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.

RENESAS

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.

To all our customers

Regarding the change of names mentioned in the document, such as Mitsubishi Electric and Mitsubishi XX, to Renesas Technology Corp.

The semiconductor operations of Hitachi and Mitsubishi Electric were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Mitsubishi Electric, Mitsubishi Electric Corporation, Mitsubishi Semiconductors, and other Mitsubishi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Note : Mitsubishi Electric will continue the business operations of high frequency & optical devices and power devices.

Renesas Technology Corp.
Customer Support Dept.
April 1, 2003

M16C/80 Group

Explanation of boot loader

1.0 Abstract

This application note describes the communication protocol specifications of the boot loader and the rewrite program prepared by the user.

2.0 Introduction

The explanation of this issue is applied to the following condition:

Applicable MCU: M16C/80 group

3.0 Contents

3.1 Overview of Bootloader

External ROM version of M16C/80 group with built-in bootloader (hereinafter referred to as "M16C/80 bootloader) contains firmware (hereinafter referred to as "bootloader") which downloads a boot program (hereinafter referred to as "rewrite program") to the microcontroller for reprogramming of the external Flash memory. Table 3.1.1 shows the product list of M16C/80 bootloader.

With a serial writer or personal computer, a rewrite program is downloaded to the internal RAM on the microcontroller for execution via serial communications with M16C/80 bootloader. Besides the said down-load function, the bootloader has another optional function, flash memory control function. This is to repro-gram a certain type of external flash memories (*1) .

The download and flash memory control functions are described in 3.2 Overview of bootloader mode 1 (clock synchronized) and 3.3 Overview of boot loader mode 2 (clock asynchronized)

*1: Mitsubishi flash memory M5M29GB/T160BVP, M5M29GB/T320BVP and MCMs combined with the these flash memories only.

3.1.1 Bootloader Mode

When a reset is released by applying an "H" level to CNVss pin, M16C/80 bootloader starts the operation in the microprocessor mode. On the other hand, when a reset is released by applying an "L" level to CNVss pin, M16C/80 bootloader starts the operation in the bootloader program and this mode is called "bootloader mode."

Table 3.1.1 Product List

As of Oct., 2001

Type No	ROM capacity	RAM capacity	Package type	Remarks
M30800SFP-BL	———	10 Kbytes	100P6S-A	External ROM version with built-in bootloader
M30800SGP-BL			100P6Q-A	
M30802SGP-BL			144P6Q-A	
M30803SFP-BL	———	24 Kbytes	100P6S-A	
M30803SGP-BL			100P6Q-A	
M30805SGP-BL			144P6Q-A	

3.1.2 Overview of Bootloader Mode

There are two bootloader modes: Bootloader mode 1, which is clock synchronized, and Bootloader mode 2, which is asynchronized. Communications with external devices are performed using a serial programmer (*1) .

These bootloader modes start when a reset is released by applying an "L" level to CNVss. Inputs/outputs of serial data are transferred in 8-bit units with UART1. The bootloader switches between mode 1 (clock synchronized) and mode 2 (clock asynchronized) according to the level of the SCLK pin when the reset is released.

To use bootloader mode 1 (clock synchronized), apply an "H" level to the SCLK pin and release the reset.
The operation uses four UART1 pins CLK1, RxD1, TxD1 and RTS1. The CLK1 pin becomes the transfer clock input pin SCLK and inputs an external transfer clock. The TxD1 pin becomes TxD. This pin is for CMOS output. The RTS1 pin becomes BUSY output and outputs an "L" level when ready for reception and an "H" when reception starts.

To use bootloader mode 2 (clock asynchronized), apply an "L" level to the SCLK pin and release the reset. The operation uses two UART1 pins RxD1 and TxD1 as RxD and TxD.

The bootloader switches whether to enable or disable the built-in pull-up function according to the level of the BUSY pin when the reset is released. Immediately after being reset, if an "L" level is applied to the BUSY pin, then the pull-up function become disable, and if an "H" then enable. Table 3.1.2 shows pin functions, and Figure 3.1.1 to 3.1.3 show pin connections for bootloader mode.

*1: Bootloader mode 1 (clock synchronized) can be used with PC card type flash memory programmer (M3A-0655G01/02) and Sunny Giken serial writer Multi Flash Write. Bootloader mode 2 (clock asynchronized) can be used with M16C Flash Starter (M3A-0806).

Note: Users are usually required to develop a serial writer together with a rewrite program.

Table 3.1.2 Pin functions

Pin	Name	I/O	Description
V_{CC}, V_{SS}	Power input		Apply 4.2V to 5.5V(*1) to Vcc pin and 0V to Vss pin
CNV_{SS}	CNV_{SS}	I	Connect to Vss pin
$\overline{RESET}$	RESET input	I	RESET input pin. While reset is "L" level, a 20 cycle or longer clock must be input to X_{IN} pin.
X_{IN}	Clock input	I	Connect a ceramic resonator or crystal oscillator between X_{IN} and X_{OUT} pins. To input an externally generated clock, input it to X_{IN} pin and open X_{OUT} pin.
X_{OUT}	Clock output	O	
BYTE	BYTE input	I	Connect this pin to Vcc or Vss.
AV_{CC}, AV_{SS}	Analog power supply input	I	Connect AV_{SS} to V_{SS} and AV_{CC} to V_{CC}, respectively.
V_{REF}	Reference voltage input	I	Enter the reference voltage for A-D converter from this pin.
$P0_0$ to $P0_7$	Input port P0	I/O	Connect to memory or input "H" or "L" level signal or open.
$P1_0$ to $P1_7$	Input port P1	I/O	Connect to memory or input "H" or "L" level signal or open.
$P2_0$ to $P2_7$	Input port P2	I/O	Connect to memory or input "H" or "L" level signal or open.
$P3_0$ to $P3_7$	Input port P3	I/O	Connect to memory or input "H" or "L" level signal or open.
$P4_0$ to $P4_7$	Input port P4	I/O	Connect to memory or input "H" or "L" level signal or open.
$P5_0$ to $P5_2$	Input port P5	I/O	Connect to memory or input "H" or "L" level signal or open.
$P5_3$ to $P5_4$	Input port P5	I	Input "H" or "L" level signal or open.
$P5_5$	HOLD input	I	Input "H" level signal.
$P5_6$	Input port P5	I	Input "H" or "L" level signal or open.
$P5_7$	RDY input	I	Input "H" level signal.
$P6_0$ to $P6_3$	Input port P6	I	Input "H" or "L" level signal or open.
$P6_4/RTS_1$	BUSY output (*2)	O	Boot loader mode 1: BUSY signal output. Boot loader mode 2: Monitors the program operation check.
$P6_5/CLK_1$	SCLK input	I	Boot loader mode 1: Serial clock input. Boot loader mode 2: Input "L" level signal.
$P6_6/RxD_1$	RxD input	I	Serial data input pin.
$P6_7/TxD_1$	TxD output	O	Serial data output pin.
$P7_0$ to $P7_7$	Input port P7	I	Input "H" or "L" level signal or open.
$P8_0$ to $P8_4$ $P8_6$, $P8_7$	Input port P8	I	Input "H" or "L" level signal or open.
$P8_5$	$\overline{NMI}$ input	I	Connect this pin to V_{CC}.
$P9_0$ to $P9_7$	Input port P9	I	Input "H" or "L" level signal or open.
$P10_0$ to $P10_7$	Input port P10	I	Input "H" or "L" level signal or open.
$P11_0$ to $P11_4$	Input port P11	I	Input "H" or "L" level signal or open.
$P12_0$ to $p12_7$	Input port P12	I	Input "H" or "L" level signal or open.
$P13_0$ to $P13_7$	Input port P13	I	Input "H" or "L" level signal or open.
$P14_0$ to $P14_6$	Input port P14	I	Input "H" or "L" level signal or open.
$P15_0$ to $P15_7$	Input port P15	I	Input "H" or "L" level signal or open.

*1: When using at 4.2 V or lower, max. operating frequency is 10MHz.

*2: For further information, please refer to "BUSY Pin Function".

• Shading indicates pins used in bootloader mode.

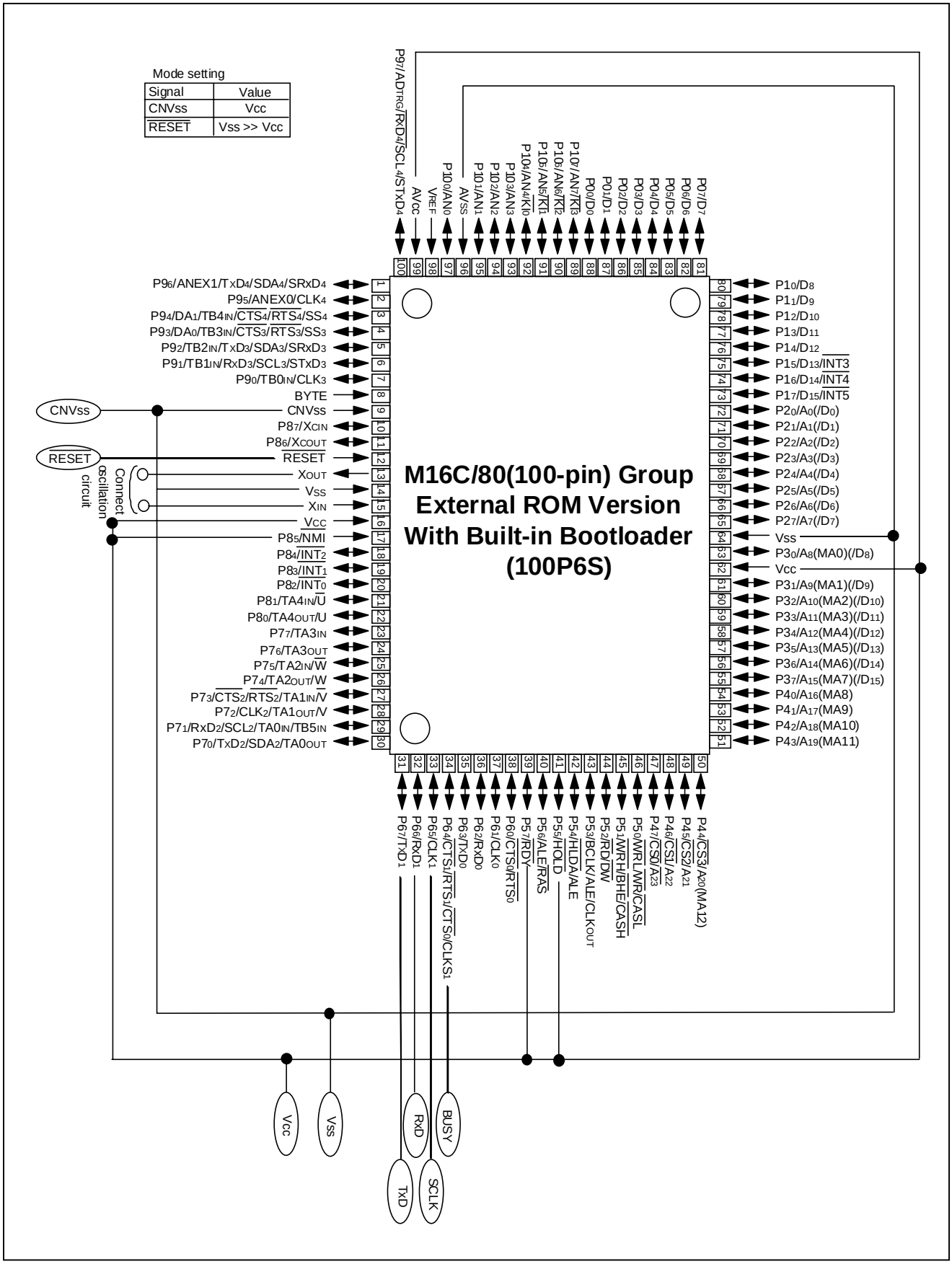


Figure 3.1.1 Pin connections for bootloader mode (100P6S)

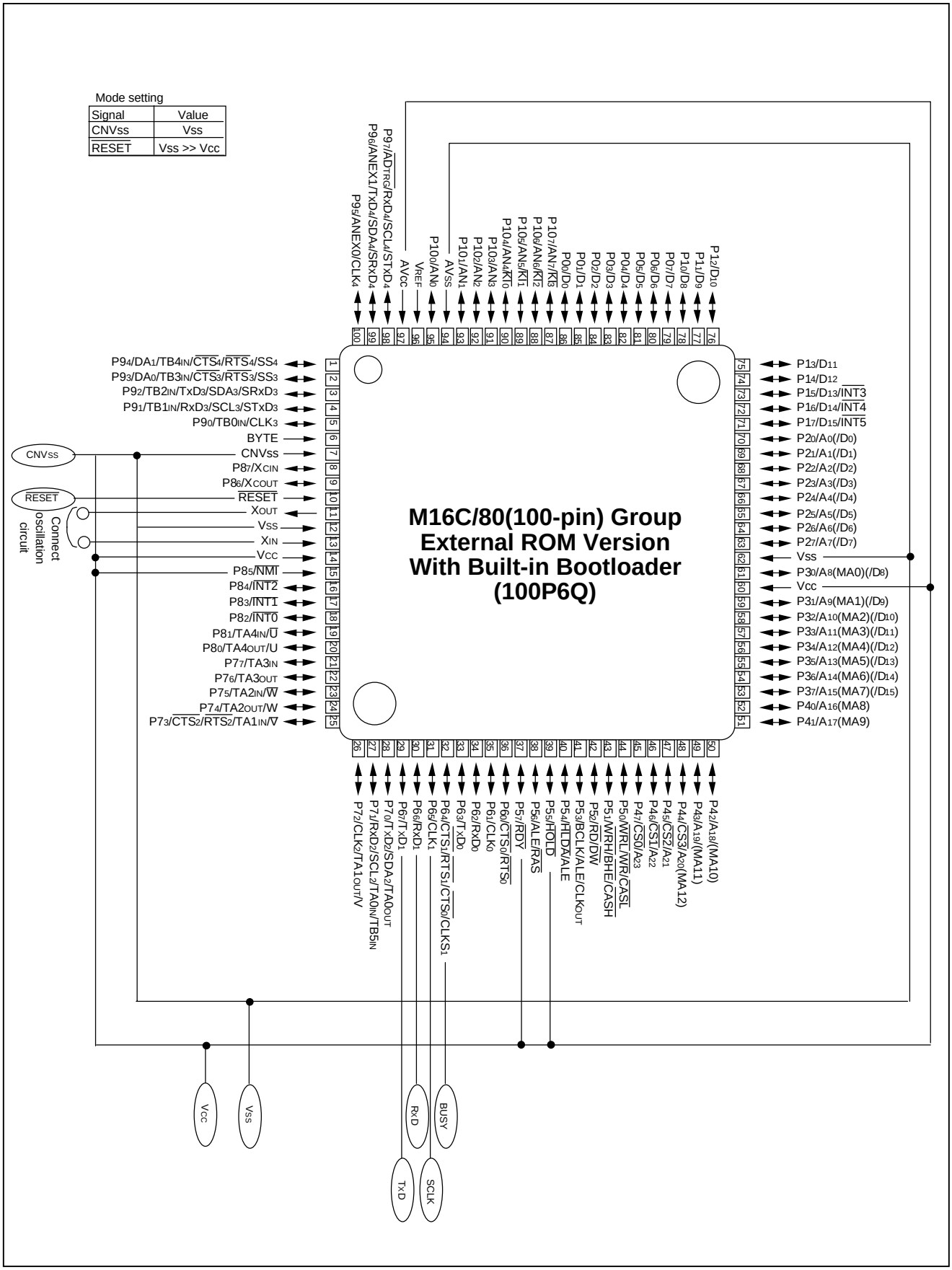


Figure 3.1.2 Pin connections for bootloader mode (100P6Q)

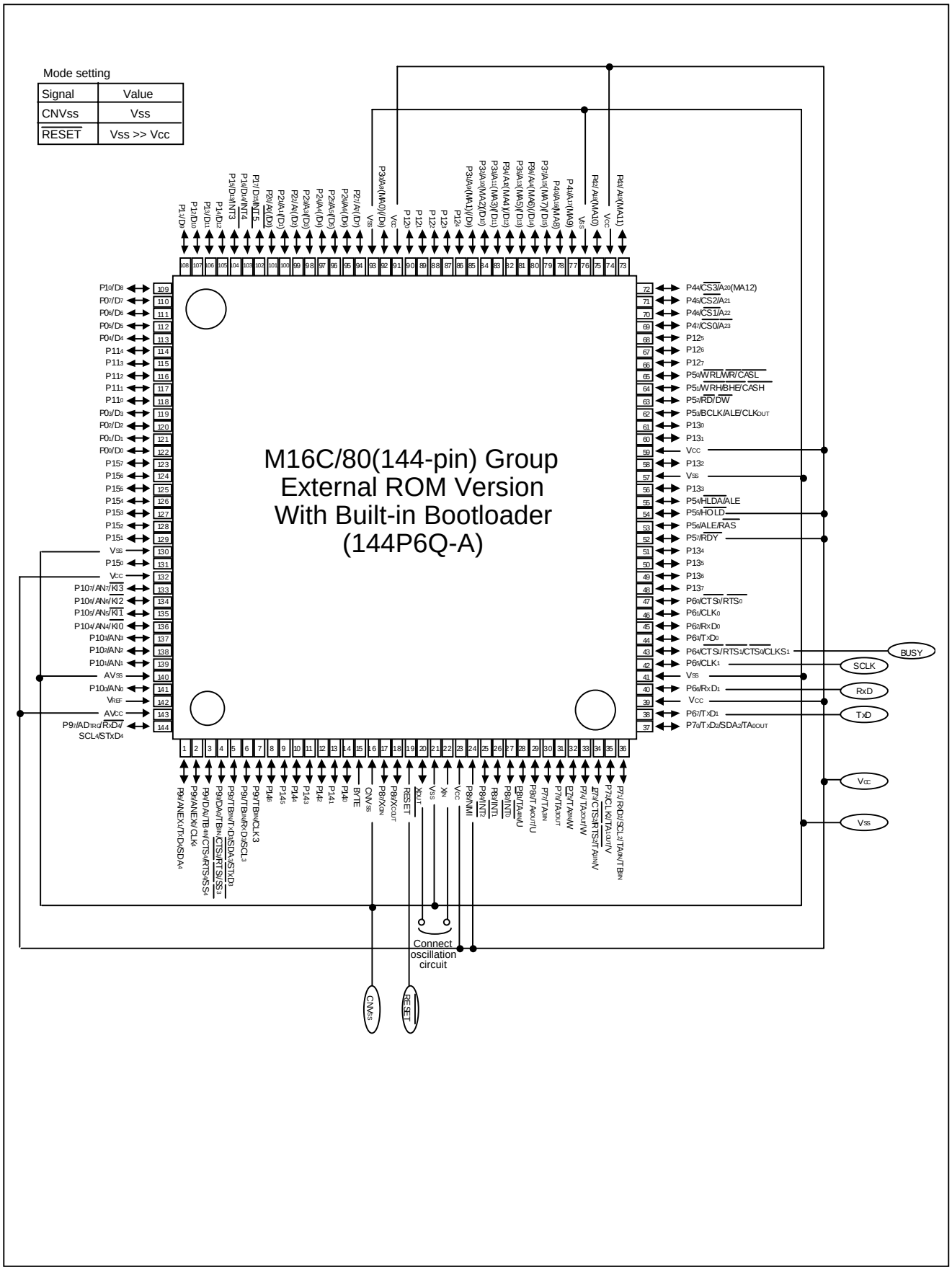


Figure 3.1.3 Pin connections for bootloader mode (144P6Q)

3.1.3 BUSY Pin Description

Immediately after being reset, the BUSY(P6_4/RTS1) pin functions as an input. And the bootloader selects whether to enable or disable the built-in pull-up function according to the level of BUSY pin at this moment. (On user's target board, please use a pull-up or pull-down of the BUSY pin to select enabling or disabling the pull-up function.) Immediately after being reset, if an "L" level is applied to the BUSY pin, the pull-up function is disabled, and if an "H" then enabled. After the selection, the BUSY pin functions as an output.

Table 3.1.3 shows pull-up pins when the internal pull-up function becomes enabled.

Table 3.1.3 Pull-up pins and settings for internal pull-up function

Pull-up pin	Setting of pull-up control register
P0 to P3 (Note)	PUR0 = $0FF_{16}$
P4, P5 (Note)	PUR1 = $0F_{16}$
P6 to P9 (P8_5 excluded)	PUR2 = $0FF_{16}$
P10 to P13	PUR3 = $0FF_{16}$
P14, 15	PUR4 = $0F_{16}$

Note: Before changing to microprocessor mode, please set the value of the P0 to P5 pull-up control registers, which function as bus, to "0" for disabling internal pull-up function.

3.2 Overview of bootloader mode 1 (clock synchronized)

In bootloader mode 1, software commands, addresses and data are input and output between the MCU and serial programmer (*1) using 4-wire clock-synchronized serial I/O (UART1). Bootloader mode 1 is engaged by releasing the reset with the SCLK pin "H" level.

In reception, software commands, addresses and program data are synchronized with the rise of the transfer clock that is input to the SCLK pin, and are then input to the MCU via the RxD pin. In transmission, the read data and status are synchronized with the fall of the transfer clock, and output from the TxD pin.

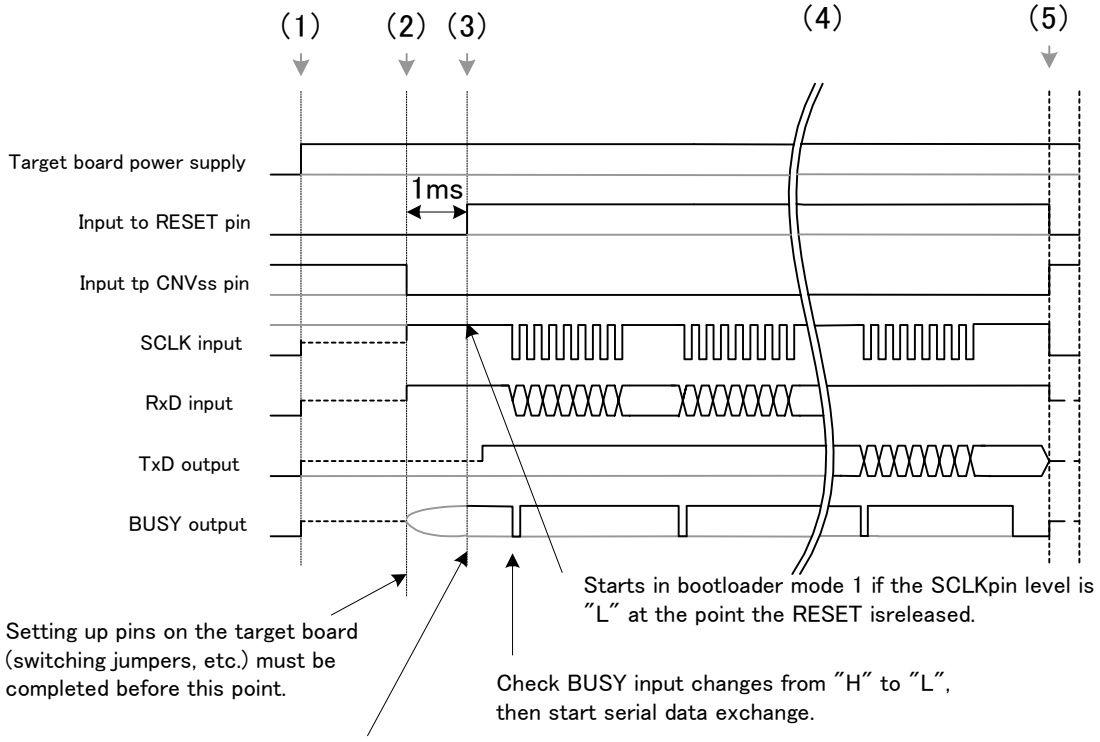
The TxD pin is for CMOS output. Transfer is in 8-bit units with LSB first. When busy, such as during transmission, reception, software command execution, the BUSY pin is "H" level. Accordingly, always start the next transfer after the BUSY pin is "L" level.(refer to figure 3.2.1 I/O Timing)

Bootloader mode 1 supports the download and the flash memory control functions. Here following are explained these function.

*1: MAEC PC card type flash memory programmer and Sunny Giken Multi Flash Writer can be used for the serial programmer

I/O Timing from Microcomputer Side

(1) Turn ON power to target board.
(2) Start flash memory write control.
(3) Enter serial writing mode by cancelling reset.
(4) Delete, write, read data from flash memory.
(5) End flash memory write control.



Immediately after being reset, the BUSY pin functions as an input.
And Bootloader sets the pull-up control registers.
After that, the BUSY pin functions as an output.
(When the BUSY pin is pulled up on the target board,
the internal pull-up resistances are valid.
When the BUSY pin is pulled down on the target board, the internal pull-up resistances are invalid.)

Figure 3.2.1 I/O Timing

3.2.1 Download Function

Functional Description

The download function of M16C/80 Bootloader is to download a rewrite program (*1) to the internal RAM in the microcomputer using serial communications and then let the processing jump to the ad-dress in the RAM where the downloaded program has been located.

*1: The rewrite program should be prepared by the user according to the following notes.

- The rewrite program should have two functions: (1) control function to write, erase and read to/from the external flash and (2) communication function to communicate with a serial writer.
- When using a stack in the rewrite program, please setup the stack pointer within the program.
- When the download is completed, the microcomputer starts the operation in single chip mode. Please change the processor mode from the single chip mode to microprocessor mode using the rewrite program before starting controls such as writing or erasing to the external flash memory.
- Please do not use any interrupts in the rewrite program.
- Please refer to the memory map of the Appendix 1 for the details of download area.

Software commands

Table 3.2.1 lists the software commands for bootloader mode 1.

Table 3.2.1 Software commands for download (Boot loader mode 1)

	Control command	1st byte transfer	2nd byte	3rd byte	4th byte	5th byte	6th byte	
1	Download function	FA_{16}	Size (low)	Size (high)	Check-sum	Data input	To required number of times	
2	Download result output	FA_{16}	Data output					
3	Version data output function	FB_{16}	Version data output	Version data output	Version data output	Version data output	Version data output	Version data output to 9th byte

Note 1: Shading indicates transfer from microcomputer to serial programmer. All other data is transferred from the serial programmer to the microcomputer.

Download

This command downloads a rewrite program to the internal RAM. The program as downloaded is stored in the internal RAM from address 600_{16} onward.

After a reset, the downloaded program is held in the internal RAM. Execute the download command as explained here following.

(1) Transfer the "FA_{16}" command code serially with the 1st byte.

(2) Transfer the program size serially with the 2nd and 3rd bytes, as follows: low order size with the 2nd byte, and high-order size with the 3rd byte.

(3) Transfer the check sum serially with the 4th byte. The check sum is added to all data sent with the 5th byte onward.

(4) The program to execute is sent with the 5th byte onward. The size of the program to be transferred will vary depending on the internal RAM size. (Please refer to "3.6 Memory Map" about the size of the rewrite program.)

When all data has been transferred, the microcomputer automatically executes the download result output command.

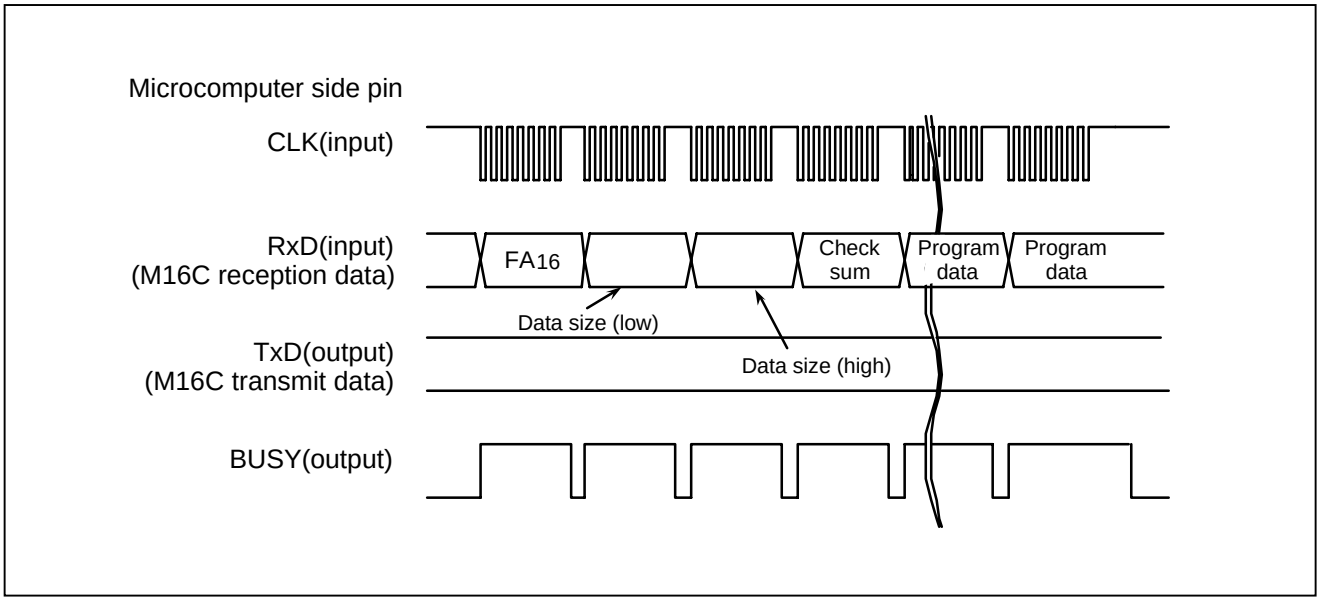


Figure 3.2.2 Timing for download

Download Result Output

After downloaded, the transferred check sum value from the serial programmer and the check sum value obtained by received data are compared. When the check sum values are matched, "FA16" and "0016"(success) are sent back, and then the processing jumps to the beginning of the downloaded program to execute it. When the values are not matched, "FA16" and "0116"(failure) are sent back and boot program stored in the microcomputer is transferred to RAM again, then this program is executed. (Return to the original state)

When the Download Function has been completed, the bootloader (microcomputer) outputs the execution result as explained here following.

(1) When the Download Function has been completed, output the "FA16" command code with the 1st byte.
(2) Output the download result code ("0016" : success / "0116" : failure) with the 2nd byte.

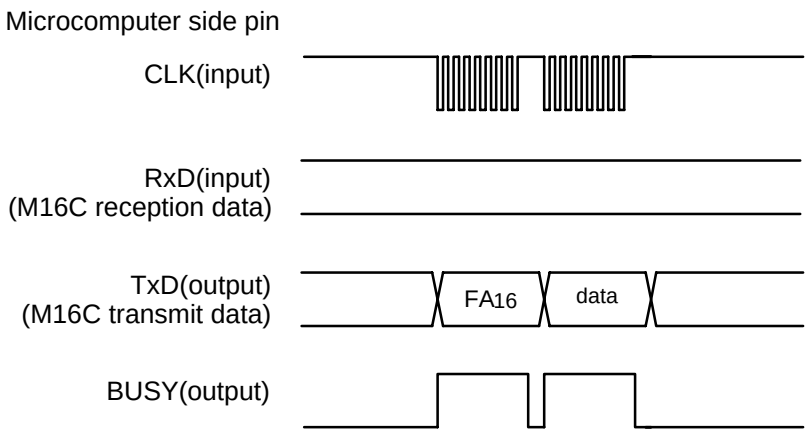


Figure 3.2.3 The timing of download result

Version Information Output Command

This command outputs the version information data of bootloader.

Execute the version information output command as explained here following.

(1) Transfer the "FB_{16}" command code serially with the 1st byte.

(2) The version information will be output serially from the 2nd byte to the 9th byte. This data is composed of 8 ASCII code characters (*1) .

*1: Version data format is 8 characters by ASCII code,
"VER. X. XX" (X:number).
It is output from "V".

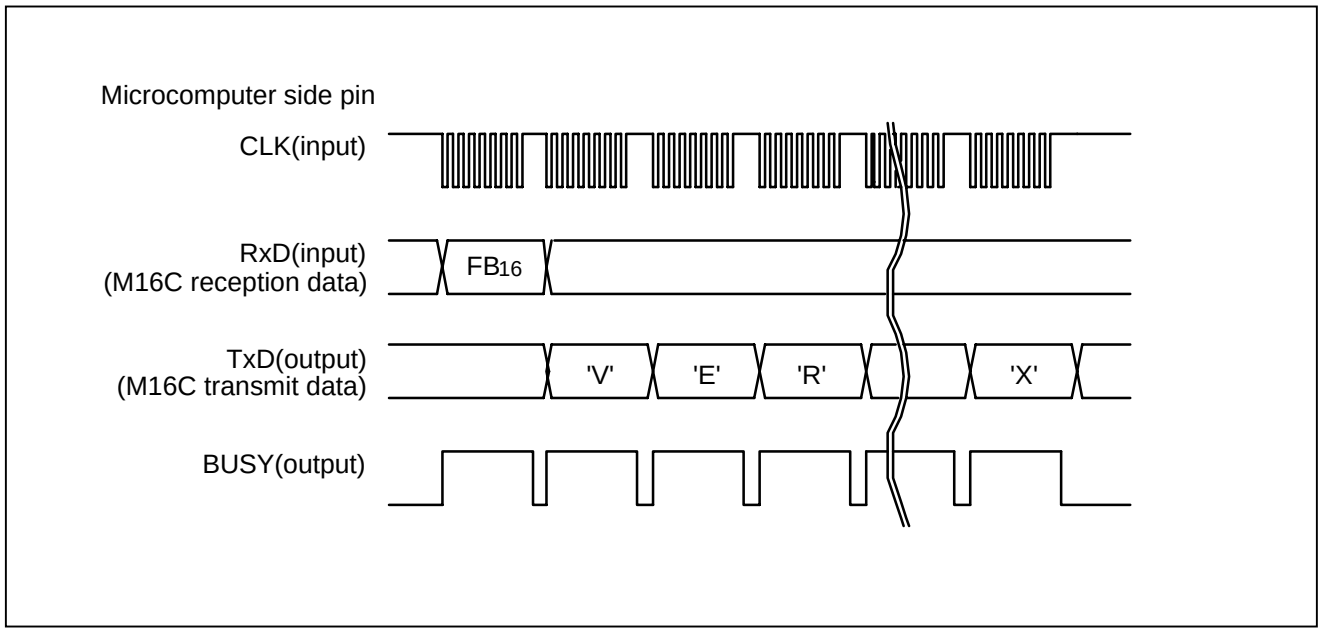


Figure 3.2.4 Timing for version information output

3.2.2 Flash Memory Control Function

Functional Description

If an external flash memory is M5M29GB/T160BVP, M5M29GB/T320BVP (made by MITSUBISHI) or MCM with these flash memories, the M16C/80 Bootloader is able to execute writing and erasing without rewrite program. (A connection example is shown in "3.7 Connection example of bootloader")

The M16C/80 Bootloader writes and erases a program to flash memory by communicating commands and data with serial programmer.

Software Commands

The following table lists the flash memory control commands and I/O data.

When only an external flash memory is M5M29GB/T160BVP, M5M29GB/T320BVP or MCM with these flash memories, the user is able to use these commands.

Table 3.2.2 Software commands for flash memory control (Boot loader mode 1)

	Control command	1st byte transfer	2nd byte	3rd byte	4th byte	5th byte	6th byte	
1	Page read	FF_{16}	Address (middle)	Address (high)	Data output	Data output	Data output	Data output to 259th byte
2	Page program	41_{16}	Address (middle)	Address (high)	Data input	Data input	Data input	Data input to 259th byte
3	Block erase	20_{16}	Address (middle)	Address (high)	$D0_{16}$			
4	Erase all unlocked blocks	$A7_{16}$	$D0_{16}$					
5	Read status register	70_{16}	SRD output	SRD1 output				
6	Clear status register	50_{16}						
7	Read lock bit status	71_{16}	Address (middle)	Address (high)	Lock bit data output			
8	Lock bit program	77_{16}	Address (middle)	Address (high)	$D0_{16}$			
9	Read check data	FD_{16}	Data output (low)	Data output (high)				

Note 1: Shading indicates transfer from microcomputer to serial programmer. All other data is transferred from the serial programmer to the microcomputer.

Note 2: SRD refers to status register data, and SRD1 refers to status register 1 data.

Page Read Command

This command reads the specified page (256 bytes) in the flash memory sequentially one byte at a time. The read area is set with a high address (A16 to A23) and middle address (A8 to A15), targeting the 256 bytes from xxxx00$_{16}$ to xxxxFF$_{16}$. (Refer to Figure 3.2.5)

Execute the page read command as explained here following.

(1) Transfer the "FF$_{16}$" command code serially with the 1st byte.

(2) Transfer addresses A$_8$ to A$_{15}$ and A$_{16}$ to A$_{23}$ with the 2nd and 3rd bytes respectively.

(3) From the 4th byte onward, data (D$_0$–D$_7$) for the page (256 bytes) specified with addresses A$_8$ to A$_{23}$ will be output sequentially from the smallest address first in sync with the fall of the clock.

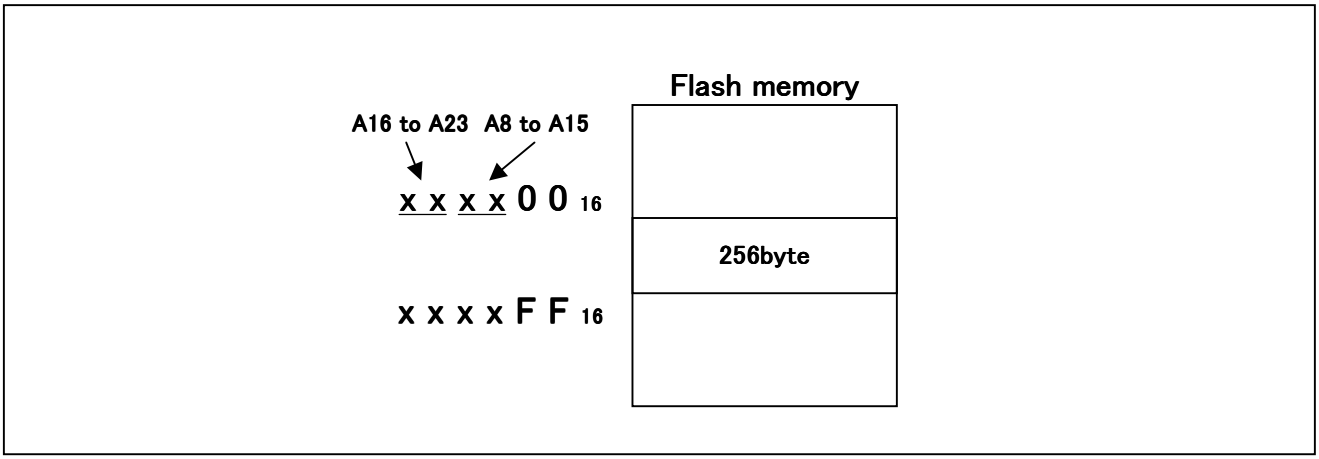


Figure 3.2.5 The designation of the address and command applicable area

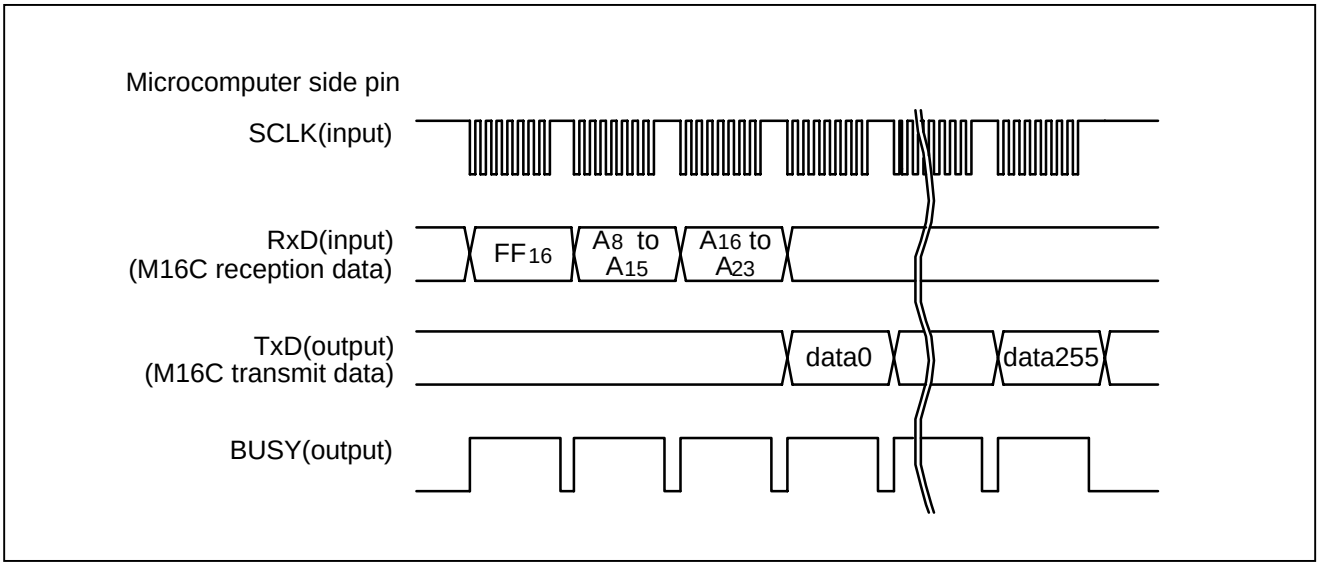


Figure 3.2.6 Timing for page read

Page Program Command

This command writes the specified page (256 bytes) in the flash memory sequentially one byte at a time. The area to be written to is set using a high address (A16 to A23) and middle address (A8 to A15), targeting the page between xxxx00$_{16}$ and xxxxFF$_{16}$.

Execute the page program command as explained here following.

(1) Transfer the "41$_{16}$" command code with the 1st byte.

(2) Transfer addresses A_8 to A_{15} and A_{16} to A_{23} with the 2nd and 3rd bytes respectively.

(3) From the 4th byte onward, as write data (D_0–D_7) for the page (256 bytes) specified with addresses A_8 to A_{23} is input sequentially from the smallest address first, that page is automatically written.

When reception setup for the next 256 bytes ends, the BUSY signal changes from the "H" to the "L" level. The result of the page program can be known by reading the status register. For more information, see the section on the Read Status Register Command.

Each block can be write-protected with the lock bit. For more information, see the section on the Lock Bit Program Command. Additional writing is not allowed with already programmed pages.

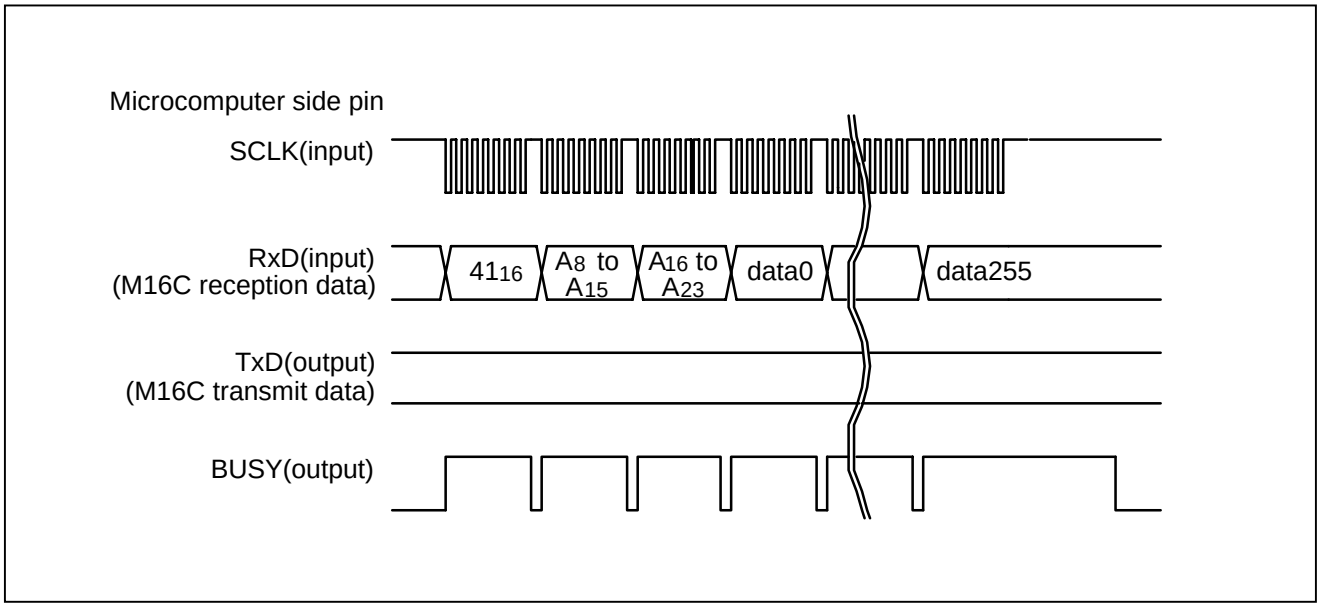


Figure 3.2.7 Timing for the page program

Block Erase Command

This command erases the data in the specified block. Execute the block erase command as explained here following.

(1) Transfer the "20_{16}" command code serially with the 1st byte.

(2) Transfer addresses A_8 to A_{15} and A_{16} to A_{23} with the 2nd and 3rd bytes respectively.

(3) Transfer the verify command code "$D0_{16}$" with the 4th byte. With the verify command code, the erase operation will start for the specified block in the flash memory.

When block erasing ends, the BUSY signal changes from the "H" to the "L" level. After block erase ends, the result of the block erase operation can be known by reading the status register. For more information, see the section on the Read Status Register Command.

Each block can be erase-protected with the lock bit. For more information, see the section on the Lock Bit Program Command.

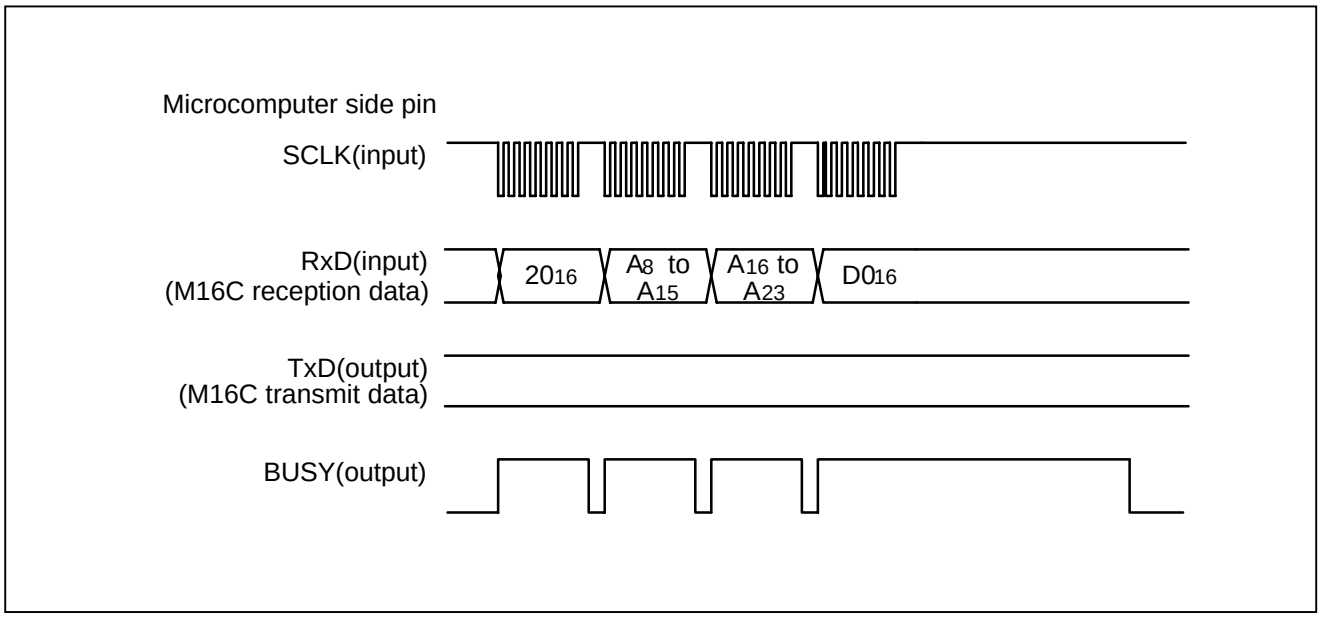


Figure 3.2.8 Timing for block erasing

Erase All Unlocked Blocks Command

This command erases the content of all blocks. Execute the erase all unlocked blocks command as explained here following.

(1) Transfer the "A7$_{16}$" command code serially with the 1st byte.

(2) Transfer the verify command code "D0$_{16}$" with the 2nd byte. With the verify command code, the erase operation will start and continue for all blocks in the flash memory.

When block erasing ends, the BUSY signal changes from the "H" to the "L" level. The result of the erase operation can be known by reading the status register. For more information, see the section on the Read Status Register Command.

Each block can be erase-protected with the lock bit. For more information, see the section on the Lock Bit Program Command.

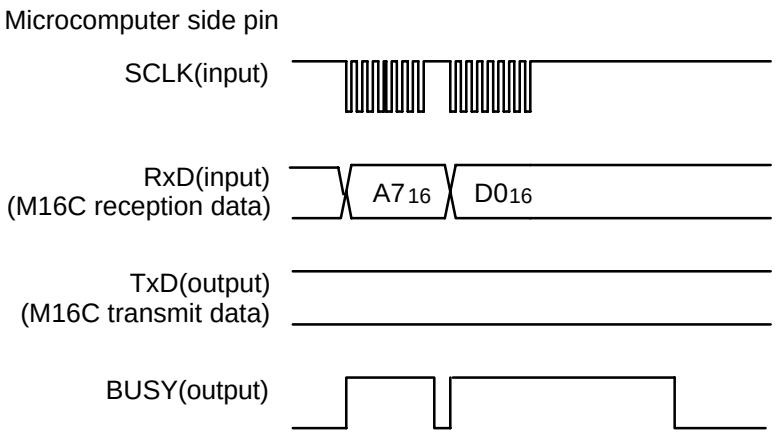


Figure 3.2.9 Timing for erasing all unlocked blocks

Read Status Register Command

This command reads status information. Execute the read status register command as explained here following.

(1) Transfer the “70_{16}” command code serially with the 1st byte.

(2) Output the contents of the status register (SRD) specified with the 2nd byte and the contents of status register 1 (SRD1) specified with the 3rd byte.

Details of "status register", refer to a section of the Status Register(SRD).

Details of "status register 1", refer to a section of the Status Register 1(SRD1).

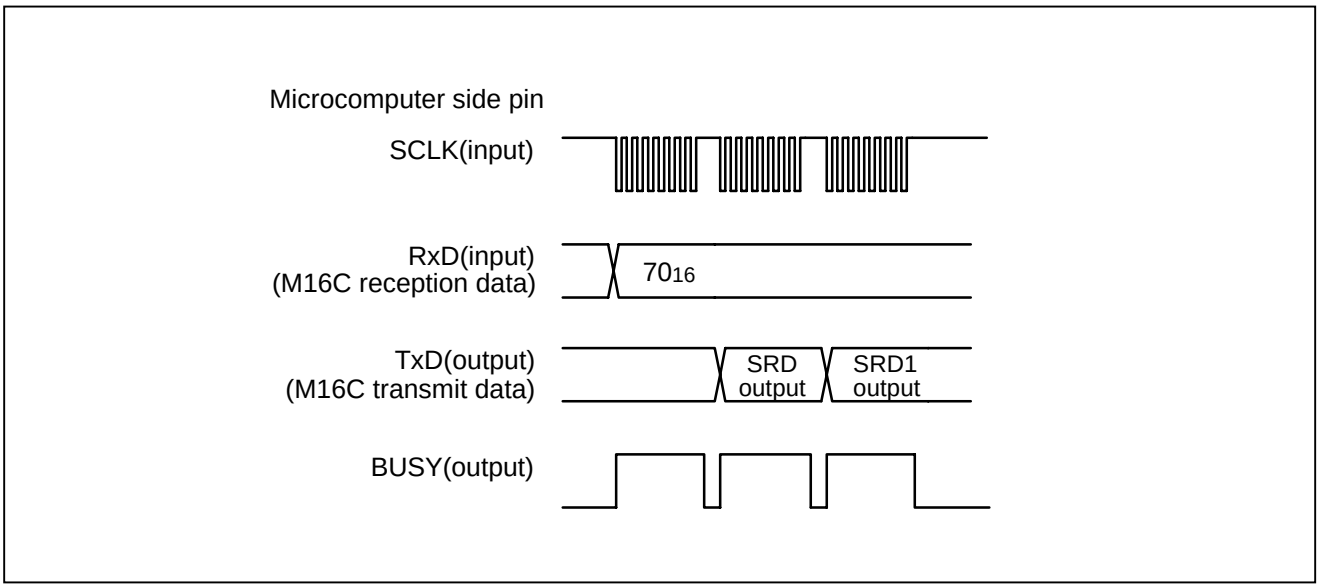


Figure 3.2.10 Timing for reading the status register

Status Register (SRD)

The status register indicates operating status of the flash memory and status such as whether an erase operation or a program ended successfully or in error. It can be read by executing the read status register command (70_{16}). Also, the status register becomes "80_{16}" by executing the clear status register command (50_{16}).

After being reset, the status register outputs “80_{16}” by executing the read status register command.

Table 3.2.3 gives the definition of each status register bit.

Table 3.2.3 Status register (SRD)

SRD0 bits	Status name	Definition	
		"1"	"0"
SR7 (bit7)	Write state machine (WSM) status	Ready	Busy
SR6 (bit6)	Reserved	-	-
SR5 (bit5)	Erase status	Terminated in error	Terminated normally
SR4 (bit4)	Program status	Terminated in error	Terminated normally
SR3 (bit3)	Block status after program	Terminated in error	Terminated normally
SR2 (bit2)	Reserved	-	-
SR1 (bit1)	Reserved	-	-
SR0 (bit0)	Reserved	-	-

Write State Machine (WSM) Status (SR7)

The write state machine (WSM) status indicates the operating status of the flash memory. When power is turned on, “1” (ready) is set for it. The bit is set to “0” (busy) during an auto write or auto erase operation, but it is set back to “1” when the operation ends.

Erase Status (SR5)

The erase status reports the operating status of the auto erase operation. If an erase error occurs, it is set to “1”. If the clear status register command is executed, the erase status is set to "0".

Program Status (SR4)

The program status reports the operating status of the auto write operation. If a write error occurs, it is set to “1”. If the clear status register command is executed, the program status is set to "0".

Block Status After Program (SR3)

If excessive data is written, “1” is set for the block status after-program at the end of the page write operation. The block status after-program becomes "0" by executing the clear status register command.

If “1” is written for any of the SR5, SR4 or SR3 bits, the page program, block erase, erase all unlocked blocks and lock bit program commands are not accepted. Before executing these commands, execute the clear status register command (50_{16}).

Status Register 1 (SRD1)

Status register 1 indicates the status of serial communications, results from check sum comparisons. It can be read after the SRD by executing the read status register command (70_{16}). Also, bit SR9 of the status register 1 becomes "0" by executing the clear status register command (50_{16}).

Table 3.2.4 gives the definition of each status register 1 bit.

Table 3.2.4 Status register 1 (SRD1)

SRD0 bits	Status name	Definition	
		"1"	"0"
SR7 (bit7)	Boot update completed bit	Update completed	Not update
SR6 (bit6)	Reserved	-	-
SR5 (bit5)	Reserved	-	-
SR4 (bit4)	Check sum match bit	Match	Mismatch
SR3 (bit3)	Reserved	-	-
SR2 (bit2)	Reserved	-	-
SR1 (bit1)	Data receive time out	Time out	Normal operation
SR0 (bit0)	Reserved	-	-

Boot Update Completed Bit (SR15)

This flag indicates whether the rewrite program was downloaded to the internal RAM or not, using the download function. After the rewrite program is transferred serially using the download function, this bit is set to "1".

Check Sum Consistency Bit (SR12)

This flag indicates whether the check sum matches or not when a rewrite program is downloaded for execution using the download function.

Data Reception Time Out (SR9)

This flag indicates when a time out error is generated during data reception. If this flag is set during data reception, the received data is discarded and the microcomputer returns to the command wait state.

Clear Status Register Command

This command clears the bits (SR3–SR5, SR9) which are set to "1" when the operation of the status register or status register 1 ends in error. When the “50_{16}” command code is sent serially with the 1st byte, the aforementioned bits are set to "0". When the clear status register operation ends, the BUSY signal changes from the “H” to the “L” level.

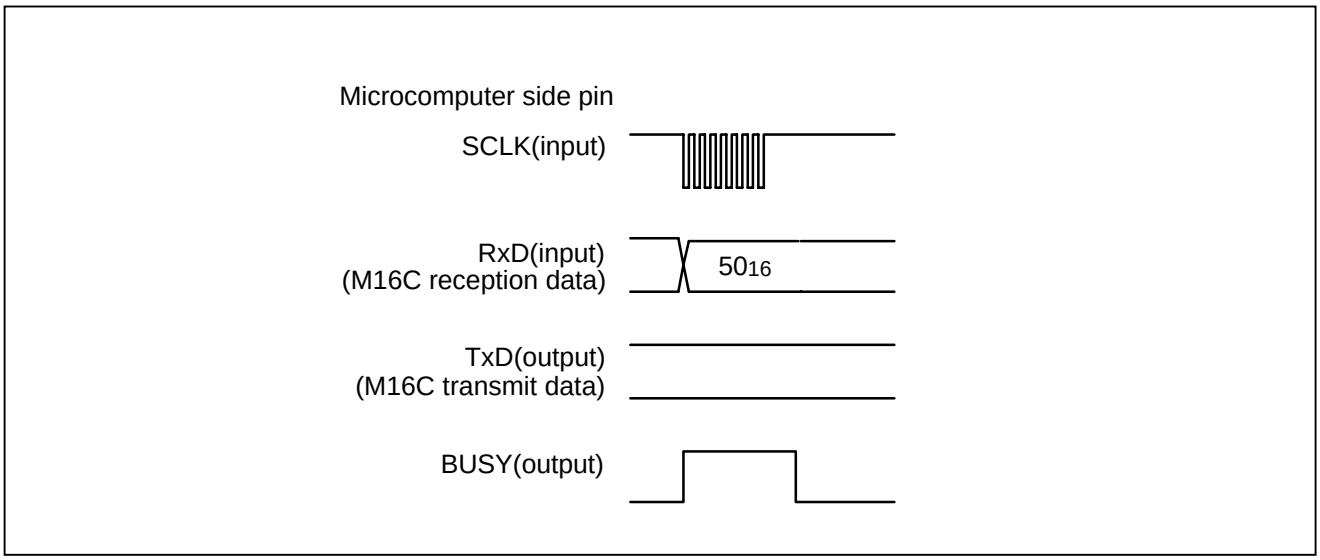


Figure 3.2.11 Timing for clearing the status register

Read Lock Bit Status Command

This command reads the lock bit status of the specified block. Execute the read lock bit status command as explained here following. Write the highest address of specified block for addresses A8 to A23. Each block can be locked or unlocked.

locked : Erase and Writing is not possible
unlocked : Erase and Writing is possible

(1) Transfer the "71_{16}" command code with the 1st byte.
(2) Transfer addresses A_8 to A_{15} and A_{16} to A_{23}, which are the highest address in the specified block with the 2nd and 3rd bytes respectively.
(3) The lock bit data is output with the 4th byte. The 6th bit of the output data shows the status. "1" indicates that the block is unlocked, "0" that it is locked.

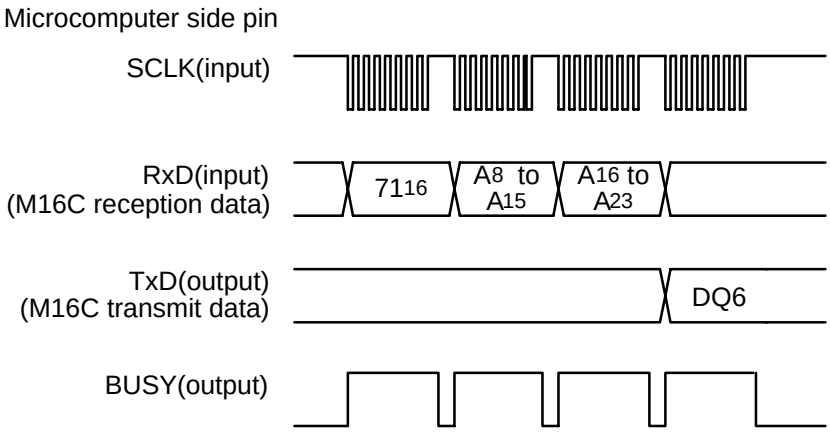


Figure 3.2.12 Timing for reading lock bit status

Lock Bit Program Command

This command writes “0” (lock) for the lock bit of the specified block. Execute the lock bit program command as explained here following. Write the highest address of specified block for addresses A8 to A23.

(1) Transfer the “77_{16}” command code serially with the 1st byte.
(2) Transfer addresses A_8 to A_{15} and A_{16} to A_{23}, which are the highest address in the specified block with the 2nd and 3rd bytes respectively.
(3) Transfer the verify command code “$D0_{16}$” with the 4th byte. With the verify command code, “0” is written for the lock bit of the specified block.

When writing ends, the BUSY signal changes from the “H” to the “L” level. Lock bit status can be read with the read lock bit status command.
If the user want to make effective the contents of the lock bit, the user need make the write protect pin of the flash memory an "L" level. If the user want to make ineffective the contents of the lock bit, the user need make the write protect pin of the flash memory an "H" level. Details of the write protect pin, refer to the data sheet of flash memory (Refer to M5M29GB/T160BVP, M5M29GB/T320BVP data sheets).
The lock bit returns to "1" (unlocked) by setting the write protect pin of the flash memory to "H" level first and then executing the block erase or erase all unlocked blocks command.

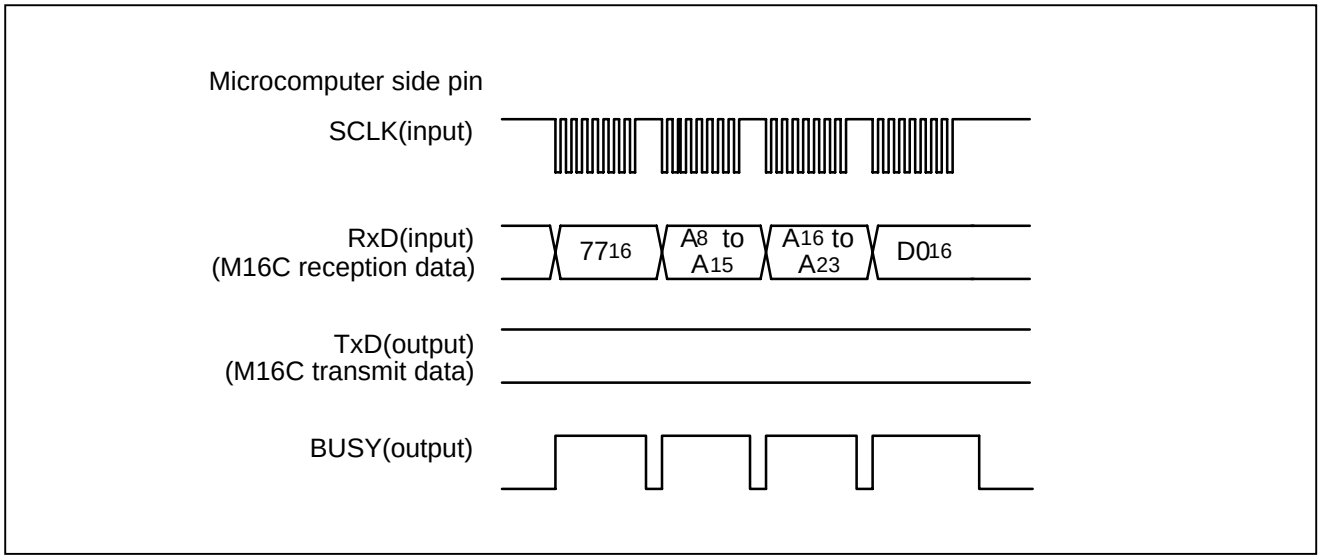


Figure 3.2.13 Timing for lock bit program

Read Check Data

This command reads the check data that confirms that the write data, which the serial programmer sent with the page program command, was successfully received by the microcontroller. After reading out the 2-byte check data, the check data becomes "0000$_{16}$". Execute the Read Check Data command as explained here following.

Table 3.2.5 Formula of check data

Check data form	Calculation method
CRC operation	CRC code is obtained using M16C CRC operation circuit.

(1) Transfer the "FD$_{16}$" command code serially with the 1st byte.

(2) The check data (low) is output with the 2nd byte and the check data (high) with the 3rd.

To use this read check data command, first execute the command and then set the check data to "0000$_{16}$". Next, execute the page program command the required number of times. After that, when the read check command is executed again, the check data for all of the written data that was sent with the page program command during this time is read.

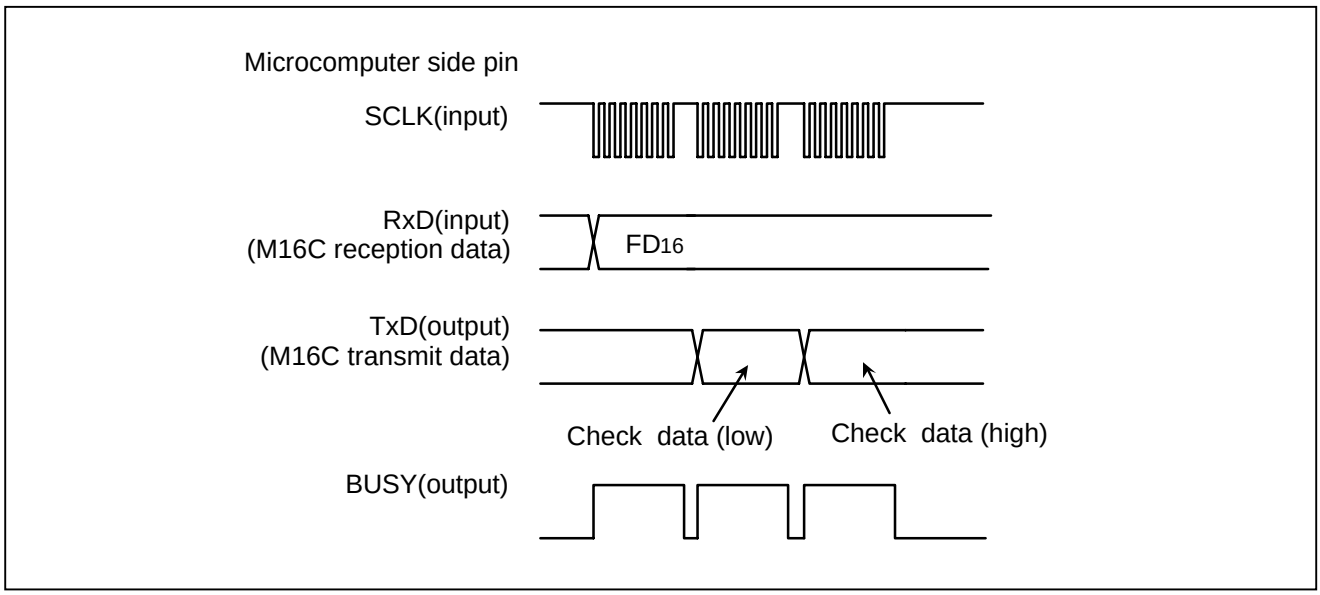


Figure 3.2.14. Timing for read check data

3.3 Overview of boot loader mode 2 (clock asynchronized)

In boot loader mode 2, software commands, addresses and data are input and output between the MCU and serial programmer (*1) using 2-wire clock-asynchronized serial I/O (UART1). To use this mode, the main clock input oscillation frequency should be no fewer than 2MHz, nor more than 20MHz. Bootloader mode 2 is engaged by applying an "L" level to the P_{65} pin to release the reset.
The TxD pin is for CMOS output. Data transfer is in 8-bit units with LSB first, 1 stop bit and parity OFF.
After the reset is released, connections can be established at 9,600 bps. And then the baud rate can also be changed from 9,600 bps to 19,200, 38,400, 57,600 or 115,200 bps.
Here following are explained initial communications with serial programmer, how frequency is identified, and two functions (Download function and Flash memory control function) supported by bootloader mode 2.

*1: M16C FlashStart can be used as the serial programmer.

3.3.1 Initial communications with serial programmer

After a reset, the bit rate generator is adjusted to 9,600 bps by establishing initial communications with serial programmer.

(1) Adjust the bit rate to 9,600 bps first, then transmit "00_{16}" from a serial programmer 16 times at transfer intervals of a minimum 15 ms. (The MCU sets the bit rate generator so that "00_{16}" can be successfully received.)

(2) The MCU outputs the "$B0_{16}$" check code and initial communications end successfully (* 2) .
Figure 3.3.1 shows a protocol of initial communication with a serial programmer. Figure 3.3.2 shows a I/O timing of initial communication.

*2. If the serial programmer cannot receive "$B0_{16}$" successfully, change the input oscillation frequency of the main clock.

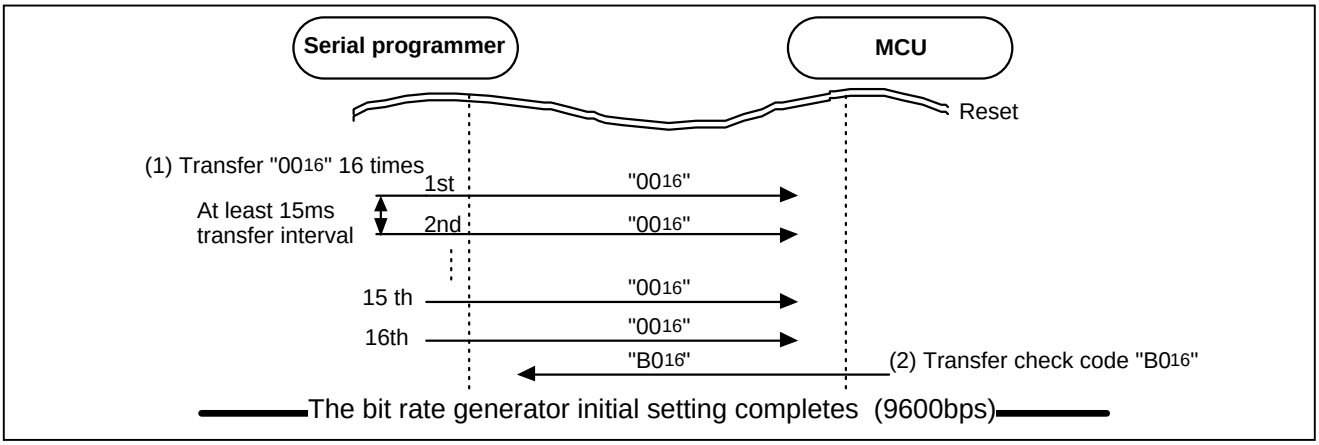


Figure 3.3.1 Serial programmer and initial communication

I/O Timing from Microcontroller Side

(1) Turn ON power to target board.
(2) Start flash memory write control.
(3) Enter serial writing mode by canceling reset.
(4) Turn ON power to serial programmer.
(5) Started initial communication.
(6) Ended initial communication.

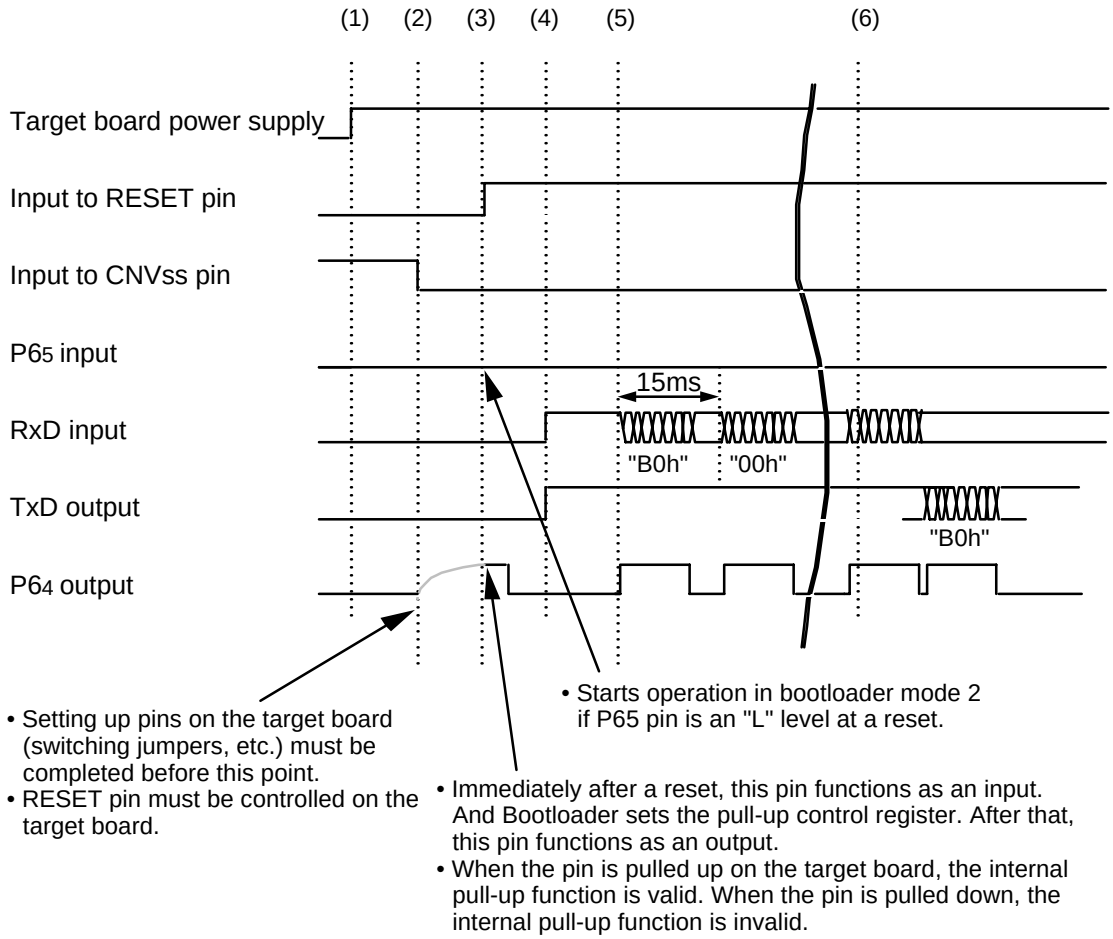


Figure 3.3.2 I/O Timing for initial communication

3.3.2 Main clock input oscillation frequency and baud rate

Desired baud rate cannot be attained with some main clock input oscillation frequencies. Table 3.3.1 gives the main clock input oscillation frequency and the baud rate that can be attained for.

Table 3.3.1 Main clock oscillation frequency and the baud rate

Main clock input operation frequency (MHz)	Baud rate 9,600bps	Baud rate 19,200bps	Baud rate 38,400bps	Baud rate 57,600bps	Baud rate 115,200bps
20MHz	√	√	√	√	√
16MHz	√	√	√	√	√
12MHz	√	√	√	√	-
10MHz	√	√	√	√	-
8MHz	√	√	√	√	-
7.3728MHz	√	√	√	√	√
6MHz	√	√	√	-	-
5MHz	√	√	√	-	-
4.5MHz	√	√	√	√	-
4.194304MHz	√	√	√	-	-
4MHz	√	√	-	-	-
3.58MHz	√	√	√	√	√
2MHz	√	-	-	-	-

√ :Communications possible

- :Communications not possible

3.3.3 Download Function

Functional Description

The download function of M16C/80 Bootloader is to download a rewrite program (*1) to internal RAM in the microcomputer using serial communications and then let the processing jump to the address in the RAM where the downloaded program has been located.

*1: The rewrite program should be prepared by the user according to the following notes.

- The rewrite program should have two functions: (1) control function to write, erase and read to/from the external flash and (2) communication function to communicate with a serial writer.
- When using a stack in the rewrite program, please setup the stack pointer within the program.
- When the download is completed, the microcomputer starts the operation in single chip mode. Please change the processor mode from the single chip mode to microprocessor mode using the rewrite program before starting controls such as writing or erasing to the external flash memory.
- Please do not use any interrupts in the rewrite program.
- For the download area, please refer to the memory map of the Appendix 1.

Software commands

Table 3.3.2 lists the software commands for bootloader mode 2.

Table 3.3.2 Software commands for download (Boot loader mode 2)

	Control command	1st byte transfer	2nd byte	3rd byte	4th byte	5th byte	6th byte	
1	Download function	FA_{16}	Size (low)	Size (high)	Check-sum	Data input	To required number of times	
2	Download result output	FA_{16}	Data input					
3	Version data output function	FB_{16}	Version data output	Version data output	Version data output	Version data output	Version data output	Version data output to 9th byte
4	Baud rate 9600	$B0_{16}$	$B0_{16}$					
5	Baud rate 19200	$B1_{16}$	$B1_{16}$					
6	Baud rate 38400	$B2_{16}$	$B2_{16}$					
7	Baud rate 57600	$B3_{16}$	$B3_{16}$					
8	Baud rate 115200	$B4_{16}$	$B4_{16}$					

Note: Shading indicates transfer from microcomputer to serial programmer. All other data is transferred from the serial programmer to the microcomputer.

Download

This command downloads a rewrite program to the internal RAM. The program as downloaded is stored in the internal RAM from address 600_{16} onward.

After a reset, the downloaded program is held in the internal RAM.

Execute the download command as explained here following.

(1) Transfer the "FA_{16}" command code serially with the 1st byte.

(2) Transfer the program size serially with the 2nd and 3rd bytes, as follows: low order size with the 2nd byte, and high-order size with the 3rd byte.

(3) Transfer the check sum serially with the 4th byte. The check sum is added to all data sent with the 5th byte onward.

(4) The program to execute is sent with the 5th byte onward. The size of the program to be transferred will vary depending on the internal RAM size. (Please refer to "3.6 Memory Map" about the size of the rewrite program.)

When all data has been transferred, the microcomputer automatically executes the download result output command.

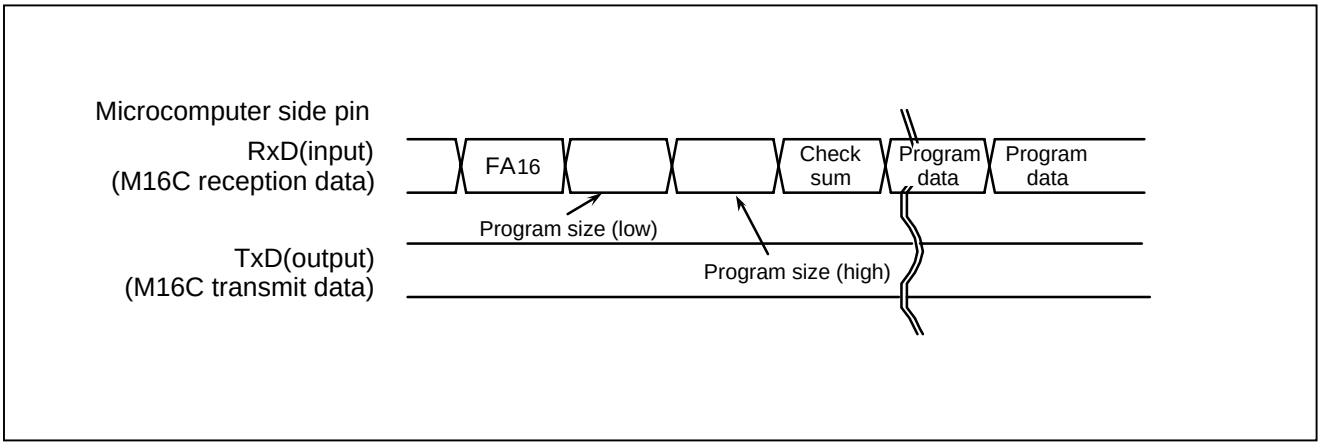


Figure 3.3.3 Timing for download

Download Result Output

After downloaded, the transferred check sum value from the serial programmer and the check sum value obtained by received data are compared. When the check sum values are matched, "FA_{16}" and "00_{16}"(success) are sent back, and then the processing jumps to the beginning of the downloaded program to execute it. When the values are not matched, " FA_{16}" and "01_{16}"(failure) are sent back and boot program stored in the microcomputer is transferred to RAM again, then this program is executed. (Return to the original state)

When the Download Function has been completed, the bootloader (microcomputer) outputs the execution result as explained here following.

(1) When the Download Function has been completed, output the "FA_{16}" command code with the 1st byte.
(2) Output the download result code ("00_{16}" : success / "01_{16}" : failure) with the 2nd byte.

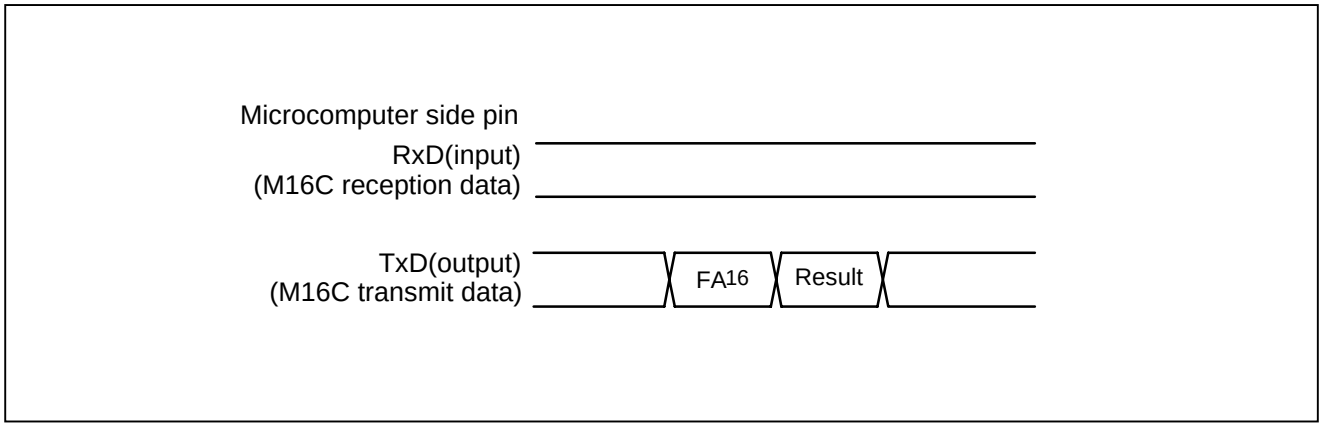


Figure 3.3.4 The timing of download result

Version Information Output Command

This command outputs the version information data of bootloader.

Execute the version information output command as explained here following.

(1) Transfer the "FB_{16}" command code serially with the 1st byte.

(2) The version information will be output serially from the 2nd byte to the 9th byte. This data is composed of 8 ASCII code characters (*1) .

*1: Version data format is 8 characters by ASCII code,

"VER. X. XX" (X: number).

It is output from "V".

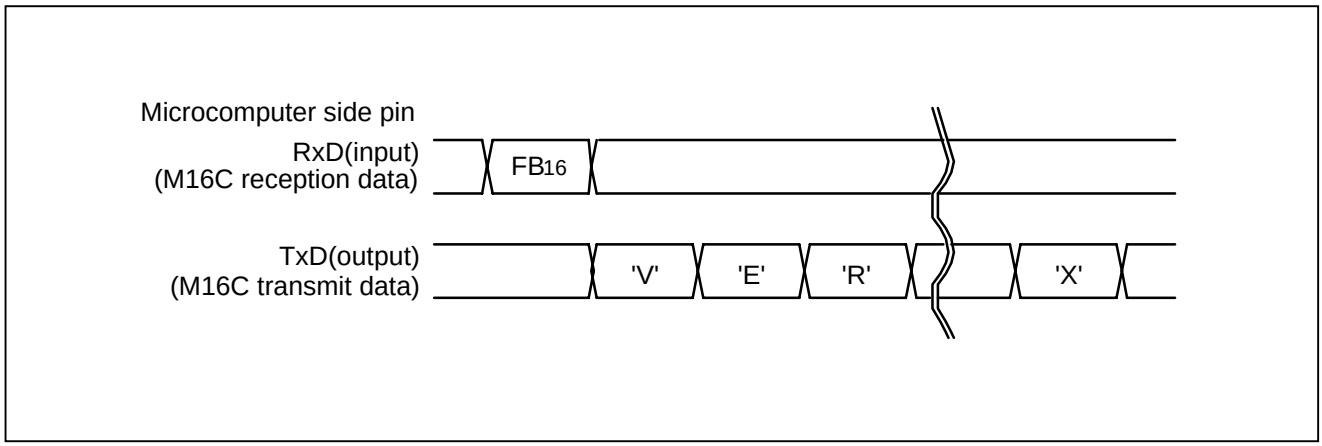


Figure 3.3.5 Timing for version information output

Baud Rate 9600

This command changes baud rate to 9,600 bps. Execute it as follows.

(1) Transfer the "B0$_{16}$" command code serially with the 1st byte.

(2) After the "B0$_{16}$" check code is output with the 2nd byte, change the baud rate to 9,600 bps.

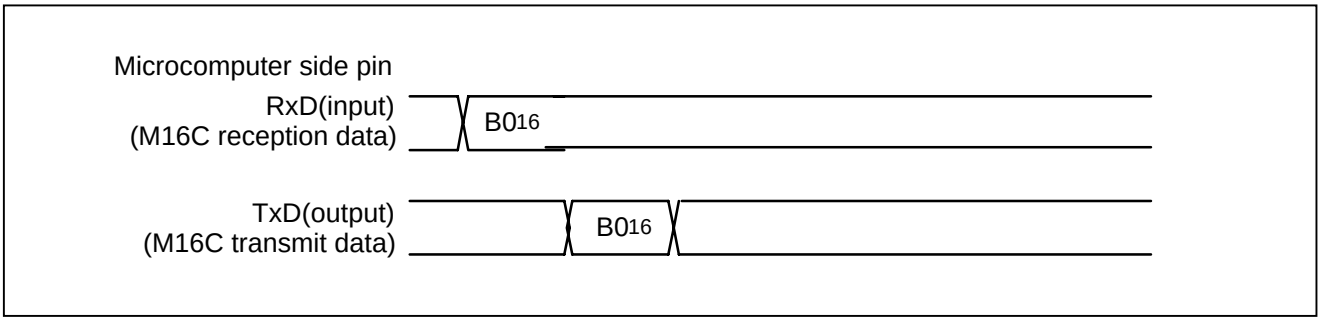


Figure 3.3.6 Timing of baud rate 9600

Baud Rate 19200

This command changes baud rate to 19,200 bps. Execute it as follows.

(1) Transfer the "B1$_{16}$" command code serially with the 1st byte.

(2) After the "B1$_{16}$" check code is output with the 2nd byte, change the baud rate to 19,200 bps.

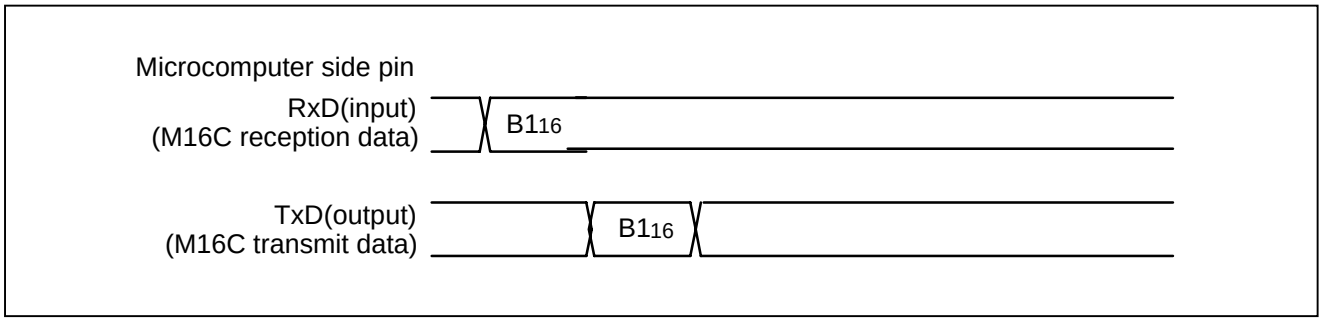


Figure 3.3.7 Timing of baud rate 19200

Baud Rate 38400

This command changes baud rate to 38,400 bps. Execute it as follows.

(1) Transfer the "B2$_{16}$" command code serially with the 1st byte.

(2) After the "B2$_{16}$" check code is output with the 2nd byte, change the baud rate to 38,400 bps.

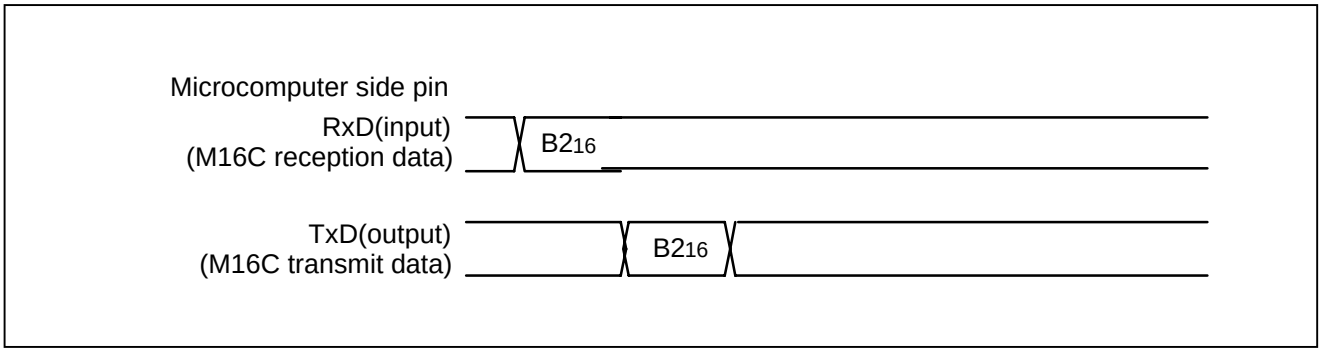


Figure 3.3.8 Timing of baud rate 38400

Baud Rate 57600

This command changes baud rate to 57,600 bps. Execute it as follows.

(1) Transfer the "B3$_{16}$" command code serially with the 1st byte.

(2) After the " B3$_{16}$" check code is output with the 2nd byte, change the baud rate to 57,600 bps.

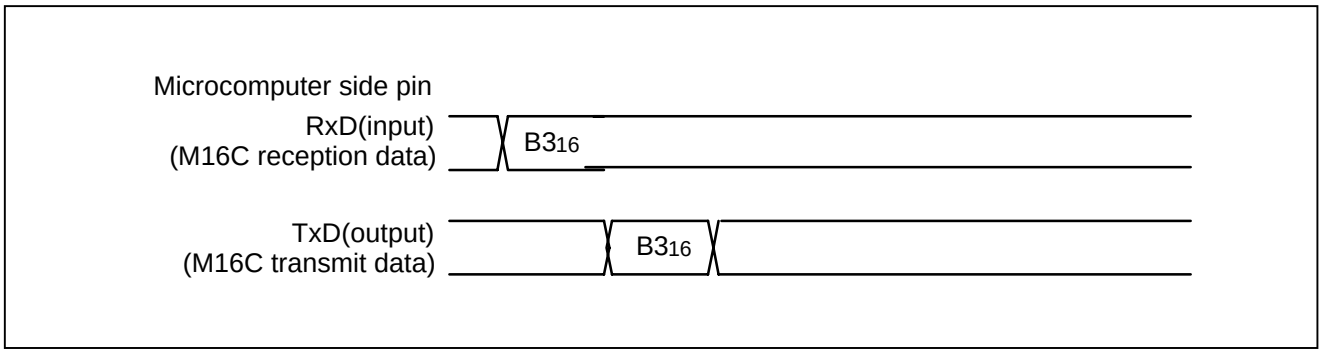


Figure 3.3.9 Timing of baud rate 57600

Baud Rate 115200

This command changes baud rate to 115,200 bps. Execute it as follows.

(1) Transfer the "$B4_{16}$" command code serially with the 1st byte.

(2) After the "$B4_{16}$" check code is output with the 2nd byte, change the baud rate to 19,200 bps.

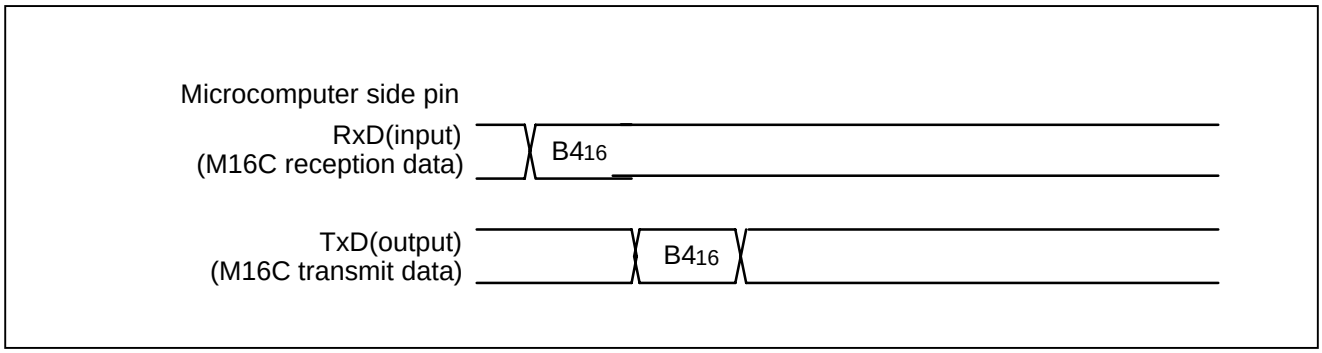


Figure 3.3.10 Timing of baud rate 115200

3.3.4 Flash Memory Control Function

Functional Description

If an external flash memory is M5M29GB/T160BVP, M5M29GB/T320BVP (made by MITSUBISHI) or MCM with these flash memories, the M16C/80 Bootloader is able to execute writing and erasing without rewrite program. (A connection example is shown in "3.7 Connection example of bootloader")

The M16C/80 Bootloader writes and erases a program to flash memory by communicating commands and data with serial programmer.

Software Commands

The following table lists the flash memory control commands and I/O data.

When only an external flash memory is M5M29GB/T160BVP, M5M29GB/T320BVP or MCM with these flash memories, the user is able to use these commands.

Commands from 10 to 14 are the commands for clock asynchronous communication control.

About these commands, refer to the section of Bootloader mode 2 Download function.

Table 3.3.3 Software commands for flash memory control (Bootloader mode 2)

	Control command	1st byte transfer	2nd byte	3rd byte	4th byte	5th byte	6th byte	
1	Page read	FF_{16}	Address (middle)	Address (high)	Data output	Data output	Data output	Data output to 259th byte
2	Page program	41_{16}	Address (middle)	Address (high)	Data input	Data input	Data input	Data input to 259th byte
3	Block erase	20_{16}	Address (middle)	Address (high)	$D0_{16}$			
4	Erase all unlocked blocks	$A7_{16}$	$D0_{16}$					
5	Read status register	70_{16}	SRD output	SRD1 output				
6	Clear status register	50_{16}						
7	Read lock bit status	71_{16}	Address (middle)	Address (high)	Lock bit data output			
8	Lock bit program	77_{16}	Address (middle)	Address (high)	$D0_{16}$			
9	Read check data	FD_{16}	Data output (low)	Data output (high)				
10	Baud rate 9600	$B0_{16}$	$B0_{16}$					
11	Baud rate 19200	$B1_{16}$	$B1_{16}$					
12	Baud rate 38400	$B2_{16}$	$B2_{16}$					
13	Baud rate 57600	$B3_{16}$	$B3_{16}$					
14	Baud rate 115200	$B4_{16}$	$B4_{16}$					

Note 1: Shading indicates transfer from microcomputer to serial programmer. All other data is transferred from the serial programmer to the microcomputer.

Note 2: SRD refers to status register data. SRD1 refers to status register 1 data.

Page Read Command

This command reads the specified page (256 bytes) in the flash memory sequentially one byte at a time. The read area is set with a high address (A16 to A23) and middle address (A8 to A15), targeting the 256 bytes from xxxx00$_{16}$ to xxxxFF$_{16}$. (Refer to Figure 3.3.11)

Execute the page read command as explained here following.

(1) Transfer the "FF$_{16}$" command code serially with the 1st byte.

(2) Transfer addresses A_8 to A_{15} and A_{16} to A_{23} with the 2nd and 3rd bytes respectively.

(3) From the 4th byte onward, data (D_0–D_7) for the page (256 bytes) specified with addresses A_8 to A_{23} will be output sequentially from the smallest address first in sync with the fall of the clock.

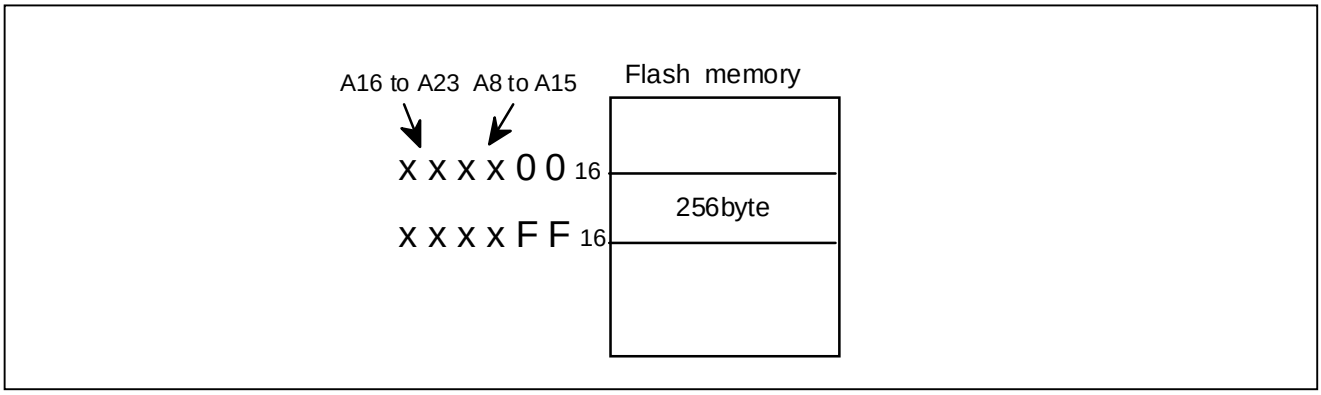


Figure 3.3.11 The designation of the address and command applicable area

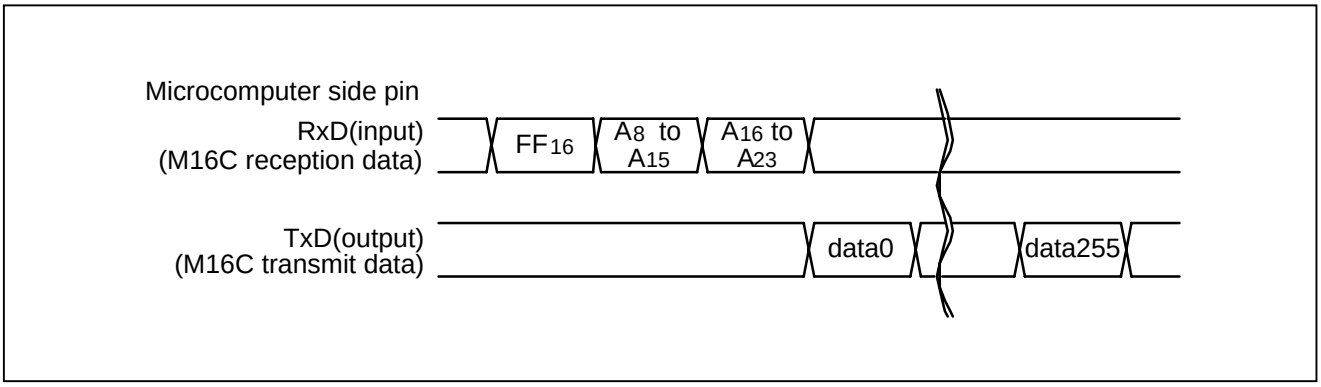


Figure 3.3.12 Timing for page read

Page Program Command

This command writes the specified page (256 bytes) in the flash memory sequentially one byte at a time. The area to be written to is set using a high address (A16 to A23) and middle address (A8 to A15), targeting the page between xxxx00$_{16}$ and xxxxFF$_{16}$.

Execute the page program command as explained here following.

(1) Transfer the "41$_{16}$" command code with the 1st byte.

(2) Transfer addresses A_8 to A_{15} and A_{16} to A_{23} with the 2nd and 3rd bytes respectively.

(3) From the 4th byte onward, as write data (D_0–D_7) for the page (256 bytes) specified with addresses A_8 to A_{23} is input sequentially from the smallest address first, that page is automatically written.

The result of the page program can be known by reading the status register. For more information, see the section on the Read Status Register Command.

Each block can be write-protected with the lock bit. For more information, see the section on the Lock Bit Program Command. Additional writing is not allowed with already programmed pages.

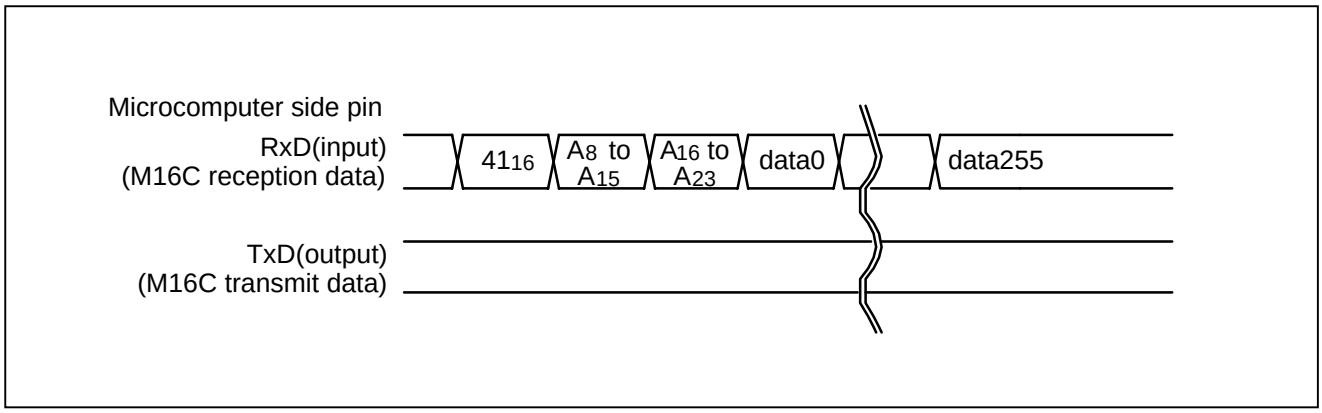


Figure 3.3.13 Timing for the page program

Block Erase Command

This command erases the data in the specified block. Execute the block erase command as explained here following.

(1) Transfer the "20_{16}" command code serially with the 1st byte.

(2) Transfer addresses A_8 to A_{15} and A_{16} to A_{23} with the 2nd and 3rd bytes respectively.

(3) Transfer the verify command code "$D0_{16}$" with the 4th byte. With the verify command code, the erase operation will start for the specified block in the flash memory.

After block erase ends, the result of the block erase operation can be known by reading the status register. For more information, see the section on the Read Status Register Command.

Each block can be erase-protected with the lock bit. For more information, see the section on the LockBit Program Command.

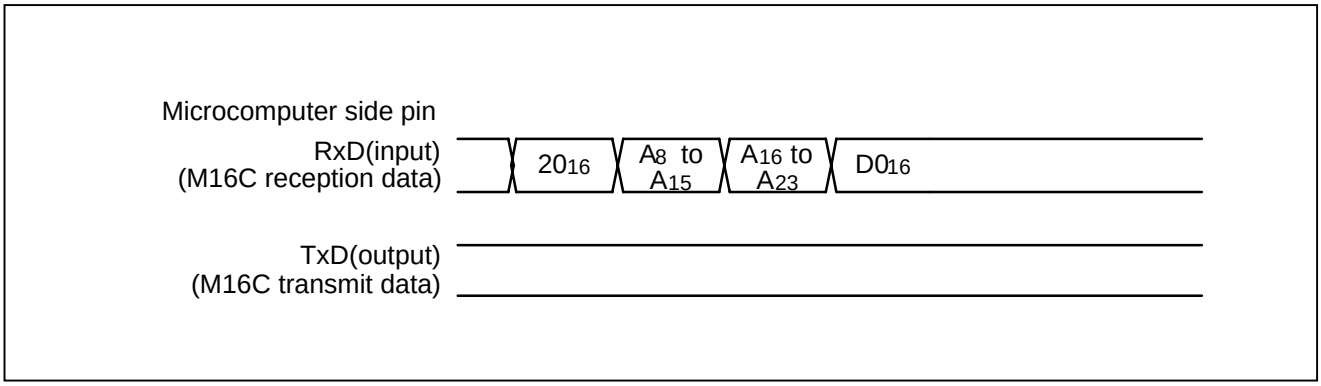


Figure 3.3.14 Timing for block erasing

Erase All Unlocked Blocks Command

This command erases the content of all blocks. Execute the erase all unlocked blocks command as explained here following.

(1) Transfer the “A7$_{16}$” command code serially with the 1st byte.

(2) Transfer the verify command code “D0$_{16}$” with the 2nd byte. With the verify command code, the erase operation will start and continue for all blocks in the flash memory.

The result of the erase operation can be known by reading the status register. For more information, see the section on the Read Status Register Command.
Each block can be erase-protected with the lock bit. For more information, see the section on the Lock Bit Program Command.

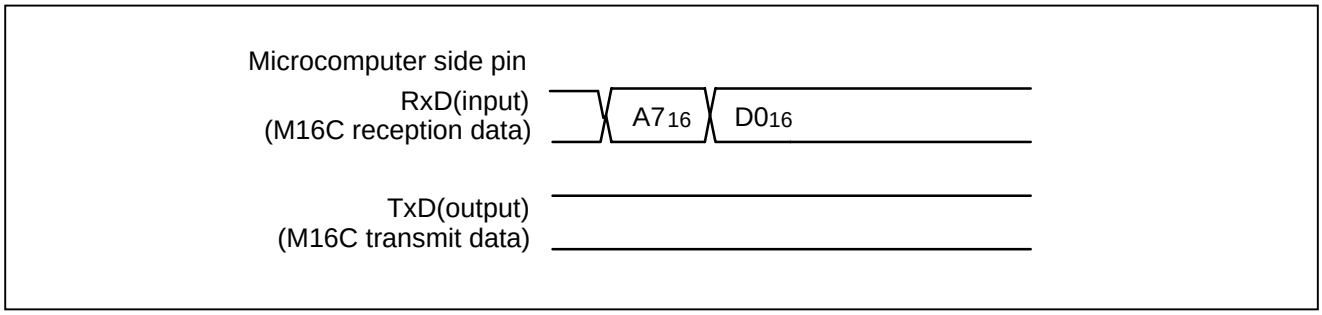


Figure 3.3.15 Timing for erasing all unlocked blocks

Read Status Register Command

This command reads status information. Execute the read status register command as explained here following.

(1) Transfer the “70$_{16}$” command code serially with the 1st byte.

(2) Output the contents of the status register (SRD) specified with the 2nd byte and the contents of status register 1 (SRD1) specified with the 3rd byte.

Details of "status register", refer to section "Status Register(SRD)" of bootloader mode 1.
Details of "status register 1", refer to section "Status Register 1(SRD1)" of bootloader mode 1.

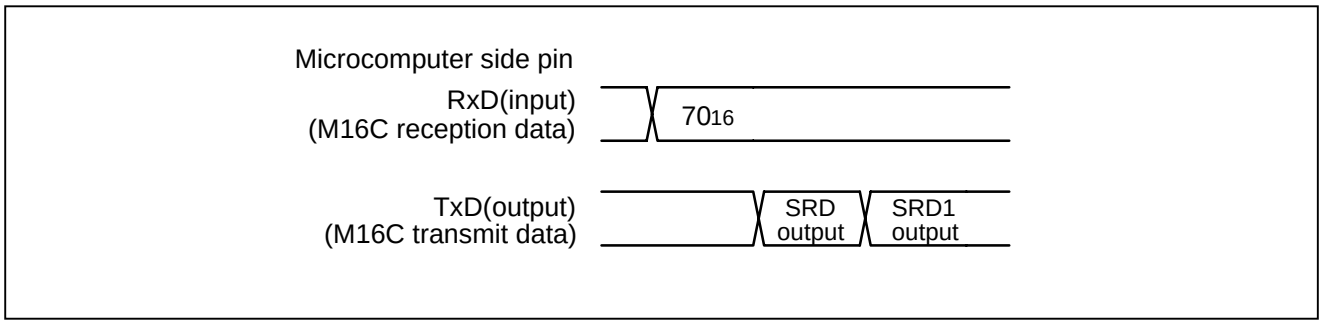


Figure 3.3.16 Timing for reading the status register

Clear Status Register Command

This command clears the bits (SR3–SR5, SR9) which are set to "1" when the operation of the status register or status register 1 ends in error. When the “50_{16}” command code is sent serially with the 1st byte, the aforementioned bits are set to "0".

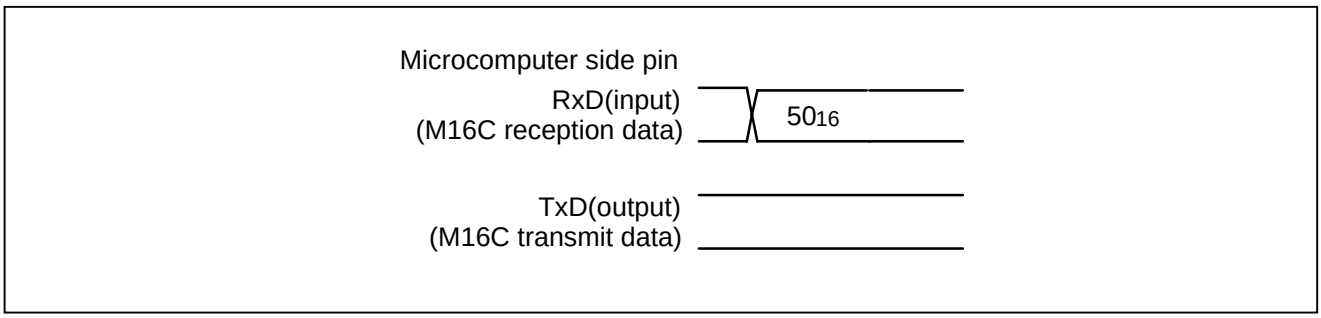


Figure 3.3.17 Timing for clearing the status register

Read Lock Bit Status Command

This command reads the lock bit status of the specified block. Execute the read lock bit status command as explained here following. Write the highest address of the specified block for addresses A8 to A23. Each block can be locked or unlocked.

locked : Erase and Writing is not possible
unlocked : Erase and Writing is possible

(1) Transfer the “71_{16}” command code with the 1st byte.

(2) Transfer addresses A_8 to A_{15} and A_{16} to A_{23}, which are the highest addresses in the specified block with the 2nd and 3rd bytes respectively.

(3) The lock bit data is output with the 4th byte. The 6th bit of the output data shows the status. “1” indicates that the block is unlocked, “0” that it is locked.

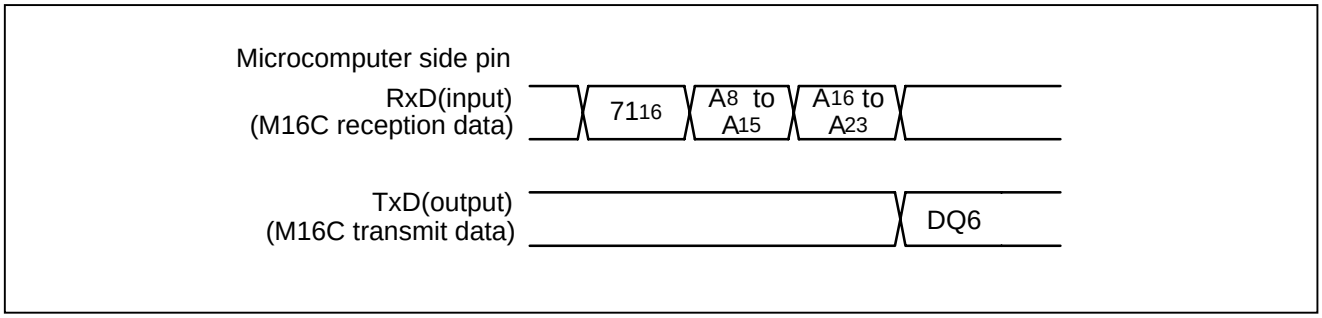


Figure 3.3.18 Timing for reading lock bit status

Lock Bit Program Command

This command writes “0” (lock) for the lock bit of the specified block. Execute the lock bit program command as explained here following. Write the highest address of the specified block for addresses A8 to A23. Each block can be locked or unlocked.

(1) Transfer the “77_{16}” command code serially with the 1st byte.

(2) Transfer addresses A_8 to A_{15} and A_{16} to A_{23}, which are the highest address in the specified block and with the 2nd and 3rd bytes respectively.

(3) Transfer the verify command code “$D0_{16}$” with the 4th byte. With the verify command code, “0” is written for the lock bit of the specified block.

Lock bit status can be read with the read lock bit status command.
If the user want to make effective the contents of the lock bit, the user need make the write protect pin of the flash memory an "L" level. If the user want to make ineffective the contents of the lock bit, the user need make the write protect pin of the flash memory an "H" level. Details of the write protect pin, refer to the data sheet of flash memory (Refer to M5M29GB/T160BVP, M5M29GB/T320BVP data sheets).
The lock bit returns to "1" (unlocked) by setting the write protect pin of the flash memory to "H" level first and then executing the block erase or erase all unlocked blocks command.

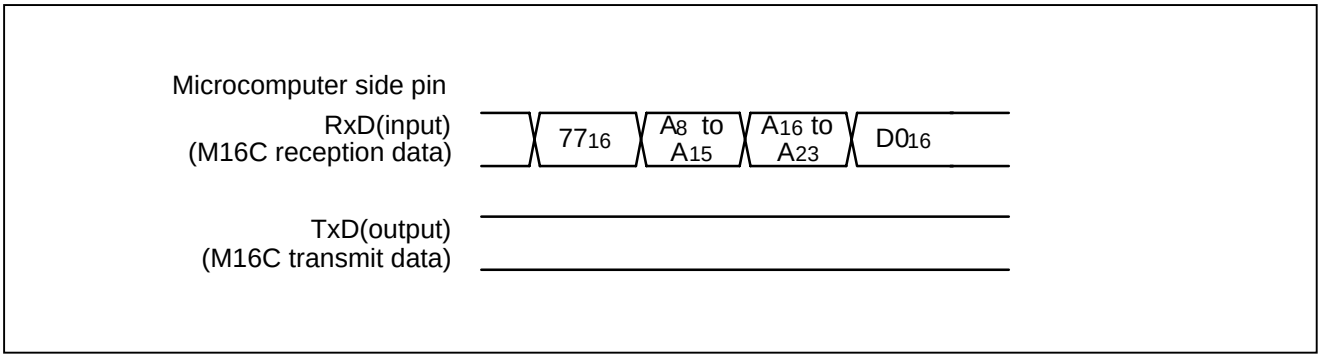


Figure 3.3.19 Timing for lock bit program

Read Check Data

This command reads the check data that confirms that the write data, which the serial programmer sent with the page program command, was successfully received by the microcontroller. After reading out the 2-byte check data, the check data becomes "0000$_{16}$". Execute the Read Check Data command as explained here following.

Table 3.3.4 Formula of check data

Check data form	Calculation method
CRC operation	CRC code is obtained using M16C CRC operation circuit.

(1) Transfer the "FD$_{16}$" command code serially with the 1st byte.

(2) The check data (low) is output with the 2nd byte and the check data (high) with the 3rd.

To use this read check data command, first execute the command and then set the check data to "0000$_{16}$". Next, execute the page program command the required number of times. After that, when the read check command is executed again, the check data for all of the written data that was sent with the page program command during this time is read.

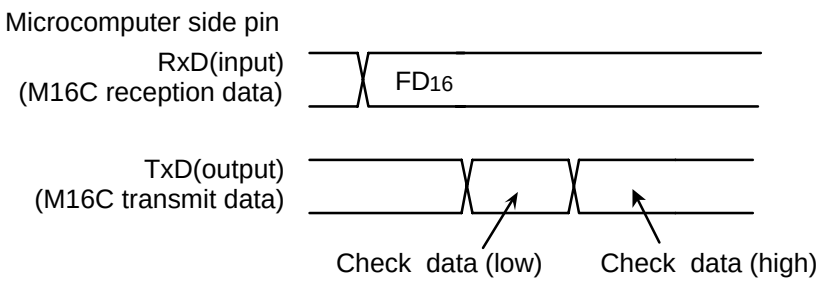


Figure 3.3.20 Timing for read check data

3.4 Examples of how to use bootloader mode 1

In bootloader mode 1, a user can download a rewrite program using MAEC (*1) card type flash memory programmer or Sunny Giken Multi Flash Write (hereinafter referred to as "MFW-1"). Here a rewrite program using MFW-1 is explained. Table 3.4.1 shows commands used when MFW-1 is used. Figure 3.4.1 shows a flow chart of rewriting sample program with MFW-1 used. For the whole program, please refer to 3.8 program list.

*1: MAEC is abbreviated name of Mitsubishi Semiconductor Application Engineering Corporation.

Table 3.4.1 Software commands (Boot loader mode 1)

No.	Control command	1st byte transfer	2nd byte	3rd byte	4th byte	5th byte	6th byte	
1	Page read	FF_{16}	Address (middle)	Address (high)	Data output	Data output	Data output	Data output to 259th byte
2	Page program	41_{16}	Address (middle)	Address (high)	Data input	Data input	Data input	Data input to 259th byte
3	Block erase	20_{16}	Address (middle)	Address (high)	$D0_{16}$			
4	Erase all unlocked blocks	$A7_{16}$	$D0_{16}$					
5	Read status register	70_{16}	SRD output	SRD1 output				
6	Clear status register	50_{16}						
7	Read lock bit status	71_{16}	Address (middle)	Address (high)	Lock bit data output			
8	Lock bit program	77_{16}	Address (middle)	Address (high)	$D0_{16}$			
9	Read check data	FD_{16}	Data output (low)	Data output (high)				
10	Download function	FA_{16}	Size (low)	Size (high)	Check-sum	Data input	To required number of times	
11	Download result output function	FA_{16}	Data output					
12	Version data output function	FB_{16}	Version data output	Version data output	Version data output	Version data output	Version data output	Version data output to 9th byte

Note 1: Shading indicates transfer from microcomputer to serial programmer. All other data is transferred from the serial programmer to the microcomputer.

Note 2: SRD refers to status register data. SRD1 refers to status register 1 data.

Note 3: Command No. 9 is unused when MFW-1 is used.

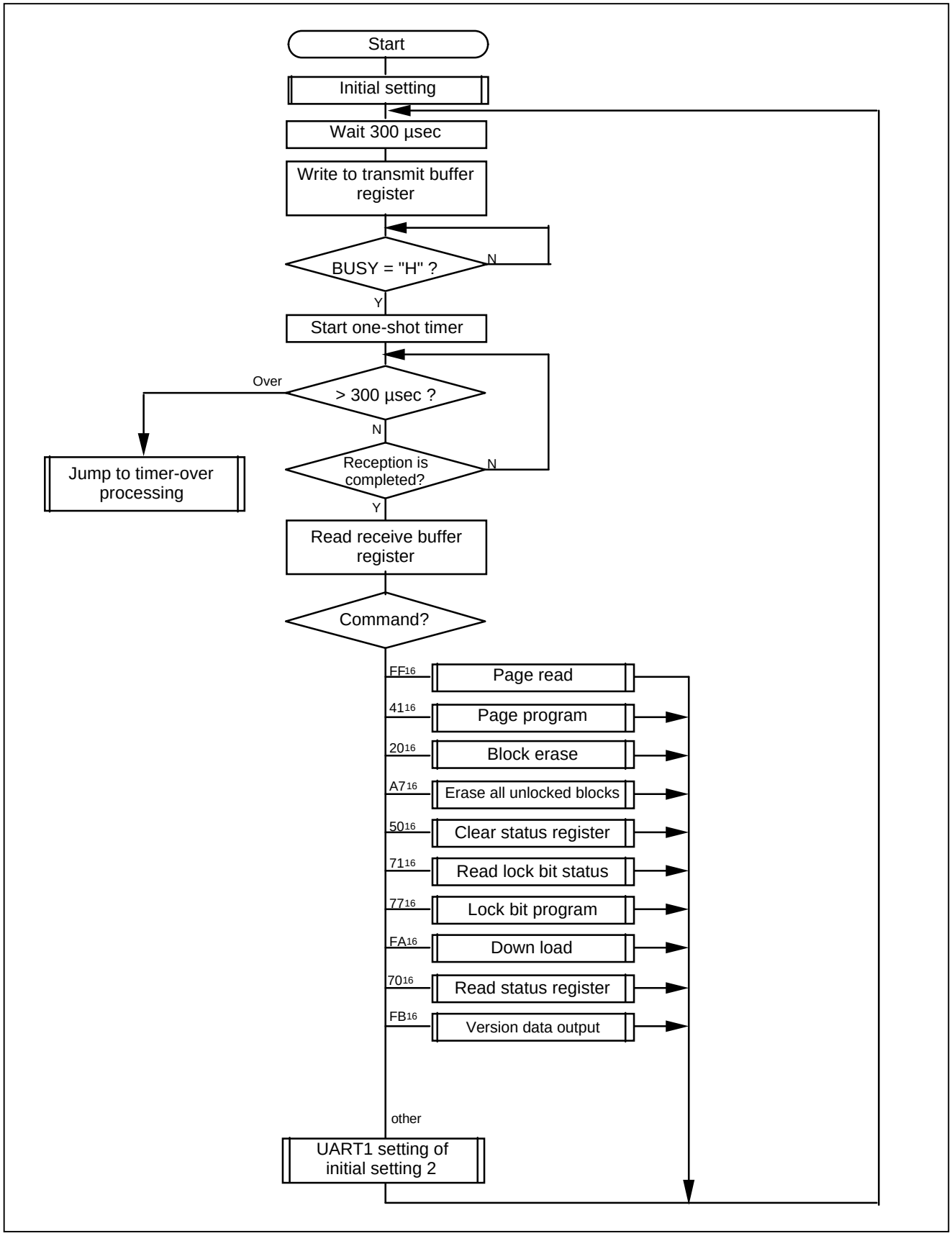


Figure 3.4.1 Flowchart of rewriting sample program with MFW-1 used

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+          Include file                                            +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        .list           off
        .include  sfr800.inc            ; SFR header include
        .include  bl80.inc              ; Bootloader definition include      (1-1)
        .list           on
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+    Version table                                                 +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
        .section  rom,code
        .org            0FFE000h        ; Download address                   (1-2)
        .byte           'VER.1.01'      ; Version infomation
;
;==================================================================
;+          Boot program start                                      +
;==================================================================
Program_start:
;-----------------------------------------------
;+          Initialize_1                +
;-----------------------------------------------
        ldc             #Istack,ISP     ; stack pointer set                  (2)
;
;-----------------------------------------------------------
;+        Processor mode register           +
;+          & System clock control register +
;-----------------------------------------------------------
CPU_set:
mov.b   #3,prcr                         ; Protect off
mov.w   #00000011b,pm0                  ; wait off, micro processor mode
mov.b   #02h,mcd                        ; f2
mov.b   #20h,cm1                        ;                                    (3)
mov.b   #08h,cm0                        ;
mov.b   #00001111b,ds                   ; data bus width 16bit
mov.b   #10101010b,wcr                  ; all 2wait
mov.b   #0,prcr                         ; Protect on
```

Figure 3.4.2 Initial setting

(1) Include file, and setting of rewrite program start address and version information

(1-1) Definition file includes the following two.

a. sfr80.inc: M16C/80 group SFR definition file

b. bl80.inc: files for RAM data declaration used in sample program and symbol definitions

(1-2) Setting of rewrite program start address and version information

When downloading a rewrite program using the bootloader, please locate the program from address 600_{16}. For the program size, please refer to the memory map of 3.6.4 When using MFW-1".

In the rewrite program downloaded with the bootloader, 8-bit version data should be set from address 600_{16}. Although you do not use the version data, it is still required to setup the data.

(2) Setting of stuck pointer

In the rewrite program, stack pointer (ISP) must be set up first. The setting value should be set in the internal RAM area not to overlap with the rewrite program. (The downloaded rewrite program is stored from address 600_{16}. Refer to 3.6 Memory Map.)

(3) Setting of the associated registers

• Changing of PM0: When the download of rewrite program is completed, the CPU starts the operation in single chip mode. And thus please change the mode to micro-processor mode.

• Setting of MCD and WCR: Set up main clock division and software wait according to the access timing with the external flash memory. (For the access timing of M16C/80 group, please refer to M16C/80 Data Sheet.)

• Setting of DS: Set up data bus width according to the connecting state of the external flash memory.

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+          Main flow  - clock synchronous serial I/O mode -            +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Main:
        jsr             Initialize_2            ; clock synchronous serial I/O mode   (4)
;
Loop_main:
        bset    ta0os
        mov.b   #0,ta0ic
Loop_main1:
        btst    ir_ta0ic                        ; 300 usec ?
        jz              Loop_main1
        mov.b   #0,ta0ic
        mov.b   #0ffh,r1l                       ; #ffh --> r1l (transfer dummy data)
        mov.b   r1l,u1tb                        ; transfer data --> transfer buffer

        bclr    busy_d                          ; busy input
?:
        btst    busy                            ; Reception start?
        jz              ?-

        bset    ta0os                           ; 300 usec timer start
?:
        btst    ir_ta0ic                        ; 300 usec ?
        jc              Time_out                ; jump Time_out at time out
        btst    ri_u1c1                         ; receive complete ?
        jz              ?-
        mov.w   u1rb,r0                         ; receive data --> r0
;
Command_check:                                                                (5)
        cmp.b   #0ffh,r0l                       ; Read          (ffh)
        jeq             Read
        cmp.b   #041h,r0l                       ; Program       (41h)
        jeq             Program
        cmp.b   #020h,r0l                       ; Erase         (20h)
        jeq             Erase
        cmp.b   #0a7h,r0l                       ; All erase (a7h)
        jeq             All_erase
        cmp.b   #050h,r0l                       ; Clear SRD (50h)
        jeq             Clear_SRD
        cmp.b   #071h,r0l                       ; Read LBS      (71h)
        jeq             Read_LB
        cmp.b   #077h,r0l                       ; LB program(77h)
        jeq             Program_LB
        cmp.b   #0fah,r0l                       ; Download      (fah)
        jeq             Download

        cmp.b   #070h,r0l                       ; Read SRD      (70h)
        jeq             Read_SRD
        cmp.b   #0fbh,r0l                       ; Version out   (fbh)
        jeq             Ver_output
Command_err:
        jsr             Initialize_21           ; command error,UART1 reset
        jmp             Loop_main               ; command error,jump Loop_main
```

Figure 3.4.3 Main routine

(4) Initial setting of communication

An initial setting of serial communication is done by subroutine jump to communication initial setting processing part.

(5) Command receive and decision process

Before starting command reception, wait 300 usec at 20 MHz first and then wait again until the BUSY pin (*1) becomes "H". After the BUSY pin turns to "H", perform the command reception. If a time-out occurs during the command reception, the processing jumps to the time-out error process. When receiving 1 byte command data without a time-out error occurred, the command check is performed successively and then the processing branches to a matched command.

*1: BUSY pin becomes "L" when the receiving preparation is completed and outputs "H" when the receiving operation starts.

```
;-----------------------------------------------------
;+          Read                                    +
;-----------------------------------------------------
Read:
        mov.w   #0,r3                       ; receive number
        mov.b   #0,addr_l                   ; addr_l = 0
;
Read_loop:
        mov.b   r1l,u1tb                    ; data transfer
        bset    ta0os                       ; ta0 start
?:
        btst    ir_ta0ic                    ; time out error ?
        jc              Time_out            ; jump Time_out at time out
        btst    ri_u1c1                     ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                     ; receive data read --> r0
;
        add.w   #1,r3                       ; r3 +1 increment
        cmp.w   #2,r3                       ; r3 = 2 ?
        jgtu            Read_data           ; jump Read_data at r3>3
        mov.w   r3,a0                       ; r3 --> a0
        mov.b   r0l,addr_l[a0]              ; Store address
        cmp.w   #2,r3                       ; r3 = 2 ?
        jltu            Read_loop           ; jump Read_loop at r3<2
;
        mov.w   addr_l,a0                   ; addr_l,m –> a0
        mov.b   addr_h,a1                   ; addr_h    --> a1
        sha.l   #16,a1
        add.l   a0,a1                       ; get read address
;
Read_data:
        ;
        ; Flash memory read & store to r1l
        ;
        add.l   #1,a1                       ; address increment
        cmp.w   #258,r3                     ; r3 = 258 ?
        jne             Read_loop           ; jump Read_loop at r<260
;
        jmp             Loop_main           ; jump Loop_main
;
```

(6_1)

Figure 3.4.4 Read command process

;

(6-1) Read command process (FF16)

This command is transmitted when any of blank, read, verify, and program/verify button of MFW-1 is pressed.

• Receive address information with the 2nd and 3rd bytes.

• Read out 1 byte data from the external flash memory and write it to r1l. (added by the user)

• Transmit the above read data to MFW-1.

• Repeat the data read-write-transfer operation 256 times.

```
;-----------------------------------------------------
;+          Program                                 +
;-----------------------------------------------------
Program:
        mov.w   #0,r3                       ; receive number
        mov.b   #0,addr_l                   ; addr_l = 0
Program_loop_1:
        mov.b   r1l,u1tb                    ; data transfer
        bset    ta0os                       ; ta0 start
        mov.b   #0,ta0ic                    ; clear time out
?:
        btst    ir_ta0ic                    ; time out error ?
        jc              Time_out            ; jump Time_out at time out
        btst    ri_u1c1                     ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                     ; receive data read --> r0
        add.w   #1,r3                       ; r3 +1 increment
        mov.w   r3,a0                       ; r3 --> a0
        mov.b   r0l,addr_l[a0]              ; Store address
        cmp.w   #258,r3                     ; r3 = 258 ?
        jltu            Program_loop_1      ; jump Program_loop_1 at r3<258
;
        mov.w   #0,r3                       ; writing number (r3=0)
Program_loop_2:
        mov.b   addr_h,a1                   ; addr_h   --> a1
        sha.l   #16,a1
        mov.w   r3,a0                       ; r3          –> a0
        mov.w   data[a0],r1                 ; data        –> r1
        mov.w   addr_l,a0                   ; addr_l,m –> a0
        add.l   a0,a1
        ;
        ; data write
        ;
        add.w   #2,addr_l                   ; address +2 increment
        add.w   #2,r3                       ; writing number +2 increment
        cmp.w   #255,r3                     ; r3 = 255 ?
        jltu            Program_loop_2      ; jump Program_loop_2 at r3<255
Program_end:
        jmp             Loop_main           ; jump Loop_main
;
```

(6_2)

Figure 3.4.5 program command process

(6-2) Program command process (4116)

This command is transmitted when either program or program/verify button of MFW-1 is pressed.

- Receive address information with the 2nd and 3rd bytes and successively receive the program data (256 bytes).
- Write the 256-byte data to the external flash memory. (added by the user)

Note: In the sample program, the increment of address (addr_1) and writing number (r3) is "+2" on the assumption that the data is written in word units.

```
;-----------------------------------------------------
;+          Block erase                          +
;-----------------------------------------------------
Erase:
        mov.w   #1,r3                       ; receive number (r3=1)
Erase_loop:
        mov.b   r1l,u1tb                    ; data transfer
        bset    ta0os                       ; ta0 start
        mov.b   #0,ta0ic                    ; clear time out
?:
        btst    ir_ta0ic                    ; time out error ?
        jc              Time_out            ; jump Time_out at time out
        btst    ri_u1c1                     ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                     ; receive data read --> r0
        mov.w   r3,a0                       ; r3 --> a0
        mov.b   r0l,addr_l[a0]              ; Store address
        add.w   #1,r3                       ; r3 +1 increment
        cmp.w   #4,r3                       ; r3=4 ?
        jltu            Erase_loop          ; jump Erase_loop at r3<4
;
        cmp.b   #0d0h,data                  ; Confirm command check
        jne             Erase_end           ; jump Erase_end at Confirm command error
        ;
        ; Block Erase
        ;
Erase_end:
        jmp             Loop_main           ; jump Loop_main
;
```

(6_3)

Figure 3.4.6 BlockErase command process

(6-3) Block erase command process (20_{16})

This command is transmitted when either erase or program button of MFW-1 is pressed. Note that this command is sent only when the erasing area is not all blocks. When the erasing area is all blocks, an All Erase command ($A7_{16}$), explained in (6-4) is sent.

- Receive address information with the 2nd and 3rd bytes and successively receive the verify command with the 4th byte.
- Check the verify command received with the 4th byte.
- Erase the data of the specified block in the external flash memory. (added by the user).

```
;-----------------------------------------------------
;+        All erase ( unlock block )          +
;-----------------------------------------------------
All_erase:
        mov.b   r1l,u1tb                        ; data transfer
        bset    ta0os                           ; ta0 start
        mov.b   #0,ta0ic                        ; clear time out
?:
        btst    ir_ta0ic                        ; time out error ?
        jc              Time_out                ; jump Time_out at time out
        btst    ri_u1c1                         ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                         ; receive data read --> r0
;
        cmp.b   #0d0h,r0l                       ; Confirm command check
        jne             All_erase_end           ; jump All_erase_end at Confirm command error
        ;
        ; All  Erase
        ;
All_erase_end:
        jmp             Loop_main               ; jump Loop_main
;
```

(6_4)

Figure 3.4.7 All Erase command process

(6-4) All erase command process (A716)

This command is transmitted when either erase or program button of MFW-1 is pressed. Note that the command is sent only when the erasing area is all blocks. When the erasing is not all blocks, an Block Erase command (2016), explained in (6-3) is sent.

• Receive the verify command with the 2th byte.

• Check the verify command received with the 2th byte.

• Erase the data of all blocks in the external flash memory. (added by the user).

```
;-----------------------------------------------------------
;+          Read SRD                          +
;-----------------------------------------------------------
Read_SRD:
        mov.w   #0,r3                           ; receive number (r3=0)

        mov.b   #80h,r1l                        ; dummy SRD set

;
Read_SRD_loop:
        mov.b   r1l,u1tb                        ; data transfer
        bset    ta0os                           ; ta0 start
        mov.b   #0,ta0ic                        ; clear time out
?:
        btst    ir_ta0ic                        ; time out error ?
        jc              Time_out                ; jump Time_out at time out
        btst    ri_u1c1                         ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                         ; receive data read --> r0
        mov.b   SRD1,r1l                        ; SRD1 data --> r1l
        add.w   #1,r3                           ; r3 +1 increment
        cmp.w   #2,r3                           ; r3=2 ?
        jltu            Read_SRD_loop           ; jump Read_SRD_loop at r3<2
;
        jmp             Loop_main               ; jump Loop_main
;
```

(6_5)

Figure 3.4.8 Read status command process

(6-5) Read status command process (70$_{16}$)

This command is used in communication control with MFW-1.

• Transfer "80$_{16}$" as SRD data with the 2th byte.

• Transfer SRD1 data with the 3rd byte.

```
;-----------------------------------------------------------
;+          Clear SRD                         +
;-----------------------------------------------------------
Clear_SRD:
;
        and.b   #10011100b,SRD1                 ; SRD1 clear
;
        jmp             Loop_main               ; jump Loop_main
;
```

(6_6)

Figure 3.4.9 Clear status command process

(6-6) Clear status command process (50$_{16}$)

This command is used in communication control with MFW-1.

• Clear SRD1 data.

```
;-----------------------------------------------------
;+          Read Lock Bit                    +
;-----------------------------------------------------
Read_LB:
        mov.w   #1,r3                       ; receive number (r3=1)
Read_LB_loop:
        mov.b   r1l,u1tb                    ; data transfer
        bset    ta0os                       ; ta0 start
        mov.b   #0,ta0ic                    ; clear time out
?:
        btst    ir_ta0ic                    ; time out error ?
        jc              Time_out            ; jump Time_out at time out
        btst    ri_u1c1                     ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                     ; receive data read --> r0
        mov.w   r3,a0                       ; r3 --> a0
        mov.b   r0l,addr_l[a0]              ; Store address
        add.w   #1,r3                       ; r3 +1 increment
        cmp.w   #3,r3                       ; r3=3 ?
        jltu            Read_LB_loop        ; jump Read_LB_loop at r3<3
        jgtu            Read_LB_end         ; jump Read_LB_end      at r3>3
;
        mov.w   #00aah,r1                   ; dummy read LB status set
;
        jmp             Read_LB_loop        ; jump    Read_LB_loop
;
Read_LB_end:
        jmp             Loop_main           ; jump Loop_main
;
```

(6_7)

Figure 3.4.10. Read lock bit command process

;

(6-7) Read lock bit command process (71$_{16}$)

This command is transmitted when either erase or program button of MFW-1 is pressed after a user selected "no change" with MFW-1.

• Receive address information with the 2nd and 3rd bytes.

• Transfer "AA$_{16}$" as lock bit data with the 4th byte.

```
;-----------------------------------------------------------
;+          Program Lock Bit                    +
;-----------------------------------------------------------
Program_LB:
        mov.w   #1,r3                           ; receive number (r3=1)
Program_LB_loop:
        mov.b   r1l,u1tb                        ; data transfer
        bset    ta0os                           ; ta0 start
        mov.b   #0,ta0ic                        ; clear time out
?:
        btst    ir_ta0ic                        ; time out error ?
        jc              Time_out                ; jump Time_out at time out
        btst    ri_u1c1                         ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                         ; receive data read --> r0
        mov.w   r3,a0                           ; r3 --> a0
        mov.b   r0l,addr_l[a0]                  ; Store address
        add.w   #1,r3                           ; r3 +1 increment
        cmp.w   #4,r3                           ; r3=4 ?
        jltu            Program_LB_loop         ; jump    Program_LB_loop at r3<4
        cmp.b   #0d0h,data                      ; Confirm command check
        jne             Program_LB_end          ;
;
Program_LB_end:
        jmp             Loop_main               ; jump Loop_main
;
```

(6_8)

Figure 3.4.11 Lock bit program command process

(6-8) Lock bit program command process (77$_{16}$)

This command is transmitted when either erase or program button of MFW-1 is pressed.

- Receive address information with the 2nd and 3rd bytes and successively receive the verify command with the 4th byte.
- Check the verify command (D0$_{16}$).

```
;-----------------------------------------------------------
;+          Version output                          +
;-----------------------------------------------------------
Ver_output:
        mov.w   #0,a0                       ; Version address offset (a0=0)
Ver_output_loop:
        mov.b   ver[a0],u1tb                ; Version data transfer
        bset    ta0os                       ; ta0 start
        mov.b   #0,ta0ic                    ; clear time out
?:
        btst    ir_ta0ic                    ; time out error ?
        jc              Time_out            ; jump Time_out at time out
        btst    ri_u1c1                     ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                     ; receive data read --> r0
        add.w   #1,a0                       ; a0 +1 increment
        cmp.w   #8,a0                       ; a0=8 ?
        jltu            Ver_output_loop     ; jump    Ver_output_loop at a0<8
Ver_output_end:
        jmp             Loop_main           ; jump Loop_main
;
```

(6_9)

Figure 3.4.12 Version output command process.

(6-9) Version output command process (FB16)

This command is used in communication control with MFW-1.

• Transfer version information with the 2nd to 9th bytes.

```
;-----------------------------------------------------------
;+          Download                                +
;-----------------------------------------------------------
Download:
        mov.b   #3,prcr                     ; Protect off
        mov.w   #0000h,pm0                  ; wait off, single chip mode
        mov.b   #02h,mcd                    ; f2
        mov.b   #20h,cm1                    ;
        mov.b   #08h,cm0                    ;
        mov.b   #0,prcr                     ; Protect on

        jmp.a   Download_program            ; jump Download_program
;
```

(6_10)

Figure 3.4.13 Download command process.

(6-10) Download command process (FA16)

When a user selects the download function, this command is sent on starting of communications with bootloader after MFW-1 startup.

• Change processor mode to single chip mode.

• Jump to the specified address (download processing area) of bootloader on the internal ROM of the microprocessor.

3.5. Examples of how to use bootloader mode 2

In bootloader mode 2, you can download a rewrite program using MAEC M16C Flash Starter. Here a rewrite program using M16C Flash Starter is explained. Table 3.5.1 shows commands when M16C Flash Starter is used. Figure 5.1 shows a flowchart of rewriting sample program with M16C Flash Starter used.

For the whole program, please refer to 3.8 program list.

Table 3.5.1 Software commands using M16C Flash Starter

No.	Control command	1st byte transfer	2nd byte	3rd byte	4th byte	5th byte	6th byte	
1	Page read	FF_{16}	Address (middle)	Address (high)	Data output	Data output	Data output	Data output to 259th byte
2	Page program	41_{16}	Address (middle)	Address (high)	Data input	Data input	Data input	Data input to 259th byte
3	~~Block erase~~	20_{16}	Address (middle)	Address (high)	$D0_{16}$			
4	Erase all unlocked blocks	$A7_{16}$	$D0_{16}$					
5	Read status register	70_{16}	SRD output	SRD1 output				
6	Clear status register	50_{16}						
7	~~Read lock bit status~~	71_{16}	Address (middle)	Address (high)	Lock bit data output			
8	~~Lock bit program~~	77_{16}	Address (middle)	Address (high)	$D0_{16}$			
9	~~Read check data~~	FD_{16}	Data output (low)	Data output (high)				
10	Download function	FA_{16}	Size (low)	Size (high)	Check-sum	Data input	To required number of times	
11	Download result output	FA_{16}	Data output					
12	Version data output function	FB_{16}	Version data output	Version data output	Version data output	Version data output	Version data output	Version data output to 9th byte
13	Baud rate 9600	$B0_{16}$	$B0_{16}$					
14	Baud rate 19200	$B1_{16}$	$B1_{16}$					
15	Baud rate 38400	$B2_{16}$	$B2_{16}$					
16	Baud rate 57600	$B3_{16}$	$B3_{16}$					
17	Baud rate 115200	$B4_{16}$	$B4_{16}$					

Note 1: Shading indicates transfer from microcontroller to serial programmer. All other data is transferred from the serial programmer to the microcontroller.

Note 2: SRD refers to status register data. SRD1 refers to status register 1 data.

Note 3: Commands No. 3 and No. 7 to No. 9 are unused with M16C Flash Starter.

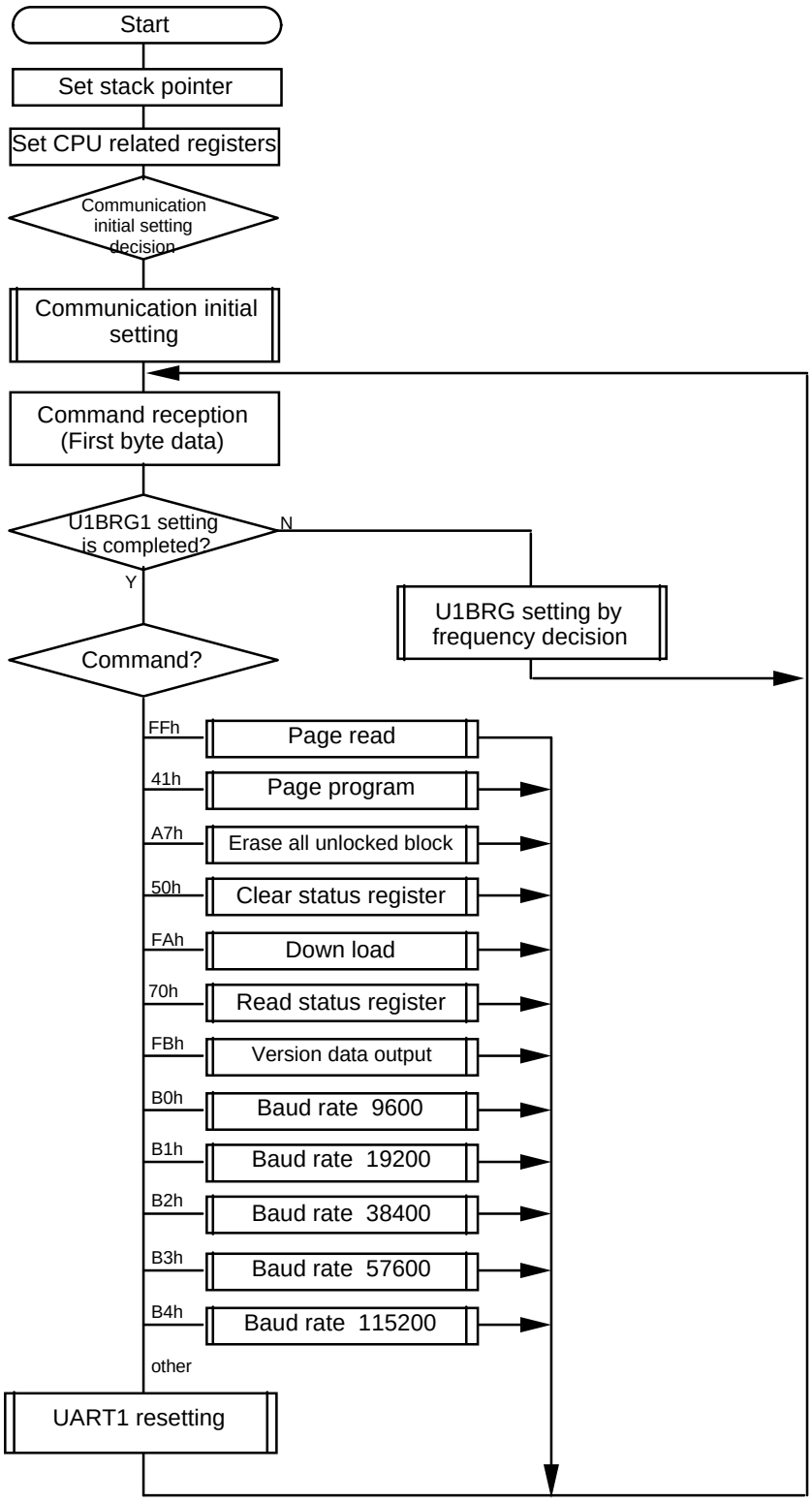


Figure 3.5.1 Flowchart of rewriting sample program with M16C Flash Starter used

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+        Include file                                                            +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
          .list          off
          .include  sfr800.inc                  ; SFR header include
          .include  bl80.inc                    ; Bootloader definition include
          .list          on
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+        Version table                                            +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
          .section  rom,code
          .org           0600h                  ; Download address
          .byte          'VER.0.01'             ; Version infomation
;
;==================================================================
;+        Boot program start                                       +
;==================================================================
Program_start:
;-----------------------------------------------
;+        Initialize_1                  +
;-----------------------------------------------
          ldc            #Istack,ISP            ; stack pointer set
;
;-----------------------------------------------------------
;+        Processor mode register          +
;+          & System clock control register  +
;-----------------------------------------------------------
CPU_set:
          mov.b    #3,prcr                     ; Protect off
          mov.w    #00000011b,pm0              ; wait off, micro processor mode
          mov.b    #02h,mcd                    ; f2
          mov.b    #20h,cm1                    ;
          mov.b    #08h,cm0                    ;
          mov.b    #00001111b,ds               ; data bus width 16bit
          mov.b    #10101010b,wcr              ; all 2wait
          mov.b    #0,prcr                     ; Protect on
```

(1-1)

(1-2)

(2)

(3)

Figure 3.5.2. Initial setting

(1) Include file, and setting of rewrite program start address and version information

(1-1) Definition file includes the following two.

a. sfr80.inc: M16C/80 group SFR definition file

b. bl80.inc: files for RAM data declaration used in sample program and symbol definitions

(1-2) Setting of rewrite program start address and version information

When downloading a rewrite program using the bootloader, please locate the program from address 600_{16}. For the program size, please refer to "3.6.3 When using M16C Flash Starter."

In the rewrite program downloaded with the bootloader, 8-bit version data should be set from address 600_{16}. Although you do not use the version data, it is still required to setup the data.

(2) Setting of stuck pointer

In the rewrite program, stack pointer (ISP) must be set up first. The setting value should be set in the internal RAM area not to overlap with the rewrite program. (The downloaded rewrite program is stored from address 600_{16}. Refer to 3.8 Memory Map.)

(3) Setting of the associated registers

- Changing of PM0: When the download of rewrite program is completed, the CPU starts the operation in single chip mode. And thus please change the mode to micro-processor mode.
- Setting of MCD and WCR: Set up main clock division and software wait according to the access timing with the external flash memory. (For the access timing of M16C/80 group, please refer to M16C/80 Data Sheet.)
- Setting of DS: Set up data bus width according to the connecting state of the external flash memory.

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+          Main flow  - UART mode -                        +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_Main:
        btst    updata_f                ;
        bmltu   updata_f                ; if "C"flag is "0", updata_f set "1"
        jc              U_Main1         ; if "C"flag is "1", initialize execute(jump U_Main1)
        jmp             U_Loop_main     ;                                                    (4)

U_Main1:
        bclr    updata_f                ;

        bclr    freq_set1               ; freq set flag clear
        bclr    freq_set2
        mov.b   #01111111b,data         ; Initialize Baud rate
        jsr             Initialize_3    ; UART mode Initialize                               (5)
        mov.b   #01000000b,r1l          ; counbter1,2 reset
        mov.b   #10000000b,r1h
        mov.w   u1rb,r0                 ; receive data --> r0
;
U_Loop_main:
        bclr    te_u1c1                 ; Transmission disabled
        bset    re_u1c1                 ; Reception enabled
?:
        btst    ri_u1c1                 ; receive complete ?
        jz              ?-
        mov.w   u1rb,r0                 ; receive data --> r0
        btst    freq_set2
        jz              U_Freq_check
;
U_Command_check:
        cmp.b   #0ffh,r0l               ; Read          (ffh)
        jeq             U_Read
        cmp.b   #041h,r0l               ; Program       (41h)
        jeq             U_Program
        cmp.b   #020h,r0l               ; Erase         (20h)
        jeq             U_Erase
        cmp.b   #0a7h,r0l               ; All erase     (a7h)
        jeq             U_All_erase
        cmp.b   #050h,r0l               ; Clear SRD     (50h)
        jeq             U_Clear_SRD
        cmp.b   #071h,r0l               ; Read LBS      (71h)
        jeq             U_Read_LB
        cmp.b   #077h,r0l               ; LB program    (77h)                                (6)
        jeq             U_Program_LB
        cmp.b   #0fah,r0l               ; Download      (fah)
        jeq             U_Download
        cmp.b   #0fdh,r0l               ; Read check    (fdh)
        jeq             U_Read_check

        cmp.b   #070h,r0l               ; Read SRD      (70h)
        jeq             U_Read_SRD
        cmp.b   #0fbh,r0l               ; Version out   (fbh)
        jeq             U_Ver_output
        cmp.b   #0b0h,r0l               ; Baud rate 9600bps (b0h)
        jeq             U_BPS_B0
        cmp.b   #0b1h,r0l               ; Baud rate 19200bps (b1h)
        jeq             U_BPS_B1
        cmp.b   #0b2h,r0l               ; Baud rate 38400bps (b2h)
        jeq             U_BPS_B2
        cmp.b   #0b3h,r0l               ; Baud rate 57600bps (b3h)
        jeq             U_BPS_B3
        cmp.b   #0b4h,r0l               ; Baud rate 115200bps (b4h)
        jeq             U_BPS_B4
        jsr             U_Initialize_31 ; command error, UART mode Initialize
        jmp             U_Loop_main     ; jump U_Loop_main
```

Figure 3.5.3 Main routine

(4) Communication initial setting decision process

If a reset is executed after the download of a rewrite program is completed, the processing branches to command decision processing part without branching to communication initial setting processing part.

(5) Initial setting of communication

This process will be executed when the system is reset after the download of a rewrite program is completed.

(6) Command decision process

After receiving 1 byte command data from M16C Flash Starter, the processing goes to judge if the setting of the bit rate generator has been completed. If not, then command check is performed. With the result of command check, the processing branches to a matched command.

```
;-----------------------------------------------------------
;+          Read - UART mode -              +
;-----------------------------------------------------------
U_Read:
        mov.w   #0,r3                       ; receive number
        mov.b   #0,addr_l                   ; addr_l = 0
?:
        btst    ri_u1c1                     ; receive complete ?
        jnc             ?-
;
        mov.w   u1rb,r0                     ; receive data read --> r0
        add.w   #1,r3                       ; r3 +1 increment
        mov.w   r3,a0                       ; r3 --> a0
        mov.b   r0l,addr_l[a0]              ; Store address
        cmp.w   #2,r3                       ; r3 = 2 ?
        jltu            ?-                  ; jump Read_loop at r3<2
;
        mov.w   addr_l,a0                   ; addr_l,m –> a0
        mov.b   addr_h,a1                   ; addr_h   --> a1
        sha.l   #16,a1                      ;
        add.l   a0,a1                       ; a1 is address-data
;
        bclr    re_u1c1                     ; Reception disabled
        bset    te_u1c1                     ; Transmission enabled
U_Read_data:
        cmp.w   #258,r3                     ; r3 = 258 ?
        jz              U_Read_end
        ;
        ; Flash memory read
        ;
        mov.b   r1l,u1tb                    ; r1l --> transmit buffer register
?:
        btst    ti_u1c1                     ; transmit buffer empty ?
        jnc             ?-
        add.l   #1,a1                       ; address increment
        add.w   #1,r3                       ; counter increment
        jmp             U_Read_data         ; jump U_Read_data
;
U_Read_end:
        btst    txept_u1c0                  ; Transmit register empty ?
        jnc             U_Read_end
        jmp             U_Loop_main
;
```

(7-1)

Figure 3.5.4 Read command process

(7-1) Read command process (FF16)

This command is transmitted when any of blank or read (B.P.R., E.P.R.) button of M16C Flash Starter is pressed.

- Receive address information with the 2nd and 3rd bytes.
- Read out 1 byte data from the external flash memory and write it to r1l. (added by the user)
- Transfer the above read data to M16C Flash Starter.
- Repeat the data read-write-transfer operation 256 times.

```
;-----------------------------------------------------
;+          Program - UART mode -          +
;-----------------------------------------------------
U_Program:
        mov.w   #0,r3                   ; receive number
        mov.b   #0,addr_l               ; addr_l = 0
        mov.w   sum,crcd                ; for Read check command
U_Program_loop:
        btst    ri_u1c1                 ; receive complete ?
        jnc             U_Program_loop
        mov.w   u1rb,r0                 ; receive data read --> r0
        add.w   #1,r3                   ; r3 + 1 increment
        mov.w   r3,a0                   ; r3 --> a0
        mov.b   r0l,addr_l[a0]          ; Store address
        cmp.w   #258,r3                 ; r3 = 258 ?
        jltu            U_Program_loop  ; jump U_Program_loop at r3<258
;
        mov.w   #0,r3                   ; writing number (r3=0)
U_Program_loop_2:
        mov.b   addr_h,a1               ; addr_h   --> a1
        sha.l   #16,a1
        mov.w   r3,a0                   ; r3          –> a0
        mov.w   data[a0],r1             ; data        –> r1
        mov.w   addr_l,a0               ; addr_l,m –> a0
        add.l   a0,a1
        ;
        ; data write
        ;
        mov.b   r1l,crcin               ; for Read check command
        mov.b   r1h,crcin
;
        add.w   #2,addr_l               ; address +2 increment
        add.w   #2,r3                   ; writing number + 2 increment
        cmp.w   #255,r3                 ; r3 = 255 ?
        jltu            U_Program_loop_2 ; jump U_Program_loop_2 at r3<255
U_Program_end:
        mov.w   crcd,sum                ; for Read check command
        jmp             U_Loop_main     ; jump U_Loop_main
;
```

(7-2)

Figure 3.5.5 Program command process

(7-2) Program command process (4116)

This command is transmitted when program (B.P.R., E.P.R.) button of M16C Flash Starter is pressed.

- Receive address information with the 2nd and 3rd bytes and successively receive the program data (256 bytes).
- Write the 256-byte data to the external flash memory. (added by the user)

Note: In the sample program, the increment of address (addr_1) and writing number (r3) is "+2" on the assumption that the data is written in word units.

```
;-----------------------------------------------------------------------------
;+                  All erase ( unlock block ) - UART mode -  +
;-----------------------------------------------------------------------------
U_All_erase:
        btst    ri_u1c1                         ; receive complete ?
        jnc             U_All_erase
;
        mov.w   u1rb,r0                         ; receive data read --> r0
        cmp.b   #0d0h,r0l                       ; Confirm command check
        jne             U_All_erase_end ; jump U_All_erase_end at Confirm command error
        ;
        ; All erase
        ;
U_All_erase_end:
        jmp             U_Loop_main             ; jump U_Loop_main
;
```

(7-3)

Figure 3.5.6 All Erase command process

(7-3) All erase command process (A716)

This command is transmitted when either erase (E.P.R.) button of M16C Flash Start is pressed.

- Receive the verify command with the 2nd byte.
- Check the verify command (D016) received with the 2nd byte.
- Erase the data of all blocks in the external flash memory. (added by the user).

```
;-----------------------------------------------------------
;+          Read SRD - UART mode          +
;-----------------------------------------------------------
U_Read_SRD:
        bclr    re_u1c1                     ; Reception disabled
;
        mov.w   #0,r3                       ; receive number (r3=0)
;
        mov.b   #80h,r1l                    ; dummy SRD set
        bset    te_u1c1                     ; Transmission enabled
U_Read_SRD_loop:
        mov.b   r1l,u1tb                    ; r1l --> transmit buffer register
?:
        btst    ti_u1c1                     ; transmit buffer empty ?
        jnc             ?-
        mov.b   SRD1,r1l                    ; SRD1 data --> r1l
        add.w   #1,r3                       ; r3 + 1 increment
        cmp.w   #2,r3                       ; r3=2 ?
        jltu            U_Read_SRD_loop     ; jump U_Read_SRD_loop at r3<2
U_Read_SRD_end:
        btst    txept_u1c0                  ; Transmit register empty ?
        jnc             U_Read_SRD_end
;
        jmp             U_Loop_main         ; jump U_Loop_main
;
```

(7-4)

Figure 3.5.7 Read status command process

(7-4) Read status command process (70$_{16}$)

This command is used in communication control with M16C Flash Starter.

- Transfer "80$_{16}$" as SRD data with the 2nd byte.
- Transfer SRD1 data with the 3rd byte.

```
;-----------------------------------------------------------
;+          Clear SRD - UART mode         +
;-----------------------------------------------------------
U_Clear_SRD:
        and.b   #10010000b,SRD1             ; SRD1 clear
;
        jmp             U_Loop_main         ; jump U_Loop_main
;
```

(7-5)

Figure 3.5.8 Clear status command process

(7-5) Clear status command process (50$_{16}$)

This command is used in communication control with M16C Flash Starter.

- Clear SRD1 data.

```
;-----------------------------------------------------------
;+          Version output - UART mode -      +
;-----------------------------------------------------------
U_Ver_output:
        mov.w   #0,a0                   ; Version address offset (a0=0)
        bclr    re_u1c1                 ; Reception disabled
        bset    te_u1c1                 ; Transmission enabled
U_Ver_loop:
        mov.b   ver[a0],u1tb            ; Version data transfer
?:
        btst    ti_u1c1                 ; transmit buffer empty ?
        jnc             ?-
        add.w   #1,a0                   ; a0 +1 increment
        cmp.w   #8,a0                   ; a0=8 ?
        jltu            U_Ver_loop      ; jump    U_Ver_loop at a0<8
U_Ver_end:
        btst    txept_u1c0              ; Transmit register empty ?
        jnc             U_Ver_end
        jmp             U_Loop_main     ; jump U_Loop_main
;
```

(7-6)

Figure 3.5.9 Version output command process

(7-6) Version output command process (FB16)

This command is used in communication control with M16C Flash Starter.

- Transfer version information with the 2nd to 9th bytes.

```
;-----------------------------------------------------
;+          Download - UART mode -      +
;-----------------------------------------------------
U_Download:
        mov.b   #3,prcr                 ; Protect off
        mov.w   #0000h,pm0              ; wait off, single chip mode
        mov.b   #02h,mcd                ; f2
        mov.b   #20h,cm1                ;
        mov.b   #08h,cm0                ;
        mov.b   #0,prcr                 ; Protect on

        jmp.a           U_Download_program      ; jump U_Download_program
;
```

(7-7)

Figure 3.5.10 Download command process

(7-7) Download command process (FA16)

This command is transmitted when download button of M16C Flash Starter is pressed.

- Change the processor mode into single chip mode.
- Jump to the specified address (download processing area) of bootloader on the internal ROM of the microprocessor.

```
;-----------------------------------------------------------
;+          Baud rate change - UART mode   +
;-----------------------------------------------------------
U_BPS_B0:
        mov.b   baud,data               ; Baud rate 9600bps
        jmp             U_BPS_SET
U_BPS_B1:
        mov.b   baud+1,data             ; Baud rate 19200bps
        jmp             U_BPS_SET
U_BPS_B2:
        mov.b   baud+2,data             ; Baud rate 38400bps
        jmp             U_BPS_SET
U_BPS_B3:
        mov.b   baud+3,data             ; Baud rate 57600bps
        jmp             U_BPS_SET
U_BPS_B4:
        mov.b   baud+4,data             ; Baud rate 115200bps
U_BPS_SET:
        bclr    re_u1c1                 ; Reception disabled
        bset    te_u1c1                 ; Transmission enabled
        mov.b   r0l,u1tb                ; r0l --> transmit buffer register
?:
        btst    ti_u1c1                 ; transmit buffer empty ?
        jnc             ?-
?:
        btst    txept_u1c0
        jnc             ?-
        bclr    te_u1c1                 ; Transmission    disabled
        jsr             U_blank_end     ; UART mode Initialize
        jmp             U_Loop_main     ; jump U_Loo_main
;
```

(7-8)

Figure 3.5.11. Baud rate change command process

(7-8) Baud rate change command process (B016, B116, B216, B316, B416)

This command is used in communication control with M16C Flash Starter.

• Create baud rate change data.

• Transmit the data of the same contents as the 1st byte with the 2nd byte.

• Change baud rate (UART re-initialization).

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+          Freq check - UART mode -                        +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_Freq_check:
        bclr    re_u1c1                 ; Reception disabled
        btst    0,r1h                   ; counter = 8 times
        jc              U_Freq_check_4
;
        btst    freq_set1
        jc              U_Freq_check_1
        btst    5,r0h                   ; fer_u1rb
        jz              U_Freq_check_3
        jmp             U_Freq_check_2
U_Freq_check_1:
        cmp.b   #00h,r0l                ; "00h"?
        jeq             U_Freq_check_3
U_Freq_check_2:
        or.b    r1h,r1l                 ; r1l = counter1 or counter2
U_Freq_check_3:
        xor.b   data,r1l                ; Baud = Baud xor r1l
        mov.b   r1l,data                ; data set
        mov.b   r1h,r1l
        rot.b   #-1,r1l
        rot.b   #-1,r1h                 ; counter sift
        rot.b   #-1,r1l
        jmp             U_Freq_check_6
;
U_Freq_check_4:
        btst    freq_set1               ; Baud get ?
        jc              U_Freq_set_1    ; Yes , finished
        bset    freq_set1
        btst    5,r0l                   ; fer_u1rb
        jz              U_Freq_check_5
        xor.b   data,r1h
        mov.b   r1h,data
U_Freq_check_5:
        mov.b   data,data+1             ; Min Baud --> data+1
        mov.b   #01000000b,r1l          ; counter reset
        mov.b   #10000000b,r1h
        mov.b   #10000000b,data         ; Reset
U_Freq_check_6:
        jsr             U_blank_end     ; UART mode Initialize
?:
        btst    p6_6
        jz              ?-
        jmp             U_Loop_main
;
U_Freq_set_1:
        cmp.b   #00h,r0l                ; "00h"?
        jeq             U_Freq_set_2
        xor.b   data,r1h
        mov.b   r1h,data
U_Freq_set_2:
        bset    freq_set2
        mov.b   data,r1l                ; Max Baud --> data
        sub.b   data+1,r1l
        shl.b   #-1,r1l
        add.b   data+1,r1l
;
        mov.b   r1l,baud                ; 9600bps
        shl.b   #-1,r1l                 ; 19200bps
        mov.b   r1l,baud+1
        shl.b   #-1,r1l                 ; 38400bps
        mov.b   r1l,baud+2
        mov.b   baud,r0l                ; 57600bps
        mov.b   #0,r0h
        divu.b  #6
        mov.b   r0l,baud+3
        mov.b   baud+3,r0l              ; 115200bps
        shl.b   #-1,r0l
        mov.b   r0l,baud+4
        mov.b   baud,data
        mov.b   #0b0h,r0l               ; "B0h" set
        jsr             U_blank_end     ; UART mode Initialize
        jmp             U_BPS_SET
;
```

(8)

Figure 3.5.12 Bit rate generator setting process

(8) Bit rate generator setting process

In the aforesaid (6) command decision process, if the setting complete flag (freq_set2) is set to uncompleted ("0") in the bit rate generator setting completion check process, the processing branches to this command. In this process (8), the bit rate generator is adjusted to match the main clock input oscillation frequency (2 MHz to 20 MHz) by receiving "00$_{16}$" at 9600 bps from M16C Flash Starter 16 times. The highest speed is taken from the first 8 transmissions and the lowest from the last 8. These values are then used to calculate the bit rate generator value for a baud rate of 9,600 bps.

(9) UART1 initialize process

UART1 associated registers are initialized in this process. This processing part is called from (5) initial setting of communication, (7-8) baud rate change command process and (8) bit rate generator setting process.

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+          Subroutine : Initialize_3 - UART mode                +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Initialize_3:
U_blank_end:
;
;----------------------------------------
;+          UART1              +
;----------------------------------------
;---- UART nit rate generator 1
;
        mov.w   data,u1brg
;
U_Initialize_31:
;
;---- Function select register B0
;
        mov.b   #00000000b,psl0
;
;---- Function select register A0
;
        mov.b   #10010000b,ps0                  ; When you hope busy output OFF, set
"#10000000b"
;
;---- UART1 transmit/receive mode register
;
        mov.b   #0,u1c1                         ; transmit/receive disable
        mov.b   #0,u1mr                         ; u1mr reset
        mov.b   #00000101b,u1mr
;                ||||||++---------- transfer data 8 bit long
;                |||||+------------ Internal clock
;                ||||+------------- one stop bit
;                ||++-------------- parity disabled
;                |+---------------- sleep mode deselected
;
;---- UART1 transmit/receive control register 0
;
        mov.b   #00000100b,u1c0
;                ||||||++---------- f1 select
;                ||||++------------ RTS select
;                |||+-------------- CRT/RTS enabled
;                ||+--------------- CMOS output(TxD)
;                ++---------------- Must always be "0"
;
;---- UART transmit/receive control register 2
;
        mov.b   #00000000b,ucon
;                ||||||++---------- Transmit buffer empty
;                |||+++------------ Invalid
;                ||+--------------- Must always be "0"
;                |+---------------- CTS/RTS shared
;                +----------------- fixed
;
;---- UART1 transmit/received control register 1
;
        mov.b   #00000000b,u1c1
;                ||||||+---------- Transmission disabled
;                |||||+----------- Transmission enabled
;                ||||+------------ Reception disabled
;                |||+------------- Reception enabled
;                ++++-------------- fixed
;
        rts
;
```

(9)

Figure 3.5.13 UART1 initialize process

3.6 Memory map

3.6.1 RAM=10K

(M30800SFP-BL, M30800SGP-BL, M30802SGP-BL)

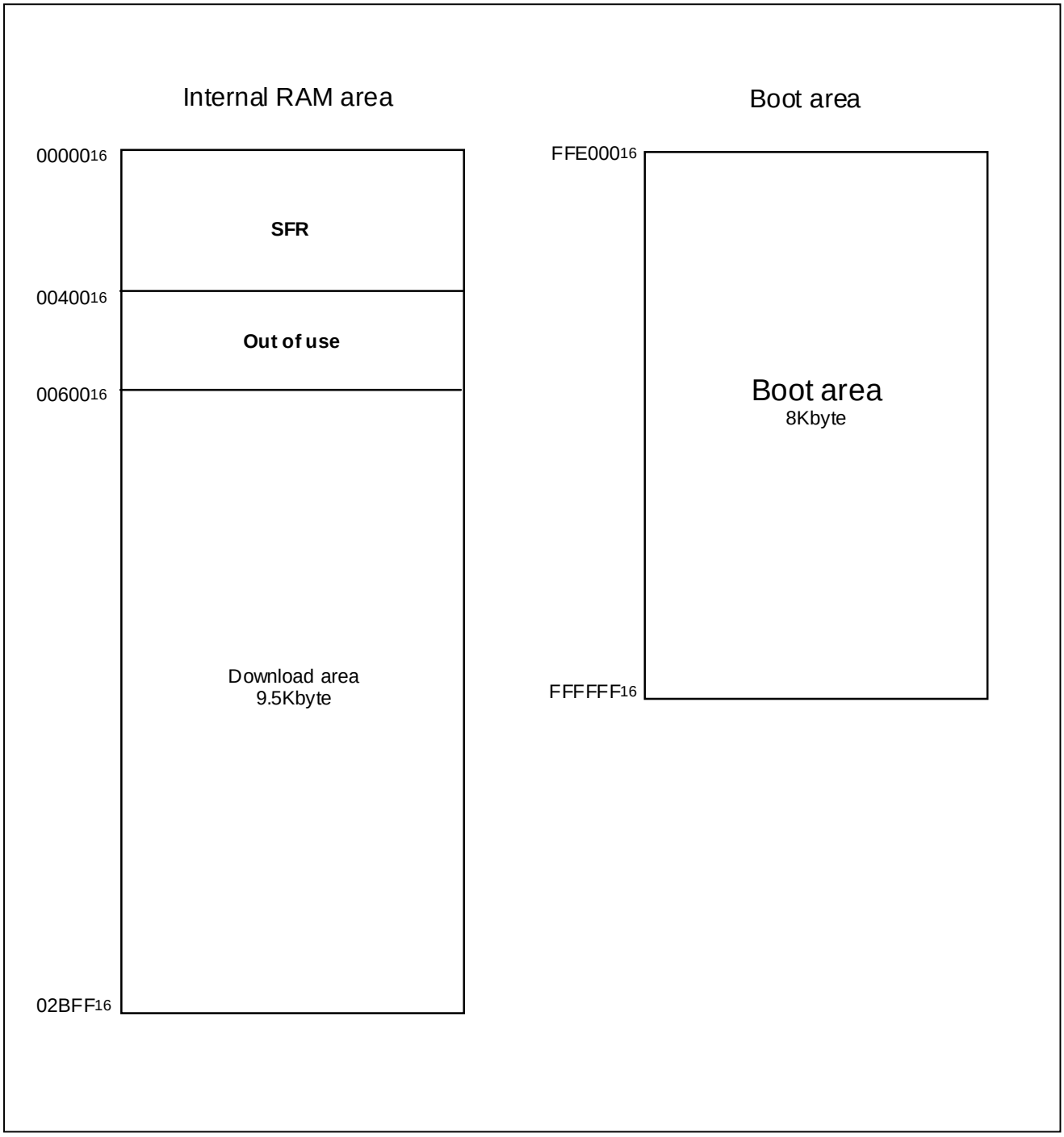


Figure 3.6.1 Memory map (RAM=10K)

3.6.2 RAM=24K

M30803SFP-BL, M30803SGP-BL, M30805SGP-BL

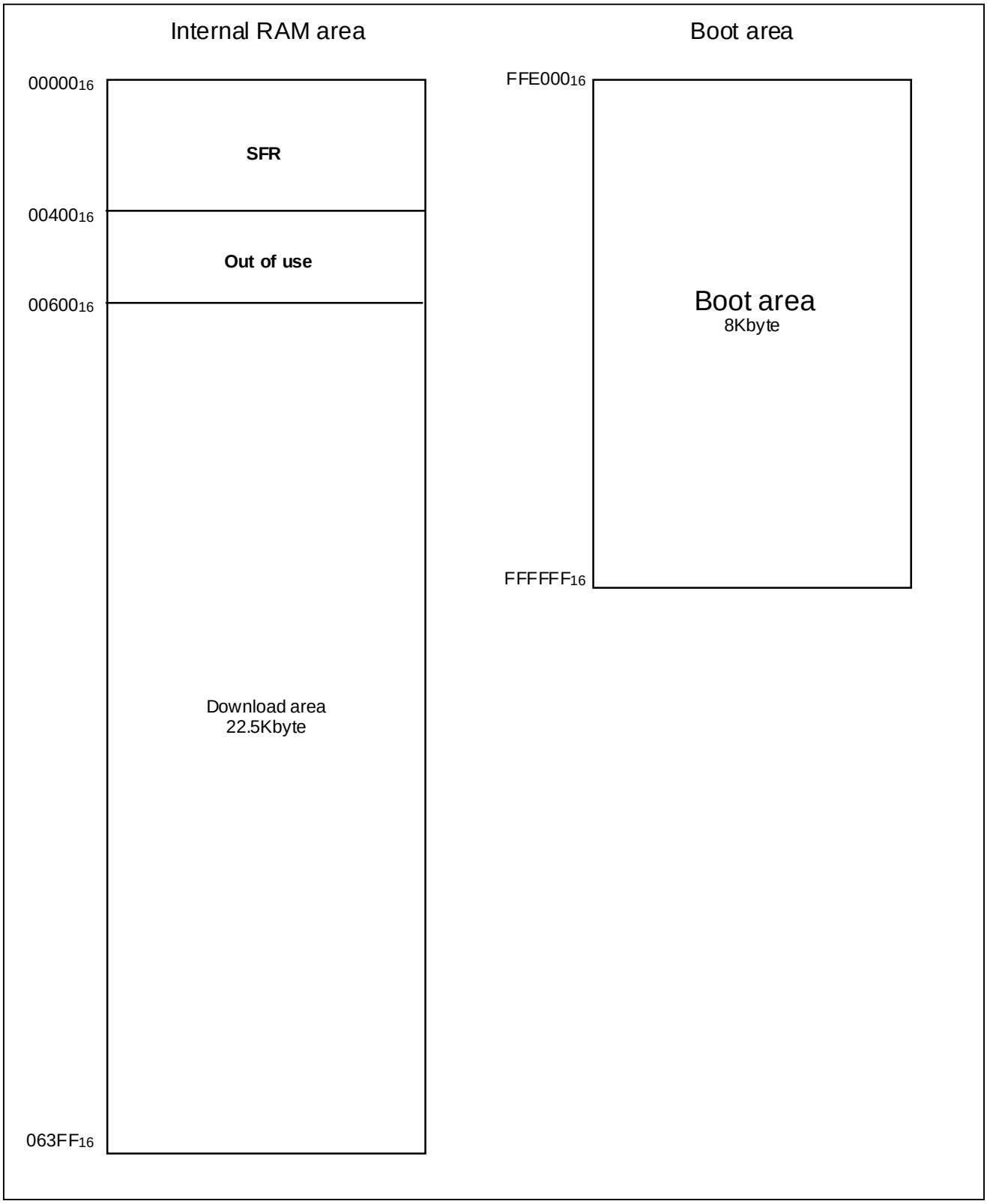


Figure 3.6.2 Memory map (RAM=24K)

3.6.3 When using M16C Flash Starter

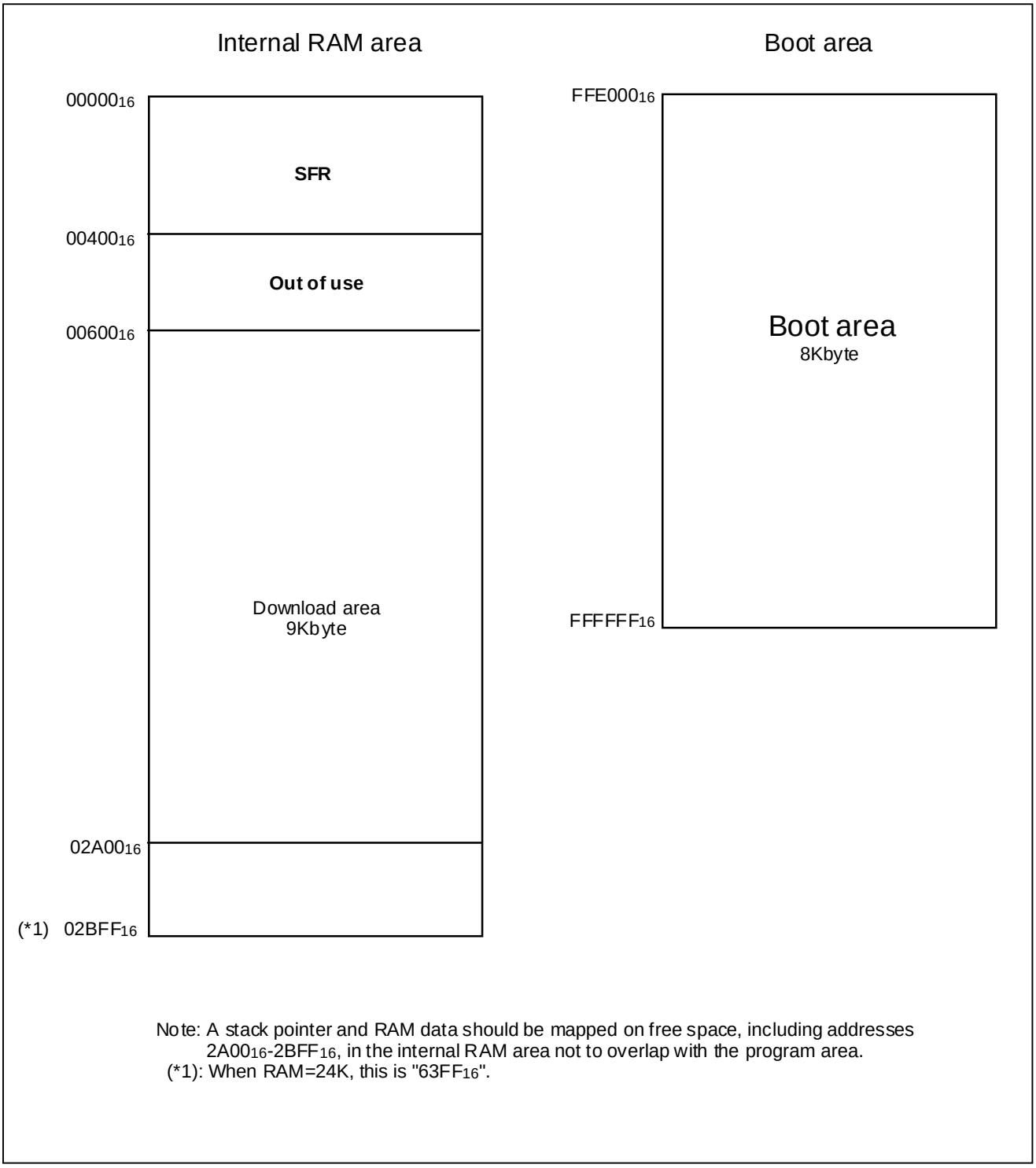
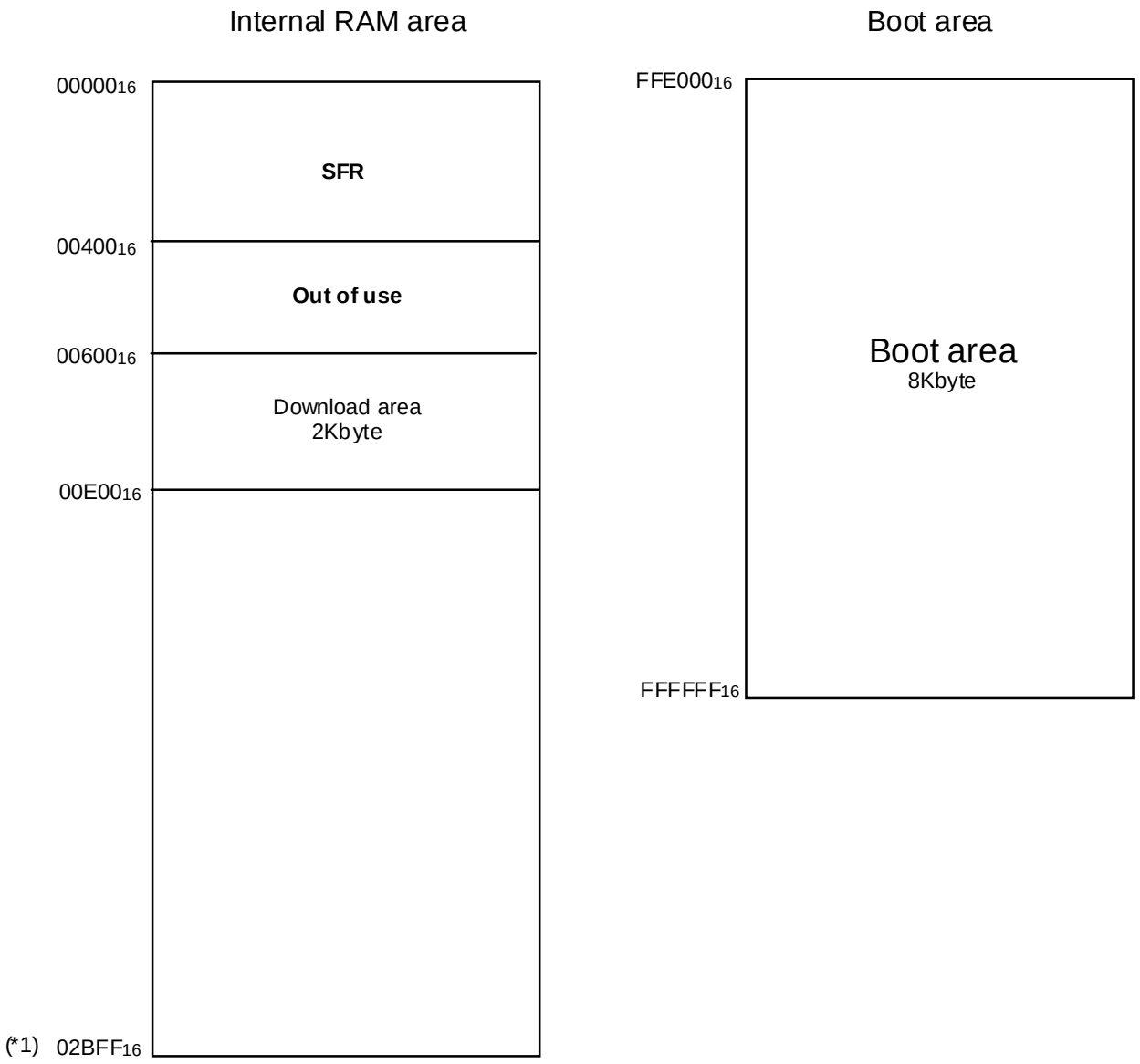


Figure 3.6.3 Memory map (when using M16C Flash Starter)

3.6.4 When using MFW-1



Note: A stack pointer and RAM data should be mapped on free space, including addresses $00E00_{16}$-$02BFF_{16}$, in the internal RAM area not to overlap with the program area.
(*1): When RAM=24K, this is "$63FF_{16}$".

Figure 3.6.4 Memory map (when using MFW-1)

3.7 Connection example of bootloader

3.7.1 Bootloader Mode 1

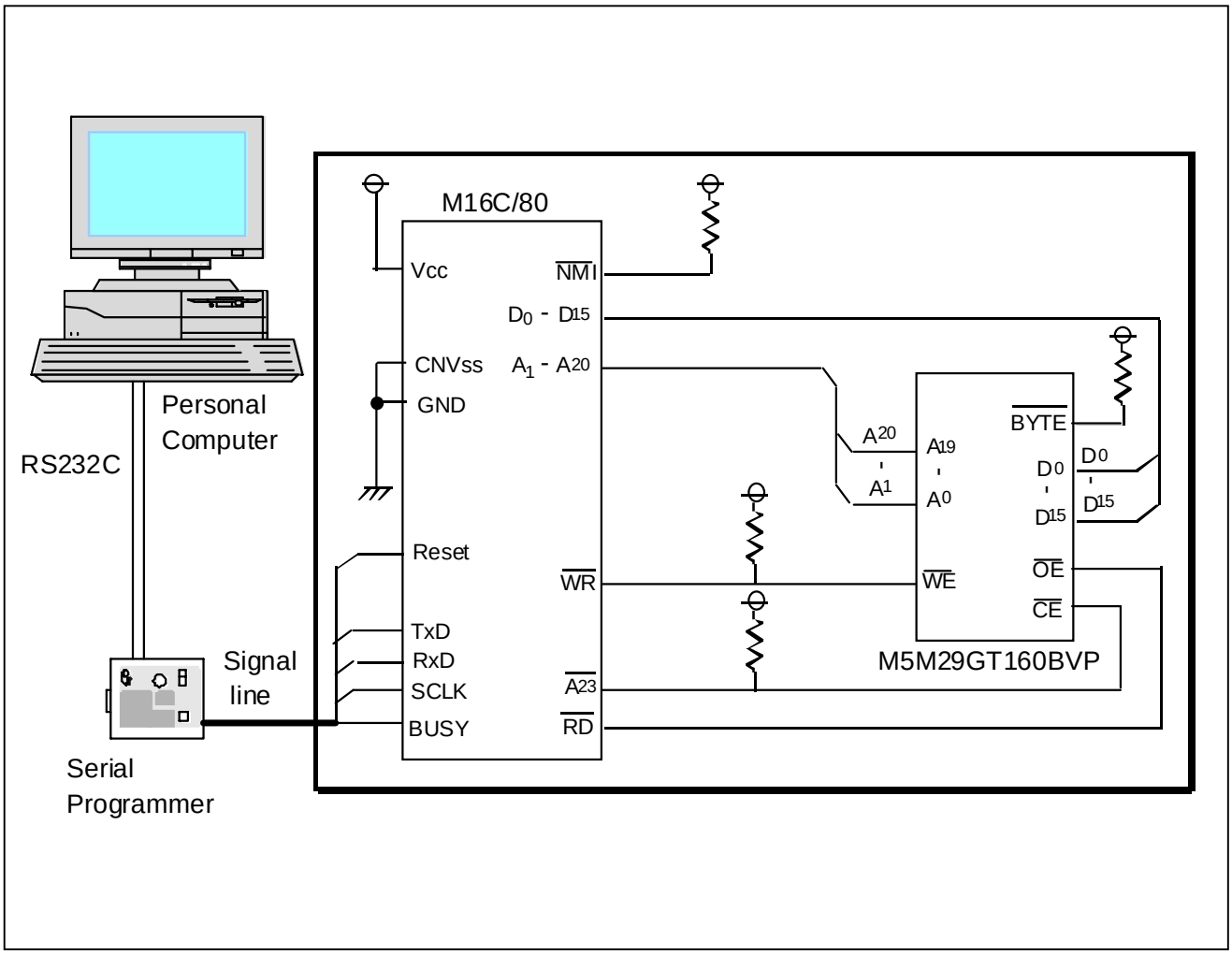


Figure 3.7.1 Connection example of bootloader mode 1

3.7.2 Bootloader Mode 2

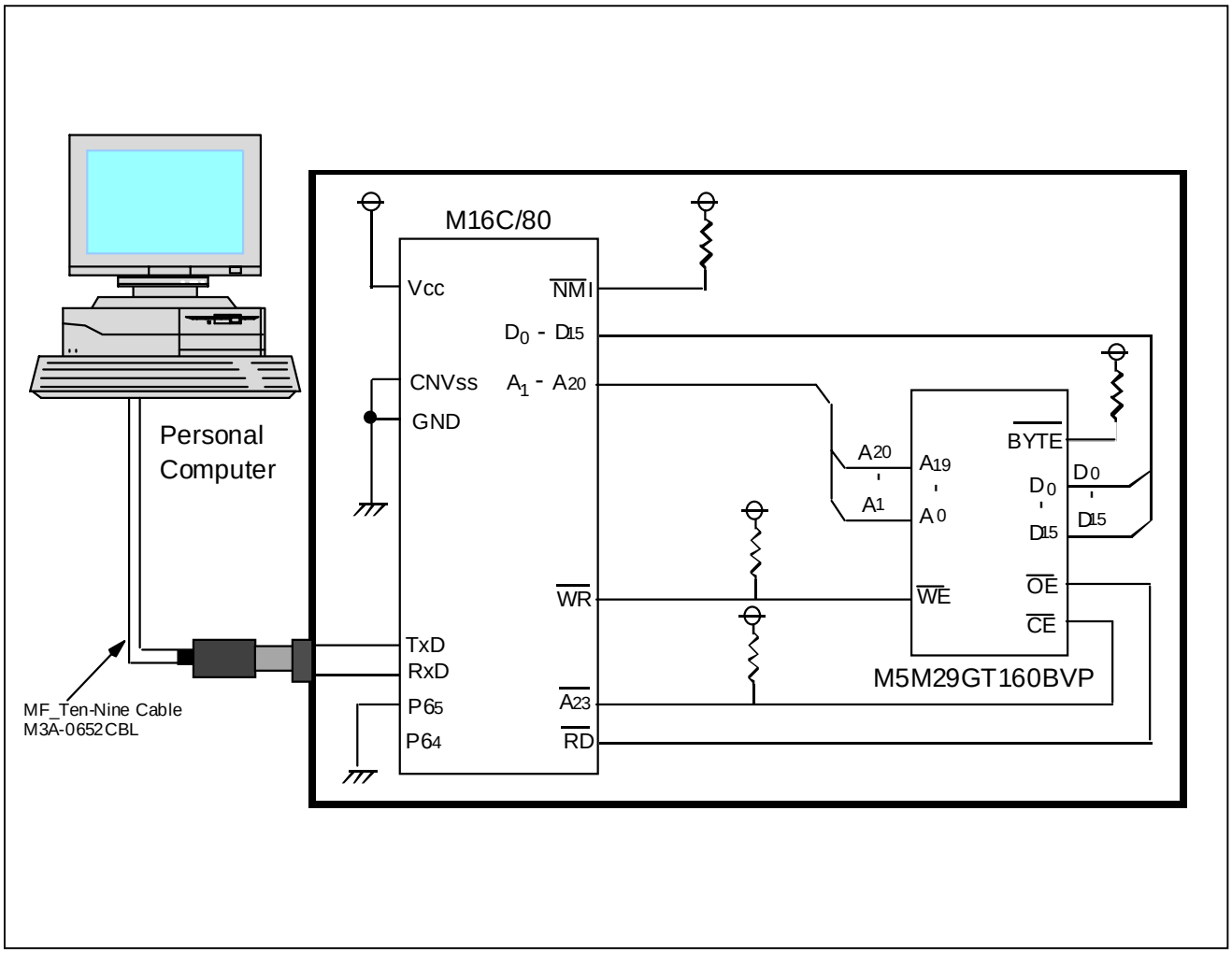


Figure 3.7.2 Connection example of bootloader mode 2

3.8 Program list

3.8.1 Sample program when using MFW-1

```
;*****************************************************************************
;*     System Name    : Rewrite Program for M16C/80 BootLoader                *
;*     File Name      : sample_Sync.a30                                       *
;*     MCU            : M3080xSGP-BL                                          *
;*     Xin            : 2M-20MHz (for Sync mode )                             *
;*---------------------------------------------------------------------------*
;*      Copyright,2001 MITSUBISHI ELECTRIC CORPORATION                        *
;*            AND MITSUBISHI SEMICONDUCTOR SYSTEM CORPORATION                 *
;*****************************************************************************
;
;
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+        Include file                                                      +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
       .list         off
       .include      sfr80.inc            ; SFR header include
       .include      bl80.inc             ; Bootloader definition include
       .list         on
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+    Version table                                           +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
       .section      rom,code
       .org          0600h                ; Download address
       .byte         'VER.1.01'           ; Version infomation
;
;==============================================================
;+        Boot program start                                  +
;==============================================================
Program_start:
;-----------------------------------------------
;+        Initialize_1                         +
;-----------------------------------------------
       ldc           #Istack,ISP          ; stack pointer set
;
;-----------------------------------------------
;+     Processor mode register                 +
;+       & System clock control register       +
;-----------------------------------------------
CPU_set:
       mov.b  #3,prcr                      ; Protect off
       mov.w  #00000011b,pm0               ; wait off, micro processor mode
       mov.b  #02h,mcd                     ; f2
       mov.b  #20h,cm1                     ;
       mov.b  #08h,cm0                     ;
       mov.b  #00001111b,ds                ; data bus width 16bit
       mov.b  #10101010b,wcr               ; all 2wait
       mov.b  #0,prcr                      ; Protect on
```

```
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+        Main flow  - clock synchronous serial I/O mode -      +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Main:
       jsr           Initialize_2         ; clock synchronous serial I/O mode
;
Loop_main:
       bset   ta0os
       mov.b  #0,ta0ic
Loop_main1:
       btst   ir_ta0ic                    ; 300 usec ?
       jz            Loop_main1
       mov.b  #0,ta0ic
       mov.b  #0ffh,r1l                   ; #ffh --> r1l (transfer dummy data)
       mov.b  r1l,u1tb                    ; transfer data --> transfer buffer

       bclr   4,pd6                       ; busy input
?:
       btst   4,p6                        ; Reception start?
       jz            ?-

       bset   ta0os                       ; 300 usec timer start
?:
       btst   ir_ta0ic                    ; 300 usec ?
       jc            Time_out             ; jump Time_out at time out
       btst   ri_u1c1                     ; receive complete ?
       jz            ?-
       mov.w  u1rb,r0                     ; receive data --> r0
;
Command_check:
       cmp.b  #0ffh,r0l                   ; Read        (ffh)
       jeq           Read
       cmp.b  #041h,r0l                   ; Program     (41h)
       jeq           Program
       cmp.b  #020h,r0l                   ; Erase       (20h)
       jeq           Erase
       cmp.b  #0a7h,r0l                   ; All erase   (a7h)
       jeq           All_erase
       cmp.b  #050h,r0l                   ; Clear SRD   (50h)
       jeq           Clear_SRD
       cmp.b  #071h,r0l                   ; Read LBS    (71h)
       jeq           Read_LB
       cmp.b  #077h,r0l                   ; LB program  (77h)
       jeq           Program_LB
       cmp.b  #0fah,r0l                   ; Download    (fah)
       jeq           Download

       cmp.b  #070h,r0l                   ; Read SRD    (70h)
       jeq           Read_SRD
       cmp.b  #0fbh,r0l                   ; Version out (fbh)
       jeq           Ver_output
Command_err:
       jsr           Initialize_21        ; command error,UART1 reset
       jmp           Loop_main            ; command error,jump Loop_main


;
```

```
;-------------------------------------------------------
;+        Read                                          +
;-------------------------------------------------------
Read:
        mov.w   #0,r3                       ; receive number
        mov.b   #0,addr_l                   ; addr_l = 0
;
Read_loop:
        mov.b   r1l,u1tb                    ; data transfer
        bset    ta0os                       ; ta0 start
        mov.b   #0,ta0ic                    ; clear time out
?:
        btst    ir_ta0ic                    ; time out error ?
        jc              Time_out            ; jump Time_out at time out
        btst    ri_u1c1                     ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                     ; receive data read --> r0
;
        add.w   #1,r3                       ; r3 +1 increment
        cmp.w   #2,r3                       ; r3 = 2 ?
        jgtu    Read_data                   ; jump Read_data at r3>3
        mov.w   r3,a0                       ; r3 --> a0
        mov.b   r0l,addr_l[a0]              ; Store address
        cmp.w   #2,r3                       ; r3 = 2 ?
        jltu    Read_loop                   ; jump Read_loop at r3<2
;
        mov.w   addr_l,a0                   ; addr_l,m --> a0
        mov.b   addr_h,a1                   ; addr_h   --> a1
        sha.l   #16,a1                      ;
        add.l   a0,a1                       ; a1 is address-data
;
Read_data:
        ;
        ; Flash memory read & sotre to r1l
        ;
        add.l   #1,a1                       ; address increment
        cmp.w   #258,r3                     ; r3 = 258 ?
        jne             Read_loop           ; jump Read_loop at r<260
;
        jmp             Loop_main           ; jump Loop_main
;
```

```
;-------------------------------------------------------------
;+        Program                                            +
;-------------------------------------------------------------
Program:
        mov.w   #0,r3                       ; receive number
        mov.b   #0,addr_l                   ; addr_l = 0
Program_loop_1:
        mov.b   r1l,u1tb                    ; data transfer
        bset    ta0os                       ; ta0 start
        mov.b   #0,ta0ic                    ; clear time out
?:
        btst    ir_ta0ic                    ; time out error ?
        jc              Time_out            ; jump Time_out at time out
        btst    ri_u1c1                     ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                     ; receive data read --> r0
        add.w   #1,r3                       ; r3 +1 increment
        mov.w   r3,a0                       ; r3 --> a0
        mov.b   r0l,addr_l[a0]              ; Store address
        cmp.w   #258,r3                     ; r3 = 258 ?
        jltu    Program_loop_1              ; jump Program_loop_1 at r3<258
;
        mov.w   #0,r3                       ; writing number (r3=0)
Program_loop_2:
        mov.b   addr_h,a1                   ; addr_h   --> a1
        sha.l   #16,a1
        mov.w   r3,a0                       ; r3      --> a0
        mov.w   data[a0],r1                 ; data    --> r1
        mov.w   addr_l,a0                   ; addr_l,m --> a0
        add.l   a0,a1
        ;
        ; data write
        ;
        add.w   #2,addr_l                   ; address +2 increment
        add.w   #2,r3                       ; writing number +2 increment
        cmp.w   #255,r3                     ; r3 = 255 ?
        jltu    Program_loop_2              ; jump Program_loop_2 at r3<255
Program_end:
        jmp             Loop_main           ; jump Loop_main
;
```

```
;-------------------------------------------------------------
;+        Block erase                                        +
;-------------------------------------------------------------
Erase:
        mov.w   #1,r3                       ; receive number (r3=1)
Erase_loop:
        mov.b   r1l,u1tb                    ; data transfer
        bset    ta0os                       ; ta0 start
        mov.b   #0,ta0ic                    ; clear time out
?:
        btst    ir_ta0ic                    ; time out error ?
        jc              Time_out            ; jump Time_out at time out
        btst    ri_u1c1                     ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                     ; receive data read --> r0
        mov.w   r3,a0                       ; r3 --> a0
        mov.b   r0l,addr_l[a0]              ; Store address
        add.w   #1,r3                       ; r3 +1 increment
        cmp.w   #4,r3                       ; r3=4 ?
        jltu    Erase_loop                  ; jump Erase_loop at r3<4
;
        cmp.b   #0d0h,data                  ; Confirm command check
        jne             Erase_end           ; jump Erase_end at Confirm command error
        ;
        ; Block Erase
        ;
Erase_end:
        jmp             Loop_main           ; jump Loop_main
;
;-------------------------------------------------------------
;+              All erase ( unlock block )                   +
;-------------------------------------------------------------
All_erase:
        mov.b   r1l,u1tb                    ; data transfer
        bset    ta0os                       ; ta0 start
        mov.b   #0,ta0ic                    ; clear time out
?:
        btst    ir_ta0ic                    ; time out error ?
        jc              Time_out            ; jump Time_out at time out
        btst    ri_u1c1                     ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                     ; receive data read --> r0
;
        cmp.b   #0d0h,r0l                   ; Confirm command check
        jne             All_erase_end       ; jump All_erase_end at Confirm command
error
;
        ;
        ; All Erase
        ;
All_erase_end:
        jmp             Loop_main           ; jump Loop_main
;
```

```
;-------------------------------------------------------------
;+        Read SRD                                           +
;-------------------------------------------------------------
Read_SRD:
        mov.w   #0,r3                       ; receive number (r3=0)

        mov.b   #80h,r1l                    ; dummy SRD set

;
Read_SRD_loop:
        mov.b   r1l,u1tb                    ; data transfer
        bset    ta0os                       ; ta0 start
        mov.b   #0,ta0ic                    ; clear time out
?:
        btst    ir_ta0ic                    ; time out error ?
        jc              Time_out            ; jump Time_out at time out
        btst    ri_u1c1                     ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                     ; receive data read --> r0
        mov.b   SRD1,r1l                    ; SRD1 data --> r1l
        add.w   #1,r3                       ; r3 +1 increment
        cmp.w   #2,r3                       ; r3=2 ?
        jltu    Read_SRD_loop               ; jump Read_SRD_loop at r3<2
;
        jmp             Loop_main           ; jump Loop_main
;
;-------------------------------------------------------------
;+        Clear SRD                                          +
;-------------------------------------------------------------
Clear_SRD:
;
        and.b   #10011100b,SRD1             ; SRD1 clear
;
        jmp             Loop_main           ; jump Loop_main
;
;-------------------------------------------------------------
;+        Read Lock Bit                                      +
;-------------------------------------------------------------
Read_LB:
        mov.w   #1,r3                       ; receive number (r3=1)
Read_LB_loop:
        mov.b   r1l,u1tb                    ; data transfer
        bset    ta0os                       ; ta0 start
        mov.b   #0,ta0ic                    ; clear time out
?:
        btst    ir_ta0ic                    ; time out error ?
        jc              Time_out            ; jump Time_out at time out
        btst    ri_u1c1                     ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                     ; receive data read --> r0
        mov.w   r3,a0                       ; r3 --> a0
        mov.b   r0l,addr_l[a0]              ; Store address
        add.w   #1,r3                       ; r3 +1 increment
        cmp.w   #3,r3                       ; r3=3 ?
        jltu    Read_LB_loop                ; jump Read_LB_loop at r3<3
        jgtu    Read_LB_end                 ; jump Read_LB_end   at r3>3
;
        mov.w   #00aah,r1                   ; dummy read LB status set
```

```
;
        jmp             Read_LB_loop            ; jump Read_LB_loop
;
Read_LB_end:
        jmp             Loop_main               ; jump Loop_main
;
;-------------------------------------------------------------
;+        Program Lock Bit                                   +
;-------------------------------------------------------------
Program_LB:
        mov.w   #1,r3                           ; receive number (r3=1)
Program_LB_loop:
        mov.b   r1l,u1tb                        ; data transfer
        bset    ta0os                           ; ta0 start
        mov.b   #0,ta0ic                        ; clear time out
?:
        btst    ir_ta0ic                        ; time out error ?
        jc              Time_out                ; jump Time_out at time out
        btst    ri_u1c1                         ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                         ; receive data read --> r0
        mov.w   r3,a0                           ; r3 --> a0
        mov.b   r0l,addr_l[a0]                  ; Store address
        add.w   #1,r3                           ; r3 +1 increment
        cmp.w   #4,r3                           ; r3=4 ?
        jltu    Program_LB_loop                 ; jump Program_LB_loop at r3<4
        cmp.b   #0d0h,data                      ; Confirm command check
        jne             Program_LB_end          ; jump Program_LB_end at Confirm command
error
;
Program_LB_end:
        jmp             Loop_main               ; jump Loop_main
;
;-------------------------------------------------------------
;+        Version output                                     +
;-------------------------------------------------------------
Ver_output:
        mov.w   #0,a0                           ; Version address offset (a0=0)
Ver_output_loop:
        mov.b   ver[a0],u1tb                    ; Version data transfer
        bset    ta0os                           ; ta0 start
        mov.b   #0,ta0ic                        ; clear time out
?:
        btst    ir_ta0ic                        ; time out error ?
        jc              Time_out                ; jump Time_out at time out
        btst    ri_u1c1                         ; receive complete ?
        jnc             ?-
        mov.w   u1rb,r0                         ; receive data read --> r0
        add.w   #1,a0                           ; a0 +1 increment
        cmp.w   #8,a0                           ; a0=8 ?
        jltu    Ver_output_loop                 ; jump Ver_output_loop at a0<8
Ver_output_end:
        jmp             Loop_main               ; jump Loop_main
;
```

```
;-----------------------------------------------------------
;+        Download                                         +
;-----------------------------------------------------------
Download:
        mov.b   #3,prcr                     ; Protect off
        mov.w   #0000h,pm0                  ; wait off, single chip mode
        mov.b   #02h,mcd                    ; f2
        mov.b   #20h,cm1                    ;
        mov.b   #08h,cm0                    ;
        mov.b   #0,prcr                     ; Protect on

        jmp.a   Download_program            ; jump Download_program
;
;-----------------------------------------------------------
;+        Time_out                                         +
;-----------------------------------------------------------
Time_out:
        bset    sr9                         ; SRD1 time out flag set
        jmp             Command_err         ; jump Command_err at time out
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+        Subroutine : Initialize_2                        +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Initialize_2:
        bset    sr10                        ; check complete at r0=ffffh
        bset    sr11
        bset    blank                       ; blank flag set
;
;----------------------------------------
;+        UART1                     +
;----------------------------------------
Initialize_21:
;
;----- Function select register A0
;
        mov.b   #10010000b,ps0
;
;----- Function select register B0
;
        mov.b   #00000000b,psl0
;
;----- UART1 transmit/receive mode register
;
        mov.b   #0,u1c1                             ; transmit/receive disable
        mov.b   #0,u1mr                             ; u1mr reset
        mov.b   #00001001b,u1mr
;               |||||+++------- clock synchronous SI/O
;               ||||+---------- external clock
;               ++++----------- fixed
;
```

```
;----- UART1 transmit/receive control register 0
;
       mov.b  #00000100b,u1c0
;              |||| |++------ f1 select
;              |||| +-------- RTS select
;              |||+---------- CTS/RTS enabled
;              ||+----------- CMOS output(TxD)
;              |+------------ falling edge select
;              +------------- LSB first
;
;----- UART transmit/receive control register 2
;
       mov.b  #00000000b,ucon
;              ||||||++------ Transmit buffer empty
;              ||||++-------- Continuous receive mode disabled
;              ||++---------- CLK/CLKS normal
;              |+------------ CTS/RTS shared
;              +------------- fixed
;
;----- UART1 transmit/receive control register 1
;
       mov.b  #00000101b,u1c1
;              |||| | +------ Transmission enabled
;              |||| +-------- Reception enabled
;              +++++--------- fixed
;
;
;-----------------------------------------
;+        Timer A0                        +
;-----------------------------------------
;----- Timer A0 mode register
;
       mov.b  #00000010b,ta0mr
;              |||| |++------- One-shot mode
;              |||| +--------- Pulse not output
;              |||+----------- One-shot start flag
;              ||+------------ fixed
;              ++------------- f1 select
;
;;;;   mov.b  #0,ta0ic                    ; clear TA0 interrupt flag
       mov.w  #6000-1,ta0                 ; set 300 usec at 20 MHz
       bset   ta0s
       mov.b  #0,ta0ic                    ; clear TA0 interrupt flag  changed 0629
;
       rts
;
;
       .end
```

3.8.2 Sample program when using M16C Flash Starter

```
;*****************************************************************************
;*     System Name    : Rewrite Program for M16C/80 BootLoader          *
;*     File Name      : sample_UART.a30                                 *
;*     MCU            : M3080xSGP-BL                                    *
;*     Xin            : 2M-20MHz (for UART mode )                       *
;*---------------------------------------------------------------------------*
;*      Copyright,2001 MITSUBISHI ELECTRIC CORPORATION                      *
;*            AND MITSUBISHI SEMICONDUCTOR SYSTEM CORPORATION               *
;*****************************************************************************
;
;
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+        Include file                                                   +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
       .list         off
       .include      sfr80.inc            ; SFR header include
       .include      bl80.inc             ; Bootloader definition include
       .list         on
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+    Version table                                           +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
       .section      rom,code
       .org          0600h                ; Download address
       .byte         'VER.1.01'           ; Version infomation
;
;==============================================================
;+        Boot program start                                  +
;==============================================================
Program_start:
;-----------------------------------------------------
;+        Initialize_1                                +
;-----------------------------------------------------
       ldc           #Istack,ISP          ; stack pointer set
;
;-----------------------------------------------------
;+     Processor mode register                        +
;+       & System clock control register              +
;-----------------------------------------------------
CPU_set:
       mov.b  #3,prcr                     ; Protect off
       mov.w  #00000011b,pm0              ; wait off, micro processor mode
       mov.b  #02h,mcd                    ; f2
       mov.b  #20h,cm1                    ;
       mov.b  #08h,cm0                    ;
       mov.b  #00001111b,ds               ; data bus width 16bit
       mov.b  #10101010b,wcr              ; all 2wait
       mov.b  #0,prcr                     ; Protect on


;
;-------------------------------------------------------------
```

```
;=========================================================================
;+     Transfer Program -- UART mode                                     +
;+             (1) Main flow                                             +
;+             (2) Communication program for flash memory control        +
;=========================================================================
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+        Main flow  - UART mode -                      +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_Main:
       btst   updata_f                    ;
       bmltu  updata_f                    ; if "C"flag is "0", updata_f set "1"
       jc             U_Main1             ;   if   "C"flag   is   "1",   initialize
execute(jump U_Main1)
       jmp            U_Loop_main         ;

U_Main1:
       bclr   updata_f                    ;

       bclr   freq_set1                   ; freq set flag clear
       bclr   freq_set2
       mov.b  #01111111b,data             ; Initialize Baud rate
       jsr            Initialize_3        ; UART mode Initialize
       mov.b  #01000000b,r1l              ; counbter1,2 reset
       mov.b  #10000000b,r1h
       mov.w  u1rb,r0                     ; receive data --> r0
;
U_Loop_main:
       bclr   te_u1c1                     ; Transmission disabled
       bset   re_u1c1                     ; Reception enabled
?:
       btst   ri_u1c1                     ; receive complete ?
       jz             ?-
       mov.w  u1rb,r0                     ; receive data --> r0
       btst   freq_set2
       jz             U_Freq_check
;
U_Command_check:
       cmp.b  #0ffh,r0l                   ; Read        (ffh)
       jeq            U_Read
       cmp.b  #041h,r0l                   ; Program     (41h)
       jeq            U_Program
       cmp.b  #0a7h,r0l                   ; All erase   (a7h)
       jeq            U_All_erase
       cmp.b  #050h,r0l                   ; Clear SRD   (50h)
       jeq            U_Clear_SRD
       cmp.b  #0fah,r0l                   ; Download    (fah)
       jeq            U_Download

       cmp.b  #070h,r0l                   ; Read SRD    (70h)
       jeq            U_Read_SRD
       cmp.b  #0fbh,r0l                   ; Version out (fbh)
       jeq            U_Ver_output
       cmp.b  #0b0h,r0l                   ; Baud rate 9600bps (b0h)
       jeq            U_BPS_B0
       cmp.b  #0b1h,r0l                   ; Baud rate 19200bps (b1h)
       jeq            U_BPS_B1
       cmp.b  #0b2h,r0l                   ; Baud rate 38400bps (b2h)
```

```
        jeq             U_BPS_B2
        cmp.b   #0b3h,r0l                   ; Baud rate 57600bps (b3h)
        jeq             U_BPS_B3
        cmp.b   #0b4h,r0l                   ; Baud rate 115200bps (b4h)
        jeq             U_BPS_B4
        jsr             U_Initialize_31     ; command error, UART mode Initialize
        jmp             U_Loop_main         ; jump U_Loop_main
;
;-----------------------------------------------------------
;+        Read - UART mode -                               +
;-----------------------------------------------------------
U_Read:
        mov.w   #0,r3                       ; receive number
        mov.b   #0,addr_l                   ; addr_l = 0
?:
        btst    ri_u1c1                     ; receive complete ?
        jnc             ?-
;
        mov.w   u1rb,r0                     ; receive data read --> r0
        add.w   #1,r3                       ; r3 +1 increment
        mov.w   r3,a0                       ; r3 --> a0
        mov.b   r0l,addr_l[a0]              ; Store address
        cmp.w   #2,r3                       ; r3 = 2 ?
        jltu    ?-                          ; jump Read_loop at r3<2
;
        mov.w   addr_l,a0                   ; addr_l,m --> a0
        mov.b   addr_h,a1                   ; addr_h   --> a1
        sha.l   #16,a1                      ;
        add.l   a0,a1                       ; a1 is address-data
;
        bclr    re_u1c1                     ; Reception disabled
        bset    te_u1c1                     ; Transmission enabled
U_Read_data:
        cmp.w   #258,r3                     ; r3 = 258 ?
        jz              U_Read_end
        ;
        ; Flash memory read & store to r1l
        ;
        mov.b   r1l,u1tb                    ; r1l --> transmit buffer register
?:
        btst    ti_u1c1                     ; transmit buffer empty ?
        jnc             ?-
        add.l   #1,a1                       ; address increment
        add.w   #1,r3                       ; counter increment
        jmp             U_Read_data         ; jump U_Read_data
;
U_Read_end:
        btst    txept_u1c0                  ; Transmit register empty ?
        jnc             U_Read_end
        jmp             U_Loop_main
;
```

```
;-------------------------------------------------------------
;+        Program - UART mode -                              +
;-------------------------------------------------------------
U_Program:
        mov.w   #0,r3                       ; receive number
        mov.b   #0,addr_l                   ; addr_l = 0
        mov.w   sum,crcd                    ; for Read check command
U_Program_loop:
        btst    ri_u1c1                     ; receive complete ?
        jnc             U_Program_loop
        mov.w   u1rb,r0                     ; receive data read --> r0
        add.w   #1,r3                       ; r3 +1 increment
        mov.w   r3,a0                       ; r3 --> a0
        mov.b   r0l,addr_l[a0]              ; Store address
        cmp.w   #258,r3                     ; r3 = 258 ?
        jltu    U_Program_loop              ; jump U_Program_loop at r3<258
;
        mov.w   #0,r3                       ; writing number (r3=0)
U_Program_loop_2:
        mov.b   addr_h,a1                   ; addr_h   --> a1
        sha.l   #16,a1
        mov.w   r3,a0                       ; r3      --> a0
        mov.w   data[a0],r1                 ; data    --> r1
        mov.w   addr_l,a0                   ; addr_l,m --> a0
        add.l   a0,a1
        ;
        ; data write
        ;
        mov.b   r1l,crcin                   ; for Read check command
        mov.b   r1h,crcin
;
        add.w   #2,addr_l                   ; address +2 increment
        add.w   #2,r3                       ; writing number +2 increment
        cmp.w   #255,r3                     ; r3 = 255 ?
        jltu    U_Program_loop_2            ; jump U_Program_loop_2 at r3<255
U_Program_end:
        mov.w   crcd,sum                    ; for Read check command
        jmp             U_Loop_main         ; jump U_Loop_main
;
;-------------------------------------------------------------
;+            All erase ( unlock block ) - UART mode -     +
;-------------------------------------------------------------
U_All_erase:
        btst    ri_u1c1                     ; receive complete ?
        jnc             U_All_erase
;
        mov.w   u1rb,r0                     ; receive data read --> r0
        cmp.b   #0d0h,r0l                   ; Confirm command check
        jne             U_All_erase_end     ; jump U_All_erase_end at Confirm command
error
        ;
        ; All erase
        ;
U_All_erase_end:
        jmp             U_Loop_main         ; jump U_Loop_main
;
```

```
;-------------------------------------------------------------
;+        Read SRD - UART mode                               +
;-------------------------------------------------------------
U_Read_SRD:
       bclr   re_u1c1                     ; Reception disabled
;
       mov.w  #0,r3                       ; receive number (r3=0)
;
       mov.b  #80h,r1l                    ; dummy SRD set
       bset   te_u1c1                     ; Transmission enabled
U_Read_SRD_loop:
       mov.b  r1l,u1tb                    ; r1l --> transmit buffer register
?:
       btst   ti_u1c1                     ; transmit buffer empty ?
       jnc            ?-
       mov.b  SRD1,r1l                    ; SRD1 data --> r1l
       add.w  #1,r3                       ; r3 +1 increment
       cmp.w  #2,r3                       ; r3=2 ?
       jltu   U_Read_SRD_loop             ; jump U_Read_SRD_loop at r3<2
U_Read_SRD_end:
       btst   txept_u1c0                  ; Transmit register empty ?
       jnc            U_Read_SRD_end
;
       jmp            U_Loop_main         ; jump U_Loop_main
;
;-------------------------------------------------------------
;+        Clear SRD - UART mode                              +
;-------------------------------------------------------------
U_Clear_SRD:
;
       and.b  #10010000b,SRD1             ; SRD1 clear
;
       jmp            U_Loop_main         ; jump U_Loop_main
;
;-------------------------------------------------------------
;+        Version output - UART mode -                      +
;-------------------------------------------------------------
U_Ver_output:
       mov.w  #0,a0                       ; Version address offset (a0=0)
       bclr   re_u1c1                     ; Reception disabled
       bset   te_u1c1                     ; Transmission enabled
U_Ver_loop:
       mov.b  ver[a0],u1tb                ; Version data transfer
?:
       btst   ti_u1c1                     ; transmit buffer empty ?
       jnc            ?-
       add.w  #1,a0                       ; a0 +1 increment
       cmp.w  #8,a0                       ; a0=8 ?
       jltu   U_Ver_loop                  ; jump U_Ver_loop at a0<8
U_Ver_end:
       btst   txept_u1c0                  ; Transmit register empty ?
       jnc            U_Ver_end
       jmp            U_Loop_main         ; jump U_Loop_main
;
```

```
;-------------------------------------------------------
;+        Download - UART mode -                       +
;-------------------------------------------------------
U_Download:
       mov.b  #3,prcr                     ; Protect off
       mov.w  #0000h,pm0                  ; wait off, single chip mode
       mov.b  #02h,mcd                    ; f2
       mov.b  #20h,cm1                    ;
       mov.b  #08h,cm0                    ;
       mov.b  #0,prcr                     ; Protect on

       jmp.a  U_Download_program          ; jump U_Download_program
;
;-------------------------------------------------------
;+        Baud rate change - UART mode                 +
;-------------------------------------------------------
U_BPS_B0:
       mov.b  baud,data                   ; Baud rate 9600bps
       jmp           U_BPS_SET
U_BPS_B1:
       mov.b  baud+1,data                 ; Baud rate 19200bps
       jmp           U_BPS_SET
U_BPS_B2:
       mov.b  baud+2,data                 ; Baud rate 38400bps
       jmp           U_BPS_SET
U_BPS_B3:
       mov.b  baud+3,data                 ; Baud rate 57600bps
       jmp           U_BPS_SET
U_BPS_B4:
       mov.b  baud+4,data                 ; Baud rate 115200bps
U_BPS_SET:
       bclr   re_u1c1                     ; Reception disabled
       bset   te_u1c1                     ; Transmission enabled
       mov.b  r0l,u1tb                    ; r0l --> transmit buffer register
?:
       btst   ti_u1c1                     ; transmit buffer empty ?
       jnc           ?-
?:
       btst   txept_u1c0
       jnc           ?-
       bclr   te_u1c1                     ; Transmission       disabled
       jsr           U_blank_end          ; UART mode Initialize
       jmp           U_Loop_main          ; jump U_Loo_main
;
```

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+        Freq check  - UART mode -                        +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_Freq_check:
       bclr   re_u1c1                      ; Reception disabled
       btst   0,r1h                        ; counter = 8 times
       jc             U_Freq_check_4
;
       btst   freq_set1
       jc             U_Freq_check_1
       btst   5,r0h                        ; fer_u1rb
       jz             U_Freq_check_3
       jmp            U_Freq_check_2
U_Freq_check_1:
       cmp.b  #00h,r0l                     ; "00h"?
       jeq            U_Freq_check_3
U_Freq_check_2:
       or.b   r1h,r1l                      ; r1l = counter1 or counter2
U_Freq_check_3:
       xor.b  data,r1l                     ; Baud = Baud xor r1l
       mov.b  r1l,data                     ; data set
       mov.b  r1h,r1l
       rot.b  #-1,r1l
       rot.b  #-1,r1h                      ; counter sift
       rot.b  #-1,r1l
       jmp            U_Freq_check_6
;
U_Freq_check_4:
       btst   freq_set1                    ; Baud get ?
       jc             U_Freq_set_1         ; Yes , finished
       bset   freq_set1
       btst   5,r0l                        ; fer_u1rb
       jz             U_Freq_check_5
       xor.b  data,r1h
       mov.b  r1h,data
U_Freq_check_5:
       mov.b  data,data+1                  ; Min Baud --> data+1
       mov.b  #01000000b,r1l               ; counter reset
       mov.b  #10000000b,r1h
       mov.b  #10000000b,data              ; Reset
U_Freq_check_6:
       jsr            U_blank_end          ; UART mode Initialize
?:
       btst   p6_6
       jz             ?-
       jmp            U_Loop_main
;
U_Freq_set_1:
       cmp.b  #00h,r0l                     ; "00h"?
       jeq            U_Freq_set_2
       xor.b  data,r1h
       mov.b  r1h,data
U_Freq_set_2:
       bset   freq_set2
       mov.b  data,r1l                     ; Max Baud --> data
       sub.b  data+1,r1l
       shl.b  #-1,r1l
       add.b  data+1,r1l
```

```
;
       mov.b  r1l,baud                     ; 9600bps
       shl.b  #-1,r1l                      ; 19200bps
       mov.b  r1l,baud+1
       shl.b  #-1,r1l                      ; 38400bps
       mov.b  r1l,baud+2
       mov.b  baud,r0l                     ; 57600bps
       mov.b  #0,r0h
       divu.b #6
       mov.b  r0l,baud+3
       mov.b  baud+3,r0l                   ; 115200bps
       shl.b  #-1,r0l
       mov.b  r0l,baud+4

       mov.b  baud,data
       mov.b  #0b0h,r0l                    ; "B0h" set
       jsr           U_blank_end           ; UART mode Initialize
       jmp           U_BPS_SET
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+        Subroutine : Initialize_3 - UART mode           +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Initialize_3:
U_blank_end:
;
;----------------------------------------
;+        UART1                     +
;----------------------------------------
;----- UART nit rate generator 1
;
       mov.w  data,u1brg
;
U_Initialize_31:
;
;----- Function select register B0
;
       mov.b  #00000000b,psl0
;
;----- Function select register A0
;
       mov.b  #10010000b,ps0                     ; When you hope busy output OFF, set
"#10000000b"
;
;----- UART1 transmit/receive mode register
;
       mov.b  #0,u1c1                            ; transmit/receive disable
       mov.b  #0,u1mr                            ; u1mr reset
       mov.b  #00000101b,u1mr
;              ||||||++---------- transfer data 8 bit long
;              |||||+------------ Internal clock
;              ||||+------------- one stop bit
;              ||++-------------- parity disabled
;              |+---------------- sleep mode deselected
;
```

```
;----- UART1 transmit/receive control register 0
;
       mov.b  #00000100b,u1c0
;              ||||||++---------- f1 select
;              ||||++------------ RTS select
;              |||+-------------- CRT/RTS enabled
;              ||+--------------- CMOS output(TxD)
;              ++---------------- Must always be "0"
;
;----- UART transmit/receive control register 2
;
       mov.b  #00000000b,ucon
;              ||||||++---------- Transmit buffer empty
;              |||+++------------ Invalid
;              ||+--------------- Must always be "0"
;              |+---------------- CTS/RTS shared
;              +----------------- fixed
;
;----- UART1 transmit/received control register 1
;
       mov.b  #00000000b,u1c1
;              |||||||+---------- Transmission disabled
;              ||||||+----------- Transmission enabled
;              |||||+------------ Reception disabled
;              ||||+------------- Reception enabled
;              ++++-------------- fixed
;
       rts
;
       .end
```

3.8.3 Include file sample for the sample program

It is include sample for the section of “3.8.1 Sample program when using MFW-1” and “3.8.2 Sample program when using M16C Flash Starter”.

```
;*********************************************************
;*                                                       *
;*  file name   : definition of Download sample program  *
;*                          for M16C/80 Bootloader       *
;*                                                       *
;*  Version     : 0.01 ( 2000- 8- 1 )                    *
;*              for Bootloader Ver.1.00                  *
;*********************************************************
;
;-------------------------------------------------------
;   define of symbols
;-------------------------------------------------------
Ram_TOP              .equ   000400h              ;;
Istack               .equ   002a00h              ;; Stack pointer
SB_base                     .equ   000400h       ;; SB base
;
Download_program     .equ   0ffe100h      ;;      Download      function      top
address(Bootloader mode1 Sync)
U_Download_program   .equ   0ffe200h      ;;      Download      function      top
address(Bootloader mode2 UART)
;
;
;
Vector     .equ   0ffffdch
;
   .section   memory,data
   .org       Ram_TOP
;
SRD:                 .blkb   1            ;; not use
SRD1:                .blkb   1            ;; SRD1
ver:                 .blkb   10           ;; version infomation
SF:                  .blkb   1            ;; status flag
unuse:               .blkb   4            ;;
addr_l:              .blkb   1            ;; address L
addr_m:              .blkb   1            ;; address M
addr_h:              .blkb   1            ;; address H
data:                .blkb   300          ;; data buffer
buff:                .blkb   20           ;;
ID_err:                     .blkb  1      ;; not use
sum:                 .blkb  2             ;;
baud:                .blkb  5             ;;
BY_sts:                     .blkb  2      ;; not use
;
```

```
sr8                     .btequ  0,SRD1          ;;
sr9                     .btequ  1,SRD1          ;; Time out bit
sr10                    .btequ  2,SRD1          ;; ID check(for Internal flash memory)
sr11                    .btequ  3,SRD1          ;; ID check(for Internal flash memory)
sr12                    .btequ  4,SRD1          ;; check sum bit
sr13                    .btequ  5,SRD1          ;;
sr14                    .btequ  6,SRD1          ;;

sr15                    .btequ  7,SRD1          ;; download check bit
;
ram_check               .btequ 0,SF             ;; not use
blank                   .btequ 1,SF             ;; not use
old_mode                .btequ 2,SF             ;; not use
freq_set1               .btequ 3,SF             ;;
freq_set2               .btequ 4,SF             ;;
updata_f                .btequ 5,SF             ;; download flag
;
;
```

Keep safety first in your circuit designs!

- Mitsubishi Electric Corporation 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 non-flammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

- These materials are intended as a reference to assist our customers in the selection of the Mitsubishi semiconductor 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 Mitsubishi Electric Corporation or a third party.
- Mitsubishi Electric Corporation 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.
- 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 Mitsubishi Electric Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Mitsubishi Electric Corporation or an authorized Mitsubishi Semiconductor product distributor for the latest product information before purchasing a product listed herein.
  The information described here may contain technical inaccuracies or typographical errors. Mitsubishi Electric Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
  Please also pay attention to information published by Mitsubishi Electric Corporation by various means, including the Mitsubishi Semiconductor home page (http://www.mitsubishichips.com).
- 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. Mitsubishi Electric Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
- Mitsubishi Electric Corporation 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 Mitsubishi Electric Corporation or an authorized Mitsubishi Semiconductor 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.
- The prior written approval of Mitsubishi Electric Corporation is necessary to reprint or reproduce in whole or in part these materials.
- 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.
- Please contact Mitsubishi Electric Corporation or an authorized Mitsubishi Semiconductor product distributor for further details on these materials or the products contained therein.