bit register used as up-counter for generating interrupt requests	H'FFFF83D4	H'0000
P_CMT.CMCOR_0		Compare match timer constant register 0 (CMCOR_0) 16-bit register used to set CMCNT compare match period When CMCOR_0 = H'1e84, 100 ms compare match period is used (Pφ/512 count, Pφ = 40 MHz)	H'FFFF83D6	H'1e84

Register Name		Function	Address	Set Value
	Bits		Bits	
DTC_R.DTMR		DTC mode register (DTMR) DTC operating mode control setting	Located in on-chip RAM	H'84a0
	SM1 SM0	Source address mode: When SM[1:0] = b'10, DTSAR is incremented after transfer	Bit 15 Bit 14	
	DM1 DM0	Destination address mode: When DM[1:0] = b'00, DTDAR is fixed	Bit 13 Bit 12	
	MD1 MD0	DTC transfer mode: When MD[1:0] = b'01, repeat mode	Bit 11 Bit 10	
	SZ1 SZ0	DTC data transfer size: When SZ[1:0] = b'00, byte (1-byte) transfer	Bit 9 Bit 8	
	DTS	DTC transfer mode select: When DTS = b'1, source side (DTSAR) is repeat area	Bit 7	
	CHNE	DTC chain transfer enable: When CHNE = b'0, chain transfer is cleared	Bit 6	
	DISEL	DTC interrupt select: When DISEL = b'1, interrupt request to CPU is generated every DTC transfer	Bit 5	
	NMIM	DTC NMI mode: When NMIM = b'0, DTC transfer is suspended by NMI	Bit 4	
DTC_R.DTCRAH		DTC transfer count register AH (DTCRAH) Specifies held value of number of transfers in DTC data transfer Set to 3 transfers	Located in on-chip RAM	H'03
DTC_R.DTCRAL		DTC transfer count register AL (DTCRAL) Specifies number of transfers in DTC data transfer Set to 3 transfers	Located in on-chip RAM	H'03
DTC_R.DTIAR		DTC initial address register (DTIAR) 32-bit register that specifies initial address of DTC transfer data transfer source	Located in on-chip RAM	&S_data[0]

Register Name		Function	Address	Set Value
	Bits		Bits	
DTC_R.DTSAR		DTC source address register (DTSAR) 32-bit register that specifies transfer source address of DTC transfer data	Located in on-chip RAM	&S_data[0]
DTC_R.DTDAR		DTC destination address register (DTDAR) 32-bit register that specifies transfer destination address of DTC transfer data	Located in on-chip RAM	&D_data
P_DTC.DTBR		DTC information base register (DTBR) Specifies upper 16 bits of memory address that stores DTC transfer information	H'FFFF8708	0xFFFF
P_DTC.DTED	CMI0	DTC enable register D (DTED) When set to 1, corresponding interrupt source is selected as DTC activation source: When CMI0 (DTED5) = b'1, CMT0 CMI0 interrupt is activation source	H'FFFF8703 Bit 5	1

(4) RAM Used

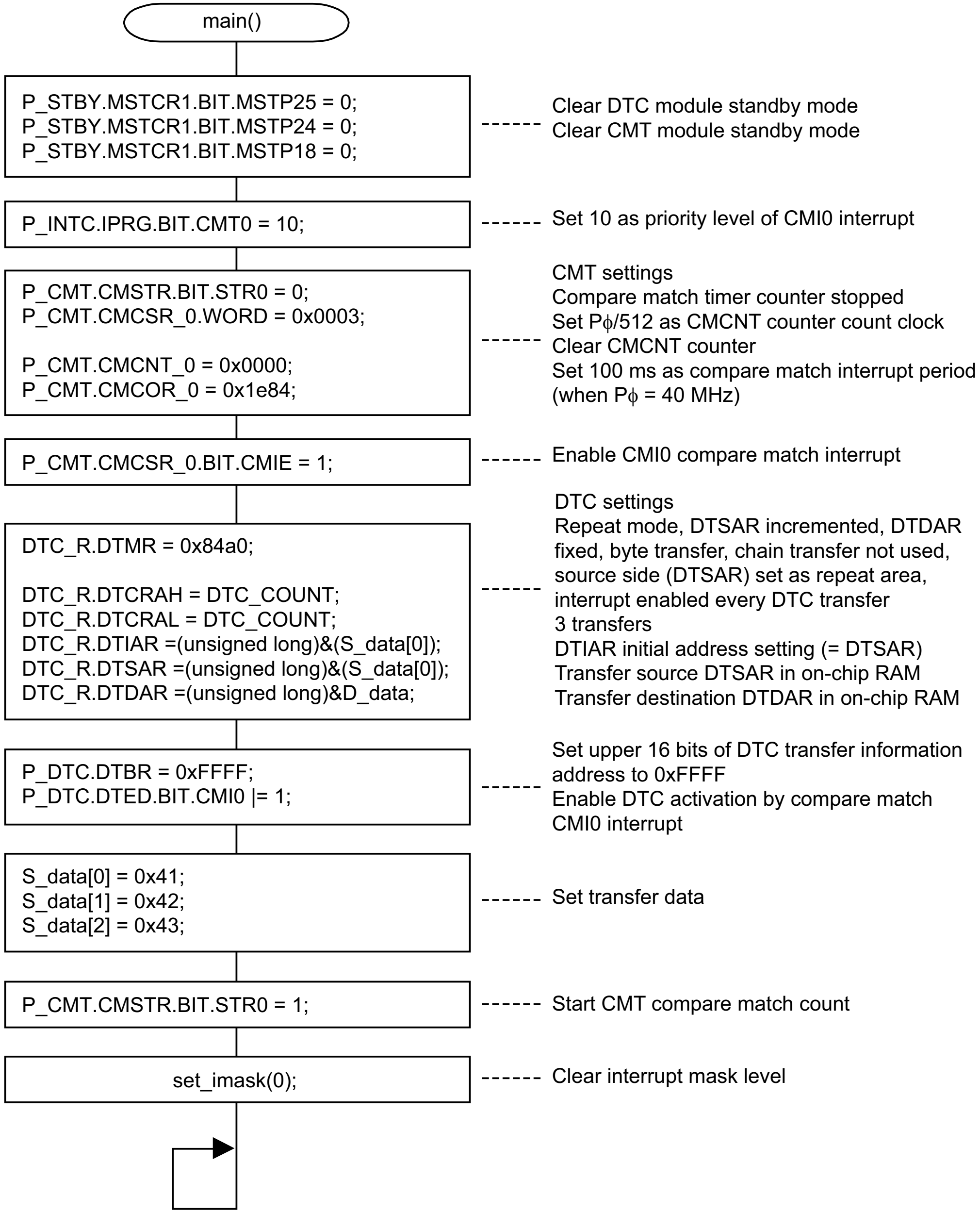
The following table shows the RAM used by this sample task.

Table 2.14 RAM Used

Label	Function	Address	Module Using RAM
S_data	DTC transfer data storage Array storing 3-byte data	On-chip RAM	Main routine
D_data	Data storage after DTC data transfer Array storing 1-byte data	On-chip RAM	Main routine

Flowcharts

(a) Main processing



main()

```
P_STBY.MSTCR1.BIT.MSTP25 = 0;
P_STBY.MSTCR1.BIT.MSTP24 = 0;
P_STBY.MSTCR1.BIT.MSTP18 = 0;
```

Clear DTC module standby mode
Clear CMT module standby mode

```
P_INTC.IPRG.BIT.CMT0 = 10;
```

Set 10 as priority level of CMI0 interrupt

```
P_CMT.CMSTR.BIT.STR0 = 0;
P_CMT.CMCSR_0.WORD = 0x0003;

P_CMT.CMCNT_0 = 0x0000;
P_CMT.CMCOR_0 = 0x1e84;
```

CMT settings
Compare match timer counter stopped
Set Pϕ/512 as CMCNT counter count clock
Clear CMCNT counter
Set 100 ms as compare match interrupt period (when Pϕ = 40 MHz)

```
P_CMT.CMCSR_0.BIT.CMIE = 1;
```

Enable CMI0 compare match interrupt

```
DTC_R.DTMR = 0x84a0;

DTC_R.DTCRAH = DTC_COUNT;
DTC_R.DTCRAL = DTC_COUNT;
DTC_R.DTIAR =(unsigned long)&(S_data[0]);
DTC_R.DTSAR =(unsigned long)&(S_data[0]);
DTC_R.DTDAR =(unsigned long)&D_data;
```

DTC settings
Repeat mode, DTSAR incremented, DTDAR fixed, byte transfer, chain transfer not used, source side (DTSAR) set as repeat area, interrupt enabled every DTC transfer
3 transfers
DTIAR initial address setting (= DTSAR)
Transfer source DTSAR in on-chip RAM
Transfer destination DTDAR in on-chip RAM

```
P_DTC.DTBR = 0xFFFF;
P_DTC.DTED.BIT.CMI0 |= 1;
```

Set upper 16 bits of DTC transfer information address to 0xFFFF
Enable DTC activation by compare match CMI0 interrupt

```
S_data[0] = 0x41;
S_data[1] = 0x42;
S_data[2] = 0x43;
```

Set transfer data

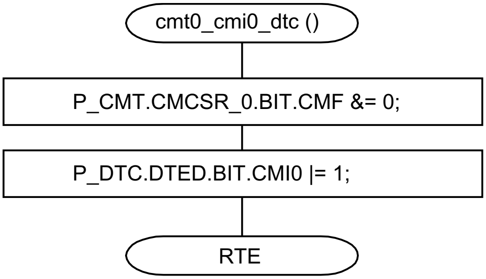
```
P_CMT.CMSTR.BIT.STR0 = 1;
```

Start CMT compare match count

```
set_imask(0);
```

Clear interrupt mask level

(b) Compare match interrupt handling



------ Clear compare match flag CMF

------ Enable DTC activation by compare match CMI0 interrupt again

Program Listing

```
/*******************************************************/
/* SH7046F Series -SH7047- Application Note            */
/* Data transfer Controller(DTC)                       */
/*     Repeat mode                                     */
/* Function                                            */
/*   :Data transfer Controller(DTC)                    */
/*   :Compare Match Timer(CMT ch0)                     */
/*                                                     */
/* External input clock          :10MHz                */
/* Internal CPU clock            :40MHz                */
/* Internal peripheral clock     :40MHz                */
/*                                                     */
/* Written   :       2002/3/1  Rev.1.0                 */
/*******************************************************/

#include "iodefine.h"
#include <machine.h>

/*------------ Symbol Definition ---------------------------------------------*/
struct st_dtc_repeat{      /* DTC  Repeat Mode information                    */
    unsigned short DTMR;   /* DTC Mode Register                               */
    unsigned char DTCRAH;  /* maintains the Transfer count                    */
    unsigned char DTCRAL;  /* Transfer counter                                */
    unsigned long DTIAR;   /* Initial Address Register                        */
    unsigned long DTSAR;   /* source address register                         */
    unsigned long DTDAR;   /* destination address register                    */
};

#define DTC_COUNT  3                       /* DTC Transmit count              */
#define DTC_R (*(volatile struct st_dtc_repeat*)0xFFFFE000)
                                           /* DTC information address         */


/*------------ Function Definition -------------------------------------------*/
void main(void);
void cmt0_cmi0_dtc(void);


/*------------ RAM allocation Definition -------------------------------------*/
unsigned char S_data[DTC_COUNT];     /* source buffer memory                  */
unsigned char D_data;                /* destination buffer memory             */



/***********************************************************/
/* main Program                                            */
/***********************************************************/
void main( void )
{

    /* Set standby mode */
    P_STBY.MSTCR1.BIT.MSTP25 = 0;            /* Disable DTC standby mode      */
    P_STBY.MSTCR1.BIT.MSTP24 = 0;            /* Disable DTC standby mode      */
    P_STBY.MSTCR2.BIT.MSTP12 = 0;            /* Disable CMT standby mode      */
    /* Set interrupt priority level (0 to 15) */
    P_INTC.IPRG.BIT.CMT0 = 10;                /* CMT0 CMI0 interrupt level 10   */



    /* Initialize CMT0 for Interval timer */
    P_CMT.CMSTR.BIT.STR0 = 0;         /* timer count stop                       */
    P_CMT.CMCSR_0.WORD = 0x0003;
                   /* CMF=0;          clear compare match flag                  */
                   /* CMIE=0;         compare match interrupt disable           */
                   /* CKS[1:0]=b'11;  clock = peripheral clock(Pϕ)/512          */
    P_CMT.CMCNT_0 = 0x0000;           /* timer counter clear                    */
    P_CMT.CMCOR_0 = 0x1e84;           /* 100ms@Pϕ=40MHz                         */

    P_CMT.CMCSR_0.BIT.CMIE = 1;       /* compare match interrupt enable         */

    /* DTC information */
    DTC_R.DTMR = 0x84a0;
                   /* SM[1:0]=b'10;   DTSAR is incremented                      */
                   /* DM[1:0]=b'00;   DTDAR is fixed                            */
                   /* MD[1:0]=b'01;   Repeat mode                               */
                   /* SZ[1:0]=b'00;   byte-size transfer                        */
                   /* DTS=1;          Source is Repeat area                     */
                   /* CHNE=0;         Chain transfer is disable                 */
                   /* DISEL=1;        Interrupt- every time                     */
                   /* NMIM=0;         NMI->Terminate DTC transfer               */

    DTC_R.DTCRAH = DTC_COUNT;                   /* maintains the Transfer count */
    DTC_R.DTCRAL = DTC_COUNT;                   /* DTC transfer Count           */
    DTC_R.DTIAR = (unsigned long)&(S_data[0]);  /* set Initial address          */
    DTC_R.DTSAR = (unsigned long)&(S_data[0]);  /* set source address           */
    DTC_R.DTDAR = (unsigned long)&D_data;       /* set destination address      */
    P_DTC.DTBR = 0xFFFF;                        /* DTC information base register*/

    /* DTC transmit enable */
    P_DTC.DTED.BIT.CMI0 |= 1;               /* interrupt sources CMT ch0(CMI0)  */

    /* set transmit data */
    S_data[0] = 0x41;
    S_data[1] = 0x42;
    S_data[2] = 0x43;


    P_CMT.CMSTR.BIT.STR0 = 1;               /* CMT0 timer count start           */

    set_imask(0);                           /* clear interrupt mask level       */

    while(1);
}
```

```
/************************************************************/
/* CMT0 Interrupt                                           */
/* Interval interrupt                                       */
/************************************************************/
#pragma interrupt(cmt0_cmi0_dtc)
void cmt0_cmi0_dtc(void)
{
    P_CMT.CMCSR_0.BIT.CMF &= 0;         /* Clear CMT0 compare match flag        */

    P_DTC.DTED.BIT.CMI0 |= 1;           /* set interrupt sources CMT ch0(CMI0)  */

}
```

2.3 Data Transfer Using DTC Block Transfer Mode (CMT, DTC)

Data Transfer Using DTC Block Transfer Mode (CMT, DTC)	Functions Used: CMT, DTC

Specifications

(1) The data transfer controller (DTC) is activated by a compare match timer (CMT) compare match interrupt, and performs data transfer from on-chip RAM to on-chip RAM, as shown in figure 2.13.

(2) Block transfer mode is used for DTC data transfer, with 3 transfers of a 4-byte data block performed as shown in figure 2.14.

(3) The DTC transfer conditions are shown in table 2.15.

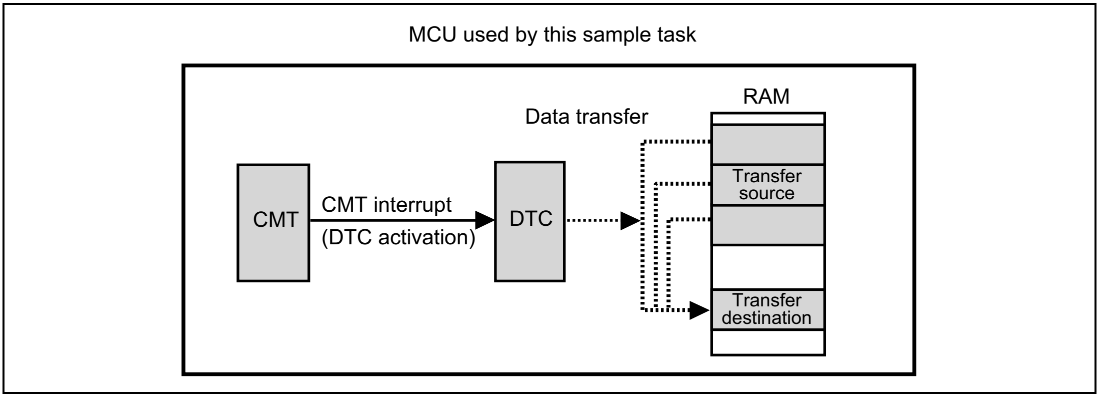


Figure 2.13 Data Transfer Using DTC

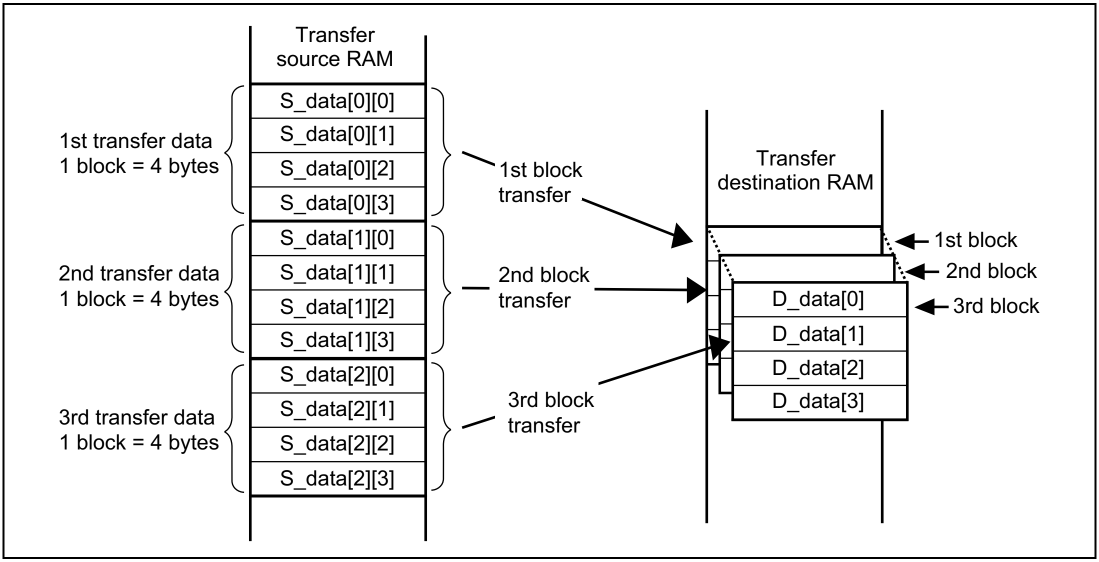


Figure 2.14 Data Transfer in DTC Normal Mode

Table 2.15 DTC Transfer Conditions

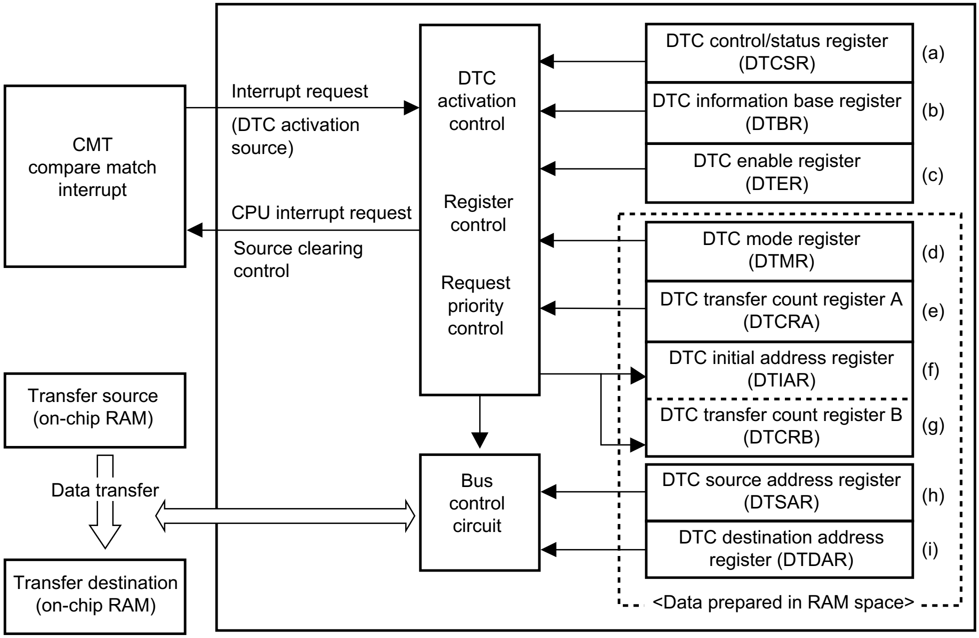
Condition	Description
Transfer mode	Block transfer mode, destination side (transfer destination) set as block area
Number of transfers	3
Block length	4
Transfer data size	Byte transfer
Transfer source	On-chip RAM
Transfer destination	On-chip RAM (block area)
Transfer source address	Transfer source address incremented after transfer
Transfer destination address	Transfer destination address incremented after transfer
Activation source	Activated by CMT ch0 compare match interrupt (CMI0)
Interrupt handling	Interrupt to CPU enabled only at end of specified data transfer

Functions Used

(1) A block diagram of the DTC is shown below. Of the three DTC transfer modes — normal mode, repeat mode, and block transfer mode —this sample task uses normal mode to perform data transfer. Data transfer is performed from on-chip RAM to on-chip RAM, using a CMT compare match interrupt as the DTC activation source. The block diagram is explained below.

- The DTC mode register (DTMR) is a 16-bit register that controls the DTC's operating mode.
- The DTC source address register (DTSAR) is a 32-bit register that specifies the transfer source address of data to be transferred by the DTC. An even address should be specified in the case of word size transfer, and an address that is a multiple of 4 in the case of longword transfer.
- The DTC destination address register (DTDAR) is a 32-bit register that specifies the transfer destination address of data to be transferred by the DTC. An even address should be specified in the case of word size transfer, and an address that is a multiple of 4 in the case of longword transfer.
- The DTC initial address register (DTIAR) is a 32-bit register that specifies the transfer source/transfer destination initial address in repeat mode. In repeat mode, when the DTS bit is 1, specify the initial transfer source address in the repeat area, and when the DTS bit is 0, specify the initial transfer destination address in the repeat area.
- DTC transfer count register A (DTCRA) is a 16-bit register that specifies the number of transfers in DTC data transfer. In normal mode, DTCRA functions as a transfer counter (1 to 65,536). In repeat mode, upper 8-bit DTCRAH holds the number of transfers, and lower 8-bit DTCRAL functions as an 8-bit transfer counter. In block transfer mode, DTCRA functions as a 16-bit transfer counter.
- DTC transfer count register B (DTCRB) is a 16-bit register that specifies the block length in block transfer mode.
- The DTC enable register (DTER) is used to select the interrupt source that activates the DTC, and comprises registers DTEA through DTEF.
- The DTC control/status register (DTCSR) is a 16-bit register that sets enabling/disabling of DTC activation by software, and sets a software activation DTC vector address. DTCSR also shows the DTC transfer status.
- The DTC information base register (DTBR) is a readable/writable 16-bit register that specifies the upper 16 bits of the memory address that stores DTC transfer information. Word or longword access must be used for DTBR. If byte access is used, the register contents will be undefined in the case of a write, and an undefined value will be returned in the case of a read.
- Information of six registers — the DTC mode register (DTMR), DTC source address register (DTSAR), DTC destination address register (DTDAR), DTC initial address register (DTIAR), DTC transfer count register A (DTCRA), and DTC transfer count register B (DTCRB) — cannot be accessed directly from the CPU. When a DTC activation

source occurs, the relevant register information is transferred to these registers from information of an arbitrary set of registers located in on-chip RAM and DTC transfer is performed, and when transfer ends, the contents of these registers are returned to RAM. Therefore, register information should be prepared in arbitrary on-chip RAM in the user program.


CMT
compare match
interrupt
Interrupt request
(DTC activation source)
CPU interrupt request
Source clearing control
DTC activation control
Register control
Request priority control
DTC control/status register (DTCSR)
(a)
DTC information base register (DTBR)
(b)
DTC enable register (DTER)
(c)
DTC mode register (DTMR)
(d)
DTC transfer count register A (DTCRA)
(e)
DTC initial address register (DTIAR)
(f)
DTC transfer count register B (DTCRB)
(g)
DTC source address register (DTSAR)
(h)
DTC destination address register (DTDAR)
(i)
<Data prepared in RAM space>
Transfer source (on-chip RAM)
Data transfer
Transfer destination (on-chip RAM)
Bus control circuit

Notes:

(a) Performs enabling/disabling of DTC activation by software, and software activation DTC vector address setting.

(b) Performs specification of the upper 16 bits of the memory address that stores DTC transfer information.

(c) Selects the interrupt source that activates the DTC; comprises six registers, DTEA through DTEF.

(d) Performs DTC operating mode setting.

(e) Specifies the number of transfers in DTC data transfer.

(f) In repeat mode, specifies the transfer source/transfer destination initial address in repeat mode. Not used in normal mode. In block transfer mode, functions as the DTCRB register.

(g) In block transfer mode, specifies the block length. Not used in normal mode. In repeat mode, functions as the DTIAR register.

(h) Specifies the transfer source address of data to be transferred by the DTC.

(i) Specifies the transfer destination address of data to be transferred by the DTC.

Figure 2.15 DTC Block Diagram

(2) A block diagram of CMT ch0 is shown is the figure below. In this task, DTC data transfer is performed using a CMT ch0 compare match interrupt as the activation source. The block diagram is explained below.

- The compare match timer start register (CMSTR) is a 16-bit register that is used to set whether the channel 0 and 1 counters (CMCNT) are operated or stopped.
- Compare match timer control/status register 0 (CMCSR_0) is a 16-bit register that performs compare match generation indication, interrupt enabling/disabling, and selection of the clock used for counting up.
- Compare match timer counter 0 (CMCNT_0) is a 16-bit register used as an up-counter for generating an interrupt request.
- Compare match timer constant register 0 (CMCOR_0) is a 16-bit register used to set the CMCNT compare match period.

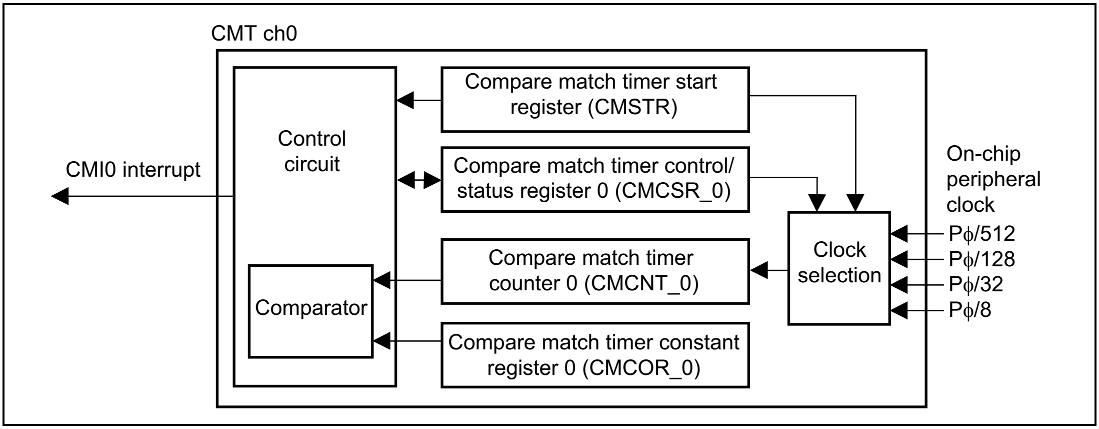

CMT ch0
CMI0 interrupt
Control circuit
Comparator
Compare match timer start register (CMSTR)
Compare match timer control/ status register 0 (CMCSR_0)
Compare match timer counter 0 (CMCNT_0)
Compare match timer constant register 0 (CMCOR_0)
Clock selection
On-chip peripheral clock
Pφ/512
Pφ/128
Pφ/32
Pφ/8

Figure 2.16 CMT Block Diagram

(4) Table 2.16 shows the function assignments used in this sample task.

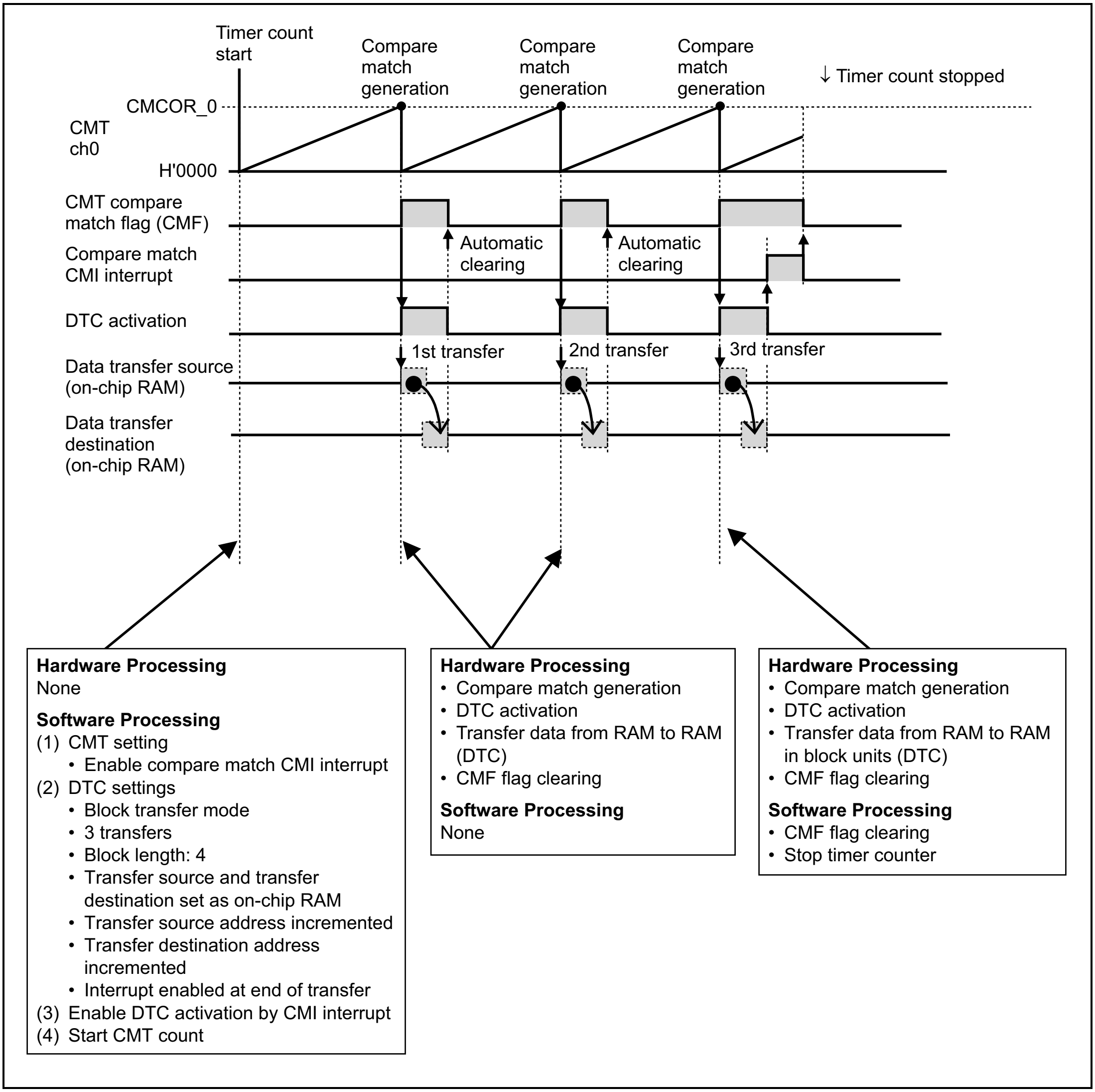
Table 2.16 Function Assignments

Function	Type	Function Assignment
DTMR	DTC	Block transfer mode, destination side (transfer destination) set as block area
DTCRA	DTC	Setting of number of transfers
DTSAR	DTC	Transfer source address setting
DTDAR	DTC	Transfer destination address setting
DTBR	DTC	Setting of DTC vector upper 16 bits
DTER	DTC	Enables DTC activation by CMT ch0 CMI interrupt
CMSTR	CMT	CMT count start
CMCSR_0	CMT ch0	Count clock selection, interrupt control
CMCNT_0	CMT ch0	Counter
CMCOR_0	CMT ch0	Period setting

Operation

(1) The principles of operation of this sample task are shown in the figure below.

Data transfer from on-chip RAM to on-chip RAM is performed by the DTC by means of hardware and software processing as shown in the figure.



Hardware Processing
None

Software Processing
(1) CMT setting
- Enable compare match CMI interrupt

(2) DTC settings
- Block transfer mode
- 3 transfers
- Block length: 4
- Transfer source and transfer destination set as on-chip RAM
- Transfer source address incremented
- Transfer destination address incremented
- Interrupt enabled at end of transfer

(3) Enable DTC activation by CMI interrupt
(4) Start CMT count

Hardware Processing
- Compare match generation
- DTC activation
- Transfer data from RAM to RAM (DTC)
- CMF flag clearing

Software Processing
None

Hardware Processing
- Compare match generation
- DTC activation
- Transfer data from RAM to RAM in block units (DTC)
- CMF flag clearing

Software Processing
- CMF flag clearing
- Stop timer counter

Figure 2.17 Principles of Operation

(2) The principles of operation of DTC activation are shown in the figure below. When executing DTC transfer, the following settings should be made before an activation source occurs.

- Make DTC register information settings and place DTC register information in RAM.
- Set the lower 16 bits of the start address (32 bits) of DTC register information in the DTC vector table.
- Set the lower 16 bits of the start address (32 bits) of DTC register information in the DTC information base register.

The DTC is activated by the following processing.

- A DTC activation source interrupt is generated.
- The lower 16 bits of the start address of DTC register information are read from the address corresponding to the DTC vector table activation source.
- The upper 16 bits of the start address of DTC register information are read from the DTC information base register (DTMR).
- The 32-bit start address of DTC register information is generated from the read start address lower 16 bits and upper 16 bits.
- The start of DTC register information is read sequentially from the DTC register information start address, and data transfer is performed.

In this task, a TXI_2 interrupt is the activation source in the case of serial transmission data transfer, and an RXI_2 interrupt is the activation source in the case of serial reception data transfer.

The following table shows the register information configuration in block transfer mode.

Table 2.17 DTC Register Information (Block Transfer Mode)

Setting Address	Register Name	Data Length
RF	DTC mode register (DTMR)	Word (2 bytes)
RF+2	DTC transfer count register A (DTCRA)	Word (2 bytes)
RF+6	DTC transfer count register B (DTCRB)	Word (2 bytes)
RF+8	DTC source address register (DTSAR)	Longword (4 bytes)
RF+12	DTC destination address register (DTDAR)	Longword (4 bytes)

RF: DTC register information start address (in on-chip RAM)

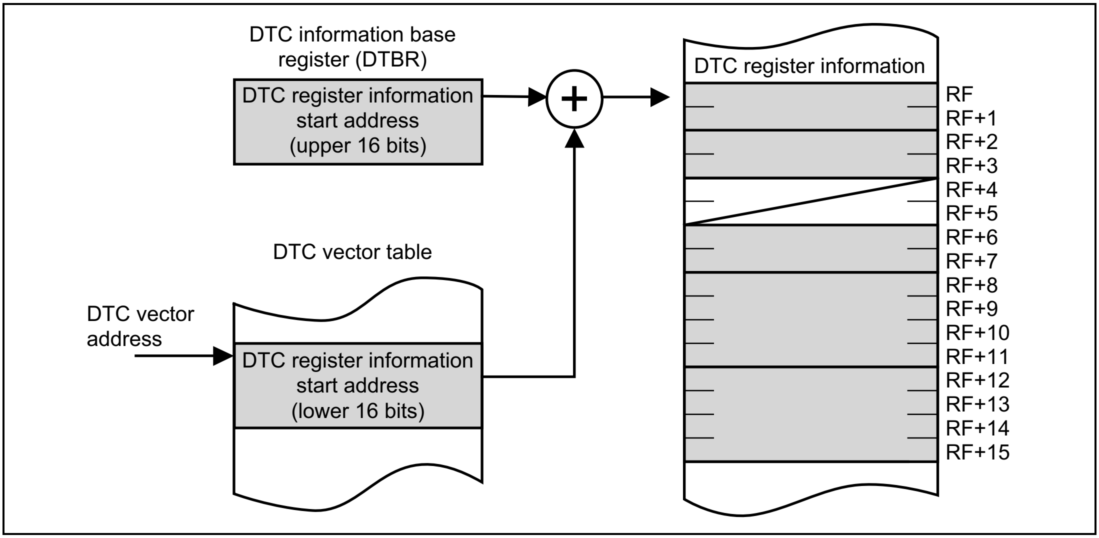

DTC information base register (DTBR)
DTC register information start address (upper 16 bits)
+
DTC register information
RF
RF+1
RF+2
RF+3
RF+4
RF+5
RF+6
RF+7
RF+8
RF+9
RF+10
RF+11
RF+12
RF+13
RF+14
RF+15
DTC vector table
DTC vector address
DTC register information start address (lower 16 bits)

Figure 2.18 Correspondence between DTC Vector Address and Transfer Information

Software

(1) Modules

The following table shows the modules used by this sample task.

Table 2.18 Modules

Module Name	Label	Function
Main routine	main	CMT timer setting, DTC initialization, timer start
CMI0 interrupt	cmt0_cmi0_dtc	CMT ch0 compare match interrupt (CMI0). Interrupt generation at end of specified number of DTC transfers

(2) Arguments

The following table shows the arguments used by this sample task.

Table 2.19 Arguments

Argument	Function	Module Name	Data Length	Input/ Output
S_data [0][0] to [2][3]	DTC transfer source transfer data storage	Main routine	1 byte	Output
D_data [0] to [3]	DTC transfer destination transfer data storage	Main routine	1 byte	Input

(3) Internal Registers Used

The following table shows the internal registers used by this sample task.

Table 2.20 Internal Registers Used

Register Name		Function	Address	Set Value
	Bits		Bits	
P_STBY.MSTCR1	MSTP25 MSTP24	Module standby control register 1 DTC module standby control bits: When MSTP25 = MSTP24 = 0, module standby release Same value is set for MSTP25 and MSTP24	H'FFFF861C Bit 9 Bit 8	B'00
P_STBY.MSTCR2	MSTP12	Module standby control register 2 CMT module standby control bit: When MSTP12 = 0, module standby release	H'FFFF861E Bit 12	0
P_INTC.IPRG	CMT0	Interrupt priority register G (IPRG) CMT0 CMI0 interrupt priority level setting: When CMT0 = b'1010 (10), CMI0 interrupt is set to priority level 10	H'FFFF8354 Bits 7 to 4	10
P_CMT.CMSTR		Compare match timer start register (CMTSR) 16-bit register that selects CMCNT operation/stoppage	H'FFFF83D0	H'0001
	STR1	Counter start 1: When STR1 = b'0, TCNT_1 count operation is stopped	Bit 1	
	STR0	Counter start 0: When STR0 = b'1, TCNT_0 counts	Bit 0	

Register Name		Function	Address	Set Value
	Bits		Bits	
P_CMT.CMCSR_0		Compare match timer control/status register 0 (CMCSR_0) Compare match generation indication, interrupt setting, timer clock setting	H'FFFF83D2	H'0043
	CMF	Compare match flag: CMF is set to 1 when CMCNT and CMCOR values match	Bit 7	
	CMIE	Compare match interrupt enable: When CMIE = 1, compare match interrupt (CMI) is enabled	Bit 6	
	CKS1 CKS0	CMCNT counter clock selection: When CKS[1:0] = b'11, count is performed using internal clock Pϕ/512	Bit 1 Bit 0	
P_CMT.CMCNT_0		Compare match timer counter 0 (CMCNT_0) 16-bit register used as up-counter for generating interrupt requests	H'FFFF83D4	H'0000
P_CMT.CMCOR_0		Compare match timer constant register 0 (CMCOR_0) 16-bit register used to set CMCNT compare match period When CMCOR_0 = H'1e84, 100 ms compare match period is used (Pϕ/512 count, Pϕ = 40 MHz)	H'FFFF83D6	H'1e84

Register Name		Function	Address	Set Value
	Bits		Bits	
DTC_B.DTMR		DTC mode register (DTMR) DTC operating mode control setting	Located in on-chip RAM	H'a800
	SM1 SM0	Source address mode: When SM[1:0] = b'10, DTSAR is incremented after transfer	Bit 15 Bit 14	
	DM1 DM0	Destination address mode: When DM[1:0] = b'10, DTDAR is incremented after transfer	Bit 13 Bit 12	
	MD1 MD0	DTC transfer mode: When MD[1:0] = b'10, block transfer mode	Bit 11 Bit 10	
	SZ1 SZ0	DTC data transfer size: When SZ[1:0] = b'00, byte (1-byte) transfer	Bit 9 Bit 8	
	DTS	DTC transfer mode select: When DTS = b'0, destination side is block area	Bit 7	
	CHNE	DTC chain transfer enable: When CHNE = b'0, chain transfer is cleared	Bit 6	
	DISEL	DTC interrupt select: When DISEL = b'0, interrupt request to CPU is generated only at end of specified data transfer	Bit 5	
	NMIM	DTC NMI mode: When NMIM = b'0, DTC transfer is suspended by NMI	Bit 4	
DTC_B.DTCRA		DTC transfer count register A (DTCRA) Specifies number of transfers in DTC data transfer Set to 3 transfers	Located in on-chip RAM	H'0003
DTC_B.DTCRB		DTC transfer count register B (DTCRB) Specifies DTC block length Set to block length of 4	Located in on-chip RAM	H'0004
DTC_B.DTSAR		DTC source address register (DTSAR) 32-bit register that specifies transfer source address of data to be transferred by DTC	Located in on-chip RAM	&S_data[0][0];

Register Name		Function	Address	Set Value
	Bits		Bits	
DTC_B.DTDAR		DTC destination address register (DTDAR) 32-bit register that specifies transfer destination address of data to be transferred by DTC	Located in on-chip RAM	&D_data[0];
P_DTC.DTBR		DTC information base register (DTBR) Specifies upper 16 bits of memory address that stores DTC transfer information	H'FFFF8708	0xFFFF
P_DTC.DTED	CMI0	DTC enable register D (DTED) When set to 1, corresponding interrupt source is selected as DTC activation source: When CMI0 (DTED5) = b'1, CMT0 CMI0 interrupt is activation source	H'FFFF8703 Bit 5	1

(4) RAM Used

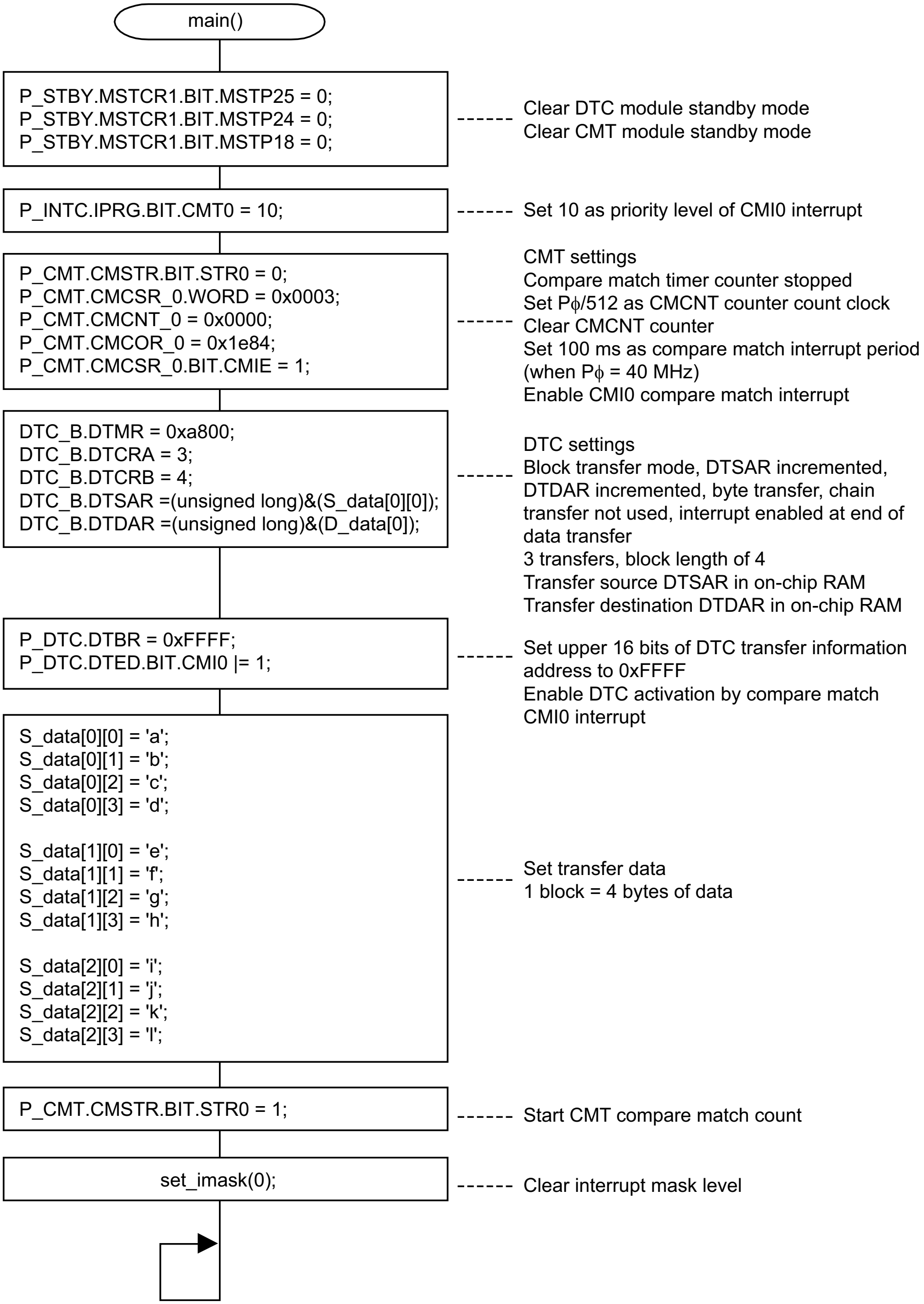
The following table shows the RAM used by this sample task.

Table 2.21 RAM Used

Label	Function	Address	Module Using RAM
S_data	DTC transfer data storage Array storing 3 blocks, each comprising 4-byte data	On-chip RAM	Main routine
D_data	Data storage after DTC data transfer Array storing 4-byte data	On-chip RAM	Main routine

Flowcharts

(a) Main processing



main()

```
P_STBY.MSTCR1.BIT.MSTP25 = 0;
P_STBY.MSTCR1.BIT.MSTP24 = 0;
P_STBY.MSTCR1.BIT.MSTP18 = 0;
```

------ Clear DTC module standby mode
Clear CMT module standby mode

```
P_INTC.IPRG.BIT.CMT0 = 10;
```

------ Set 10 as priority level of CMI0 interrupt

```
P_CMT.CMSTR.BIT.STR0 = 0;
P_CMT.CMCSR_0.WORD = 0x0003;
P_CMT.CMCNT_0 = 0x0000;
P_CMT.CMCOR_0 = 0x1e84;
P_CMT.CMCSR_0.BIT.CMIE = 1;
```

------ CMT settings
Compare match timer counter stopped
Set Pϕ/512 as CMCNT counter count clock
Clear CMCNT counter
Set 100 ms as compare match interrupt period (when Pϕ = 40 MHz)
Enable CMI0 compare match interrupt

```
DTC_B.DTMR = 0xa800;
DTC_B.DTCRA = 3;
DTC_B.DTCRB = 4;
DTC_B.DTSAR =(unsigned long)&(S_data[0][0]);
DTC_B.DTDAR =(unsigned long)&(D_data[0]);
```

------ DTC settings
Block transfer mode, DTSAR incremented, DTDAR incremented, byte transfer, chain transfer not used, interrupt enabled at end of data transfer
3 transfers, block length of 4
Transfer source DTSAR in on-chip RAM
Transfer destination DTDAR in on-chip RAM

```
P_DTC.DTBR = 0xFFFF;
P_DTC.DTED.BIT.CMI0 |= 1;
```

------ Set upper 16 bits of DTC transfer information address to 0xFFFF
Enable DTC activation by compare match CMI0 interrupt

```
S_data[0][0] = 'a';
S_data[0][1] = 'b';
S_data[0][2] = 'c';
S_data[0][3] = 'd';

S_data[1][0] = 'e';
S_data[1][1] = 'f';
S_data[1][2] = 'g';
S_data[1][3] = 'h';

S_data[2][0] = 'i';
S_data[2][1] = 'j';
S_data[2][2] = 'k';
S_data[2][3] = 'l';
```

------ Set transfer data
1 block = 4 bytes of data

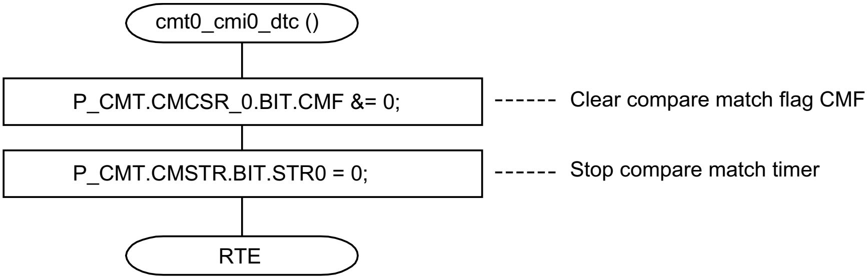
```
P_CMT.CMSTR.BIT.STR0 = 1;
```

------ Start CMT compare match count

```
set_imask(0);
```

------ Clear interrupt mask level

(b) Compare match interrupt handling


cmt0_cmi0_dtc ()
P_CMT.CMCSR_0.BIT.CMF &= 0;
------ Clear compare match flag CMF
P_CMT.CMSTR.BIT.STR0 = 0;
------ Stop compare match timer
RTE

Program Listing

```
/*******************************************************/
/* SH7046F Series -SH7047- Application Note            */
/* Data transfer Controller(DTC)                       */
/*     Block Transfer mode                             */
/* Function                                            */
/*  :Data transfer Controller(DTC)                     */
/*  :Compare Match Timer(CMT ch0)                      */
/*                                                     */
/* External input clock       :10MHz                   */
/* Internal CPU clock         :40MHz                   */
/* Internal peripheral clock  :40MHz                   */
/*                                                     */
/* Written    :    2002/3/1  Rev.1.0                   */
/*******************************************************/


#include "iodefine.h"
#include <machine.h>


/*------------ Symbol Definition ---------------------------------------------*/
struct st_dtc_block{                /* DTC Block Transfer Mode information      */
    unsigned short DTMR;            /* DTC Mode Register                        */
    unsigned short DTCRA;           /* Transfer counter                         */
    unsigned short dummy;           /* Reserved                                 */
    unsigned short DTCRB;           /* Block length                             */
    unsigned long DTSAR;            /* source address register                  */
    unsigned long DTDAR;            /* destination address register             */
};

#define DTC_COUNT  3                /* DTC Transmit count                       */
#define DTC_BLOCK_LENG  4           /* DTC block length                         */
#define DTC_B (*(volatile struct st_dtc_block*)0xFFFFE000)
                                    /* DTC information address                  */

/*------------ Function Definition -------------------------------------------*/
void main(void);
void cmt0_cmi0_dtc(void);


/*------------ RAM allocation Definition -------------------------------------*/
unsigned char S_data[DTC_COUNT][DTC_BLOCK_LENG]; /* source buffer memory        */
unsigned char D_data[DTC_BLOCK_LENG];            /* destination buffer memory   */



/***********************************************************/
/* main Program                                            */
/***********************************************************/
void main( void )
{

    /* Set standby mode  */
    P_STBY.MSTCR1.BIT.MSTP25 = 0;   /* Disable DTC standby mode                 */
    P_STBY.MSTCR1.BIT.MSTP24 = 0;   /* Disable DTC standby mode                 */
    P_STBY.MSTCR2.BIT.MSTP12 = 0;   /* Disable CMT standby mode                 */


    /* Set interrupt priority level (0 to 15)  */
    P_INTC.IPRG.BIT.CMT0 = 10;      /* CMT0 CMI0 interrupt level 5              */
                                    /* Initialize CMT0 for Interval timer       */
    P_CMT.CMSTR.BIT.STR0 = 0;       /* timer count stop                         */
    P_CMT.CMCSR_0.WORD = 0x0003;
    /* CMF=0;                       clear compare match flag                    */
    /* CMIE=0;                      compare match interrupt disable             */
    /* CKS[1:0]=b'11;               clock = peripheral clock(Pϕ)/512            */
    P_CMT.CMCNT_0 = 0x0000;         /* timer counter clear                      */
    P_CMT.CMCOR_0 = 0x1e84;         /* 100ms@Pϕ=40MHz                           */
    P_CMT.CMCSR_0.BIT.CMIE = 1;     /* compare match interrupt enable

    /* DTC information  */
    DTC_B.DTMR = 0xa800;            /*                                          */
               /* SM[1:0]=b'10;     DTSAR is incremented                        */
               /* DM[1:0]=b'00;     DTDAR is incremented                        */
               /* MD[1:0]=b'10;     Block transfer mode                         */
               /* SZ[1:0]=b'00;     byte-size transfer                          */
               /* DTS=0;            destination is Block area                   */
               /* CHNE=0;           Chain transfer is canceled                  */
               /* DISEL=0;          Interrupt->transfer ends                    */
               /* NMIM=0;           NMI->Terminate DTC transfer                 */
    DTC_B.DTCRA = DTC_COUNT;        /* DTC transfer Count                       */
    DTC_B.DTCRB = DTC_BLOCK_LENG;   /* DTC transfer Block length                */
    DTC_B.DTSAR =(unsigned long)&(S_data[0][0]);  /* set source address         */
    DTC_B.DTDAR =(unsigned long)&(D_data[0]);     /* set destination address    */
    P_DTC.DTBR = 0xFFFF;            /* DTC information base register            */

    /* DTC transmit enable  */
    P_DTC.DTED.BIT.CMI0 |= 1;       /* interrupt sources CMT ch0(CMI0)          */

    /* set transmit data  block 1  */
    S_data[0][0] = 'a';
    S_data[0][1] = 'b';
    S_data[0][2] = 'c';
    S_data[0][3] = 'd';
    /* set transmit data  block 2  */
    S_data[1][0] = 'e';
    S_data[1][1] = 'f';
    S_data[1][2] = 'g';
    S_data[1][3] = 'h';
    /*  set transmit data  block 3  */
    S_data[2][0] = 'i';
    S_data[2][1] = 'j';
    S_data[2][2] = 'k';
    S_data[2][3] = 'l';

    P_CMT.CMSTR.BIT.STR0 = 1;       /* CMT0 timer count start                   */
    set_imask(0);                   /* clear interrupt mask level               */
    while(1);
}
```

```
/************************************************************/
/* CMT0 Interrupt                                           */
/* Interval interrupt                                       */
/************************************************************/
#pragma interrupt(cmt0_cmi0_dtc)
void cmt0_cmi0_dtc(void)
{
    P_CMT.CMCSR_0.BIT.CMF &= 0;     /* Clear CMT0 compare match flag            */
    P_CMT.CMSTR.BIT.STR0 = 0;       /* CMT0 timer count stop                    */

}
```

2.4 Data Transfer Using DTC Chain Transfer (CMT, DTC)

Data Transfer Using DTC Chain Transfer (CMT, DTC)	**Functions Used: CMT, DTC**

Specifications

(1) The data transfer controller (DTC) is activated by a compare match timer (CMT) compare match interrupt, and performs data transfer from on-chip RAM to on-chip RAM, as shown in figure 2.19.

(2) DTC data transfer is performed by means of chain transfer, with 3-byte transfers performed consecutively as shown in figure 2.20.

(3) The DTC transfer conditions are shown in table 2.22.

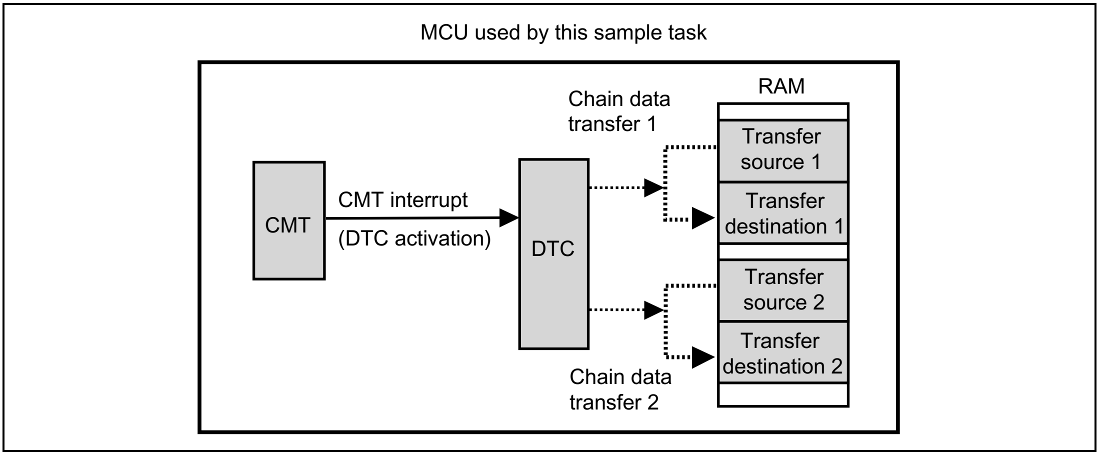


Figure 2.19 Data Transfer Using DTC

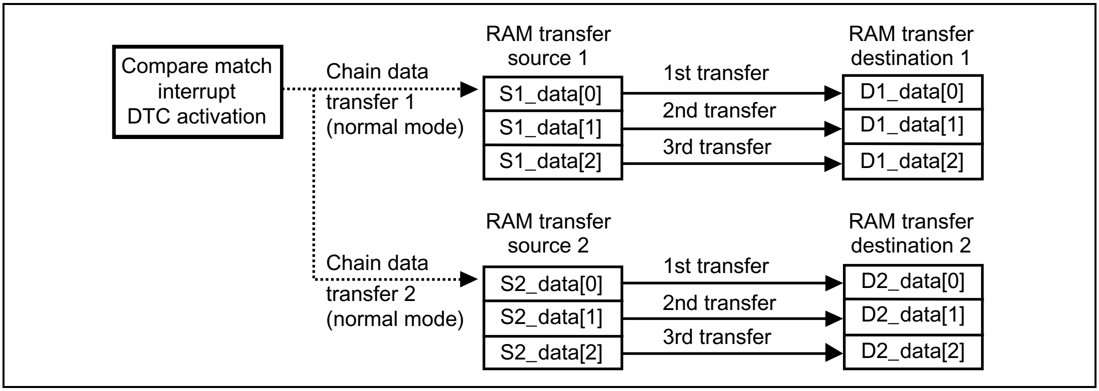


Figure 2.20 Data Transfer in DTC Normal Mode

Table 2.22 DTC Transfer Conditions

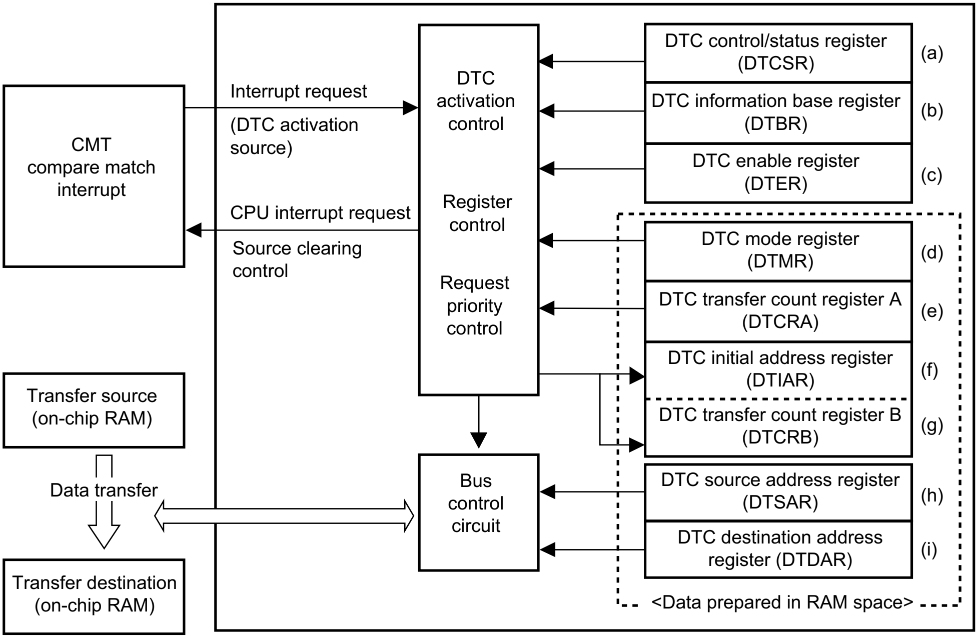
Condition	Description of Chain Transfer 1	Description of Chain Transfer 2
Transfer mode	Normal mode, chain transfer	Normal mode
Number of transfers	3	3
Transfer data size	Byte transfer	Byte transfer
Transfer source	On-chip RAM	On-chip RAM
Transfer destination	On-chip RAM	On-chip RAM
Transfer source address	Transfer source address incremented after transfer	Transfer source address incremented after transfer
Transfer destination address	Transfer destination address incremented after transfer	Transfer destination address incremented after transfer
Activation source	Activated by CMT ch0 compare match interrupt (CMI0)	Executed at end of chain transfer 1
Interrupt handling	None (because of chain transfer)	Interrupt to CPU enabled only at end of specified data transfer

Functions Used

(1) A block diagram of the DTC is shown below. Of the three DTC transfer modes — normal mode, repeat mode, and block transfer mode —this sample task uses normal mode to perform data transfer. Data transfer is performed from on-chip RAM to on-chip RAM, using a CMT compare match interrupt as the DTC activation source. The block diagram is explained below.

- The DTC mode register (DTMR) is a 16-bit register that controls the DTC's operating mode.
- The DTC source address register (DTSAR) is a 32-bit register that specifies the transfer source address of data to be transferred by the DTC. An even address should be specified in the case of word size transfer, and an address that is a multiple of 4 in the case of longword transfer.
- The DTC destination address register (DTDAR) is a 32-bit register that specifies the transfer destination address of data to be transferred by the DTC. An even address should be specified in the case of word size transfer, and an address that is a multiple of 4 in the case of longword transfer.
- The DTC initial address register (DTIAR) is a 32-bit register that specifies the transfer source/transfer destination initial address in repeat mode. In repeat mode, when the DTS bit is 1, specify the initial transfer source address in the repeat area, and when the DTS bit is 0, specify the initial transfer destination address in the repeat area.
- DTC transfer count register A (DTCRA) is a 16-bit register that specifies the number of transfers in DTC data transfer. In normal mode, DTCRA functions as a transfer counter (1 to 65,536). In repeat mode, upper 8-bit DTCRAH holds the number of transfers, and lower 8-bit DTCRAL functions as an 8-bit transfer counter. In block transfer mode, DTCRA functions as a 16-bit transfer counter.
- DTC transfer count register B (DTCRB) is a 16-bit register that specifies the block length in block transfer mode.
- The DTC enable register (DTER) is used to select the interrupt source that activates the DTC, and comprises registers DTEA through DTEF.
- The DTC control/status register (DTCSR) is a 16-bit register that sets enabling/disabling of DTC activation by software, and sets a software activation DTC vector address. DTCSR also shows the DTC transfer status.
- The DTC information base register (DTBR) is a readable/writable 16-bit register that specifies the upper 16 bits of the memory address that stores DTC transfer information. Word or longword access must be used for DTBR. If byte access is used, the register contents will be undefined in the case of a write, and an undefined value will be returned in the case of a read.
- Information of six registers — the DTC mode register (DTMR), DTC source address register (DTSAR), DTC destination address register (DTDAR), DTC initial address register (DTIAR), DTC transfer count register A (DTCRA), and DTC transfer count register B (DTCRB) — cannot be accessed directly from the CPU. When a DTC activation

source occurs, the relevant register information is transferred to these registers from information of an arbitrary set of registers located in on-chip RAM and DTC transfer is performed, and when transfer ends, the contents of these registers are returned to RAM. Therefore, register information should be prepared in arbitrary on-chip RAM in the user program.


CMT compare match interrupt
Interrupt request
(DTC activation source)
CPU interrupt request
Source clearing control
DTC activation control
Register control
Request priority control
DTC control/status register (DTCSR)
(a)
DTC information base register (DTBR)
(b)
DTC enable register (DTER)
(c)
DTC mode register (DTMR)
(d)
DTC transfer count register A (DTCRA)
(e)
DTC initial address register (DTIAR)
(f)
DTC transfer count register B (DTCRB)
(g)
DTC source address register (DTSAR)
(h)
DTC destination address register (DTDAR)
(i)
<Data prepared in RAM space>
Transfer source (on-chip RAM)
Data transfer
Transfer destination (on-chip RAM)
Bus control circuit

Notes:

(a) Performs enabling/disabling of DTC activation by software, and software activation DTC vector address setting.

(b) Performs specification of the upper 16 bits of the memory address that stores DTC transfer information.

(c) Selects the interrupt source that activates the DTC; comprises six registers, DTEA through DTEF.

(d) Performs DTC operating mode setting.

(e) Specifies the number of transfers in DTC data transfer.

(f) In repeat mode, specifies the transfer source/transfer destination initial address in repeat mode. Not used in normal mode. In block transfer mode, functions as the DTCRB register.

(g) In block transfer mode, specifies the block length. Not used in normal mode. In repeat mode, functions as the DTIAR register.

(h) Specifies the transfer source address of data to be transferred by the DTC.

(i) Specifies the transfer destination address of data to be transferred by the DTC.

Figure 2.21 DTC Block Diagram

(2) A block diagram of CMT ch0 is shown is the figure below. In this task, DTC data transfer is performed using a CMT ch0 compare match interrupt as the activation source. The block diagram is explained below.

- The compare match timer start register (CMSTR) is a 16-bit register that is used to set whether the channel 0 and 1 counters (CMCNT) are operated or stopped.
- Compare match timer control/status register 0 (CMCSR_0) is a 16-bit register that performs compare match generation indication, interrupt enabling/disabling, and selection of the clock used for counting up.
- Compare match timer counter 0 (CMCNT_0) is a 16-bit register used as an up-counter for generating an interrupt request.
- Compare match timer constant register 0 (CMCOR_0) is a 16-bit register used to set the CMCNT compare match period.

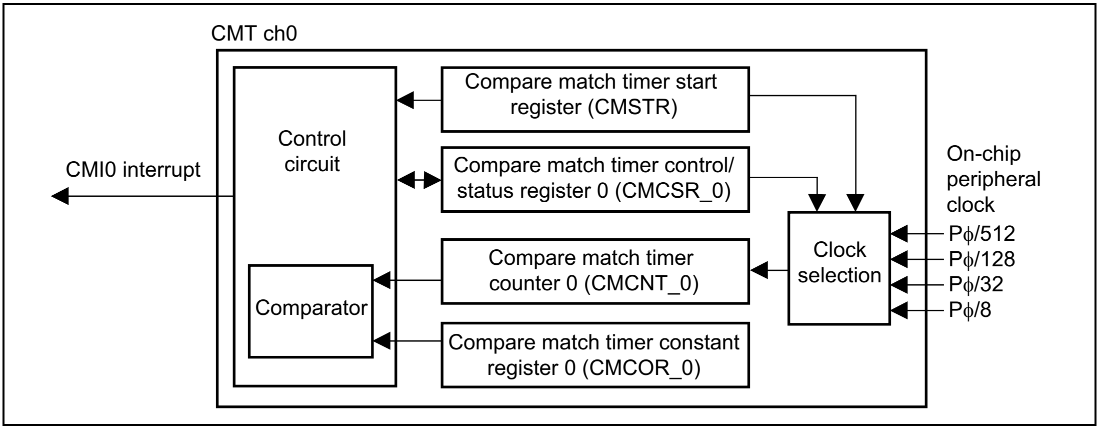

CMT ch0
CMI0 interrupt
Control circuit
Comparator
Compare match timer start register (CMSTR)
Compare match timer control/ status register 0 (CMCSR_0)
Compare match timer counter 0 (CMCNT_0)
Compare match timer constant register 0 (CMCOR_0)
Clock selection
On-chip peripheral clock
Pϕ/512
Pϕ/128
Pϕ/32
Pϕ/8

Figure 2.22 CMT Block Diagram

(3) Table 2.23 shows the function assignments used in this sample task.

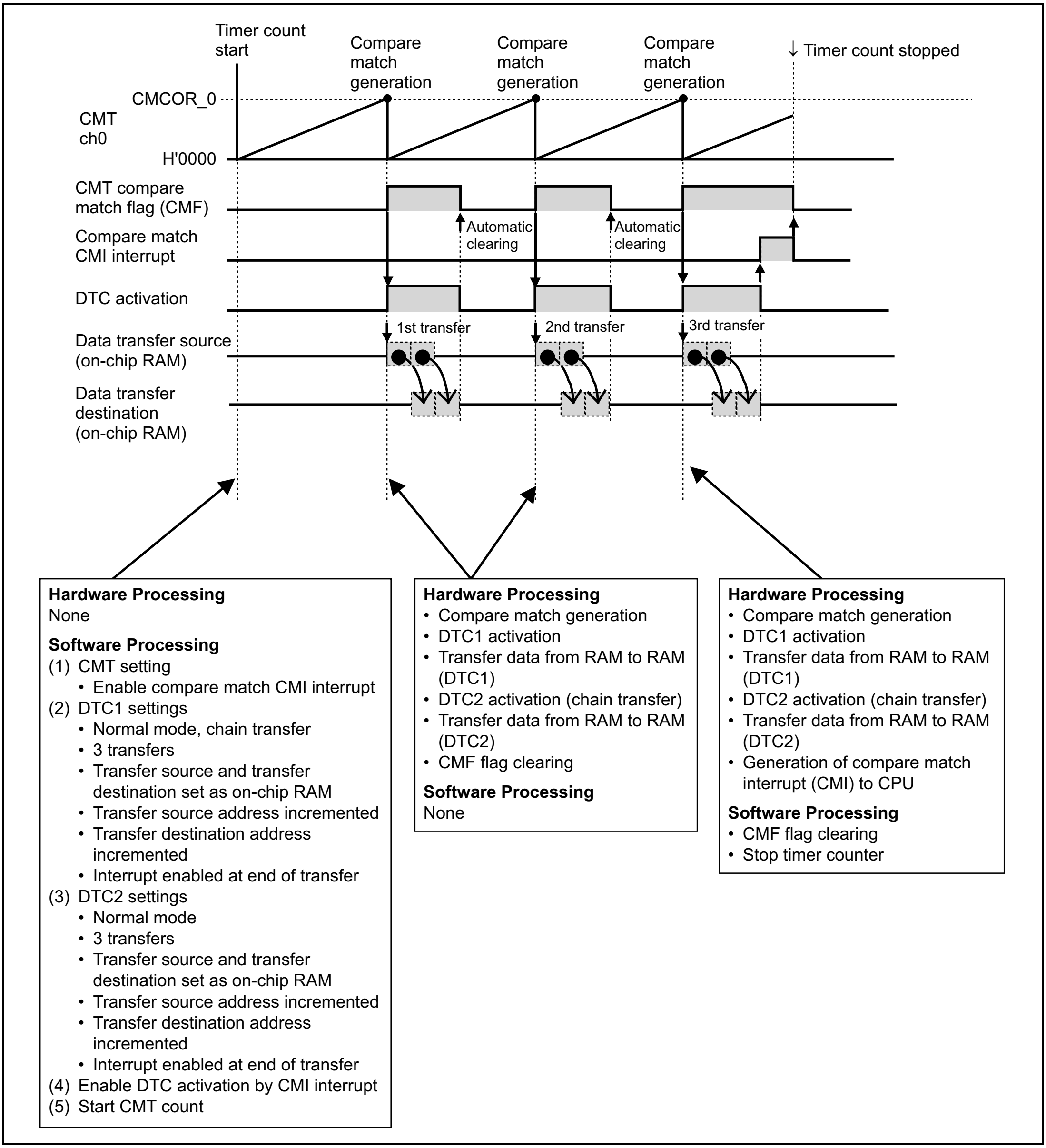
Table 2.23 Function Assignments

Function	Type	Function Assignment
DTMR	DTC	Sets DTC to normal mode, use of chain transfer
DTCRA	DTC	Setting of number of transfers
DTSAR	DTC	Transfer source address setting
DTDAR	DTC	Transfer destination address setting
DTBR	DTC	Setting of DTC vector upper 16 bits
DTER	DTC	Enables DTC activation by CMT ch0 CMI interrupt
CMSTR	CMT	CMT count start
CMCSR_0	CMT ch0	Count clock selection, interrupt control
CMCNT_0	CMT ch0	Counter
CMCOR_0	CMT ch0	Period setting

Operation

(1) The principles of operation of this sample task are shown in the figure below.

Data transfer from on-chip RAM to on-chip RAM is performed by the DTC by means of hardware and software processing as shown in the figure.



Hardware Processing
None

Software Processing
(1) CMT setting
- Enable compare match CMI interrupt

(2) DTC1 settings
- Normal mode, chain transfer
- 3 transfers
- Transfer source and transfer destination set as on-chip RAM
- Transfer source address incremented
- Transfer destination address incremented
- Interrupt enabled at end of transfer

(3) DTC2 settings
- Normal mode
- 3 transfers
- Transfer source and transfer destination set as on-chip RAM
- Transfer source address incremented
- Transfer destination address incremented
- Interrupt enabled at end of transfer

(4) Enable DTC activation by CMI interrupt
(5) Start CMT count

Hardware Processing
- Compare match generation
- DTC1 activation
- Transfer data from RAM to RAM (DTC1)
- DTC2 activation (chain transfer)
- Transfer data from RAM to RAM (DTC2)
- CMF flag clearing

Software Processing
None

Hardware Processing
- Compare match generation
- DTC1 activation
- Transfer data from RAM to RAM (DTC1)
- DTC2 activation (chain transfer)
- Transfer data from RAM to RAM (DTC2)
- Generation of compare match interrupt (CMI) to CPU

Software Processing
- CMF flag clearing
- Stop timer counter

Figure 2.23 Principles of Operation

(2) The principles of operation of DTC activation are shown in the figure below. When executing DTC transfer, the following settings should be made before an activation source occurs.

- Make DTC register information settings and place DTC register information in RAM.
- Set the lower 16 bits of the start address (32 bits) of DTC register information in the DTC vector table.
- Set the lower 16 bits of the start address (32 bits) of DTC register information in the DTC information base register.

The DTC is activated by the following processing.

- A DTC activation source interrupt is generated.
- The lower 16 bits of the start address of DTC register information are read from the address corresponding to the DTC vector table activation source.
- The upper 16 bits of the start address of DTC register information are read from the DTC information base register (DTMR).
- The 32-bit start address of DTC register information is generated from the read start address lower 16 bits and upper 16 bits.
- The start of DTC register information is read sequentially from the DTC register information start address, and data transfer is performed.

In this task, a CMT compare match interrupt is used as the DTC activation source.

The following table shows the register information configuration when using chain transfer in normal mode.

Table 2.24 DTC Register Information (Normal Mode, Chain Transfer)

Setting Address	Register Name	Data Length
RF	DTC mode register (DTMR)	Word (2 bytes)
RF+2	DTC transfer count register A (DTCRA)	Word (2 bytes)
RF+8	DTC source address register (DTSAR)	Longword (4 bytes)
RF+12	DTC destination address register (DTDAR)	Longword (4 bytes)

RF: DTC register information start address (in on-chip RAM)

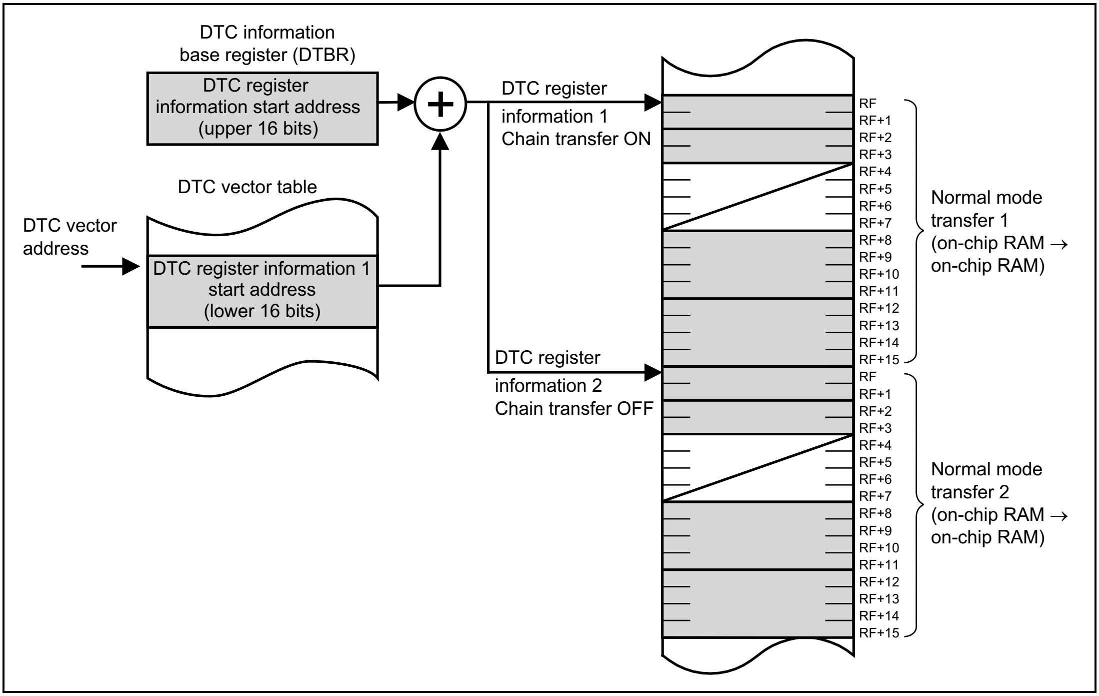

DTC information base register (DTBR)
DTC register information start address (upper 16 bits)
DTC vector table
DTC vector address
DTC register information 1 start address (lower 16 bits)
DTC register information 1 Chain transfer ON
DTC register information 2 Chain transfer OFF
RF
RF+1
RF+2
RF+3
RF+4
RF+5
RF+6
RF+7
RF+8
RF+9
RF+10
RF+11
RF+12
RF+13
RF+14
RF+15
Normal mode transfer 1 (on-chip RAM → on-chip RAM)
RF
RF+1
RF+2
RF+3
RF+4
RF+5
RF+6
RF+7
RF+8
RF+9
RF+10
RF+11
RF+12
RF+13
RF+14
RF+15
Normal mode transfer 2 (on-chip RAM → on-chip RAM)

Figure 2.24 Correspondence between DTC Vector Address and Transfer Information (Chain Transfer in Normal Mode)

Software

(1) Modules

The following table shows the modules used by this sample task.

Table 2.25 Modules

Module Name	Label	Function
Main routine	main	CMT timer setting, DTC initialization, timer start
CMI0 interrupt	cmt0_cmi0_dtc	CMT ch0 compare match interrupt (CMI0). Interrupt generation at end of specified number of DTC transfers

(2) Arguments

The following table shows the arguments used by this sample task.

Table 2.26 Arguments

Argument	Function	Module Name	Data Length	Input/ Output
S1_data [0] to [2]	DTC1 transfer source transfer data storage	Main routine	1 byte	Output
D1_data [0] to [2]	DTC1 transfer destination transfer data storage	Main routine	1 byte	Input
S2_data [0] to [2]	DTC2 transfer source transfer data storage	Main routine	1 byte	Output
D2_data [0] to [2]	DTC2 transfer destination transfer data storage	Main routine	1 byte	Input

(3) Internal Registers Used

The following table shows the internal registers used by this sample task.

Table 2.27 Internal Registers Used

Register Name		Function	Address	Set Value
	Bits		Bits	
P_STBY.MSTCR1	MSTP25 MSTP24	Module standby control register 1 DTC module standby control bits: When MSTP25 = MSTP24 = 0, module standby release Same value is set for MSTP25 and MSTP24	H'FFFF861C Bit 9 Bit 8	B'00
P_STBY.MSTCR2	MSTP12	Module standby control register 2 CMT module standby control bit: When MSTP12 = 0, module standby release	H'FFFF861E Bit 12	0
P_INTC.IPRG	CMT0	Interrupt priority register G (IPRG) CMT0 CMI0 interrupt priority level setting: When CMT0 = b'1010 (10), CMI0 interrupt is set to priority level 10	H'FFFF8354 Bits 7 to 4	10
P_CMT.CMSTR		Compare match timer start register (CMTSR) 16-bit register that selects CMCNT operation/stoppage	H'FFFF83D0	H'0001
	STR1	Counter start 1: When STR1 = b'0, TCNT_1 count operation is stopped	Bit 1	
	STR0	Counter start 0: When STR0 = b'1, TCNT_0 counts	Bit 0	

Register Name		Function	Address	Set Value
	Bits		Bits	
P_CMT.CMCSR_0		Compare match timer control/status register 0 (CMCSR_0) Compare match generation indication, interrupt setting, timer clock setting	H'FFFF83D2	H'0043
	CMF	Compare match flag: CMF is set to 1 when CMCNT and CMCOR values match	Bit 7	
	CMIE	Compare match interrupt enable: When CMIE = 1, compare match interrupt (CMI) is enabled	Bit 6	
	CKS1 CKS0	CMCNT counter clock selection: When CKS[1:0] = b'11, count is performed using internal clock Pϕ/512	Bit 1 Bit 0	
P_CMT.CMCNT_0		Compare match timer counter 0 (CMCNT_0) 16-bit register used as up-counter for generating interrupt requests	H'FFFF83D4	H'0000
P_CMT.CMCOR_0		Compare match timer constant register 0 (CMCOR_0) 16-bit register used to set CMCNT compare match period When CMCOR_0 = H'1e84, 100 ms compare match period is used (Pϕ/512 count, Pϕ = 40 MHz)	H'FFFF83D6	H'1e84

Register Name		Function	Address	Set Value
	Bits		Bits	
DTC_N1.DTMR		DTC mode register (DTMR) DTC operating mode control setting	Located in on-chip RAM	H'a040
	SM1 SM0	Source address mode: When SM[1:0] = b'10, DTSAR is incremented after transfer	Bit 15 Bit 14	
	DM1 DM0	Destination address mode: When DM[1:0] = b'10, DTDAR is incremented after transfer	Bit 13 Bit 12	
	MD1 MD0	DTC transfer mode: When MD[1:0] = b'00, normal mode	Bit 11 Bit 10	
	SZ1 SZ0	DTC data transfer size: When SZ[1:0] = b'00, byte (1-byte) transfer	Bit 9 Bit 8	
	DTS	DTC transfer mode select: When DTS = b'0, destination side is block area	Bit 7	
	CHNE	DTC chain transfer enable: When CHNE = b'1, chain transfer is enabled	Bit 6	
	DISEL	DTC interrupt select: When DISEL = b'0, interrupt request to CPU is generated only at end of specified data transfer	Bit 5	
	NMIM	DTC NMI mode: When NMIM = b'0, DTC transfer is suspended by NMI	Bit 4	
DTC_N1.DTCRA		DTC transfer count register A (DTCRA) Specifies number of transfers in DTC data transfer Set to 3 transfers	Located in on-chip RAM	H'0003
DTC_N1.DTSAR		DTC source address register (DTSAR) 32-bit register that specifies transfer source address of data to be transferred by DTC	Located in on-chip RAM	&S1_data[0]
DTC_N1.DTDAR		DTC destination address register (DTDAR) 32-bit register that specifies transfer destination address of data to be transferred by DTC	Located in on-chip RAM	&D1_data[0]

Register Name		Function	Address	Set Value
	Bits		Bits	
DTC_N2.DTMR		DTC mode register (DTMR) DTC operating mode control setting	Located in on-chip RAM	H'a000
	SM1 SM0	Source address mode: When SM[1:0] = b'10, DTSAR is incremented after transfer	Bit 15 Bit 14	
	DM1 DM0	Destination address mode: When DM[1:0] = b'10, DTDAR is incremented after transfer	Bit 13 Bit 12	
	MD1 MD0	DTC transfer mode: When MD[1:0] = b'00, normal mode	Bit 11 Bit 10	
	SZ1 SZ0	DTC data transfer size: When SZ[1:0] = b'00, byte (1-byte) transfer	Bit 9 Bit 8	
	DTS	DTC transfer mode select: When DTS = b'0, destination side is block area	Bit 7	
	CHNE	DTC chain transfer enable: When CHNE = b'1, chain transfer is enabled	Bit 6	
	DISEL	DTC interrupt select: When DISEL = b'0, interrupt request to CPU is generated only at end of specified data transfer	Bit 5	
	NMIM	DTC NMI mode: When NMIM = b'0, DTC transfer is suspended by NMI	Bit 4	
DTC_N2.DTCRA		DTC transfer count register A (DTCRA) Specifies number of transfers in DTC data transfer Set to 3 transfers	Located in on-chip RAM	H'0003
DTC_N2.DTSAR		DTC source address register (DTSAR) 32-bit register that specifies transfer source address of data to be transferred by DTC	Located in on-chip RAM	&S2_data[0]
DTC_N2.DTDAR		DTC destination address register (DTDAR) 32-bit register that specifies transfer destination address of data to be transferred by DTC	Located in on-chip RAM	&D2_data[0]

Register Name		Function	Address	Set Value
	Bits		Bits	
P_DTC.DTBR		DTC information base register (DTBR) Specifies upper 16 bits of memory address that stores DTC transfer information	H'FFFF8708	0xFFFF
P_DTC.DTED	CMI0	DTC enable register D (DTED) When set to 1, corresponding interrupt source is selected as DTC activation source: When CMI0 (DTED5) = b'1, CMT0 CMI0 interrupt is activation source	H'FFFF8703 Bit 5	1

(4) RAM Used

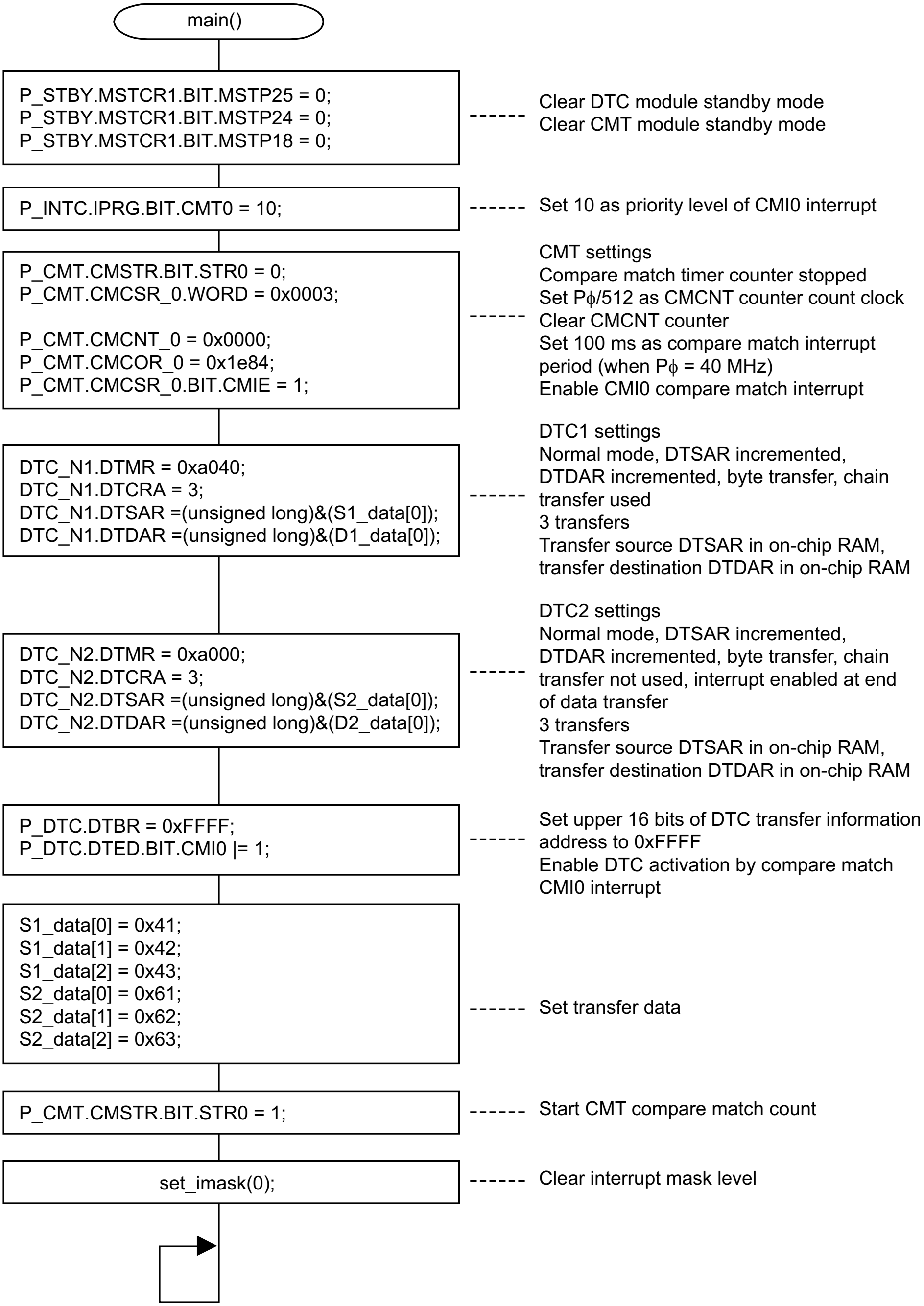
The following table shows the RAM used by this sample task.

Table 2.28 RAM Used

Label	Function	Address	Module Using RAM
S1_data	DTC1 transfer data storage Array storing 3-byte data	On-chip RAM	Main routine
D1_data	Data storage after DTC1 data transfer Array storing 3-byte data	On-chip RAM	Main routine
S2_data	DTC2 transfer data storage Array storing 3-byte data	On-chip RAM	Main routine
D2_data	Data storage after DTC2 data transfer Array storing 3-byte data	On-chip RAM	Main routine

Flowcharts

(a) Main processing



main()

```
P_STBY.MSTCR1.BIT.MSTP25 = 0;
P_STBY.MSTCR1.BIT.MSTP24 = 0;
P_STBY.MSTCR1.BIT.MSTP18 = 0;
```

Clear DTC module standby mode
Clear CMT module standby mode

```
P_INTC.IPRG.BIT.CMT0 = 10;
```

Set 10 as priority level of CMI0 interrupt

```
P_CMT.CMSTR.BIT.STR0 = 0;
P_CMT.CMCSR_0.WORD = 0x0003;

P_CMT.CMCNT_0 = 0x0000;
P_CMT.CMCOR_0 = 0x1e84;
P_CMT.CMCSR_0.BIT.CMIE = 1;
```

CMT settings
Compare match timer counter stopped
Set Pϕ/512 as CMCNT counter count clock
Clear CMCNT counter
Set 100 ms as compare match interrupt period (when Pϕ = 40 MHz)
Enable CMI0 compare match interrupt

```
DTC_N1.DTMR = 0xa040;
DTC_N1.DTCRA = 3;
DTC_N1.DTSAR =(unsigned long)&(S1_data[0]);
DTC_N1.DTDAR =(unsigned long)&(D1_data[0]);
```

DTC1 settings
Normal mode, DTSAR incremented, DTDAR incremented, byte transfer, chain transfer used
3 transfers
Transfer source DTSAR in on-chip RAM, transfer destination DTDAR in on-chip RAM

```
DTC_N2.DTMR = 0xa000;
DTC_N2.DTCRA = 3;
DTC_N2.DTSAR =(unsigned long)&(S2_data[0]);
DTC_N2.DTDAR =(unsigned long)&(D2_data[0]);
```

DTC2 settings
Normal mode, DTSAR incremented, DTDAR incremented, byte transfer, chain transfer not used, interrupt enabled at end of data transfer
3 transfers
Transfer source DTSAR in on-chip RAM, transfer destination DTDAR in on-chip RAM

```
P_DTC.DTBR = 0xFFFF;
P_DTC.DTED.BIT.CMI0 |= 1;
```

Set upper 16 bits of DTC transfer information address to 0xFFFF
Enable DTC activation by compare match CMI0 interrupt

```
S1_data[0] = 0x41;
S1_data[1] = 0x42;
S1_data[2] = 0x43;
S2_data[0] = 0x61;
S2_data[1] = 0x62;
S2_data[2] = 0x63;
```

Set transfer data

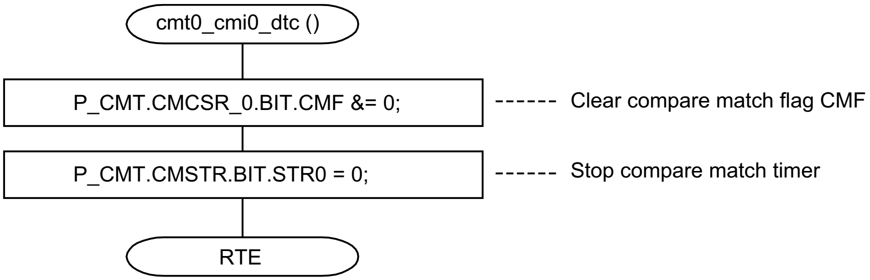
```
P_CMT.CMSTR.BIT.STR0 = 1;
```

Start CMT compare match count

```
set_imask(0);
```

Clear interrupt mask level

(b) Compare match interrupt handling


cmt0_cmi0_dtc ()
P_CMT.CMCSR_0.BIT.CMF &= 0;
------ Clear compare match flag CMF
P_CMT.CMSTR.BIT.STR0 = 0;
------ Stop compare match timer
RTE

Program Listing

```
/********************************************************/
/* SH7046F Series -SH7047- Application Note            */
/* Data transfer Controller(DTC)                        */
/*     Normal mode with Chain Transfer                  */
/* Function                                             */
/*  :Data transfer Controller(DTC)                      */
/*  :Compare Match Timer(CMT ch0)                       */
/*                                                      */
/* External input clock       :10MHz                    */
/* Internal CPU clock         :40MHz                    */
/* Internal peripheral clock  :40MHz                    */
/*                                                      */
/* Written     :     2002/3/1  Rev.1.0                  */
/********************************************************/

#include "iodefine.h"
#include <machine.h>


/*------------ Symbol Definition -----------------------------------------------*/
struct st_dtc_n{                    /* DTC Normal Transfer Mode information     */
    unsigned short DTMR;            /* DTC Mode Register                        */
    unsigned short DTCRA;           /* Transfer counter                         */
    unsigned short dummy1;          /* Reserved                                 */
    unsigned short dummy2;          /* Reserved                                 */
    unsigned long DTSAR;            /* source address register                  */
    unsigned long DTDAR;            /* destination address register             */
};

#define DTC_COUNT1  3               /* DTC Transmit count                       */
#define DTC_COUNT2  3               /* DTC Transmit count                       */
#define DTC_N1 (*(volatile struct st_dtc_n*)0xFFFFE000)
                                    /* DTC information address                  */
#define DTC_N2 (*(volatile struct st_dtc_n*)0xFFFFE010)
                                    /* DTC information address                  */


/*------------ Function Definition ---------------------------------------------*/
void main(void);
void cmt0_cmi0_dtc(void);


/*------------ RAM allocation Definition ---------------------------------------*/
unsigned char S1_data[DTC_COUNT1];           /* buffer memory                   */
unsigned char D1_data[DTC_COUNT1];           /* buffer memory                   */
unsigned char S2_data[DTC_COUNT2];           /* buffer memory                   */
unsigned char D2_data[DTC_COUNT2];           /* buffer memory                   */


/************************************************************/
/* main Program                                             */
/************************************************************/
void main( void )
{
    /* Set standby mode  */
    P_STBY.MSTCR1.BIT.MSTP25 = 0;     /* Disable DTC standby mode   */
    P_STBY.MSTCR1.BIT.MSTP24 = 0;     /* Disable DTC standby mode   */
    P_STBY.MSTCR2.BIT.MSTP12 = 0;     /* Disable CMT standby mode   */

    /* Set interrupt priority level (0 to 15)  */
    P_INTC.IPRG.BIT.CMT0 = 10;        /* CMT0 CMI0 interrupt level 10           */

    /* Initialize CMT0 for Interval timer   */
    P_CMT.CMSTR.BIT.STR0 = 0;       /* timer count stop                         */
    P_CMT.CMCSR_0.WORD = 0x0003;
             /* CMF=0;              clear compare match flag                    */
             /* CMIE=0;             compare match interrupt disable             */
             /* CKS[1:0]=b'11;      clock = peripheral clock(Pφ)/512            */
    P_CMT.CMCNT_0 = 0x0000;         /* timer counter clear                      */
    P_CMT.CMCOR_0 = 0x1e84;         /* 100ms clock=Pφ/512 Pφ=40MHz              */
    P_CMT.CMCSR_0.BIT.CMIE = 1;     /* compare match interrupt enable           */

    /* DTC1 information  */
    DTC_N1.DTMR = 0xa040;           /*                                          */
             /* SM[1:0]=b'10;       DTSAR is incremented                        */
             /* DM[1:0]=b'10;       DTDAR is incremented                        */
             /* MD[1:0]=b'00;       Normal transfer mode                        */
             /* SZ[1:0]=b'00;       byte-size transfer                          */
             /* DTS=0;              Source is block area                        */
             /* CHNE=1;             Chain transfer is enable                    */
             /* DISEL=0;            Interrupt->transfer ends                    */
             /* NMIM=0;             NMI->Terminate DTC transfer                 */
    DTC_N1.DTCRA = DTC_COUNT1;        /* DTC transfer Count  */
    DTC_N1.DTSAR =(unsigned long)&(S1_data[0]);    /* set source address        */
    DTC_N1.DTDAR =(unsigned long)&(D1_data[0]);    /* set destination address   */
    /* DTC2 information */
    DTC_N2.DTMR = 0xa000;           /*                                          */
             /* SM[1:0]=b'10;       DTSAR is incremented                        */
             /* DM[1:0]=b'10;       DTDAR is incremented                        */
             /* MD[1:0]=b'00;       Normal transfer mode                        */
             /* SZ[1:0]=b'00;       byte-size transfer                          */
             /* DTS=0;              Source is block area                        */
             /* CHNE=0;             Chain transfer is canceled                  */
             /* DISEL=0;            Interrupt->transfer ends                    */
             /* NMIM=0;             NMI->Terminate DTC transfer                 */
    DTC_N2.DTCRA = DTC_COUNT2;      /* DTC transfer Count                       */
    DTC_N2.DTSAR =(unsigned long)&(S2_data[0]);   /* set source address         */
    DTC_N2.DTDAR =(unsigned long)&(D2_data[0]);   /* set destination address    */

    P_DTC.DTBR = 0xFFFF;            /* DTC information base register            */
    P_DTC.DTED.BIT.CMI0 |= 1;       /* interrupt sources CMT ch0(CMI0)          */

    /*  set transmit data  */
    S1_data[0] = 0x41;
    S1_data[1] = 0x42;
    S1_data[2] = 0x43;
    S2_data[0] = 0x61;
    S2_data[1] = 0x62;
    S2_data[2] = 0x63;
    set_imask(0);                   /* clear interrupt mask level                */
    P_CMT.CMSTR.BIT.STR0 = 1;       /* CMT0 timer count start                    */
    while(1);
}
/************************************************************/
/* CMT0 Interrupt                                           */
/* Interval interrupt                                       */
/************************************************************/
#pragma interrupt(cmt0_cmi0_dtc)
void cmt0_cmi0_dtc(void)
{
    P_CMT.CMCSR_0.BIT.CMF &= 0;     /* Clear CMT0 compare match flag             */
    P_CMT.CMSTR.BIT.STR0 = 0;       /* CMT0 timer count stop                     */
}
```

2.5 Asynchronous Serial Data Simultaneous Transmission/Reception and DTC Data Transfer (SCI, DTC)

Asynchronous Serial Data Simultaneous Transmission/Reception and DTC Data Transfer (SCI, DTC)	Functions Used: SCI, DTC

Specifications

(1) Simultaneous 3-byte data transmit and receive operations are performed using the asynchronous serial transfer function and DTC data transfer function, as shown in figure 2.25.

(2) Serial transmit data transfer and storage of serial receive data in on-chip RAM are performed using the data transfer controller (DTC) transfer function as shown in figure 2.26.

(3) The transmit/receive data format is: 8-bit data length, even parity, 1-bit stop bit length. Communication is performed at a bit rate of 1 Mbps using the LSB-first method in which data is transmitted and received starting from the least significant bit.

(4) The DTC transfer conditions are shown in table 2.26.

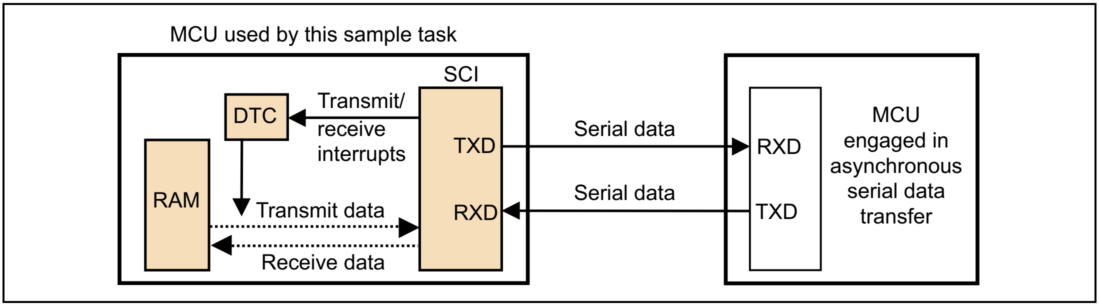


Figure 2.25 Asynchronous Serial Data Simultaneous Transmission/Reception

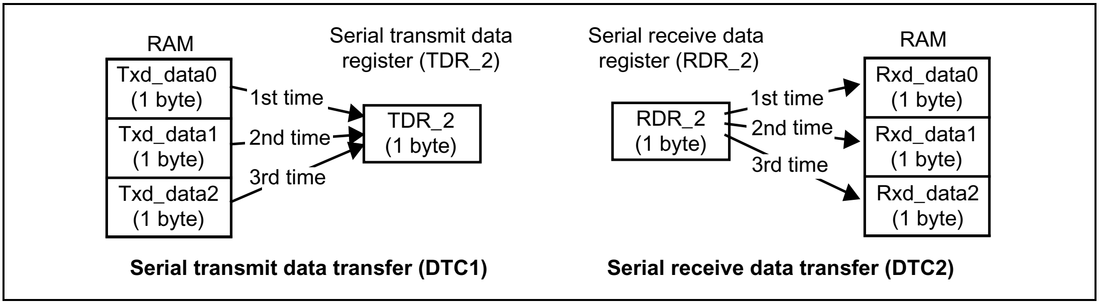


Figure 2.26 Data Transfer Using DTC

Table 2.29 DTC Transfer Conditions

Condition	Serial Transmission DTC Transfer Condition (DTC1)	Serial Reception DTC Transfer Condition (DTC2)
Transfer mode	Normal mode	Normal mode
Number of transfers	3	3
Transfer data size	Byte transfer	Byte transfer
Transfer source	On-chip RAM	Serial receive data register (RDR_2)
Transfer destination	Serial transmit data register (TDR_2)	On-chip RAM
Transfer source address	Transfer source address incremented after transfer	Transfer source address fixed
Transfer destination address	Transfer destination address fixed	Transfer destination address incremented after transfer
Activation source	Activated by SCI ch2 transmit interrupt (TXI_2)	Activated by SCI ch2 receive interrupt (RXI_2)
Interrupt handling	Interrupt to CPU enabled only at end of specified data transfer	Interrupt to CPU enabled only at end of specified data transfer

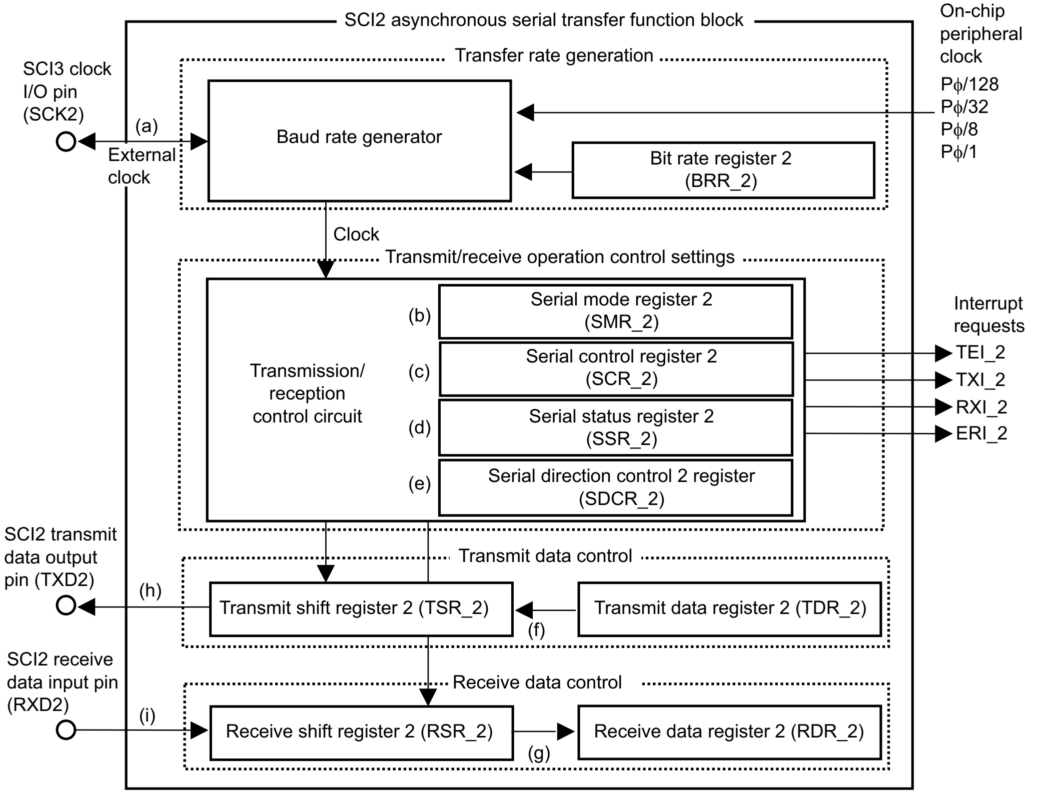
Functions Used

(1) This sample task performs simultaneous asynchronous serial data transmit/receive operations using the serial communication interface (SCI) and data transfer controller (DTC).

(a) A block diagram of simultaneous asynchronous serial data transmit/receive operations is shown in figure 2.27. The asynchronous serial data simultaneous transmission/reception block diagram is explained below.

- In asynchronous mode, serial data communication is carried out using the asynchronous method in which synchronization is implemented on a character-by-character basis. Using the asynchronous method, serial communication can be carried out with a standard asynchronous communication LSI such as a Universal Asynchronous Receiver/Transmitter (UART) or Asynchronous Communication Interface Adapter (ACIA). A function for serial communication between a number of processors (multiprocessor communication function) is also provided in asynchronous mode.
- On-chip peripheral clock Pϕ is the reference clock for operating on-chip peripheral functions.
- Receive shift register 2 (RSR_2) is used to receive serial data. Serial data input to RSR_2 from the RXD2 pin is set in the order of reception, starting from the LSB (bit 0), and converted to parallel data. When one byte of data is received, it is automatically transferred to RDR_2. RSR_2 cannot be directly read or written to by the CPU.
- Receive data register 2 (RDR_2) is an 8-bit register that stores received serial data. When reception of one byte of data is completed, the received data is transferred from RSR_2 to RDR_2, and the receive operation is terminated. RSR_2 then becomes able to receive. RSR_2 and RDR_2 are double-buffered, allowing continuous receive operations. RDR_2 is a receive-only register, and cannot be written to by the CPU.
- Transmit shift register 2 (TSR_2) is used to transmit serial data. Transmit data from TDR_2 is first transferred to TSR_2, and sent to the TXD pin in order starting from the LSB (bit 0) to implement serial data transmission. When one byte of data has been transmitted, the next transmitted is automatically transferred from TDR_2 to TSR_2, and transmission is started. However, data transfer from TDR_2 to TSR_2 is not performed if data has not been written to TDR_2 (if TDRE is set to 1). TSR_2 cannot be directly read or written to by the CPU.
- Transmit data register 2 (TDR_2) is an 8-bit register that stores transmit data. When the TSR_2 “empty” state is detected, transmit data written to TDR_2 is transferred to TSR_2 and serial data transmission is started. Writing the next transmit data to TDR_2 during TSR_2 serial data transmission enables continuous transmission to be performed. TDR_2 can be read or written to by the CPU at any time.
- Serial mode register 2 (SMR_2) is an 8-bit register for setting the serial data communication format and selecting the clock source of the on-chip baud rate generator.

- Serial control register 2 (SCR_2) is an 8-bit register that performs transmit/receive operation and transmit/receive clock source selection.
- Serial status register 2 (SSR_2) contains SCI2 status flags and a transmit/receive multiprocessor bit. TDRE, RDRF, OER, PER, and FER can only be cleared.
- The serial direction control 2 register (SDCR_2) performs LSB-first/MSB-first selection by means of the DIR bit. In the case of an 8-bit length, LSB-first or MSB-first can be selected regardless of the serial communication mode. In the case of a 7-bit length, LSB-first must be selected.
- Bit rate register 2 (BRR_2) is an 8-bit register for adjusting the bit rate.


SCI2 asynchronous serial transfer function block
On-chip peripheral clock
Pϕ/128
Pϕ/32
Pϕ/8
Pϕ/1
SCI3 clock I/O pin (SCK2)
(a)
External clock
Transfer rate generation
Baud rate generator
Bit rate register 2 (BRR_2)
Clock
Transmit/receive operation control settings
Transmission/ reception control circuit
(b)
Serial mode register 2 (SMR_2)
(c)
Serial control register 2 (SCR_2)
(d)
Serial status register 2 (SSR_2)
(e)
Serial direction control 2 register (SDCR_2)
Interrupt requests
TEI_2
TXI_2
RXI_2
ERI_2
SCI2 transmit data output pin (TXD2)
(h)
Transmit data control
Transmit shift register 2 (TSR_2)
Transmit data register 2 (TDR_2)
(f)
SCI2 receive data input pin (RXD2)
(i)
Receive data control
Receive shift register 2 (RSR_2)
Receive data register 2 (RDR_2)
(g)

Notes:
(a) Outputs the clock.
(b) Performs serial data communication format setting and baud rate generator clock source selection.
(c) Performs transmit/receive operation, clock source, and SCK pin function selection.
(d) Indicates the SCI2 operation status by means of status flags (transmit data register empty, receive data register full, overrun error, framing error, parity error).
(e) Performs LSB-first/MSB-first selection.
(f) Transmit data written to TDR_2 is transferred to TSR_2 on detection of a TSR_2 "empty" state.
(g) On completion of reception of 1 byte of data, received data is transferred from RSR_2 to RDR_2.

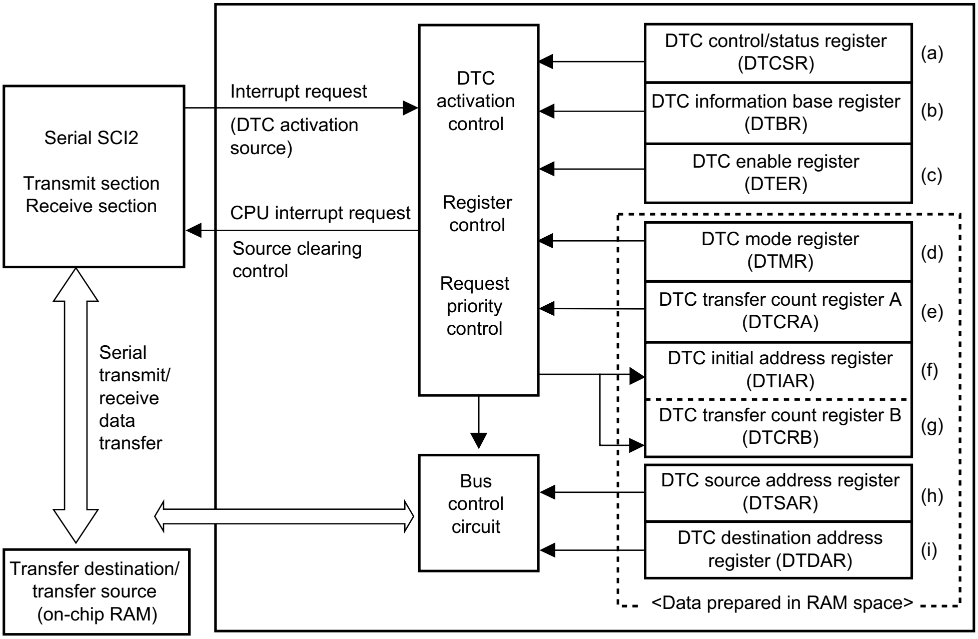
Figure 2.27 Asynchronous Serial Data Transmission/Reception Block Diagram

(b) A block diagram of the DTC is shown below. Of the three DTC transfer modes — normal mode, repeat mode, and block transfer mode —this sample task uses normal mode to perform transfer of serial transmit/receive data. DTC data transfer is performed using the serial transmission TXI interrupt and serial reception RXI interrupt as DTC activation sources. The block diagram is explained below.

- The DTC mode register (DTMR) is a 16-bit register that controls the DTC's operating mode.
- The DTC source address register (DTSAR) is a 32-bit register that specifies the transfer source address of data to be transferred by the DTC. An even address should be specified in the case of word size transfer, and an address that is a multiple of 4 in the case of longword transfer.
- The DTC destination address register (DTDAR) is a 32-bit register that specifies the transfer destination address of data to be transferred by the DTC. An even address should be specified in the case of word size transfer, and an address that is a multiple of 4 in the case of longword transfer.
- The DTC initial address register (DTIAR) is a 32-bit register that specifies the transfer source/transfer destination initial address in repeat mode. In repeat mode, when the DTS bit is 1, specify the initial transfer source address in the repeat area, and when the DTS bit is 0, specify the initial transfer destination address in the repeat area.
- DTC transfer count register A (DTCRA) is a 16-bit register that specifies the number of transfers in DTC data transfer. In normal mode, DTCRA functions as a transfer counter (1 to 65,536). In repeat mode, upper 8-bit DTCRAH holds the number of transfers, and lower 8-bit DTCRAL functions as an 8-bit transfer counter. In block transfer mode, DTCRA functions as a 16-bit transfer counter.
- DTC transfer count register B (DTCRB) is a 16-bit register that specifies the block length in block transfer mode.
- The DTC enable register (DTER) is used to select the interrupt source that activates the DTC, and comprises registers DTEA through DTEF.
- The DTC control/status register (DTCSR) is a 16-bit register that sets enabling/disabling of DTC activation by software, and sets a software activation DTC vector address. DTCSR also shows the DTC transfer status.
- The DTC information base register (DTBR) is a readable/writable 16-bit register that specifies the upper 16 bits of the memory address that stores DTC transfer information. Word or longword access must be used for DTBR. If byte access is used, the register contents will be undefined in the case of a write, and an undefined value will be returned in the case of a read.
- Information of six registers — the DTC mode register (DTMR), DTC source address register (DTSAR), DTC destination address register (DTDAR), DTC initial address register (DTIAR), DTC transfer count register A (DTCRA), and DTC transfer count register B (DTCRB) — cannot be accessed directly from the CPU. When a DTC activation source occurs, the relevant register information is transferred to these

registers from information of an arbitrary set of registers located in on-chip RAM and DTC transfer is performed, and when transfer ends, the contents of these registers are returned to RAM. Therefore, register information should be prepared in arbitrary on-chip RAM in the user program.

- This task uses DTC normal mode for both serial transmit data transfer and serial receive data transfer. Two sets of normal mode register information (DTMR, DTSAR, DTDAR, DTCRA, and DTCRB) are provided, for serial transmission use and for serial reception use.


Serial SCI2
Transmit section
Receive section
Interrupt request
(DTC activation source)
CPU interrupt request
Source clearing control
DTC activation control
Register control
Request priority control
Bus control circuit
Serial transmit/ receive data transfer
Transfer destination/ transfer source (on-chip RAM)
DTC control/status register (DTCSR)
(a)
DTC information base register (DTBR)
(b)
DTC enable register (DTER)
(c)
DTC mode register (DTMR)
(d)
DTC transfer count register A (DTCRA)
(e)
DTC initial address register (DTIAR)
(f)
DTC transfer count register B (DTCRB)
(g)
DTC source address register (DTSAR)
(h)
DTC destination address register (DTDAR)
(i)
<Data prepared in RAM space>

Notes:
(a) Performs enabling/disabling of DTC activation by software, and software activation DTC vector address setting.
(b) Performs specification of the upper 16 bits of the memory address that stores DTC transfer information.
(c) Selects the interrupt source that activates the DTC; comprises six registers, DTEA through DTEF.
(d) Performs DTC operating mode setting.
(e) Specifies the number of transfers in DTC data transfer.
(f) In repeat mode, specifies the transfer source/transfer destination initial address in repeat mode. Not used in normal mode. In block transfer mode, functions as the DTCRB register.
(g) In block transfer mode, specifies the block length. Not used in normal mode. In repeat mode, functions as the DTIAR register.
(h) Specifies the transfer source address of data to be transferred by the DTC.
(i) Specifies the transfer destination address of data to be transferred by the DTC.

Figure 2.28 DTC Block Diagram

(2) Table 2.30 shows the function assignments used in this sample task.

Table 2.30 Function Assignments

Function	Type	Function Assignment
TXD2	Pin	Channel 2 transmit data output pin
RXD2	Pin	Channel 2 receive data input pin
SMR_2	SCI2	Communication format setting; set to asynchronous mode
SCR_2	SCI2	Enables transmit/receive operation, interrupts
SSR_2	SCI2	Status flags indicating SCI2 operation status
SDCR_2	SCI2	Set to LSB-first transfer
BBR_2	SCI2	Sets transmit/receive bit rate
TSR_2	SCI2	Register for transmitting serial data
TDR_2	SCI2	Register that stores transmit data
RSR_2	SCI2	Register for receiving serial data
RDR_2	SCI2	Register that stores receive data
DTMR	DTC	Sets DTC to normal transfer mode
DTCRA	DTC	Setting of number of transfers
DTSAR	DTC	Transfer source address setting
DTDAR	DTC	Transfer destination address setting
DTBR	DTC	Setting of upper 16 bits of DTC vector
DTER	DTC	Enables DTC activation in serial reception/serial transmission

Operation

(1) The principles of operation of this sample task are shown in the figure below.

Simultaneous transmission/reception of asynchronous serial data is performed by means of hardware and software processing as shown in the figure.

(a) Transmit processing

- 3-byte data is transmitted by asynchronous serial communication.
- 3-byte transmit data is transferred from on-chip RAM to the SCI using the DTC.
- The serial TXI_2 interrupt is used for DTC activation.

(b) Receive processing

- 3-byte data is received by asynchronous serial communication.
- 3-byte receive data is transferred from the SCI to on-chip RAM using the DTC.
- The serial RXI_2 interrupt is used for DTC activation.

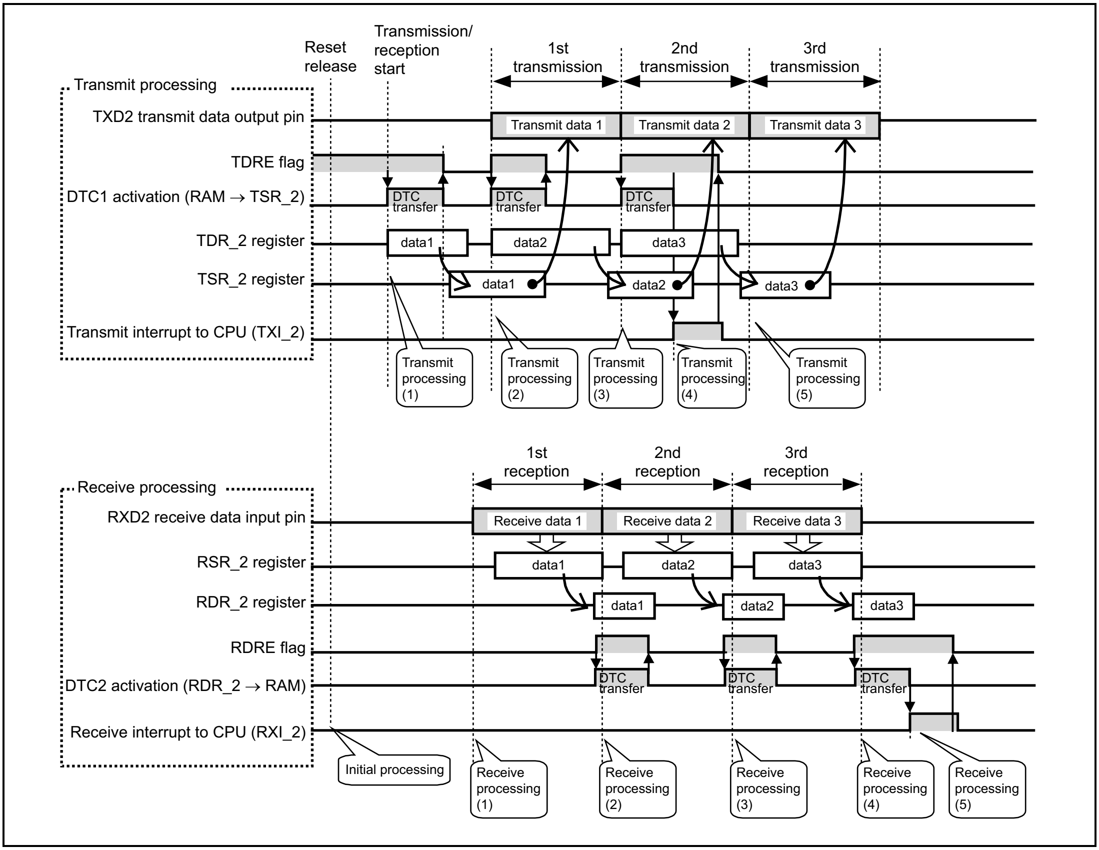


Figure 2.29 Principles of Operation

The figure is explained below.

Initial processing	Software Processing (1) SCI2 settings • Set asynchronous mode, LSB-first transfer • Enable transmit interrupts, receive interrupts, transmit operation, receive operation (2) DTC settings • Set normal transfer mode • Enable DTC1 activation by serial transmit interrupt (TXI_2), DTC2 activation by serial receive interrupt (RXI_2) • Initial processing		
Transmit processing (1)	Hardware Processing • DTC1 activation by TXI_2 interrupt (1st time) • Transfer transmit data 1 from on-chip RAM to TDR_2 register (DTC1) • Clear TDRE (DTC1)	Receive processing (1)	Hardware Processing • Reception start Capture receive data 1 in RSR register
	Software Processing None		Software Processing None
Transmit processing (2)	Hardware Processing • When TDRE flag is 0, transfer transmit data 1 from TDR_2 to TSR_2 register (SCI2) • Set TDRE flag to 1 (SCI2) • Start transmission (SCI2) • DTC1 activation by TXI_2 interrupt (2nd time) • Transfer transmit data 2 from on-chip RAM to TDR_2 (DTC1) • Clear TDRE flag (DTC1)	Receive processing (2)	Hardware Processing • Transfer receive data 1 from RSR to RDR register (SCI2) • Set RDRE flag to 1 (SCI2) • Start reception of next frame (SCI2) • Capture receive data 2 in RSR register (SCI2) • DTC2 activation by RXI_2 interrupt (1st time) • Transfer receive data 1 from RDR_2 to on-chip RAM (DTC2) • Clear RDRE flag (DTC2)
	Software Processing None		Software Processing None

Transmit processing (3)	Hardware Processing • When last bit is sent, check TDRE flag (SCI2) • When TDRE flag is 0, transfer transmit data 2 from TDR_2 to TSR_2 register (SCI2) • Set TDRE flag to 1 (SCI2) • Start transmission of next frame (SCI2) • DTC1 activation by TXI_2 interrupt (3rd time) • Transfer transmit data 3 from on-chip RAM to TDR_2 and terminate (DTC1); TDRE flag is not cleared	Receive processing (3)	Hardware Processing • Transfer receive data 2 from RSR to RDR register (SCI2) • Set RDRE flag to 1 (SCI2) • Start reception of next frame (SCI2) • Capture receive data 3 in RSR register • DTC2 activation by RXI_2 interrupt (2nd time) • Transfer receive data 2 from RDR_2 to on-chip RAM (DTC2) • Clear RDRE flag (DTC2)
	Software Processing None		Software Processing None
Transmit processing (4)	Hardware Processing • Generate TXI_2 interrupt to CPU	Receive processing (4)	Hardware Processing • Transfer receive data 3 from RSR to RDR register (SCI2) • Set RDRE flag to 1 (SCI2) • DTC2 activation by RXI_2 interrupt (2nd time) • Transfer receive data 2 from RDR_2 to on-chip RAM (DTC2) • Clear RDRE flag (DTC2)
	Software Processing • Clear TDRE flag • Disable TXI_2 interrupt		Software Processing None
Transmit processing (5)	Hardware Processing • When last bit is sent, check TDRE flag (SCI2) • When TDRE flag is 0, transfer transmit data 3 from TDR_2 to TSR_2 register (SCI2) • Set TDRE flag to 1 (SCI2) • Start transmission of last frame (SCI2)	Receive processing (5)	Hardware Processing • Generate TXI_2 interrupt to CPU
	Software Processing None		Software Processing • Clear TDRE flag • Disable TXI_2 interrupt

(2) The principles of operation of DTC activation are shown in the figure below. When executing DTC transfer, the following settings should be made before an activation source occurs.

- Make DTC register information settings and place DTC register information in RAM.
- Set the lower 16 bits of the start address (32 bits) of DTC register information in the DTC vector table.
- Set the lower 16 bits of the start address (32 bits) of DTC register information in the DTC information base register.

The DTC is activated by the following processing.

- A DTC activation source interrupt is generated.
- The lower 16 bits of the start address of DTC register information are read from the address corresponding to the DTC vector table activation source.
- The upper 16 bits of the start address of DTC register information are read from the DTC information base register (DTMR).
- The 32-bit start address of DTC register information is generated from the read start address lower 16 bits and upper 16 bits.
- The start of DTC register information is read sequentially from the DTC register information start address, and data transfer is performed.

In this task, a TXI_2 interrupt is used as the DTC activation source in serial transmission data transfer, and an RXI_2 interrupt in serial reception data transfer

The following table shows the register information configuration in normal mode.

Table 2.31 DTC Register Information (Normal Mode)

Setting Address	Register Name	Data Length
RF	DTC mode register (DTMR)	Word (2 bytes)
RF+2	DTC transfer count register A (DTCRA)	Word (2 bytes)
RF+8	DTC source address register (DTSAR)	Longword (4 bytes)
RF+12	DTC destination address register (DTDAR)	Longword (4 bytes)

RF: DTC register information start address (in on-chip RAM)

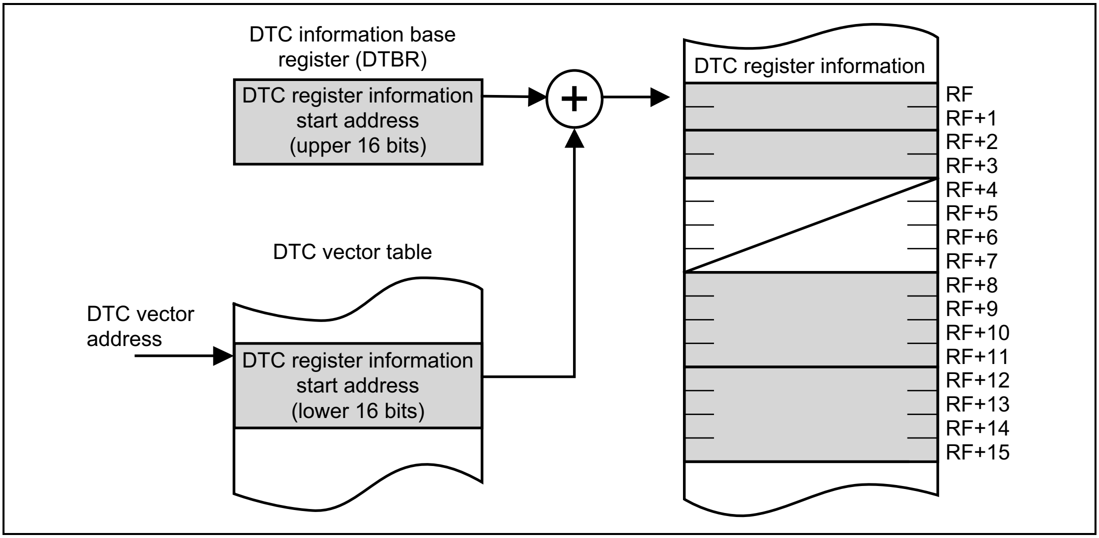

DTC information base register (DTBR)
DTC register information start address (upper 16 bits)
DTC vector table
DTC vector address
DTC register information start address (lower 16 bits)
DTC register information
RF
RF+1
RF+2
RF+3
RF+4
RF+5
RF+6
RF+7
RF+8
RF+9
RF+10
RF+11
RF+12
RF+13
RF+14
RF+15

Figure 2.30 Correspondence between DTC Vector Address and Transfer Information

Software

(1) Modules

The following table shows the modules used by this sample task.

Table 2.32 Modules

Module Name	Label	Function
Main routine	main	SCI ch2 asynchronous serial communication and DTC initialization, serial communication start
SCI transmit end interrupt	txi2_end	SCI ch2 transmit end interrupt. Interrupt generation at end of specified number of DTC transfers
SCI receive end interrupt	rxi2_end	SCI ch2 receive end interrupt. Interrupt generation at end of specified number of DTC transfers

(2) Arguments

The following table shows the arguments used by this sample task.

Table 2.33 Arguments

Argument	Function	Module Name	Data Length	Input/ Output
Txd_data[0] to [2]	Asynchronous serial transmit data storage	Main routine	1 byte	Output
Rxd_data[0] to [2]	Asynchronous serial receive data storage	Main routine	1 byte	Input

(3) Internal Registers Used

The following table shows the internal registers used by this sample task.

Table 2.34 Internal Registers Used

Register Name		Function	Address	Set Value
	Bits		Bits	
P_STBY.MSTCR1	MSTP25 MSTP24	Module standby control register 1 DTC module standby control bits: When MSTP25 = MSTP24 = 0, module standby release Same value is set for MSTP25 and MSTP24	H'FFFF861C Bit 9 Bit 8	B'00
	MSTP18	Module standby control register 2 Serial Communication Interface 2 standby control bit: When MSTP18 = 0, module standby release	H'FFFF861C Bit 2	0
P_INTC.IPRI	SCI2	Interrupt priority register I (IPRI) Interrupt priority level setting of SCI2 interrupts (ERI, RXI, TXI, TEI): When SCI2 = b'1010 (10), interrupts are set to priority level 10	H'FFFF835C Bits 12 to 15	10

Register Name		Function	Address	Set Value
	Bits		Bits	
DTC_1.DTMR		DTC mode register (DTMR) DTC operating mode control setting. For serial transmission use	Located in on-chip RAM	H'8000
	SM1 SM0	Source address mode: When SM[1:0] = b'10, DTSAR is incremented after transfer	Bit 15 Bit 14	
	DM1 DM0	Destination address mode: When DM[1:0] = b'00, DTDAR is fixed	Bit 13 Bit 12	
	MD1 MD0	DTC transfer mode: When MD[1:0] = b'00, normal transfer mode	Bit 11 Bit 10	
	SZ1 SZ0	DTC data transfer size: When SZ[1:0] = b'00, byte (1-byte) transfer	Bit 9 Bit 8	
	DTS	DTC transfer mode select: When DTS = b'0, destination side is block area Not used in normal mode	Bit 7	
	CHNE	DTC chain transfer enable: When CHNE = b'0, chain transfer is cleared	Bit 6	
	DISEL	DTC interrupt select: When DISEL = b'0, interrupt request to CPU is generated only at end of specified data transfer	Bit 5	
	NMIM	DTC NMI mode: When NMIM = b'0, DTC transfer is suspended by NMI	Bit 4	
DTC_1.DTCRA		DTC transfer count register A (DTCRA) Specifies number of transfers in DTC data transfer Set to 3 transfers	Located in on-chip RAM	H'0003
DTC_1.DTSAR		DTC source address register (DTSAR) 32-bit register that specifies transfer source address of data to be transferred by DTC Set to start address of transmit data storage area	Located in on-chip RAM	Txd_data
DTC_1.DTDAR		DTC destination address register (DTDAR) 32-bit register that specifies transfer destination address of data to be transferred by DTC Set to serial transmit data register (TDR_2)	Located in on-chip RAM	&P_SCI2.TDR

Register Name		Function	Address	Set Value
	Bits		Bits	
DTC_2.DTMR		DTC mode register (DTMR) DTC operating mode control setting. For serial reception use	Located in on-chip RAM	H'2000
	SM1 SM0	Source address mode: When SM[1:0] = b'00, DTSAR is fixed	Bit 15 Bit 14	
	DM1 DM0	Destination address mode: When DM[1:0] = b'10, DTDAR is incremented after transfer	Bit 13 Bit 12	
	MD1 MD0	DTC transfer mode: When MD[1:0] = b'00, normal transfer mode	Bit 11 Bit 10	
	SZ1 SZ0	DTC data transfer size: When SZ[1:0] = b'00, byte (1-byte) transfer	Bit 9 Bit 8	
	DTS	DTC transfer mode select: When DTS = b'0, destination side is block area Not used in normal mode	Bit 7	
	CHNE	DTC chain transfer enable: When CHNE = b'0, chain transfer is cleared	Bit 6	
	DISEL	DTC interrupt select: When DISEL = b'0, interrupt request to CPU is generated only at end of specified data transfer	Bit 5	
	NMIM	DTC NMI mode: When NMIM = b'0, DTC transfer is suspended by NMI	Bit 4	
DTC_2.DTCRA		DTC transfer count register A (DTCRA) Specifies number of transfers in DTC data transfer Set to 3 transfers	Located in on-chip RAM	H'03
DTC_2.DTSAR		DTC source address register (DTSAR) 32-bit register that specifies transfer source address of data to be transferred by DTC Set to serial receive data register (RDR_2)	Located in on-chip RAM	&P_SCI2.RDR
DTC_2.DTDAR		DTC destination address register (DTDAR) 32-bit register that specifies transfer destination address of data to be transferred by DTC Set to start address of receive data storage area	Located in on-chip RAM	Rxd_data
P_DTC.DTBR		DTC information base register (DTBR) Specifies upper 16 bits of memory address that stores DTC transfer information	H'FFFF8708	0xFFFF

Register Name		Function	Address	Set Value
	Bits		Bits	
P_DTC.DTEE	TXI_2	DTC enable register E (DTEE) When TXI_2 (DTEE2) = b'1, SCI2 transmit end interrupt (TXI_2) is activation source	H'FFFF8710 Bit 2	1
	RXI_2	DTC enable register E (DTEE) When RXI_2 (DTEE3) = b'1, SCI2 receive end interrupt (RXI_2) is activation source	H'FFFF8710 Bit 3	1
P_SCI2.SCR.BYTE		Serial control register 2 (SCR_2) Transmission/reception control, interrupt control, transmit/receive clock source selection	H'FFFF81C2	H'f0
	TIE	Transmit interrupt enable: When TIE = 1, TXI interrupt requests are enabled	Bit 7	
	RIE	Receive interrupt enable: When RIE = 1, RXI and ERI interrupt requests are enabled	Bit 6	
	TE	Transmit enable: When TE = 1, transmit operation is enabled	Bit 5	
	RE	Receive enable: When RE = 1, receive operation is enabled	Bit 4	
	MPIE	Multiprocessor interrupt enable (Only valid in asynchronous mode when MP = 1 in SMR) In this task, MP = 0, so setting is invalid	Bit 3	
	TEIE	Transmit end interrupt enable: When TEIE = 0, TEI interrupt requests are disabled	Bit 2	
	CKE1 CKE0	Clock enable 1-0 Clock source and SCK pin function selection: When CKEI[1:0] = b'00, clock source is internal clock, and SCK pin is input pin (input pin ignored)	Bit 1 Bit 0	

Register Name		Function	Address	Set Value
	Bits		Bits	
P_SCI2.SMR.BYTE		Serial mode register 2 (SMR_2) Communication format and on-chip baud rate generator clock selection	H'FFFF81C0	H'20
	C/A	Communication mode: When C/A = 0, operation in asynchronous mode	Bit 7	
	CHR	Character length (valid only in asynchronous mode): When CHR = 0, transmission/reception using 8-bit data length	Bit 6	
	PE	Parity enable (valid only in asynchronous mode): When PE = 1, parity bit is added when transmitting and parity is checked when receiving	Bit 5	
	O/E	Parity mode (valid only in asynchronous mode when PE = 1): When O/E = 0, transmission/reception using even parity	Bit 4	
	STOP	Stop bit length (valid only in asynchronous mode) Selects stop bit length when transmitting: When STOP = 0, 1 stop bit	Bit 3	
	MP	Multiprocessor mode (valid only in asynchronous mode) When MP = 0, multiprocessor communication function is disabled	Bit 2	
	CKS1 CKS0	Clock select 1-0 Selection of on-chip baud rate generator clock source: When CKS[1:0] = b'00, set to Pϕ/1 clock (n = 0)	Bit 1 Bit 0	
P_SCI2.SDCR	DIR	Serial direction control register 2 (SDCR_2) Data transfer direction: When DIR = 0, TDR contents are transmitted LSB-first, and receive data is stored in RDR LSB-first	H'FFFF81C6 Bit 3	1
P_SCI2.BRR		Bit rate register 2 (BRR_2): When BRR_2 = 21, bit rate is approx. 57600 bps (when clock source = Pϕ/1, Pϕ = 40 MHz)	H'FFFF81C1	9
P_SCI2.TDR		Transmit data register 2 (TDR_2) 8-bit register that stores transmit data	H'FFFF81C3	
P_SCI2.RDR		Receive data register 2 (RDR_2) 8-bit register that stores receive data	H'FFFF81C5	

Register Name		Function	Address	Set Value
	Bits		Bits	
P_SCI2.SSR	TDRE	Serial status register 2 (SSR_2) Transmit data register empty flag	H'FFFF81C4	1
	RDRF	Serial status register 2 (SSR_2) Receive data register full flag	H'FFFF81C4	0
	ORER	Serial status register 2 (SSR_2) Overrun error flag	H'FFFF81C4	0
	FER	Serial status register 2 (SSR_2) Framing error flag	H'FFFF81C4	0
	PER	Serial status register 2 (SSR_2) Parity error flag	H'FFFF81C4	0
P_PORTA.PACRL3	PA0MD2	Port A control register L3 Port A control register L2 PA0 mode bits, PA0/A0/POE0/RXD2 pin function selection: When (PA0MD2, PA0MD[1], PA0MD[0] = b'110, pin function is set to RXD2 input (SCI)	H'FFFF838A Bit 0	b'1
P_PORTA.PACRL2	PA0MD[1] PA0MD[0]		H'FFFF838E Bit 1 Bit 0	b'10
P_PORTA.PACRL3	PA1MD2	Port A control register L3 Port A control register L2 PA1 mode bits, PA1/A0/POE1/TXD2 pin function selection: When (PA1MD2, PA1MD[1], PA1MD[0] = b'110, pin function is set to TXD2 output (SCI)	H'FFFF838A Bit 1	b'1
P_PORTA.PACRL2	PA1MD[1] PA1MD[0]		H'FFFF838E Bit 3 Bit 2	b'10

(4) RAM Used

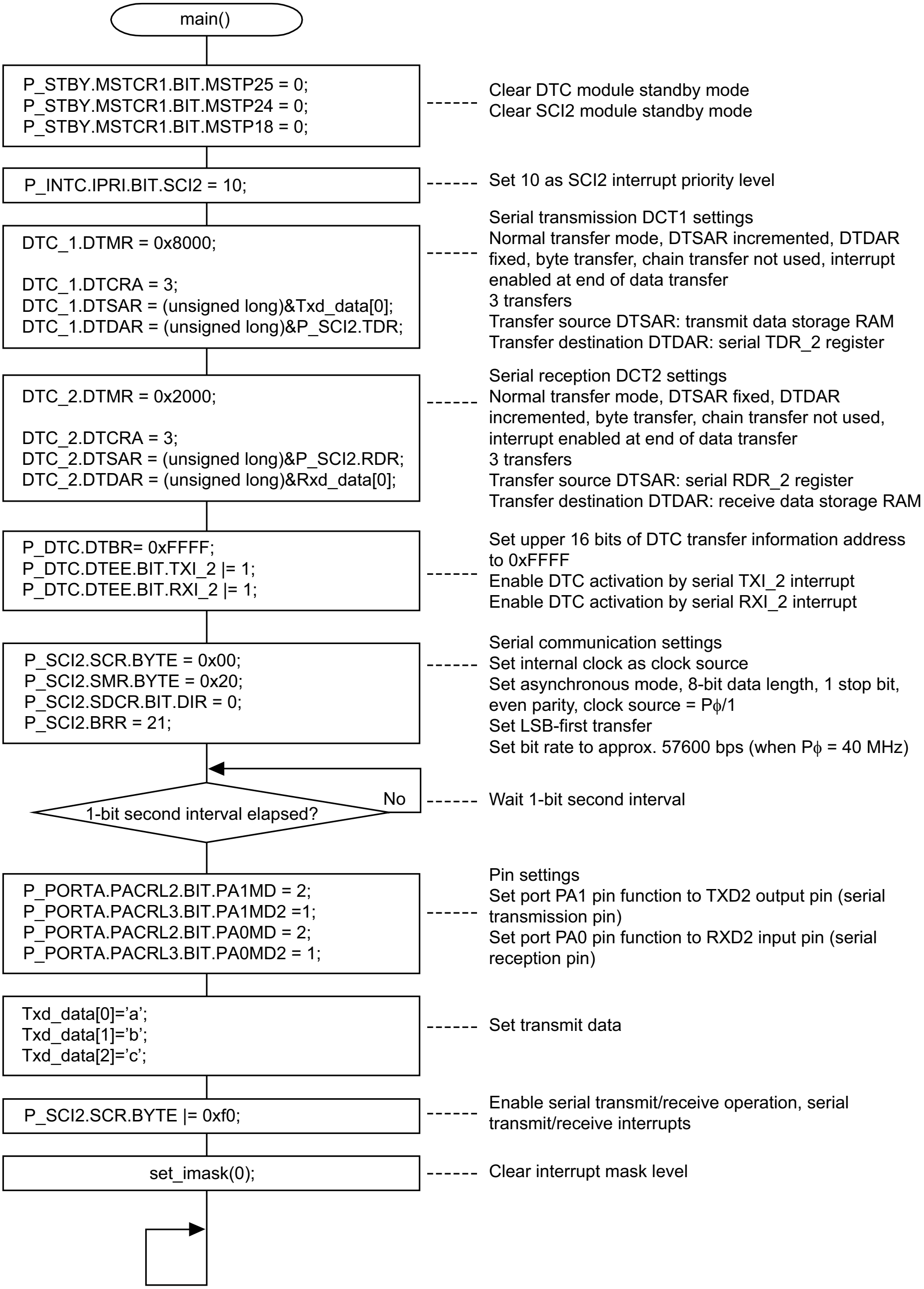
The following table shows the RAM used by this sample task.

Table 2.35 RAM Used

Label	Function	Address	Module Using RAM
Txd_data[0]	Stores 1st byte of asynchronous serial transmit data	On-chip RAM	Main routine
Txd_data[1]	Stores 2nd byte of asynchronous serial transmit data	On-chip RAM	Main routine
Txd_data[2]	Stores 3rd byte of asynchronous serial transmit data	On-chip RAM	Main routine
Rxd_data[0]	Stores 1st byte of asynchronous serial receive data	On-chip RAM	Main routine
Rxd_data[1]	Stores 2nd byte of asynchronous serial receive data	On-chip RAM	Main routine
Rxd_data[2]	Stores 3rd byte of asynchronous serial receive data	On-chip RAM	Main routine

Flowcharts

(a) Main processing



main()

```
P_STBY.MSTCR1.BIT.MSTP25 = 0;
P_STBY.MSTCR1.BIT.MSTP24 = 0;
P_STBY.MSTCR1.BIT.MSTP18 = 0;
```

Clear DTC module standby mode
Clear SCI2 module standby mode

```
P_INTC.IPRI.BIT.SCI2 = 10;
```

Set 10 as SCI2 interrupt priority level

```
DTC_1.DTMR = 0x8000;

DTC_1.DTCRA = 3;
DTC_1.DTSAR = (unsigned long)&Txd_data[0];
DTC_1.DTDAR = (unsigned long)&P_SCI2.TDR;
```

Serial transmission DCT1 settings
Normal transfer mode, DTSAR incremented, DTDAR fixed, byte transfer, chain transfer not used, interrupt enabled at end of data transfer
3 transfers
Transfer source DTSAR: transmit data storage RAM
Transfer destination DTDAR: serial TDR_2 register

```
DTC_2.DTMR = 0x2000;

DTC_2.DTCRA = 3;
DTC_2.DTSAR = (unsigned long)&P_SCI2.RDR;
DTC_2.DTDAR = (unsigned long)&Rxd_data[0];
```

Serial reception DCT2 settings
Normal transfer mode, DTSAR fixed, DTDAR incremented, byte transfer, chain transfer not used, interrupt enabled at end of data transfer
3 transfers
Transfer source DTSAR: serial RDR_2 register
Transfer destination DTDAR: receive data storage RAM

```
P_DTC.DTBR= 0xFFFF;
P_DTC.DTEE.BIT.TXI_2 |= 1;
P_DTC.DTEE.BIT.RXI_2 |= 1;
```

Set upper 16 bits of DTC transfer information address to 0xFFFF
Enable DTC activation by serial TXI_2 interrupt
Enable DTC activation by serial RXI_2 interrupt

```
P_SCI2.SCR.BYTE = 0x00;
P_SCI2.SMR.BYTE = 0x20;
P_SCI2.SDCR.BIT.DIR = 0;
P_SCI2.BRR = 21;
```

Serial communication settings
Set internal clock as clock source
Set asynchronous mode, 8-bit data length, 1 stop bit, even parity, clock source = Pϕ/1
Set LSB-first transfer
Set bit rate to approx. 57600 bps (when Pϕ = 40 MHz)

1-bit second interval elapsed?
No

Wait 1-bit second interval

```
P_PORTA.PACRL2.BIT.PA1MD = 2;
P_PORTA.PACRL3.BIT.PA1MD2 =1;
P_PORTA.PACRL2.BIT.PA0MD = 2;
P_PORTA.PACRL3.BIT.PA0MD2 = 1;
```

Pin settings
Set port PA1 pin function to TXD2 output pin (serial transmission pin)
Set port PA0 pin function to RXD2 input pin (serial reception pin)

```
Txd_data[0]='a';
Txd_data[1]='b';
Txd_data[2]='c';
```

Set transmit data

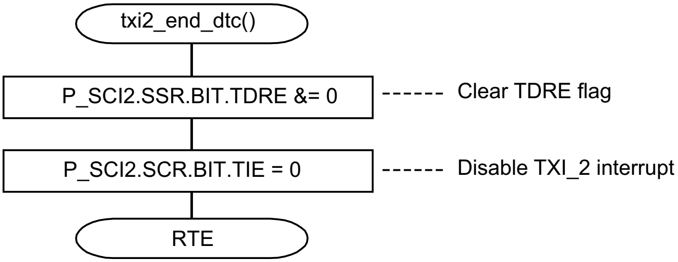
```
P_SCI2.SCR.BYTE |= 0xf0;
```

Enable serial transmit/receive operation, serial transmit/receive interrupts

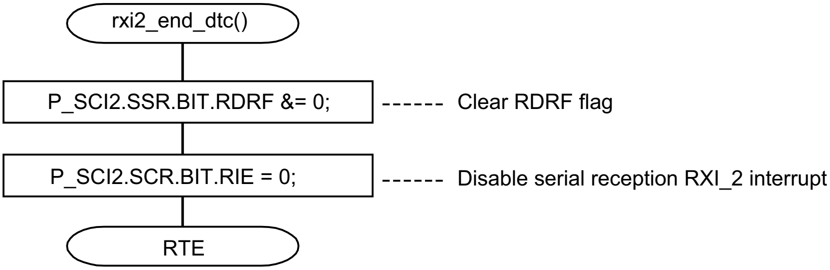
```
set_imask(0);
```

Clear interrupt mask level

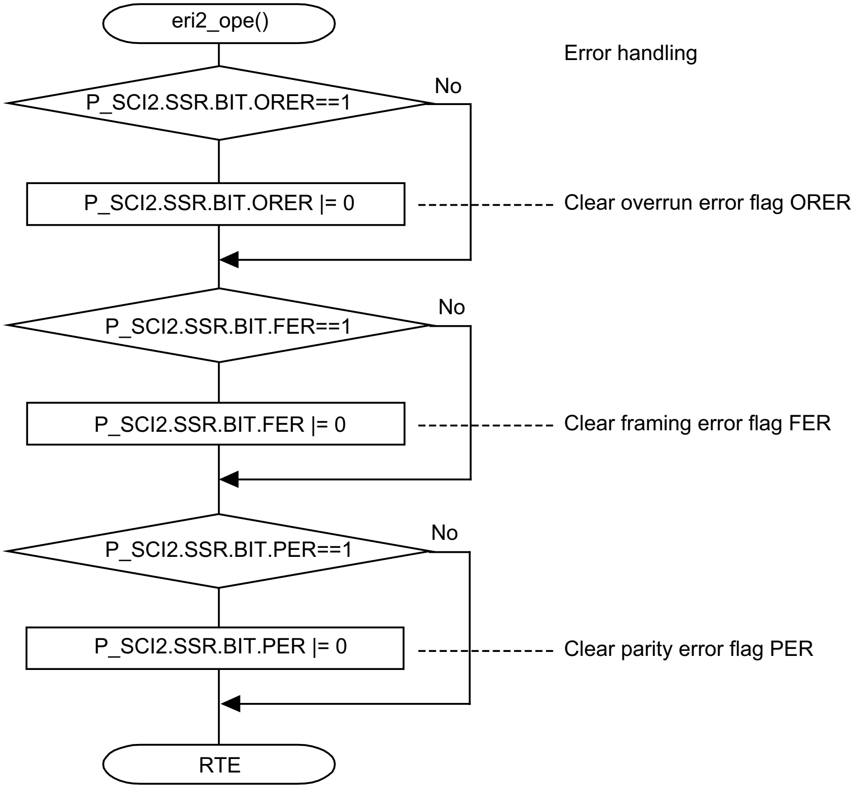
(b) Serial transmission TXI_2 interrupt handling



(c) Serial reception RXI_2 interrupt handling


rxi2_end_dtc()
P_SCI2.SSR.BIT.RDRF &= 0;
Clear RDRF flag
P_SCI2.SCR.BIT.RIE = 0;
Disable serial reception RXI_2 interrupt
RTE

(d) Serial error interrupt handling



Program Listing

```
/********************************************************/
/* SH7046F Series -SH7047- Application Note            */
/* Synchronous Serial Data Transmission with DTC        */
/*                                                      */
/* Function                                             */
/*  :Serial Communication Interface(SCI)                */
/*    Asynchronous Serial Mode                          */
/*     -Transmitting/Receiving-                         */
/*  :Data Transfer Controller(DTC)                      */
/*                                                      */
/* External input clock      :10MHz                     */
/* Internal CPU clock        :40MHz                     */
/* Internal peripheral clock :40MHz                     */
/*                                                      */
/* Written     :     2002/3/01  Rev.1.0                 */
/********************************************************/

#include "iodefine.h"
#include <machine.h>

/*------------ Symbol Definition ------------------------*/
struct st_dtc_tn {                  /* DTC Normal Transfer information          */
    unsigned short DTMR;            /* DTC Mode Register                        */
    unsigned short DTCRA;           /* transfer counter                         */
    unsigned short dummy1;          /* Reserved                                 */
    unsigned short dummy2;          /* Reserved                                 */
    unsigned long DTSAR;            /* source address register                  */
    unsigned long DTDAR;            /* destination address register             */
};

#define DTC_COUNT  3                                      /* DTC Transmit count */
#define DTC_1 (*(volatile struct st_dtc_tn *)0xFFFFE000)  /* Transmit DTC       */
#define DTC_2 (*(volatile struct st_dtc_tn *)0xFFFFE010)  /* Receive DTC        */


/*------------ RAM allocation Definition  --------------------*/
volatile unsigned char Txd_data[DTC_COUNT];               /* Transmit data      */
volatile unsigned char Rxd_data[DTC_COUNT];               /* Receive data       */


/*------------ Function Definition -------------------------*/
void main(void);
void txi2_end(void);
void rxi2_end(void);


/************************************************************/
/* main Program                                             */
/************************************************************/
void main( void )
{
    unsigned long i;

    /* Set standby mode  */
    P_STBY.MSTCR1.BIT.MSTP25 = 0;   /* Disable DTC standby mode                 */
    P_STBY.MSTCR1.BIT.MSTP24 = 0;
    P_STBY.MSTCR1.BIT.MSTP18 = 0;   /* Disable SCI2 standby mode                */

    /* Set interrupt priority level (0 to 15)  */
    P_INTC.IPRI.BIT.SCI2 = 10;      /* SCI2 interrupt level 10                  */

    /* SIC2 Transmit DTC information  */
    DTC_1.DTMR = 0x8000;
                  /* SM[1:0]=b'10;  DTSAR is incremented                        */
                  /* DM[1:0]=0;     DTDAR is  fixed                             */
                  /* MD[1:0]=0;     Transfer mode :Normal mode                  */
                  /* SZ[1:0]=0;     Byte-size transfer                          */
                  /* DTS=0;         destination is block area(no use)           */
                  /* CHNE=0;        Chain transfer is canceled                  */
                  /* DISEL=0;       Interrupt->transfer ends                    */
                  /* NMIM=0;        NMI->Terminate DTC transfer                 */
    DTC_1.DTCRA = DTC_COUNT;                     /* Transfer Count              */
    DTC_1.DTSAR = (unsigned long)&Txd_data[0];   /* set SCI2 Transmit data      */
    DTC_1.DTDAR = (unsigned long)&P_SCI2.TDR;    /* set SCI2 TDR register       */

    /* SIC2 Receive DTC information  */
    DTC_2.DTMR = 0x2000;
                  /* SM[1:0]=0;     DTSAR is fixed                              */
                  /* DM[1:0]=b'10;  DTDAR is incremented                        */
                  /* MD[1:0]=0;     Transfer mode :Normal mode                  */
                  /* SZ[1:0]=0;     Byte-size transfer                          */
                  /* DTS=0;         destination is block area(no use)           */
                  /* CHNE=0;        Chain transfer is canceled                  */
                  /* DISEL=0;       Interrupt->transfer ends                    */
                  /* NMIM=0;        NMI->Terminate DTC transfer                 */
    DTC_2.DTCRA = DTC_COUNT;        /* Transfer Count                           */
    DTC_2.DTSAR = (unsigned long)&P_SCI2.RDR;    /* set SCI2 RDR register       */
    DTC_2.DTDAR = (unsigned long)&Rxd_data[0];   /* set SCI2 Receive Buffer     */

    P_DTC.DTBR = 0xFFFF;                     /* information base register       */
    /* DTC Transmit enable  */
    P_DTC.DTEE.BIT.TXI_2 |= 1;               /* interrupt sources: TXI_2(SCI2)  */
    P_DTC.DTEE.BIT.RXI_2 |= 1;               /* interrupt sources: RXI_2(SCI2)  */

    /* Initialize SCI2 clocked synchronous mode  */
    P_SCI2.SCR.BYTE = 0x00;
                  /* TIE=0;         clear TIE                                   */
                  /* RIE=0;         clear RIE                                   */
                  /* TE=0;          clear TE                                    */
                  /* RE=0;          clear RE                                    */
                  /* MPIE=0;        clear MPIE,TEIE                             */
                  /* TEIE=0;        clear TEIE                                  */
                  /* CKE[1:0]=b'00; clock source: internal ,SCK :input          */
    P_SCI2.SMR.BYTE = 0x20;
                  /* CA=0;          Asynchronous mode                           */
                  /* CHR=0;         data length 8bits                           */
                  /* PE=1;          parity enable                               */
                  /* OE=0;          even parity                                 */
                  /* STOP=0;        1 stop bit                                  */
                  /* MP=0;          disable Multiprocessor Mode                 */
                  /* CKS[1:0]=b'00; clock source =Pϕ/1                          */
    P_SCI2.SDCR.BIT.DIR = 0;                    /* LSB first send               */
    P_SCI2.BRR = 21;                            /* 57600bps@ Pϕ=40MHz           */
    for( i=0; i < 0x500 ; i++);                 /* Wait 1bit over               */

    /* Initialize SCI2 port   */
    P_PORTA.PACRL2.BIT.PA1MD = 2;               /* set TXD2(PA1:73pin@SH7047)   */
    P_PORTA.PACRL3.BIT.PA1MD2 =1;
    P_PORTA.PACRL2.BIT.PA0MD = 2;               /* set RXD2(PA0:75pin@SH7047)   */
    P_PORTA.PACRL3.BIT.PA0MD2 = 1;

    /* set transmit data  */
    Txd_data[0] = ‘a’;
    Txd_data[1] = ‘b’;
    Txd_data[2] = ‘c’;

    P_SCI2.SCR.BYTE |= 0xf0;        /* Transmit/Receive Enable                  */
                       /* TIE=1;    TXI_2 interrupt Enable                      */
                       /* RIE=1;    RXI_2,ERI_2 interrupt Enable                */
                       /* TE=1;     Transmit Enable                             */
                       /* RE=1;     Receive Enable                              */

    set_imask(0);                   /* clear interrupt mask level               */

    while(1);

}

/************************************************************/
/* SIC2:TXI_2 Interrupt                                     */
/* Transmission of DTC data transfer termination            */
/************************************************************/
#pragma interrupt(txi2_end)
void txi2_end(void)
{
    P_SCI2.SSR.BIT.TDRE &= 0;       /* TDRE=0 flag clear                        */
    P_SCI2.SCR.BIT.TIE = 0;         /* TXI_2 interrupt disable                  */
}

/************************************************************/
/* SIC2 RXI_2 Interrupt                                     */
/* Reception of DTC data transfer termination               */
/************************************************************/
#pragma interrupt(rxi2_end)
void rxi2_end(void)
{
    P_SCI2.SSR.BIT.RDRF &= 0;       /* RDRF=0 flag clear                        */
    P_SCI2.SCR.BIT.RIE = 0;         /* RXI_2,ERI_2 interrupt disable            */
}

/************************************************************/
/* SIC2:ERI_2 Interrupt                                     */
/* SCI Reception Error                                     */
/************************************************************/
#pragma interrupt(eri2_ope)
void eri2_ope(void)
```

```
{
    if(P_SCI2.SSR.BIT.ORER==1){      /* Overrun Error                      */
       P_SCI2.SSR.BIT.ORER |= 0;     /* ORER=0 flag clear                  */
    }
    if(P_SCI2.SSR.BIT.FER==1){       /* Framing Error                      */
       P_SCI2.SSR.BIT.FER |= 0;      /* FER=0 flag clear                   */
    }
    if(P_SCI2.SSR.BIT.PER==1){       /* Parity Error                       */
       P_SCI2.SSR.BIT.PER |= 0;      /* PER=0 flag clear                   */
    }

}
```

2.6 Synchronous Serial Data Simultaneous Transmission/Reception and DTC Data Transfer (SCI, DTC)

Synchronous Serial Data Simultaneous Transmission/Reception and DTC Data Transfer (SCI, DTC)	Functions Used: SCI, DTC

Specifications

(1) Simultaneous 3-byte data transmit and receive operations are performed using the synchronous serial transfer function and DTC data transfer function, as shown in figure 2.31.

(2) Serial transmit data transfer and storage of serial receive data in on-chip RAM are performed using the data transfer controller (DTC) transfer function as shown in figure 2.32.

(3) Communication is performed at a bit rate of 1 Mbps, using a fixed 8-bit transmit/receive data length and the LSB-first method in which data is transmitted and received starting from the least significant bit.

(4) The DTC transfer conditions are shown in table 2.36.

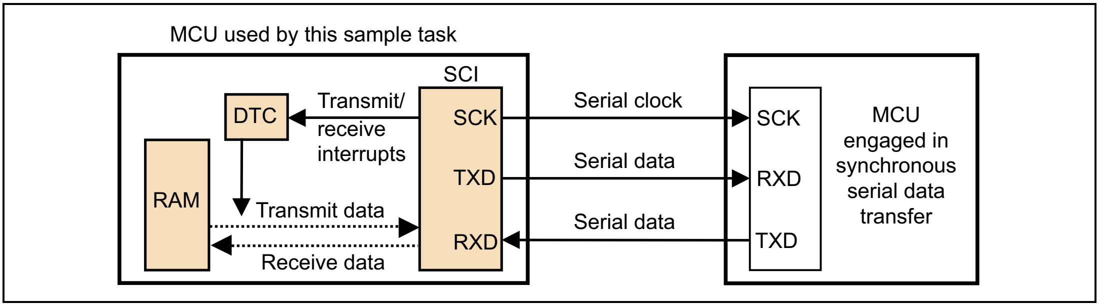

MCU used by this sample task
SCI
DTC
Transmit/ receive interrupts
SCK
TXD
RXD
RAM
Transmit data
Receive data
Serial clock
Serial data
Serial data
SCK
RXD
TXD
MCU engaged in synchronous serial data transfer

Figure 2.31 Synchronous Serial Data Simultaneous Transmission/Reception

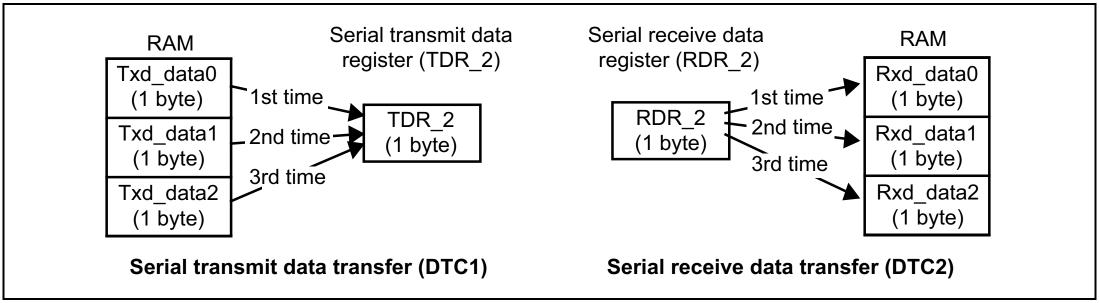


Figure 2.32 Data Transfer Using DTC

Table 2.36 DTC Transfer Conditions

Condition	Serial Transmission DTC Transfer Condition (DTC1)	Serial Reception DTC Transfer Condition (DTC2)
Transfer mode	Normal mode	Normal mode
Number of transfers	3	3
Transfer data size	Byte transfer	Byte transfer
Transfer source	On-chip RAM	Serial receive data register (RDR_2)
Transfer destination	Serial transmit data register (TDR_2)	On-chip RAM
Transfer source address	Transfer source address incremented after transfer	Transfer source address fixed
Transfer destination address	Transfer destination address fixed	Transfer destination address incremented after transfer
Activation source	Activated by SCI ch2 transmit interrupt (TXI_2)	Activated by SCI ch2 receive interrupt (RXI_2)
Interrupt handling	Interrupt to CPU enabled only at end of specified data transfer	Interrupt to CPU enabled only at end of specified data transfer

Functions Used

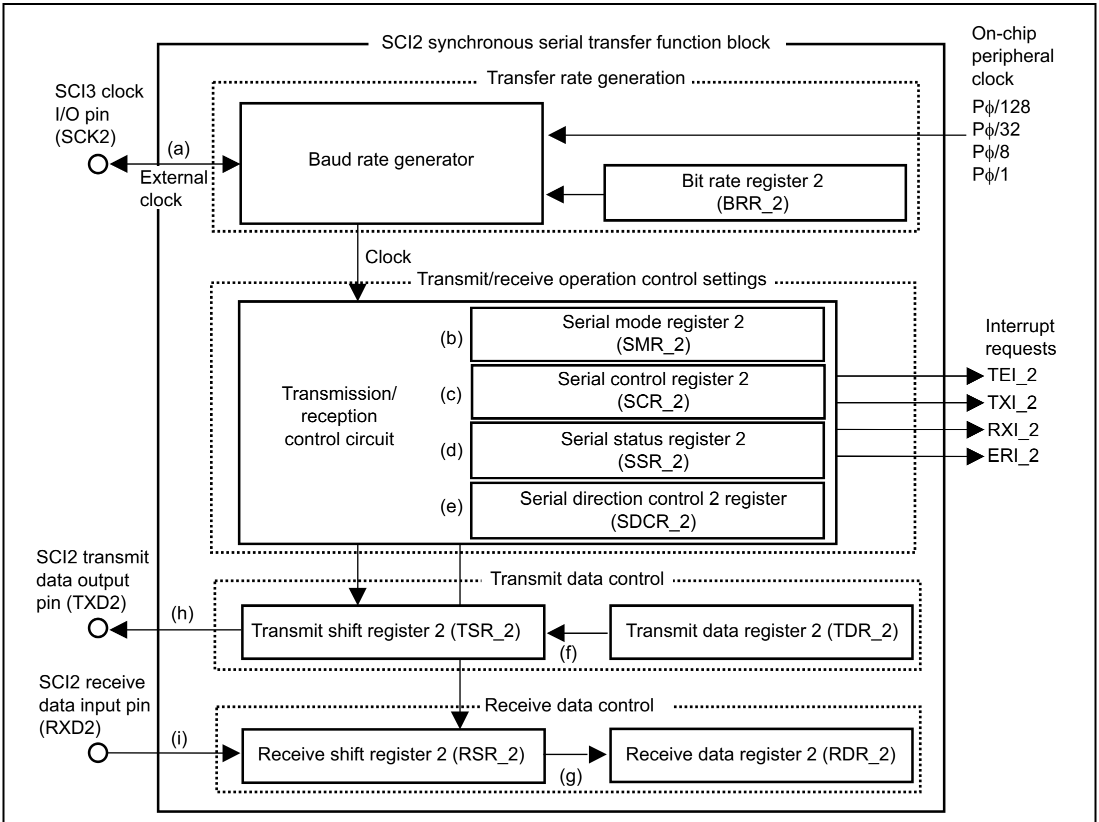
(1) This sample task performs simultaneous synchronous serial data transmit/receive operations using the serial communication interface (SCI) and Data Transfer Controller (DTC).

(a) A block diagram of simultaneous synchronous serial data transmit/receive operations is shown in figure 2.33. The synchronous serial data simultaneous transmission/reception block diagram is explained below.

- On-chip peripheral clock Pϕ is the reference clock for operating on-chip peripheral functions.
- In synchronous mode, a fixed 8-bit data length is used.
- Receive shift register 2 (RSR_2) is used to receive serial data. Serial data input to RSR_2 from the RXD2 pin is set in the order of reception, starting from the LSB (bit 0), and converted to parallel data. When one byte of data is received, it is automatically transferred to RDR_2. RSR_2 cannot be directly read or written to by the CPU.
- Receive data register 2 (RDR_2) is an 8-bit register that stores received serial data. When reception of one byte of data is completed, the received data is transferred from RSR_2 to RDR_2, and the receive operation is terminated. RSR_2 then becomes able to receive. RSR_2 and RDR_2 are double-buffered, allowing continuous receive operations. RDR_2 is a receive-only register, and cannot be written to by the CPU.
- Transmit shift register 2 (TSR_2) is used to transmit serial data. Transmit data from TDR_2 is first transferred to TSR_2, and sent to the TXD pin in order starting from the LSB (bit 0) to implement serial data transmission. When one byte of data has been transmitted, the next transmitted is automatically transferred from TDR_2 to TSR_2, and transmission is started. However, data transfer from TDR_2 to TSR_2 is not performed if data has not been written to TDR_2 (if TDRE is set to 1). TSR_2 cannot be directly read or written to by the CPU.
- Transmit data register 2 (TDR_2) is an 8-bit register that stores transmit data. When the TSR_2 “empty” state is detected, transmit data written to TDR_2 is transferred to TSR_2 and serial data transmission is started. Writing the next transmit data to TDR_2 during TSR_2 serial data transmission enables continuous transmission to be performed. TDR_2 can be read or written to by the CPU at any time.
- Serial mode register 2 (SMR_2) is an 8-bit register for setting the serial data communication format and selecting the clock source of the on-chip baud rate generator.
- Serial control register 2 (SCR_2) is an 8-bit register that performs transmit/receive operation and transmit/receive clock source selection.
- Serial status register 2 (SSR_2) contains SCI2 status flags and a transmit/receive multiprocessor bit. TDRE, RDRF, OER, PER, and FER can only be cleared.
- The serial direction control 2 register (SDCR_2) performs LSB-first/MSB-first selection by means of the DIR bit. In the case of an 8-bit length, LSB-first or MSB-

first can be selected regardless of the serial communication mode. In the case of a 7-bit length, LSB-first must be selected.

- Bit rate register 2 (BRR_2) is an 8-bit register for adjusting the bit rate.
- Transmit data is output from one fall of the transfer clock until the next. Receive data is captured on a rise of the transfer clock.


SCI2 synchronous serial transfer function block
On-chip peripheral clock
Pϕ/128
Pϕ/32
Pϕ/8
Pϕ/1
SCI3 clock I/O pin (SCK2)
Transfer rate generation
(a)
Baud rate generator
External clock
Bit rate register 2 (BRR_2)
Clock
Transmit/receive operation control settings
Transmission/ reception control circuit
(b)
Serial mode register 2 (SMR_2)
(c)
Serial control register 2 (SCR_2)
(d)
Serial status register 2 (SSR_2)
(e)
Serial direction control 2 register (SDCR_2)
Interrupt requests
TEI_2
TXI_2
RXI_2
ERI_2
SCI2 transmit data output pin (TXD2)
(h)
Transmit data control
Transmit shift register 2 (TSR_2)
(f)
Transmit data register 2 (TDR_2)
SCI2 receive data input pin (RXD2)
(i)
Receive data control
Receive shift register 2 (RSR_2)
(g)
Receive data register 2 (RDR_2)

Notes:
(a) Outputs the serial clock.
(b) Performs serial data communication format setting and baud rate generator clock source selection.
(c) Performs transmit/receive operation and synchronous mode clock output pin selection.
(d) Indicates the SCI2 operation status by means of status flags (transmit data register empty, receive data register full, overrun error).
(e) Performs LSB-first/MSB-first selection.
(f) Transmit data written to TDR_2 is transferred to TSR_2 on detection of a TSR_2 "empty" state.
(g) On completion of reception of 1 byte of data, received data is transferred from RSR_2 to RDR_2.

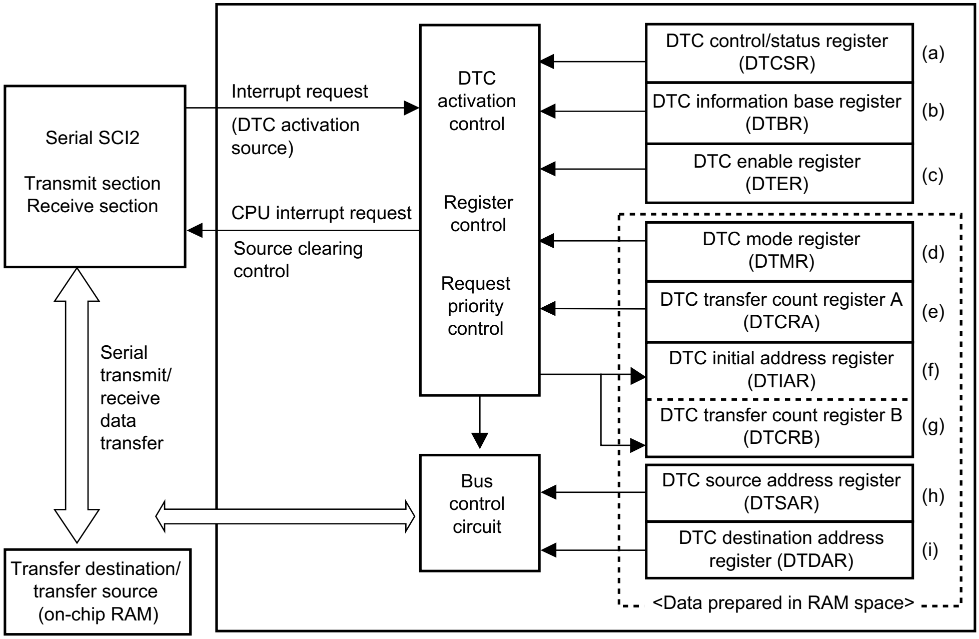
Figure 2.33 Synchronous Serial Data Transmission/Reception Block Diagram

(b) A block diagram of the DTC is shown below. Of the three DTC transfer modes — normal mode, repeat mode, and block transfer mode —this sample task uses normal mode to perform transfer of serial transmit/receive data. DTC data transfer is performed using the serial transmission TXI interrupt and serial reception RXI interrupt as DTC activation sources. The block diagram is explained below.

- The DTC mode register (DTMR) is a 16-bit register that controls the DTC's operating mode.
- The DTC source address register (DTSAR) is a 32-bit register that specifies the transfer source address of data to be transferred by the DTC. An even address should be specified in the case of word size transfer, and an address that is a multiple of 4 in the case of longword transfer.
- The DTC destination address register (DTDAR) is a 32-bit register that specifies the transfer destination address of data to be transferred by the DTC. An even address should be specified in the case of word size transfer, and an address that is a multiple of 4 in the case of longword transfer.
- The DTC initial address register (DTIAR) is a 32-bit register that specifies the transfer source/transfer destination initial address in repeat mode. In repeat mode, when the DTS bit is 1, specify the initial transfer source address in the repeat area, and when the DTS bit is 0, specify the initial transfer destination address in the repeat area.
- DTC transfer count register A (DTCRA) is a 16-bit register that specifies the number of transfers in DTC data transfer. In normal mode, DTCRA functions as a transfer counter (1 to 65,536). In repeat mode, upper 8-bit DTCRAH holds the number of transfers, and lower 8-bit DTCRAL functions as an 8-bit transfer counter. In block transfer mode, DTCRA functions as a 16-bit transfer counter.
- DTC transfer count register B (DTCRB) is a 16-bit register that specifies the block length in block transfer mode.
- The DTC enable register (DTER) is used to select the interrupt source that activates the DTC, and comprises registers DTEA through DTEF.
- The DTC control/status register (DTCSR) is a 16-bit register that sets enabling/disabling of DTC activation by software, and sets a software activation DTC vector address. DTCSR also shows the DTC transfer status.
- The DTC information base register (DTBR) is a readable/writable 16-bit register that specifies the upper 16 bits of the memory address that stores DTC transfer information. Word or longword access must be used for DTBR. If byte access is used, the register contents will be undefined in the case of a write, and an undefined value will be returned in the case of a read.
- Information of six registers — the DTC mode register (DTMR), DTC source address register (DTSAR), DTC destination address register (DTDAR), DTC initial address register (DTIAR), DTC transfer count register A (DTCRA), and DTC transfer count register B (DTCRB) — cannot be accessed directly from the CPU. When a DTC activation source occurs, the relevant register information is transferred to these

registers from information of an arbitrary set of registers located in on-chip RAM and DTC transfer is performed, and when transfer ends, the contents of these registers are returned to RAM. Therefore, register information should be prepared in arbitrary on-chip RAM in the user program.

- This task uses DTC normal mode for both serial transmit data transfer and serial receive data transfer. Two sets of normal mode register information (DTMR, DTSAR, DTDAR, DTCRA, and DTCRB) are provided, for serial transmission use and for serial reception use.



Serial SCI2
Transmit section
Receive section
Interrupt request
(DTC activation source)
CPU interrupt request
Source clearing control
Serial transmit/ receive data transfer
Transfer destination/ transfer source (on-chip RAM)
DTC activation control
Register control
Request priority control
Bus control circuit
DTC control/status register (DTCSR)
(a)
DTC information base register (DTBR)
(b)
DTC enable register (DTER)
(c)
DTC mode register (DTMR)
(d)
DTC transfer count register A (DTCRA)
(e)
DTC initial address register (DTIAR)
(f)
DTC transfer count register B (DTCRB)
(g)
DTC source address register (DTSAR)
(h)
DTC destination address register (DTDAR)
(i)
<Data prepared in RAM space>

Notes:
(a) Performs enabling/disabling of DTC activation by software, and software activation DTC vector address setting.
(b) Performs specification of the upper 16 bits of the memory address that stores DTC transfer information.
(c) Selects the interrupt source that activates the DTC; comprises six registers, DTEA through DTEF.
(d) Performs DTC operating mode setting.
(e) Specifies the number of transfers in DTC data transfer.
(f) In repeat mode, specifies the transfer source/transfer destination initial address in repeat mode. Not used in normal mode. In block transfer mode, functions as the DTCRB register.
(g) In block transfer mode, specifies the block length. Not used in normal mode. In repeat mode, functions as the DTIAR register.
(h) Specifies the transfer source address of data to be transferred by the DTC.
(i) Specifies the transfer destination address of data to be transferred by the DTC.

Figure 2.34 DTC Block Diagram

(5) Table 2.37 shows the function assignments used in this sample task.

Table 2.37 Function Assignments

Function	Type	Function Assignment
SCK2	Pin	Channel 2 clock output pin
TXD2	Pin	Channel 2 transmit data output pin
RXD2	Pin	Channel 2 receive data input pin
SMR_2	SCI2	Communication format setting; set to synchronous mode
SCR_2	SCI2	Enables transmit/receive operation, interrupts; SCK2 set as clock output pin
SSR_2	SCI2	Status flags indicating SCI2 operation status
SDCR_2	SCI2	Set to LSB-first transfer
BBR_2	SCI2	Sets transmit/receive bit rate
TSR_2	SCI2	Register for transmitting serial data
TDR_2	SCI2	Register that stores transmit data
RSR_2	SCI2	Register for receiving serial data
RDR_2	SCI2	Register that stores receive data
DTMR	DTC	Sets DTC to normal transfer mode
DTCRA	DTC	Setting of number of transfers
DTSAR	DTC	Transfer source address setting
DTDAR	DTC	Transfer destination address setting
DTBR	DTC	Setting of upper 16 bits of DTC vector
DTER	DTC	Enables DTC activation in serial reception/serial transmission

Operation

(1) The principles of operation of this sample task are shown in the figure below.

Simultaneous transmission/reception of synchronous serial data is performed by means of hardware and software processing as shown in the figure.

(a) Transmit processing

- 3-byte data is transmitted by synchronous serial communication.
- 3-byte transmit data is transferred from on-chip RAM to the SCI using the DTC.
- The serial TXI_2 interrupt is used for DTC activation.

(b) Receive processing

- 3-byte data is received by synchronous serial communication.
- 3-byte receive data is transferred from the SCI to on-chip RAM using the DTC.
- The serial RXI_2 interrupt is used for DTC activation.

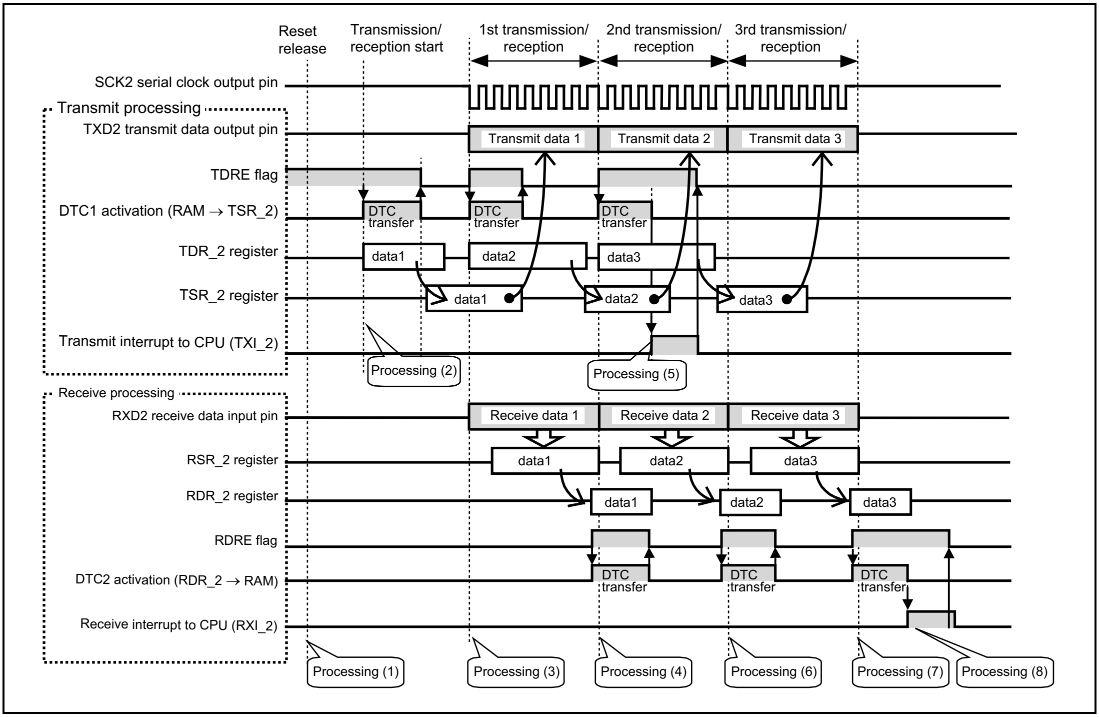


Figure 2.35 Principles of Operation

The figure is explained below.

	Serial Transmission Side Processing	Serial Reception Side Processing
Processing (1)	Initial Settings (1) SCI2 settings • Set synchronous mode, SCK serial clock output, LSB-first transfer • Enable transmit interrupts, receive interrupts, transmit operation, receive operation (2) DTC settings • Set normal transfer mode • Enable DTC1 activation by serial transmit interrupt (TXI_2), DTC2 activation by serial receive interrupt (RXI_2)	
Processing (2)	Hardware Processing • DTC1 activation by TXI_2 interrupt (1st time) • Transfer transmit data 1 from on-chip RAM to TDR_2 register (DTC1) • Clear TDRE (DTC1)	None
	Software Processing None	
Processing (3)	Hardware Processing • When TDRE flag is 0, transfer transmit data 1 from TDR_2 to TSR_2 register (SCI2) • Set TDRE flag to 1 (SCI2) • Start transmission (SCI2) • DTC1 activation by TXI_2 interrupt (2nd time) • Transfer transmit data 2 from on-chip RAM to TDR_2 (DTC1) • Clear TDRE flag (DTC1)	Hardware Processing • Reception start • Capture receive data 1 in RSR register
	Software Processing None	Software Processing None
Processing (4)	Hardware Processing • When last bit is sent, check TDRE flag (SCI2) • When TDRE flag is 0, transfer transmit data 2 from TDR_2 to TSR_2 register (SCI2) • Set TDRE flag to 1 (SCI2) • Start transmission of next frame (SCI2) • DTC1 activation by TXI_2 interrupt (3rd time) • Transfer transmit data 3 from on-chip RAM to TDR_2 and terminate (DTC1); TDRE flag is not cleared	Hardware Processing • Transfer receive data 1 from RSR to RDR register (SCI2) • Set RDRE flag to 1 (SCI2) • Start reception of next frame (SCI2) • Capture receive data 2 in RSR register (SCI2) • DTC2 activation by RXI_2 interrupt (1st time) • Transfer receive data 1 from RDR_2 to on-chip RAM (DTC2) • Clear RDRE flag (DTC2)
	Software Processing None	Software Processing None

	Serial Transmission Side Processing	**Serial Reception Side Processing**
Processing (5)	Hardware Processing • Generation of TXI_2 interrupt to CPU	None
	Software Processing • Clear TDRE flag • Disable TXI_2 interrupt	
Processing (6)	Hardware Processing • When last bit is sent, check TDRE flag (SCI2) • When TDRE flag is 0, transfer transmit data 3 from TDR_2 to TSR_2 register (SCI2) • Set TDRE flag to 1 (SCI2) • Start transmission of next frame (SCI2)	Hardware Processing • Transfer receive data 2 from RSR to RDR register (SCI2) • Set RDRE flag to 1 (SCI2) • Start reception of next frame (SCI2) • Capture receive data 3 in RSR register • DTC2 activation by RXI_2 interrupt (2nd time) • Transfer receive data 2 from RDR_2 to on-chip RAM (DTC2) • Clear RDRE flag (DTC2)
	Software Processing None	Software Processing None
Processing (7)	None	Hardware Processing • Transfer receive data 3 from RSR to RDR register (SCI2) • Set RDRE flag to 1 (SCI2) • DTC2 activation by RXI_2 interrupt (2nd time) • Transfer receive data 2 from RDR_2 to on-chip RAM (DTC2) • Clear RDRE flag (DTC2)
		Software Processing None
Processing (8)	None	Hardware Processing As RDRE flag is set to 1, TXI_2 interrupt to CPU is generated
		Software Processing • Clear TDRE flag • Disable TXI_2 interrupt

(2) The principles of operation of DTC activation are shown in the figure below. When executing DTC transfer, the following settings should be made before an activation source occurs.

- Make DTC register information settings and place DTC register information in RAM.
- Set the lower 16 bits of the start address (32 bits) of DTC register information in the DTC vector table.
- Set the lower 16 bits of the start address (32 bits) of DTC register information in the DTC information base register.

The DTC is activated by the following processing.

- A DTC activation source interrupt is generated.
- The lower 16 bits of the start address of DTC register information are read from the address corresponding to the DTC vector table activation source.
- The upper 16 bits of the start address of DTC register information are read from the DTC information base register (DTMR).
- The 32-bit start address of DTC register information is generated from the read start address lower 16 bits and upper 16 bits.
- The start of DTC register information is read sequentially from the DTC register information start address, and data transfer is performed.

In this task, a TXI_2 interrupt is used as the DTC activation source in serial transmission data transfer, and an RXI_2 interrupt in serial reception data transfer

The following table shows the register information configuration in normal mode.

Table 2.38 DTC Register Information (Normal Mode)

Setting Address	Register Name	Data Length
RF	DTC mode register (DTMR)	Word (2 bytes)
RF+2	DTC transfer count register A (DTCRA)	Word (2 bytes)
RF+8	DTC source address register (DTSAR)	Longword (4 bytes)
RF+12	DTC destination address register (DTDAR)	Longword (4 bytes)

RF: DTC register information start address (in on-chip RAM)

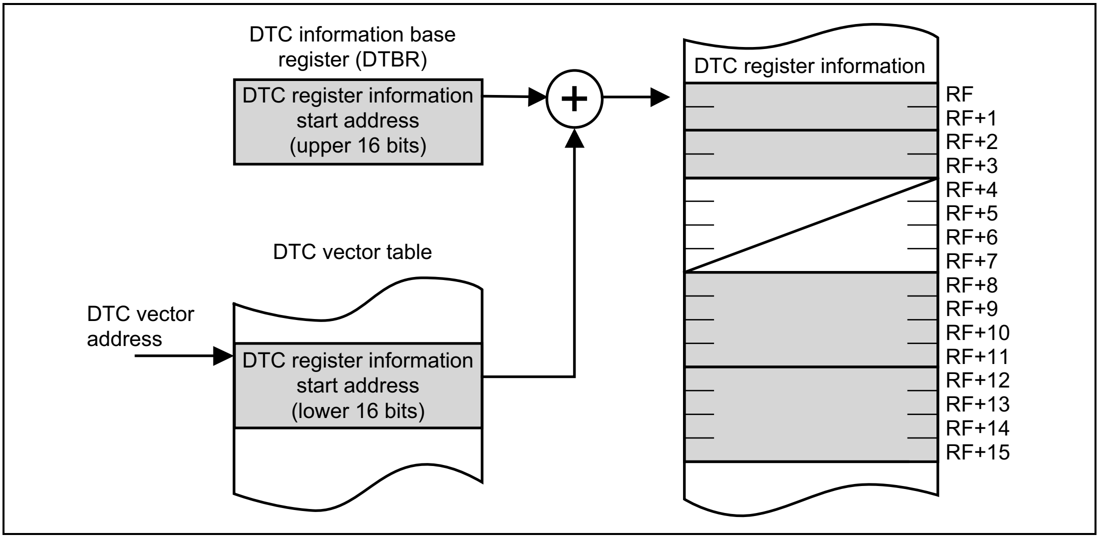

DTC information base register (DTBR)
DTC register information start address (upper 16 bits)
+
DTC vector table
DTC vector address
DTC register information start address (lower 16 bits)
DTC register information
RF
RF+1
RF+2
RF+3
RF+4
RF+5
RF+6
RF+7
RF+8
RF+9
RF+10
RF+11
RF+12
RF+13
RF+14
RF+15

Figure 2.36 Correspondence between DTC Vector Address and Transfer Information

Software

(1) Modules

The following table shows the modules used by this sample task.

Table 2.39 Modules

Module Name	Label	Function
Main routine	main	SCI ch2 asynchronous serial communication and DTC initialization, serial communication start
SCI transmit end interrupt	txi2_end	SCI ch2 transmit end interrupt. Interrupt generation at end of specified number of DTC transfers
SCI receive end interrupt	rxi2_end	SCI ch2 receive end interrupt. Interrupt generation at end of specified number of DTC transfers

(2) Arguments

The following table shows the arguments used by this sample task.

Table 2.40 Arguments

Argument	Function	Module Name	Data Length	Input/ Output
Txd_data[0] to [2]	Asynchronous serial transmit data storage	Main routine	1 byte	Output
Rxd_data[0] to [2]	Asynchronous serial receive data storage	Main routine	1 byte	Input

(3) Internal Registers Used

The following table shows the internal registers used by this sample task.

Table 2.41 Internal Registers Used

Register Name		Function	Address	Set Value
	Bits		Bits	
P_STBY.MSTCR1	MSTP25 MSTP24	Module standby control register 1 DTC module standby control bits: When MSTP25 = MSTP24 = 0, module standby release Same value is set for MSTP25 and MSTP24	H'FFFF861C Bit 9 Bit 8	B'00
	MSTP18	Module standby control register 2 Serial Communication Interface 2 standby control bit: When MSTP18 = 0, module standby release	H'FFFF861C Bit 2	0
P_INTC.IPRI	SCI2	Interrupt priority register I (IPRI) Interrupt priority level setting of SCI2 interrupts (ERI, RXI, TXI, TEI): When SCI2 = b'1010 (10), interrupts are set to priority level 10	H'FFFF835C Bits 12 to 15	10

Register Name		Function	Address	Set Value
	Bits		Bits	
DTC_1.DTMR		DTC mode register (DTMR) DTC operating mode control setting. For serial transmission use	Located in on-chip RAM	H'8000
	SM1 SM0	Source address mode: When SM[1:0] = b'10, DTSAR is incremented after transfer	Bit 15 Bit 14	
	DM1 DM0	Destination address mode: When DM[1:0] = b'00, DTDAR is fixed	Bit 13 Bit 12	
	MD1 MD0	DTC transfer mode: When MD[1:0] = b'00, normal transfer mode	Bit 11 Bit 10	
	SZ1 SZ0	DTC data transfer size: When SZ[1:0] = b'00, byte (1-byte) transfer	Bit 9 Bit 8	
	DTS	DTC transfer mode select: When DTS = b'0, destination side is block area Not used in normal mode	Bit 7	
	CHNE	DTC chain transfer enable: When CHNE = b'0, chain transfer is cleared	Bit 6	
	DISEL	DTC interrupt select: When DISEL = b'0, interrupt request to CPU is generated only at end of specified data transfer	Bit 5	
	NMIM	DTC NMI mode: When NMIM = b'0, DTC transfer is suspended by NMI	Bit 4	
DTC_1.DTCRA		DTC transfer count register A (DTCRA) Specifies number of transfers in DTC data transfer Set to 3 transfers	Located in on-chip RAM	H'0003
DTC_1.DTSAR		DTC source address register (DTSAR) 32-bit register that specifies transfer source address of data to be transferred by DTC Set to start address of transmit data storage area	Located in on-chip RAM	Txd_data
DTC_1.DTDAR		DTC destination address register (DTDAR) 32-bit register that specifies transfer destination address of data to be transferred by DTC Set to serial transmit data register (TDR_2)	Located in on-chip RAM	&P_SCI2.TDR

Register Name		Function	Address	Set Value
	Bits		Bits	
DTC_2.DTMR		DTC mode register (DTMR) DTC operating mode control setting. For serial reception use	Located in on-chip RAM	H'2000
	SM1 SM0	Source address mode: When SM[1:0] = b'00, DTSAR is fixed	Bit 15 Bit 14	
	DM1 DM0	Destination address mode: When DM[1:0] = b'10, DTDAR is incremented after transfer	Bit 13 Bit 12	
	MD1 MD0	DTC transfer mode: When MD[1:0] = b'00, normal transfer mode	Bit 11 Bit 10	
	SZ1 SZ0	DTC data transfer size: When SZ[1:0] = b'00, byte (1-byte) transfer	Bit 9 Bit 8	
	DTS	DTC transfer mode select: When DTS = b'0, destination side is block area Not used in normal mode	Bit 7	
	CHNE	DTC chain transfer enable: When CHNE = b'0, chain transfer is cleared	Bit 6	
	DISEL	DTC interrupt select: When DISEL = b'0, interrupt request to CPU is generated only at end of specified data transfer	Bit 5	
	NMIM	DTC NMI mode: When NMIM = b'0, DTC transfer is suspended by NMI	Bit 4	
DTC_2.DTCRA		DTC transfer count register A (DTCRA) Specifies number of transfers in DTC data transfer Set to 3 transfers	Located in on-chip RAM	H'0003
DTC_2.DTSAR		DTC source address register (DTSAR) 32-bit register that specifies transfer source address of data to be transferred by DTC Set to serial receive data register (RDR_2)	Located in on-chip RAM	&P_SCI2.RDR
DTC_2.DTDAR		DTC destination address register (DTDAR) 32-bit register that specifies transfer destination address of data to be transferred by DTC Set to start address of receive data storage area	Located in on-chip RAM	Rxd_data
P_DTC.DTBR		DTC information base register (DTBR) Specifies upper 16 bits of memory address that stores DTC transfer information	H'FFFF8708	0xFFFF

Register Name		Function	Address	Set Value
	Bits		Bits	
P_DTC.DTEE	TXI_2	DTC enable register E (DTEE) When TXI_2 (DTEE2) = b'1, SCI2 transmit end interrupt (TXI_2) is activation source	H'FFFF8710 Bit 2	1
	RXI_2	DTC enable register E (DTEE) When RXI_2 (DTEE3) = b'1, SCI2 receive end interrupt (RXI_2) is activation source	H'FFFF8710 Bit 3	1
P_SCI2.SCR.BYTE		Serial control register 2 (SCR_2) Transmission/reception control, interrupt control, transmit/receive clock source selection	H'FFFF81C2	H'f1
	TIE	Transmit interrupt enable: When TIE = 1, TXI interrupt requests are enabled	Bit 7	
	RIE	Receive interrupt enable: When RIE = 1, RXI and ERI interrupt requests are enabled	Bit 6	
	TE	Transmit enable: When TE = 1, transmit operation is enabled	Bit 5	
	RE	Receive enable: When RE = 1, receive operation is enabled	Bit 4	
	MPIE	Multiprocessor interrupt enable (Only valid in asynchronous mode when MP = 1 in SMR) In this task, setting is invalid	Bit 3	
	TEIE	Transmit end interrupt enable: When TEIE = 0, TEI interrupt requests are disabled	Bit 2	
	CKE1 CKE0	Clock enable 1-0 Clock source and SCK pin function selection: When CKEI[1:0] = b'01, clock source is internal clock, and SCK pin is serial clock input	Bit 1 Bit 0	

Register Name		Function	Address	Set Value
	Bits		Bits	
P_SCI2.SMR.BYTE		Serial mode register 2 (SMR_2) Communication format and on-chip baud rate generator clock selection	H'FFFF81C0	H'80
	C/A	Communication mode: When C/A = 1, operation in synchronous mode	Bit 7	
	CHR	Character length (valid only in asynchronous mode): When CHR = 0, transmission/reception using 8-bit data length In synchronous mode, fixed 8-bit data length is used In this task, setting is invalid	Bit 6	
	PE	Parity enable (valid only in asynchronous mode): When PE = 1, parity bit is added when transmitting and parity is checked when receiving In this task, setting is invalid	Bit 5	
	O/E	Parity mode (valid only in asynchronous mode when PE = 1): When O/E = 0, transmission/reception using even parity In this task, setting is invalid	Bit 4	
	STOP	Stop bit length (valid only in asynchronous mode) Selects stop bit length when transmitting: When STOP = 0, 1 stop bit In this task, setting is invalid	Bit 3	
	MP	Multiprocessor mode (valid only in asynchronous mode) When MP = 1, multiprocessor communication function is enabled In this task, setting is invalid	Bit 2	
	CKS1 CKS0	Clock select 1-0 Selection of on-chip baud rate generator clock source: When CKS[1:0] = b'00, set to Pϕ/1 clock (n = 0)	Bit 1 Bit 0	
P_SCI2.SDCR	DIR	Serial direction control register 2 (SDCR_2) Data transfer direction: When DIR = 0, TDR contents are transmitted LSB-first, and receive data is stored in RDR LSB-first	H'FFFF81C6 Bit 3	1

Register Name		Function	Address	Set Value
	Bits		Bits	
P_SCI2.BRR		Bit rate register 2 (BRR_2): When BRR_2 = 9, bit rate is 1 Mbps (when clock source = Pϕ/1, Pϕ = 40 MHz)	H'FFFF81C1	9
P_SCI2.TDR		Transmit data register 2 (TDR_2) 8-bit register that stores transmit data	H'FFFF81C3	
P_SCI2.RDR		Receive data register 2 (RDR_2) 8-bit register that stores receive data	H'FFFF81C5	
P_SCI2.SSR	TDRE	Serial status register 2 (SSR_2) Transmit data register empty	H'FFFF81C4	1
	RDRF	Serial status register 2 (SSR_2) Receive data register full	H'FFFF81C4	0
	ORER	Serial status register 2 (SSR_2) Overrun error	H'FFFF81C4	0
P_PORTA.PACRL3	PA0MD2	Port A control register L3 Port A control register L2 PA0 mode bits, PA0/A0/POE0/RXD2 pin function selection: When (PA0MD2, PA0MD[1], PA0MD[0] = b'110, pin function is set to RXD2 input (SCI)	H'FFFF838A Bit 0	b'1
P_PORTA.PACRL2	PA0MD[1] PA0MD[0]		H'FFFF838E Bit 1 Bit 0	b'10
P_PORTA.PACRL3	PA1MD2	Port A control register L3 Port A control register L2 PA1 mode bits, PA1/A0/POE1/TXD2 pin function selection: When (PA1MD2, PA1MD[1], PA1MD[0] = b'110, pin function is set to TXD2 output (SCI)	H'FFFF838A Bit 1	b'1
P_PORTA.PACRL2	PA1MD[1] PA1MD[0]		H'FFFF838E Bit 3 Bit 2	b'10
P_PORTA.PACRL3	PA2MD2	Port A control register L3 Port A control register L2 PA2 mode bits, PA2/IRQ0/A2/PCI0/SCK2 pin function selection: When (PA2MD2, PA2MD[1], PA2MD[0] = b'110, pin function is set to SCK2 input/output (SCI)	H'FFFF838A Bit 2	b'1
P_PORTA.PACRL2	PA2MD[1] PA2MD[0]		H'FFFF838E Bit 5 Bit 4	b'10
P_PORTA.PAIORL	PA2IOR	Port A IO register L Sets port A pin input/output direction When (PA2IOR = 1, SCK2 (PA2) pin is set as output pin	H'FFFF8386 Bit 2	b'1

(4) RAM Used

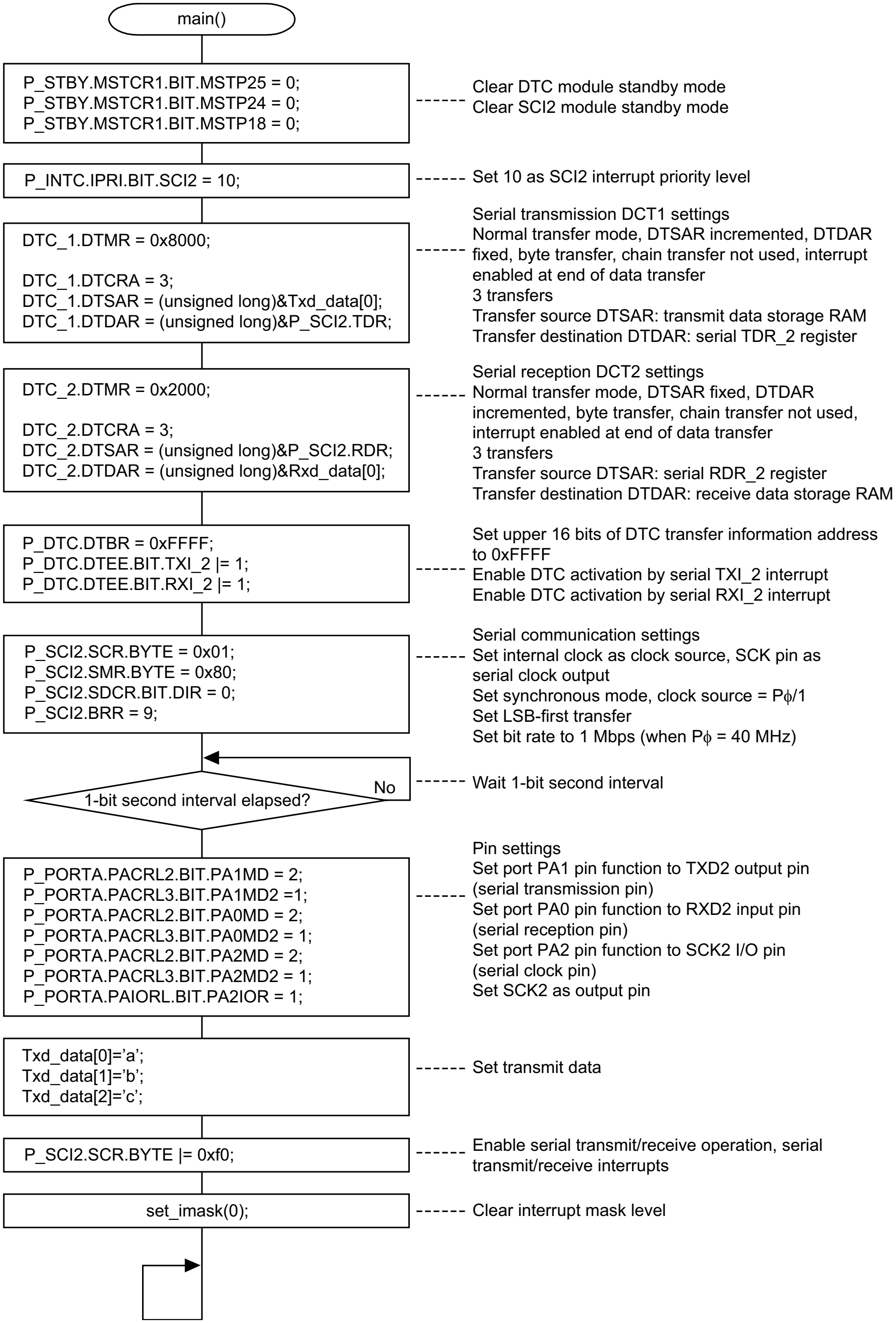
The following table shows the RAM used by this sample task.

Table 2.42 RAM Used

Label	Function	Address	Module Using RAM
Txd_data[0]	Stores 1st byte of synchronous serial transmit data	On-chip RAM	Main routine
Txd_data[1]	Stores 2nd byte of synchronous serial transmit data	On-chip RAM	Main routine
Txd_data[2]	Stores 3rd byte of synchronous serial transmit data	On-chip RAM	Main routine
Rxd_data[0]	Stores 1st byte of synchronous serial receive data	On-chip RAM	Main routine
Rxd_data[1]	Stores 2nd byte of synchronous serial receive data	On-chip RAM	Main routine
Rxd_data[2]	Stores 3rd byte of synchronous serial receive data	On-chip RAM	Main routine

Flowcharts

(a) Main processing



```
P_STBY.MSTCR1.BIT.MSTP25 = 0;
P_STBY.MSTCR1.BIT.MSTP24 = 0;
P_STBY.MSTCR1.BIT.MSTP18 = 0;
```

Clear DTC module standby mode
Clear SCI2 module standby mode

```
P_INTC.IPRI.BIT.SCI2 = 10;
```

Set 10 as SCI2 interrupt priority level

```
DTC_1.DTMR = 0x8000;

DTC_1.DTCRA = 3;
DTC_1.DTSAR = (unsigned long)&Txd_data[0];
DTC_1.DTDAR = (unsigned long)&P_SCI2.TDR;
```

Serial transmission DCT1 settings
Normal transfer mode, DTSAR incremented, DTDAR fixed, byte transfer, chain transfer not used, interrupt enabled at end of data transfer
3 transfers
Transfer source DTSAR: transmit data storage RAM
Transfer destination DTDAR: serial TDR_2 register

```
DTC_2.DTMR = 0x2000;

DTC_2.DTCRA = 3;
DTC_2.DTSAR = (unsigned long)&P_SCI2.RDR;
DTC_2.DTDAR = (unsigned long)&Rxd_data[0];
```

Serial reception DCT2 settings
Normal transfer mode, DTSAR fixed, DTDAR incremented, byte transfer, chain transfer not used, interrupt enabled at end of data transfer
3 transfers
Transfer source DTSAR: serial RDR_2 register
Transfer destination DTDAR: receive data storage RAM

```
P_DTC.DTBR = 0xFFFF;
P_DTC.DTEE.BIT.TXI_2 |= 1;
P_DTC.DTEE.BIT.RXI_2 |= 1;
```

Set upper 16 bits of DTC transfer information address to 0xFFFF
Enable DTC activation by serial TXI_2 interrupt
Enable DTC activation by serial RXI_2 interrupt

```
P_SCI2.SCR.BYTE = 0x01;
P_SCI2.SMR.BYTE = 0x80;
P_SCI2.SDCR.BIT.DIR = 0;
P_SCI2.BRR = 9;
```

Serial communication settings
Set internal clock as clock source, SCK pin as serial clock output
Set synchronous mode, clock source = Pϕ/1
Set LSB-first transfer
Set bit rate to 1 Mbps (when Pϕ = 40 MHz)

Wait 1-bit second interval

```
P_PORTA.PACRL2.BIT.PA1MD = 2;
P_PORTA.PACRL3.BIT.PA1MD2 =1;
P_PORTA.PACRL2.BIT.PA0MD = 2;
P_PORTA.PACRL3.BIT.PA0MD2 = 1;
P_PORTA.PACRL2.BIT.PA2MD = 2;
P_PORTA.PACRL3.BIT.PA2MD2 = 1;
P_PORTA.PAIORL.BIT.PA2IOR = 1;
```

Pin settings
Set port PA1 pin function to TXD2 output pin (serial transmission pin)
Set port PA0 pin function to RXD2 input pin (serial reception pin)
Set port PA2 pin function to SCK2 I/O pin (serial clock pin)
Set SCK2 as output pin

```
Txd_data[0]='a';
Txd_data[1]='b';
Txd_data[2]='c';
```

Set transmit data

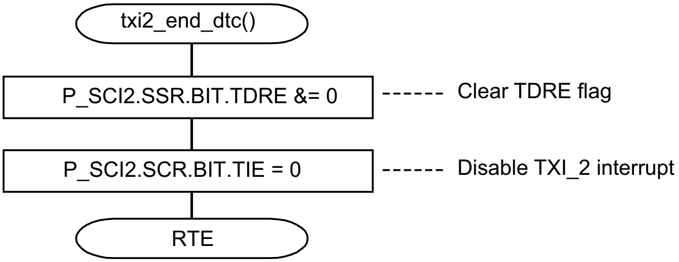
```
P_SCI2.SCR.BYTE |= 0xf0;
```

Enable serial transmit/receive operation, serial transmit/receive interrupts

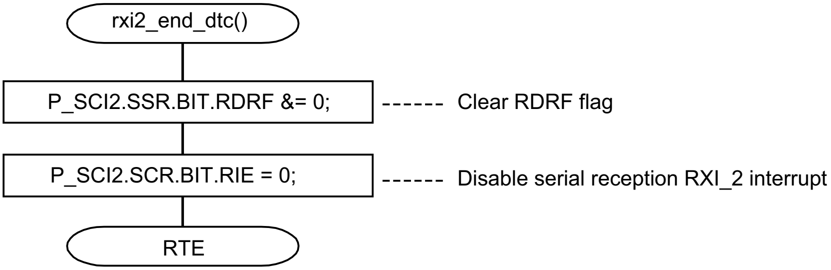
```
set_imask(0);
```

Clear interrupt mask level

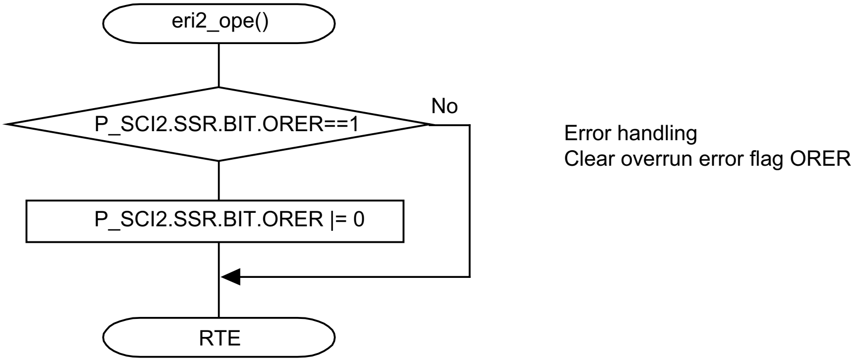
(b) Serial transmission TXI_2 interrupt handling



(c) Serial reception RXI_2 interrupt handling


rxi2_end_dtc()
P_SCI2.SSR.BIT.RDRF &= 0;
Clear RDRF flag
P_SCI2.SCR.BIT.RIE = 0;
Disable serial reception RXI_2 interrupt
RTE

(d) Serial error interrupt handling



Program Listing

```
/********************************************************/
/* SH7046F Series -SH7047- Application Note             */
/* Synchronous Serial Data Transmission with DTC        */
/*                                                      */
/* Function                                             */
/* :Serial Communication Interface(SCI)                 */
/*   Synchronous Serial Mode                            */
/*    -Transmitting/Receiving-                          */
/* :Data Transfer Controller(DTC)                       */
/*                                                      */
/* External input clock      :10MHz                     */
/* Internal CPU clock        :40MHz                     */
/* Internal peripheral clock :40MHz                     */
/*                                                      */
/* Written   :   2002/3/01  Rev.1.0                     */
/********************************************************/

#include "iodefine.h"
#include <machine.h>

/*------------ Symbol Definition ------------------------*/
struct st_dtc_tn {                  /* DTC Normal Transfer information          */
    unsigned short DTMR;            /* DTC Mode Register                        */
    unsigned short DTCRA;           /* transfer counter                         */
    unsigned short dummy1;          /* Reserved                                 */
    unsigned short dummy2;          /* Reserved                                 */
    unsigned long DTSAR;            /* source address register                  */
    unsigned long DTDAR;            /* destination address register             */
};

#define DTC_COUNT  3                                      /* DTC Transmit count */
#define DTC_1 (*(volatile struct st_dtc_tn *)0xFFFFE000)  /* Transmit DTC       */
#define DTC_2 (*(volatile struct st_dtc_tn *)0xFFFFE010)  /* Receive DTC        */


/*------------ RAM allocation Definition  --------------------*/
volatile unsigned char Txd_data[DTC_COUNT];   /* Transmit data                  */
volatile unsigned char Rxd_data[DTC_COUNT];   /* Receive data                   */


/*------------ Function Definition -------------------------*/
void main(void);
void txi2_end(void);
void rxi2_end(void);


/************************************************************/
/* main Program                                             */
/************************************************************/
void main( void )
{
    unsigned long i;

    /* Set standby mode  */
    P_STBY.MSTCR1.BIT.MSTP25 = 0;   /* Disable DTC standby mode                 */
    P_STBY.MSTCR1.BIT.MSTP24 = 0;
    P_STBY.MSTCR1.BIT.MSTP18 = 0;   /* Disable SCI2 standby mode                */

    /* Set interrupt priority level (0 to 15)  */
    P_INTC.IPRI.BIT.SCI2 = 10;      /* SCI2 interrupt level 10                  */

    /* SIC2 Transmit DTC information  */
    DTC_1.DTMR = 0x8000;
                 /* SM[1:0]=b'10;   DTSAR is incremented                        */
                 /* DM[1:0]=0;      DTDAR is  fixed                             */
                 /* MD[1:0]=0;      Transfer mode :Normal mode                  */
                 /* SZ[1:0]=0;      Byte-size transfer                          */
                 /* DTS=0;          destination is block area(no use)           */
                 /* CHNE=0;         Chain transfer is canceled                  */
                 /* DISEL=0;        Interrupt->transfer ends                    */
                 /* NMIM=0;         NMI->Terminate DTC transfer                 */
    DTC_1.DTCRA = DTC_COUNT;                    /* Transfer Count               */
    DTC_1.DTSAR = (unsigned long)&Txd_data[0];  /* set SCI2 Transmit data       */
    DTC_1.DTDAR = (unsigned long)&P_SCI2.TDR;   /* set SCI2 TDR register        */


    /* SIC2 Receive DTC information  */
    DTC_2.DTMR = 0x2000;
                /* SM[1:0]=0;       DTSAR is fixed                              */
                /* DM[1:0]=b'10;    DTDAR is incremented                        */
                /* MD[1:0]=0;       Transfer mode :Normal mode                  */
                /* SZ[1:0]=0;       Byte-size transfer                          */
                /* DTS=0;           destination is block area(no use)           */
                /* CHNE=0;          Chain transfer is canceled                  */
                /* DISEL=0;         Interrupt->transfer ends                    */
                /* NMIM=0;          NMI->Terminate DTC transfer                 */
    DTC_2.DTCRA = DTC_COUNT;        /* Transfer Count                           */
    DTC_2.DTSAR = (unsigned long)&P_SCI2.RDR;   /* set SCI2 RDR register        */
    DTC_2.DTDAR = (unsigned long)&Rxd_data[0];  /* set SCI2 Receive Buffer      */

    P_DTC.DTBR = 0xFFFF;            /* information base register                */
    /* DTC Transmit enable  */
    P_DTC.DTEE.BIT.TXI_2 |= 1;      /* interrupt sources: TXI_2(SCI2)           */
    P_DTC.DTEE.BIT.RXI_2 |= 1;      /* interrupt sources: RXI_2(SCI2)           */


    /* Initialize SCI2 clocked synchronous mode      */
    P_SCI2.SCR.BYTE = 0x01;
                /* TIE=0;           clear TIE                                   */
                /* RIE=0;           clear RIE                                   */
                /* TE=0;            clear TE                                    */
                /* RE=0;            clear RE                                    */
                /* MPIE=0;          clear MPIE,TEIE                             */
                /* TEIE=0;          clear TEIE                                  */
                /* CKE[1:0]=b'01;   clock: external ,SCK:output                 */
    P_SCI2.SMR.BYTE = 0x80;
                /* CA=1;            clocked synchronous mode                    */
                /* CKS[1:0]=b'00;   clock source =Pϕ/1                          */

    P_SCI2.SDCR.BIT.DIR = 0;        /* LSB first send                           */
    P_SCI2.BRR = 9;                 /* 1Mbps@ Pφ=40MHz                          */
    for( i=0; i < 0x100 ; i++);     /* Wait 1bit over                           */


    /* Initialize SCI2 port   */
    P_PORTA.PACRL2.BIT.PA1MD = 2;        /* set TXD2(PA1:73pin@SH7047)          */
    P_PORTA.PACRL3.BIT.PA1MD2 =1;
    P_PORTA.PACRL2.BIT.PA0MD = 2;        /* set RXD2(PA0:75pin@SH7047)          */
    P_PORTA.PACRL3.BIT.PA0MD2 = 1;
    P_PORTA.PACRL2.BIT.PA2MD = 2;        /* set SCK2(PA2:71pin@SH7047)          */
    P_PORTA.PACRL3.BIT.PA2MD2 = 1;
    P_PORTA.PAIORL.BIT.PA2IOR = 1;       /* set SCK2 output                     */


    /* set transmit data  */
    Txd_data[0] = ’a’;
    Txd_data[1] = ’b’;
    Txd_data[2] = ’c’;

    P_SCI2.SCR.BYTE |= 0xf0;     /* Transmit/Receive Enable                     */
                /* TIE=1;        TXI_2 interrupt Enable                         */
                /* RIE=1;        RXI_2,ERI_2 interrupt Enable                   */
                /* TE=1;         Transmit Enable                                */
                /* RE=1;         Receive Enable                                 */

    set_imask(0);                /* clear interrupt mask level                  */

    while(1);

}



/************************************************************/
/* SIC2:TXI_2 Interrupt                                     */
/* Transmission of DTC data transfer termination            */
/************************************************************/
#pragma interrupt(txi2_end)
void txi2_end(void)
{
    P_SCI2.SSR.BIT.TDRE &= 0;       /* TDRE=0 flag clear                        */

    P_SCI2.SCR.BIT.TIE = 0;         /* TXI_2 interrupt disable                  */
}



/************************************************************/
/* SIC2 RXI_2 Interrupt                                     */
/* Reception of DTC data transfer termination               */
/************************************************************/
#pragma interrupt(rxi2_end)
void rxi2_end(void)
{
    P_SCI2.SSR.BIT.RDRF &= 0;       /* RDRF=0 flag clear                        */
    P_SCI2.SCR.BIT.RIE = 0;          /* RXI_2,ERI_2 interrupt disable            */
}



/*************************************************************/
/* SIC2:ERI_2 Interrupt                                      */
/* SCI Reception Error                                       */
/*************************************************************/
#pragma interrupt(eri2_ope)
void eri2_ope(void)
{
    if(P_SCI2.SSR.BIT.ORER==1){      /* Overrun Error                            */
       P_SCI2.SSR.BIT.ORER |= 0;     /* ORER=0 flag clear                        */
    }
}
```

2.7 Start of A/D Conversion by MTU, and Conversion Result Storage (A/D, DTC)

Start of A/D Conversion by MTU, and Conversion Result Storage (A/D, DTC)	Functions Used: MTU, DTC, A/D

Specifications

(1) Voltages are applied to AD input channel pins for 8 channels, and A/D conversion results are stored in on-chip RAM by the data transfer controller (DTC), as shown in figure 2.37.

(2) Two A/D converter modules (AD0, AD1) are simultaneously activated by a multifunction timer pulse unit (MTU) timer ch0 TGRA_0 compare match.

(3) A/D conversion by each of the two modules (AD0, AD1) is set to 4-channel scan mode and single-cycle scanning, and the two modules perform sampling simultaneously. A/D conversion for 8 channels (AN8 through AN15) is performed in response to a single MTU compare match.

(4) The DTC uses block transfer mode. As shown in figure 2.38, the DTC is activated by an A/D module 0 A/D conversion end interrupt, and stores the conversion results for the 8 channels of A/D module 0 and A/D module 1 in on-chip RAM with a single transfer.

(5) In this task, 3 block transfers are performed, storing conversion results for 8 channels × 3 = 24 channels (48 bytes, 2 bytes per channel) in on-chip RAM.

(6) The DTC transfer conditions are shown in table 2.43.

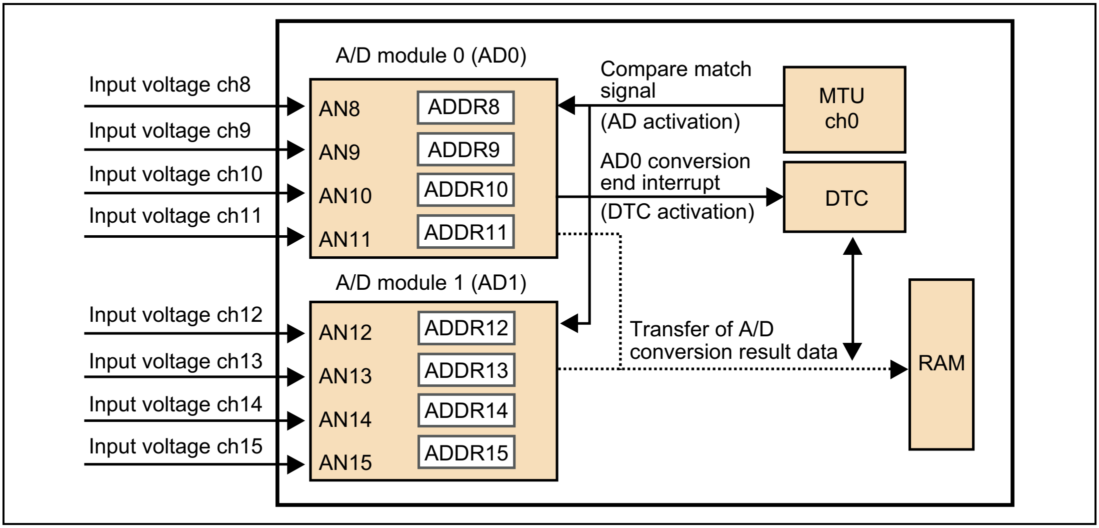


Figure 2.37 Block Diagram of AD Input Voltage Measurement

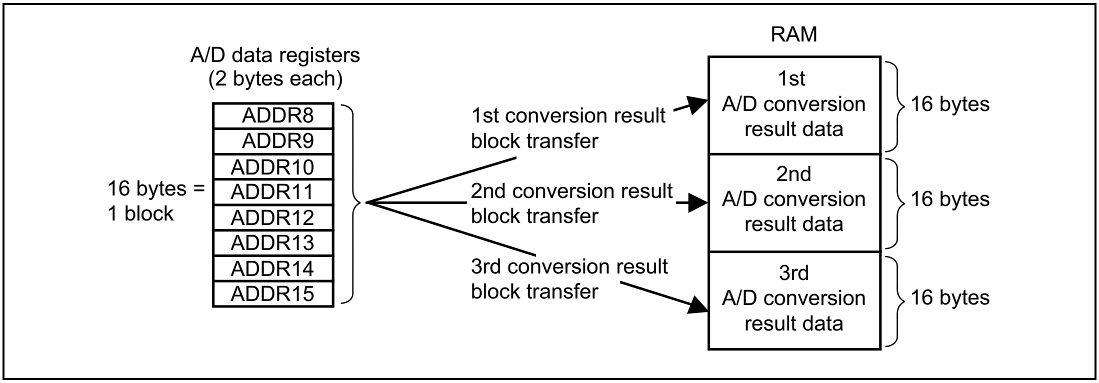


Figure 2.38 Data Transfer Using DTC

Table 2.43 DTC Transfer Conditions

Condition	Description
Transfer mode	Block transfer mode, source side (transfer source) set as block area
Number of transfers	3
Block length	8
Transfer data size	Byte transfer
Transfer source	A/D converter AD data register
Transfer destination	On-chip RAM
Transfer source address	Transfer source address incremented after transfer
Transfer destination address	Transfer destination address incremented after transfer
Activation source	Activated by MTU ch0 compare match
Interrupt handling	Interrupt to CPU enabled only at end of specified data transfer

Functions Used

(1) In this sample task, A/D conversion is started by an MTU timer ch0 compare match, and conversion results are stored in on-chip RAM by the DTC.

(a) Figure 2.39 shows a block diagram of MTU timer ch0. In this task, functions are used that generate a TGRA compare match at 128 ms intervals, and automatically start A/D conversion by means of this compare match signal without software intervention. The block diagram is explained below.

- Timer control register 0 (TCR_0) is an 8-bit register that controls TCNT. TCR_0 selects the TCNT counter clearing source, the input clock edge, and the TCNT counter clock.
- Timer mode register 0 (TMDR_0) is an 8-bit register that performs operating mode setting and buffer operation setting.
- Timer I/O control register H_0 (TIORH_0) is an 8-bit register that controls timer general register B_0 (TGRB_0) and timer general register A_0 (TGRA_0).
- Timer I/O control register L_0 (TIORL_0) is an 8-bit register that controls timer general register C_0 (TGRC_0) and timer general register D_0 (TGRD_0).
- Timer interrupt enable register 0 (TIER_0) is an 8-bit register that controls interrupt request enabling and disabling.
- Timer status register 0 (TSR_0) is an 8-bit register used for status indication.
- Timer counter 0 (TCNT_0) is a 16-bit counter. TCNT_0 must always be accessed as a 16-bit unit; 8-bit access is prohibited.
- Timer general register A_0 (TGRA_0) is a 16-bit output compare/input capture dual-function register.
- Timer general register B_0 (TGRB_0) is a 16-bit output compare/input capture dual-function register.
- Timer general register C_0 (TGRC_0) is a 16-bit output compare/input capture dual-function register. TGRC_0 can be set to operate as a buffer register in combination with TGRA_0.
- Timer general register D_0 (TGRD_0) is a 16-bit output compare/input capture dual-function register. TGRD_0 can be set to operate as a buffer register in combination with TGRB_0.
- The timer start register (TSTR) is an 8-bit register that selects TCNT operation or stoppage.

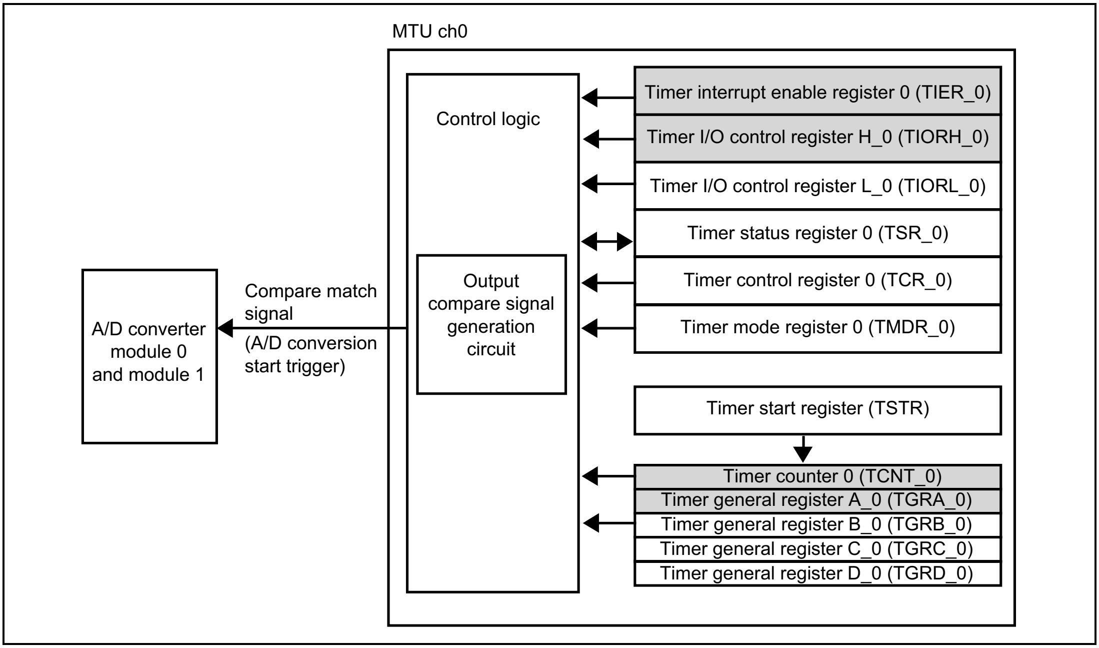

MTU ch0
Control logic
Output compare signal generation circuit
Timer interrupt enable register 0 (TIER_0)
Timer I/O control register H_0 (TIORH_0)
Timer I/O control register L_0 (TIORL_0)
Timer status register 0 (TSR_0)
Timer control register 0 (TCR_0)
Timer mode register 0 (TMDR_0)
Timer start register (TSTR)
Timer counter 0 (TCNT_0)
Timer general register A_0 (TGRA_0)
Timer general register B_0 (TGRB_0)
Timer general register C_0 (TGRC_0)
Timer general register D_0 (TGRD_0)
A/D converter module 0 and module 1
Compare match signal
(A/D conversion start trigger)

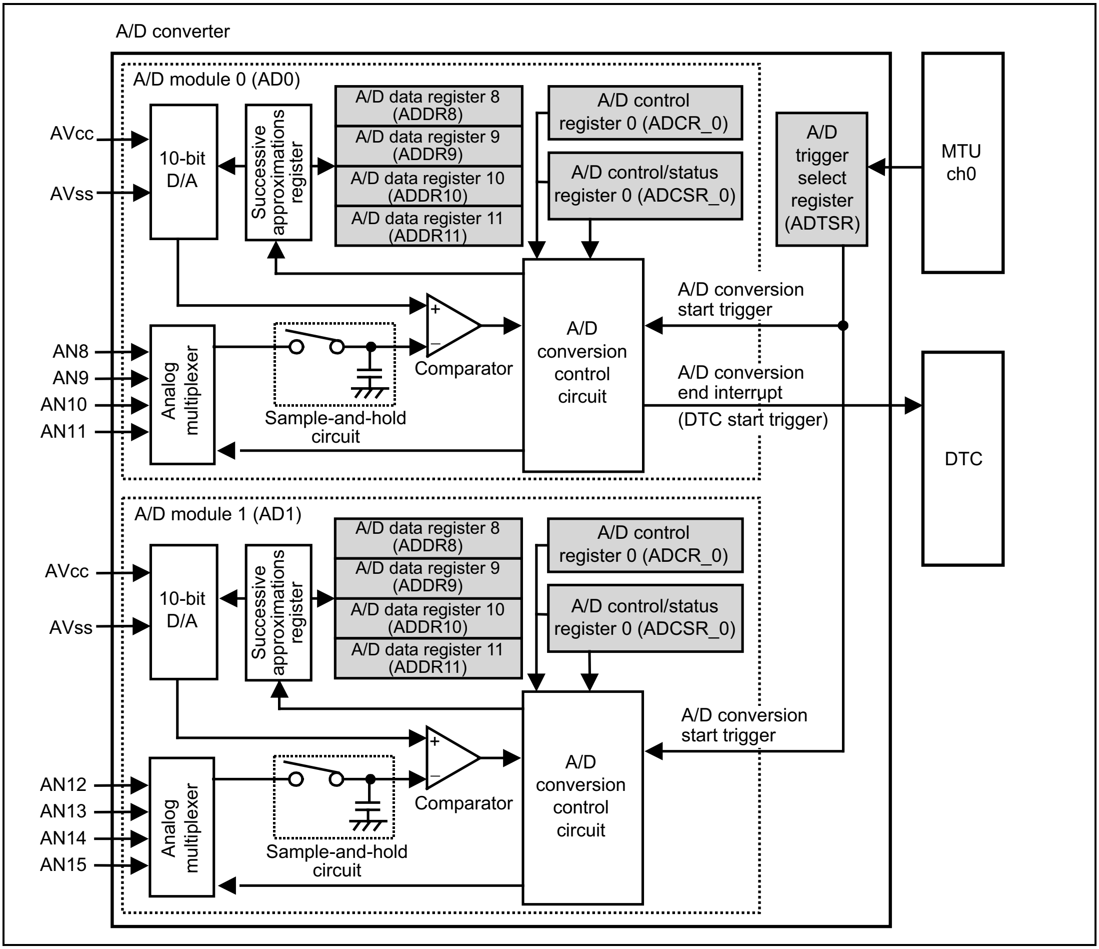
Figure 2.39 Block Diagram of MTU Timer Ch0

(b) Figure 2.40 shows a block diagram of the A/D converter. The A/D converter performs conversion from analog to digital form using the following functions. When an A/D conversion end interrupt is generated, the DTC is activated and starts transferring conversion results.

- A function that performs A/D conversion once on a number of channels (ch8 through ch11, ch12 through ch15) (4-channel, single-cycle scan mode)
- A function that simultaneously samples and converts A/D module 0 (ch8 through ch11) and A/D module 1 (ch12 through ch15) input voltages (simultaneous sampling)
- A function that starts A/D conversion using an MTU compare match as a conversion start trigger
- A function that activates the DTC when A/D conversion ends

The block diagram is explained below.

- A/D data registers 8 through 11 (ADDR8 through ADDR11) are 16-bit read-only registers for storing A/D conversion results, with 10-bit conversion data being stored in bits 15 to 6.
- A/D control/status registers 0 and 1 (ADCSR_0, 1) control A/D conversion operations
- A/D control registers 0 and 1 (ADCR_0, 1) control starting of A/D conversion by means of an external trigger and perform operating clock selection.
- The A/D trigger select register (ADTSR) enables starting of A/D conversion by means of an external trigger.



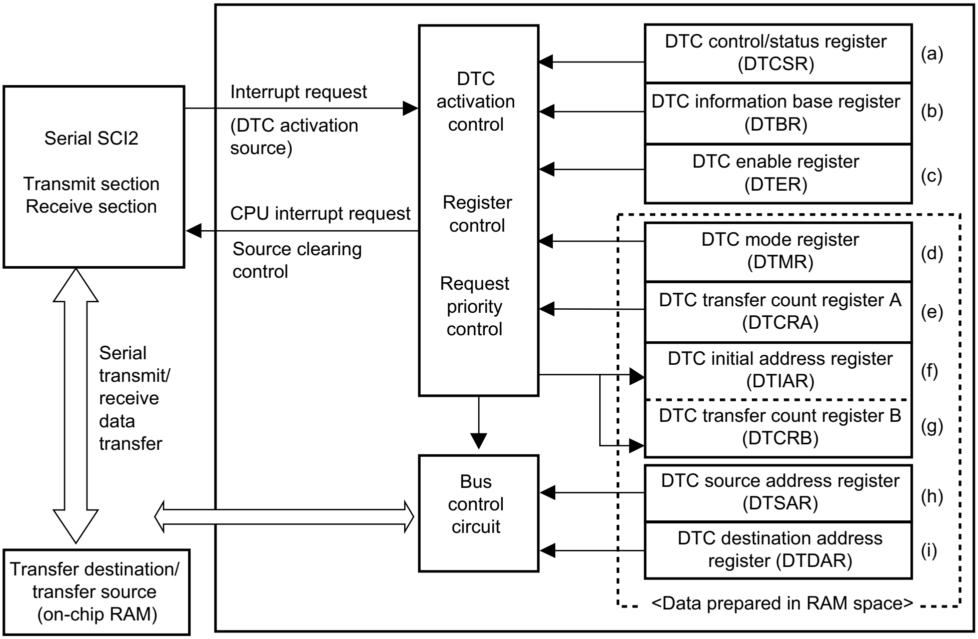
A/D converter
A/D module 0 (AD0)
AVcc
AVss
10-bit D/A
Successive approximations register
A/D data register 8 (ADDR8)
A/D data register 9 (ADDR9)
A/D data register 10 (ADDR10)
A/D data register 11 (ADDR11)
A/D control register 0 (ADCR_0)
A/D control/status register 0 (ADCSR_0)
A/D trigger select register (ADTSR)
MTU ch0
AN8
AN9
AN10
AN11
Analog multiplexer
Sample-and-hold circuit
Comparator
A/D conversion control circuit
A/D conversion start trigger
A/D conversion end interrupt
(DTC start trigger)
DTC
A/D module 1 (AD1)
AVcc
AVss
10-bit D/A
Successive approximations register
A/D data register 8 (ADDR8)
A/D data register 9 (ADDR9)
A/D data register 10 (ADDR10)
A/D data register 11 (ADDR11)
A/D control register 0 (ADCR_0)
A/D control/status register 0 (ADCSR_0)
A/D conversion start trigger
AN12
AN13
AN14
AN15
Analog multiplexer
Sample-and-hold circuit
Comparator
A/D conversion control circuit

Figure 2.40 Block Diagram of Voltage Measurement and A/D Conversion

(c) A block diagram of the DTC is shown below. Of the three DTC transfer modes — normal mode, repeat mode, and block transfer mode —this sample task uses block transfer mode to perform transfer of A/D conversion result data. DTC data transfer is performed using A/D conversion end interrupt ADI0 as the DTC activation source. The block diagram is explained below.

- The DTC mode register (DTMR) is a 16-bit register that controls the DTC's operating mode.
- The DTC source address register (DTSAR) is a 32-bit register that specifies the transfer source address of data to be transferred by the DTC. An even address should be specified in the case of word size transfer, and an address that is a multiple of 4 in the case of longword transfer.

- The DTC destination address register (DTDAR) is a 32-bit register that specifies the transfer destination address of data to be transferred by the DTC. An even address should be specified in the case of word size transfer, and an address that is a multiple of 4 in the case of longword transfer.
- The DTC initial address register (DTIAR) is a 32-bit register that specifies the transfer source/transfer destination initial address in repeat mode. In repeat mode, when the DTS bit is 1, specify the initial transfer source address in the repeat area, and when the DTS bit is 0, specify the initial transfer destination address in the repeat area.
- DTC transfer count register A (DTCRA) is a 16-bit register that specifies the number of transfers in DTC data transfer. In normal mode, DTCRA functions as a transfer counter (1 to 65,536). In repeat mode, upper 8-bit DTCRAH holds the number of transfers, and lower 8-bit DTCRAL functions as an 8-bit transfer counter. In block transfer mode, DTCRA functions as a 16-bit transfer counter.
- DTC transfer count register B (DTCRB) is a 16-bit register that specifies the block length in block transfer mode.
- The DTC enable register (DTER) is used to select the interrupt source that activates the DTC, and comprises registers DTEA through DTEF.
- The DTC control/status register (DTCSR) is a 16-bit register that sets enabling/disabling of DTC activation by software, and sets a software activation DTC vector address. DTCSR also shows the DTC transfer status.
- The DTC information base register (DTBR) is a readable/writable 16-bit register that specifies the upper 16 bits of the memory address that stores DTC transfer information. Word or longword access must be used for DTBR. If byte access is used, the register contents will be undefined in the case of a write, and an undefined value will be returned in the case of a read.
- Information of six registers — the DTC mode register (DTMR), DTC source address register (DTSAR), DTC destination address register (DTDAR), DTC initial address register (DTIAR), DTC transfer count register A (DTCRA), and DTC transfer count register B (DTCRB) — cannot be accessed directly from the CPU. When a DTC activation source occurs, the relevant register information is transferred to these registers from information of an arbitrary set of registers located in on-chip RAM and DTC transfer is performed, and when transfer ends, the contents of these registers are returned to RAM. Therefore, register information should be prepared in arbitrary on-chip RAM in the user program.


Serial SCI2
Transmit section
Receive section
Interrupt request
(DTC activation source)
CPU interrupt request
Source clearing control
DTC activation control
Register control
Request priority control
Bus control circuit
Serial transmit/ receive data transfer
Transfer destination/ transfer source (on-chip RAM)
DTC control/status register (DTCSR)
(a)
DTC information base register (DTBR)
(b)
DTC enable register (DTER)
(c)
DTC mode register (DTMR)
(d)
DTC transfer count register A (DTCRA)
(e)
DTC initial address register (DTIAR)
(f)
DTC transfer count register B (DTCRB)
(g)
DTC source address register (DTSAR)
(h)
DTC destination address register (DTDAR)
(i)
<Data prepared in RAM space>

Notes:
(a) Performs enabling/disabling of DTC activation by software, and software activation DTC vector address setting.
(b) Performs specification of the upper 16 bits of the memory address that stores DTC transfer information.
(c) Selects the interrupt source that activates the DTC; comprises six registers, DTEA through DTEF.
(d) Performs DTC operating mode setting.
(e) Specifies the number of transfers in DTC data transfer.
(f) In repeat mode, specifies the transfer source/transfer destination initial address in repeat mode. Not used in normal mode. In block transfer mode, functions as the DTCRB register.
(g) In block transfer mode, specifies the block length. Not used in normal mode. In repeat mode, functions as the DTIAR register.
(h) Specifies the transfer source address of data to be transferred by the DTC.
(i) Specifies the transfer destination address of data to be transferred by the DTC.

Figure 2.41 DTC Block Diagram

(2) Table 2.44 shows the function assignments used in this sample task.

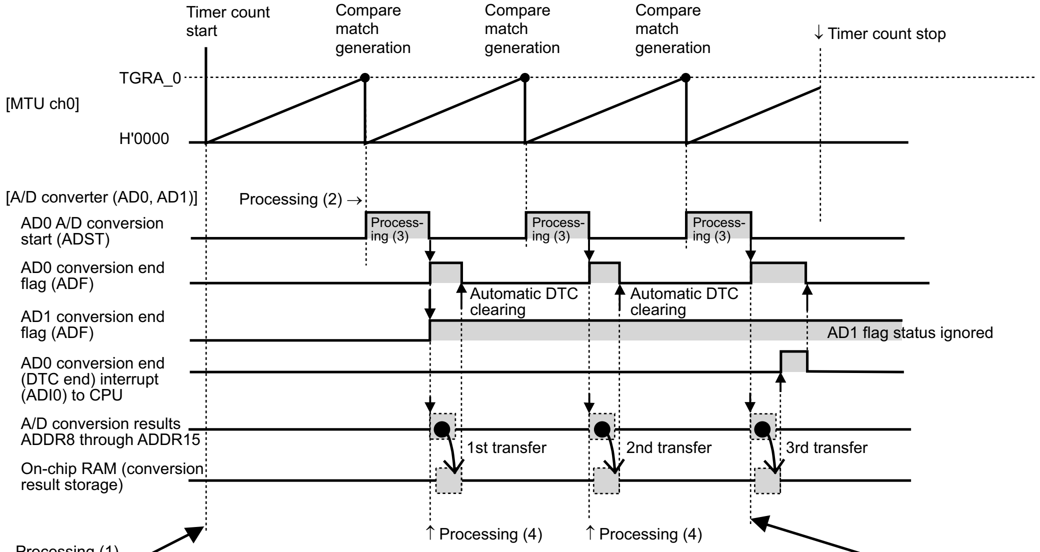
Table 2.44 Function Assignments

Function	Type	Function Assignment
AN8 to AN11	Pin	Analog measurement pins
TCR_0	MTU ch0	Selection of counter clearing source
TIER_0	MTU ch0	Enables A/D conversion start request generation
TGRA_0	MTU ch0	Sampling period setting
ADCR_0	A/D0	A/D conversion mode and measurement pin setting
ADCSR_0	A/D0	Conversion time and activation source setting
ADCR_1	A/D1	A/D conversion mode and measurement pin setting
ADCSR_1	A/D1	Conversion time and activation source setting
ADDR8□ADDR11	A/D0	A/D module 0 conversion result storage registers
ADDR12□ADDR15	A/D1	A/D module 1 conversion result storage registers
ADTSR	AD	Sets MTU trigger to start A/D conversion
DTMR	DTC	Sets DTC to block transfer mode
DTCRA	DTC	Setting of number of transfers
DTCRB	DTC	Block length setting
DTSAR	DTC	Transfer source address setting
DTDAR	DTC	Transfer destination address setting
DTBR	DTC	Setting of upper 16 bits of DTC vector
DTEC	DTC	Enables DTC activation at end of A/D conversion

Operation

(1) The principles of operation of this sample task are shown in the figure below.

A/D conversion is started by an MTU ch0 TGRA_0 compare match, and voltages at input pins AN8 through AN11 and AN12 through AN15 are converted sequentially. After conversion is completed for all the specified channels, the DTC is activated and the conversion results are transferred to RAM.



Processing (1)

Hardware Processing
None

Software Processing
Initialization
(1) MTU settings
- Enable A/D (AD0, AD1) activation by TGRA_0 compare match
- A/D (AD0, AD1) sampling period setting

(2) A/D converter settings
- Set single-cycle scan mode, 4ch scan mode as A/D conversion mode
- Set AN8 through AN11 and AN12 through AN15 as analog input channels
- Enable AD0 A/D conversion end interrupt (ADI0)
- Enable start of A/D conversion (AD0, AD1) by MTU conversion start trigger

(3) DTC initialization
(4) Start MTU/ch0 count operation

Processing (2)

Hardware Processing
(1) TGRA_0 compare match generation
(2) MTU timer counter clearing
(3) Start of A/D conversion (AD0, AD1)

Software Processing
None

Processing (3)

Hardware Processing
(1) Execute A/D conversion of AN8 through AN11 and AN12 through AN15
(2) Store conversion results sequentially in ADDR8 through ADDR11, ADDR12 through ADDR15

Software Processing
None

Processing (4)

Hardware Processing
(1) A/D conversion end interrupt (ADI0) generation, DTC activation
(2) Transfer of conversion data from A/D data registers to RAM by DTC
(3) ADI0 interrupt clearing (AD0 ADF bit clearing) by DTC (Interrupt to CPU not generated)

Software Processing
None

Processing (5)

Hardware Processing
(1) A/D conversion end interrupt (ADI0) generation, DTC activation
(2) Transfer of conversion data from A/D data registers to RAM by DTC (DTC transfer end)
(3) A/D conversion end CPU interrupt generation

Software Processing
(1) A/D conversion end interrupt handling
- Clear AD0 A/D conversion end flag (ADF)
- Stop MTU timer counter

Figure 2.42 Principles of Operation

(2) The principles of operation of DTC activation are shown in the figure below. When executing DTC transfer, the following settings should be made before an activation source occurs.

- Make DTC register information settings and place DTC register information in RAM.
- Set the lower 16 bits of the start address (32 bits) of DTC register information in the DTC vector table.
- Set the lower 16 bits of the start address (32 bits) of DTC register information in the DTC information base register.

The DTC is activated by the following processing.

- A DTC activation source interrupt is generated.
- The lower 16 bits of the start address of DTC register information are read from the address corresponding to the DTC vector table activation source.
- The upper 16 bits of the start address of DTC register information are read from the DTC information base register (DTMR).
- The 32-bit start address of DTC register information is generated from the read start address lower 16 bits and upper 16 bits.
- The start of DTC register information is read sequentially from the DTC register information start address, and data transfer is performed.

In this task, an AD0 A/D conversion end interrupt is used as the DTC activation source.

The following table shows the register information configuration in block transfer mode.

Table 2.45 DTC Register Information (Block Transfer Mode)

Setting Address	Register Name	Data Length
RF	DTC mode register (DTMR)	Word (2 bytes)
RF+2	DTC transfer count register A (DTCRA)	Word (2 bytes)
RF+6	DTC transfer count register B (DTCRB)	Word (2 bytes)
RF+8	DTC source address register (DTSAR)	Longword (4 bytes)
RF+12	DTC destination address register (DTDAR)	Longword (4 bytes)

RF: DTC register information start address (in on-chip RAM)

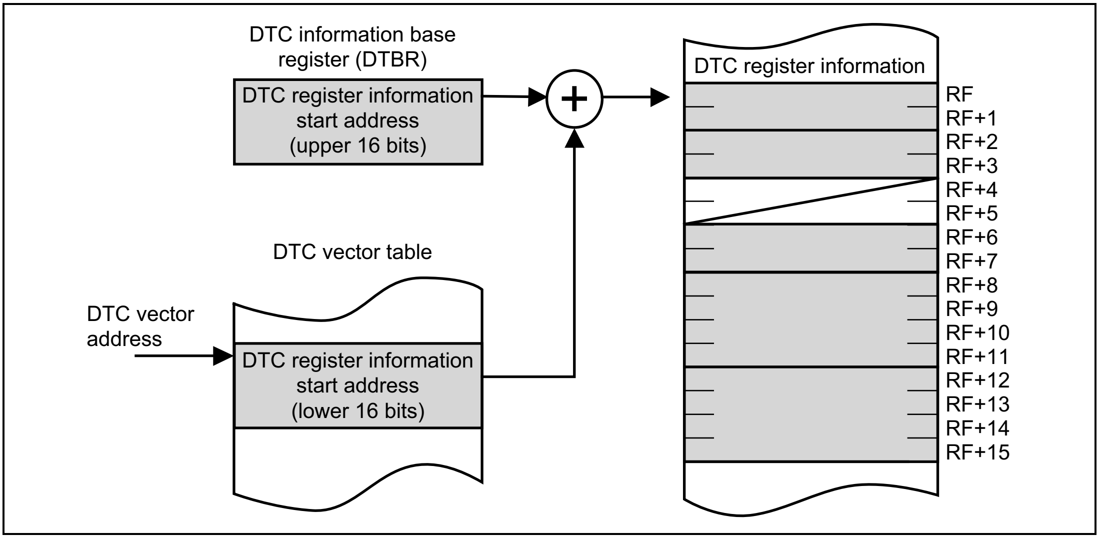

DTC information base register (DTBR)
DTC register information start address (upper 16 bits)
DTC vector table
DTC vector address
DTC register information start address (lower 16 bits)
+
DTC register information
RF
RF+1
RF+2
RF+3
RF+4
RF+5
RF+6
RF+7
RF+8
RF+9
RF+10
RF+11
RF+12
RF+13
RF+14
RF+15

Figure 2.43 Correspondence between DTC Vector Address and Transfer Information

Software

(1) Modules

The following table shows the modules used by this sample task.

Table 2.46 Modules

Module Name	Label	Function
Main routine	main	Performs initialization of MTU channel 0, A/D converter (AD0, AD1), and DTC
A/D conversion end (AD0) interrupt	ad_adi0_dtc	AD0 module A/D conversion end interrupt. Interrupt generation at end of specified number of DTC transfers

(2) Arguments

This sample task does not use any arguments.

(3) Internal Registers Used

The following table shows the internal registers used by this sample task.

Table 2.47 Internal Registers Used

Register Name		Function	Address	Set Value
	Bits		Bits	
P_STBY.MSTCR1	MSTP25 MSTP24	Module standby control register 1 DTC module standby control bits: When MSTP25 = MSTP24 = 1, module standby state When MSTP25 = MSTP24 = 0, module standby release Same value is set for MSTP25 and MSTP24	H'FFFF861C Bit 9 Bit 8	B'00
P_STBY.MSTCR2	MSTP13	Module standby control register 2 MTU module standby control bit: When MSTP13 = 1, module standby state When MSTP13 = 0, module standby release	H'FFFF861E Bit 13	0
	MSTP5	Module standby control register 2 A/D converter (AD1) module standby control bit: When MSTP5 = 1, module standby state When MSTP5 = 0, module standby release	H'FFFF861E Bit 5	0
	MSTP4	Module standby control register 2 A/D converter (AD0) module standby control bit: When MSTP4 = 1, module standby state When MSTP4 = 0, module standby release	H'FFFF861E Bit 4	0
P_INTC.IPRG	AD01	Interrupt priority register G (IPRG) A/D converter (AD0 and AD1) A/D conversion end interrupt (ADI0 and ADI1) interrupt priority level setting: When AD01 = b'1000 (8), ADI0 and ADI1 interrupts are set to priority level 8	H'FFFF8354 Bits 12 to 15	8

Register Name		Function	Address	Set Value
	Bits		Bits	
DTC_B.DTMR		DTC mode register (DTMR) DTC operating mode control setting	Located in on-chip RAM	H'a980
	SM1 SM0	Source address mode: When SM[1:0] = b'10, DTSAR is incremented after transfer	Bit 15 Bit 14	
	DM1 DM0	Destination address mode: When DM[1:0] = b'10, DTDAR is incremented after transfer	Bit 13 Bit 12	
	MD1 MD0	DTC transfer mode: When MD[1:0] = b'10, block transfer mode	Bit 11 Bit 10	
	SZ1 SZ0	DTC data transfer size: When SZ[1:0] = b'01, word (2-byte) transfer	Bit 9 Bit 8	
	DTS	DTC transfer mode select: When DTS = b'1, source side is block area	Bit 7	
	CHNE	DTC chain transfer enable: When CHNE = b'0, chain transfer is cleared	Bit 6	
	DISEL	DTC interrupt select: When DISEL = b'0, interrupt request to CPU is generated only at end of specified data transfer	Bit 5	
	NMIM	DTC NMI mode: When NMIM = b'0, DTC transfer is suspended by NMI	Bit 4	
DTC_B.DTCRA		DTC transfer count register A (DTCRA) Specifies number of transfers in DTC data transfer Set to 3 transfers	Located in on-chip RAM	H'03
DTC_B.DTCRB		DTC transfer count register B (DTCRB) Specifies block length in block transfer mode Set to 8 blocks, same as number of A/D data registers	Located in on-chip RAM	H'08
DTC_B.DTSAR		DTC source address register (DTSAR) 32-bit register that specifies transfer source address of data to be transferred by DTC	Located in on-chip RAM	P_AD.ADDR8.WORD
DTC_B.DTDAR		DTC destination address register (DTDAR) 32-bit register that specifies transfer destination address of data to be transferred by DTC	Located in on-chip RAM	Ad_data;

Register Name		Function	Address	Set Value
	Bits		Bits	
P_DTC.DTBR		DTC information base register (DTBR) Specifies upper 16 bits of memory address that stores DTC transfer information	H'FFFF8708	0xFFFF
P_DTC.DTEC	ADI0	DTC enable register E (DTEC) When set to 1, corresponding interrupt source is selected as DTC activation source: When ADI0 (DTEC6) = b'1, A/D converter (AD0) A/D conversion end interrupt (ADI0) is activation source	H'FFFF8702 Bit 6	1
P_MTU34.TSTR		MTU timer start register (TSTR) Selects TCNT operation/stoppage	H'FFFF8240	H'01
	CST4	Counter start 4: When CST4 = b'0, TCNT_4 count operation is stopped	Bit 7	
	CST3	Counter start 3: When CST3 = b'0, TCNT_3 count operation is stopped	Bit 6	
	CST2	Counter start 2: When CST2 = b'0, TCNT_2 count operation is stopped	Bit 2	
	CST1	Counter start 1: When CST1 = b'0, TCNT_1 count operation is stopped	Bit 1	
	CST0	Counter start 0: When CST0 = b'1, TCNT_0 counts	Bit 0	
P_MTU0.TCR_0		MTU timer control register 0 (TCR_0) TCNT control register	H'FFFF8260	H'23
	CCLR2 CCLR1 CCLR0	TCNT_0 counter clearing source selection: When CCLR[2:0] = b'001, TCNT clearing is performed by TGRA compare match/input capture	Bit 7 Bit 6 Bit 5	
	CKEG1 CKEG0	Input clock edge selection: When CKEG[1:0] = b'00, counting is performed on rising edge	Bit 4 Bit 3	
	TPSC2 TPSC1 TPSC0	TCNT counter clock selection: When TPSC[2:0] = b'011, counting is performed using internal clock Pϕ/64	Bit 2 Bit 1 Bit 0	

Register Name		Function	Address	Set Value
	Bits		Bits	
P_MTU0.TMDR_0		MTU timer mode register 0 (TMDR_0) Performs operating mode setting for each channel	H'FFFF8261	H'00
	BFB	Buffer operation B When BFB = b'0, TGRB and TGRD operate normally	Bit 5	
	BFA	Buffer operation A When BFA = b'0, TGRA and TGRC operate normally	Bit 4	
	MD3	Timer operating mode setting: When MD[3:0] = b'0000, timer is set to normal operating mode	Bit 3	
	MD2		Bit 2	
	MD1		Bit 1	
	MD0		Bit 0	
P_MTU0.TIORH_0		MTU timer I/O control register H_0 (TIORH_0) Controls TGR	H'FFFF8262	H'00
	IOB3	I/O control B3-0: When IOB[3:0] = b'0000, TGRB_0 is output compare match register and output is disabled for TIOC0B pin	Bit 7	
	IOB2		Bit 6	
	IOB1		Bit 5	
	IOB0		Bit 4	
	IOA3	I/O control A3-0: When IOA[3:0] = b'0000, TGRA_0 is output compare match register and output is disabled for TIOC0A pin	Bit 3	
	IOA2		Bit 2	
	IOA1		Bit 1	
	IOA0		Bit 0	
P_MTU0.TIORL_0		MTU timer I/O control register L_0 (TIORL_0) Controls TGR	H'FFFF8263	H'00
	IOD3	I/O control D3-0: When IOD[3:0] = b'0000, TGRD_0 is output compare match register and output is disabled for TIOC0D pin	Bit 7	
	IOD2		Bit 6	
	IOD1		Bit 5	
	IOD0		Bit 4	
	IOC3	I/O control C3-0: When IOC[3:0] = b'0000, TGRC_0 is output compare match register and output is disabled for TIOC0D pin	Bit 3	
	IOC2		Bit 2	
	IOC1		Bit 1	
	IOC0		Bit 0	

Register Name		Function	Address	Set Value
	Bits		Bits	
P_MTU0.TIER_0		MTU timer interrupt enable register 0 (TIER_0) Controls interrupt request enabling and disabling for each channel	H'FFFF8264	H'c0
	TTGE	A/D conversion start request enable Enables or disables A/D converter start request generation by TGRA input capture/compare match: When TTGE = b'1, A/D conversion start request generation is enabled	Bit 7	
	TGIEU	Underflow interrupt enable Enables or disables interrupt request (TCIU) by means of TCFU flag in TSR: When TGIEU = b'0, interrupt request by TCFU (TCIU) is disabled	Bit 5	
	TGIEV	Overflow interrupt enable Enables or disables interrupt request (TCIV) by means of TCFV flag in TSR: When TGIEV = b'0, interrupt request by TCFV (TCIV) is disabled	Bit 4	
	TGIED	TGR interrupt enable D Enables or disables interrupt request (TGID) by means of TGFD bit in TSR: When TGIED = b'0, interrupt request by TGFD bit (TGID) is disabled	Bit 3	
	TGIEC	TGR interrupt enable C Enables or disables interrupt request (TGIC) by means of TGFC bit in TSR: When TGIEC = b'0, interrupt request by TGFC bit (TGIC) is disabled	Bit 2	
	TGIEB	TGR interrupt enable B Enables or disables interrupt request (TGIB) by means of TGFB bit in TSR: When TGIEB = b'0, interrupt request by TGFB bit (TGIB) is disabled	Bit 1	
	TGIEA	TGR interrupt enable A Enables or disables interrupt request (TGIA) by means of TGFA bit in TSR: When TGIEA = b'1, interrupt request by TGFA bit (TGIA) is enabled	Bit 0	
P_MTU0.TCNT_0		MTU timer counter 0 (TCNT_0)	H'FFFF8266	H'0000
P_MTU0.TGRA_0		MTU timer general register A_0 (TGRA_0)	H'FFFF8268	H'9c40

Register Name		Function	Address	Set Value
	Bits		Bits	
P_AD.ADCSR_0		A/D control/status register 0 (ADCSR_0) Controls A/D conversion operations	H'FFFF8480	H'5f
	ADIE	A/D interrupt (ADI) enable: When ADIE = b'1, ADI0 interrupt is enabled	Bit 6	
	ADM1 ADM0	A/D conversion operating mode selection: When ADM[1:0] = b'01, 4-channel scan mode is selected	Bit 5 Bit 4	
	CH2 CH1 CH0	Channel select Selection of analog input channels for A/D conversion When CH[2:0] = b'111, A/D input pins AN12 through AN15 are selected	Bit 2 Bit 1 Bit 0	
P_AD.ADCR_0		A/D control register 0 (ADCR_0) Control of A/D conversion start by external trigger and operating clock selection	H'FFFF8489	H'e7
	TRGE	Trigger enable: When TRGE = b'1, start of A/D conversion by trigger is enabled	Bit 7	
	CKS1 CKS0	Clock select: When CKS[1:0] = b'11, conversion is performed using Pϕ/4	Bit 6 Bit 5	
	ADST	A/D start: When ADST = b'0, A/D converter is in standby state (in this task, conversion is started by MTU timer trigger)	Bit 4	
	ADCS	A/D continuous scan: When ADCS = b'0, single-cycle scan	Bit 3	
P_AD.ADCSR_1		A/D control/status register 1 (ADCSR_1) Controls A/D conversion operations	H'FFFF8481	H'1f
	ADIE	A/D interrupt (ADI) enable: When ADIE = b'0, ADI1 interrupt is disabled	Bit 6	
	ADM1 ADM0	A/D conversion operating mode selection: When ADM[1:0] = b'01, 4-channel scan mode is selected	Bit 5 Bit 4	
	CH2 CH1 CH0	Channel select Selection of analog input channels for A/D conversion When CH[2:0] = b'111, A/D input pins AN8 through AN11 are selected	Bit 2 Bit 1 Bit 0	

Register Name		Function	Address	Set Value
	Bits		Bits	
P_AD.ADCR_1		A/D control register 1 (ADCR_1) Control of A/D conversion start by external trigger and operating clock selection	H'FFFF8489	H'e7
	TRGE	Trigger enable: When TRGE = b'1, start of A/D conversion by trigger is enabled	Bit 7	
	CKS1 CKS0	Clock select: When CKS[1:0] = b'11, conversion is performed using Pϕ/4	Bit 6 Bit 5	
	ADST	A/D start: When ADST = b'0, A/D converter is in standby state (in this task, conversion is started by MTU timer trigger)	Bit 4	
	ADCS	A/D continuous scan: When ADCS = b'0, single-cycle scan	Bit 3	
P_AD.ADTSR		A/D trigger select register (ADTSR) Enables A/D module conversion start by trigger signal	H'FFFF87F4	H'0a
	TRG2S1 TRG2S0	A/D trigger 2 select: When TRG2S[1:0] = b'00, external trigger pin (ADTRG) or MTU trigger is selected (not used)	Bit 5 Bit 4	
	TRG1S1 TRG1S0	A/D trigger 1 select: When TRG1S[1:0] = b'10, MTU conversion start trigger is selected	Bit 3 Bit 2	
	TRG0S1 TRG0S0	A/D trigger 0 select: When TRG0S[1:0] = b'10, MTU conversion start trigger is selected	Bit 1 Bit 0	

(4) RAM Used

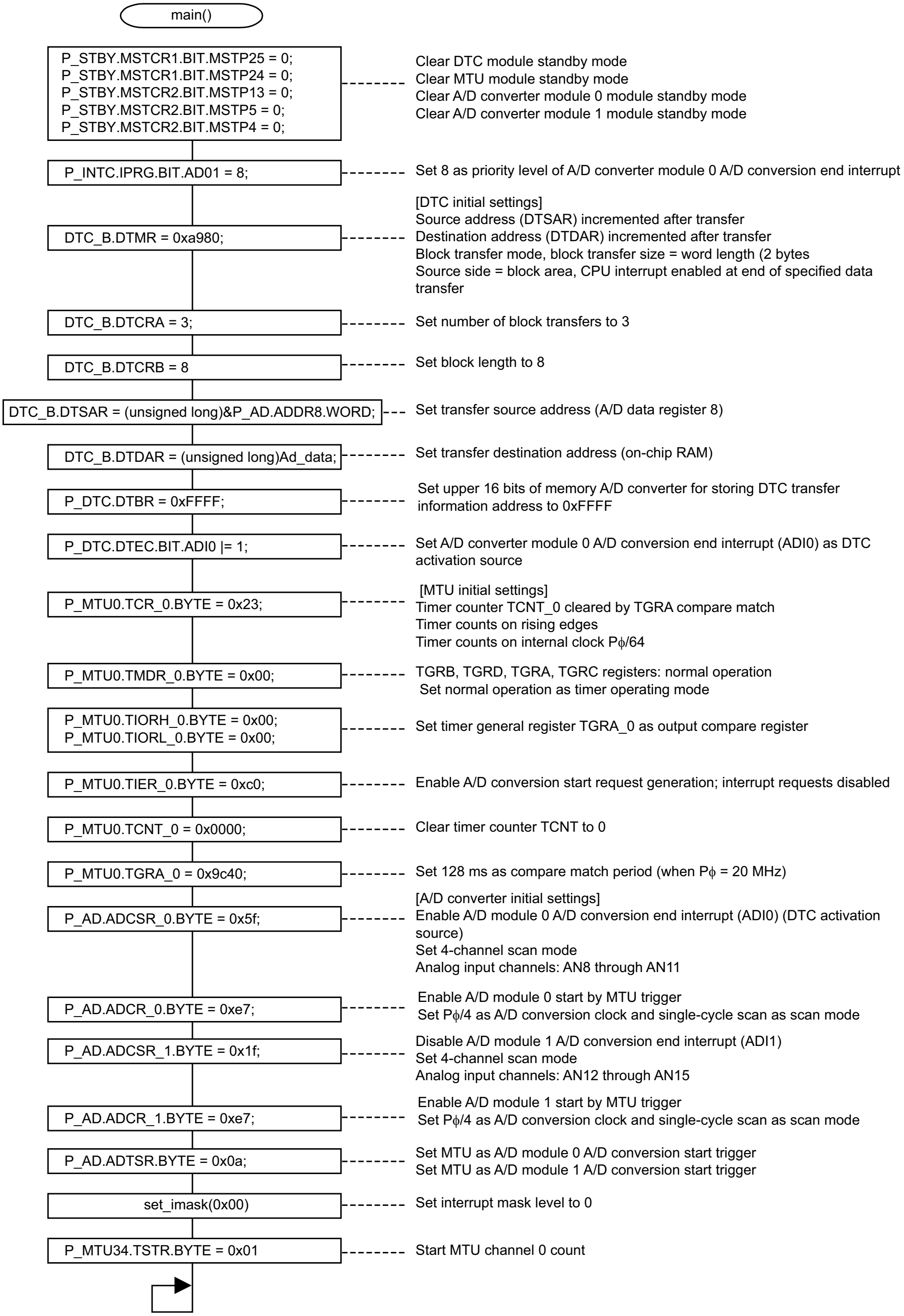
The following table shows the RAM used by this sample task.

Table 2.48 RAM Used

Label	Function	Address	Module Using RAM
Ad_data	Storage of A/D conversion data (2 bytes) 8×3 unsigned short type two-dimensional array Stores 8ch×3 set of A/D conversion result data	On-chip RAM	Main routine

Flowcharts

(a) Main processing



main()	

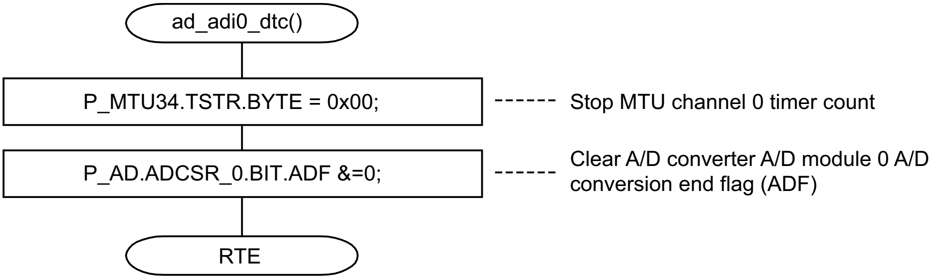
P_STBY.MSTCR1.BIT.MSTP25 = 0;
P_STBY.MSTCR1.BIT.MSTP24 = 0;
P_STBY.MSTCR2.BIT.MSTP13 = 0;
P_STBY.MSTCR2.BIT.MSTP5 = 0;
P_STBY.MSTCR2.BIT.MSTP4 = 0;
	Clear DTC module standby mode
	Clear MTU module standby mode
	Clear A/D converter module 0 module standby mode
	Clear A/D converter module 1 module standby mode

P_INTC.IPRG.BIT.AD01 = 8;	Set 8 as priority level of A/D converter module 0 A/D conversion end interrupt
DTC_B.DTMR = 0xa980;	[DTC initial settings] Source address (DTSAR) incremented after transfer Destination address (DTDAR) incremented after transfer Block transfer mode, block transfer size = word length (2 bytes Source side = block area, CPU interrupt enabled at end of specified data transfer
DTC_B.DTCRA = 3;	Set number of block transfers to 3
DTC_B.DTCRB = 8	Set block length to 8
DTC_B.DTSAR = (unsigned long)&P_AD.ADDR8.WORD;	Set transfer source address (A/D data register 8)
DTC_B.DTDAR = (unsigned long)Ad_data;	Set transfer destination address (on-chip RAM)
P_DTC.DTBR = 0xFFFF;	Set upper 16 bits of memory A/D converter for storing DTC transfer information address to 0xFFFF
P_DTC.DTEC.BIT.ADI0 |= 1;	Set A/D converter module 0 A/D conversion end interrupt (ADI0) as DTC activation source
P_MTU0.TCR_0.BYTE = 0x23;	[MTU initial settings] Timer counter TCNT_0 cleared by TGRA compare match Timer counts on rising edges Timer counts on internal clock Pϕ/64
P_MTU0.TMDR_0.BYTE = 0x00;	TGRB, TGRD, TGRA, TGRC registers: normal operation Set normal operation as timer operating mode

P_MTU0.TIORH_0.BYTE = 0x00;
P_MTU0.TIORL_0.BYTE = 0x00;
	Set timer general register TGRA_0 as output compare register

P_MTU0.TIER_0.BYTE = 0xc0;	Enable A/D conversion start request generation; interrupt requests disabled
P_MTU0.TCNT_0 = 0x0000;	Clear timer counter TCNT to 0
P_MTU0.TGRA_0 = 0x9c40;	Set 128 ms as compare match period (when Pϕ = 20 MHz)
P_AD.ADCSR_0.BYTE = 0x5f;	[A/D converter initial settings] Enable A/D module 0 A/D conversion end interrupt (ADI0) (DTC activation source) Set 4-channel scan mode Analog input channels: AN8 through AN11
P_AD.ADCR_0.BYTE = 0xe7;	Enable A/D module 0 start by MTU trigger Set Pϕ/4 as A/D conversion clock and single-cycle scan as scan mode
P_AD.ADCSR_1.BYTE = 0x1f;	Disable A/D module 1 A/D conversion end interrupt (ADI1) Set 4-channel scan mode Analog input channels: AN12 through AN15
P_AD.ADCR_1.BYTE = 0xe7;	Enable A/D module 1 start by MTU trigger Set Pϕ/4 as A/D conversion clock and single-cycle scan as scan mode
P_AD.ADTSR.BYTE = 0x0a;	Set MTU as A/D module 0 A/D conversion start trigger Set MTU as A/D module 1 A/D conversion start trigger
set_imask(0x00)	Set interrupt mask level to 0
P_MTU34.TSTR.BYTE = 0x01	Start MTU channel 0 count

(b) A/D conversion end (A/D module 0) interrupt handling


ad_adi0_dtc()
P_MTU34.TSTR.BYTE = 0x00;
Stop MTU channel 0 timer count
P_AD.ADCSR_0.BIT.ADF &=0;
Clear A/D converter A/D module 0 A/D conversion end flag (ADF)
RTE

Program Listing

```
/********************************************************/
/* SH7046F Series -SH7047- Application Note             */
/*      A/D Conversion with DTC Transmission            */
/* Function                                             */
/*  :Data transfer Controller(DTC)                      */
/*  :Multi-Function Timer Pulse Unit(MTU ch0)           */
/*  :A/D Converter(A/D ch0 ch1)                         */
/*                                                      */
/* External Input clock       :10MHz                    */
/* Internal CPU clock         :40MHz                    */
/* Internal Peripheral clock  :40MHz                    */
/*                                                      */
/* Written     :     2001/12/01  Rev.1.0                */
/*                                                      */
/********************************************************/

#include "iodefine_7047v13.1.h"
#include <machine.h>

/*------------ Symbol Definition -----------------------------------------------*/
struct st_dtc_b{                    /* DTC Block Transfer Mode information      */
    unsigned short DTMR;            /* DTC Mode Register                        */
    unsigned short DTCRA;           /* Transfer counter                         */
    unsigned short dummy;           /* Reserved                                 */
    unsigned short DTCRB;           /* Block length                             */
    unsigned long DTSAR;            /* source address register                  */
    unsigned long DTDAR;            /* destination address register             */
};

#define DTC_COUNT  3                /* DTC Transmit count                       */
#define DTC_BLOCK_LENG  8           /* DTC Block length                         */


/*------------ Function Definition ---------------------------------------------*/
void main(void);
void ad_adi0_dtc(void);


/*------------ RAM allocation Definition ---------------------------------------*/
unsigned short Ad_data[DTC_COUNT][DTC_BLOCK_LENG];   /* buffer memory           */
#define DTC_B (*(volatile struct st_dtc_b*)0xFFFFE000)
                                                    /* DTC information address  */


/************************************************************/
/* main Program        */
/************************************************************/
void main( void )
{
    /* Set standby mode  */
    P_STBY.MSTCR1.BIT.MSTP25 = 0;          /* Disable DTC standby mode          */
    P_STBY.MSTCR1.BIT.MSTP24 = 0;          /* Disable DTC standby mode          */
    P_STBY.MSTCR2.BIT.MSTP13 = 0;          /* Disable MTU standby mode          */
    P_STBY.MSTCR2.BIT.MSTP5 = 0;           /* Disable AD1 standby mode          */
    P_STBY.MSTCR2.BIT.MSTP4 = 0;          /* Disable AD0 standby mode           */


    /* Set interrupt priority level (0 to 15) */
    P_INTC.IPRG.BIT.AD01 = 8;             /* A/D ADI0,1 interrupt level8        */



    /* DTC information  */
    DTC_B.DTMR = 0xa980;            /*                                          */
                /* SM[1:0]=b’10;    DTSAR is incremented                        */
                /* DM[1:0]=b’10;    DTDAR is incremented                        */
                /* MD[1:0]=b’10;    Block transfer mode                         */
                /* SZ[1:0]=b’01;    word-size transfer                          */
                /* DTS=b’1;         Source is block area                        */
                /* CHNE=b’0;        Chain transfer is canceled                  */
                /* DISEL=b’0;       Interrupt->transfer ends                    */
                /* NMIM=b’0;        NMI->Terminate DTC transfer                 */

    DTC_B.DTCRA = DTC_COUNT;             /* DTC transfer Count                  */
    DTC_B.DTCRB = DTC_BLOCK_LENG;        /* DTC transfer Block length           */
    DTC_B.DTSAR = (unsigned long)&P_AD.ADDR8.WORD;  /* set source address       */
     DTC_B.DTDAR = (unsigned long)Ad_data;          /* set destination address  */

    P_DTC.DTBR = 0xFFFF;                 /* DTC information base register       */
              /* DTC transmit enable  */
    P_DTC.DTEC.BIT.ADI0 |= 1;            /* interrupt sources AD ch0(ADI0)      */


    /* Initialize MTU channel 0  */
    P_MTU0.TCR_0.BYTE = 0x23;            /*                                     */
                /* CCLR[2:0]=b’001;      TCNT cleared by TGRA compare match     */
                /* CKEG[1:0]=b’00;       Count at rising edge                   */
                /* TPSC[2:0]=b’011;      TCNT use Internal clock Pφ/64          */

    P_MTU0.TMDR_0.BYTE = 0x00;      /* TGRB,TGRD,TGRA,TGRD operate normally     */
                /* MD[3:0]=b’0000;  Normal timer operation mode                 */

    P_MTU0.TIORH_0.BYTE = 0x00;
                             /* TGRA_0:Output compare register, Output disabled */
          /* IOB[3:0]=b’0000;    TGRB_0:Output compare register, Output disabled*/
          /* IOA[3:0]=b’0000;    TGRA_0Output compare register, Output disabled */

    P_MTU0.TIORL_0.BYTE = 0x00;  /*                                             */
          /* IOD[3:0]=b’0000;    TGRD_0:Output compare register, Output disabled*/
          /* IOC[3:0]=b’0000;    TGRC_0Output compare register, Output disabled */

    P_MTU0.TIER_0.BYTE = 0xc0;   /*                                             */
          /* TTGE=1;             Enable, A/D conversion start by TGRA compare   */
          /* TGIEA=0;            all Interrupt requests disabled                */

    P_MTU0.TCNT_0 = 0x0000;      /* clear TCNT counter                          */
    P_MTU0.TGRA_0 = 0x9c40;      /* compare match=64ms □Pφ/64  Pφ=40MHz         */
```

```
    /* Initialize A/D chanel0,chanel1                                          */
    P_AD.ADCSR_0.BYTE = 0x5f;    /* */
           /* ADIE=b’1;          ch0 A/D ADI0 Interrupt Enable                  */
           /* ADM[1:0]=b’01;     ch0 conversion mode -> 4channel scan mode      */
           /* CH[2:0]=b111’;     Analog Input Channels = AN8 to AN11            */

    P_AD.ADCR_0.BYTE = 0xe7;     /* */
           /* TRGE=b’1;          ch0 conversion triggering is enabled           */
           /* CKS[1:0]=b’11;     ch0 Clock Select  Pφ/4                         */
           /* ADST=b’0;          ch0 stops A/D conversion                       */
           /* ADCS=b’0;          ch0 Single-cycle scan                          */
    P_AD.ADCSR_1.BYTE = 0x1f;    /* */
           /* ADIE=b’0;          ch1 A/D ADI1 Interrupt disable                 */
           /* ADM[1:0]=b’01;     ch1 conversion mode -> 4channel scan mode      */
           /* CH[2:0]=b’111;     ch1 Analog Input Channels = AN12 to AN15       */
    P_AD.ADCR_1.BYTE = 0xe7;     /* */
           /* TRGE=b’1;          ch1 conversion triggering is enabled           */
           /* CKS[1:0]=b’11;     ch1 Clock Select  Pφ/4                         */
           /* ADST=b’0;          ch1 stops A/D conversion                       */
           /* ADCS=b’0;          ch1 Single-cycle scan                          */


    P_AD.ADTSR.BYTE = 0x0a;      /* A/D ch0,ch1 conversion start by MTU trigger */
           /* TRG1S[1:0]=b’10;   ch1 conversion start by MTU trigger            */
           /* TRG0S[1:0]=b’10;   ch0 conversion start by MTU trigger            */


    set_imask(0x00);             /* clear interrupt mask level                  */


    P_MTU34.TSTR.BYTE = 0x01;    /* MTU ch0 timer count start                   */


    while(1);

}



/************************************************************/
/*       ADI interrupt                                      */
/************************************************************/
#pragma interrupt(ad_adi0_dtc)
void ad_adi0_dtc(void)
{

    P_MTU34.TSTR.BYTE = 0x00;       /* MTU ch0 timer count stop                 */

    P_AD.ADCSR_0.BIT.ADF &=0;       /* ch0 ADF□flag clear                       */

}
```

Section 3 Appendix

```
/***********************************************************************/
/*                                                                     */
/*  FILE        :iodefine.h                                            */
/*  DATE        :Fri, Nov 07, 2003                                     */
/*  DESCRIPTION :Definition of I/O Register                            */
/*  CPU TYPE    :SH7046                                                */
/*                                                                     */
/*  This file is generated by Renesas Project Generator (Ver.3.1).     */
/*                                                                     */
/***********************************************************************/



/***********************************************************************/
/*      7047 Include File                          Ver.HEW2.0_2001.12 */
/***********************************************************************/
struct st_sci {                                        /* struct SCI   */
       union {                                         /* SMR          */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char CA:1;                /*    C/A       */
                    unsigned char CHR:1;               /*    CHR       */
                    unsigned char PE:1;                /*    PE        */
                    unsigned char OE:1;                /*    O/E       */
                    unsigned char STOP:1;              /*    STOP      */
                    unsigned char MP:1;                /*    MP        */
                    unsigned char CKS:2;               /*    CKS       */
                    } BIT;                             /*              */
             } SMR;                                    /*              */
       unsigned char BRR;                              /* BRR          */
       union {                                         /* SCR          */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char TIE:1;               /*    TIE       */
                    unsigned char RIE:1;               /*    RIE       */
                    unsigned char TE:1;                /*    TE        */
                    unsigned char RE:1;                /*    RE        */
                    unsigned char MPIE:1;              /*    MPIE      */
                    unsigned char TEIE:1;              /*    TEIE      */
                    unsigned char CKE:2;               /*    CKE       */
                    } BIT;                             /*              */
             } SCR;                                    /*              */
       unsigned char TDR;                              /* TDR          */
       union {                                         /* SSR          */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char TDRE:1;              /*    TDRE      */
                    unsigned char RDRF:1;              /*    RDRF      */
                    unsigned char ORER:1;              /*    ORER      */
                    unsigned char FER:1;               /*    FER       */
                    unsigned char PER:1;               /*    PER       */
                    unsigned char TEND:1;              /*    TEND      */
                    unsigned char MPB:1;               /*    MPB       */
                    unsigned char MPBT:1;              /*    MPBT      */
                    } BIT;                             /*              */
             } SSR;                                    /*              */
       unsigned char RDR;                              /* RDR          */
       union {                                         /* SDCR         */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char :4;                  /*              */
                    unsigned char DIR:1;               /*    DIR       */
                    unsigned char :3;                  /*              */
                    } BIT;                             /*              */
             } SDCR;                                   /*              */
};                                                     /*              */
struct st_mtu34 {                                      /* struct MTU34 */
       union {                                         /* TCR_3        */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char CCLR:3;              /*    CCLR      */
                    unsigned char CKEG:2;              /*    CKEG      */
                    unsigned char TPSC:3;              /*    TPSC      */
                    } BIT;                             /*              */
             } TCR_3;                                  /*              */
       union {                                         /* TCR_4        */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char CCLR:3;              /*    CCLR      */
                    unsigned char CKEG:2;              /*    CKEG      */
                    unsigned char TPSC:3;              /*    TPSC      */
                    } BIT;                             /*              */
             } TCR_4;                                  /*              */
       union {                                         /* TMDR_3       */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char :2;                  /*              */
                    unsigned char BFB:1;               /*    BFB       */
                    unsigned char BFA:1;               /*    BFA       */
                    unsigned char MD:4;                /*    MD        */
                    } BIT;                             /*              */
             } TMDR_3;                                 /*              */
       union {                                         /* TMDR_4       */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char :2;                  /*              */
                    unsigned char BFB:1;               /*    BFB       */
                    unsigned char BFA:1;               /*    BFA       */
                    unsigned char MD:4;                /*    MD        */
                    } BIT;                             /*              */
             } TMDR_4;                                 /*              */
       union {                                         /* TIORH_3      */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char IOB:4;               /*    IOB       */
                    unsigned char IOA:4;               /*    IOA       */
                    } BIT;                             /*              */
             } TIORH_3;                                /*              */
       union {                                         /* TIORL_3      */
```

```
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char IOD:4;               /*    IOD       */
             unsigned char IOC:4;               /*    IOC       */
             } BIT;                             /*              */
      } TIORL_3;                                /*              */
union {                                         /* TIORH_4      */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char IOB:4;               /*    IOB       */
             unsigned char IOA:4;               /*    IOA       */
             } BIT;                             /*              */
      } TIORH_4;                                /*              */
union {                                         /* TIORL_4      */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char IOD:4;               /*    IOD       */
             unsigned char IOC:4;               /*    IOC       */
             } BIT;                             /*              */
      } TIORL_4;                                /*              */
union {                                         /* TIER_3       */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char TTGE:1;              /*    TTGE      */
             unsigned char :2;                  /*              */
             unsigned char TCIEV:1;             /*    TCIEV     */
             unsigned char TGIED:1;             /*    TGIED     */
             unsigned char TGIEC:1;             /*    TGIEC     */
             unsigned char TGIEB:1;             /*    TGIEB     */
             unsigned char TGIEA:1;             /*    TGIEA     */
             } BIT;                             /*              */
      } TIER_3;                                 /*              */
union {                                         /* TIER_4       */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char TTGE:1;              /*    TTGE      */
             unsigned char :2;                  /*              */
             unsigned char TCIEV:1;             /*    TCIEV     */
             unsigned char TGIED:1;             /*    TGIED     */
             unsigned char TGIEC:1;             /*    TGIEC     */
             unsigned char TGIEB:1;             /*    TGIEB     */
             unsigned char TGIEA:1;             /*    TGIEA     */
             } BIT;                             /*              */
      } TIER_4;                                 /*              */
union {                                         /* TOER         */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char :2;                  /*              */
             unsigned char OE4D:1;              /*    OE4D      */
             unsigned char OE4C:1;              /*    OE4C      */
             unsigned char OE3D:1;              /*    OE3D      */
             unsigned char OE4B:1;              /*    OE4B      */
             unsigned char OE4A:1;              /*    OE4A      */
             unsigned char OE3B:1;              /*    OE3B      */
             } BIT;                             /*              */
      } TOER;                                   /*              */
```

```
union {                                         /* TOCR         */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char :1;                  /*              */
             unsigned char PSYE:1;              /*    PSYE      */
             unsigned char :4;                  /*              */
             unsigned char OLSN:1;              /*    OLSN      */
             unsigned char OLSP:1;              /*    OLSP      */
             } BIT;                             /*              */
      } TOCR;                                   /*              */
unsigned char wk0[1];                           /*              */
union {                                         /* TGCR         */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char :1;                  /*              */
             unsigned char BDC:1;               /*    BDC       */
             unsigned char N:1;                 /*    N         */
             unsigned char P:1;                 /*    P         */
             unsigned char FB:1;                /*    FB        */
             unsigned char WF:1;                /*    WF        */
             unsigned char VF:1;                /*    VF        */
             unsigned char UF:1;                /*    UF        */
             } BIT;                             /*              */
      } TGCR;                                   /*              */
unsigned char wk1[2];                           /*              */
unsigned short TCNT_3;                          /* TCNT_3       */
unsigned short TCNT_4;                          /* TCNT_4       */
unsigned short TCDR;                            /* TCDR         */
unsigned short TDDR;                            /* TDDR         */
unsigned short TGRA_3;                          /* TGRA_3       */
unsigned short TGRB_3;                          /* TGRB_3       */
unsigned short TGRA_4;                          /* TGRA_4       */
unsigned short TGRB_4;                          /* TGRB_4       */
unsigned short TCNTS;                           /* TCNTS        */
unsigned short TCBR;                            /* TCBR         */
unsigned short TGRC_3;                          /* TGRC_3       */
unsigned short TGRD_3;                          /* TGRD_3       */
unsigned short TGRC_4;                          /* TGRC_4       */
unsigned short TGRD_4;                          /* TGRD_4       */
union {                                         /* TSR_3        */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char TDFD:1;              /*    TDFD      */
             unsigned char :2;                  /*              */
             unsigned char TCFV:1;              /*    TCFV      */
             unsigned char TGFD:1;              /*    TGFD      */
             unsigned char TGFC:1;              /*    TGFC      */
             unsigned char TGFB:1;              /*    TGFB      */
             unsigned char TGFA:1;              /*    TGFA      */
             } BIT;                             /*              */
      } TSR_3;                                  /*              */
union {                                         /* TSR_4        */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char TDFD:1;              /*    TDFD      */
             unsigned char :2;                  /*              */
```

```
                    unsigned char TCFV:1;               /*    TCFV      */
                    unsigned char TGFD:1;               /*    TGFD      */
                    unsigned char TGFC:1;               /*    TGFC      */
                    unsigned char TGFB:1;               /*    TGFB      */
                    unsigned char TGFA:1;               /*    TGFA      */
                    } BIT;                              /*              */
             } TSR_4;                                   /*              */
       unsigned char wk2[18];                           /*              */
       union {                                          /* TSTR         */
             unsigned char BYTE;                        /*  Byte Access */
             struct {                                   /*  Bit Access  */
                    unsigned char CST4:1;               /*    CST4      */
                    unsigned char CST3:1;               /*    CST3      */
                    unsigned char :3;                   /*              */
                    unsigned char CST:3;                /*    CST       */
                    } BIT;                              /*              */
             } TSTR;                                    /*              */
       union {                                          /* TSYR         */
             unsigned char BYTE;                        /*  Byte Access */
             struct {                                   /*  Bit Access  */
                    unsigned char SYNC4:1;              /*    SYNC4     */
                    unsigned char SYNC3:1;              /*    SYNC3     */
                    unsigned char :3;                   /*              */
                    unsigned char SYNC2:1;              /*    SYNC2     */
                    unsigned char SYNC1:1;              /*    SYNC1     */
                    unsigned char SYNC0:1;              /*    SYNC0     */
                    } BIT;                              /*              */
             } TSYR;                                    /*              */
};                                                      /*              */
struct st_mtu0 {                                        /* struct MTU0  */
       union {                                          /* TCR_0        */
             unsigned char BYTE;                        /*  Byte Access */
             struct {                                   /*  Bit Access  */
                    unsigned char CCLR:3;               /*    CCLR      */
                    unsigned char CKEG:2;               /*    CKEG      */
                    unsigned char TPSC:3;               /*    TPSC      */
                    } BIT;                              /*              */
             } TCR_0;                                   /*              */
       union {                                          /* TMDR_0       */
             unsigned char BYTE;                        /*  Byte Access */
             struct {                                   /*  Bit Access  */
                    unsigned char :2;                   /*              */
                    unsigned char BFB:1;                /*    BFB       */
                    unsigned char BFA:1;                /*    BFA       */
                    unsigned char MD:4;                 /*    MD        */
                    } BIT;                              /*              */
             } TMDR_0;                                  /*              */
       union {                                          /* TIORH_0      */
             unsigned char BYTE;                        /*  Byte Access */
             struct {                                   /*  Bit Access  */
                    unsigned char IOB:4;                /*    IOB       */
                    unsigned char IOA:4;                /*    IOA       */
                    } BIT;                              /*              */
             } TIORH_0;                                 /*              */
       union {                                          /* TIORL_0      */
             unsigned char BYTE;                        /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char IOD:4;               /*    IOD       */
                    unsigned char IOC:4;               /*    IOC       */
                    } BIT;                             /*              */
             } TIORL_0;                                /*              */
       union {                                         /* TIER_0       */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char TTGE:1;              /*    TTGE      */
                    unsigned char :2;                  /*              */
                    unsigned char TCIEV:1;             /*    TCIEV     */
                    unsigned char TGIED:1;             /*    TGIED     */
                    unsigned char TGIEC:1;             /*    TGIEC     */
                    unsigned char TGIEB:1;             /*    TGIEB     */
                    unsigned char TGIEA:1;             /*    TGIEA     */
                    } BIT;                             /*              */
             } TIER_0;                                 /*              */
       union {                                         /* TSR_0        */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char :3;                  /*              */
                    unsigned char TCFV:1;              /*    TCFV      */
                    unsigned char TGFD:1;              /*    TGFD      */
                    unsigned char TGFC:1;              /*    TGFC      */
                    unsigned char TGFB:1;              /*    TGFB      */
                    unsigned char TGFA:1;              /*    TGFA      */
                    } BIT;                             /*              */
             } TSR_0;                                  /*              */
       unsigned short TCNT_0;                          /* TCNT_0       */
       unsigned short TGRA_0;                          /* TGRA_0       */
       unsigned short TGRB_0;                          /* TGRB_0       */
       unsigned short TGRC_0;                          /* TGRC_0       */
       unsigned short TGRD_0;                          /* TGRD_0       */
};                                                     /*              */
struct st_mtu1 {                                       /* struct MTU1  */
       union {                                         /* TCR_1        */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char :1;                  /*              */
                    unsigned char CCLR:2;              /*    CCLR      */
                    unsigned char CKEG:2;              /*    CKEG      */
                    unsigned char TPSC:3;              /*    TPSC      */
                    } BIT;                             /*              */
             } TCR_1;                                  /*              */
       union {                                         /* TMDR_1       */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char :4;                  /*              */
                    unsigned char MD:4;                /*    MD        */
                    } BIT;                             /*              */
             } TMDR_1;                                 /*              */
       union {                                         /* TIOR_1       */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char IOB:4;               /*    IOB       */
                    unsigned char IOA:4;               /*    IOA       */
                    } BIT;                             /*              */
             } TIOR_1;                                 /*              */
       unsigned char wk0[1];                           /*              */
       union {                                         /* TIER_1       */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char TTGE:1;              /*    TTGE      */
                    unsigned char :1;                  /*              */
                    unsigned char TCIEU:1;             /*    TCIEU     */
                    unsigned char TCIEV:1;             /*    TCIEV     */
                    unsigned char :2;                  /*              */
                    unsigned char TGIEB:1;             /*    TGIEB     */
                    unsigned char TGIEA:1;             /*    TGIEA     */
                    } BIT;                             /*              */
             } TIER_1;                                 /*              */
       union {                                         /* TSR_1        */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char TCFD:1;              /*    TCFD      */
                    unsigned char :1;                  /*              */
                    unsigned char TCFU:1;              /*    TCFU      */
                    unsigned char TCFV:1;              /*    TCFV      */
                    unsigned char :2;                  /*              */
                    unsigned char TGFB:1;              /*    TGFB      */
                    unsigned char TGFA:1;              /*    TGFA      */
                    } BIT;                             /*              */
             } TSR_1;                                  /*              */
       unsigned short TCNT_1;                          /* TCNT_1       */
       unsigned short TGRA_1;                          /* TGRA_1       */
       unsigned short TGRB_1;                          /* TGRB_1       */
};                                                     /*              */
struct st_mtu2 {                                       /* struct MTU2  */
       union {                                         /* TCR_2        */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char :1;                  /*              */
                    unsigned char CCLR:2;              /*    CCLR      */
                    unsigned char CKEG:2;              /*    CKEG      */
                    unsigned char TPSC:3;              /*    TPSC      */
                    } BIT;                             /*              */
             } TCR_2;                                  /*              */
       union {                                         /* TMDR_2       */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char :4;                  /*              */
                    unsigned char MD:4;                /*    MD        */
                    } BIT;                             /*              */
             } TMDR_2;                                 /*              */
       union {                                         /* TIOR_2       */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char IOB:4;               /*    IOB       */
                    unsigned char IOA:4;               /*    IOA       */
                    } BIT;                             /*              */
             } TIOR_2;                                 /*              */
       unsigned char wk0[1];                           /*              */
       union {                                         /* TIER_2       */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char TTGE:1;              /*    TTGE      */
                    unsigned char :1;                  /*              */
                    unsigned char TCIEU:1;             /*    TCIEU     */
                    unsigned char TCIEV:1;             /*    TCIEV     */
                    unsigned char :2;                  /*              */
                    unsigned char TGIEB:1;             /*    TGIEB     */
                    unsigned char TGIEA:1;             /*    TGIEA     */
                    } BIT;                             /*              */
             } TIER_2;                                 /*              */
       union {                                         /* TSR_2        */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char TCFD:1;              /*    TCFD      */
                    unsigned char :1;                  /*              */
                    unsigned char TCFU:1;              /*    TCFU      */
                    unsigned char TCFV:1;              /*    TCFV      */
                    unsigned char :2;                  /*              */
                    unsigned char TGFB:1;              /*    TGFB      */
                    unsigned char TGFA:1;              /*    TGFA      */
                    } BIT;                             /*              */
             } TSR_2;                                  /*              */
       unsigned short TCNT_2;                          /* TCNT_2       */
       unsigned short TGRA_2;                          /* TGRA_2       */
       unsigned short TGRB_2;                          /* TGRB_2       */
};                                                     /*              */
struct st_intc {                                       /* struct INTC  */
       union {                                         /* IPRA         */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short IRQ0:4;             /*    IRQ0      */
                    unsigned short IRQ1:4;             /*    IRQ1      */
                    unsigned short IRQ2:4;             /*    IRQ2      */
                    unsigned short IRQ3:4;             /*    IRQ3      */
                    } BIT;                             /*              */
             } IPRA;                                   /*              */
       unsigned char wk0[4];                           /*              */
       union {                                         /* IPRD         */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short TGI_0:4;            /*    TGI_0     */
                    unsigned short TCI_0:4;            /*    TCI_0     */
                    unsigned short TGI_1:4;            /*    TGI_1     */
                    unsigned short TCI_1:4;            /*    TCI_1     */
                    } BIT;                             /*              */
             } IPRD;                                   /*              */
       union {                                         /* IPRE         */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short TGI_2:4;            /*    TGI_2     */
                    unsigned short TCI_2:4;            /*    TCI_2     */
                    unsigned short TGI_3:4;            /*    TGI_3     */
                    unsigned short TCI_3:4;            /*    TCI_3     */
                    } BIT;                             /*              */
```

```
      } IPRE;                                   /*              */
union {                                         /* IPRF         */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short TGI_4:4;            /*    TGI_4     */
             unsigned short TCI_4:4;            /*    TCI_4     */
             unsigned short :8;                 /*              */
             } BIT;                             /*              */
      } IPRF;                                   /*              */
union {                                         /* IPRG         */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short AD01:4;             /*    A/D0,1    */
             unsigned short DTC:4;              /*    DTC       */
             unsigned short CMT0:4;             /*    CMT0      */
             unsigned short CMT1:4;             /*    CMT1      */
             } BIT;                             /*              */
      } IPRG;                                   /*              */
union {                                         /* IPRH         */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short WDT:4;              /*    WDT       */
             unsigned short IOMTU:4;            /*    I/O(MTU)  */
             unsigned short :8;                 /*              */
             } BIT;                             /*              */
      } IPRH;                                   /*              */
union {                                         /* ICR1         */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short NMIL:1;             /*    NMIL      */
             unsigned short :6;                 /*              */
             unsigned short NMIE:1;             /*    NMIE      */
             unsigned short IRQ0S:1;            /*    IRQ0S     */
             unsigned short IRQ1S:1;            /*    IRQ1S     */
             unsigned short IRQ2S:1;            /*    IRQ2S     */
             unsigned short IRQ3S:1;            /*    IRQ3S     */
             unsigned short :4;                 /*              */
             } BIT;                             /*              */
      } ICR1;                                   /*              */
union {                                         /* ISR          */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short :8;                 /*              */
             unsigned short IRQ0F:1;            /*    IRQ0F     */
             unsigned short IRQ1F:1;            /*    IRQ1F     */
             unsigned short IRQ2F:1;            /*    IRQ2F     */
             unsigned short IRQ3F:1;            /*    IRQ3F     */
             unsigned short :4;                 /*              */
             } BIT;                             /*              */
      } ISR;                                    /*              */
union {                                         /* IPRI         */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short SCI2:4;             /*    SCI2      */
             unsigned short SCI3:4;             /*    SCI3      */
             unsigned short SCI4:4;             /*    SCI4      */
```

```
                        unsigned short MMT:4;               /*    MMT       */
                        } BIT;                              /*              */
                } IPRI;                                     /*              */
        union {                                             /* IPRJ         */
                unsigned short WORD;                        /*  Word Access */
                struct {                                    /*  Bit Access  */
                        unsigned short AD2:4;               /*    A/D2      */
                        unsigned short :12;                 /*              */
                        } BIT;                              /*              */
                } IPRJ;                                     /*              */
        union {                                             /* IPRK         */
                unsigned short WORD;                        /*  Word Access */
                struct {                                    /*  Bit Access  */
                        unsigned short IOMMT:4;             /*    I/O(MMT)  */
                        unsigned short :4;                  /*              */
                        unsigned short HCAN2:4;             /*    HCAN1     */
                        unsigned short :4;                  /*              */
                        } BIT;                              /*              */
                } IPRK;                                     /*              */
        unsigned char wk1[4];                               /*              */
        union {                                             /* ICR2         */
                unsigned short WORD;                        /*  Word Access */
                struct {                                    /*  Bit Access  */
                        unsigned short IRQ0ES:2;            /*    IRQ0ES    */
                        unsigned short IRQ1ES:2;            /*    IRQ1ES    */
                        unsigned short IRQ2ES:2;            /*    IRQ2ES    */
                        unsigned short IRQ3ES:2;            /*    IRQ3ES    */
                        unsigned short :8;                  /*              */
                        } BIT;                              /*              */
                } ICR2;                                     /*              */
};                                                          /*              */
struct st_porta {                                           /* struct PORTA */
        union {                                             /* PADRL        */
                unsigned short WORD;                        /*  Word Access */
                struct {                                    /*  Bit Access  */
                        unsigned short PA15DR:1;            /*    PA15DR    */
                        unsigned short PA14DR:1;            /*    PA14DR    */
                        unsigned short PA13DR:1;            /*    PA13DR    */
                        unsigned short PA12DR:1;            /*    PA12DR    */
                        unsigned short PA11DR:1;            /*    PA11DR    */
                        unsigned short PA10DR:1;            /*    PA10DR    */
                        unsigned short PA9DR:1;             /*    PA9DR     */
                        unsigned short PA8DR:1;             /*    PA8DR     */
                        unsigned short PA7DR:1;             /*    PA7DR     */
                        unsigned short PA6DR:1;             /*    PA6DR     */
                        unsigned short PA5DR:1;             /*    PA5DR     */
                        unsigned short PA4DR:1;             /*    PA4DR     */
                        unsigned short PA3DR:1;             /*    PA3DR     */
                        unsigned short PA2DR:1;             /*    PA2DR     */
                        unsigned short PA1DR:1;             /*    PA1DR     */
                        unsigned short PA0DR:1;             /*    PA0DR     */
                        } BIT;                              /*              */
                } PADRL;                                    /*              */
        unsigned char wk0[2];                               /*              */
        union {                                             /* PAIORL       */
                unsigned short WORD;                        /*  Word Access */
```

```
      struct {                                  /*  Bit Access  */
             unsigned short PA15IOR:1;          /*    PA15IOR   */
             unsigned short PA14IOR:1;          /*    PA14IOR   */
             unsigned short PA13IOR:1;          /*    PA13IOR   */
             unsigned short PA12IOR:1;          /*    PA12IOR   */
             unsigned short PA11IOR:1;          /*    PA11IOR   */
             unsigned short PA10IOR:1;          /*    PA10IOR   */
             unsigned short PA9IOR:1;           /*    PA9IOR    */
             unsigned short PA8IOR:1;           /*    PA8IOR    */
             unsigned short PA7IOR:1;           /*    PA7IOR    */
             unsigned short PA6IOR:1;           /*    PA6IOR    */
             unsigned short PA5IOR:1;           /*    PA5IOR    */
             unsigned short PA4IOR:1;           /*    PA4IOR    */
             unsigned short PA3IOR:1;           /*    PA3IOR    */
             unsigned short PA2IOR:1;           /*    PA2IOR    */
             unsigned short PA1IOR:1;           /*    PA1IOR    */
             unsigned short PA0IOR:1;           /*    PA0IOR    */
             } BIT;                             /*              */
      } PAIORL;                                 /*              */
unsigned char wk1[2];                           /*              */
union {                                         /* PACRL3       */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short PA15MD2:1;          /*    PA15MD2   */
             unsigned short PA14MD2:1;          /*    PA14MD2   */
             unsigned short PA13MD2:1;          /*    PA13MD2   */
             unsigned short PA12MD2:1;          /*    PA12MD2   */
             unsigned short PA11MD2:1;          /*    PA11MD2   */
             unsigned short PA10MD2:1;          /*    PA10MD2   */
             unsigned short PA9MD2:1;           /*    PA9MD2    */
             unsigned short PA8MD2:1;           /*    PA8MD2    */
             unsigned short PA7MD2:1;           /*    PA7MD2    */
             unsigned short PA6MD2:1;           /*    PA6MD2    */
             unsigned short PA5MD2:1;           /*    PA5MD2    */
             unsigned short PA4MD2:1;           /*    PA4MD2    */
             unsigned short PA3MD2:1;           /*    PA3MD2    */
             unsigned short PA2MD2:1;           /*    PA2MD2    */
             unsigned short PA1MD2:1;           /*    PA1MD2    */
             unsigned short PA0MD2:1;           /*    PA0MD2    */
             } BIT;                             /*              */
      } PACRL3;                                 /*              */
union {                                         /* PACRL1       */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short PA15MD:2;           /*    PA15MD    */
             unsigned short PA14MD:2;           /*    PA14MD    */
             unsigned short PA13MD:2;           /*    PA13MD    */
             unsigned short PA12MD:2;           /*    PA12MD    */
             unsigned short PA11MD:2;           /*    PA11MD    */
             unsigned short PA10MD:2;           /*    PA10MD    */
             unsigned short PA9MD:2;            /*    PA9MD     */
             unsigned short PA8MD:2;            /*    PA8MD     */
             } BIT;                             /*              */
      } PACRL1;                                 /*              */
union {                                         /* PACRL2       */
      unsigned short WORD;                      /*  Word Access */
```

```
             struct {                                  /*  Bit Access  */
                    unsigned short PA7MD:2;            /*    PA7MD     */
                    unsigned short PA6MD:2;            /*    PA6MD     */
                    unsigned short PA5MD:2;            /*    PA5MD     */
                    unsigned short PA4MD:2;            /*    PA4MD     */
                    unsigned short PA3MD:2;            /*    PA3MD     */
                    unsigned short PA2MD:2;            /*    PA2MD     */
                    unsigned short PA1MD:2;            /*    PA1MD     */
                    unsigned short PA0MD:2;            /*    PA0MD     */
                    } BIT;                             /*              */
             } PACRL2;                                 /*              */
};                                                     /*              */
struct st_portb {                                      /* struct PORTB */
       union {                                         /* PBDR         */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :10;                /*              */
                    unsigned short PB5DR:1;            /*    PB5DR     */
                    unsigned short PB4DR:1;            /*    PB4DR     */
                    unsigned short PB3DR:1;            /*    PB3DR     */
                    unsigned short PB2DR:1;            /*    PB2DR     */
                    unsigned short PB1DR:1;            /*    PB1DR     */
                    unsigned short PB0DR:1;            /*    PB0DR     */
                    } BIT;                             /*              */
             } PBDR;                                   /*              */
       unsigned char wk0[2];                           /*              */
       union {                                         /* PBIOR        */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :10;                /*              */
                    unsigned short PB5IOR:1;           /*    PB5IOR    */
                    unsigned short PB4IOR:1;           /*    PB4IOR    */
                    unsigned short PB3IOR:1;           /*    PB3IOR    */
                    unsigned short PB2IOR:1;           /*    PB2IOR    */
                    unsigned short PB1IOR:1;           /*    PB1IOR    */
                    unsigned short PB0IOR:1;           /*    PB0IOR    */
                    } BIT;                             /*              */
             } PBIOR;                                  /*              */
       unsigned char wk1[2];                           /*              */
       union {                                         /* PBCR1        */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :2;                 /*              */
                    unsigned short PB5MD2:1;           /*    PB5MD2    */
                    unsigned short PB4MD2:1;           /*    PB4MD2    */
                    unsigned short PB3MD2:1;           /*    PB3MD2    */
                    unsigned short PB2MD2:1;           /*    PB2MD2    */
                    unsigned short PB1MD2:1;           /*    PB1MD2    */
                    unsigned short :9;                 /*              */
                    } BIT;                             /*              */
             } PBCR1;                                  /*              */
       union {                                         /* PBCR2        */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :4;                 /*              */
                    unsigned short PB5MD:2;            /*    PB5MD     */
                    unsigned short PB4MD:2;             /*    PB4MD     */
                    unsigned short PB3MD:2;             /*    PB3MD     */
                    unsigned short PB2MD:2;             /*    PB2MD     */
                    unsigned short PB1MD:2;             /*    PB1MD     */
                    unsigned short PB0MD:2;             /*    PB0MD     */
                    } BIT;                              /*              */
             } PBCR2;                                   /*              */
};                                                      /*              */
struct st_portd {                                       /* struct PORTD */
       union {                                          /* PDDRL        */
             unsigned short WORD;                       /*  Word Access */
             struct {                                   /*  Bit Access  */
                    unsigned short :7;                  /*              */
                    unsigned short PD8DR:1;             /*    PD8DR     */
                    unsigned short PD7DR:1;             /*    PD7DR     */
                    unsigned short PD6DR:1;             /*    PD6DR     */
                    unsigned short PD5DR:1;             /*    PD5DR     */
                    unsigned short PD4DR:1;             /*    PD4DR     */
                    unsigned short PD3DR:1;             /*    PD3DR     */
                    unsigned short PD2DR:1;             /*    PD2DR     */
                    unsigned short PD1DR:1;             /*    PD1DR     */
                    unsigned short PD0DR:1;             /*    PD0DR     */
                    } BIT;                              /*              */
             } PDDRL;                                   /*              */
       unsigned char wk0[2];                            /*              */
       union {                                          /* PDIORL       */
             unsigned short WORD;                       /*  Word Access */
             struct {                                   /*  Bit Access  */
                    unsigned short :7;                  /*              */
                    unsigned short PD8IOR:1;            /*    PD8IOR    */
                    unsigned short PD7IOR:1;            /*    PD7IOR    */
                    unsigned short PD6IOR:1;            /*    PD6IOR    */
                    unsigned short PD5IOR:1;            /*    PD5IOR    */
                    unsigned short PD4IOR:1;            /*    PD4IOR    */
                    unsigned short PD3IOR:1;            /*    PD3IOR    */
                    unsigned short PD2IOR:1;            /*    PD2IOR    */
                    unsigned short PD1IOR:1;            /*    PD1IOR    */
                    unsigned short PD0IOR:1;            /*    PD0IOR    */
                    } BIT;                              /*              */
             } PDIORL;                                  /*              */
       unsigned char wk1[4];                            /*              */
       union {                                          /* PDCRL1       */
             unsigned short WORD;                       /*  Word Access */
             struct {                                   /*  Bit Access  */
                    unsigned short :7;                  /*              */
                    unsigned short PD8MD0:1;            /*    PD8MD0    */
                    unsigned short PD7MD0:1;            /*    PD7MD0    */
                    unsigned short PD6MD0:1;            /*    PD6MD0    */
                    unsigned short PD5MD0:1;            /*    PD5MD0    */
                    unsigned short PD4MD0:1;            /*    PD4MD0    */
                    unsigned short PD3MD0:1;            /*    PD3MD0    */
                    unsigned short PD2MD0:1;            /*    PD2MD0    */
                    unsigned short PD1MD0:1;            /*    PD1MD0    */
                    unsigned short PD0MD0:1;            /*    PD0MD0    */
                    } BIT;                              /*              */
             } PDCRL1;                                  /*              */
       union {                                         /* PDCRL2       */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :7;                 /*              */
                    unsigned short PD8MD1:1;           /*    PD8MD1    */
                    unsigned short PD7MD1:1;           /*    PD7MD1    */
                    unsigned short PD6MD1:1;           /*    PD6MD1    */
                    unsigned short PD5MD1:1;           /*    PD5MD1    */
                    unsigned short PD4MD1:1;           /*    PD4MD1    */
                    unsigned short PD3MD1:1;           /*    PD3MD1    */
                    unsigned short PD2MD1:1;           /*    PD2MD1    */
                    unsigned short PD1MD1:1;           /*    PD1MD1    */
                    unsigned short PD0MD1:1;           /*    PD0MD1    */
                    } BIT;                             /*              */
             } PDCRL2;                                 /*              */
};                                                     /*              */
struct st_porte {                                      /* struct PORTE */
       union {                                         /* PEDRL        */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short PE15DR:1;           /*    PE15DR    */
                    unsigned short PE14DR:1;           /*    PE14DR    */
                    unsigned short PE13DR:1;           /*    PE13DR    */
                    unsigned short PE12DR:1;           /*    PE12DR    */
                    unsigned short PE11DR:1;           /*    PE11DR    */
                    unsigned short PE10DR:1;           /*    PE10DR    */
                    unsigned short PE9DR:1;            /*    PE9DR     */
                    unsigned short PE8DR:1;            /*    PE8DR     */
                    unsigned short PE7DR:1;            /*    PE7DR     */
                    unsigned short PE6DR:1;            /*    PE6DR     */
                    unsigned short PE5DR:1;            /*    PE5DR     */
                    unsigned short PE4DR:1;            /*    PE4DR     */
                    unsigned short PE3DR:1;            /*    PE3DR     */
                    unsigned short PE2DR:1;            /*    PE2DR     */
                    unsigned short PE1DR:1;            /*    PE1DR     */
                    unsigned short PE0DR:1;            /*    PE0DR     */
                    } BIT;                             /*              */
             } PEDRL;                                  /*              */
       unsigned char wk0[2];                           /*              */
       union {                                         /* PEIORL       */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short PE15IOR:1;          /*    PE15IOR   */
                    unsigned short PE14IOR:1;          /*    PE14IOR   */
                    unsigned short PE13IOR:1;          /*    PE13IOR   */
                    unsigned short PE12IOR:1;          /*    PE12IOR   */
                    unsigned short PE11IOR:1;          /*    PE11IOR   */
                    unsigned short PE10IOR:1;          /*    PE10IOR   */
                    unsigned short PE9IOR:1;           /*    PE9IOR    */
                    unsigned short PE8IOR:1;           /*    PE8IOR    */
                    unsigned short PE7IOR:1;           /*    PE7IOR    */
                    unsigned short PE6IOR:1;           /*    PE6IOR    */
                    unsigned short PE5IOR:1;           /*    PE5IOR    */
                    unsigned short PE4IOR:1;           /*    PE4IOR    */
                    unsigned short PE3IOR:1;           /*    PE3IOR    */
                    unsigned short PE2IOR:1;           /*    PE2IOR    */
```

```
                     unsigned short PE1IOR:1;           /*    PE1IOR    */
                     unsigned short PE0IOR:1;           /*    PE0IOR    */
                     } BIT;                             /*              */
              } PEIORL;                                 /*              */
        union {                                         /* PEIORH       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Bit Access  */
                     unsigned short :10;                /*              */
                     unsigned short PE21IOR:1;          /*    PE21IOR   */
                     unsigned short PE20IOR:1;          /*    PE20IOR   */
                     unsigned short PE19IOR:1;          /*    PE19IOR   */
                     unsigned short PE18IOR:1;          /*    PE18IOR   */
                     unsigned short PE17IOR:1;          /*    PE17IOR   */
                     unsigned short PE16IOR:1;          /*    PE16IOR   */
                     } BIT;                             /*              */
              } PEIORH;                                 /*              */
        union {                                         /* PECRL1       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Bit Access  */
                     unsigned short PE15MD:2;           /*    PE15MD    */
                     unsigned short PE14MD:2;           /*    PE14MD    */
                     unsigned short PE13MD:2;           /*    PE13MD    */
                     unsigned short PE12MD:2;           /*    PE12MD    */
                     unsigned short PE11MD:2;           /*    PE11MD    */
                     unsigned short PE10MD:2;           /*    PE10MD    */
                     unsigned short PE9MD:2;            /*    PE9MD     */
                     unsigned short PE8MD:2;            /*    PE8MD     */
                     } BIT;                             /*              */
              } PECRL1;                                 /*              */
        union {                                         /* PECRL2       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Bit Access  */
                     unsigned short PE7MD:2;            /*    PE7MD     */
                     unsigned short PE6MD:2;            /*    PE6MD     */
                     unsigned short PE5MD:2;            /*    PE5MD     */
                     unsigned short PE4MD:2;            /*    PE4MD     */
                     unsigned short PE3MD:2;            /*    PE3MD     */
                     unsigned short PE2MD:2;            /*    PE2MD     */
                     unsigned short PE1MD:2;            /*    PE1MD     */
                     unsigned short PE0MD:2;            /*    PE0MD     */
                     } BIT;                             /*              */
              } PECRL2;                                 /*              */
        union {                                         /* PECRH        */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Bit Access  */
                     unsigned short :4;                 /*              */
                     unsigned short PE21MD:2;           /*    PE21MD    */
                     unsigned short PE20MD:2;           /*    PE20MD    */
                     unsigned short PE19MD:2;           /*    PE19MD    */
                     unsigned short PE18MD:2;           /*    PE18MD    */
                     unsigned short PE17MD:2;           /*    PE17MD    */
                     unsigned short PE16MD:2;           /*    PE16MD    */
                     } BIT;                             /*              */
              } PECRH;                                  /*              */
        union {                                         /* PEDRH        */
              unsigned short WORD;                      /*  Word Access */
```

```
             struct {                                  /*  Bit Access  */
                    unsigned short :10;                /*              */
                    unsigned short PE21DR:1;           /*    PE21DR    */
                    unsigned short PE20DR:1;           /*    PE20DR    */
                    unsigned short PE19DR:1;           /*    PE19DR    */
                    unsigned short PE18DR:1;           /*    PE18DR    */
                    unsigned short PE17DR:1;           /*    PE17DR    */
                    unsigned short PE16DR:1;           /*    PE16DR    */
                    } BIT;                             /*              */
             } PEDRH;                                  /*              */
};                                                     /*              */
struct st_portf {                                      /* struct PORTF */
       union {                                         /* PFDR         */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short PF15DR:1;           /*    PF15DR    */
                    unsigned short PF14DR:1;           /*    PF14DR    */
                    unsigned short PF13DR:1;           /*    PF13DR    */
                    unsigned short PF12DR:1;           /*    PF12DR    */
                    unsigned short PF11DR:1;           /*    PF11DR    */
                    unsigned short PF10DR:1;           /*    PF10DR    */
                    unsigned short PF9DR:1;            /*    PF9DR     */
                    unsigned short PF8DR:1;            /*    PF8DR     */
                    unsigned short PF7DR:1;            /*    PF7DR     */
                    unsigned short PF6DR:1;            /*    PF6DR     */
                    unsigned short PF5DR:1;            /*    PF5DR     */
                    unsigned short PF4DR:1;            /*    PF4DR     */
                    unsigned short PF3DR:1;            /*    PF3DR     */
                    unsigned short PF2DR:1;            /*    PF2DR     */
                    unsigned short PF1DR:1;            /*    PF1DR     */
                    unsigned short PF0DR:1;            /*    PF0DR     */
                    } BIT;                             /*              */
             } PFDR;                                   /*              */
};                                                     /*              */
struct st_mtu {                                        /* struct MTU   */
       union {                                         /* ICSR1        */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short POE3F:1;            /*    POE3F     */
                    unsigned short POE2F:1;            /*    POE2F     */
                    unsigned short POE1F:1;            /*    POE1F     */
                    unsigned short POE0F:1;            /*    POE0F     */
                    unsigned short :3;                 /*              */
                    unsigned short PIE:1;              /*    PIE       */
                    unsigned short POE3M:2;            /*    POE3M     */
                    unsigned short POE2M:2;            /*    POE2M     */
                    unsigned short POE1M:2;            /*    POE1M     */
                    unsigned short POE0M:2;            /*    POE0M     */
                    } BIT;                             /*              */
             } ICSR1;                                  /*              */
       union {                                         /* OCSR         */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short OSF:1;              /*    OSF       */
                    unsigned short :5;                 /*              */
                    unsigned short OCE:1;              /*    OCE       */
                    unsigned short OIE:1;              /*    OIE       */
                    unsigned short :8;                 /*              */
                    } BIT;                             /*              */
             } OCSR;                                   /*              */
};                                                     /*              */
struct st_mmt {                                        /* struct MMT   */
       union {                                         /* ICSR2        */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :1;                 /*              */
                    unsigned short POE6F:1;            /*    POE6F     */
                    unsigned short POE5F:1;            /*    POE5F     */
                    unsigned short POE4F:1;            /*    POE4F     */
                    unsigned short :3;                 /*              */
                    unsigned short PIE:1;              /*    PIE       */
                    unsigned short :2;                 /*              */
                    unsigned short POE6M:2;            /*    POE6M     */
                    unsigned short POE5M:2;            /*    POE5M     */
                    unsigned short POE4M:2;            /*    POE4M     */
                    } BIT;                             /*              */
             } ICSR2;                                  /*              */
       unsigned char wk0[1594];                        /*              */
       union {                                         /* MMT_TMDR     */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char CKS:4;               /*    CKS       */
                    unsigned char OLSN:1;              /*    OLSN      */
                    unsigned char OLSP:1;              /*    OLSP      */
                    unsigned char MD:2;                /*    MD        */
                    } BIT;                             /*              */
             } MMT_TMDR;                               /*              */
       unsigned char wk1[1];                           /*              */
       union {                                         /* TCNR         */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char TTGE:1;              /*    TTGE      */
                    unsigned char CST:1;               /*    CST       */
                    unsigned char RPRO:1;              /*    RPRO      */
                    unsigned char :3;                  /*              */
                    unsigned char TGIEN:1;             /*    TGIEN     */
                    unsigned char TGIEM:1;             /*    TGIEM     */
                    } BIT;                             /*              */
             } TCNR;                                   /*              */
       unsigned char wk2[1];                           /*              */
       union {                                         /* MMT_TSR      */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char TCFD:1;              /*    TCFD      */
                    unsigned char :5;                  /*              */
                    unsigned char TGFN:1;              /*    TGFN      */
                    unsigned char TGFM:1;              /*    TGFM      */
                    } BIT;                             /*              */
             } MMT_TSR;                                /*              */
       unsigned char wk3[1];                           /*              */
       unsigned short MMT_TCNT;                        /* MMT_TCNT     */
       unsigned short TPDR;                            /* TPDR         */
       unsigned short TPBR;                            /* TPBR         */
       unsigned short MMT_TDDR;                        /* MMT_TDDR     */
       unsigned char wk4[2];                           /*              */
       unsigned short TBRU_B;                          /* TBRU_B       */
       unsigned short TGRUU;                           /* TGRUU        */
       unsigned short TGRU;                            /* TGRU         */
       unsigned short TGRUD;                           /* TGRUD        */
       unsigned short TDCNT0;                          /* TDCNT0       */
       unsigned short TDCNT1;                          /* TDCNT1       */
       unsigned short TBRU_F;                          /* TBRU_F       */
       unsigned char wk5[2];                           /*              */
       unsigned short TBRV_B;                          /* TBRV_B       */
       unsigned short TGRVU;                           /* TGRVU        */
       unsigned short TGRV;                            /* TGRV         */
       unsigned short TGRVD;                           /* TGRVD        */
       unsigned short TDCNT2;                          /* TDCNT2       */
       unsigned short TDCNT3;                          /* TDCNT3       */
       unsigned short TBRV_F;                          /* TBRV_F       */
       unsigned char wk6[2];                           /*              */
       unsigned short TBRW_B;                          /* TBRW_B       */
       unsigned short TGRWU;                           /* TGRWU        */
       unsigned short TGRW;                            /* TGRW         */
       unsigned short TGRWD;                           /* TGRWD        */
       unsigned short TDCNT4;                          /* TDCNT4       */
       unsigned short TDCNT5;                          /* TDCNT5       */
       unsigned short TBRW_F;                          /* TBRW_F       */
};                                                     /*              */
struct st_portg {                                      /* struct PORTG */
       union {                                         /* PGDR         */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char :4;                  /*              */
                    unsigned char PG3DR:1;             /*    PG3DR     */
                    unsigned char PG2DR:1;             /*    PG2DR     */
                    unsigned char PG1DR:1;             /*    PG1DR     */
                    unsigned char PG0DR:1;             /*    PG0DR     */
                    } BIT;                             /*              */
             } PGDR;                                   /*              */
};                                                     /*              */
struct st_cmt {                                        /* struct CMT   */
       union {                                         /* CMSTR        */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :14;                /*              */
                    unsigned short STR:2;              /*    STR       */
                    } BIT;                             /*              */
             } CMSTR;                                  /*              */
       union {                                         /* CMCSR_0      */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :8;                 /*              */
                    unsigned short CMF:1;              /*    CMF       */
                    unsigned short CMIE:1;             /*    CMIE      */
                    unsigned short :4;                 /*              */
                    unsigned short CKS:2;              /*    CKS       */
                    } BIT;                             /*              */
```

```
            } CMCSR_0;                                          /*              */
        unsigned short CMCNT_0;                                 /* CMCNT_0      */
        unsigned short CMCOR_0;                                 /* CMCOR_0      */
        union {                                                 /* CMCSR_1      */
            unsigned short WORD;                                /*  Word Access */
            struct {                                            /*  Bit Access  */
                   unsigned short :8;                           /*              */
                   unsigned short CMF:1;                        /*    CMF       */
                   unsigned short CMIE:1;                       /*    CMIE      */
                   unsigned short :4;                           /*              */
                   unsigned short CKS:2;                        /*    CKS       */
                   } BIT;                                       /*              */
            } CMCSR_1;                                          /*              */
        unsigned short CMCNT_1;                                 /* CMCNT_1      */
        unsigned short CMCOR_1;                                 /* CMCOR_1      */
};                                                              /*              */
struct st_ad {                                                  /* struct AD    */
        union {                                                 /* ADDR0        */
            unsigned short WORD;                                /*  Word Access */
            struct {                                            /*  Byte Access */
                   unsigned char ADH;                           /*    AD H      */
                   unsigned char wk;                            /*              */
                   } BYTE;                                      /*              */
            struct {                                            /*  Bit Access  */
                   unsigned short AD:10;                        /*    AD        */
                   unsigned short :6;                           /*              */
                   } BIT;                                       /*              */
            } ADDR0;                                            /*              */
        union {                                                 /* ADDR1        */
            unsigned short WORD;                                /*  Word Access */
            struct {                                            /*  Byte Access */
                   unsigned char ADH;                           /*    AD H      */
                   unsigned char wk;                            /*              */
                   } BYTE;                                      /*              */
            struct {                                            /*  Bit Access  */
                   unsigned short AD:10;                        /*    AD        */
                   unsigned short :6;                           /*              */
                   } BIT;                                       /*              */
            } ADDR1;                                            /*              */
        union {                                                 /* ADDR2        */
            unsigned short WORD;                                /*  Word Access */
            struct {                                            /*  Byte Access */
                   unsigned char ADH;                           /*    AD H      */
                   unsigned char wk;                            /*              */
                   } BYTE;                                      /*              */
            struct {                                            /*  Bit Access  */
                   unsigned short AD:10;                        /*    AD        */
                   unsigned short :6;                           /*              */
                   } BIT;                                       /*              */
            } ADDR2;                                            /*              */
        union {                                                 /* ADDR3        */
            unsigned short WORD;                                /*  Word Access */
            struct {                                            /*  Byte Access */
                   unsigned char ADH;                           /*    AD H      */
                   unsigned char wk;                            /*              */
                   } BYTE;                                      /*              */
```

```
          struct {                                   /*  Bit Access  */
                 unsigned short AD:10;               /*    AD        */
                 unsigned short :6;                  /*              */
                 } BIT;                              /*              */
          } ADDR3;                                   /*              */
    union {                                          /* ADDR4       */
          unsigned short WORD;                       /*  Word Access */
          struct {                                   /*  Byte Access */
                 unsigned char ADH;                  /*    AD H      */
                 unsigned char wk;                   /*              */
                 } BYTE;                             /*              */
          struct {                                   /*  Bit Access  */
                 unsigned short AD:10;               /*    AD        */
                 unsigned short :6;                  /*              */
                 } BIT;                              /*              */
          } ADDR4;                                   /*              */
    union {                                          /* ADDR5       */
          unsigned short WORD;                       /*  Word Access */
          struct {                                   /*  Byte Access */
                 unsigned char ADH;                  /*    AD H      */
                 unsigned char wk;                   /*              */
                 } BYTE;                             /*              */
          struct {                                   /*  Bit Access  */
                 unsigned short AD:10;               /*    AD        */
                 unsigned short :6;                  /*              */
                 } BIT;                              /*              */
          } ADDR5;                                   /*              */
    union {                                          /* ADDR6       */
          unsigned short WORD;                       /*  Word Access */
          struct {                                   /*  Byte Access */
                 unsigned char ADH;                  /*    AD H      */
                 unsigned char wk;                   /*              */
                 } BYTE;                             /*              */
          struct {                                   /*  Bit Access  */
                 unsigned short AD:10;               /*    AD        */
                 unsigned short :6;                  /*              */
                 } BIT;                              /*              */
          } ADDR6;                                   /*              */
    union {                                          /* ADDR7       */
          unsigned short WORD;                       /*  Word Access */
          struct {                                   /*  Byte Access */
                 unsigned char ADH;                  /*    AD H      */
                 unsigned char wk;                   /*              */
                 } BYTE;                             /*              */
          struct {                                   /*  Bit Access  */
                 unsigned short AD:10;               /*    AD        */
                 unsigned short :6;                  /*              */
                 } BIT;                              /*              */
          } ADDR7;                                   /*              */
    union {                                          /* ADDR8       */
          unsigned short WORD;                       /*  Word Access */
          struct {                                   /*  Byte Access */
                 unsigned char ADH;                  /*    AD H      */
                 unsigned char wk;                   /*              */
                 } BYTE;                             /*              */
          struct {                                   /*  Bit Access  */
```

```
                     unsigned short AD:10;              /*    AD        */
                     unsigned short :6;                 /*              */
                     } BIT;                             /*              */
              } ADDR8;                                  /*              */
        union {                                         /* ADDR9        */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Byte Access */
                     unsigned char ADH;                 /*    AD H      */
                     unsigned char wk;                  /*              */
                     } BYTE;                            /*              */
              struct {                                  /*  Bit Access  */
                     unsigned short AD:10;              /*    AD        */
                     unsigned short :6;                 /*              */
                     } BIT;                             /*              */
              } ADDR9;                                  /*              */
        union {                                         /* ADDR10       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Byte Access */
                     unsigned char ADH;                 /*    AD H      */
                     unsigned char wk;                  /*              */
                     } BYTE;                            /*              */
              struct {                                  /*  Bit Access  */
                     unsigned short AD:10;              /*    AD        */
                     unsigned short :6;                 /*              */
                     } BIT;                             /*              */
              } ADDR10;                                 /*              */
        union {                                         /* ADDR11       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Byte Access */
                     unsigned char ADH;                 /*    AD H      */
                     unsigned char wk;                  /*              */
                     } BYTE;                            /*              */
              struct {                                  /*  Bit Access  */
                     unsigned short AD:10;              /*    AD        */
                     unsigned short :6;                 /*              */
                     } BIT;                             /*              */
              } ADDR11;                                 /*              */
        union {                                         /* ADDR12       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Byte Access */
                     unsigned char ADH;                 /*    AD H      */
                     unsigned char wk;                  /*              */
                     } BYTE;                            /*              */
              struct {                                  /*  Bit Access  */
                     unsigned short AD:10;              /*    AD        */
                     unsigned short :6;                 /*              */
                     } BIT;                             /*              */
              } ADDR12;                                 /*              */
        union {                                         /* ADDR13       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Byte Access */
                     unsigned char ADH;                 /*    AD H      */
                     unsigned char wk;                  /*              */
                     } BYTE;                            /*              */
              struct {                                  /*  Bit Access  */
                     unsigned short AD:10;              /*    AD        */
                     unsigned short :6;                 /*              */
                     } BIT;                             /*              */
              } ADDR13;                                 /*              */
        union {                                         /* ADDR14       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Byte Access */
                     unsigned char ADH;                 /*    AD H      */
                     unsigned char wk;                  /*              */
                     } BYTE;                            /*              */
              struct {                                  /*  Bit Access  */
                     unsigned short AD:10;              /*    AD        */
                     unsigned short :6;                 /*              */
                     } BIT;                             /*              */
              } ADDR14;                                 /*              */
        union {                                         /* ADDR15       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Byte Access */
                     unsigned char ADH;                 /*    AD H      */
                     unsigned char wk;                  /*              */
                     } BYTE;                            /*              */
              struct {                                  /*  Bit Access  */
                     unsigned short AD:10;              /*    AD        */
                     unsigned short :6;                 /*              */
                     } BIT;                             /*              */
              } ADDR15;                                 /*              */
        union {                                         /* ADDR16       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Byte Access */
                     unsigned char ADH;                 /*    AD H      */
                     unsigned char wk;                  /*              */
                     } BYTE;                            /*              */
              struct {                                  /*  Bit Access  */
                     unsigned short AD:10;              /*    AD        */
                     unsigned short :6;                 /*              */
                     } BIT;                             /*              */
              } ADDR16;                                 /*              */
        union {                                         /* ADDR17       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Byte Access */
                     unsigned char ADH;                 /*    AD H      */
                     unsigned char wk;                  /*              */
                     } BYTE;                            /*              */
              struct {                                  /*  Bit Access  */
                     unsigned short AD:10;              /*    AD        */
                     unsigned short :6;                 /*              */
                     } BIT;                             /*              */
              } ADDR17;                                 /*              */
        union {                                         /* ADDR18       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Byte Access */
                     unsigned char ADH;                 /*    AD H      */
                     unsigned char wk;                  /*              */
                     } BYTE;                            /*              */
              struct {                                  /*  Bit Access  */
                     unsigned short AD:10;              /*    AD        */
                     unsigned short :6;                 /*              */
```

```
             } BIT;                             /*              */
      } ADDR18;                                 /*              */
union {                                         /* ADDR19       */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Byte Access */
             unsigned char ADH;                 /*    AD H      */
             unsigned char wk;                  /*              */
             } BYTE;                            /*              */
      struct {                                  /*  Bit Access  */
             unsigned short AD:10;              /*    AD        */
             unsigned short :6;                 /*              */
             } BIT;                             /*              */
      } ADDR19;                                 /*              */
unsigned char wk0[56];                          /*              */
union {                                         /* ADCSR_0      */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char ADF:1;               /*    ADF       */
             unsigned char ADIE:1;              /*    ADIE      */
             unsigned char ADM:2;               /*    ADM       */
             unsigned char :1;                  /*              */
             unsigned char CH:3;                /*    CH        */
             } BIT;                             /*              */
      } ADCSR_0;                                /*              */
union {                                         /* ADCSR_1      */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char ADF:1;               /*    ADF       */
             unsigned char ADIE:1;              /*    ADIE      */
             unsigned char ADM:2;               /*    ADM       */
             unsigned char :1;                  /*              */
             unsigned char CH:3;                /*    CH        */
             } BIT;                             /*              */
      } ADCSR_1;                                /*              */
union {                                         /* ADCSR_2      */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char ADF:1;               /*    ADF       */
             unsigned char ADIE:1;              /*    ADIE      */
             unsigned char ADM:2;               /*    ADM       */
             unsigned char :1;                  /*              */
             unsigned char CH:3;                /*    CH        */
             } BIT;                             /*              */
      } ADCSR_2;                                /*              */
unsigned char wk1[5];                           /*              */
union {                                         /* ADCR_0       */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char TRGE:1;              /*    TRGE      */
             unsigned char CKS:2;               /*    CKS       */
             unsigned char ADST:1;              /*    ADST      */
             unsigned char ADCS:1;              /*    ADCS      */
             unsigned char :3;                  /*              */
             } BIT;                             /*              */
      } ADCR_0;                                 /*              */
union {                                         /* ADCR_1       */
```

```
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char TRGE:1;              /*    TRGE      */
                    unsigned char CKS:2;               /*    CKS       */
                    unsigned char ADST:1;              /*    ADST      */
                    unsigned char ADCS:1;              /*    ADCS      */
                    unsigned char :3;                  /*              */
                    } BIT;                             /*              */
             } ADCR_1;                                 /*              */
       union {                                         /* ADCR_2       */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char TRGE:1;              /*    TRGE      */
                    unsigned char CKS:2;               /*    CKS       */
                    unsigned char ADST:1;              /*    ADST      */
                    unsigned char ADCS:1;              /*    ADCS      */
                    unsigned char :3;                  /*              */
                    } BIT;                             /*              */
             } ADCR_2;                                 /*              */
       unsigned char wk2[873];                         /*              */
       union {                                         /* ADTSR        */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char :2;                  /*              */
                    unsigned char TRG2S:2;             /*    TRG2S     */
                    unsigned char TRG1S:2;             /*    TRG1S     */
                    unsigned char TRG0S:2;             /*    TRG0S     */
                    } BIT;                             /*              */
             } ADTSR;                                  /*              */
};                                                     /*              */
struct st_flash {                                      /* struct FLASH */
       union {                                         /* FLMCR1       */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char FWE:1;               /*    FWE       */
                    unsigned char SWE:1;               /*    SWE       */
                    unsigned char ESU:1;               /*    ESU       */
                    unsigned char PSU:1;               /*    PSU       */
                    unsigned char EV:1;                /*    EV        */
                    unsigned char PV:1;                /*    PV        */
                    unsigned char E:1;                 /*    E         */
                    unsigned char P:1;                 /*    P         */
                    } BIT;                             /*              */
             } FLMCR1;                                 /*              */
       union {                                         /* FLMCR2       */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char FLER:1;              /*    FLER      */
                    unsigned char :7;                  /*              */
                    } BIT;                             /*              */
             } FLMCR2;                                 /*              */
       union {                                         /* EBR1         */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char EB:8;                /*    EB        */
                    } BIT;                             /*              */
             } EBR1;                                   /*              */
       union {                                         /* EBR2         */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char :4;                  /*              */
                    unsigned char EB11:1;              /*    EB11      */
                    unsigned char EB10:1;              /*    EB10      */
                    unsigned char EB9:1;               /*    EB9       */
                    unsigned char EB8:1;               /*    EB8       */
                    } BIT;                             /*              */
             } EBR2;                                   /*              */
       unsigned char wk0[164];                         /*              */
       union {                                         /* RAMER        */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :12;                /*              */
                    unsigned short RAMS:1;             /*    RAMS      */
                    unsigned short RAM:3;              /*    RAM       */
                    } BIT;                             /*              */
             } RAMER;                                  /*              */
};                                                     /*              */
struct st_ubc {                                        /* struct UBC   */
       unsigned short UBARH;                           /* UBARH        */
       unsigned short UBARL;                           /* UBARL        */
       union {                                         /* UBAMRH       */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short UBM31:1;            /*    UBM31     */
                    unsigned short UBM30:1;            /*    UBM30     */
                    unsigned short UBM29:1;            /*    UBM29     */
                    unsigned short UBM28:1;            /*    UBM28     */
                    unsigned short UBM27:1;            /*    UBM27     */
                    unsigned short UBM26:1;            /*    UBM26     */
                    unsigned short UBM25:1;            /*    UBM25     */
                    unsigned short UBM24:1;            /*    UBM24     */
                    unsigned short UBM23:1;            /*    UBM23     */
                    unsigned short UBM22:1;            /*    UBM22     */
                    unsigned short UBM21:1;            /*    UBM21     */
                    unsigned short UBM20:1;            /*    UBM20     */
                    unsigned short UBM19:1;            /*    UBM19     */
                    unsigned short UBM18:1;            /*    UBM18     */
                    unsigned short UBM17:1;            /*    UBM17     */
                    unsigned short UBM16:1;            /*    UBM16     */
                    } BIT;                             /*              */
             } UBAMRH;                                 /*              */
       union {                                         /* UBAMRL       */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short UBM15:1;            /*    UBM15     */
                    unsigned short UBM14:1;            /*    UBM14     */
                    unsigned short UBM13:1;            /*    UBM13     */
                    unsigned short UBM12:1;            /*    UBM12     */
                    unsigned short UBM11:1;            /*    UBM11     */
                    unsigned short UBM10:1;            /*    UBM10     */
                    unsigned short UBM9:1;             /*    UBM9      */
                    unsigned short UBM8:1;             /*    UBM8      */
                    unsigned short UBM7:1;             /*    UBM7      */
                    unsigned short UBM6:1;             /*    UBM6      */
                    unsigned short UBM5:1;             /*    UBM5      */
                    unsigned short UBM4:1;             /*    UBM4      */
                    unsigned short UBM3:1;             /*    UBM3      */
                    unsigned short UBM2:1;             /*    UBM2      */
                    unsigned short UBM1:1;             /*    UBM1      */
                    unsigned short UBM0:1;             /*    UBM0      */
                    } BIT;                             /*              */
             } UBAMRL;                                 /*              */
       union {                                         /* UBBR         */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :8;                 /*              */
                    unsigned short CP:2;               /*    CP        */
                    unsigned short ID:2;               /*    ID        */
                    unsigned short RW:2;               /*    RW        */
                    unsigned short SZ:2;               /*    SZ        */
                    } BIT;                             /*              */
             } UBBR;                                   /*              */
       union {                                         /* UBCR         */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :13;                /*              */
                    unsigned short CKS:2;              /*    CKS       */
                    unsigned short UBID:1;             /*    UBID      */
                    } BIT;                             /*              */
             } UBCR;                                   /*              */
};                                                     /*              */
struct st_wdt {                                        /* struct WDT   */
       union {                                         /* TCSR         */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char OVF:1;               /*    OVF       */
                    unsigned char WTIT:1;              /*    WT/IT     */
                    unsigned char TME:1;               /*    TME       */
                    unsigned char :2;                  /*              */
                    unsigned char CKS:3;               /*    CKS       */
                    } BIT;                             /*              */
             } TCSR;                                   /*              */
       unsigned char TCNT;                             /* TCNT         */
       union {                                         /* RSTCSR       */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char WOVF:1;              /*    WOVF      */
                    unsigned char RSTE:1;              /*    RSTE      */
                    unsigned char RSTS:1;              /*    RSTS      */
                    unsigned char :5;                  /*              */
                    } BIT;                             /*              */
             } RSTCSR;                                 /*              */
};                                                     /*              */
struct st_stby {                                       /* struct STBY  */
       union {                                         /* SBYCR        */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char SSBY:1;              /*    SSBY      */
                    unsigned char HIZ:1;               /*    HIZ       */
                    unsigned char :5;                  /*              */
                    unsigned char IRQEL:1;             /*    IRQEL     */
                    } BIT;                             /*              */
             } SBYCR;                                  /*              */
       unsigned char wk0[3];                           /*              */
       union {                                         /* SYSCR        */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char :6;                  /*              */
                    unsigned char AUDSRST:1;           /*    AUDSRST   */
                    unsigned char RAME:1;              /*    RAME      */
                    } BIT;                             /*              */
             } SYSCR;                                  /*              */
       unsigned char wk1[3];                           /*              */
       union {                                         /* MSTCR1       */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :4;                 /*              */
                    unsigned short MSTP27:1;           /*    MSTP27    */
                    unsigned short MSTP26:1;           /*    MSTP26    */
                    unsigned short MSTP25:1;           /*    MSTP25    */
                    unsigned short MSTP24:1;           /*    MSTP24    */
                    unsigned short :3;                 /*              */
                    unsigned short MSTP20:1;           /*    MSTP20    */
                    unsigned short MSTP19:1;           /*    MSTP19    */
                    unsigned short MSTP18:1;           /*    MSTP18    */
                    unsigned short :2;                 /*              */
                    } BIT;                             /*              */
             } MSTCR1;                                 /*              */
       union {                                         /* MSTCR2       */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :1;                 /*              */
                    unsigned short MSTP14:1;           /*    MSTP14    */
                    unsigned short MSTP13:1;           /*    MSTP13    */
                    unsigned short MSTP12:1;           /*    MSTP12    */
                    unsigned short :2;                 /*              */
                    unsigned short MSTP9:1;            /*    MSTP9     */
                    unsigned short :2;                 /*              */
                    unsigned short MSTP6:1;            /*    MSTP6     */
                    unsigned short MSTP5:1;            /*    MSTP5     */
                    unsigned short MSTP4:1;            /*    MSTP4     */
                    unsigned short MSTP3:1;            /*    MSTP3     */
                    unsigned short MSTP2:1;            /*    MSTP2     */
                    unsigned short :1;                 /*              */
                    unsigned short MSTP0:1;            /*    MSTP0     */
                    } BIT;                             /*              */
             } MSTCR2;                                 /*              */
};                                                     /*              */
struct st_bsc {                                        /* struct BSC   */
       union {                                         /* BCR1         */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :1;                 /*              */
                    unsigned short MMTRWE:1;           /*    MMTRWE    */
                    unsigned short MTURWE:1;           /*    MTURWE    */
                    unsigned short :12;                /*              */
                    unsigned short A0SZ:1;             /*    A0SZ      */
                    } BIT;                             /*              */
             } BCR1;                                   /*              */
       union {                                         /* BCR2         */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :6;                 /*              */
                    unsigned short IW:2;               /*    IW        */
                    unsigned short :3;                 /*              */
                    unsigned short CW0:1;              /*    CW0       */
                    unsigned short :3;                 /*              */
                    unsigned short SW0:1;              /*    SW0       */
                    } BIT;                             /*              */
             } BCR2;                                   /*              */
       union {                                         /* WCR1         */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :12;                /*              */
                    unsigned short W:4;                /*    W         */
                    } BIT;                             /*              */
             } WCR1;                                   /*              */
};                                                     /*              */
struct st_dtc {                                        /* struct DTC   */
       union {                                         /* DTEA         */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char TGI4A:1;             /*              */
                    unsigned char TGI4B:1;             /*              */
                    unsigned char TGI4C:1;             /*              */
                    unsigned char TGI4D:1;             /*              */
                    unsigned char TGI4V:1;             /*              */
                    unsigned char TGI3A:1;             /*              */
                    unsigned char TGI3B:1;             /*              */
                    unsigned char TGI3C:1;             /*              */
                    } BIT;                             /*              */
             } DTEA;                                   /*              */
       union {                                         /* DTEB         */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char TGI3D:1;             /*              */
                    unsigned char TGI2A:1;             /*              */
                    unsigned char TGI2B:1;             /*              */
                    unsigned char TGI1A:1;             /*              */
                    unsigned char TGI1B:1;             /*              */
                    unsigned char TGI0A:1;             /*              */
                    unsigned char TGI0B:1;             /*              */
                    unsigned char TGI0C:1;             /*              */
                    } BIT;                             /*              */
             } DTEB;                                   /*              */
       union {                                         /* DTEC         */
             unsigned char BYTE;                       /*  Byte Access */
             struct {                                  /*  Bit Access  */
                    unsigned char TGI0D:1;             /*              */
                    unsigned char ADI0:1;              /*              */
             unsigned char IRQ0:1;              /*              */
             unsigned char IRQ1:1;              /*              */
             unsigned char IRQ2:1;              /*              */
             unsigned char IRQ3:1;              /*              */
             unsigned char b1:1;                /*              */
             unsigned char b0:1;                /*              */
             } BIT;                             /*              */
      } DTEC;                                   /*              */
union {                                         /* DTED         */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char b7:1;                /*              */
             unsigned char b6:1;                /*              */
             unsigned char CMI0:1;              /*              */
             unsigned char CMI1:1;              /*              */
             unsigned char b3:1;                /*              */
             unsigned char b2:1;                /*              */
             unsigned char b1:1;                /*              */
             unsigned char b0:1;                /*              */
             } BIT;                             /*              */
      } DTED;                                   /*              */
unsigned char wk0[2];                           /*              */
union {                                         /* DTCSR        */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short :5;                 /*              */
             unsigned short NMIF:1;             /*    NMIF      */
             unsigned short AE:1;               /*    AE        */
             unsigned short SWDTE:1;            /*    SWDTE     */
             unsigned short DTVEC7:1;           /*    DTVEC7    */
             unsigned short DTVEC6:1;           /*    DTVEC6    */
             unsigned short DTVEC5:1;           /*    DTVEC5    */
             unsigned short DTVEC4:1;           /*    DTVEC4    */
             unsigned short DTVEC3:1;           /*    DTVEC3    */
             unsigned short DTVEC2:1;           /*    DTVEC2    */
             unsigned short DTVEC1:1;           /*    DTVEC1    */
             unsigned short DTVEC0:1;           /*    DTVEC0    */
             } BIT;                             /*              */
      } DTCSR;                                  /*              */
unsigned short DTBR;                            /* DTBR         */
unsigned char wk1[6];                           /*              */
union {                                         /* DTEE         */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char b7:1;                /*              */
             unsigned char b6:1;                /*              */
             unsigned char ADI1:1;              /*              */
             unsigned char ADI2:1;              /*              */
             unsigned char RXI_2:1;             /*              */
             unsigned char TXI_2:1;             /*              */
             unsigned char RXI_3:1;             /*              */
             unsigned char TXI_3:1;             /*              */
             } BIT;                             /*              */
      } DTEE;                                   /*              */
union {                                         /* DTEF         */
      unsigned char BYTE;                       /*  Byte Access */
```

```
             struct {                                  /*  Bit Access  */
                    unsigned char RXI_4:1;             /*              */
                    unsigned char TXI_4:1;             /*              */
                    unsigned char TGN:1;               /*              */
                    unsigned char TGM:1;               /*              */
                    unsigned char b3:1;                /*              */
                    unsigned char RM1:1;               /*              */
                    unsigned char b1:1;                /*              */
                    unsigned char b0:1;                /*              */
                    } BIT;                             /*              */
             } DTEF;                                   /*              */
};                                                     /*              */
struct st_hudi {                                       /* struct HUDI  */
       union {                                         /* SDIR         */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short TS:4;               /*    TS        */
                    unsigned short :12;                /*              */
                    } BIT;                             /*              */
             } SDIR;                                   /*              */
       union {                                         /* SDSR         */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :15;                /*              */
                    unsigned short SDTRF:1;            /*    SDTRF     */
                    } BIT;                             /*              */
             } SDSR;                                   /*              */
       unsigned short SDDRH;                           /* SDDRH        */
       unsigned short SDDRL;                           /* SDDRL        */
};                                                     /*              */
struct st_hcan2 {                                      /* struct HCAN2 */
       union {                                         /* MCR          */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :8;                 /*              */
                    unsigned short MCR7:1;             /*    MCR7      */
                    unsigned short :1;                 /*              */
                    unsigned short MCR5:1;             /*    MCR5      */
                    unsigned short :2;                 /*              */
                    unsigned short MCR2:1;             /*    MCR2      */
                    unsigned short MCR1:1;             /*    MCR1      */
                    unsigned short MCR0:1;             /*    MCR0      */
                    } BIT;                             /*              */
             } MCR;                                    /*              */
       union {                                         /* GSR          */
             unsigned short WORD;                      /*  Word Access */
             struct {                                  /*  Bit Access  */
                    unsigned short :10;                /*              */
                    unsigned short GSR5:1;             /*    GSR5      */
                    unsigned short GSR4:1;             /*    GSR4      */
                    unsigned short GSR3:1;             /*    GSR3      */
                    unsigned short GSR2:1;             /*    GSR2      */
                    unsigned short GSR1:1;             /*    GSR1      */
                    unsigned short GSR0:1;             /*    GSR0      */
                    } BIT;                             /*              */
             } GSR;                                    /*              */
```

```
union {                                         /* HCAN2_BCR1   */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short TSG1:4;             /*    TSG1      */
             unsigned short :1;                 /*              */
             unsigned short TSG2:3;             /*    TSG2      */
             unsigned short :2;                 /*              */
             unsigned short SJW:2;              /*    SJW       */
             unsigned short :3;                 /*              */
             unsigned short BSP:1;              /*    BSP       */
             } BIT;                             /*              */
      } HCAN2_BCR1;                             /*              */
union {                                         /* HCAN2_BCR0   */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short :8;                 /*              */
             unsigned short BRP:8;              /*    BRP       */
             } BIT;                             /*              */
      } HCAN2_BCR0;                             /*              */
union {                                         /* IRR          */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short IRR15:1;            /*    IRR15     */
             unsigned short IRR14:1;            /*    IRR14     */
             unsigned short IRR13:1;            /*    IRR13     */
             unsigned short IRR12:1;            /*    IRR12     */
             unsigned short :2;                 /*              */
             unsigned short IRR9:1;             /*    IRR9      */
             unsigned short IRR8:1;             /*    IRR8      */
             unsigned short IRR7:1;             /*    IRR7      */
             unsigned short IRR6:1;             /*    IRR6      */
             unsigned short IRR5:1;             /*    IRR5      */
             unsigned short IRR4:1;             /*    IRR4      */
             unsigned short IRR3:1;             /*    IRR3      */
             unsigned short IRR2:1;             /*    IRR2      */
             unsigned short IRR1:1;             /*    IRR1      */
             unsigned short IRR0:1;             /*    IRR0      */
             } BIT;                             /*              */
      } IRR;                                    /*              */
union {                                         /* IMR          */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short IMR15:1;            /*    IMR15     */
             unsigned short IMR14:1;            /*    IMR14     */
             unsigned short IMR13:1;            /*    IMR13     */
             unsigned short IMR12:1;            /*    IMR12     */
             unsigned short :2;                 /*              */
             unsigned short IMR9:1;             /*    IMR9      */
             unsigned short IMR8:1;             /*    IMR8      */
             unsigned short IMR7:1;             /*    IMR7      */
             unsigned short IMR6:1;             /*    IMR6      */
             unsigned short IMR5:1;             /*    IMR5      */
             unsigned short IMR4:1;             /*    IMR4      */
             unsigned short IMR3:1;             /*    IMR3      */
             unsigned short IMR2:1;             /*    IMR2      */
             unsigned short IMR1:1;             /*    IMR1      */
             unsigned short :1;                 /*              */
             } BIT;                             /*              */
      } IMR;                                    /*              */
unsigned char TEC;                              /* TEC          */
unsigned char REC;                              /* REC          */
unsigned char wk0[18];                          /*              */
union {                                         /* TXPR1        */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short TXPR31:1;           /*    TXPR31    */
             unsigned short TXPR30:1;           /*    TXPR30    */
             unsigned short TXPR29:1;           /*    TXPR29    */
             unsigned short TXPR28:1;           /*    TXPR28    */
             unsigned short TXPR27:1;           /*    TXPR27    */
             unsigned short TXPR26:1;           /*    TXPR26    */
             unsigned short TXPR25:1;           /*    TXPR25    */
             unsigned short TXPR24:1;           /*    TXPR24    */
             unsigned short TXPR23:1;           /*    TXPR23    */
             unsigned short TXPR22:1;           /*    TXPR22    */
             unsigned short TXPR21:1;           /*    TXPR21    */
             unsigned short TXPR20:1;           /*    TXPR20    */
             unsigned short TXPR19:1;           /*    TXPR19    */
             unsigned short TXPR18:1;           /*    TXPR18    */
             unsigned short TXPR17:1;           /*    TXPR17    */
             unsigned short TXPR16:1;           /*    TXPR16    */
             } BIT;                             /*              */
      } TXPR1;                                  /*              */
union {                                         /* TXPR0        */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short TXPR15:1;           /*    TXPR15    */
             unsigned short TXPR14:1;           /*    TXPR14    */
             unsigned short TXPR13:1;           /*    TXPR13    */
             unsigned short TXPR12:1;           /*    TXPR12    */
             unsigned short TXPR11:1;           /*    TXPR11    */
             unsigned short TXPR10:1;           /*    TXPR10    */
             unsigned short TXPR9:1;            /*    TXPR9     */
             unsigned short TXPR8:1;            /*    TXPR8     */
             unsigned short TXPR7:1;            /*    TXPR7     */
             unsigned short TXPR6:1;            /*    TXPR6     */
             unsigned short TXPR5:1;            /*    TXPR5     */
             unsigned short TXPR4:1;            /*    TXPR4     */
             unsigned short TXPR3:1;            /*    TXPR3     */
             unsigned short TXPR2:1;            /*    TXPR2     */
             unsigned short TXPR1:1;            /*    TXPR1     */
             unsigned short :1;                 /*              */
             } BIT;                             /*              */
      } TXPR0;                                  /*              */
unsigned char wk1[4];                           /*              */
union {                                         /* TXCR1        */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short TXCR31:1;           /*    TXCR31    */
             unsigned short TXCR30:1;           /*    TXCR30    */
             unsigned short TCR29:1;            /*    TCR29     */
             unsigned short TXCR28:1;           /*    TXCR28    */
```

```
                   unsigned short TXCR27:1;           /*    TXCR27    */
                   unsigned short TSCR26:1;           /*    TSCR26    */
                   unsigned short TXCR25:1;           /*    TXCR25    */
                   unsigned short TXCR24:1;           /*    TXCR24    */
                   unsigned short TXCR23:1;           /*    TXCR23    */
                   unsigned short TXCR22:1;           /*    TXCR22    */
                   unsigned short TXCR21:1;           /*    TXCR21    */
                   unsigned short TXCR20:1;           /*    TXCR20    */
                   unsigned short TXCR19:1;           /*    TXCR19    */
                   unsigned short TXCR18:1;           /*    TXCR18    */
                   unsigned short TXCR17:1;           /*    TXCR17    */
                   unsigned short TXCR16:1;           /*    TXCR16    */
                   } BIT;                             /*              */
            } TXCR1;                                  /*              */
      union {                                         /* TXCR0        */
            unsigned short WORD;                      /*  Word Access */
            struct {                                  /*  Bit Access  */
                   unsigned short TXCR15:1;           /*    TXCR15    */
                   unsigned short TXCR14:1;           /*    TXCR14    */
                   unsigned short TCR13:1;            /*    TCR13     */
                   unsigned short TXCR12:1;           /*    TXCR12    */
                   unsigned short TXCR11:1;           /*    TXCR11    */
                   unsigned short TSCR10:1;           /*    TSCR10    */
                   unsigned short TXCR9:1;            /*    TXCR9     */
                   unsigned short TXCR8:1;            /*    TXCR8     */
                   unsigned short TXCR7:1;            /*    TXCR7     */
                   unsigned short TXCR6:1;            /*    TXCR6     */
                   unsigned short TXCR5:1;            /*    TXCR5     */
                   unsigned short TXCR4:1;            /*    TXCR4     */
                   unsigned short TXCR3:1;            /*    TXCR3     */
                   unsigned short TXCR2:1;            /*    TXCR2     */
                   unsigned short TXCR1:1;            /*    TXCR1     */
                   unsigned short :1;                 /*              */
                   } BIT;                             /*              */
            } TXCR0;                                  /*              */
      unsigned char wk2[4];                           /*              */
      union {                                         /* TXACK1       */
            unsigned short WORD;                      /*  Word Access */
            struct {                                  /*  Bit Access  */
                   unsigned short TXACK31:1;          /*    TXACK31   */
                   unsigned short TXACK30:1;          /*    TXACK30   */
                   unsigned short TXACK29:1;          /*    TXACK29   */
                   unsigned short TXACK28:1;          /*    TXACK28   */
                   unsigned short TXACK27:1;          /*    TXACK27   */
                   unsigned short TXACK26:1;          /*    TXACK26   */
                   unsigned short TXACK25:1;          /*    TXACK25   */
                   unsigned short TXACK24:1;          /*    TXACK24   */
                   unsigned short TXACK23:1;          /*    TXACK23   */
                   unsigned short TXACK22:1;          /*    TXACK22   */
                   unsigned short TXACK21:1;          /*    TXACK21   */
                   unsigned short TXACK20:1;          /*    TXACK20   */
                   unsigned short TXACK19:1;          /*    TXACK19   */
                   unsigned short TXACK18:1;          /*    TXACK18   */
                   unsigned short TXACK17:1;          /*    TXACK17   */
                   unsigned short TXACK16:1;          /*    TXACK16   */
                   } BIT;                             /*              */
```

```
              } TXACK1;                                 /*              */
        union {                                         /* TXACK0       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Bit Access  */
                     unsigned short TXACK15:1;          /*    TXACK15   */
                     unsigned short TXACK14:1;          /*    TXACK14   */
                     unsigned short TXACK13:1;          /*    TXACK13   */
                     unsigned short TXACK12:1;          /*    TXACK12   */
                     unsigned short TXACK11:1;          /*    TXACK11   */
                     unsigned short TXACK10:1;          /*    TXACK10   */
                     unsigned short TXACK9:1;           /*    TXACK9    */
                     unsigned short TXACK8:1;           /*    TXACK8    */
                     unsigned short TXACK7:1;           /*    TXACK7    */
                     unsigned short TXACK6:1;           /*    TXACK6    */
                     unsigned short TXACK5:1;           /*    TXACK5    */
                     unsigned short TXACK4:1;           /*    TXACK4    */
                     unsigned short TXACK3:1;           /*    TXACK3    */
                     unsigned short TXACK2:1;           /*    TXACK2    */
                     unsigned short TXACK1:1;           /*    TXACK1    */
                     unsigned short :1;                 /*              */
                     } BIT;                             /*              */
              } TXACK0;                                 /*              */
        unsigned char wk3[4];                           /*              */
        union {                                         /* ABACK1       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Bit Access  */
                     unsigned short ABACK31:1;          /*    ABACK31   */
                     unsigned short ABACK30:1;          /*    ABACK30   */
                     unsigned short ABACK29:1;          /*    ABACK29   */
                     unsigned short ABACK28:1;          /*    ABACK28   */
                     unsigned short ABACK27:1;          /*    ABACK27   */
                     unsigned short ABACK26:1;          /*    ABACK26   */
                     unsigned short ABACK25:1;          /*    ABACK25   */
                     unsigned short ABACK24:1;          /*    ABACK24   */
                     unsigned short ABACK23:1;          /*    ABACK23   */
                     unsigned short ABACK22:1;          /*    ABACK22   */
                     unsigned short ABACK21:1;          /*    ABACK21   */
                     unsigned short ABACK20:1;          /*    ABACK20   */
                     unsigned short ABACK19:1;          /*    ABACK19   */
                     unsigned short ABACK18:1;          /*    ABACK18   */
                     unsigned short ABACK17:1;          /*    ABACK17   */
                     unsigned short ABACK16:1;          /*    ABACK16   */
                     } BIT;                             /*              */
              } ABACK1;                                 /*              */
        union {                                         /* ABACK0       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Bit Access  */
                     unsigned short ABACK15:1;          /*    ABACK15   */
                     unsigned short ABACK14:1;          /*    ABACK14   */
                     unsigned short ABACK13:1;          /*    ABACK13   */
                     unsigned short ABACK12:1;          /*    ABACK12   */
                     unsigned short ABACK11:1;          /*    ABACK11   */
                     unsigned short ABACK10:1;          /*    ABACK10   */
                     unsigned short ABACK9:1;           /*    ABACK9    */
                     unsigned short ABACK8:1;           /*    ABACK8    */
                     unsigned short ABACK7:1;           /*    ABACK7    */
```

```
             unsigned short ABACK6:1;           /*    ABACK6    */
             unsigned short ABACK5:1;           /*    ABACK5    */
             unsigned short ABACK4:1;           /*    ABACK4    */
             unsigned short ABACK3:1;           /*    ABACK3    */
             unsigned short ABACK2:1;           /*    ABACK2    */
             unsigned short ABACK1:1;           /*    ABACK1    */
             unsigned short :1;                 /*              */
             } BIT;                             /*              */
      } ABACK0;                                 /*              */
unsigned char wk4[4];                           /*              */
union {                                         /* RXPR1        */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short RXPR31:1;           /*    RXPR31    */
             unsigned short RXPR30:1;           /*    RXPR30    */
             unsigned short RXPR29:1;           /*    RXPR29    */
             unsigned short RXPR28:1;           /*    RXPR28    */
             unsigned short RXPR27:1;           /*    RXPR27    */
             unsigned short RXPR26:1;           /*    RXPR26    */
             unsigned short RXPR25:1;           /*    RXPR25    */
             unsigned short RXPR24:1;           /*    RXPR24    */
             unsigned short RXPR23:1;           /*    RXPR23    */
             unsigned short RXPR22:1;           /*    RXPR22    */
             unsigned short RXPR21:1;           /*    RXPR21    */
             unsigned short RXPR20:1;           /*    RXPR20    */
             unsigned short RXPR19:1;           /*    RXPR19    */
             unsigned short RXPR18:1;           /*    RXPR18    */
             unsigned short RXPR17:1;           /*    RXPR17    */
             unsigned short RXPR16:1;           /*    RXPR16    */
             } BIT;                             /*              */
      } RXPR1;                                  /*              */
union {                                         /* RXPR0        */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short RXPR15:1;           /*    RXPR15    */
             unsigned short RXPR14:1;           /*    RXPR14    */
             unsigned short RXPR13:1;           /*    RXPR13    */
             unsigned short RXPR12:1;           /*    RXPR12    */
             unsigned short RXPR11:1;           /*    RXPR11    */
             unsigned short RXPR10:1;           /*    RXPR10    */
             unsigned short RXPR9:1;            /*    RXPR9     */
             unsigned short RXPR8:1;            /*    RXPR8     */
             unsigned short RXPR7:1;            /*    RXPR7     */
             unsigned short RXPR6:1;            /*    RXPR6     */
             unsigned short RXPR5:1;            /*    RXPR5     */
             unsigned short RXPR4:1;            /*    RXPR4     */
             unsigned short RXPR3:1;            /*    RXPR3     */
             unsigned short RXPR2:1;            /*    RXPR2     */
             unsigned short RXPR1:1;            /*    RXPR1     */
             unsigned short RXPR0:1;            /*    RXPR0     */
             } BIT;                             /*              */
      } RXPR0;                                  /*              */
unsigned char wk5[4];                           /*              */
union {                                         /* RFPR1        */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
```

```
                     unsigned short RFPR31:1;           /*    RFPR31    */
                     unsigned short RFPR30:1;           /*    RFPR30    */
                     unsigned short RFPR29:1;           /*    RFPR29    */
                     unsigned short RFPR28:1;           /*    RFPR28    */
                     unsigned short RFPR27:1;           /*    RFPR27    */
                     unsigned short RFPR26:1;           /*    RFPR26    */
                     unsigned short RFPR25:1;           /*    RFPR25    */
                     unsigned short RFPR24:1;           /*    RFPR24    */
                     unsigned short RFPR23:1;           /*    RFPR23    */
                     unsigned short RFPR22:1;           /*    RFPR22    */
                     unsigned short RFPR21:1;           /*    RFPR21    */
                     unsigned short RFPR20:1;           /*    RFPR20    */
                     unsigned short RFPR19:1;           /*    RFPR19    */
                     unsigned short RFPR18:1;           /*    RFPR18    */
                     unsigned short RFPR17:1;           /*    RFPR17    */
                     unsigned short RFPR16:1;           /*    RFPR16    */
                     } BIT;                             /*              */
              } RFPR1;                                  /*              */
        union {                                         /* RFPR0        */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Bit Access  */
                     unsigned short RFPR15:1;           /*    RFPR15    */
                     unsigned short RFPR14:1;           /*    RFPR14    */
                     unsigned short RFPR13:1;           /*    RFPR13    */
                     unsigned short RFPR12:1;           /*    RFPR12    */
                     unsigned short RFPR11:1;           /*    RFPR11    */
                     unsigned short RFPR10:1;           /*    RFPR10    */
                     unsigned short RFPR9:1;            /*    RFPR9     */
                     unsigned short RFPR8:1;            /*    RFPR8     */
                     unsigned short RFPR7:1;            /*    RFPR7     */
                     unsigned short RFPR6:1;            /*    RFPR6     */
                     unsigned short RFPR5:1;            /*    RFPR5     */
                     unsigned short RFPR4:1;            /*    RFPR4     */
                     unsigned short RFPR3:1;            /*    RFPR3     */
                     unsigned short RFPR2:1;            /*    RFPR2     */
                     unsigned short RFPR1:1;            /*    RFPR1     */
                     unsigned short RFPR0:1;            /*    RFPR0     */
                     } BIT;                             /*              */
              } RFPR0;                                  /*              */
        unsigned char wk6[4];                           /*              */
        union {                                         /* MBIMR1       */
              unsigned short WORD;                      /*  Word Access */
              struct {                                  /*  Bit Access  */
                     unsigned short MBIMR31:1;          /*    MBIMR31   */
                     unsigned short MBIMR30:1;          /*    MBIMR30   */
                     unsigned short MBIMR29:1;          /*    MBIMR29   */
                     unsigned short MBIMR28:1;          /*    MBIMR28   */
                     unsigned short MBIMR27:1;          /*    MBIMR27   */
                     unsigned short MBIMR26:1;          /*    MBIMR26   */
                     unsigned short MBIMR25:1;          /*    MBIMR25   */
                     unsigned short MBIMR24:1;          /*    MBIMR24   */
                     unsigned short MBIMR23:1;          /*    MBIMR23   */
                     unsigned short MBIMR22:1;          /*    MBIMR22   */
                     unsigned short MBIMR21:1;          /*    MBIMR21   */
                     unsigned short MBIMR20:1;          /*    MBIMR20   */
                     unsigned short MBIMR19:1;          /*    MBIMR19   */
```

```
             unsigned short MBIMR18:1;          /*    MBIMR18   */
             unsigned short MBIMR17:1;          /*    MBIMR17   */
             unsigned short MBIMR16:1;          /*    MBIMR16   */
             } BIT;                             /*              */
      } MBIMR1;                                 /*              */
union {                                         /* MBIMR0       */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short MBIMR15:1;          /*    MBIMR15   */
             unsigned short MBIMR14:1;          /*    MBIMR14   */
             unsigned short MBIMR13:1;          /*    MBIMR13   */
             unsigned short MBIMR12:1;          /*    MBIMR12   */
             unsigned short MBIMR11:1;          /*    MBIMR11   */
             unsigned short MBIMR10:1;          /*    MBIMR10   */
             unsigned short MBIMR9:1;           /*    MBIMR9    */
             unsigned short MBIMR8:1;           /*    MBIMR8    */
             unsigned short MBIMR7:1;           /*    MBIMR7    */
             unsigned short MBIMR6:1;           /*    MBIMR6    */
             unsigned short MBIMR5:1;           /*    MBIMR5    */
             unsigned short MBIMR4:1;           /*    MBIMR4    */
             unsigned short MBIMR3:1;           /*    MBIMR3    */
             unsigned short MBIMR2:1;           /*    MBIMR2    */
             unsigned short MBIMR1:1;           /*    MBIMR1    */
             unsigned short MBIMR0:1;           /*    MBIMR0    */
             } BIT;                             /*              */
      } MBIMR0;                                 /*              */
unsigned char wk7[4];                           /*              */
union {                                         /* UMSR1        */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short UMSR31:1;           /*    UMSR31    */
             unsigned short UMSR30:1;           /*    UMSR30    */
             unsigned short UMSR29:1;           /*    UMSR29    */
             unsigned short UMSR28:1;           /*    UMSR28    */
             unsigned short UMSR27:1;           /*    UMSR27    */
             unsigned short UMSR26:1;           /*    UMSR26    */
             unsigned short UMSR25:1;           /*    UMSR25    */
             unsigned short UMSR24:1;           /*    UMSR24    */
             unsigned short UMSR23:1;           /*    UMSR23    */
             unsigned short UMSR22:1;           /*    UMSR22    */
             unsigned short UMSR21:1;           /*    UMSR21    */
             unsigned short UMSR20:1;           /*    UMSR20    */
             unsigned short UMSR19:1;           /*    UMSR19    */
             unsigned short UMSR18:1;           /*    UMSR18    */
             unsigned short UMSR17:1;           /*    UMSR17    */
             unsigned short UMSR16:1;           /*    UMSR16    */
             } BIT;                             /*              */
      } UMSR1;                                  /*              */
union {                                         /* UMSR0        */
      unsigned short WORD;                      /*  Word Access */
      struct {                                  /*  Bit Access  */
             unsigned short UMSR15:1;           /*    UMSR15    */
             unsigned short UMSR14:1;           /*    UMSR14    */
             unsigned short UMSR13:1;           /*    UMSR13    */
             unsigned short UMSR12:1;           /*    UMSR12    */
             unsigned short UMSR11:1;           /*    UMSR11    */
```

```
                unsigned short UMSR10:1;           /*    UMSR10    */
                unsigned short UMSR9:1;            /*    UMSR9     */
                } BIT;                             /*              */
         } UMSR0;                                  /*              */
   unsigned char wk8[36];                          /*              */
   unsigned short TCNTR;                           /* TCNTR        */
   union {                                         /* TCR          */
         unsigned short WORD;                      /*  Word Access */
         struct {                                  /*  Bit Access  */
                unsigned short TCR15:1;            /*    TCR15     */
                unsigned short TCR14:1;            /*    TCR14     */
                unsigned short TCR13:1;            /*    TCR13     */
                unsigned short TCR12:1;            /*    TCR12     */
                unsigned short TCR11:1;            /*    TCR11     */
                unsigned short TCR10:1;            /*    TCR10     */
                unsigned short TCR9:1;             /*    TCR9      */
                unsigned short TCR8:1;             /*    TCR8      */
                unsigned short TCR7:1;             /*    TCR7      */
                unsigned short :1;                 /*              */
                unsigned short TPSC:6;             /*    TPSC      */
                } BIT;                             /*              */
         } TCR;                                    /*              */
   union {                                         /* TSR          */
         unsigned short WORD;                      /*  Word Access */
         struct {                                  /*  Bit Access  */
                unsigned short :13;                /*              */
                unsigned short TSR2:1;             /*    TSR2      */
                unsigned short TSR1:1;             /*    TSR1      */
                unsigned short TSR0:1;             /*    TSR0      */
                } BIT;                             /*              */
         } TSR;                                    /*              */
   unsigned short TDCR;                            /* TDCR         */
   unsigned short LOSR;                            /* LOSR         */
   unsigned char wk9[2];                           /*              */
   unsigned short HCAN2_ICR0;                      /* HCAN2_ICR0   */
   unsigned short HCAN2_ICR1;                      /* HCAN2_ICR1   */
   unsigned short TCMR0;                           /* TCMR0        */
   unsigned short TCMR1;                           /* TCMR1        */
   unsigned char wk10[108];                        /*              */

struct st_mb {
       union {                                     /* MB0          */
         unsigned char BYTE;                       /*  Byte Access */
         struct {                                  /*  Bit Access  */
                unsigned char :1;                  /*              */
                unsigned char STDID10:1;           /*    STDID10   */
                unsigned char STDID9:1;            /*    STDID9    */
                unsigned char STDID8:1;            /*    STDID8    */
                unsigned char STDID7:1;            /*    STDID7    */
                unsigned char STDID6:1;            /*    STDID6    */
                unsigned char STDID5:1;            /*    STDID5    */
                unsigned char STDID4:1;            /*    STDID4    */
                } BIT;                             /*              */
         } MB0;                                    /*              */
   union {                                         /* MB1          */
         unsigned char BYTE;                       /*  Byte Access */
```

```
      struct {                                  /*  Bit Access  */
             unsigned char STDID:4;             /*    STDID     */
             unsigned char RTR:1;               /*    RTR       */
             unsigned char IDE:1;               /*    IDE       */
             unsigned char EXTID17:1;           /*    EXTID17   */
             unsigned char EXTID16:1;           /*    EXTID16   */
             } BIT;                             /*              */
      } MB1;                                    /*              */
union {                                         /* MB2          */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char EXTID15:1;           /*    EXTID15   */
             unsigned char EXTID14:1;           /*    EXTID14   */
             unsigned char EXTID13:1;           /*    EXTID13   */
             unsigned char EXTID12:1;           /*    EXTID12   */
             unsigned char EXTID11:1;           /*    EXTID11   */
             unsigned char EXTID10:1;           /*    EXTID10   */
             unsigned char EXTID9:1;            /*    EXTID9    */
             unsigned char EXTID8:1;            /*    EXTID8    */
             } BIT;                             /*              */
      } MB2;                                    /*              */
union {                                         /* MB3          */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char EXTID7:1;            /*    EXTID7    */
             unsigned char EXTID6:1;            /*    EXTID6    */
             unsigned char EXTID5:1;            /*    EXTID5    */
             unsigned char EXTID4:1;            /*    EXTID4    */
             unsigned char EXTID3:1;            /*    EXTID3    */
             unsigned char EXTID2:1;            /*    EXTID2    */
             unsigned char EXTID1:1;            /*    EXTID1    */
             unsigned char EXTID0:1;            /*    EXTID0    */
             } BIT;                             /*              */
      } MB3;                                    /*              */
union {                                         /* MB4          */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char CCM:1;               /*    CCM       */
             unsigned char TTE:1;               /*    TTE       */
             unsigned char NMC:1;               /*    NMC       */
             unsigned char ATX:1;               /*    ATX       */
             unsigned char DART:1;              /*    DART      */
             unsigned char MBC:3;               /*    MBC       */
             } BIT;                             /*              */
      } MB4;                                    /*              */
union {                                         /* MB5          */
      unsigned char BYTE;                       /*  Byte Access */
      struct {                                  /*  Bit Access  */
             unsigned char PTE:1;               /*    PTE       */
             unsigned char TCT:1;               /*    TCT       */
             unsigned char CBE:1;               /*    CBE       */
             unsigned char :1;                  /*              */
             unsigned char DLC:4;               /*    DLC       */
             } BIT;                             /*              */
      } MB5;                                    /*              */
unsigned char TIME_STAMP;                       /* TIME_STAMP   */
```

```
       unsigned char wk11[1];                          /*              */
       unsigned char MSG_DATA[8];                      /* MSG_DATA     */
       unsigned char LAFM0[2];                         /* LAFM0        */
       unsigned char LAFM1[2];                         /* LAFM1        */
       unsigned char wk12[12];                         /*              */
        }mb[32];
};                                                     /*              */

#define P_SCI2 (*(volatile struct st_sci *)0xFFFF81C0)   /* SCI2 Address  */
#define P_SCI3 (*(volatile struct st_sci *)0xFFFF81D0)   /* SCI3 Address  */
#define P_SCI4 (*(volatile struct st_sci *)0xFFFF81E0)   /* SCI4 Address  */
#define P_MTU34 (*(volatile struct st_mtu34 *)0xFFFF8200)/* MTU34 Address */
#define P_MTU0 (*(volatile struct st_mtu0 *)0xFFFF8260)  /* MTU0 Address  */
#define P_MTU1 (*(volatile struct st_mtu1 *)0xFFFF8280)  /* MTU1 Address  */
#define P_MTU2 (*(volatile struct st_mtu2 *)0xFFFF82A0)  /* MTU2 Address  */
#define P_INTC (*(volatile struct st_intc *)0xFFFF8348)  /* INTC Address  */
#define P_PORTA (*(volatile struct st_porta *)0xFFFF8382)/* PORTA Address */
#define P_PORTB (*(volatile struct st_portb *)0xFFFF8390)/* PORTB Address */
#define P_PORTD (*(volatile struct st_portd *)0xFFFF83A2)/* PORTD Address */
#define P_PORTE (*(volatile struct st_porte *)0xFFFF83B0)/* PORTE Address */
#define P_PORTF (*(volatile struct st_portf *)0xFFFF83B2)/* PORTF Address */
#define P_MTU (*(volatile struct st_mtu *)0xFFFF83C0)    /* MTU Address   */
#define P_MMT (*(volatile struct st_mmt *)0xFFFF83C4)    /* MMT Address   */
#define P_PORTG (*(volatile struct st_portg *)0xFFFF83CD)/* PORTG Address */
#define P_CMT (*(volatile struct st_cmt *)0xFFFF83D0)    /* CMT Address   */
#define P_AD (*(volatile struct st_ad *)0xFFFF8420)      /* AD Address    */
#define P_FLASH (*(volatile struct st_flash *)0xFFFF8580)/* FLASH Address */
#define P_UBC (*(volatile struct st_ubc *)0xFFFF8600)    /* UBC Address   */
#define P_WDT (*(volatile struct st_wdt *)0xFFFF8610)    /* WDT Address   */
#define P_STBY (*(volatile struct st_stby *)0xFFFF8614)  /* STBY Address  */
#define P_BSC (*(volatile struct st_bsc *)0xFFFF8620)    /* BSC Address   */
#define P_DTC (*(volatile struct st_dtc *)0xFFFF8700)    /* DTC Address   */
#define P_HUDI (*(volatile struct st_hudi *)0xFFFF8A50)  /* HUDI Address  */
#define P_HCAN2 (*(volatile struct st_hcan2 *)0xFFFFB000)/* HCAN2 Address */
```

SH7046 Series On-Chip Peripheral Functions — DTC Volume Application Note

Publication Date: 1st Edition, November 14, 2003
Published by: Sales Strategic Planning Div.
Renesas Technology Corp.
Edited by: Technical Documentation & Information Department
Renesas Kodaira Semiconductor Co., Ltd.

©2003 Renesas Technology Corp. All rights reserved. Printed in Japan.

Renesas Technology Corp. Sales Strategic Planning Div. Nippon Bldg., 2-6-2, Ohte-machi, Chiyoda-ku, Tokyo 100-0004, Japan

RENESAS SALES OFFICES

http://www.renesas.com

Renesas Technology America, Inc.
450 Holger Way, San Jose, CA 95134-1368, U.S.A
Tel: <1> (408) 382-7500 Fax: <1> (408) 382-7501

Renesas Technology Europe Limited.
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, United Kingdom
Tel: <44> (1628) 585 100, Fax: <44> (1628) 585 900

Renesas Technology Europe GmbH
Dornacher Str. 3, D-85622 Feldkirchen, Germany
Tel: <49> (89) 380 70 0, Fax: <49> (89) 929 30 11

Renesas Technology Hong Kong Ltd.
7/F., North Tower, World Finance Centre, Harbour City, Canton Road, Hong Kong
Tel: <852> 2265-6688, Fax: <852> 2375-6836

Renesas Technology Taiwan Co., Ltd.
FL 10, #99, Fu-Hsing N. Rd., Taipei, Taiwan
Tel: <886> (2) 2715-2888, Fax: <886> (2) 2713-2999

Renesas Technology (Shanghai) Co., Ltd.
26/F., Ruijin Building, No.205 Maoming Road (S), Shanghai 200020, China
Tel: <86> (21) 6472-1001, Fax: <86> (21) 6415-2952

Renesas Technology Singapore Pte. Ltd.
1, Harbour Front Avenue, #06-10, Keppel Bay Tower, Singapore 098632
Tel: <65> 6213-0200, Fax: <65> 6278-8001

SH7046 Series
On-Chip Peripheral Functions — DTC Volume
Application Note

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan