SH7216 Group

Updating Program Code by Reprogramming Flash Memory in User Program Mode Using MMC

R01AN0769EJ0101
Rev.1.01
Jun. 15, 2012

Introduction

This application note describes an example to update the program codes by reprogramming the on-chip flash memory in user program mode using the MMC in the SH7216.

The features of the example to update the program codes in this application note are described below.

- Reprograms in the on-chip flash memory area using the program file data for updating in Motorola S-record format which is stored in the MMC.
- Reboots after reprogramming (updating) to execute the updated program.
- Checksum is implemented to handle the failure on reprogramming caused by the unintentional processing disruption and other causes.

Target Device

SH7216

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

Contents

1. Specifications

In this application, program is updated by reprogramming the on-chip flash memory using user program mode.

The program in the on-chip flash memory area is reprogrammed by the program file for updating in Motorola S-record format that is stored in the MMC. After reprogramming, the watchdog timer resets the flash memory to execute the updated program.

Table 1 lists the peripheral functions for use and their applications. Figure 1 shows the system configuration.

Table 1 Peripheral Functions and Their Applications

Peripheral Function	Application
Flash memory (on-chip flash memory)	Program storage area
Sequencer dedicated to the on-chip flash memory (FCU)	Reprogram the on-chip flash memory
Serial communication interface including FIFO (SCIF)	Message communication
Renesas serial peripheral interface (RSPI)	Read from the MMC
Watchdog timer (WDT)	Reboot the system after reprogramming

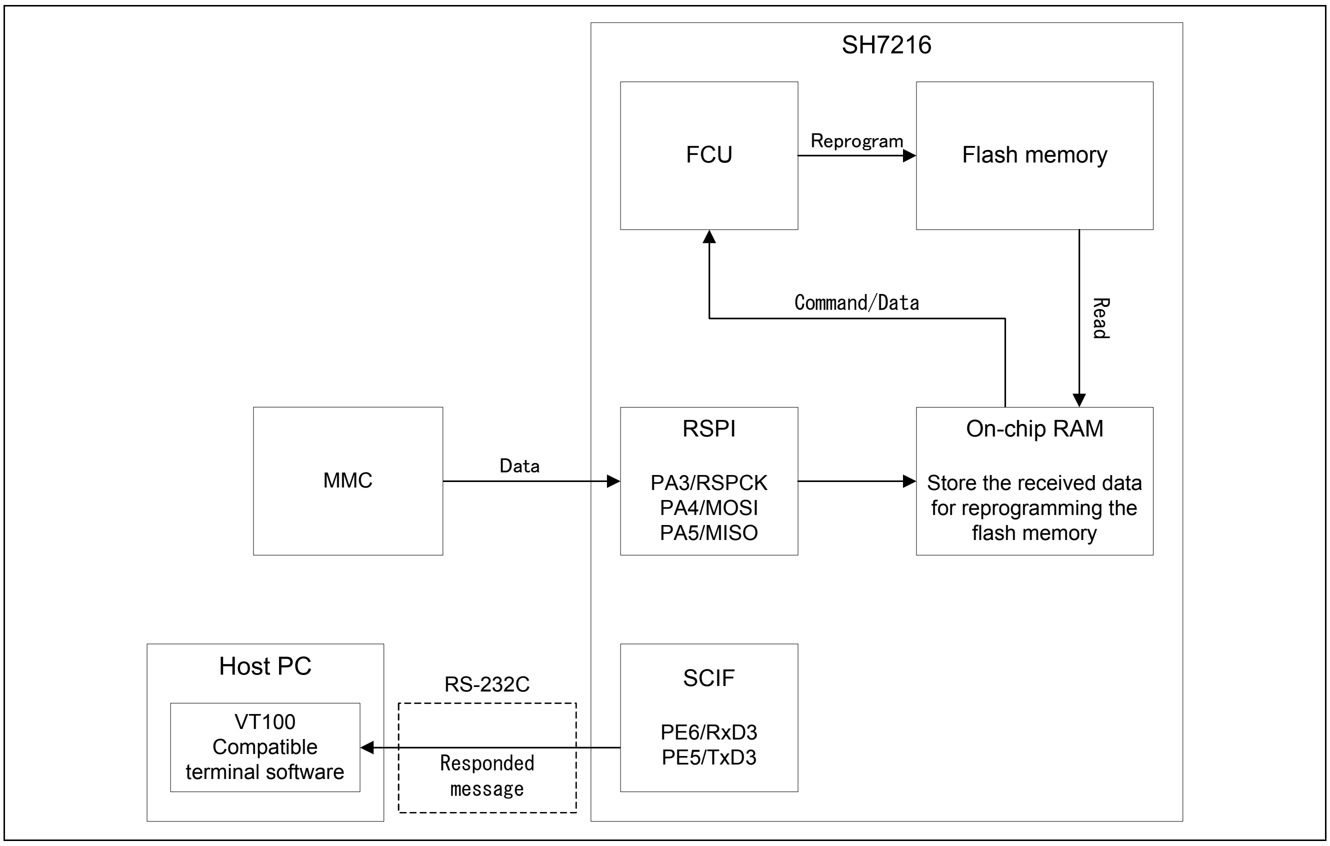


Figure 1 System Configuration

2. Operation Conditions

The sample code accompanying this application note has been run and confirmed under the conditions below.

Table 2 Operation Conditions

Item	Contents
MCU used	SH7216
Operating frequency	Internal clock (Iφ): 200MHz Bus clock (Bφ): 50MHz Peripheral clock (Pφ): 50MHz
Operating voltage	3.3V (Vcc)
Integrated development environment	Renesas Electronics Corporation High-performance Embedded Workshop Ver.4.08.00
C compiler	Renesas Electronics Corporation SuperH RISC engine family C/C++ compiler package V.9.03 Release 00
	Compile options: -cpu=sh2afpu -fpu=single -include="$(WORKSPDIR)\inc","$(WORKSPDIR)\src\mmc" -object="$(CONFIGDIR)\$(FILELEAF).obj" -debug -ifunc -gbr=auto -chgincpath -errorpath -global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1 - nologo
Operating mode	User program mode
Communication setting in the terminal software	• 9600bps • Data length: 8 bits • Parity: None • One stop bit • Flow control: None
Sample code version	1.00
Board used	R0K572167C001BR (CPU board) R0K572167B000BR (expansion board)
Tool used	VT100 compatible terminal software
Media used	HITACHI MultiMediaCard^{TM} (32MB)

3. Related Application Notes

For additional information associated with this document, refer to the following application notes.

- SH7216 Group Example of Initialization (document No.: REJ06B0899-0101)
- SH Family Simple Flash API for SH2 and SH2A (document No.: R01AN0719ER)
- SH7216 Group Updating Program Code by Reprogramming On-chip Flash Memory in User Program Mode (document No.: R01AN0686EJ)
- SH7216 Group Accessing MultiMediaCard Using the Renesas Serial Peripheral Interface (document No.: R01AN0039EJ)

4. Peripheral Functions

This chapter provides supplementary information on peripheral functions used in this application note. Refer to the SH7216 Group User's Manual: Hardware for basic information.

4.1 Flash Memory

The data received from the MMC are stored in the on-chip flash memory using the SH2, SH2A simple flash API.

The SH7216 includes the flash memory (user MAT) to store program codes. The flash memory varies according to the capacity as 1Mbyte, 768Kbytes, and 512Kbytes.

The user MAT is divided in the 8-Kbyte, 64-Kbyte, and 128-Kbyte blocks. User MATs are erased by these block units in user program mode.

Writing in the on-chip flash memory is enabled only in the erased area in multiples of 256bytes. The 256-byte boundary address is used for the programming start address.

In this application, the block 04 (H'0000 8000 to H'0000 9FFF) is allocated for updating data. The last 256byte area in this block (H'0000 9F00 to H'0000 9FFF) is used for checksum verification. The backup programs are stored in the block 05 (H'0000 A000 to H'0000 BFFF).

From now on, the block as the erase unit is described as "EB". For example, "EB04" represents the block 04 for erasing.

Figure 2 shows the user MAT block configuration.

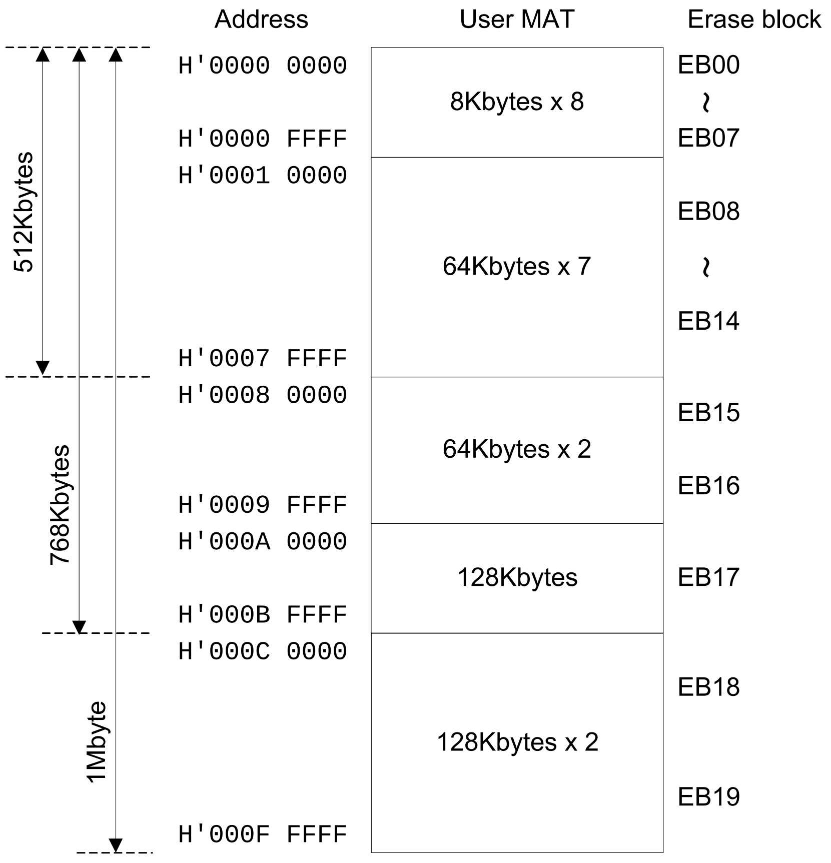


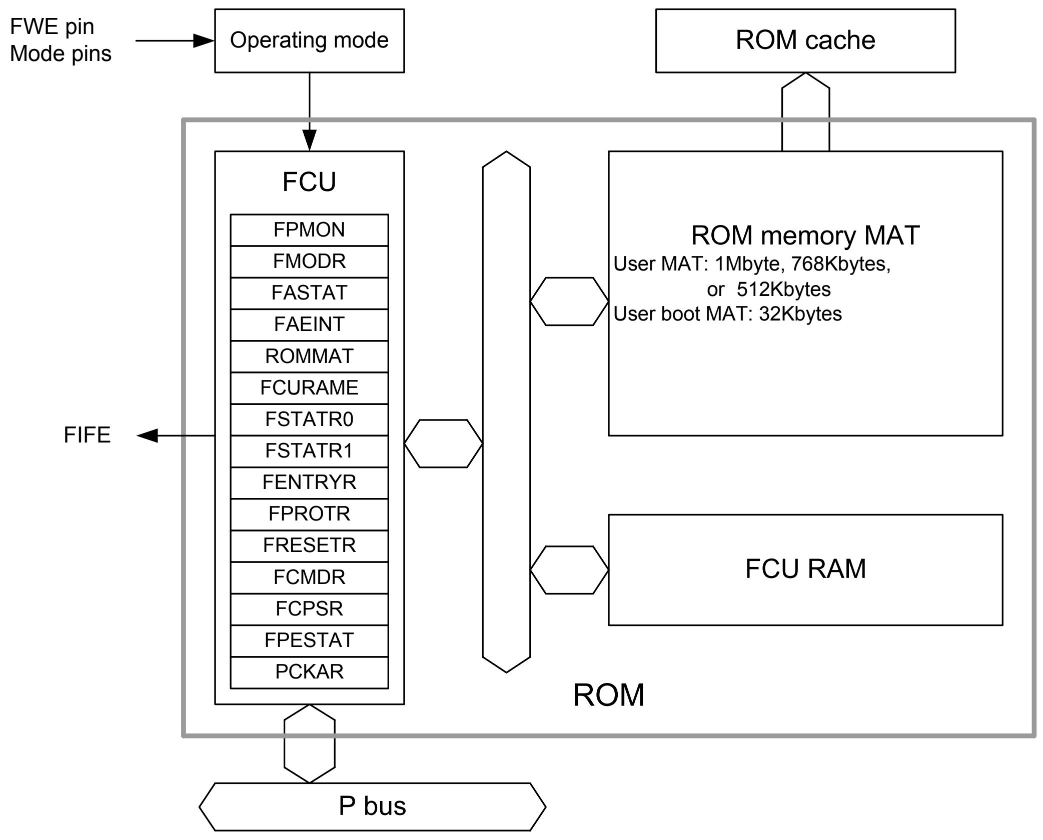
Figure 2 User MAT Block Configuration

4.2 On-chip Flash Memory Dedicated Sequencer (FCU)

SH2, SH2A simple flash API uses the flash memory sequencer (FCU) for writing/erasing data in the on-chip flash memory.

To use the FCU, the firmware for FCU (FCU firm) needs to be transferred to the FCURAM area. After that, writing to/erasing from the on-chip flash memory is enabled by issuing the FCU command.

Figure 3 shows the block diagram in the on-chip flash memory.



[Legend]

FPMON:	Flash pin monitor register
FMODR:	Flash mode register
FASTAT:	Flash access status register
FAEINT:	Flash access error interrupt enable register
ROMMAT:	ROM MAT select register
FCURAME:	FCURAM enable register
FSTATR0, 1:	Flash status registers 0 and 1
FENTRYR:	Flash P/E mode entry register
FPROTR:	Flash protect register
FRESETR:	Flash reset register
FCMDR:	FCU command register
FCPSR:	FCU processing switch register
FPESTAT:	Flash P/E status register
PCKAR:	Peripheral clock notification register
FIFE:	Flash interface error interrupt

Figure 3 On-chip Flash Block Diagram

4.3 Serial Communication Interface with FIFO (SCI)

The SCIF is available for two types of serial communication modes: the asynchronous communication mode and the clock synchronous communication mode. Each channel includes 16-stage transmit/receive FIFO register independently, which enables efficient and high-speed continuous communication.

For the details, refer to the "Section 17 Serial Communication Interface with FIFO (SCIF)" in the "SH7214 Group, SH7216 Group User’s Manual: Hardware".

4.4 Watchdog timer (WDT)

This application uses a watchdog timer to reset the flash memory after rewriting a sample code in it.

For the details, refer to the "Section 15 Watchdog timer (WDT)" in the "SH7214 Group, SH7216 Group User’s Manual: Hardware".

4.5 Renesas Serial Peripheral Interface (RSPI)

This application uses the RSPI to access the MMC in SPI mode, and to perform the SPI communication. The RSPI is capable of full-duplex synchronous, high-speed serial communication function with multiple processors and peripheral devices.

For the details, see the "Section 18 Renesas Serial Peripheral Interface (RSPI) " in the "SH7214 Group, SH7216 Group User’s Manual: Hardware".

5. Hardware

5.1 Pins Used

Table 3 lists the pins used in this application and their functions.

Table 3 Pins Used and Their Functions

Pin Name	I/O	Function
PA3/RSPCK	Output	RSPI clock output
PA4/MOSI	Output	RSPI master transmit data
PA5/MISO	Input	RSPI slave transmit data
PE5/TxD3	Output	SCIF serial data output
PE6/RxD3	Input	SCIF serial data input

6. Software

6.1 Operation Overview

This application reprograms data in the area in the on-chip flash memory using the program file data for updating in the Motorola S-record format sent from the MMC. The operation overview is described in this section.

6.1.1 Section Assignment

An access to the on-chip flash memory is disabled when reprogramming is in progress in the on-chip flash memory. Therefore, all the programs to be used during reprogramming the on-chip flash memory need to be transferred to the areas outside the on-chip flash memory. When assigning the section in this application, all the programs that are used during reprogramming are set to be transferred to the on-chip RAM. The details on the section assignment, refer to the "Section 3.4 SH7216 Group, SH7239 Series" in the "Simple Flash API for SH2 and SH2A".

Table 4 lists the programs and the sections to be used during reprogramming the on-chip flash memory in this application. For the details on processing in each programs, see the sections "6.6 Functions", "6.7 Function Specification", and "6.8 Processing Flowcharts".

Table 4 Programs and Sections Used during Reprogramming On-chip Flash Memory

Program	Function Name	ROM Section Name	RAM Section Name
Flash memory reprogramming processing	rom_write Enter_PE_Mode Exit_PE_Mode R_FlashErase R_FlashWrite	PFRAM	RPFRAM

For the failure in reprogramming (updating) caused by an unintentional interruption in the on-chip flash memory, this application allocates the separate section area to store the backup programs. The identical program processing are stored in the reprogramming area and the extra storage area in their default setting before receiving data. Table 5 shows the correspondence between these areas.

Table 5 Correspondence between Reprogramming Area and Extra Area

Area	Start Address (Block)	Stored Function	ROM Section Name
Reprogramming area	H’0000 8000（EB04）	flash_write_sample	MasterPRG
Extra area	H’0000 A000（EB05）	flash_write_spare	SparePRG

6.1.2 On-chip Flash Memory Reprogramming Operation Overview

Figure 4 shows the on-chip flash memory reprogramming overview in this application.

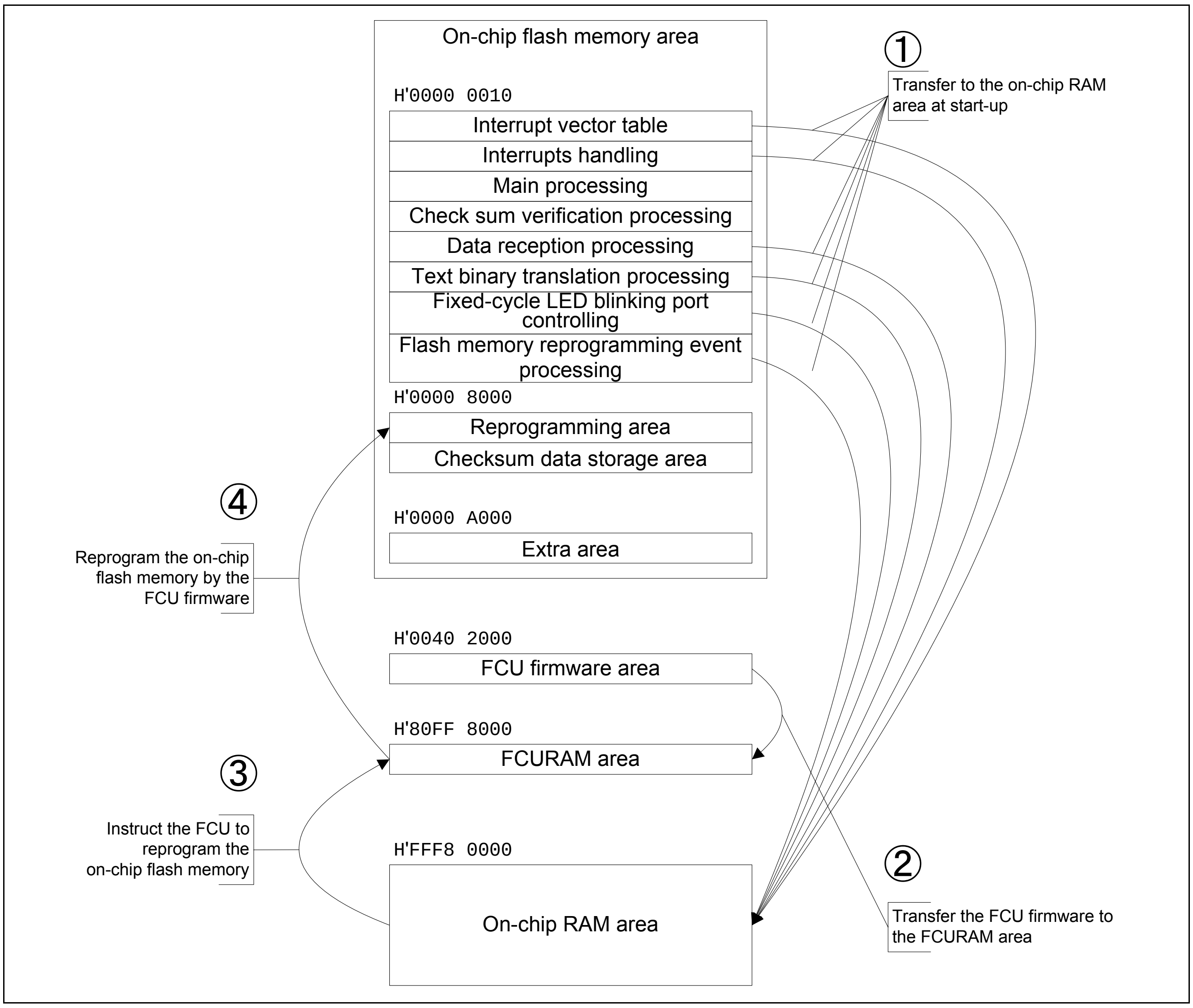

On-chip flash memory area
H'0000 0010
Interrupt vector table
Interrupts handling
Main processing
Check sum verification processing
Data reception processing
Text binary translation processing
Fixed-cycle LED blinking port controlling
Flash memory reprogramming event processing
H'0000 8000
Reprogramming area
Checksum data storage area
H'0000 A000
Extra area
H'0040 2000
FCU firmware area
H'80FF 8000
FCURAM area
H'FFF8 0000
On-chip RAM area
①
Transfer to the on-chip RAM area at start-up
②
Transfer the FCU firmware to the FCURAM area
③
Instruct the FCU to reprogram the on-chip flash memory
④
Reprogram the on-chip flash memory by the FCU firmware

Figure 4 Operation Overview of Reprogramming the On-chip Flash Memory

1. After clearing the reset, transfers the program specified in dbsct.c (the ROM section) to the on-chip RAM area.
2. Transfers the firmware in the on-chip flash memory dedicated sequencer (FCU) for reprogramming the on-chip flash memory from the FCU firmware area to the FCURAM area.
3. Instructs the FCU firmware transferred to the FCURAM area to reprogram the on-chip flash memory. (In this application, calling the simple flash API instructs the FCU firmware to perform reprogramming.)
4. Executes reprogramming the on-chip flash memory by the FCU firmware.

6.1.3 From a Start-up to a Normal Operation

Starting up the system through necessary initializations in the main processing, transmit the message, "Generate IRQ6 interrupt for transition to flash programming event" to the host PC. Then, call the "checksum verification processing" function to determine by using checksum if the program codes in the reprogramming area (EB04) do not contain any defects.

For checksum verification, this application uses the "program code size" and the "checksum data" which is given by single-byte size in the program. In the checksum verification processing function, the data sizes are added from the start address (H'0000 8000) in the reprogramming area by single-byte size for the same number of times as the program size times. When comparing the result of this addition with the checksum verification data calculated at receiving a data (stored in the last 256-byte area in EB04; details are referred to in the following section), if these data make a match, the program in the reprogramming area is executed. If they make a mismatch, the program in the extra area is executed. Note that a dummy checksum data has been stored in the reprogramming area to make a match in checksum test without fail at an initial start-up.

The initially stored program blinks the LED using a compare match timer (CMT). It calls the "fixed-cycle LED blinking port controlling" function in the compare match interrupt that is generated in 100ms cycle to update the LED blinking display pattern.

6.1.4 Flash Memory Reprogramming Event Processing by IRQ6 Input

When the IRQ6 interrupt is generated by the IRQ6 input (detects a trailing edge /pushes the IRQ6 switch on the board) according to the message transmitted to the host PC, the processing is transited to the flash memory reprogramming event processing.

In the flash memory reprogramming event processing, firstly, the message "--> IRQ6 detected!" is transmitted to erase the reprogramming area, EB04. Then, the message "Send subroutine code to update program in Motorola S-record format Check card insert " is transmitted to the host PC to switch to standby state waiting for the MMC insertion.

When the MMC insertion is detected, the program file data in Motorola S-record format for updating stored in the root directory is read. After reading by the sectors, the after mentioned data analysis is performed for every single byte to store data in the write data storage buffer (write buffer). Furthermore, after analyzing one-sector data, the data in the write buffer is stored in the on-chip flash memory. This processing is repeated until the final data is stored in the on-chip flash memory. Additionally, in this application, the write buffer has double structure. Writing in the on-chip flash memory is carried by a single-buffer unit.

6.1.5 Receiving Program File for Updating Stored in MMC

This application searches and reads the specified file (a.mot) from the MMC formatted in the FAT file system. As the sample codes in this application do not include the FAT library, they have the restrictions described in Table 7.

Table 6 MMC Restrictions

Item	Contents	Description
FAT type	FAT16	Does not support FAT12 and FAT32
Sector size	512bytes	
File name	a.mot	Make sure to save program files for updating in this name.
File path	H:\\a.mot	Varies depending on the environment in the host PC as "H" is the drive number. Save the file in the root directory.
File size	Within 1-cluster size	Cannot read files striding multiple clusters.
Detective entry in root directory	Within 1 sector	May not detect correctly when many files are stored in the root directory besides the program file for updating as only the first sector in the root directory is detected.

6.1.6 Data Analysis

When a data is received for one sector size from the MMC, the data is stored in the buffer by the single-byte size. Detecting the linefeed code, the data stored in the analysis buffer is determined to be as a record filling one row. Through the data analysis processing described below, the necessary data for updating is extracted.

The first character in the analysis buffer is checked if it is "S". If it is "S", the data is regarded as Motorola S-record format. Then the second character is to be analyzed. If the first character is not "S", the data is determined to be invalid, and another data is stored from the start of the analysis buffer.

Figure 5 shows the Motorola S-record format data. Referring to it, the data analysis processing is explained below. The data shown in the Figure 5 is colored according to the functions.

```
S00E000073616D706C6520206D6F74DF
S11380007FFC04E0420AE50001E00014E72566432
S113801076F6E065E23F315903F434510076267078
S11380208061242084642F62C90FCB30806466F29F
S11380308465C90FCB30806506E042008466C90FB1
S1138040CB3080668466C9F0CB0380668467C90F31
S1138050CB3080678467C9F0CB0380670000BEBB68
S1138060816C00005F5D816D06E0420000005F5D91
                   ≀
S10B82C000090009000B7F0412
S9030000FC
```

Figure 5 Sample Data in Motorola S-record Format

- In the first row, the second character 0 in red indicates a header record which contains no program data. When receiving a header record, the processing returns to standby state until the next record is received in the second row.
- When the second row is verified as the Motorola S-record format by the first character "S" in black, the second character 1 in red indicates that the second row is a data record. In the Motorola S-record format, the second character from the top given by numeral in red indicates the record type.
- The third and forth characters in blue indicate 1-byte record length in hexadecimal notation. The four characters from the fifth to eighth in green indicate the lower 2bytes in the storage address to store the initial data in the record.
- The ninth character and later in orange represent the data part that indicates 1byte with every two characters. In data analysis processing, every set of two characters after ninth character in orange is converted in a binary data (the "text binary translation processing" function is called). The converted 1byte data is stored in the write buffer sequentially. At the same time, the 1byte data is added (as a checksum verification data) to the previous data for checksum test after conversion, and furthermore the number of the data is counted as the program code size. When this processing is repeated before the last two characters in black, the data analysis processing ends
- 9 in the second character in the record data indicates the end of the record. See the last row in Figure 5. When determining the record end, the reception processing is not performed and analysis processing is terminated. The data reception ends without storing the received data. However, at this time, if the data size in the write buffer is smaller than 256bytes which is the unit for flash memory writing, H'FF should be added till the data size reaches 256bytes.

In this application, the write buffer has double structure of 256-byte size. Every time a write buffer is filled with the 256-byte data, the storage is switched to the other write buffer in the data analysis processing. When one of the write buffers becomes full, flash memory write is enabled, and the buffer data is written in the on-chip flash memory when the data analysis processing ends.

6.1.7 Processing after Data Analysis and Data Reprogramming

When determining the end of record by data analysis processing, and writing all the received data in the on-chip flash memory, the data calculated at data analysis for checksum verification is written in the on-chip flash memory. The checksum data includes a program code size and a checksum data for 2bytes each. In this application, the checksum data is stored in the last 256byte area (H'0000 7F00 to H'0000 7FFF) in the reprogramming area (EB04). But only 4bytes in this area are used practically. The program code size is stored in the area, H'0000 9F00 to H'0000 9F01, and the checksum data, in the area, H'0000 9F02 to H'0000 9F03.

After writing the data for checksum verification, a watchdog timer is set. The program becomes standby state waiting for a reset by timer counter overflow.

6.1.8 Communication Control Sequence

Figure 6 shows the communication control sequence in this application.

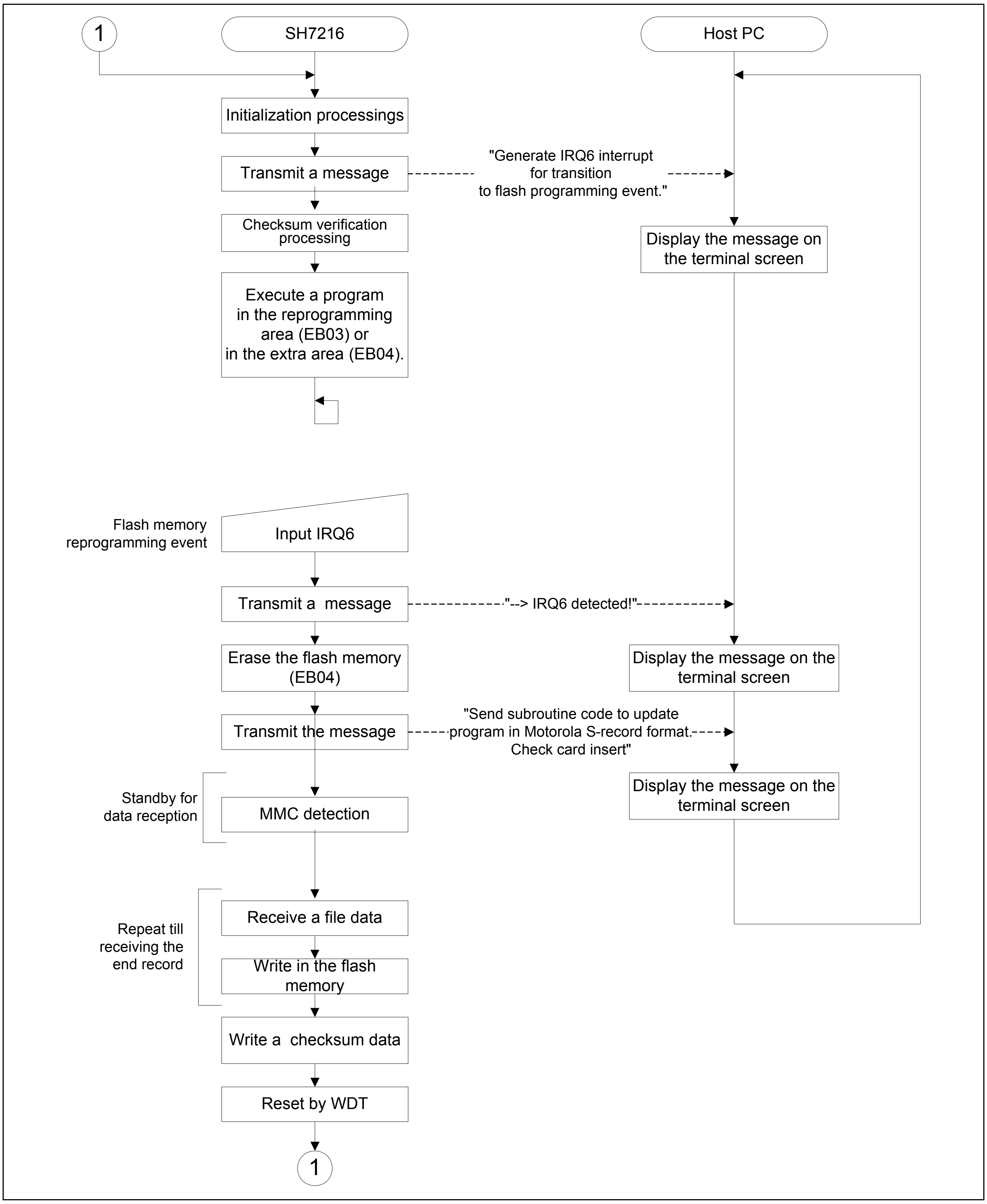

1
SH7216
Host PC
Initialization processings
Transmit a message
"Generate IRQ6 interrupt for transition to flash programming event."
Checksum verification processing
Display the message on the terminal screen
Execute a program in the reprogramming area (EB03) or in the extra area (EB04).
Flash memory reprogramming event
Input IRQ6
Transmit a message
"--> IRQ6 detected!"
Erase the flash memory (EB04)
Display the message on the terminal screen
Transmit the message
"Send subroutine code to update program in Motorola S-record format. Check card insert"
Display the message on the terminal screen
Standby for data reception
MMC detection
Repeat till receiving the end record
Receive a file data
Write in the flash memory
Write a checksum data
Reset by WDT
1

Figure 6 Communication Control Sequence

6.2 File Composition

Table 7 lists the file compositions in this application. Note that the files created automatically in the integrated development environment are excluded on the list.

Table 7 File Composition

File	Outline	Remarks
main.c	Main processing, Checksum verification processing, Flash memory reprogramming event processing, Data reception processing, Fixed-cycle LED blinking port control	Functions called by interrupts are also allocated
intprg.c	Interrupts handling	Calls interrupts handling functions by IRQ6 input and compare match
vecttbl.c	Interrupt vector table	---
Flash_API_SH7216.c	Simple flash API	For reprogramming on-chip flash memory by FCU firmware
Flash_API_SH7216.h	Simple flash API header file	ditto
siochar.c lowsrc.c	SCIF controlling	Used for RS-232C communication
port_LED_sample.c	Sample program for updating	The initial program to be stored in a reprogramming area and an extra area
mmc_api.c	MMC driver API	Allocates API to use the MMC driver
mmc.h	MMC driver header file	ditto
mmc_cmd.c	MMC command handling	Allocates functions for MMC command handling
mmc_cmd.h	MMC command handling header file	ditto
mmc_spi_bus.c	MMC SPI mode processing	Allocates functions to control the MMC in SPI mode
mmc_spi.h	MMC SPI mode processing header file	ditto
mmc_time.c	Software timer controlling	--
io_rspi_sh7216.c	RSPI controlling	Allocates functions for SPI communication

6.3 List of Constants

Table 8 lists the constants used in the sample code.

Table 8 Constants Used in the Sample Code

Constant Name	Setting Value	Contents
SLOT0	0	Slot number of the MMC
BLOCK_4	4	Block number in the reprogramming area
MAX_SCT_SIZE	512	Buffer size for reading sector data in the MMC
BUFFER_SIZE	256	Buffer size for flash data writing
LED_PATTERN_NUM	10	Number of LED blinking display pattern

6.4 List of Structures and Unions

Figure 8 shows the list of the structures and unions used in the sample code.

```
/* ---- Structures to control FAT Information ---- */
typedef struct{
        unsigned char     fat_type;          /* 4/6:FAT16, 11:FAT32 */
        unsigned long     mmc_bsr_area;      /* First sector in the BSR area */
        unsigned long     mmc_fat_area;      /* First sector in the FAT area*/
        unsigned long     mmc_rdir_area;     /* First sector in the root directory */
        unsigned long     mmc_data_area;     /* First sector in the data area */
        unsigned short    sctsz;             /* Sector size (byte count) */
        unsigned char     clussz;            /* Cluster size (sector count) */
        unsigned short    fatsz;             /* FAT area size (sector count) */
        unsigned char     fatnum;            /* Number of FAT areas */
        unsigned short    rdirnum;           /* Entry counts in the root directory */
        unsigned char     f_entry[32];       /* Entry information in the target file */
        unsigned short    f_clusno;          /* First cluster number in the target file */
        unsigned long     f_filesz;          /* Target file size */
}MMC_FSYS;
```

Figure 7 Structures and Unions

6.5 List of Variables

Table 9 lists the global variables. Table 10 lists the const type variables.

Table 9 Global Variables

Type	Variable Name	Contents	Function Used
MMC_FSYS	mmc_fsys	Structure to save the received FAT file system information	int_fcu_flash_write smpl_fopen search_file
unsigned char	mmc_ReadDataBuff0	Analysis data storage buffer 0 (array)	analyze_read_data
unsigned char	mmc_ReadDataBuff1	Analysis data storage buffer 1 (array)	analyze_read_data
unsigned char	sct_buff	MMC sector receive buffer (array)	int_fcu_flash_write smpl_fopen
int	f_WriteDataBuff0_Full	Write buffer 0 full flag	analyze_read_data int_fcu_flash_write
int	f_WriteDataBuff1_Full	Write buffer 1 full flag	analyze_read_data int_fcu_flash_write
int	f_status_EndRecord	End record reception flag	analyze_read_data int_fcu_flash_write
int	cnt_store_ReadData	Analysis data storage counter	analyze_read_data
int	cnt_store_WriteDataBuff	Write data storage counter	analyze_read_data
int	cnt_led_wink	LED blinking display pattern counter	int_cmt_led_control
unsigned short	chksm_size	Program code size of a write data	analyze_read_data int_fcu_flash_write
unsigned short	chksm_data	Checksum data of a write data	analyze_read_data int_fcu_flash_write
unsigned short	chksm_Mem	Dummy data for checksum verification in the reprogramming area at the initial start-up (array)	—
unsigned short	cmt_LedPattern	LED blinking display pattern data in the reprogramming area and the extra storage area at the initial start-up	int_cmt_led_control

Table 10 const-Type Variables

Type	Variable Name	Contents	Function Used
const char	msg_VT100_ClearScreen	VT100 compatible ESC sequence	main
const char	msg_StartComment	IRQ6 input request message	main
const char	msg_IRQ6_Detected	IRQ6 input detection message	int_fcu_flash_write
const char	msg_Motorola_format	File transmit request message in the Motorola S-record format	int_fcu_flash_write
const char	msg_MMC_Attach	MMC card insertion request message	int_fcu_flash_write

6.6 Functions

Table 11 lists the functions used in this application.

Table 11 Functions

Function Name	Outline
main	Main processing
io_init_irq6	IRQ6 initialization processing
check_sum_check	Checksum verification in the reprogramming area
hex2bin	Text binary translation processing
INT_IRQ6	IRQ6 interrupt handling
INT_CMT_CMI0	CMT (channel 0) compare match interrupt handling
analyze_read_data	Data analysis processing
int_cmt_led_control	Fixed-cycle LED blinking port controlling
int_fcu_flash_write	Flash memory reprogramming event processing
smpl_fopen	MMC file information obtaining
search_file	Directory entry detection
swapl	4-byte swapping
swapw	2-byte swapping
io_output_msg	Message output processing
flash_write_sample (1)	CMT Initialization for a 100ms fixed-cycle timer
flash_write_spare	An identical program as flash_write_sample (1) (stored in the extra area)
mmc_init_driver	MMC initialization in MMC driver
mmc_attach	Attach confirmation in MMC driver
mmc_read_data	Data read processing in MMC driver
R_FlashErase	Erasing in simple flash API
R_FlashWrite	Data writing in simple flash API

Note: (1) The sample program to be updated at the initial start-up. This program is stored in the reprogramming area.

6.7 Function Specification

This section describes the specification of functions used in the sample code.

main

Outline	Main processing
Header	None
Declaration	void main(void)
Description	After initialization processing, transmits the IRQ6 input request message to the host PC for a checksum test. According to the checksum test result, executes the program placed in the reprogramming area or in the extra storage area.
Argument	None
Returned Value	None
Remarks	

io_init_irq6

Outline	IRQ6 initialization processing
Header	None
Declaration	void io_init_irq6(void)
Description	Initializes the IRQ6. After setting the pin PA20 function to the IRQ6 input, set to detect the interrupt request on trailing edge of IRQ6 input by the interrupt controller. Then, sets the interrupt priority level for the IRQ6.
Argument	None
Returned Value	None
Remarks	

check_sum_check

Outline	Checksum verification in the reprogramming area
Header	None
Declaration	int check_sum_check(void)
Description	Calculates sum values from the start address (H'0000 8000) in the reprogramming area based on the program code size and checksum data stored in the end 256byte area (H'0000 9F00 to H'0000 9FFF) in the reprogramming area to test the match.
Argument	None
Returned Value	• 0: checksum matched • 1: checksum mismatched
Remarks	In this application, a dummy checksum data (chksm_Mem) is stored in advance in the last 256byte area in the reprogramming area so to make a match in the checksum test at the initial start-up.

hex2bin

Outline	Text binary translation processing
Header	None
Declaration	int hex2bin(unsigned char upper, unsigned char lower)
Description	Converts the text data in two characters to 1byte of binary data. When the text data given to the argument is 0 to 9 or A to F, they are regarded as valid data, and are converted to the binary notation as H'0 to H'F. 4-bit left shifting the converted result of the first argument (upper), and getting the logical sum with the converted result of the second argument (lower), returns the result as the 1byte of binary data.
Argument	• First argument : upper — A text data for the upper four bits • Second argument: lower — A text data for the lower four bits
Returned Value	• 0 to 255: 1byte of binary data • -1: improper input data
Remarks	

INT_IRQ6

Outline	IRQ6 interrupt handling
Header	None
Declaration	void INT_IRQ6(void)
Description	Executes a flash reprogramming event processing (int_fcu_flash_write function).
Argument	None
Returned Value	None
Remarks	

INT_CMT_CMI0

Outline	CMT (channel 0) compare match interrupt handling
Header	None
Declaration	void INT_CMT_CMI0(void)
Description	Executes a fixed-cycle LED blinking port controlling (int_cmt_led_control function)
Argument	None
Returned Value	None
Remarks	Used for LED blinking in the sample code.

analyze_read_data

Outline	Data analysis processing
Header	None
Declaration	void analyze_read_data(unsigned char ch)
Description	Stores the argument data in the analysis data storage buffer (a analysis buffer), and extracts the necessary data for updating. When receiving the linefeed code ("\r" or "\n"), the data that have been stored in the analysis buffer is recognized to have filled one-row. Furthermore, if the record data is in the Motorola S-record format, extracts the write data from the record. When extracting the write data, converts the write data in the text format by the two-character unit into 1byte of binary data, and stores it in the write data storage buffer (a write buffer in the dual structure). In multiplying 256bytes, the write buffer for storing is switched alternately. For checksum verification after a flash memory writing, adds a binary translated write data, and counts the number of the total write data. Receiving the end record in the Motorola S-record format, if the stored data count is below 256bytes in the write buffer, adds H'FF until the data count reaches 256byte size.
Argument	unsigned char ch Data in characters for analysis
Returned Value	None
Remarks	

int_cmt_led_control

Outline	Fixed-cycle LED blinking port controlling
Header	None
Declaration	void int_cmt_led_control(void)
Description	Clears the CMT (channel 0) compare match interrupt source flag to change the port output pattern for the fixed-cycle LED blinking at ports PE9 and PE11 to PE15.
Argument	None
Returned Value	None
Remarks	

int_fcu_flash_write

Outline	Flash memory reprogramming event processing
Header	None
Declaration	void int_fcu_flash_write(void)
Description	Erases the reprogramming area (EB04) after serial-transmitting the messages to the host PC. Once erased the reprogramming area, the program becomes standby state waiting for the MMC insertion. When detecting the MMC insertion, reads the program file for updating to analyzes data. When the data analysis processing shows that the write data storage buffer is full and enabled to write, the buffer data is written in the on-chip flash memory. After detecting the end record by the data analysis processing, and finishing writing the reception data in the on-chip flash memory, writes the data (program code size and check sum data) for checksum verification which was calculated when analyzing the data in the end 256byte area (H'0000 7F00 to H'0000 7FFF) in the reprogramming area (EB04). Then, sets a watchdog timer, and waits for a reset by a timer counter overflow.
Argument	None
Returned Value	None
Remarks	

smpl_fopen

Outline	Open the update program	
Header	None	
Declaration	int smpl_fopen(const char *fname, const char *mode)	
Description	Searches for the FAT file system information in the MMC to detect the area that the updated program is stored. The detected information is stored in the structure variable, mmc_fsys.	
Argument	const char *fname	File name, supports only short file names
	const char *mode	Specifies file open (invalid)
Returned Value	-1: Error	
	0: Normal end	
Remarks		

search_file

Outline	Search for a directory entry	
Header	None	
Declaration	int search_file(unsigned char *buff, unsigned char *fname)	
Description	Searches for the update program in the directory entry.	
Argument	unsigned char *buff	Buffer that a directory entry is received
	unsigned char *fname	File name to search for
Returned Value	-1: Error	
	0 or larger: Entry number	
Remarks		

swapl, swapw

Outline	Endian conversion (into long word/ into word)	
Header	None	
Declaration	unsigned long swapl(unsigned char *addr) unsigned short swapw(unsigned char *addr)	
Description	Reads a little-endian data as a big-endian data	
Argument	unsigned char *addr	First address before conversion
Returned Value	Converted data	
Remarks		

io_output_msg

Outline	Message output	
Header	None	
Declaration	void io_output_msg(char *msg_string)	
Description	Outputs a message to the host PC	
Argument	char *msg_string	Message to output
Returned Value	None	
Remarks		

flash_write_sample, flash_write_spare

Outline	CMT initialization for 100-ms fixed-cycle timer
Header	None
Declaration	void flash_write_sample(void), void flash_write_spare(void)
Description	Initializes the CMT (channel 0) for the 100ms fixed-cycle timer. Sets the pin function at ports PE9 and PE11 to PE 15 to output for the fixed-cycle LED blinking and enables a compare match interrupt.
Argument	None
Returned Value	None
Remarks	

For the details on mmc_init_driver, mmc_attach, mmc_read_data , refer to the Application Note "SH7216 Group Accessing MultiMediaCard Using the Renesas Serial Peripheral Interface (document No.: R01AN0039EJ)"

For the details on R_FlashErase, R_FlashWrite, see the Application Note "SH Family Simple Flash API for SH2 and SH2A (document No.: R01AN0719ER) ".

6.8 Processing Flowcharts

6.8.1 Main Processing

Figure 8 shows the procedure of main processing.

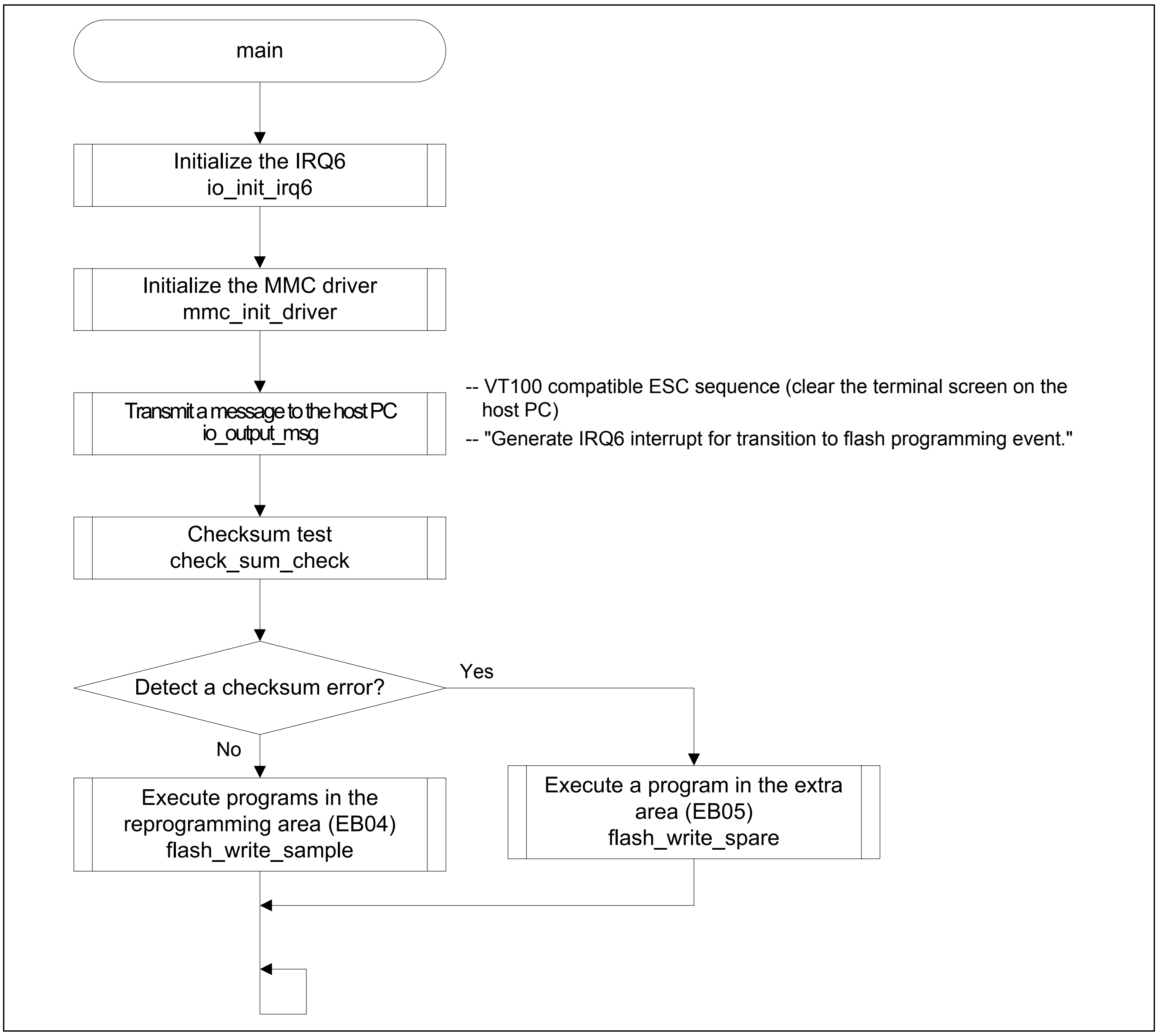

main
Initialize the IRQ6
io_init_irq6
Initialize the MMC driver
mmc_init_driver
Transmit a message to the host PC
io_output_msg
-- VT100 compatible ESC sequence (clear the terminal screen on the host PC)
-- "Generate IRQ6 interrupt for transition to flash programming event."
Checksum test
check_sum_check
Detect a checksum error?
Yes
No
Execute programs in the reprogramming area (EB04)
flash_write_sample
Execute a program in the extra area (EB05)
flash_write_spare

Figure 8 Main Processing

6.8.2 Flash Memory Reprogramming Event Processing

Figure 9 and Figure 11 show the procedures of flash memory reprogramming event processing.

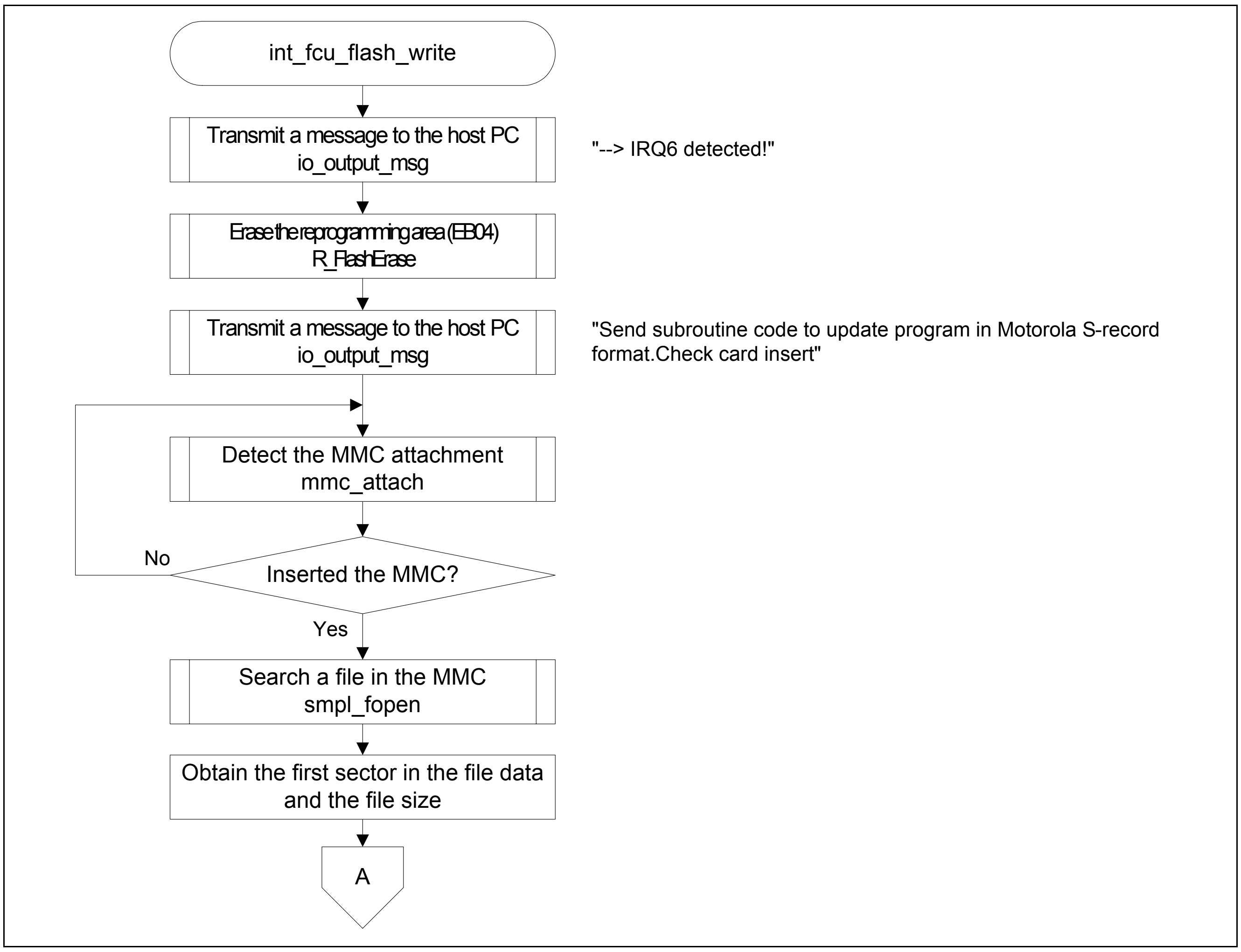

int_fcu_flash_write
Transmit a message to the host PC io_output_msg
"--> IRQ6 detected!"
Erase the reprogramming area (EB04) R_FlashErase
Transmit a message to the host PC io_output_msg
"Send subroutine code to update program in Motorola S-record format.Check card insert"
Detect the MMC attachment mmc_attach
Inserted the MMC?
No
Yes
Search a file in the MMC smpl_fopen
Obtain the first sector in the file data and the file size
A

Figure 9 Flash Memory Reprogramming Event Processing (1)

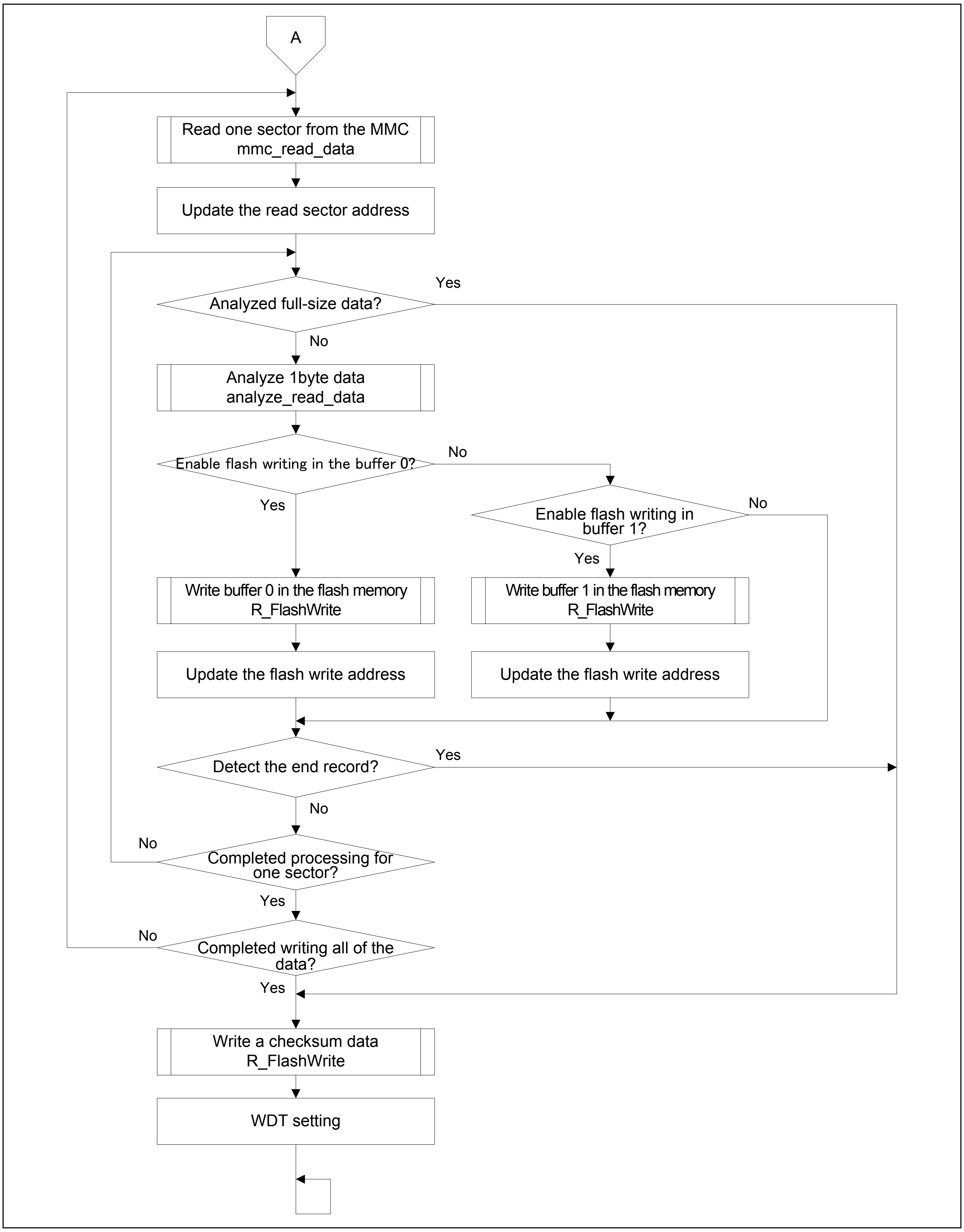

A
Read one sector from the MMC
mmc_read_data
Update the read sector address
Analyzed full-size data?
Yes
No
Analyze 1byte data
analyze_read_data
Enable flash writing in the buffer 0?
No
Yes
Enable flash writing in buffer 1?
No
Yes
Write buffer 0 in the flash memory
R_FlashWrite
Write buffer 1 in the flash memory
R_FlashWrite
Update the flash write address
Update the flash write address
Detect the end record?
Yes
No
Completed processing for one sector?
No
Yes
Completed writing all of the data?
No
Yes
Write a checksum data
R_FlashWrite
WDT setting

Figure 10 Flash Memory Reprogramming Event processing (2)

6.8.3 Data Analysis Processing

Figure 11 shows the procedure of data analysis processing.

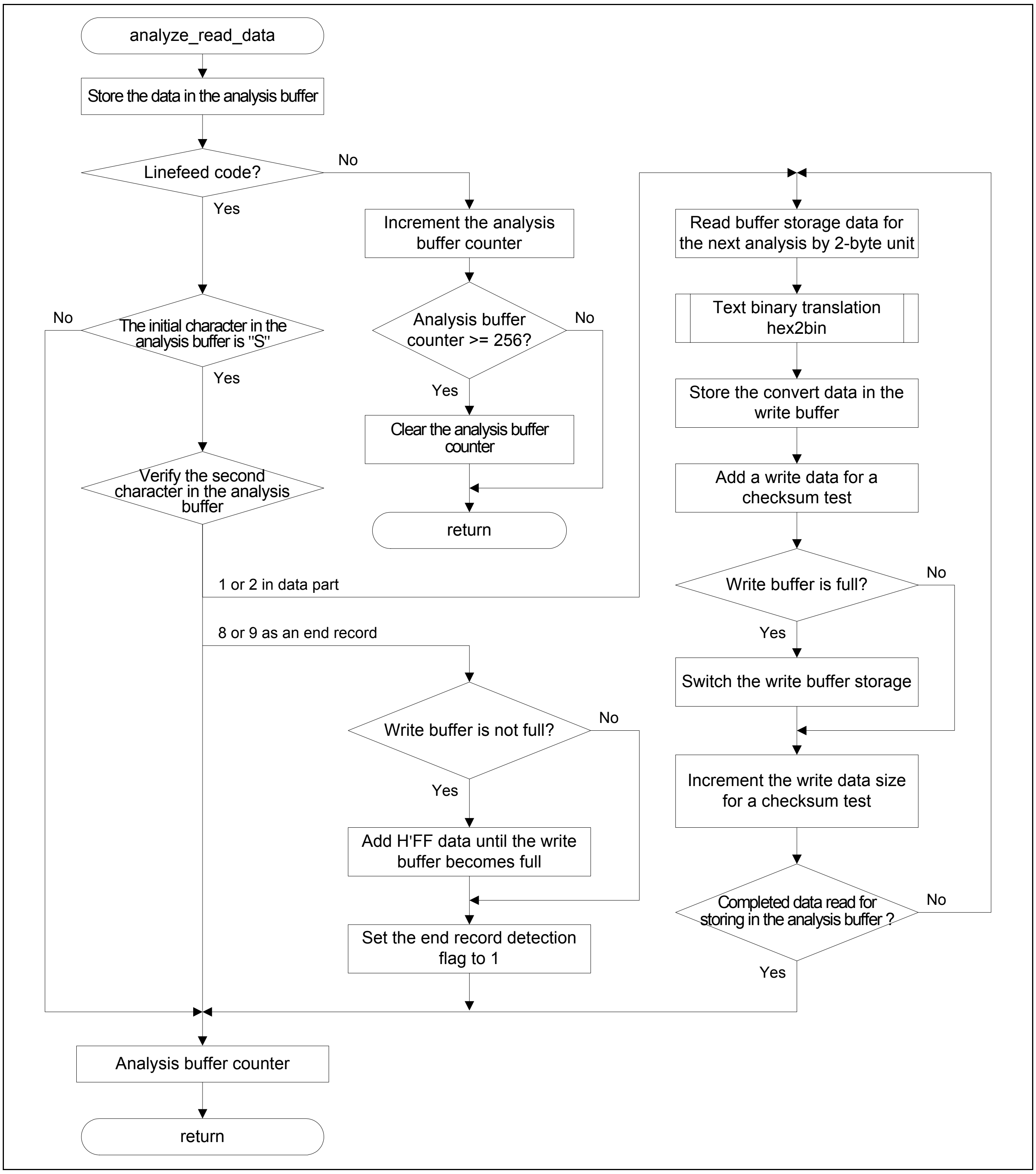
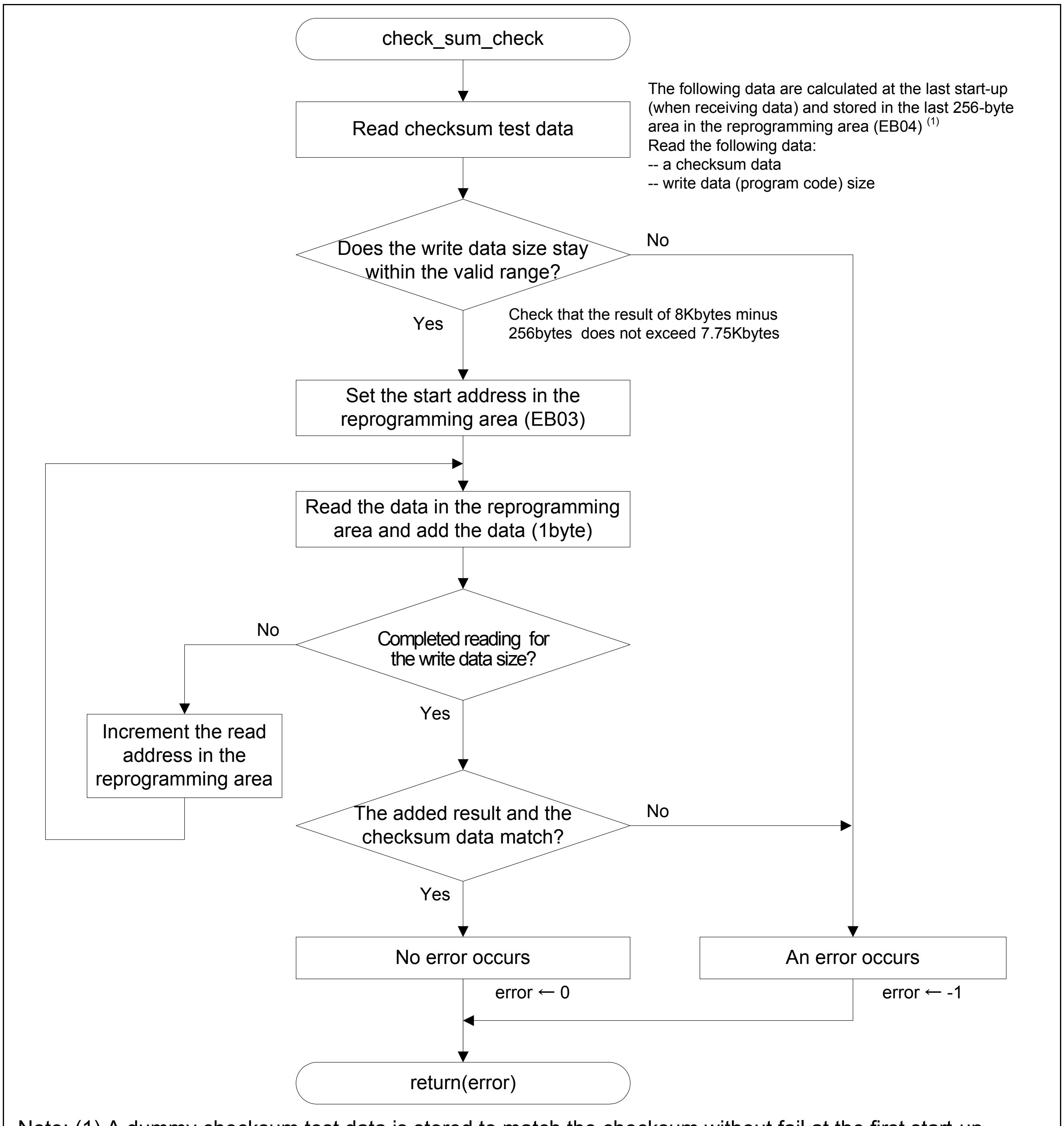

analyze_read_data
Store the data in the analysis buffer
Linefeed code?
No
Yes
Increment the analysis buffer counter
Analysis buffer counter >= 256?
No
Yes
Clear the analysis buffer counter
return
The initial character in the analysis buffer is "S"
No
Yes
Verify the second character in the analysis buffer
1 or 2 in data part
8 or 9 as an end record
Read buffer storage data for the next analysis by 2-byte unit
Text binary translation hex2bin
Store the convert data in the write buffer
Add a write data for a checksum test
Write buffer is full?
No
Yes
Switch the write buffer storage
Increment the write data size for a checksum test
Completed data read for storing in the analysis buffer ?
No
Yes
Write buffer is not full?
No
Yes
Add H'FF data until the write buffer becomes full
Set the end record detection flag to 1
Analysis buffer counter
return

Figure 11 Data Reception Processing

6.8.4 Checksum verification Processing

Figure 12 shows the procedure of a checksum verification.


check_sum_check
Read checksum test data
The following data are calculated at the last start-up (when receiving data) and stored in the last 256-byte area in the reprogramming area (EB04) (1)
Read the following data:
-- a checksum data
-- write data (program code) size
Does the write data size stay within the valid range?
No
Yes
Check that the result of 8Kbytes minus 256bytes does not exceed 7.75Kbytes
Set the start address in the reprogramming area (EB03)
Read the data in the reprogramming area and add the data (1byte)
No
Completed reading for the write data size?
Yes
Increment the read address in the reprogramming area
The added result and the checksum data match?
No
Yes
No error occurs
An error occurs
error ← 0
error ← -1
return(error)

Note: (1) A dummy checksum test data is stored to match the checksum without fail at the first start-up.

Figure 12 Checksum Verification Processing

6.8.5 Text Binary Translation Processing

Figure 13 shows the procedure of text binary conversion.

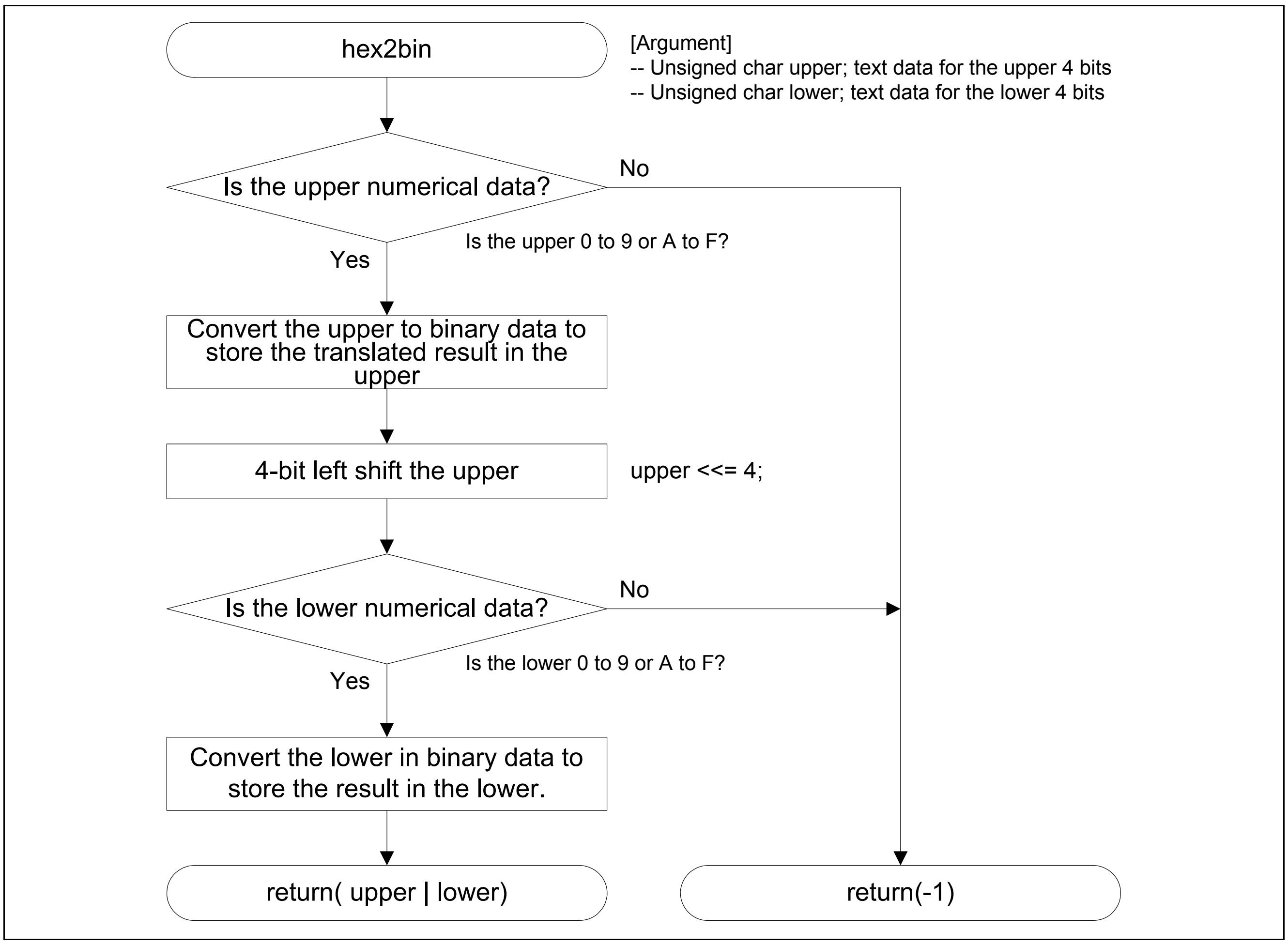
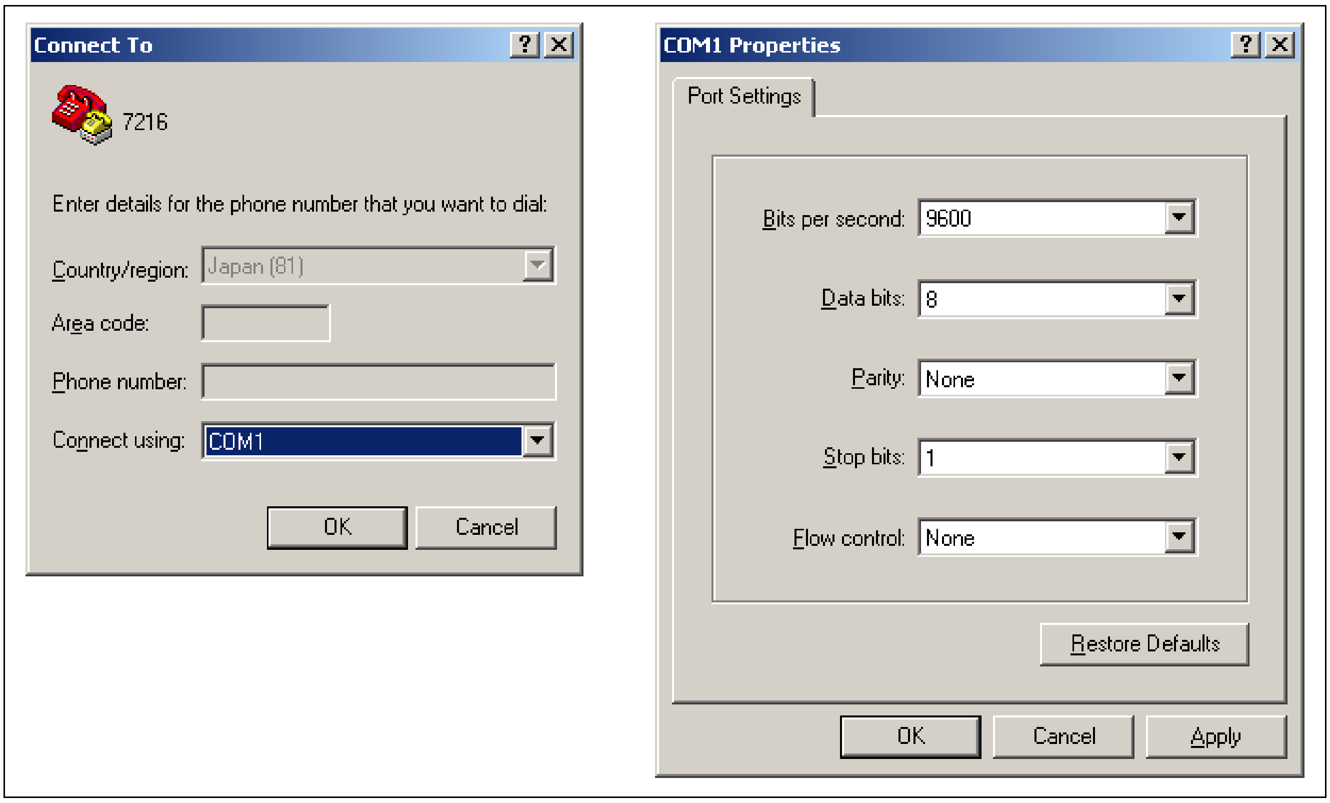

hex2bin
[Argument]
-- Unsigned char upper; text data for the upper 4 bits
-- Unsigned char lower; text data for the lower 4 bits
Is the upper numerical data?
No
Is the upper 0 to 9 or A to F?
Yes
Convert the upper to binary data to store the translated result in the upper
4-bit left shift the upper
upper <<= 4;
Is the lower numerical data?
No
Is the lower 0 to 9 or A to F?
Yes
Convert the lower in binary data to store the result in the lower.
return(upper | lower)
return(-1)

Figure 13 Text Binary Translation Processing

7. Operation Overview

This application uses the VT100 compatible terminal software on the host PC. This chapter describes the processing example using the hyper terminal which is normally equipped on the Microsoft Windows series.

By starting the hyper terminal, the "Connect To" dialog box appears. Select the icon by typing the name of the connection. The dialog box shown on he left side in Figure 14 appears. Select the serial port number in the box of "Connect using", and the property setting dialog box as shown on the right side in Figure 14 appears. Specify the setting values in he boxes referring to the list in the Table 12.


Connect To

7216

Enter details for the phone number that you want to dial:

Country/region: Japan (81)

Area code:

Phone number:

Connect using: COM1

COM1 Properties

Port Settings

Bits per second: 9600

Data bits: 8

Parity: None

Stop bits: 1

Flow control: None

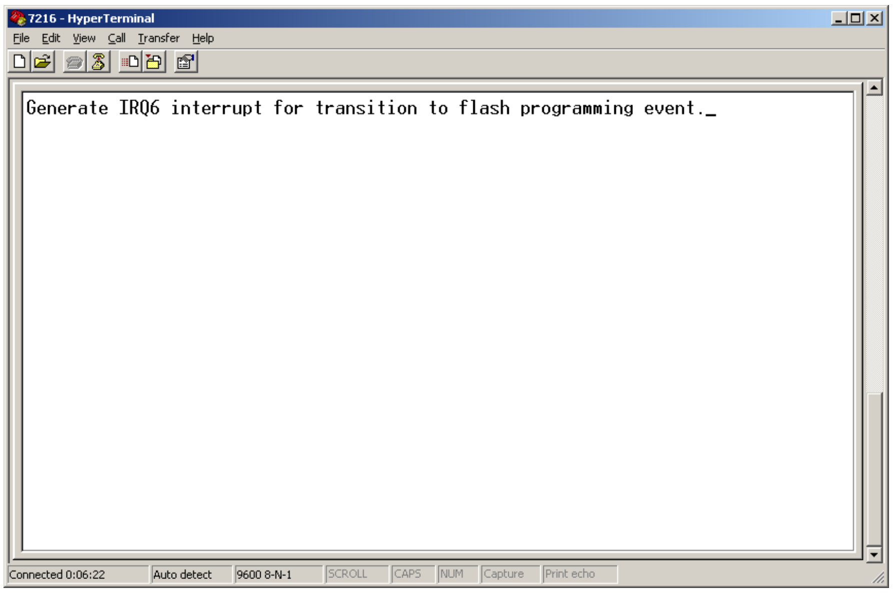
Figure 14 Hyper Terminal Connection Setting and the Port Setting

Table 12 Port Setting

Port	Setting Value
Bits per second	9600
Data bits	8
Parity	None
Stop bits	1
Flow control	None

After completing the upper setting, connect the host PC to the SH7216 board by the serial cable, RS-232C.

After setting along with the instruction in the previous page, start the sample code on the SH7216 board, and the SH7216 transmits the ESC sequence. At this time, the terminal software display is cleared on the host PC. Next, the SH7216 transmits the message, "Generate IRQ6 interrupt for transition to flash programming event." to the host PC as shown in the Figure 15.

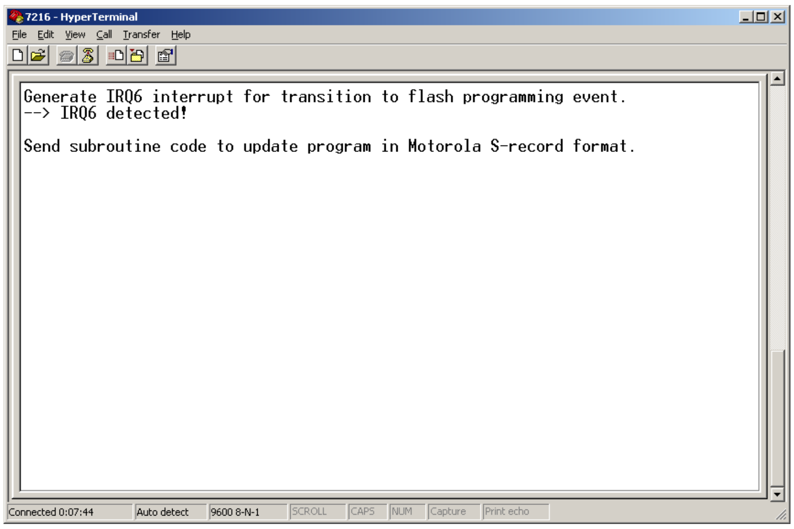


```
Generate IRQ6 interrupt for transition to flash programming event._
```

Figure 15   Hyper Terminal Display at Start-up Sample Code

Then, the SH7216 executes the program stored in the updating area (EB04), which blinks the LED on the board in a pattern in the fixed 100ms cycle.

Keeping this state and pushing the IRQ6 switch on the SH7216 board, the message, "--> IRQ6 detected!" is transmitted to the host PC. Once the IRQ6 interrupt is generated, the SH7216 board starts the flash memory reprogramming event processing to erase the updating area (ES04). When completing erasing the updating area, the SH7216 transmits the message, "Send subroutine code to update program in Motorola S-record format. Check card insert " to the host PC and becomes standby state as shown in the Figure 16 to wait for the MMC is inserted.



```
Generate IRQ6 interrupt for transition to flash programming event.
--> IRQ6 detected!

Send subroutine code to update program in Motorola S-record format.
```

Figure 16   Terminal Soft Screen when Waiting for a Data

When the MMC is inserted to the card slot, the SH7216 board extracts the valid data (the program code) to write in the updating area (EB04). Once the data is received, and program code is written and updated, the SH7216 board writes the data for a checksum verification in the on-chip flash memory. Then, the SH7216 board sets a dogwatch timer and it becomes standby state to wait for a reset by the timer counter overflow.

When the watchdog timer restarts the SH7216 board, it executes the updated program. At this time, the LED on the board gives a different pattern of blinking from the previous pattern.

In case of a failure in the data reception or flash memory reprogramming (updating) before restarting the program, the SH7216 board determines it as a checksum error when restarting by reset input. In this case, the SH7216 executes the program in the extra area (EB05).

8. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

9. Reference Documents

Hardware Manual
SH7214 Group, SH7216 Group User's Manual: Hardware Rev.3.00
The latest version can be downloaded from the Renesas Electronics website.

Development Tool Manual
SuperH C/C++ Compiler Package V.9.04 User's Manual Rev.1.01
The latest version can be downloaded from the Renesas Electronics website.

Website and Support

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

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

Revision History	SH7216 Group Application Note Updating Program Code by Reprogramming Flash Memory in User Program Mode Using MMC

Rev.	Date	Description	
		Page	Summary
1.00	Sep. 29, 2011	—	First edition issued
1.01	Jun. 15, 2012	—	Sample code (simple flash API) revised

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.
   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products
   Before changing from one product to another, i.e. to one with a different type number, confirm that the change will not lead to problems.
   — The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

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

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

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

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

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

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

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

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

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

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

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

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