The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

How to Check the Ordering Part Number

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

For More Information

Please contact your local sales office for additional information about Cypress products and solutions.

About Cypress

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

AN204448

Dual CPU Operation on Traveo™ Family, MB9D560 Series Microcontroller

Associated Part Family:	Series Name	Product Number
	MB9D560	MB9DF564MAE/F565MAE/F566MAE MB9DF564MGE/F565MGE/F566MGE

This application note is intended for persons who are considering the use of Traveo family MB9D560.

Contents

1 Introduction 1
1.1 About this Document 1
1.2 Environment of Development 1
2 About Access to the Memory 2
2.1 Path of Access to the Memory 2
2.2 Recommended Method of Accessing to the Memory 3
2.3 If the Recommended Method Isn't Done 3
3 About Access to the Register 5
3.1 Bit-band Unit 5
3.2 Recommended Method of Accessing to the Register 5
3.3 If the Recommended Method Isn't Done 5
4 About Using the Peripheral 10
4.1 Recommended Method of Using the Peripheral 10
4.2 If the Recommended Method Isn't Done 13
5 Reference 18
A Appendix 19
A.1 How to Take Synchronization between Cores 19
Document History 21
Worldwide Sales and Design Support 22
Products 22
PSoC® Solutions 22
Cypress Developer Community 22
Technical Support 22

1 Introduction

1.1 About this Document

This application note is intended for persons who are considering the use of Traveo family MB9D560.

MB9D560 is equipped with two CPU cores. Therefore, there are dual-core microcomputer specific important points when using of the peripheral and the memory. Those important points on software implementation are described in this application note.

1.2 Environment of Development

Contents is described in this application note has been developed in the environment shown in Table 1.

Table 1. Environment of Development

Microcomputer	MB9DF566MGB
Integrated Development Environments	MULTI v6.1.4
Evaluation Board	MB2198-770-02-E0
Optimization	Optimize for Speed

2 About Access to the Memory

2.1 Path of Access to the Memory

The MCU is equipped with memories as described below.

- TCRAM
- TCFLASH
- EAM
- WorkFLASH

All cores have access to all memories.

TCRAM and TCFLASH that are equipped for each core have access restrictions as described in Table 2

Figure 1. Connection Diagram of Memories

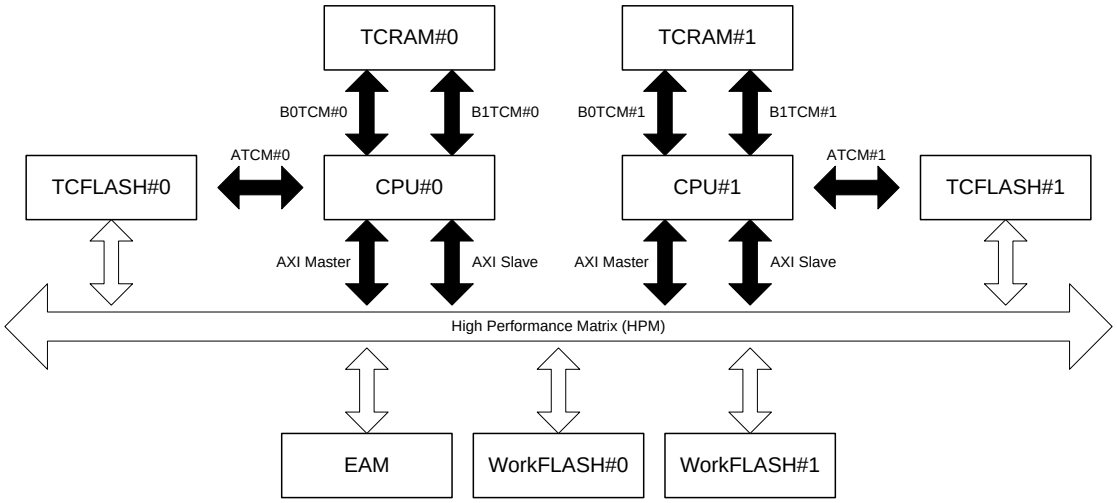


Table 2. Prohibited Matters of Each Access Path

Access Target	Path	Prohibited Matters
Own core's TCRAM	Own core - TCM - TCRAM	-
Another core's TCRAM	Own core - AXI master - AXI slave - Another core - TCM - TCRAM	-
Own core's TCFLASH	Own core - TCM - TCFLASH	Write access
	Own core - AXI master - TCFLASH	Read access
Another core's TCFLASH	Own core - AXI master - AXI slave - Another core - TCM - TCFLASH	Write access
	Own core - AXI master - TCFLASH	Read access
EAM	Own core - AXI master - EAM	-
WorkFLASH	Own core - AXI master - WorkFLASH	-

2.2 Recommended Method of Accessing to the Memory

As follows according to use of the memory, it is desirable to take synchronization between cores, or use memory protection unit (MPU). Because, it is possible to prevent that data in the memory is rewritten to unintended value.

For details on how to use the memory protection unit (MPU), see “ARM® Architecture Reference Manual ARM® v7-A and ARM® v7-R edition (ARM DDI 0406B)”.

For details on how to take synchronization between cores, see “A.1”.

- Case of accessing to one memory by only one core.

  Prevent access to unintended area by the own core, by using memory protection unit (MPU).

- Case of writing to one memory by only one core.

  Prevent write to unintended area by the own core, by using memory protection unit (MPU).

- Case of writing to one memory by multiple cores.

  Prevent write to one address by multiple cores, by separating the write destination address that each core will access.

  Prevent write to unintended area by the own core, by using memory protection unit (MPU).

- Case of writing to one address by multiple cores.

  Prevent write to the address by another core when one of the cores is writing to the address, by taking synchronization between cores.

  Have to follow the following when executing the functions of Table 3.

- Execute those functions after confirming that all cores aren't accessing the memory of operation subject, by taking synchronization between cores.

- All cores mustn't access to the memory of operation subject, while those functions are executed.

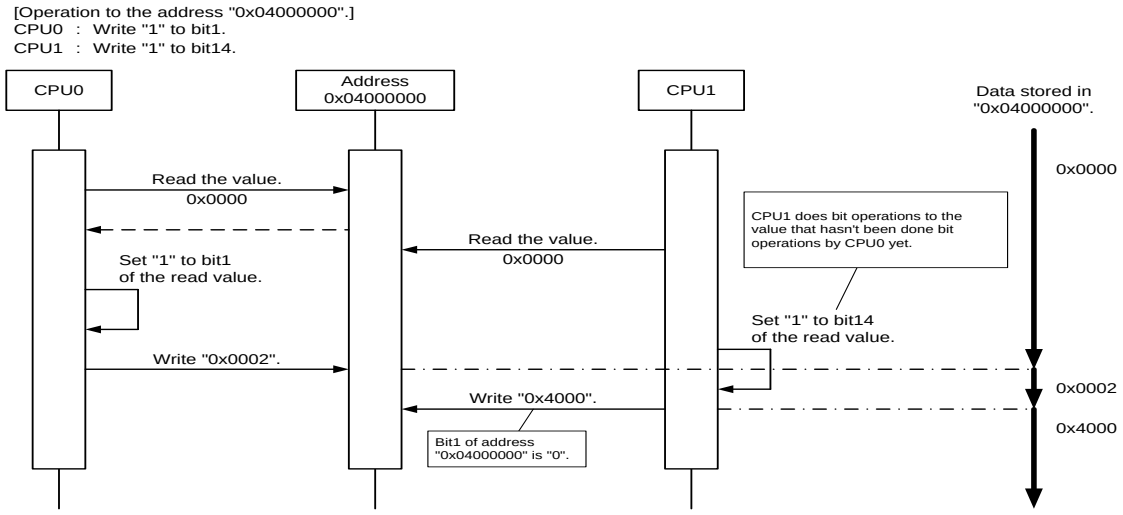
2.3 If the Recommended Method Isn't Done

There is possibility that problems occur, if the method described in “2.2Recommended Method of Accessing to the Memory” isn't done. About problems that may occur, are described in the following.

2.3.1 Case of Performing the RMW Access from Multiple Cores

By timing, there is possibility that the unintended bits are changed when one address is done RMW (Read-Modify-Write) access by multiple cores. The example is described in Figure 2.

Figure 2. Example of Doing RMW Access to One Address by Multiple Cores



2.3.2 Case of performing unintended behavior

There is possibility that data in the memory is destroyed by unintended writing when the core performs unintended behavior. By using memory protection unit (MPU), it is possible to prevent write to unintended area by the own core.

2.3.3 Case of Satisfying Conditions of Prohibiting Access to Memory

If one of the cores does the prohibited matters of Table 3 when conditions of Table 3 are satisfied by other core, it causes bus error. The example is described in Figure 3.

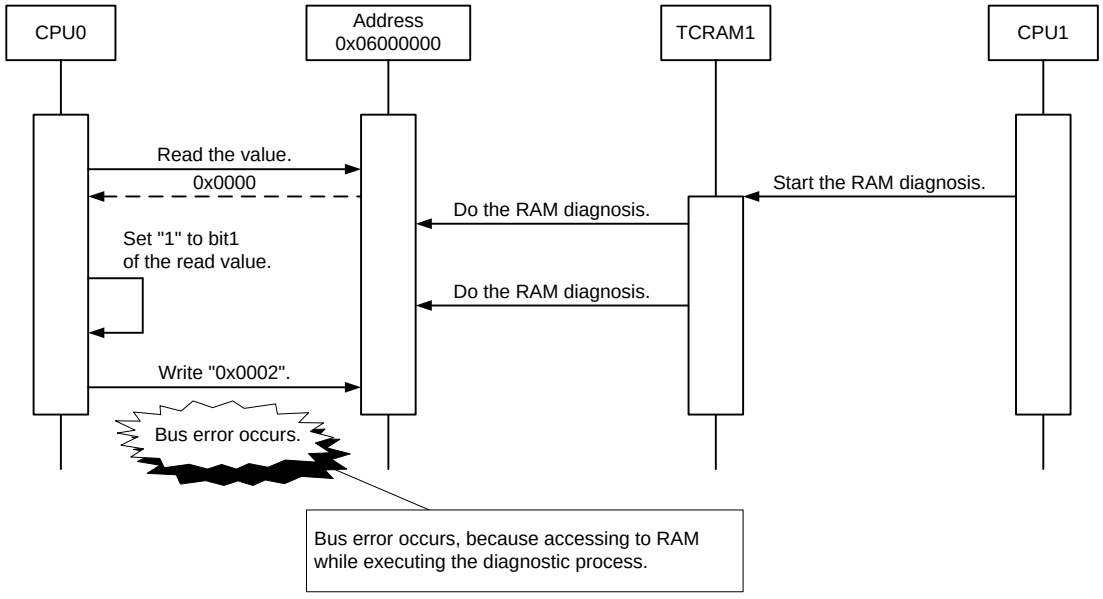
Table 3. Conditions of Prohibiting Access to Memory

Peripheral	Conditions	Prohibited Matters
TCFLASH	TCFCFGn_FCFGR.WE = “0”	Write to TCFLAH or erase data in TCFLASH.
	TCFCFGn_FCFGR.WE = “1”	Read from TCFLASH via the TCM interface.
	The automatic algorithm is in progress.	Read from TCFLASH via the TCM interface.
TCRAM	RAM diagnosis is in progress.	Access to RAM
	RAM initialization is in progress.	Access to RAM
WorkFLASH	WFCFGxx_CR.WE = “0”	Write to WorkFLASH
	The command sequencer isn't idle.	Access to WorkFLASH

n : Number of Cortex-R5F core to which the peripheral is connected (n = 0, 1)

xx : Unit number (xx = 00, 01, 02, 03)

Figure 3. Examples of Accessing RAM While Executing RAM Diagnosis



3 About Access to the Register

3.1 Bit-band Unit

In this microcomputer, it is possible to operate the corresponding one bit in the bit-band area by access to one byte in the bit-band alias area. Have to use the bit-band unit if do bit operation of the register corresponding to the bit-band. Because the value of unintended bits doesn't change when the bit is operated by using the bit-band unit.

For details on how to use the bit-band unit, see “Hardware manual”.

* If using the definition in the IO header file which is offered, the register is accessed by using a bit-band unit.

3.2 Recommended Method of Accessing to the Register

Have to access to registers as follows.

- About accessing to registers by multiple cores.

  Prevent accessing to one register by multiple cores, by separating registers that each core will access.

- About bit operation of the register corresponding to the bit-band.

  Use the bit-band unit if do bit operation of the register corresponding to the bit-band.

- About the timing for write to the unprotect register.

  Write to the protection target register without doing other processing after write to unprotect register (Table 7).

- About writing to the unprotect register in Memory & Config Group.

  Disable all interrupts before unprotecting the register in Memory & Config Group, because Memory & Config Group contains the IRC.

3.3 If the Recommended Method Isn't Done

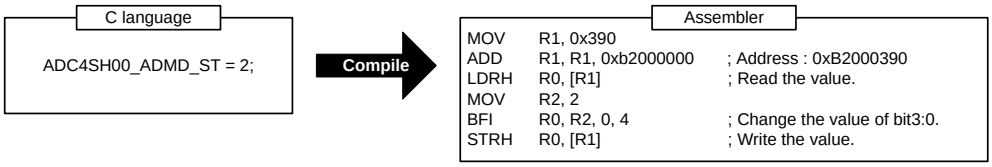
There is possibility that problems occur, if the method described in “3.2Recommended Method of Accessing to the Register” isn't done. About problems that may occur, are described in the following.

3.3.1 Case of Not Use Bit-band Unit

If implement bit operation that doesn't use bit-band unit by C language, compiler outputs the assembler code that does read-modify-write access. Therefore, by timing, there is possibility that the unintended bit is changed when one register is done bit operation by multiple cores. The example is described in Figure 4 and Figure 5.

Figure 4. Examples of Source Code for Doing Bit Operation

[Processing of CPU0]
Write "2" to ST[bit3:0] of 12-bit 4ch A/D converter's A/D mode setting register (ADC4SH00_ADMD).



[Processing of CPU1]
Write "9" to CT[bit11:8] of 12-bit 4ch A/D converter's A/D mode setting register (ADC4SH00_ADMD).

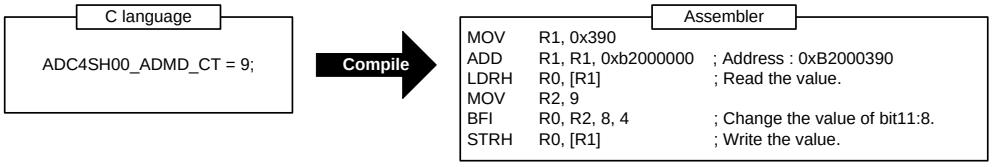
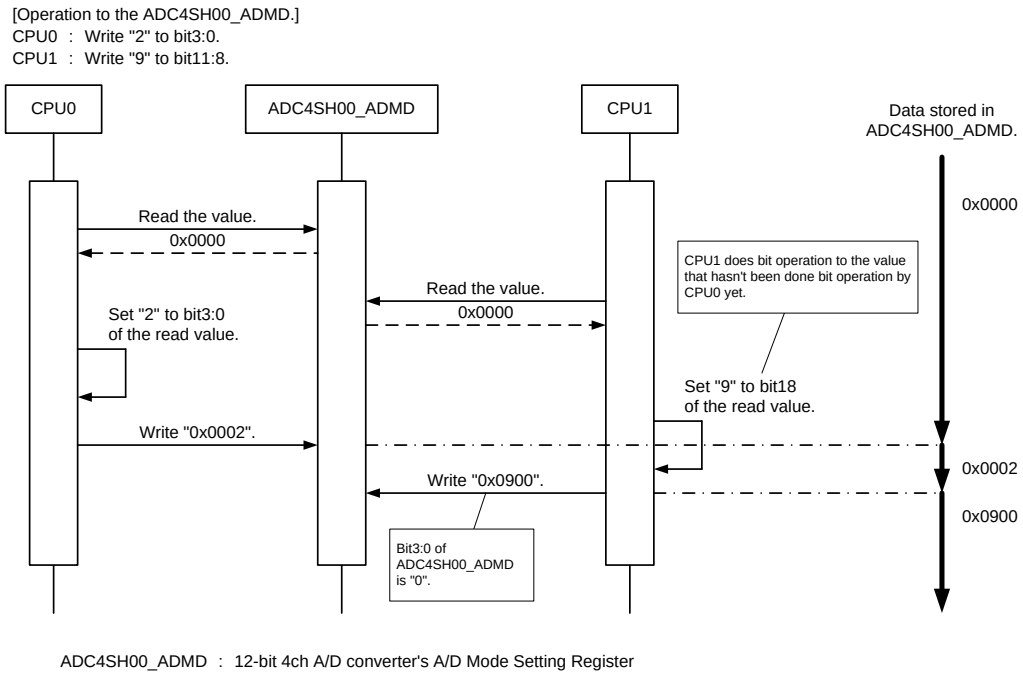


Figure 5. Example of Bit Operation to One Register by Multiple Cores



ADC4SH00_ADMD : 12-bit 4ch A/D converter's A/D Mode Setting Register

3.3.2 Case of Writing to the Key Code Register

Have to write key code to those registers continuously and in predetermined order, in order to operate the key code registers (Table 4) By timing, there is possibility that it is written the key code of unintended order when one key code register is written by multiple cores. The example is described in Figure 6.

In the type unequipped with the key code function, writing key code isn't necessary for the following registers. In this type, writing to the following registers has no effect on the operation. See "option" of 'Data sheet' for the presence of the key code function.

- GPIO_KEYCDR
- PPC_KEYCDR
- KEYCDR
- DACxx_KEYCDR (xx = 00, 01)

Table 4. Key Code Registers

Abbreviated Register Name	Register Name	Function
GPIO_KEYCDR *1	GPIO Key Code Register	Release protection for preventing erroneous writing.
PPC_KEYCDR *1	PPC Key Code Register	Release protection for preventing erroneous writing.
LAMERT	LIN Assist Mode Error Test Register	Enable the pseudo error settings or disable that.
CIF1	Control Register	Output reset to the FlexRay macro.
LCK	Lock Register	Allow writing of READY command.
KEYCDR *1	Key Code Register	Release protection for preventing erroneous writing.
WFGnn_DTMNS	16-bit Dead Timer Minus Control Register	Configure the minus control of the dead time function.

Abbreviated Register Name	Register Name	Function
DACxx_KEYCDR *1	Key Code Register	Release protection for preventing erroneous writing.
MVACC	Calculation Control Instruction Register	Release constant parameter protection.

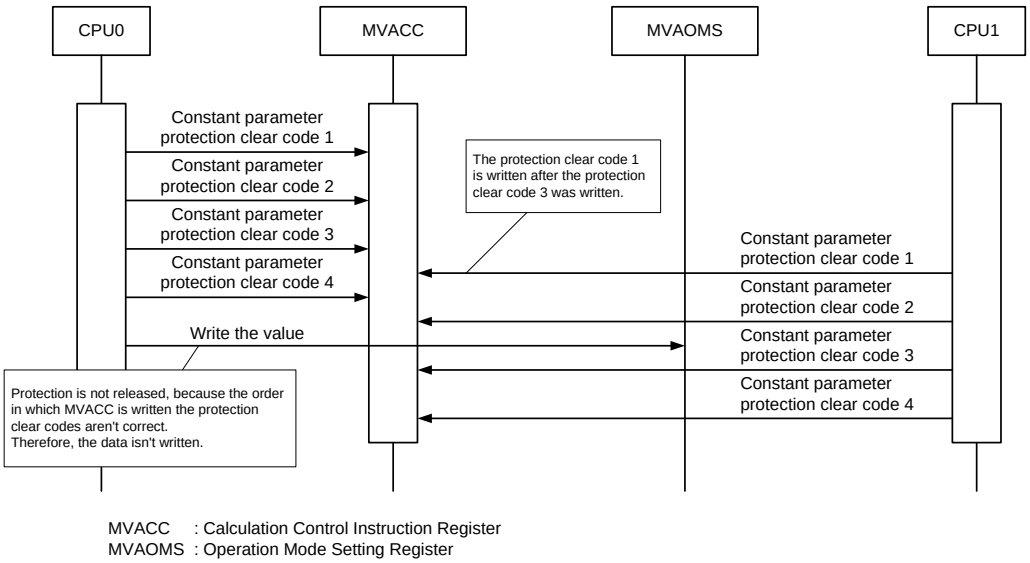
nn : channel number (nn = 00 ~ 03)

xx : channel number (xx = 00, 01)

1. Only in the type equipped with the key code function, have to write key code.

Those registers aren't described in the Table 4, because occur bus error when another core accesses to registers that are separated per core.

Figure 6. Example of Writing to One Key Code Register by Multiple Cores



MVACC : Calculation Control Instruction Register
MVAOMS : Operation Mode Setting Register

3.3.3 Case of Writing to the Unlock Register

Have to unlock before write to the register that is protected by unlock registers (Table 5), when writing to those registers. By timing, there is possibility that the protection target register isn't written when one unlock register is written by multiple cores. The example is described in Figure 7.

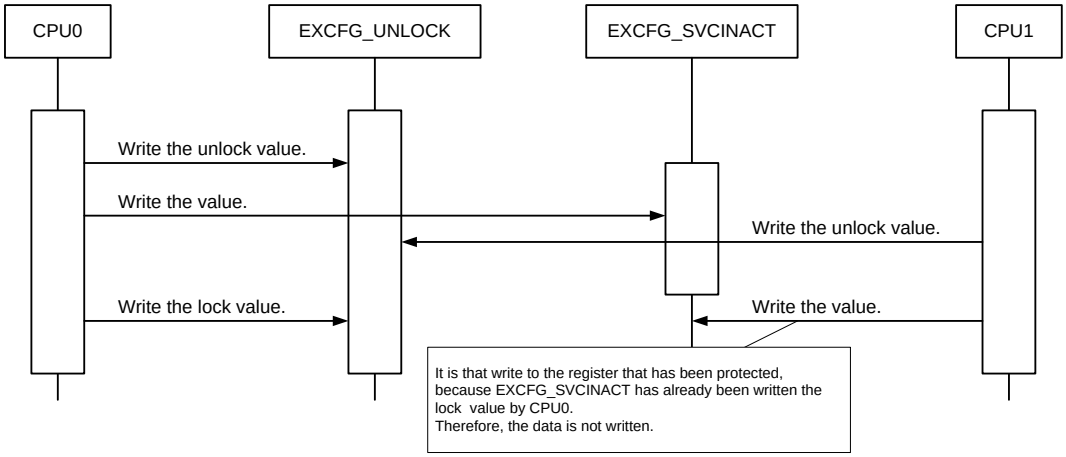
Table 5. Unlock Registers

Abbreviated Register Name	Register Name	Function
EXCFG_UNLOCK	EXCFG Lock Release Register	Control the write lock of the register of the BootROM hardware interface.
NMID_UNLOCK	NMID Lock Release Register	Control the write lock of the register of the NMI distribution unit.
MPUHm_UNLOCK	MPU AHB Unlock Register	Control the write lock of the MPU AHB register.
MVACC	Calculation Control Instruction Register	Control the protection of the register of the constant parameter protection target.

m : Indicate that the register is an instance "m" of the module. (m = 0)

Those registers aren't described in the Table 5, because occur bus error when another core accesses to registers that are separated per core.

Figure 7. Example of Writing to One Unlock Register by Multiple Cores



EXCFG_UNLOCK : EXCFG Lock Release Register
EXCFG_SVCINACT : EXCFG Inactive Set - Supervisor Call Vector Register

3.3.4 Case of Writing to the Consecutive Write Prohibited Register

By timing, there is possibility that the consecutive write prohibition registers (Table 6) are written continuously when one of those registers is written by multiple cores. The example is described in Figure 8.

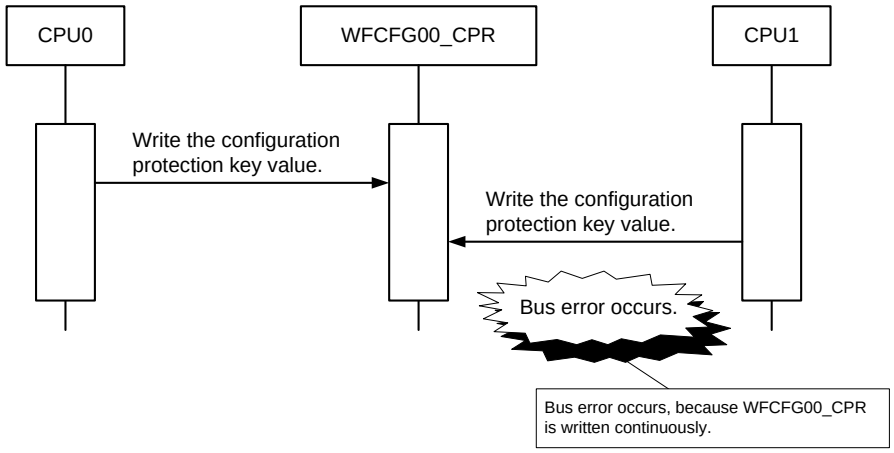
Table 6. Consecutive Write Prohibited Registers

Abbreviated Register Name	Register Name	Function
WFCFGxx_CPR	WorkFLASH00/01/02/03 Configuration Protection Key Register	Protect the protection target register from unintended writing.

xx : Unit number (xx = 00, 01, 02, 03)

Those registers aren't described in the Table 6, because occur bus error when another core accesses to registers that are separated per core.

Figure 8. Example of Writing to One Consecutive Write Prohibited Register by Multiple Cores

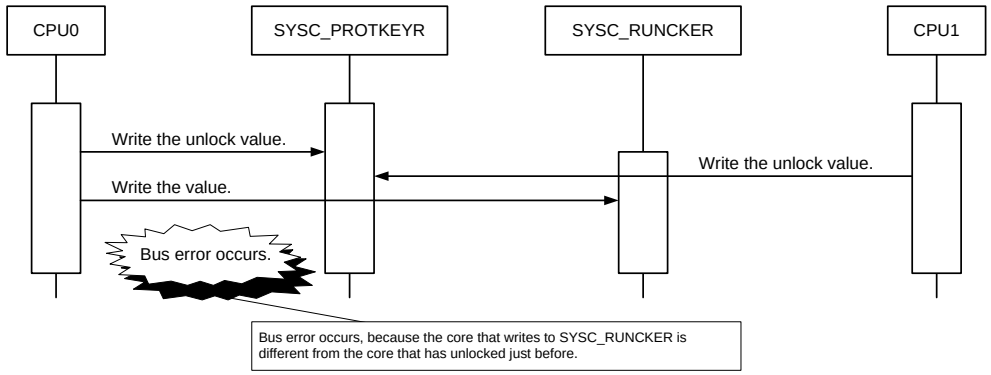


WFCFG00_CPR : WorkFLASH Configuration Protection Key Register

3.3.5 Case of Writing to the Protection Key Setting Register

They do not match when the protection key setting register (SYSC_PROTKEYR) is written by multiple cores, the core that writes the unlock value to it and the core that writes to the protection target register. The example is described in Figure 9.

Figure 9. Example of Writing to One Protection Key Setting Register by Multiple Cores



SYSC_PROTKEYR : Protection Key Setting Register
SYSC_RUNCKER : RUN Clock Enable Register

3.3.6 Case of Using the Unprotect Register

Unprotect registers (Table 7) are different from the register that starts protection of the protection target register when written the lock value. Unprotect registers start it when either of the following processes were done.

- Write to the protection target register.
- Write to register of the same group as the protection target register.

There are unprotect registers that start protection when accessed by another core. If writing to the register that is protected, bus error occurs.

The Memory & Config Group's register that is released protection is protected again when interrupt occurs, because Memory & Config Group contains the IRC. Therefore, all interrupts must have been disabled beforehand when it is released protection.

Table 7. Conditions of Starting to Protection

Peripheral Name [Group Name]	Unprotect Register Name < Abbreviated Register Name >	Whether to Start Protection			
		Own Core Tw	Own Core Gw	Another Core Tw	Another Core Gw
System Controller (SYSC) [MCU Config Group]	Protection Key Setting Register <SYSC_PROTKEYR>	Start protection	Start protection	Bus error occurs	Start protection
TCFLASH [Memory & Config Group]	TCFLASH0/1/2/3 Configuration Protection Key Register <TCFCFGn_FCPROTKEY>	Start protection	Start protection	Bus error occurs	Bus error occurs *1
WorkFLASH [Memory & Config Group]	WorkFLASH00/01/02/03 Configuration Protection Key Register <WFCFGxx_CPR>	Start protection	Start protection	Bus error occurs	Start protection *1
HW-WDT [MCU Config Group]	Hardware Watchdog Protection Register <HWDG_PROT>	Start protection	Start protection *2	Bus error occurs	Bus error occurs *1
SW-WDT [MCU Config Group]	Software Watchdog Protection Register <SWDGn_PROT>	Start protection	Start protection *2	Bus error occurs	Bus error occurs *1

n : Number of Cortex-R5F core to which the peripheral is connected (n = 0, 1)

xx : Unit number (xx = 00, 01, 02, 03)

Own core Tw : Write to the protection target register from own core.

Own core Gw : Write to the register of the same group as the protection target register from own core. *3

Another core Tw : Write to the protection target register from another core.

Another core Gw : Write to the register of the same group as the protection target register from another core. *3

1. Writing to the register of different peripheral in the same group does not affect operation.
2. Writing to the unprotected register of same peripheral does not affect operation.
3. It includes the register that isn't protected in the same peripheral.

4 About Using the Peripheral

4.1 Recommended Method of Using the Peripheral

Have to use the peripheral as follows.

- About the channel that is accessed by core.

Separate the channel that each core will access. The example is described in Table 8.

- About the address that is set to the starting address of the exception handler.

Setting of exception handler start address is a common setting for all cores. Therefore, have to note the following points.

  - It doesn't have to be set from multiple cores.
  - The settings are reflected in all cores.
  - It has to be set the address that is no problem for the all cores.

The example is described in Figure 10 and Figure 11.

- About the setting of CAN.

Set the CAN after confirming that the setting of the CAN prescaler has been completed, when setting the CAN at each core. The example is described in Figure 12.

- About the timing of executing the software reset.

Execute the software reset after confirming that the following sequence aren't executing. The example is described in Figure 13.

  - Watchdog register write protection sequence
  - Watchdog counter clear protection trigger sequence

- About the processing after setting the software reset bit.

Mustn't perform all the sequences of the watchdog after setting the software reset bit, without relating to the CPU operating mode.

It takes time before the reset is executed after setting a software reset bit. And it is indefinite. Therefore, have to wait in an infinite loop until the reset is executed. The example is described in Figure 13.

Table 8. Example of Using Peripheral

Core	Peripheral
CPU0	WorkFLASH#0
	CAN ch0
	32-bit free-run timer ch0
	32-bit free-run timer ch2
	12-bit A/D converter ch0
	12-bit A/D converter ch2
CPU1	WorkFLASH#1
	CAN ch1
	32-bit free-run timer ch1
	32-bit free-run timer ch3
	12-bit A/D converter ch1
	12-bit A/D converter ch3

Figure 10. Example of Setting the Starting Address of the Exception Handler (CPU0)

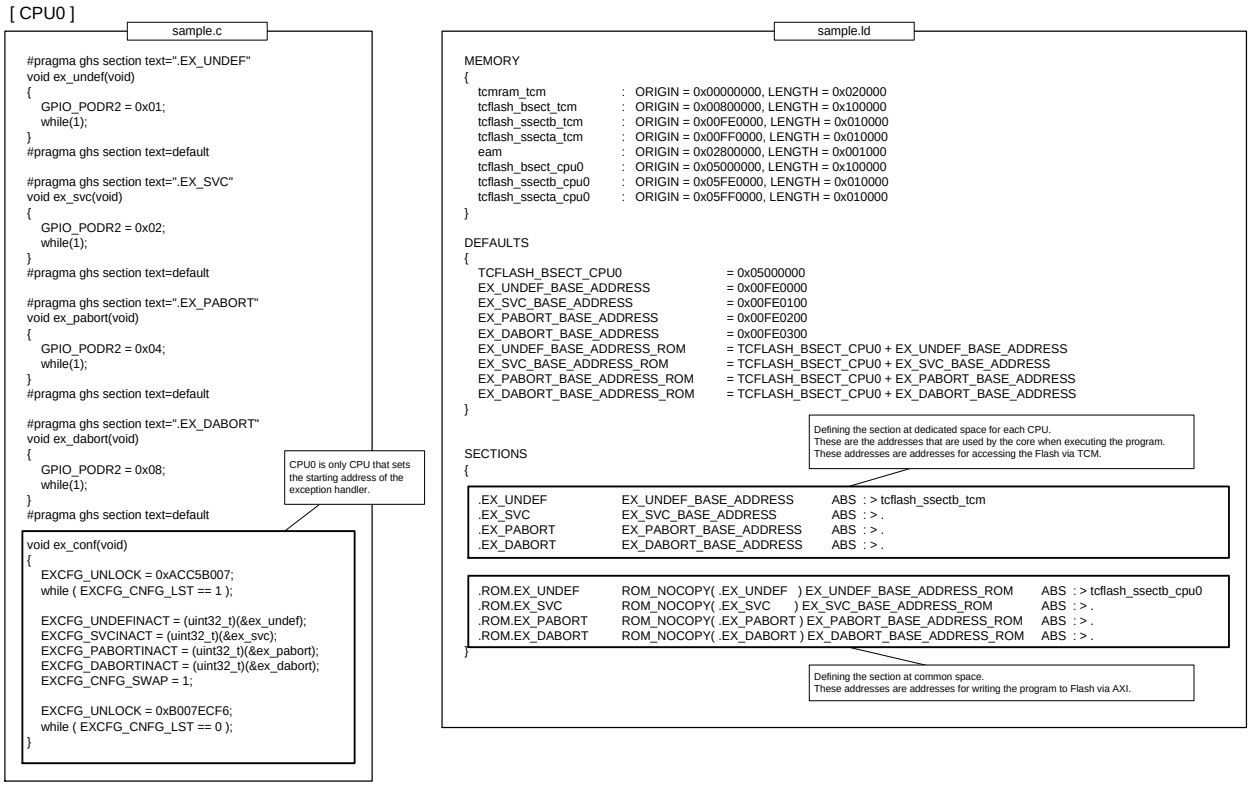


Figure 11. Example of Setting the Starting Address of the Exception Handler (CPU1)

[CPU1]

sample.c

```
#pragma ghs section text=".EX_UNDEF"
void ex_undef(void)
{
    GPIO_PODR2 = 0x10;
    while(1);
}
#pragma ghs section text=default

#pragma ghs section text=".EX_SVC"
void ex_svc(void)
{
    GPIO_PODR2 = 0x20;
    while(1);
}
#pragma ghs section text=default

#pragma ghs section text=".EX_PABORT"
void ex_pabort(void)
{
    GPIO_PODR2 = 0x40;
    while(1);
}
#pragma ghs section text=default

#pragma ghs section text=".EX_DABORT"
void ex_dabort(void)
{
    GPIO_PODR2 = 0x80;
    while(1);
}
#pragma ghs section text=default
```

Starting address of the exception handler isn't set by CPU1. Therefore, have to be prevented from being deleted by the optimization because these functions aren't referenced by CPU1.

sample.ld

```
MEMORY
{
    tcmram_tcm              :  ORIGIN = 0x00000000, LENGTH = 0x020000
    tcflash_bsect_tcm       :  ORIGIN = 0x00800000, LENGTH = 0x100000
    tcflash_ssectb_tcm      :  ORIGIN = 0x00FE0000, LENGTH = 0x010000
    tcflash_ssecta_tcm      :  ORIGIN = 0x00FF0000, LENGTH = 0x010000
    eam                     :  ORIGIN = 0x02800000, LENGTH = 0x001000
    tcflash_bsect_cpu1      :  ORIGIN = 0x07000000, LENGTH = 0x100000
    tcflash_ssectb_cpu1     :  ORIGIN = 0x07FE0000, LENGTH = 0x010000
    tcflash_ssecta_cpu1     :  ORIGIN = 0x07FF0000, LENGTH = 0x010000
}

DEFAULTS
{
    TCFLASH_BSECT_CPU0              = 0x07000000
    EX_UNDEF_BASE_ADDRESS           = 0x00FE0000
    EX_SVC_BASE_ADDRESS             = 0x00FE0100
    EX_PABORT_BASE_ADDRESS          = 0x00FE0200
    EX_DABORT_BASE_ADDRESS          = 0x00FE0300
    EX_UNDEF_BASE_ADDRESS_ROM       = TCFLASH_BSECT_CPU0 + EX_UNDEF_BASE_ADDRESS
    EX_SVC_BASE_ADDRESS_ROM         = TCFLASH_BSECT_CPU0 + EX_SVC_BASE_ADDRESS
    EX_PABORT_BASE_ADDRESS_ROM      = TCFLASH_BSECT_CPU0 + EX_PABORT_BASE_ADDRESS
    EX_DABORT_BASE_ADDRESS_ROM      = TCFLASH_BSECT_CPU0 + EX_DABORT_BASE_ADDRESS
}

SECTIONS
{
    .EX_UNDEF       EX_UNDEF_BASE_ADDRESS       ABS     : > tcflash_ssectb_tcm
    .EX_SVC         EX_SVC_BASE_ADDRESS         ABS     : > .
    .EX_PABORT      EX_PABORT_BASE_ADDRESS      ABS     : > .
    .EX_DABORT      EX_DABORT_BASE_ADDRESS      ABS     : > .

    .ROM.EX_UNDEF   ROM_NOCOPY( .EX_UNDEF   ) EX_UNDEF_BASE_ADDRESS_ROM     ABS : > tcflash_ssectb_cpu1
    .ROM.EX_SVC     ROM_NOCOPY( .EX_SVC     ) EX_SVC_BASE_ADDRESS_ROM       ABS : > .
    .ROM.EX_PABORT  ROM_NOCOPY( .EX_PABORT  ) EX_PABORT_BASE_ADDRESS_ROM    ABS : > .
    .ROM.EX_DABORT  ROM_NOCOPY( .EX_DABORT  ) EX_DABORT_BASE_ADDRESS_ROM    ABS : > .
}
```

Defining the section at dedicated space for each CPU. These are the addresses that are used by the core when executing the program. These addresses are addresses for accessing the Flash via TCM.

Defining the section at common space. These addresses are addresses for writing the program to Flash via AXI.

Figure 12. Example of Setting the CAN

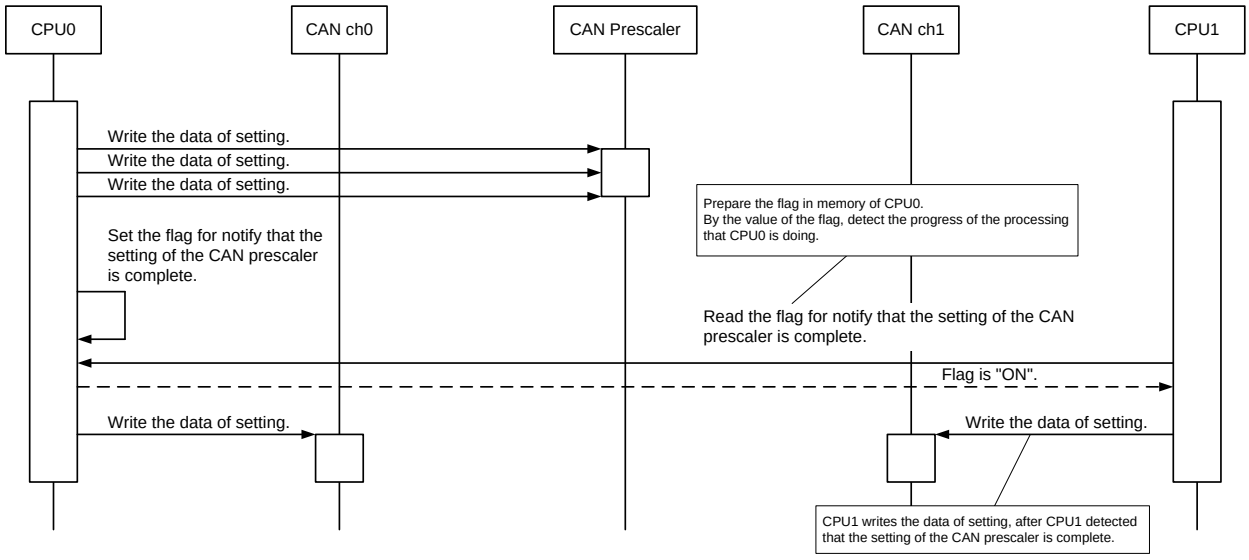
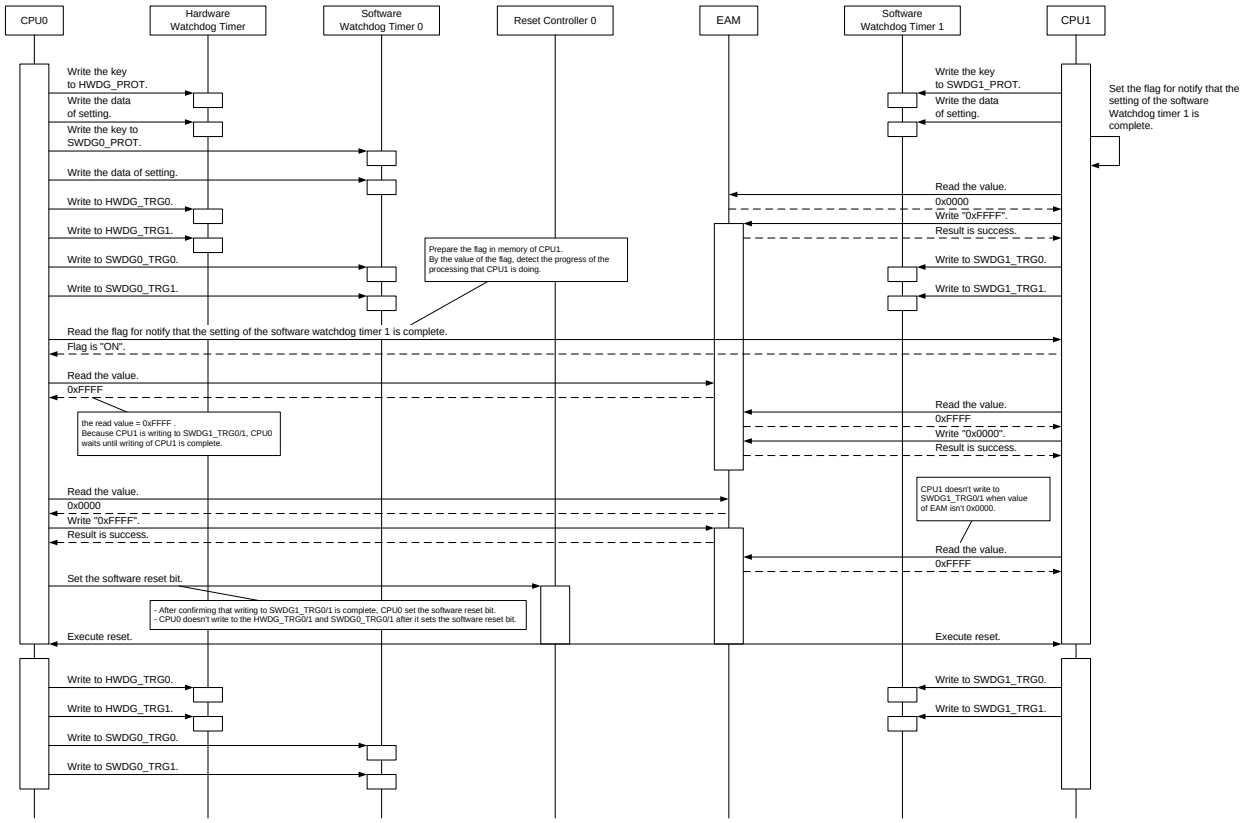


Figure 13. Example of Executing the Software Reset



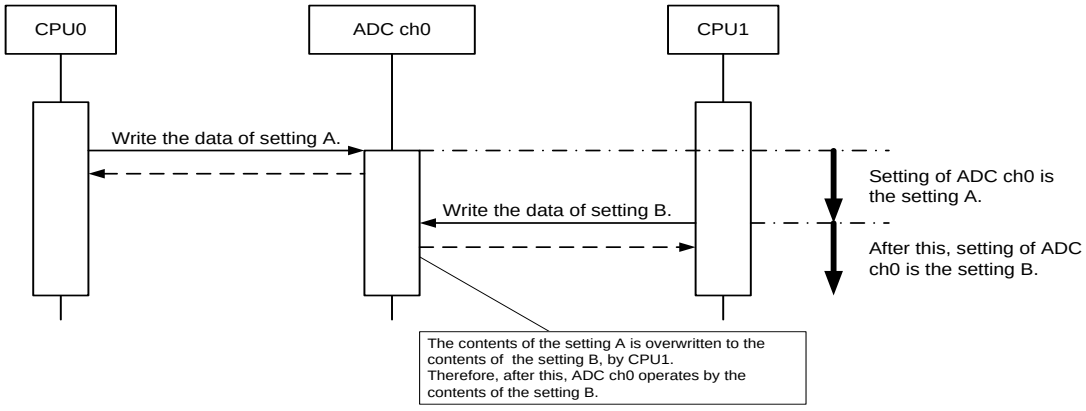
4.2 If the Recommended Method Isn't Done

There is possibility that problems occur, if the method described in “4.1Recommended Method of Using the Peripheral” isn't done. About the problem that may occur, are described in the following.

4.2.1 Case of Setting the Peripheral

If multiple cores set the different data on the same channel of same peripheral, the data set later are valid. That is, the peripheral operates with the setting that is set at the last. The example is described in Figure 14.

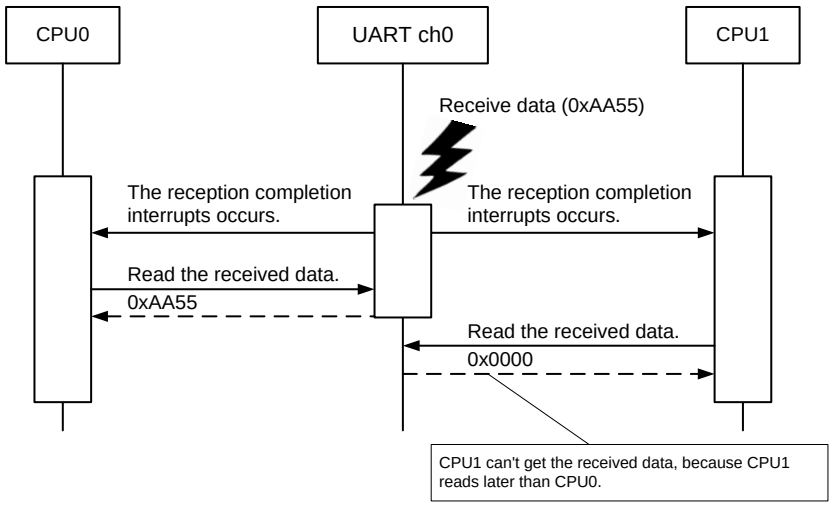
Figure 14. Example of Setting by Multiple Cores



4.2.2 Case of Reading from the Peripheral

There are registers that the read value changes by order which it was read, if multiple cores read the data from the same channel of the same peripheral. From this fact, the read result depends on the timing of reading. The example is described in Figure 15.

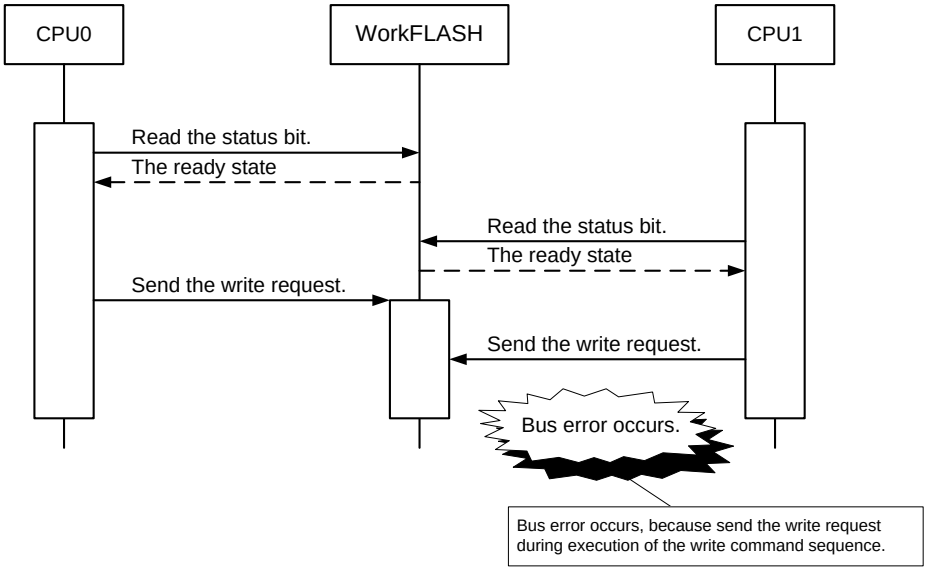
Figure 15. Example of Reading by Multiple Cores



4.2.3 Case of Using Common Peripheral

By timing, there is possibility that the status of the peripheral is changed unusable state after checking it, if multiple cores use the same channel of the same peripheral. The example is described in Figure 16

Figure 16. Example of Using Common Peripheral



4.2.4 Case of Setting the Starting Address of the Exception Handler

It is able to setting the starting address of the exception handler by using the register of Table 9.

The address is used by all cores without relying on core that has set the starting address of the exception handler.

Ex.) If the exception occurs in CPU1, when the start address of the exception handler has set by CPU0

1. The start address of the exception handler is set to "0x00FE0000" by CPU0.
2. The exception occurs in CPU1 after the processing of No.1 has been completed.
3. Processing of CPU1 jumps to the "0x00FE0000". (CPU0 continues the normal process, because the exception has not occurred in CPU0.)

If the jump destination address is not the start address of the processing of the exception handler, it may cause undesired operation.

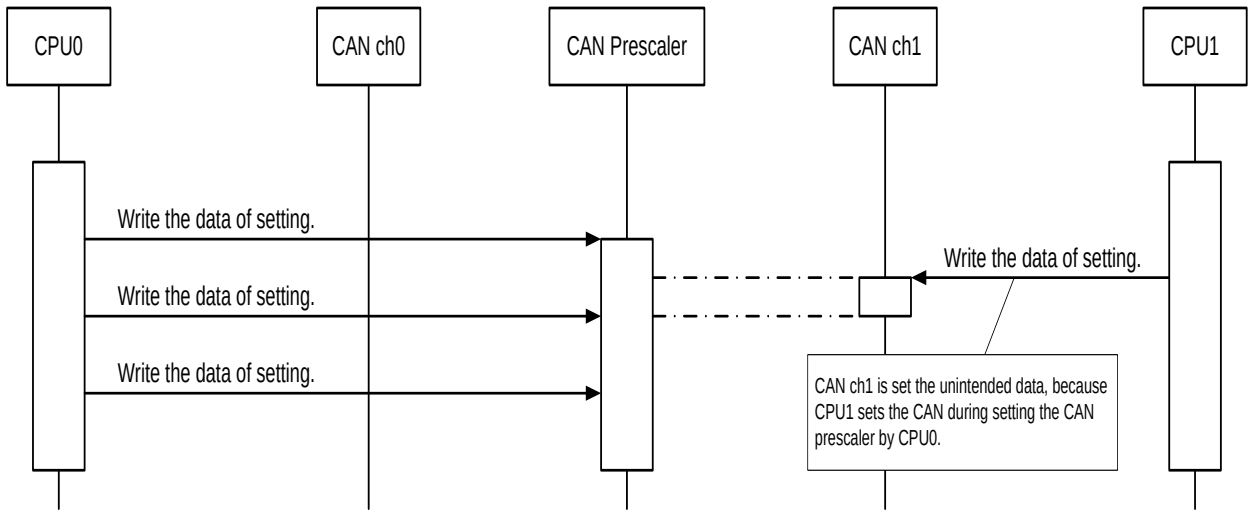
Table 9. Registers for Setting the Starting Address of the Exception Handler

Abbreviated Register Name	Register Name
EXCFG_UNDEFINACT	EXCFG Inactive Set - Undefined Instruction Vector Register
EXCFG_SVCINACT	EXCFG Inactive Set - Supervisor Call Vector Register
EXCFG_PABORTINACT	EXCFG Inactive Set - Prefetch Abort Vector Register
EXCFG_DABORTINACT	EXCFG Inactive Set - Data Abort Vector Register
EXCFG_UNDEFACT	EXCFG Active Set - Undefined Instruction Vector Register
EXCFG_SVCACT	EXCFG Active Set - Supervisor Call Vector Register
EXCFG_PABORTACT	EXCFG Active Set - Prefetch Abort Vector Register
EXCFG_DABORTACT	EXCFG Active Set - Data Abort Vector Register

4.2.5 Case of Setting the CAN

It isn't able to set the CAN during setting the CAN prescaler, by the configuration of the hardware. If set the CAN while setting the CAN prescaler, the set value changes to an unintended value. The example is described in Figure 17.

Figure 17. Example of Setting to CAN While Setting CAN Prescaler



4.2.6 Case of Executing the Software Reset

The protection register of hardware watchdog and software watchdog is not cleared by soft reset (software reset). Those trigger 0/1 register is also the same. Therefore, to continue the sequence at the time of reset even after it has been reset. If the software reset is executed while the sequence of Table 10 is executing, sequence violation occurs at around the reset.

The example of executing software reset while executing the watchdog register write protection sequence is described in Figure 18 and Figure 19.

The example of executing software reset while executing the watchdog counter clear protection trigger sequence is described in Figure 20 and Figure 21

The example of executing the watchdog counter clear protection trigger sequence after setting the software reset bit is described in Figure 22 and Figure 23.

Table 10. Sequences for Watchdog

Sequence	Function
Watchdog register write protection sequence	Release protection of the register for setting the watchdog.
Watchdog counter clear protection trigger sequence	Clear the watchdog counter.

Figure 18. Example of Violating the Register Write Protection Sequence of Hardware Watchdog

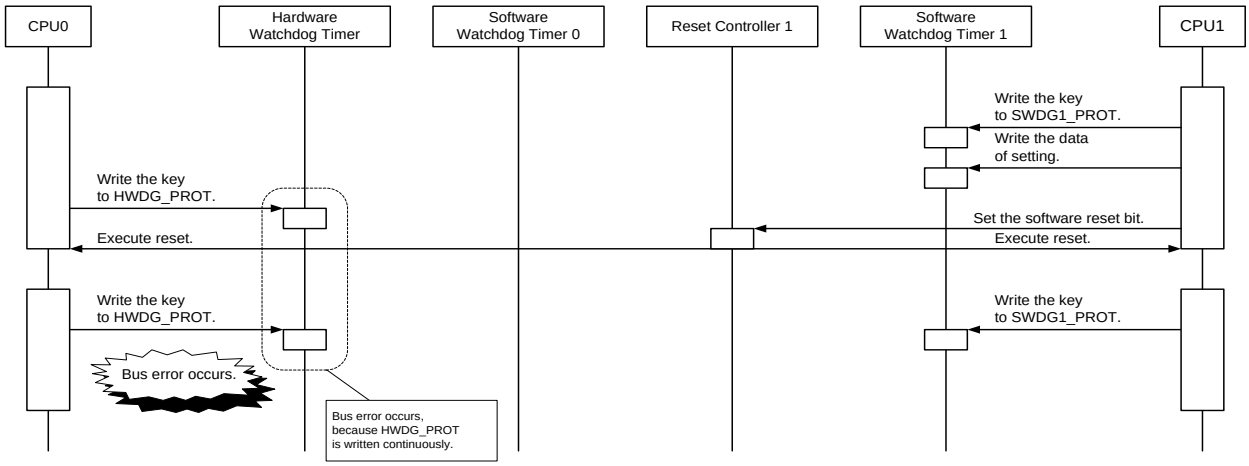


Figure 19. Example of Violating the Register Write Protection Sequence of Software Watchdog

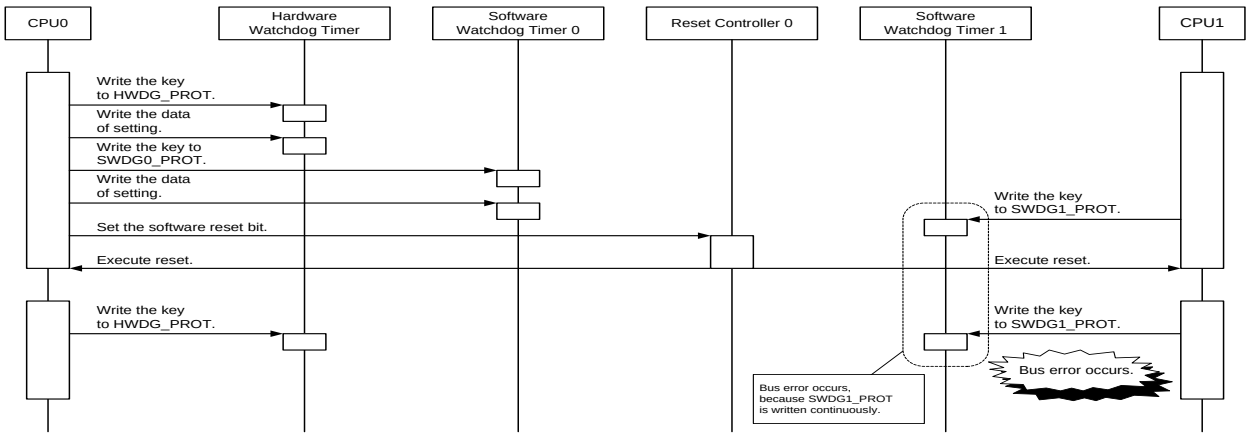


Figure 20. Example of Violating the Hardware Watchdog Counter Clear Protection Trigger Sequence

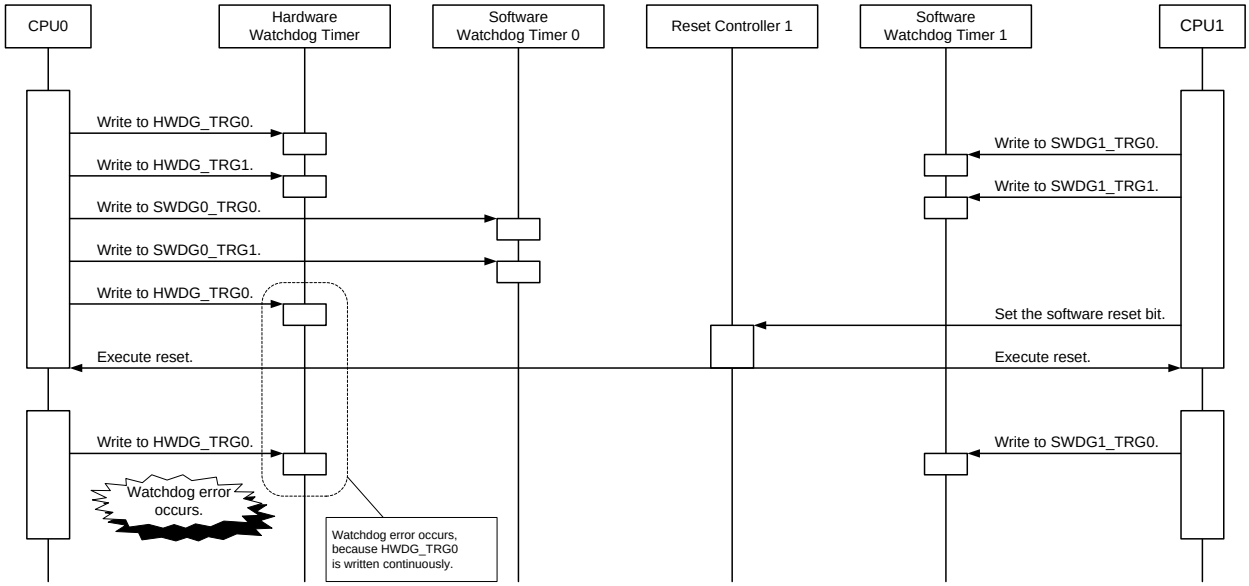


Figure 21. Example of Violating the Software Watchdog Counter Clear Protection Trigger Sequence

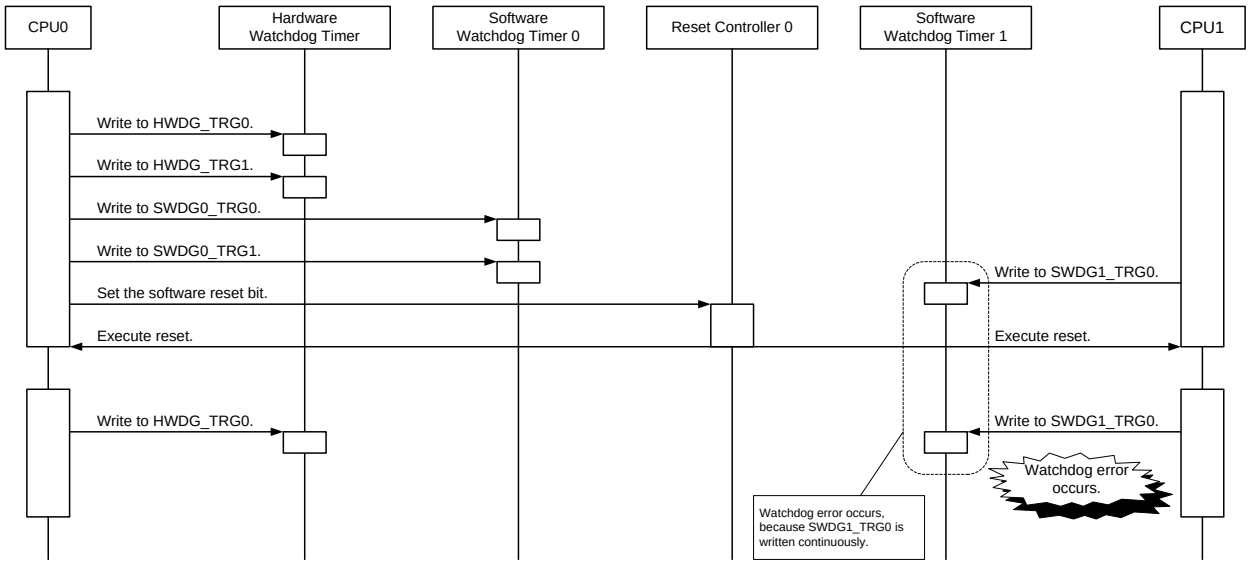


Figure 22. Example of Executing Sequence after Setting Software Reset Bit (Hardware Watchdog)

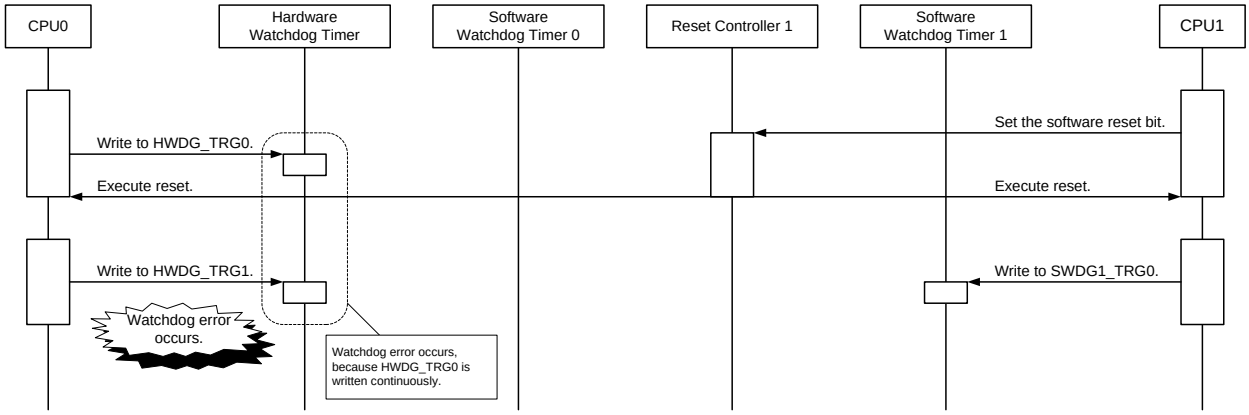
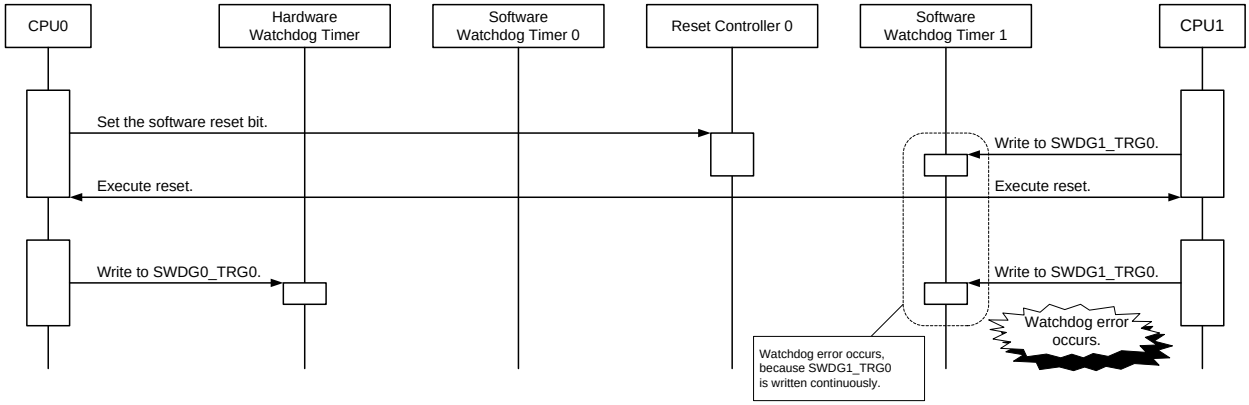


Figure 23. Example of Executing Sequence after Setting Software Reset Bit (Software Watchdog)



5 Reference

1. 32-BIT MICROCONTROLLER Spansion Traveo Family MB9D560 HARDWARE MANUAL
2. 32-bit Microcontroller MB9D560 DATA SHEET

A Appendix

A.1 How to Take Synchronization between Cores

The three examples are described in the following. These examples are method of taking synchronization between cores.

1. Using the flag by the memory or the register (Figure 24).
2. Using the semaphore by EAM (Figure 25).
3. Using the inter-processor communication unit (Figure 26).

Figure 24. Example of Taking the Synchronization by Flag

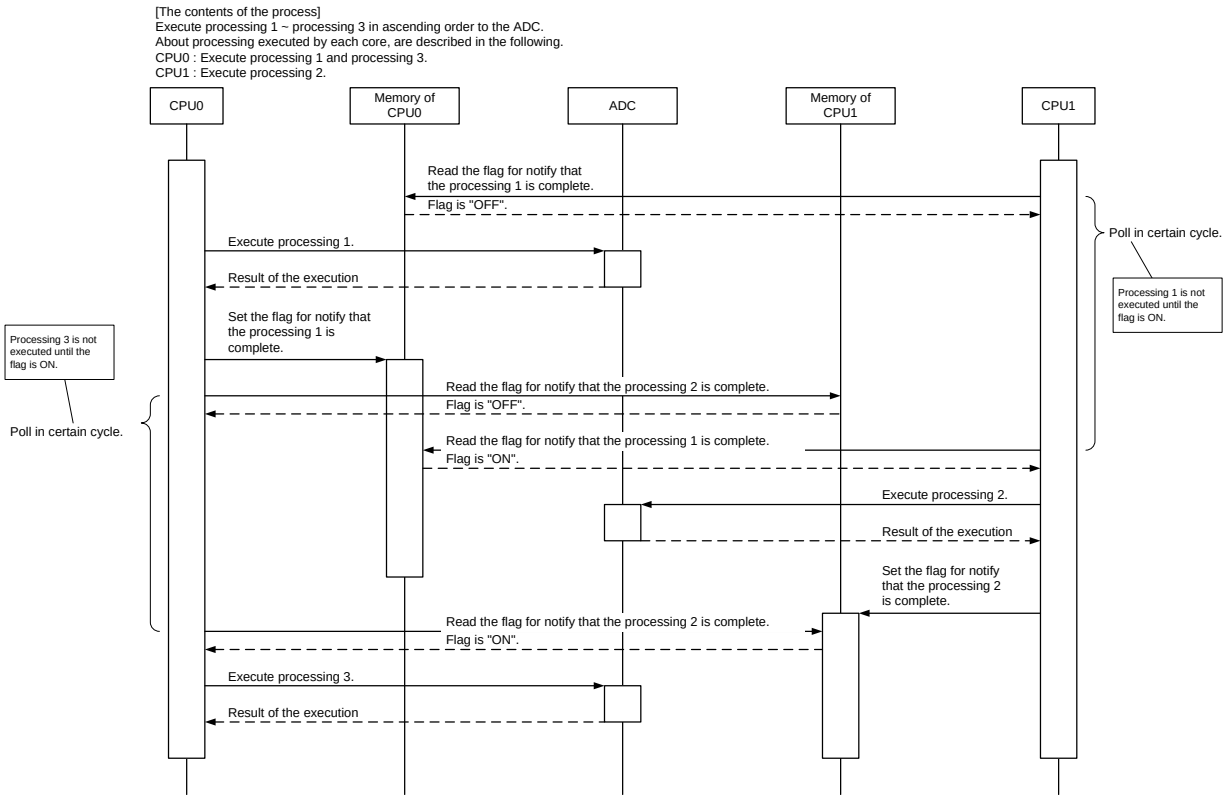


Figure 25. Example of Taking the Synchronization by Semaphore

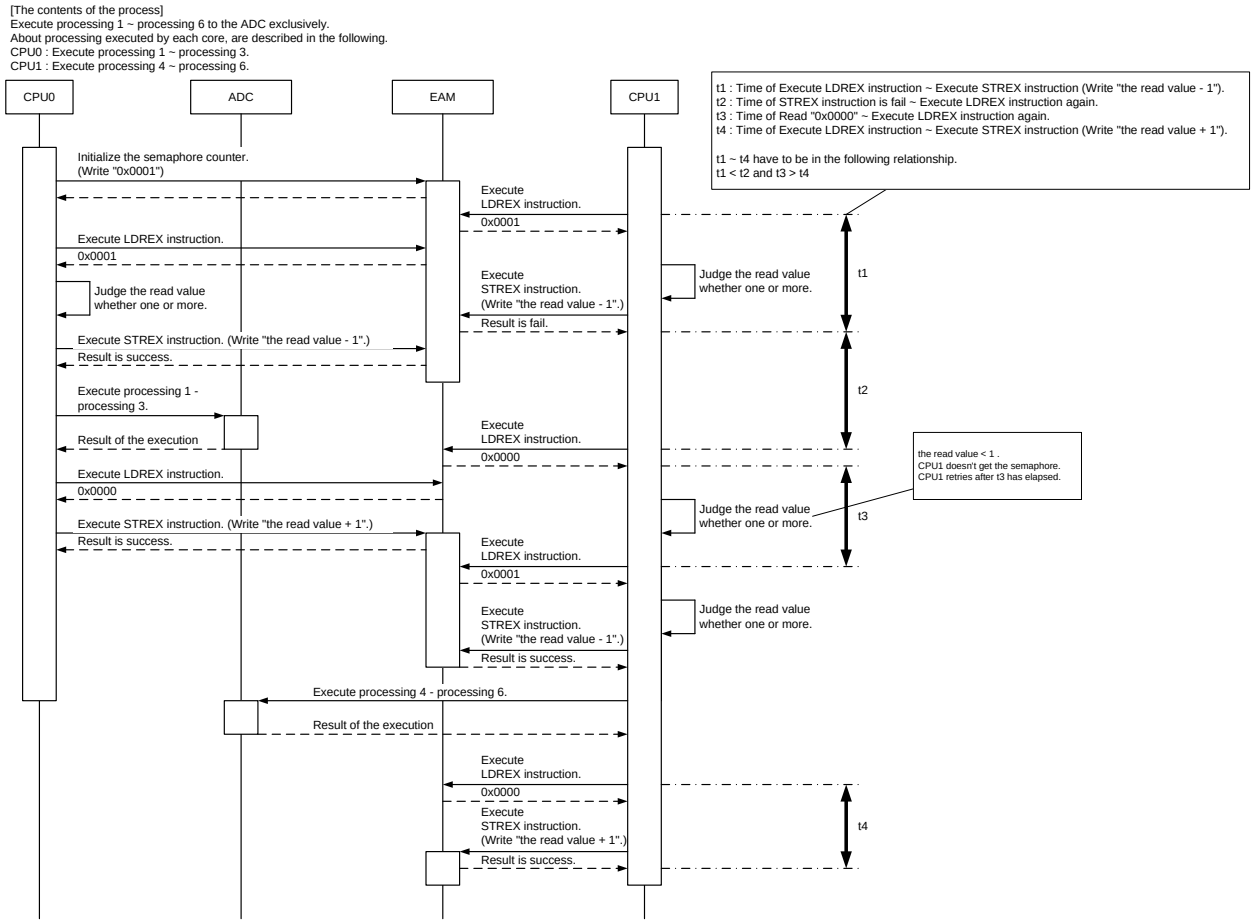
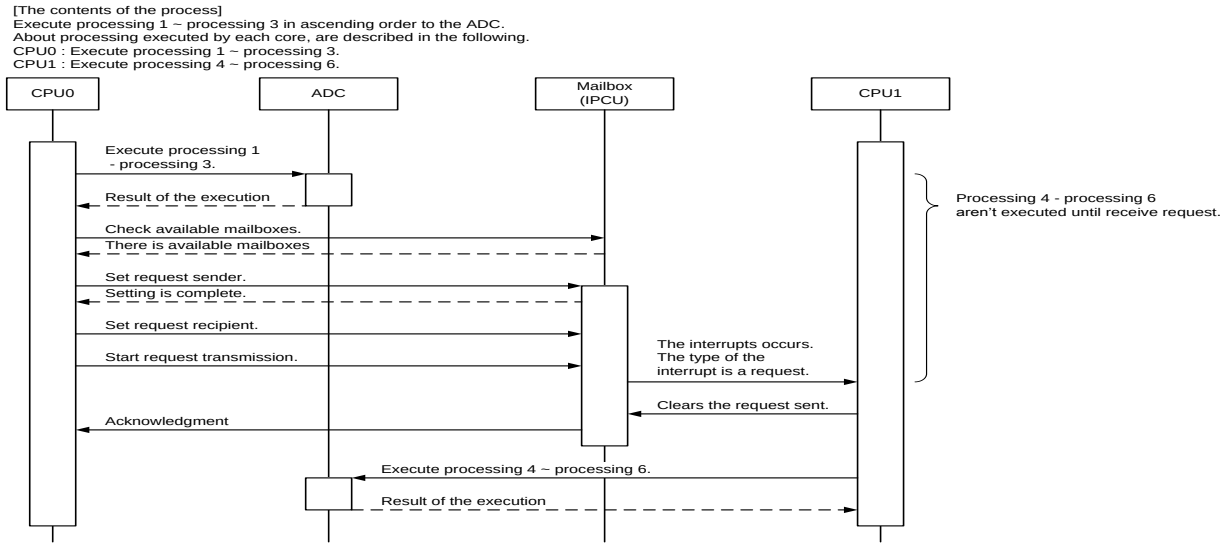


Figure 26. Example of Taking the Synchronization by Inter-processor Communication Unit



Document History

Document Title: AN204448 - Dual CPU Operation on Traveo™ Family, MB9D560 Series Microcontroller

Document Number: 002-04448

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	KHAS	07/16/2015	Initial Release
*A	5040683	KHAS	12/08/2015	Migrated Spansion Application Note from MB9D560_AN708-00007-1v0-E to Cypress format
*B	5874697	AESATMP8	09/06/2017	Updated logo and Copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709