M16C/60, M16C/20, M16C/80 Series
Application Note
< Flash Memory Control>
Preliminary

Mitsubishi Electric Corporation Kitaitami Works
Mitsubishi Electric Semiconductor Systems Corporation
Mitsubishi Electric System LSI Design Corporation

REV.A

Preface

This applecation note is the reference to control flash memory included in the M16C of Mitsubishi 16-bit microcomputers.
For details about instruction, please refer to the software manual supplied with your microcomputer.

Guide to Using This Manual

This manual is a reference for making boot rogram.
This manual consists of five chapters. Do not need to take a look at every chapter.
Look at a necessary chapter, according to a flowchart shown below.

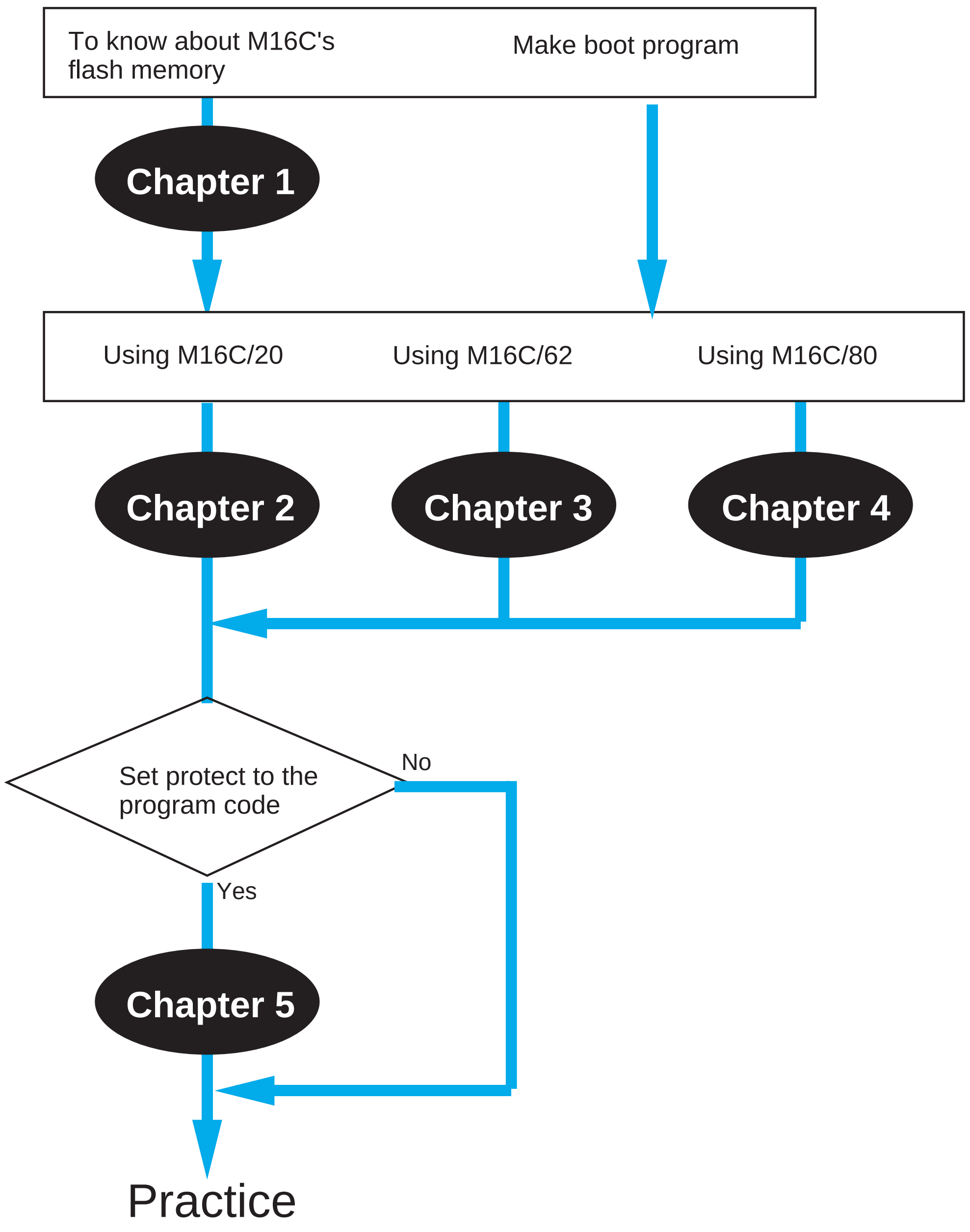


Table of Contents

Chapter 1

Outline of M16C Internal Flash MCU

1.1 What Is Flash Memory?

Flash memory has since around the middle of 1980s been available as an electrically programmable/erasable memory product that does not require battery backup. In recent years, flash memory has been widely used in portable telephones (including PHS), notebook personal computers, and various other portable equipment. This section describes advantages of flash memory and available types.

Advantages of Flash Memory

Flash memory is electrically rewritable nonvolatile memory. Compared to other products such as EPROM and EEPROM that have the same functionality, flash memory is significantly advantageous in chip size and cost. The following lists advantages of flash memory:

(1) Small chip size

The EEPROM memory cell consists of two transistors, whereas that of flash memory consists of one transistor as does EPROM. Therefore, when manufactured using the same fabrication method to provide the same memory capacity as other memory products, flash memory can be manufactured in smaller chip size.

(2) Encapsulated in plastic packages

EPROM requires a window-fitted package because its data is erased by irradiating ultraviolet rays upon it, and cannot be encapsulated in a plastic package. On the other hand, flash memory does not require a window-fitted package because it is electrically erasable, and can be encapsulated in a plastic package.

(3) Unnecessary IC sockets

When mounted on the circuit board, EPROM requires use of an IC socket because it needs to be placed in a ultraviolet radiation unit to erase its data. On the other hand, flash memory can be mounted directly on the circuit board because its data can be electrically erased.

Table 1.1.1 Comparison between Flash Memory and EPROM and EEPROM

	Flash memory	EPROM	EEPROM
Chip size	Small	Small	Not small
Capsulated in plastic packages	Possible	Possible	Impossible
IC socket	Not use	use	Not use

Types of Flash Memory

There are several types of cell structures that comprise flash memory. These, for example, include NOR, DINOR, and AND types. Each type is taken advantage of when using flash memory.

(1) NOR type

The NOR-type cell structure is the same as that of EPROM. Also, because cells can be read at random, EPROM can easily be replaced with this type of flash memory.

(2) DINOR type

The DINOR-type cell structure draws on the tunnel effect to write and erase data. Therefore, this type of flash memory does not require a large current for program/erase operation, making it suitable for use in low-voltage, single-power supply applications.

(3) AND type

The feature of AND type is a reduced cell size, so that its cell structure is ideal for large-capacity memory. Because cell data is read out by serial access, this type of flash memory is used mainly for data storage purposes.

1.2 M16C Family and Flash Memory

The M16C family microcomputers have been well received in consumer electronics and industrial fields for their numerous features including high-efficiency C language support, high performance, superior noise characteristics, and low power consumption. In addition to conventional mask ROM and OTP, a new product with built-in flash memory has been added to the lineup. This section explains about the flash memory that is incorporated in the M16C family microcomputers.

Flash Memory Incorporated in The M16C Family

The M16C family microcomputers incorporate suitable types of flash memory to meet the application needs of each group. The M16C/20 group incorporates the NOR type, while the M16C/62 and 80 groups incorporate the DINOR type.

Table 1.2.1 Flash Memory Incorporated in The M16C Family

	M16C/20 group	M16C/62 and 80 groups
Cell structure	NOR type	DINOR type
Access method	Random access	Random access
Programming method	Byte write	Page write
Erasing method	Collective erase	Block by block erase

Read/Write Modes

The flash memory built into the M16C can be read/written in three modes.

(1) Parallel input/output mode

In this mode, a general-purpose programmer or the suitable flash programmer are used to read or write data. Part of MCU pins are used to send control signals. No software is required.

(2) Serial input/output mode

Read/write operations are controlled via serial interface using part of MCU pins. Data can be read/written using the suitable serial programmer. Software is required, which can be either the standard boot program or a user boot program that supports the protocol of the serial programmer.

(3) CPU rewrite mode

Read/write operations are controlled by setting the control registers in a user program. Software is required, which includes a read program, a write program, and a user boot program including a RAM transfer program.

Standard Boot Program and User Boot Program

The boot program used to control read/writes to flash memory can be the standard boot program that is already included in flash memory or a boot program created by the user to suit the application system.
The standard boot program is prepared by the manufacturer to control flash memory using the manufacturer-designated method. Normally, this program is stored in the boot ROM area. On the other hand, the user boot program is created by the user to control flash memory using the user's own exclusive method. Normally, this program will be stored in the user ROM area.
Table 1.2.2 compares between the standard boot program and user boot program.

Table 1.2.2 Comparing between The Standard Boot Program and User Boot Program

	Standard write program	User boot program
Source	Supplied by manufacturer	Created by user
Required hardware (internal functions)	Fixed	Free
Stored location	Boot ROM area	User ROM area

1.3 Controlling Flash Memory On-Board

To control flash memory on-board, you need to follow a predetermined procedure. This section describes the procedure for controlling flash memory.

Outline of Operation

To control the flash memory built in the M16C, you need to have a program (write control program), known as the boot program, which is necessary to program/erase the flash memory and a program to transfer the said program to RAM. These programs must be written into memory using a general-purpose programmer or a dedicated serial programmer beforehand.
To control flash memory, first transfer the write control program to a RAM area using the RAM transfer program. Then execute the write control program from RAM to write to the flash memory on-board.
Figure 1.3.1 shows an outline of operation when controlling flash memory.

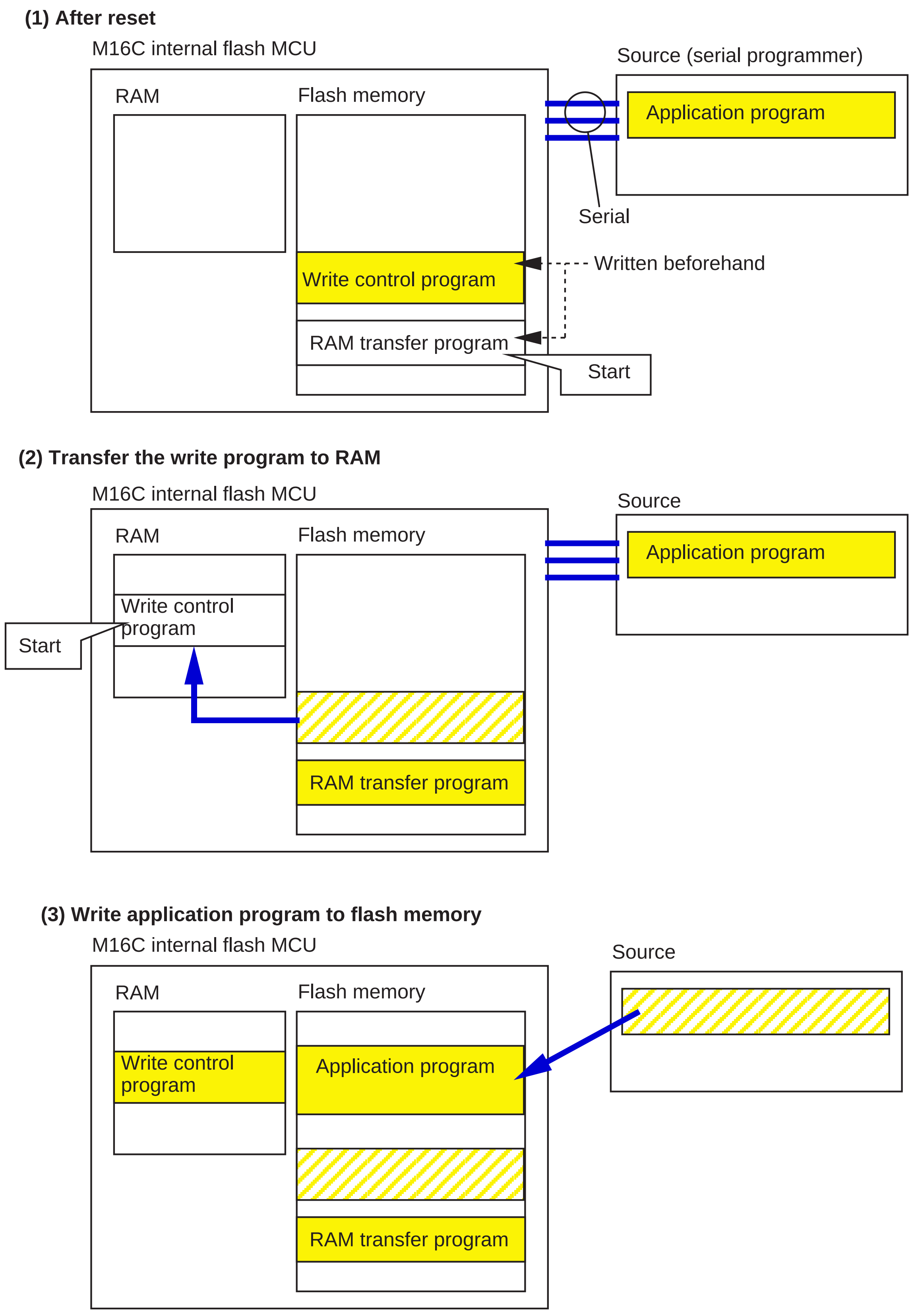


Figure 1.3.1 Operation When Controlling Flash Memory

Flow of Boot Program

Figure 1.3.2 shows a boot program flowchart.
Detail algorithms necessary to program/erase flash memory vary with each group. This is explained in Chapter 2 for the M16C/20 group, and in Chapter 3 for the M16C/62 group. For the M16C/80 group, this is explained in Chapter 4.

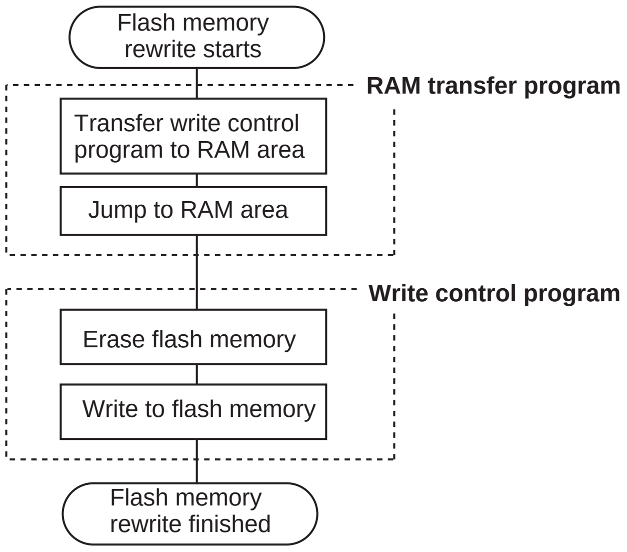

Flash memory rewrite starts
RAM transfer program
Transfer write control program to RAM area
Jump to RAM area
Write control program
Erase flash memory
Write to flash memory
Flash memory rewrite finished

Figure 1.3.2 Flow of Boot Program

Chapter 2

M16C/20 Group

2.1 Outline of Hardware

The M16C/20 group contains NOR-type flash memory.
This section shows hardware information about the M16C/20 group which we think is necessary to create a boot program.

Internal Flash Memory Outline

Table 2.1.1 shows the outline performance of M30201F4 of the M16C/20 group.

Table 2.1.1. Outline Performance of M30201F4

Item		Performance
Power supply voltage		4.0V to 5.5 V (f(X_{IN})=10MHz)
Program/erase voltage		V_{PP}=12V ± 5% (f(X_{IN})=10MHz)
		V_{CC}=5V ± 5% (f(X_{IN})=10MHz)
Flash memory operation mode		Three modes (parallel I/O, standard serial I/O, CPU rewrite)
Erase block division	User ROM area	See Figure 2.1.1.
	Boot ROM area	One division (3.5 Kbytes) (Note)
Program method		In units of byte
Erase method		Collective erase
Program/erase control method		Program/erase control by software command
Number of commands		6 commands
Program/erase count		100 times
ROM code protect		Parallel I/O mode is supported.

Note: The boot ROM area contains a standard serial I/O mode control program which is stored in it when shipped from the factory. This area can be erased and programmed in only parallel I/O mode.

Memory Map

Figure 2.1.1 shows a memory map of the M30201F4. Among the memory areas are a boot ROM area and a user ROM area. Both areas can be accessed for program, read, verify, and erase in parallel input/output mode. In CPU rewrite mode, however, the boot ROM area cannot be accessed for program, verify, and erase.

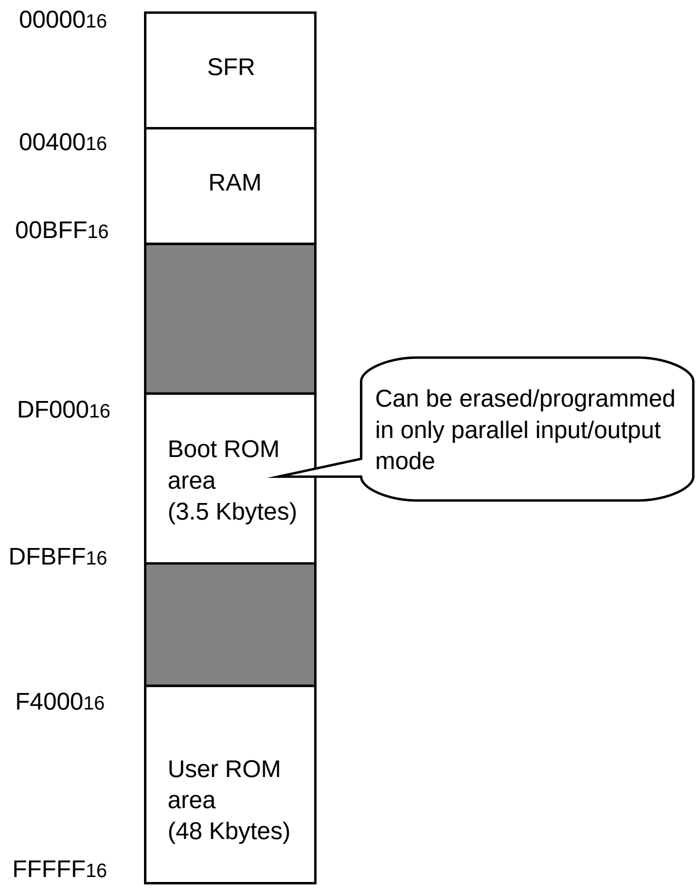

00000₁₆
SFR
00400₁₆
RAM
00BFF₁₆
DF000₁₆
Boot ROM area (3.5 Kbytes)
Can be erased/programmed in only parallel input/output mode
DFBFF₁₆
F4000₁₆
User ROM area (48 Kbytes)
FFFFF₁₆

Figure 2.1.1 M30201F4 Memory Map

Related Register Configuration

Figure 2.1.2 shows related registers for making user boot program.

Flash memory control register 0

b7	b6	b5	b4	b3	b2	b1	b0
0	╳	1	0	0		╳	

Symbol	Address	When reset
FCON0	$03B4_{16}$	00100000_2

Bit symbol	Bit name	Function	R	W
FCON00	CPU rewrite mode select bit	0: CPU rewrite mode is invalid 1: CPU rewrite mode is valid	○	○
Reserved bit		This bit can not write. The value, if read, turns out to be indeterminate.	—	—
FCON02	CPU rewrite mode monitor flag	0: CPU rewrite mode is invalid 1: CPU rewrite mode is valid	○	—
Reserved bit		Must always be set to "0".	○	○
Reserved bit		Must always be set to "1".	○	○
Nothing is assigned. In an attempt to write this bit, write "0". The value, if read, turns out to be "0".			—	—
Reserved bit		Must always be set to "0".	○	○

Flash memory control register 1

b7	b6	b5	b4	b3	b2	b1	b0
╳	╳	╳	╳	╳	╳	0	0

Symbol	Address	When reset
FCON1	$03B5_{16}$	$XXXXXX00_2$

Bit symbol	Bit name	Function	R	W
Reserved bit		Must always be set to "0".	○	○
Nothing is assigned. In an attempt to write these bits, write "0". The value, if read, turns out to be indeterminate.			—	—

Flash command register

b7	b6	b5	b4	b3	b2	b1	b0

Symbol	Address	When reset
FCMD	$03B6_{16}$	00_{16}

Function		R	W
Writing of software command		×	○
<Software command name>	<Command code>		
•Read command	"00_{16}"		
•Program command	"40_{16}"		
•Program verify command	"$C0_{16}$"		
•Erase command	"20_{16}" **+**"20_{16}"		
•Erase verify command	"$A0_{16}$"		
•Reset command	"FF_{16}" **+**"FF_6"		

Figure 2.1.2 Related Register Configuration

Flash Control Circuit

The M16C/20's flash control circuit controls the program, read, verify, and erase operations performed on the internal flash memory. Operation modes are selected by writing commands to the Flash Memory Control Register (addresses $03B4_{16}$, $03B5_{16}$) and Flash Command Register (address $03B6_{16}$). Among the memory areas are a boot ROM area and a user ROM area. Both areas can be accessed for program, read, verify, and erase in parallel input/output mode. In CPU rewrite mode, however, the boot ROM area cannot be accessed for program, verify, and erase.

Software Commands

Table 2.1.2 lists software commands.
When CPU rewrite mode is effective, write software commands to the Flash Command Register to specify the program or erase operations to be performed.

Table 2.1.2 Software Command List

Command	First bus cycle			Second bus cycle		
	Mode	Address	Data (D_0 to D_7)	Mode	Address	Data (D_0 to D_7)
Read	Write	$03B6_{16}$	00_{16}			
Program	Write	$03B6_{16}$	40_{16}	Write	Program address	Program data
Program verify	Write	$03B6_{16}$	$C0_{16}$	Read	Verify address	Verify data
Erase	Write	$03B6_{16}$	20_{16}	Write	$03B6_{16}$	20_{16}
Erase verify	Write	$03B6_{16}$	$A0_{16}$	Read	Verify address	Verify data
Reset	Write	$03B6_{16}$	FF_{16}	Write	$03B6_{16}$	FF_{16}

Read Command (00_{16})

The read mode is entered by writing the command code “00_{16}” to the flash command register in the first bus cycle. When an address to be read is input in one of the bus cycles that follow, the content of the specified address is read out at the data bus (D_0–D_7), 8 bits at a time.
The read mode is retained intact until another command is written.
After reset and after the reset command is executed, the read mode is set.

Program Command (40_{16})

The program mode is entered by writing the command code “40_{16}” to the flash command register in the first bus cycle. When the user execute an instruction to write byte data to the desired address (e.g., STE instruction) in the second bus cycle, the flash memory control circuit executes the program operation. The program operation requires approximately 20 µs. Wait for 20 µs or more before the user go to the next processing.
During program operation, the watchdog timer remains idle, with the value “$7FFF_{16}$” set in it.

Note 1: The write operation is not completed immediately by writing a program command once. The user must always execute a program-verify command after each program command executed. And if verification fails, the user need to execute the program command repeatedly until the verification passes. See Figure 2.1.3 for an example of a programming flowchart.

Program-Verify Command ($C0_{16}$)

The program-verify mode is entered by writing the command code “$C0_{16}$” to the flash command register in the first bus cycle. When the user execute an instruction (e.g., LDE instruction) to read byte data from the address to be verified (the previously programmed address) in the second bus cycle, the content that has actually been written to the address is read out from the memory.
The CPU compares this read data with the data that it previously wrote to the address using the program command. If the compared data do not match, the user need to execute the program and program-verify operations one more time.

Erase Command (20_{16} + 20_{16})

The flash memory control circuit executes an erase operation by writing command code “20_{16}” to the flash command register in the first bus cycle and the same command code to the flash command register again in the second bus cycle. The erase operation requires approximately 20 ms. Wait for 20 ms or more before the user go to the next processing.
Before this erase command can be performed, all memory locations to be erased must have had data “00_{16}” written to by using the program and program-verify commands. During erase operation, the watchdog timer remains idle, with the value “$7FFF_{16}$ set in it.

Note 1: The erase operation is not completed immediately by writing an erase command once. The user must always execute an erase-verify command after each erase command executed. And if verification fails, the user need to execute the erase command repeatedly until the verification passes. See Figure 2.1.3 for an example of an erase flowchart.

Erase-Verify Command ($A0_{16}$)

The erase-verify mode is entered by writing the command code “$A0_{16}$” to the flash command register in the first bus cycle. When the user execute an instruction to read byte data from the address to be verified (e.g., LDE instruction) in the second bus cycle, the content of the address is read out.
The CPU must sequentially erase-verify memory contents one address at a time, over the entire area erased. If any address is encountered whose content is not “FF_{16}” (not erased), the CPU must stop erase-verify at that point and execute erase and erase-verify operations one more time.
Note 1: If any unerased memory location is encountered during erase-verify operation, be sure to execute erase and erase-verify operations one more time. In this case, however, the user does not need to write data “00_{16}” to memory before erasing.

Reset Command (FF_{16} + FF_{16})

The reset command is used to stop the program command or the erase command in the middle of operation. After writing command code “40_{16}” or “20_{16}” twice to the flash command register, write command code “FF_{16}” to the flash command register in the first bus cycle and the same command code to the flash command register again in the second bus cycle. The program command or erase command is disabled, with the flash memory placed in read mode.

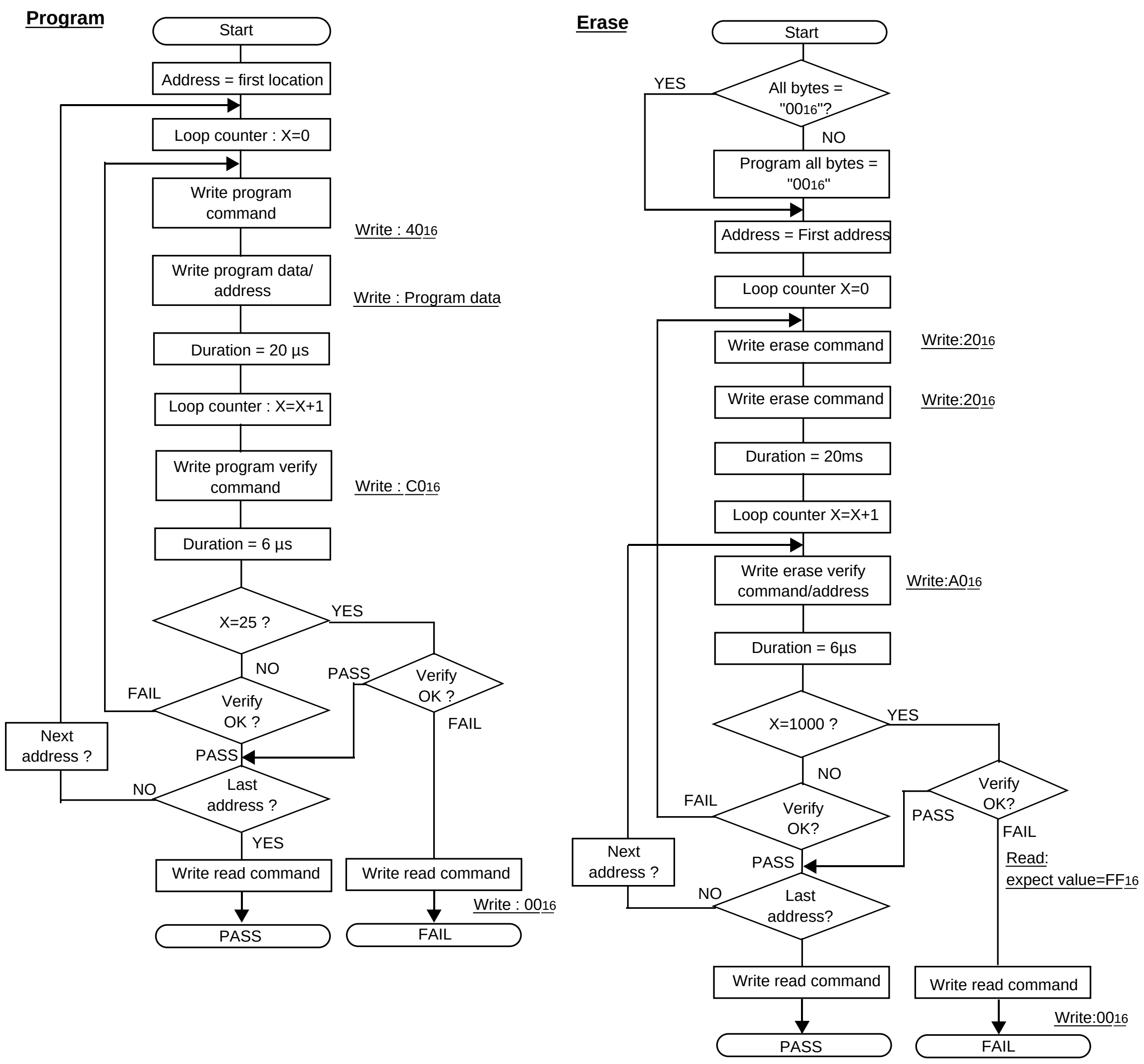


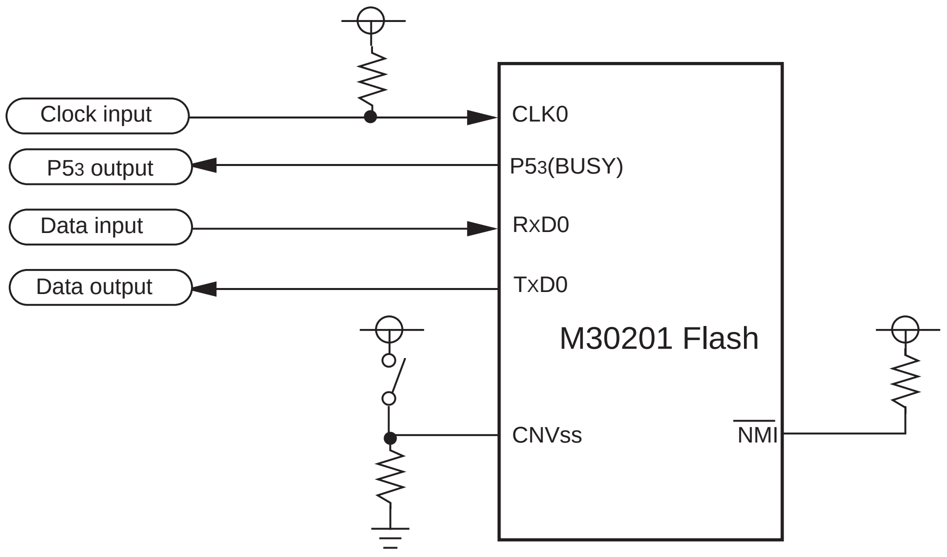
Figure 2.1.3 Program and Erase Execution Flowchart in The CPU Rewrite Mode

2.2 Developing The Boot Program

The standard boot program that was built into the boot ROM area of the flash microcomputer when shipped from the factory can be used to program/erase the flash memory. In this case, the hardware resources (internal functions) used for control are fixed. Therefore, if you want to control flash memory in the way suitable for your system, you need to create a boot program for yourself.
This section shows an algorithm for the boot program (e.g., for erase and program) that you must at least have in order to control the flash memory of the M16C/20 group.

System Example

By using the internal peripheral function of UART0 and a serial programmer to control flash memory, the following shows an example of device connections is shown in Figure 2.2.1. Assignments of internal peripheral functions are listed in Table 2.2.1.



(1) Control pins and external circuitry will vary according to peripheral unit (programmer). For more information, see the peripheral unit (programmer) manual.
(2) In this example, the microprocessor mode and standard serial I/O mode are switched via a switch.

Figure 2.2.1 Example of Device Connection

Table 2.2.1 Assignments of Internal Peripheral Functions

Peripheral function	Usage	Setting example
UART1	Used for transfer/receive of serial programmer and data	• Clock synchronous serial I/O • External clock used
Timer A0	Used for time-over judgment of serial transfer/ receive Used to watch time during program and erase	• One-shot timer mode • 300 µs(at 10MHz) • 20 µs (at 10 MHz)
Timer B0	Used for BUSY waveform output time during serial transmission/reception Used for wait time during verify	• Timer mode • 6 µs (at 10 MHz)

Flow of The Main Processing

Figure 2.2.2 shows a flow of the main processing.
After initializing the CPU, transfer the write control program to RAM. When transfer is finished, jump to RAM and execute the write control program from RAM.

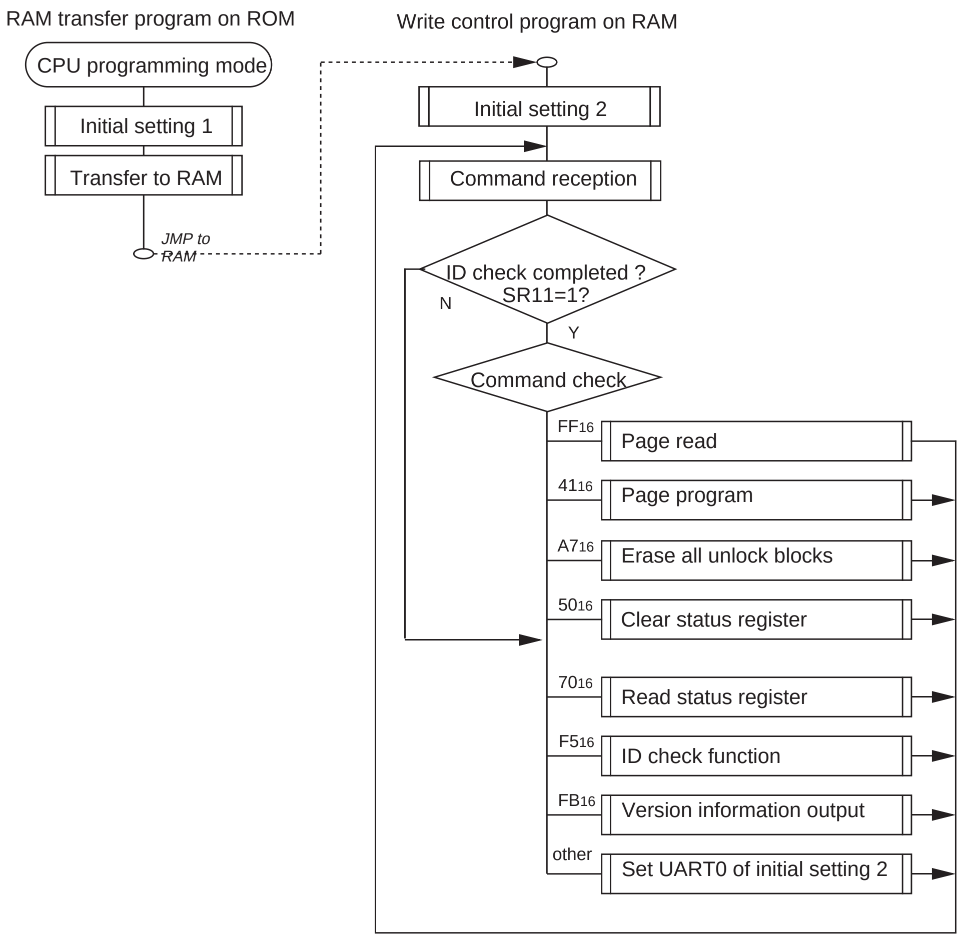


Figure 2.2.2 Flow of The Main Processing

Initialization 1 (CPU, Memory)

The CPU and RAM and the peripheral functions used for programming flash memory are initialized. Figure 2.2.3 shows a flow of CPU and memory initialization. To clear RAM, use of string instructions (e.g., SSTR.W) will prove effective.

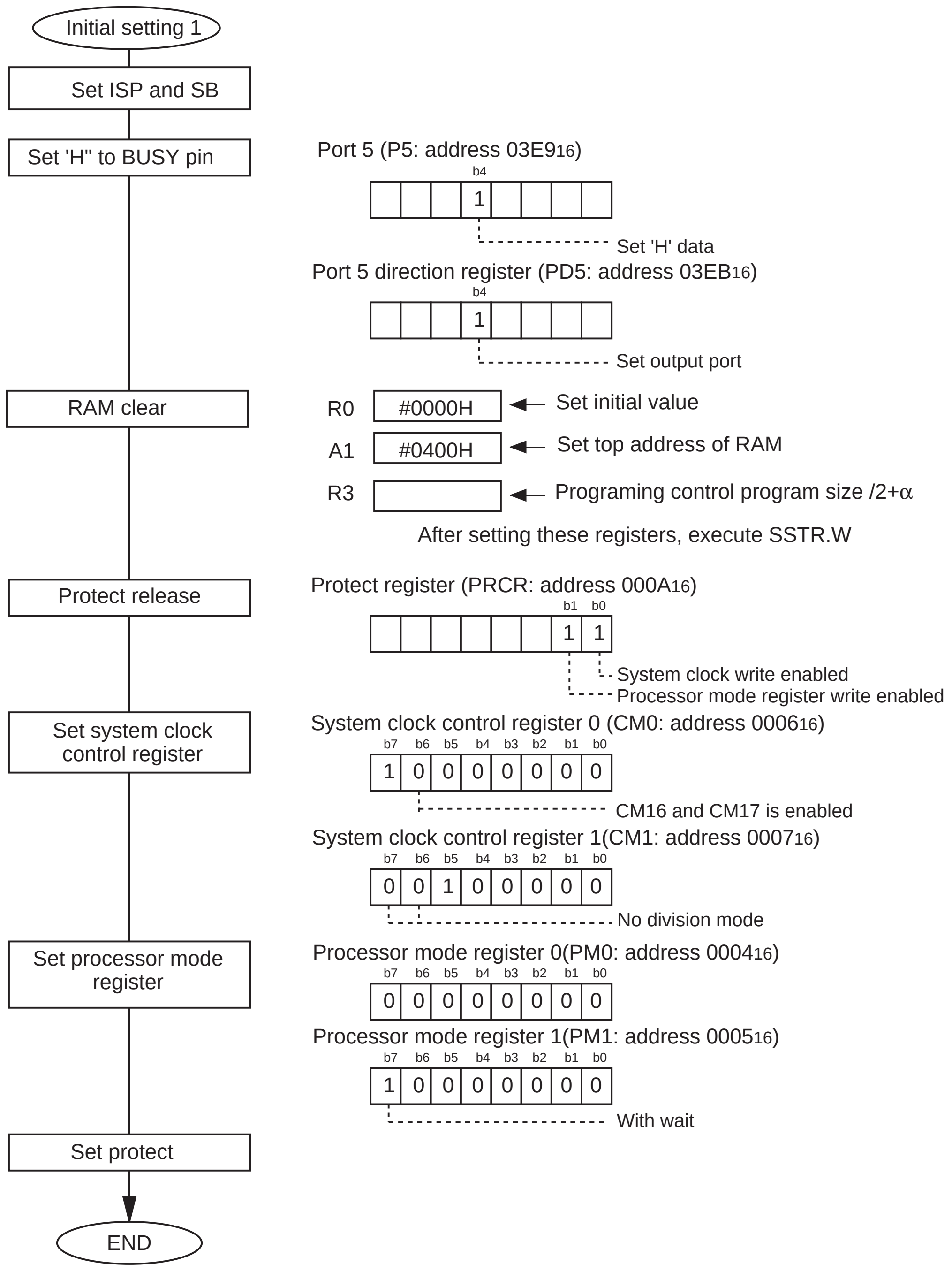

Initial setting 1
Set ISP and SB
Set 'H" to BUSY pin
RAM clear
Protect release
Set system clock control register
Set processor mode register
Set protect
END
Port 5 (P5: address 03E9₁₆)
b4
1
Set 'H' data
Port 5 direction register (PD5: address 03EB₁₆)
b4
1
Set output port
R0 #0000H Set initial value
A1 #0400H Set top address of RAM
R3 Programing control program size /2+α
After setting these registers, execute SSTR.W
Protect register (PRCR: address 000A₁₆)
b1 b0
1 1
System clock write enabled
Processor mode register write enabled
System clock control register 0 (CM0: address 0006₁₆)
b7 b6 b5 b4 b3 b2 b1 b0
1 0 0 0 0 0 0 0
CM16 and CM17 is enabled
System clock control register 1(CM1: address 0007₁₆)
b7 b6 b5 b4 b3 b2 b1 b0
0 0 1 0 0 0 0 0
No division mode
Processor mode register 0(PM0: address 0004₁₆)
b7 b6 b5 b4 b3 b2 b1 b0
0 0 0 0 0 0 0 0
Processor mode register 1(PM1: address 0005₁₆)
b7 b6 b5 b4 b3 b2 b1 b0
1 0 0 0 0 0 0 0
With wait

Figure 2.2.3 Initialization 1 (CPU, Memory)

Transfer to RAM Area

The write control program is transferred to RAM. After transferring, jump to write control program on RAM.
To transfer, use of string instructions (SMOVF.W) will prove effective.
Figure 2.2.4 shows the algorithm.

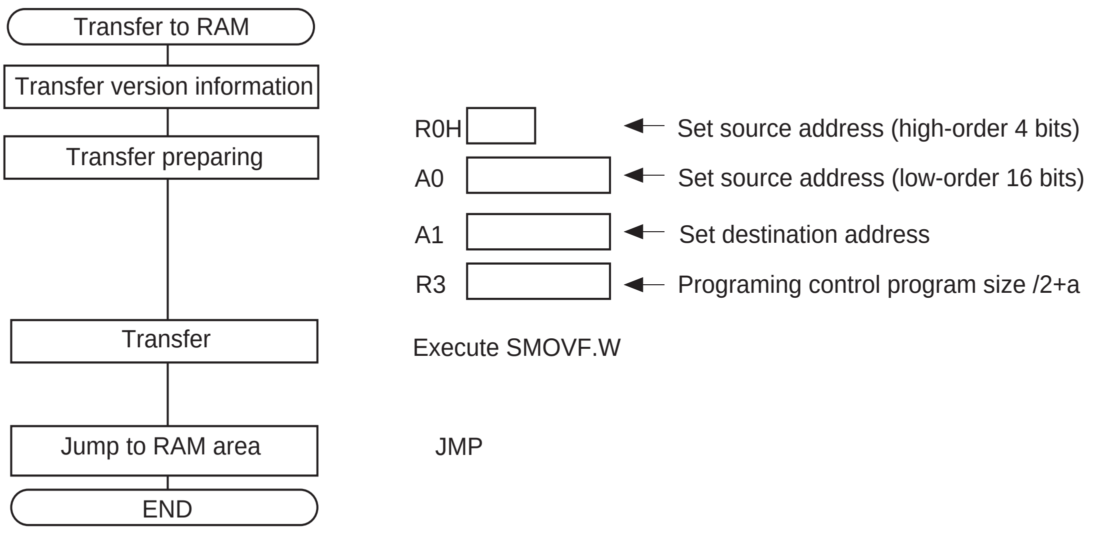

Transfer to RAM
Transfer version information
Transfer preparing
Transfer
Jump to RAM area
END
R0H
A0
A1
R3
Set source address (high-order 4 bits)
Set source address (low-order 16 bits)
Set destination address
Programing control program size /2+a
Execute SMOVF.W
JMP

Figure 2.2.4 Transfer to RAM Area

Initialization 2

Set of write to Flash memory is executed. The flash mode register is set in M16C/20 group.
Figure 2.2.5 shows a algorithm.

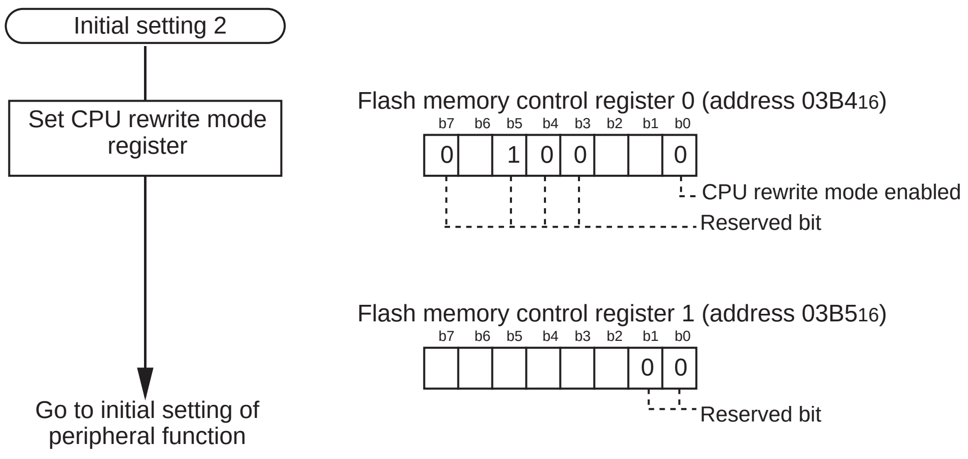

Initial setting 2
Set CPU rewrite mode register
Go to initial setting of peripheral function
Flash memory control register 0 (address 03B416)
b7 b6 b5 b4 b3 b2 b1 b0
0 1 0 0 0
CPU rewrite mode enabled
Reserved bit
Flash memory control register 1 (address 03B516)
b7 b6 b5 b4 b3 b2 b1 b0
0 0
Reserved bit

Figure 2.2.5 Initialization 2

Initialization 2 (Peripheral Function)

The peripheral functions used for programming to flash memory is initialized. Figure 2.2.6 shows initialization of UART0 for data transmit and timer A0 and B0 for time-out calculation.

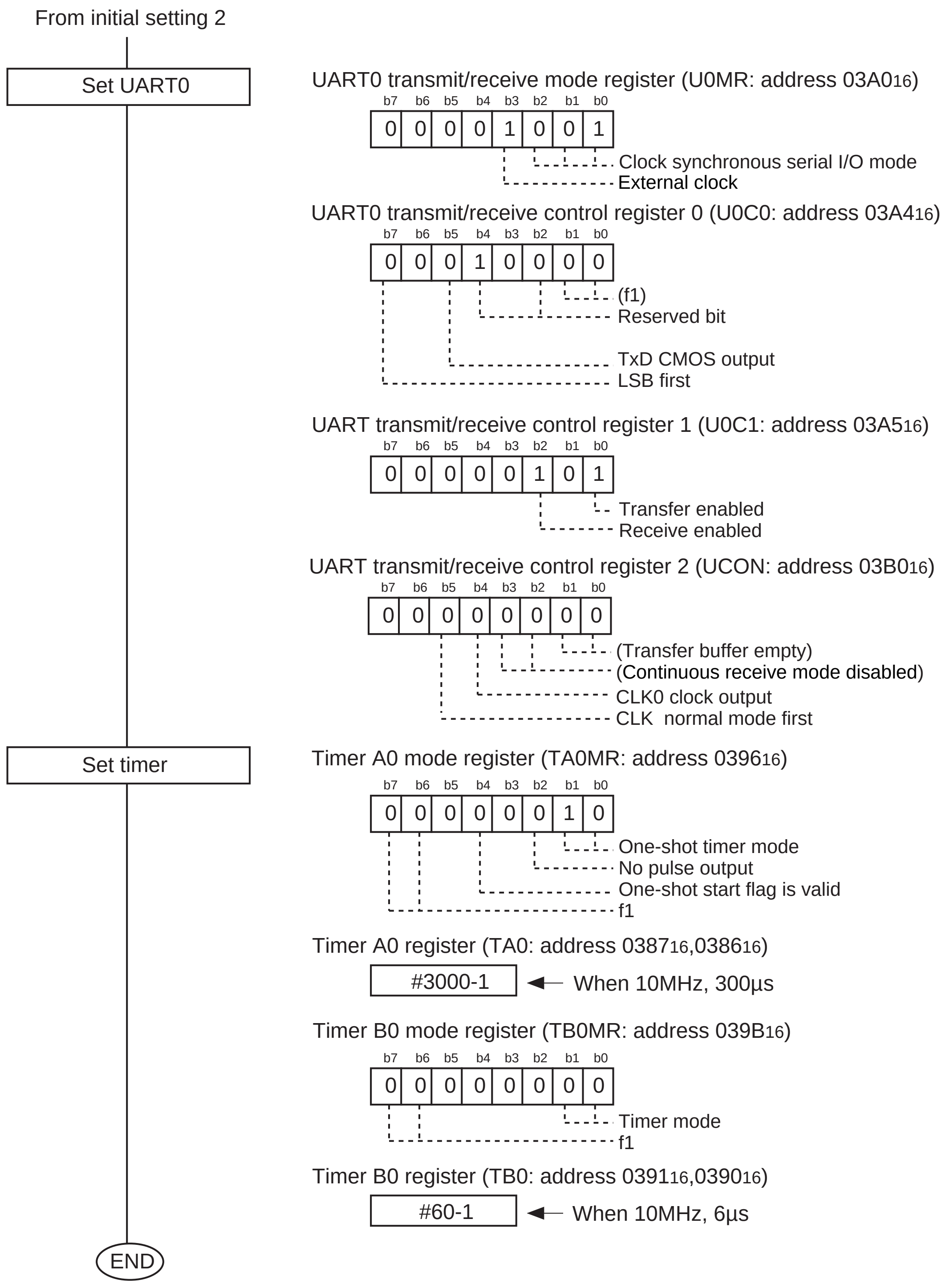

From initial setting 2
Set UART0
UART0 transmit/receive mode register (U0MR: address 03A016)
b7 b6 b5 b4 b3 b2 b1 b0
0 0 0 0 1 0 0 1
Clock synchronous serial I/O mode
External clock
UART0 transmit/receive control register 0 (U0C0: address 03A416)
0 0 0 1 0 0 0 0
(f1)
Reserved bit
TxD CMOS output
LSB first
UART transmit/receive control register 1 (U0C1: address 03A516)
0 0 0 0 0 1 0 1
Transfer enabled
Receive enabled
UART transmit/receive control register 2 (UCON: address 03B016)
0 0 0 0 0 0 0 0
(Transfer buffer empty)
(Continuous receive mode disabled)
CLK0 clock output
CLK normal mode first
Set timer
Timer A0 mode register (TA0MR: address 039616)
0 0 0 0 0 0 1 0
One-shot timer mode
No pulse output
One-shot start flag is valid
f1
Timer A0 register (TA0: address 038716,038616)
#3000-1
When 10MHz, 300µs
Timer B0 mode register (TB0MR: address 039B16)
0 0 0 0 0 0 0 0
Timer mode
f1
Timer B0 register (TB0: address 039116,039016)
#60-1
When 10MHz, 6µs
END

Figure 2.2.6 Initialization 2 (Peripheral Function)

Receiving Commands

Receive commands is received from the serial programmer.
After a wait time of 300 µs, write dummy data to the transmit buffer, pull the BUSY signal low, and wait for data from the serial programmer. After receiving data, return the BUSY signal back high and read the received data.
Figure 2.2.7 shows a flow of control. Figure 2.2.8 shows BUSY signal control timing.

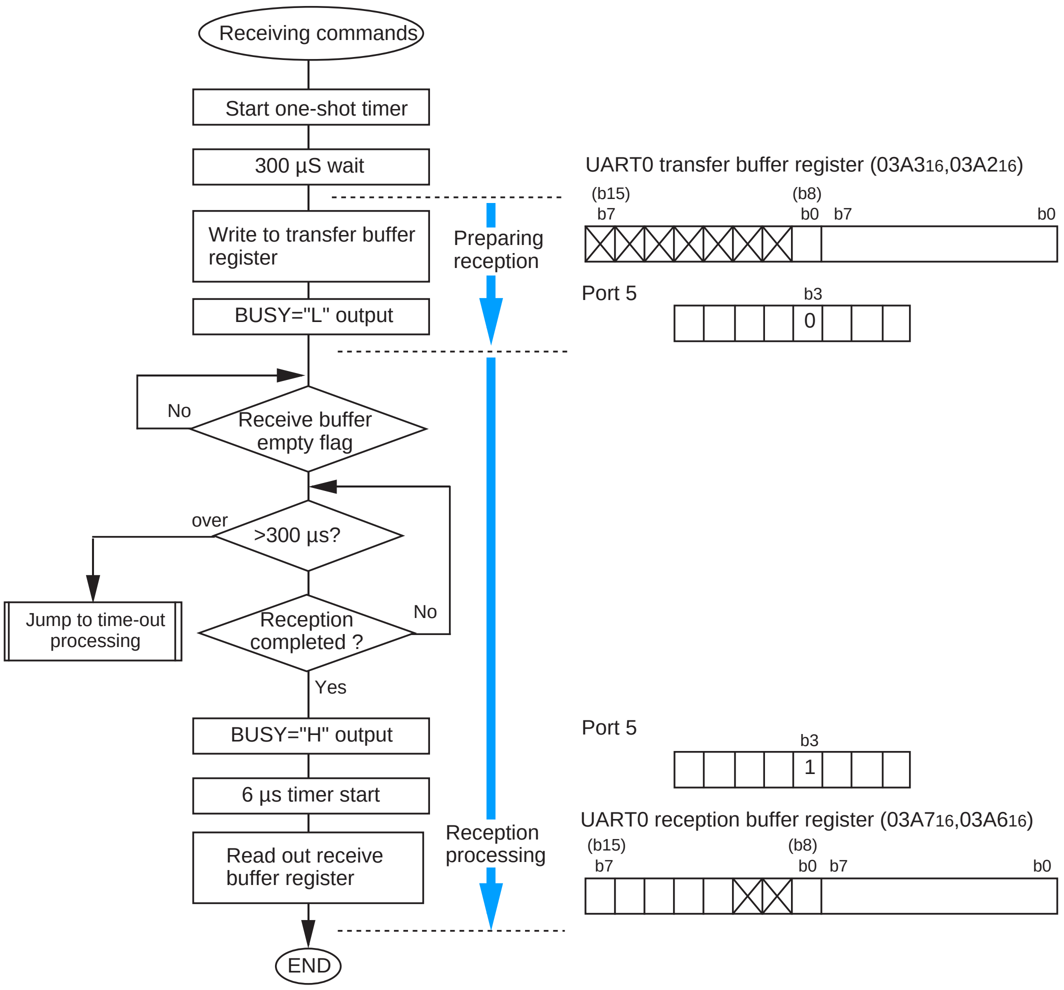


Figure 2.2.7 Receiving Commands

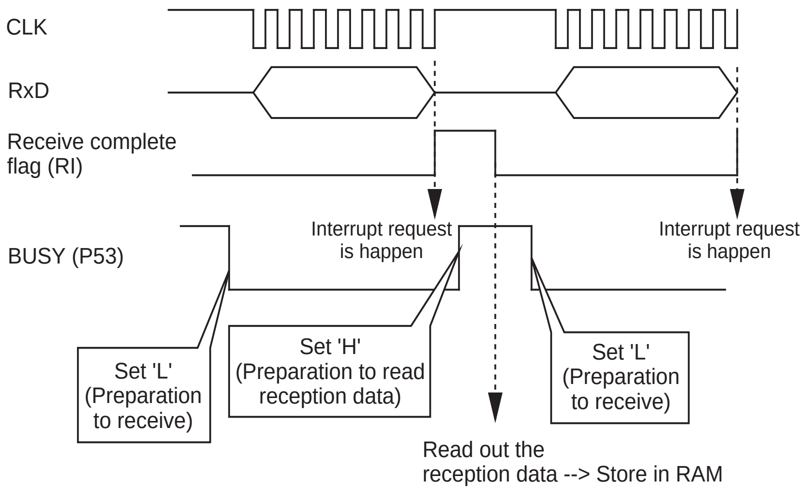

CLK
RxD
Receive complete flag (RI)
BUSY (P53)
Interrupt request is happen
Interrupt request is happen
Set 'L' (Preparation to receive)
Set 'H' (Preparation to read reception data)
Set 'L' (Preparation to receive)
Read out the reception data --> Store in RAM

Figure 2.2.8 BUSY Signal Control Timing

Page Read

Read a specified page (256 bytes) in the user ROM area, one byte at a time, and transmit the read data via serial I/O. The locations to read are addresses $xxx00_{16}$ through $xxxFF_{16}$, with data sequentially transmitted beginning with address $xxx00_{16}$.
Receive two bytes of address from the serial programmer and store it in RAM. Write read command (00_{16}) to the Flash Command Register, read data from flash memory one byte at a time, and transmit it via serial I/O. When you finished sending one page of data (256 bytes), terminate the processing.
Figures 2.2.9 show processing flows.

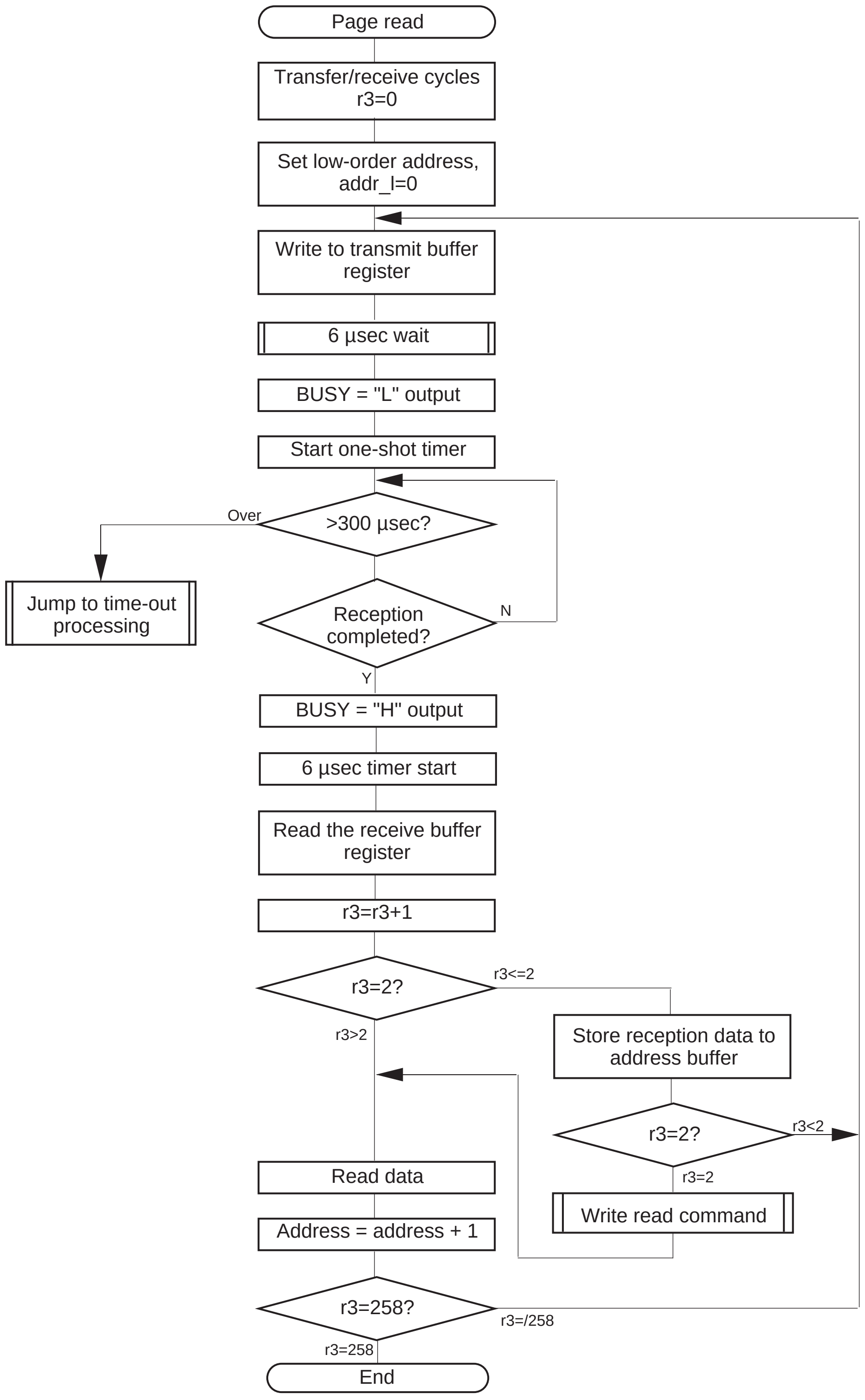


Figure 2.2.9 Page Read

Page Program

Write data to the user ROM area in units of 256 bytes. The locations to write are addresses xxx00$_{16}$ through xxxFF$_{16}$, with data sequentially received beginning with address xxx00$_{16}$.
Receive a total of 258 bytes of data, consisting of two bytes of address and 256 bytes of write data, from the serial programmer and store them in RAM.
After receiving all of these data, check for error.
If the CPU rewrite mode monitor flag is invalid (FCON02 = 0), assume a program error (SRD4 = 1). When you received an invalid address, assume an address error (SRD8 = 1). If the flag is valid (FCON02 = 1), write Program command (40$_{16}$) to the Flash Command Register and execute an instruction that writes the address to be programmed and the byte data.
After an elapse of 20 µs, verify Program.
Write Program Verify command (C0$_{16}$) to the Flash Command Register and after an elapse of 6 µs, execute an instruction that reads the programmed address. If the read data matches the written data, program the next address. If the data do not match, execute Program and Program Verify operations over again. If the data still do not match after repeating this 25 times, assume a program error (SRD4 = 1). If an error is found after programming 256 bytes or by error determination, write Reset command (FF$_{16}$) to the Flash Command Register twice in succession. Then write Read command (00$_{16}$) and return to the main routine.
Figures 2.2.10 and 2.2.11 show processing flows.

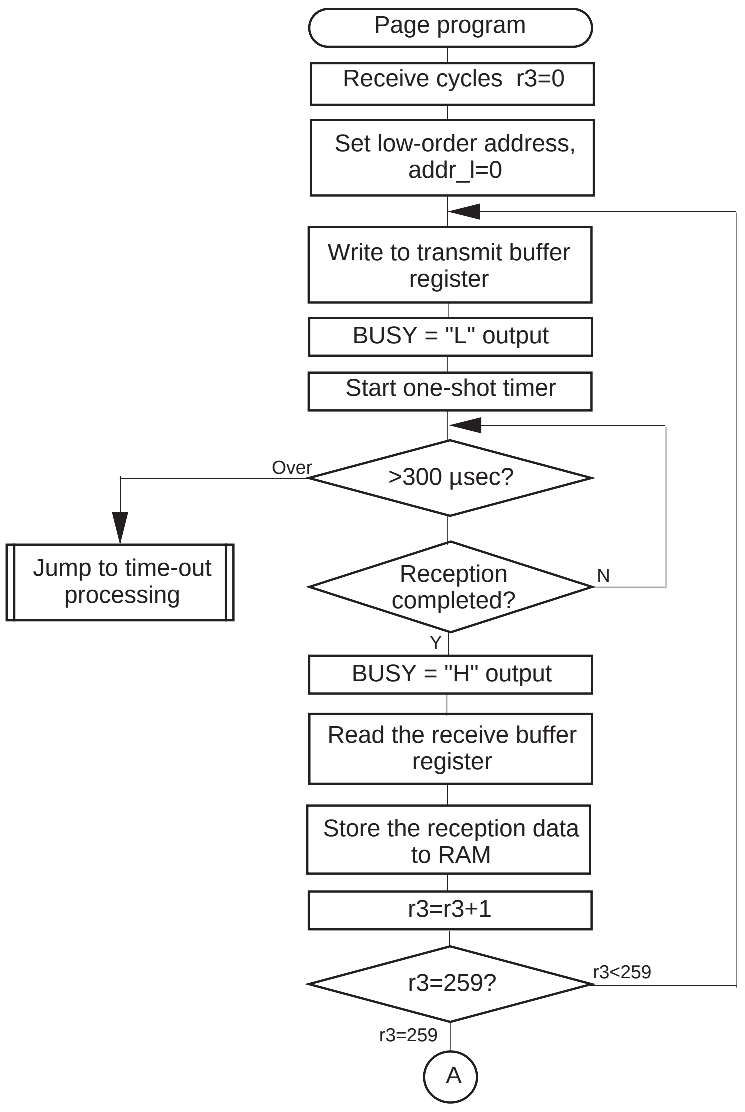

Page program
Receive cycles r3=0
Set low-order address, addr_l=0
Write to transmit buffer register
BUSY = "L" output
Start one-shot timer
>300 µsec?
Over
Jump to time-out processing
Reception completed?
N
Y
BUSY = "H" output
Read the receive buffer register
Store the reception data to RAM
r3=r3+1
r3=259?
r3<259
r3=259
A

Figure 2.2.10 Page Program (1)

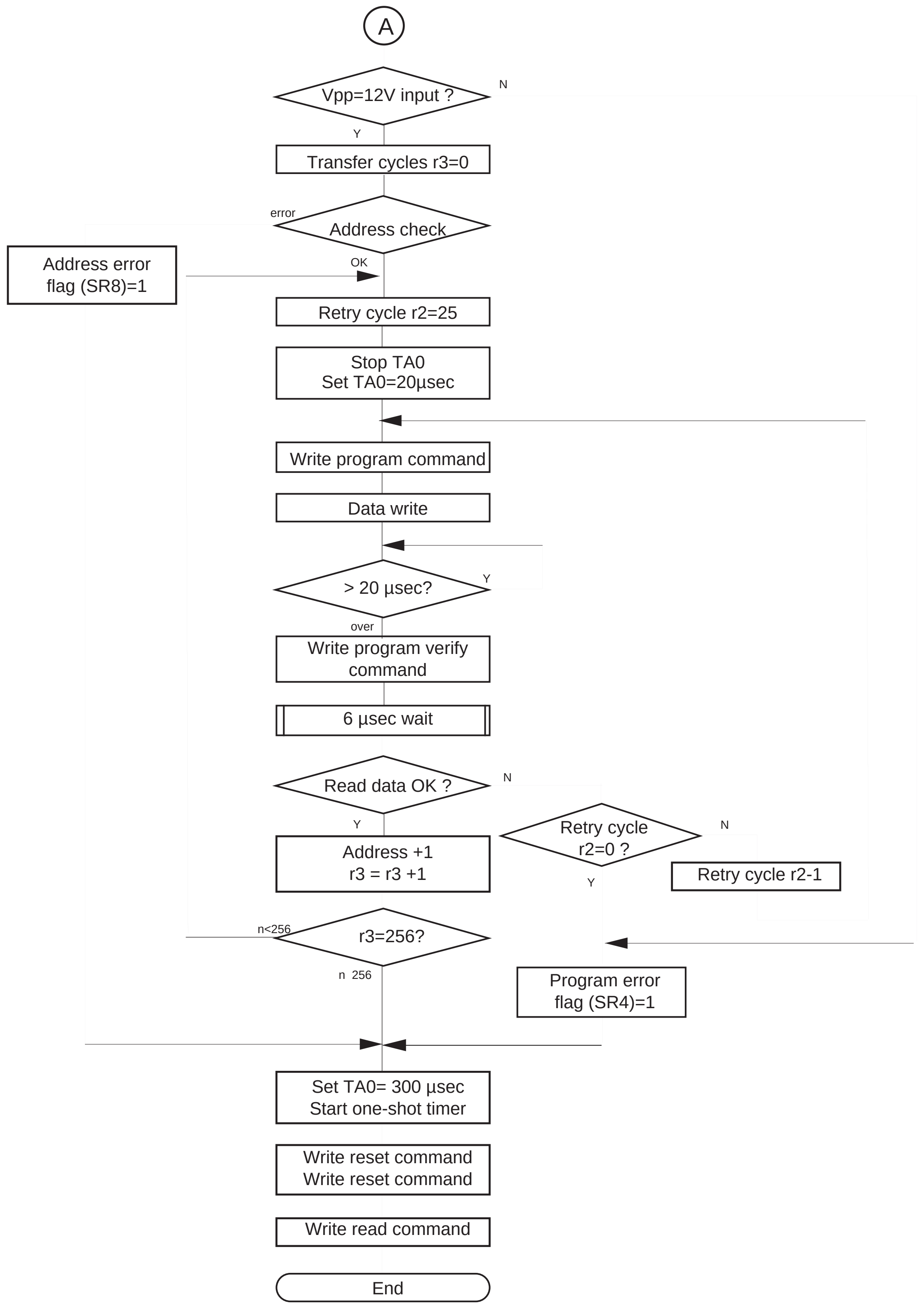


Figure 2.2.11 Page Program (2)

All Erase (Erase All Unlock Blocks)

Erase the entire user ROM area of flash memory.
If All Erase command is received from the serial programmer, continue and receive one more byte of data. After confirming that this second byte of data is the verify command, check for error.
If the CPU rewrite mode monitor flag is invalid (FCON02 = 0), assume an erase error (SRD5 = 1). If the flag is valid (FCON02 = 1), write Program command (40_{16}) to the Flash Command Register, set the start address $F4000_{16}$ and end address $FFFFF_{16}$, set #00_{16} in the write data, and then execute an instruction to write the data.
After an elapse of 20 µs, verify Program.
Write Program Verify command ($C0_{16}$) to the Flash Command Register and after an elapse of 6 µs, execute an instruction that reads the programmed address. If the read data matches the written data, program the next address. If the data do not match, execute Program and Program Verify operations over again. If the data still do not match after repeating this 25 times, assume an erase error (SRD5 = 1).
After you finished writing #00_{16} in the entire area, write Erase command (20_{16}) to the Flash Command Register twice in succession. After an elapse of 20 ms, write Erase Verify command ($A0_{16}$) to the Flash Command Register and after an elapse of 6 µs, execute an instruction that reads the erased address. If the read data matches #FF_{16}, check the next address. If the data do not match, execute Erase and Erase Verify operations over again. If the data still do not match after repeating this 1000 times, assume an erase error (SRD5 = 1).
If an error is found after erase is finished or by error determination, write Reset command (FF_{16}) to the Flash Command Register twice in succession. Then write Read command (00_{16}) and return to the main routine.
Figures 2.2.12 and 2.2.13 show processing flows.

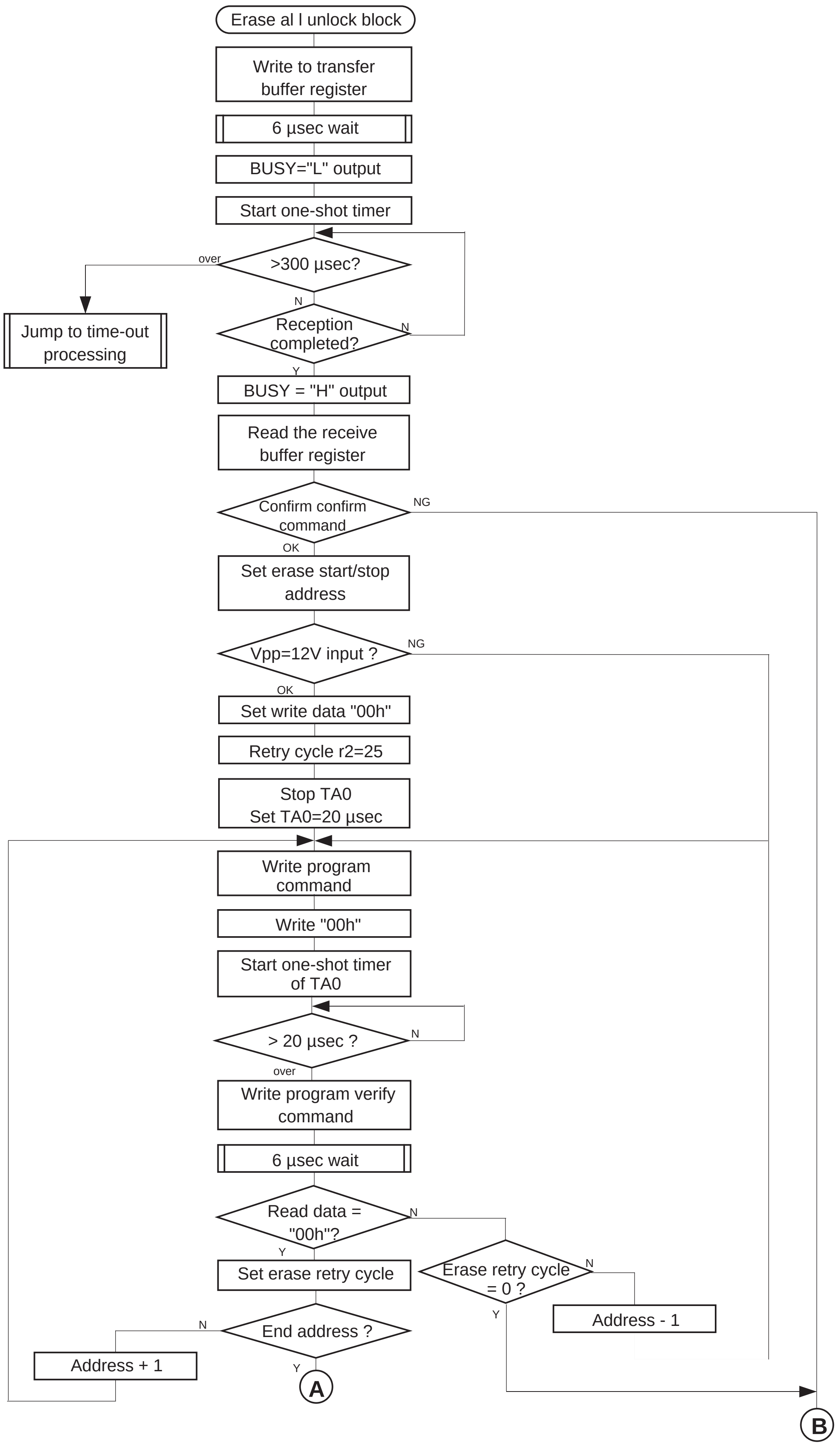


Figure 2.2.12 Erase All Unlock Block (1)

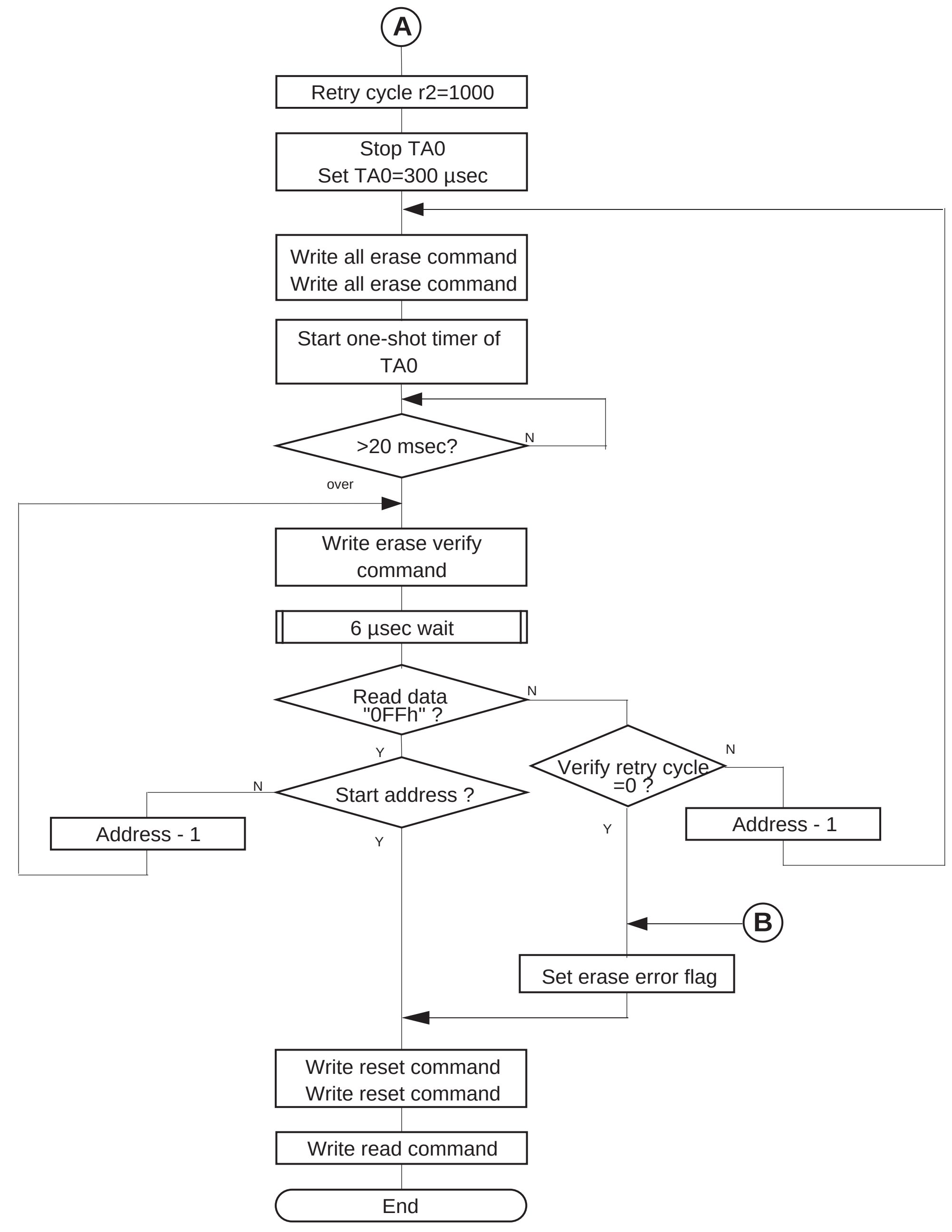


Figure 2.2.13 Erase All Unlock Block (2)

Read Status Register

Transmit two bytes of status data indicating the flash memory's operating status via serial I/O.
Write status data (SRD) to the Transmit Buffer Register and transmit it.
After you finished sending, write status register 1 (SRD1) to the Transmit Buffer Register and transmit it.
After you finished sending, return to the main routine.
Figure 2.2.14 shows a processing flow.

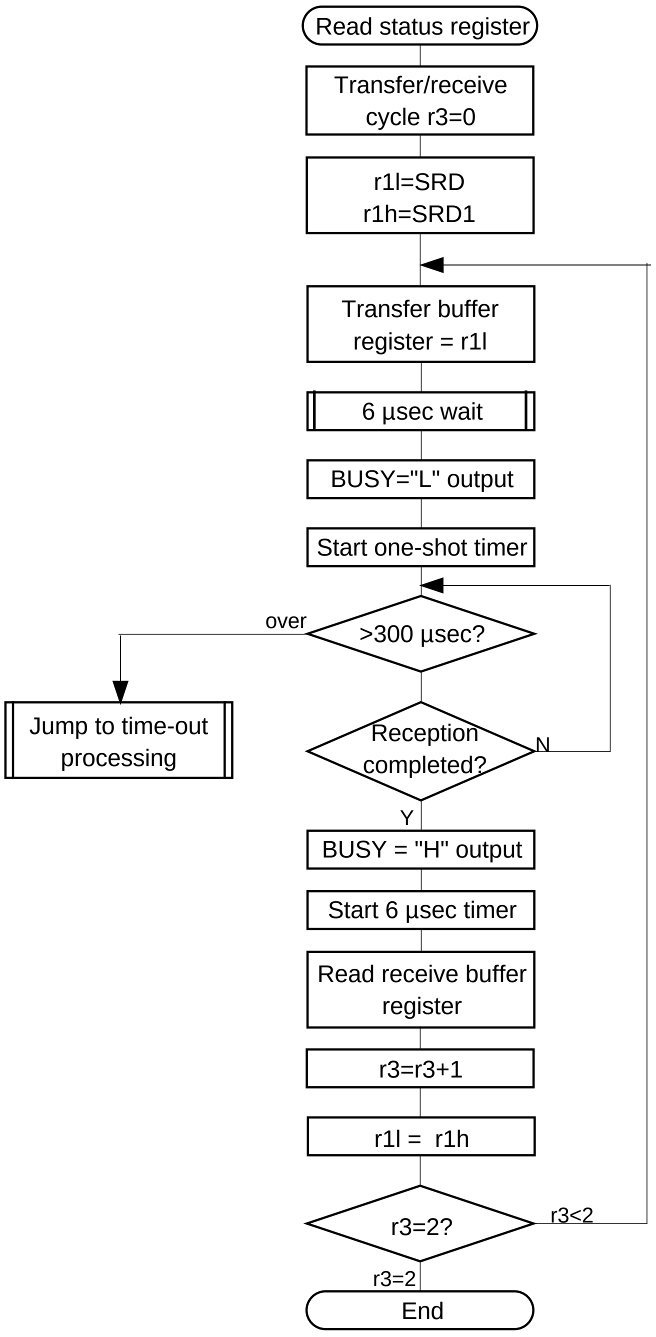


Figure 2.2.14 Read Status Register

Clear Status Register

Status register error information is cleared.
$\#80_{16}$ is written into the status register (SRD).
The logic sum for the status register 1 (SRD1) is obtained on $\#9C_{16}$ is cleared. Processing returns to the main part.
Figure 2.2.15 shows a processing flow.

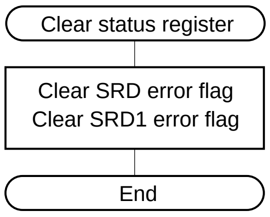


Figure 2.2.15 Clear Status Register

ID Check

The ID data stored in the flash memory is compared with the data received by serial I/O. Three bytes of the address data, one byte of ID size and some bytes of ID check data are received via serial I/O.
After these data reception, this process judges whether the flash memory is blank or not. When blank, the ID check is ended and processing returns to the main part. When something is written in the ROM, the received ID address, the ID data size and ID data contents are checked. When mismatch, ID check error is generated (SR10 = 1, SR11 = 0) and processing returns to the main part. When match, the ID check is ended (SR10 = 1, SR11 = 1) and processing returns to the main part.
Figure 2.2.16 shows a processing flow.

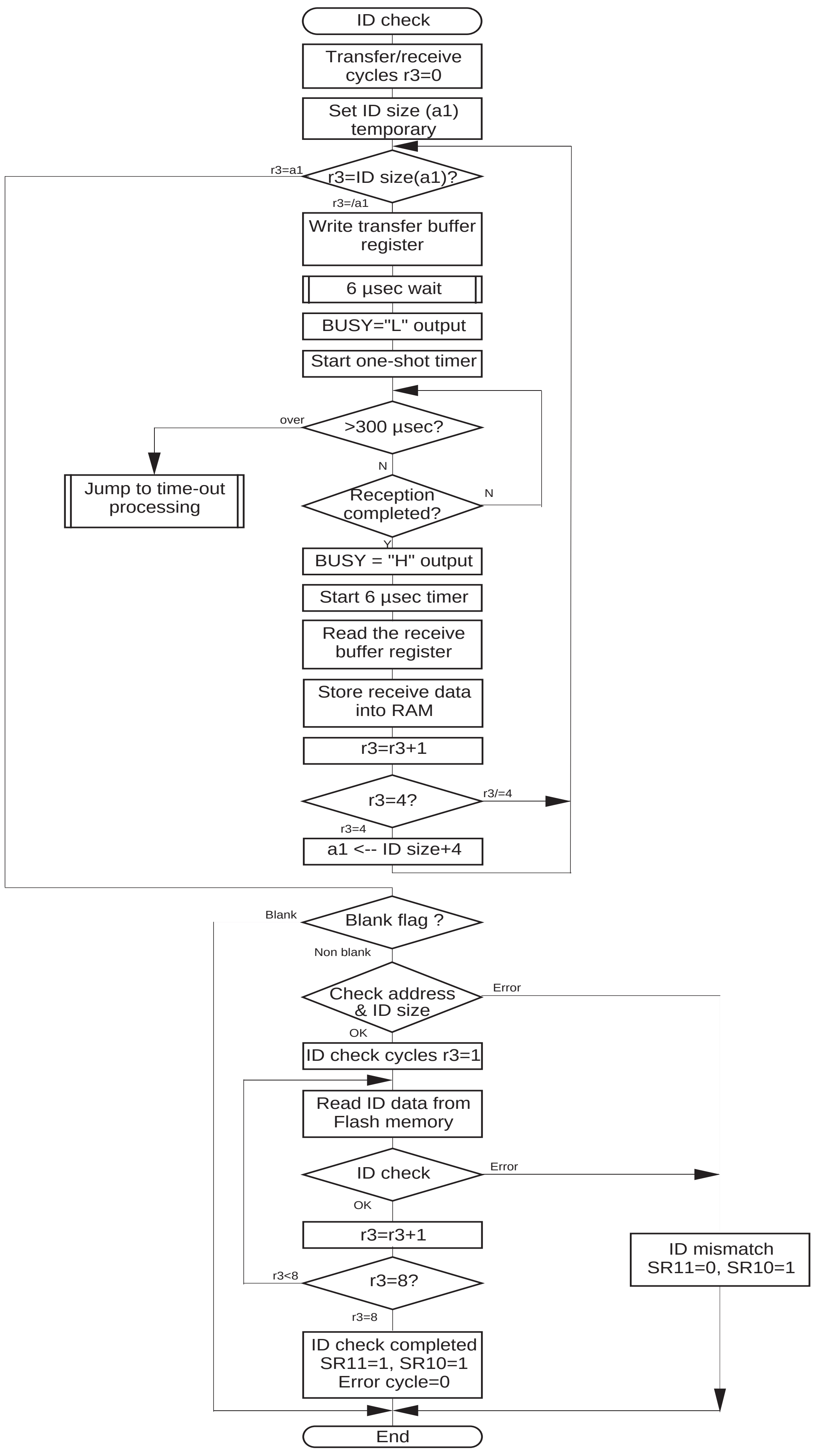


Figure 2.2.16 ID Check

Version Information Output

The version information of boot program is sent via serial I/O.
Version information is read and written in the transmit buffer register.
After all version information is send, processing jumps to main .
Figure 2.2.17 shows a processing flow.

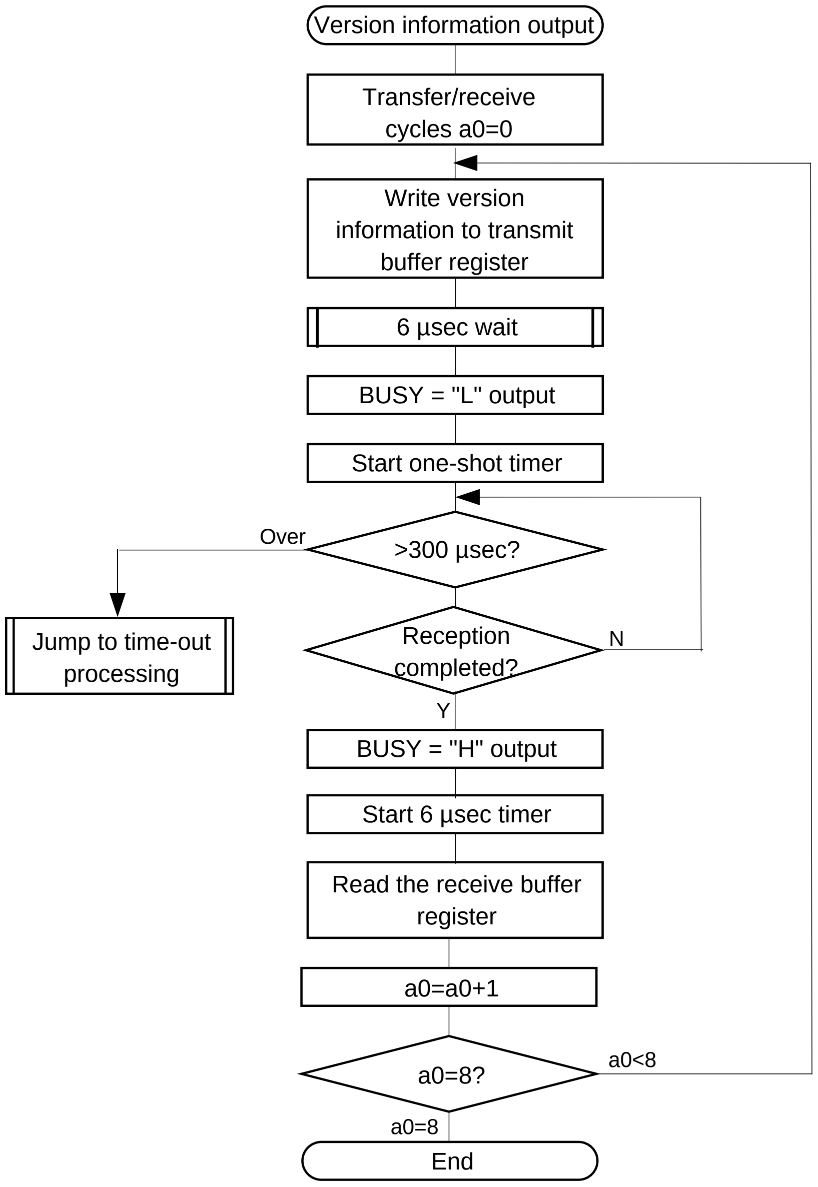

Version information output
Transfer/receive cycles a0=0
Write version information to transmit buffer register
6 µsec wait
BUSY = "L" output
Start one-shot timer
>300 µsec?
Over
Jump to time-out processing
Reception completed?
N
Y
BUSY = "H" output
Start 6 µsec timer
Read the receive buffer register
a0=a0+1
a0=8?
a0<8
a0=8
End

Figure 2.2.17 Version Information Output

Time-Out Processing

Time-out flag (SR9) is set to 1 and initiale setting 2 of main routine is executed again.
Figure 2.2.18 shows a processing flow.

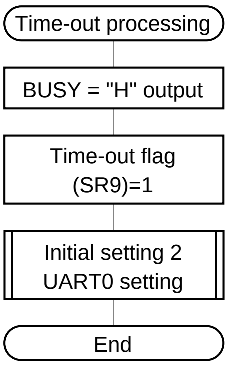


Figure 2.2.18 Time-Out Processing

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.
Table 2.2.2 shows the definition of each status register bit.

Table 2.2.2 Status Register (SRD)

Each bit of SRD	Status name	Definition	
		"1"	"0"
SR7 (bit7)	Status bit	Ready	Busy
SR6 (bit6)	Reserved	-	-
SR5 (bit5)	Erase bit	Terminated in error	Terminated normally
SR4 (bit4)	Program bit	Terminated in error	Terminated normally
SR3 (bit3)	Reserved	-	-
SR2 (bit2)	Reserved	-	-
SR1 (bit1)	Reserved	-	-
SR0 (bit0)	Reserved	-	-

State Bit (SR7)

The status bit indicates the operating status of the flash memory. When power is turned on, "1" (ready) is set for it.

Erase Bit (SR5)

The erase bit indicates the operating status of the auto erase operation. If an erase error occurs, it is set to "1". When the erase status is cleared, it 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". When the program status is cleared, it is set to "0".

Status Register 1 (SRD1)

Status register 1 indicates the status of serial communications, results from ID checks and results from check sum comparisons.
Table 2.2.3 gives the definition of each status register 1 bit. "00_{16}" is output when power is turned ON and the flag status is maintained even after the reset.

Table 2.2.3 Status Register 1 (SRD1)

Each bit of SRD1	Status name	Definition	
		"1"	"0"
SR15 (bit7)	Boot update completed bit	Update completed	Not update
SR14 (bit6)	Reserved	-	-
SR13 (bit5)	Reserved	-	-
SR12 (bit4)	Checksum match bit	Match	Mismatch
SR11 (bit3) SR10 (bit2)	ID check completed bits	00 Not verified 01 Verification mismatch 10 Reserved 11 Verified	
SR9 (bit1)	Data receive time out	Time out	Normal operation
SR8 (bit0)	Reserved	-	-

Boot Update Completed Bit (SR15)

This flag indicates whether the control program was downloaded to the RAM or not, using the download function.

Checksum Match Bit (SR12)

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

ID Check Completed Bits (SR11 and SR10)

These flags indicate the result of ID checks. Some commands cannot be accepted without an ID check.

Data Reception Time Out (SR9)

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

2.3 Sample List

This section shows a sample list of the program described in Section 2.2.
In addition to the processing explained in Section 2.2, the sample shown below includes the programmer command processing used by a synchronous serial programmer and the command processing used by an asynchronous serial communication programmer (M16C Flash Start).

Source

```
;********************************************************
;*  System Name   : Sample Program for M16C/20 Flash    *
;*  File Name     : M201SAMP.a30                        *
;*  MCU           : M30201F6                            *
;*  Xin           ; 2MHz - 10MHz (for UART mode)        *
;*  CPU           ; 1wait,f1                            *
;*  Assembler     : AS30 ver 3.00                       *
;*  Linker        : LN30 ver 3.00                       *
;*-----------------------------------------------------*
;*   Copyright,1999 MITSUBISHI ELECTRIC CORPORATION     *
;*        AND MITSUBISHI SYSTEM LSI DESIGN CORPORATION  *
;*-----------------------------------------------------*
;++++++++ Include file ++++++++++++++++++++++++++++++++++
;
    .list       off
    .include    sfr20.inc
    .include    flash201.inc
    .list       on
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Version table                                   +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
    .section    rom,code
    .org        Version
    .byte       'VER.0.02(VER.1.04)'
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Program section start                           +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
    .section    prog,code
    .org        Boot_TOP
    .sb         SB_base
    .sbsym      SRD
    .sbsym      SRD1
    .sbsym      ver
    .sbsym      SF
    .sbsym      addr_l
    .sbsym      addr_m
    .sbsym      addr_h
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Boot program start                              +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
Reset:
    ldc     #Istack,ISP          ; stack pointer set
    ldc     #SB_base,SB          ; SB register set
;
    mov.b   #00000001b,pd5       ; TxD-output,BUSY/RxD/CLK-input
    mov.b   #00000001b,p5        ; TxD-"H"output
;
;--------------------------------------
```

```
;+     Hot start & RAM clear          +
;-------------------------------------
    mov.w   #0,a0
Start_check:
    cmp.w   #55aah,buff[a0]
    jne     RAM_clear
    add.w   #2,a0
    cmp.w   #18,a0
    jltu    Start_check
    bset    ram_check           ; RAM check OK flag set
    jmp     CPU_set
;
RAM_clear:
    mov.w   #0,r0
    mov.w   #(Ram_END+1-Ram_TOP)/2,r3
    mov.w   #Ram_TOP,a1
    sstr.w
;
    mov.w   #0,a0
Buff_set:
    mov.w   #55aah,buff[a0]
    add.w   #2,a0
    cmp.w   #18,a0
    jltu    Buff_set
;
;-------------------------------------
;+   CPU set & Serial I/O mode check  +
;-------------------------------------
CPU_set:
    btst    busy
    bmc     s_mode
    bset    busy                ; BUSY-"H"output
    bset    busy_d
;
   jsr     Initialize_1
    mov.b   #80h,SRD
    and.b   #9eh,SRD1
    bset    sr7                 ; RADY
;
Reload_chack:
    btst    sr15                ; Update
    jc      Transfer_end
    btst    ram_check           ; Reload ?
    jz      Version_inf
    btst    s_mode
    bxor    old_mode
    jnc     Transfer_end
;
;-------------------------------------
;+     Version information            +
;-------------------------------------
Version_inf:
    mov.w   #0,a0               ; a0=0
Ver_loop:
    lde.w   Version+9[a0],ver[a0] ; Version data store
    add.w   #2,a0               ; address increment
    cmp.w   #8,a0               ; a0=8 ?
    jltu    Ver_loop            ; jump Ver_loop at a0<8
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Program_transfer clock synchronous mode         +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
    btst    s_mode              ; Serial I/O mode select
    jz      Transfer2           ; UART mode
;
```

```
Transfer1:
    bset    old_mode            ; clock synchronous mode
    mov.w   #(Trans_TOP1 & 0ffffh),a0
    mov.b   #(Trans_TOP1>> 16),r1h
    mov.w   #Ram_progTOP,a1
    mov.w   #(Trans_END1- Trans_TOP1 + 1)/2,r3
    smovf.w                     ; String move
    jmp     Transfer_end
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Program_transfer UART mode                       +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Transfer2:
    bclr    old_mode            ; UART mode
    mov.w   #(Trans_TOP2 & 0ffffh),a0
    mov.b   #(Trans_TOP2>> 16),r1h
    mov.w   #Ram_progTOP,a1
    mov.w   #(Trans_END2- Trans_TOP2 + 1)/2,r3
    smovf.w                     ; String move
Transfer_end:
;
;------------------------------------------------
;+     Jump to RAM                              +
;------------------------------------------------
    jmp     Ram_progTOP
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Subroutine : Initialize_1                        +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Initialize_1:
;--------------------------------------------------------
;+     Processor mode register                          +
;+        &  System clock control register              +
;--------------------------------------------------------
    mov.b   #3,prcr             ; Protect off
    mov.w   #8000h,pm0          ; 1wait
    mov.w   #2008h,cm0          ; f1 select
    mov.b   #0,prcr             ; Protect on
;
;--------------------------------------------------------
;+     ID address & size store                          +
;--------------------------------------------------------
    mov.w   #0ffdfh,ID          ; ID address 0fffdfh store
    mov.w   #0070fh,ID+2        ; ID size 7 store
;
    rts
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Subroutine : Download program                    +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    .org    Download_program
;
    mov.w   #0,r3               ; receive number (r3=0)
    mov.w   #0,a1               ; sumcheck buffer
    bclr    sr15                ; Download flag reset
    bclr    sr12                ; Check sum flag reset
Download_loop:
    mov.b   r1l,u0tb            ; data transfer
    jsr     wait_loop           ; BUSY "H" 6usec check
    bclr    busy                ; BUSY "L"
    bset    ta0os               ; 300usec start
?:
    btst    ir_ta0ic            ; Time over check
    bmc     sr9                 ; Time over flag set
    bmc     busy                ; BUSY  "H"
    jc      Version_inf         ; jump Version_inf at time out
    btst    ri_u0c1             ; Receive complete ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    bset    tb0s                ; 6usec timer start
    mov.w   u0rb,r0             ; receive data --> r0
    add.w   #1,r3               ; r3+1 increment
    cmp.w   #3,r3               ; r3=3 ?
    jgtu    Version_store       ; jump Version_store at r3>3
    mov.w   r3,a0               ; r3 --> a0
    mov.b   r0l,addr_l[a0]      ; Store program size
    mov.w   #0,a0               ; a0 initialize
    cmp.w   #3,r3               ; r3 = 3?
    jne     Download_loop       ; No,Download_loop
    cmp.w   #0,addr_m           ; program size = 0 ?
    jz      Version_inf         ; jump to Version_inf at program size error
    jmp     Download_loop       ; jump Download_loop
;
Version_store:
    cmp.w   #11,r3              ; r3 = 11?
    jgtu    Program_store       ; jump  Program_store at r3 > 11
    mov.b   r0l,ver[a0]         ; version data store to RAM
    jmp     Program_store_1
;
Program_store:
    mov.b   r0l,Ram_progTOP-8[a0]   ; program data store to RAM
Program_store_1:
    add.b   r0l,a1              ; add data to a1
    add.w   #1,a0               ; a0(download offset)+1 increment
    cmp.w   addr_m,a0           ; a0 = program size (addr_m,h)?
    jltu    Download_loop       ; jump  Download_loop at a0< program size
    jmp     SUM_Check           ; compare check sum
;;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Subroutine : Download program  UART              +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    .org    U_Download_program
;
    mov.w   #0,r3               ; receive number (r3=0)
    mov.w   #0,a1               ; sumcheck buffer
    bclr    sr15                ; Download flag reset
    bclr    sr12                ; Check sum flag reset
U_Download_loop:
    bclr    busy                ; BUSY  "L"
?:
    btst    ri_u0c1             ; Receive complete ?
    jnc     ?-
    bset    busy                ; BUSY  "H"
    mov.w   u0rb,r0             ; receive data --> r0
    add.w   #1,r3               ; r3+1 increment
    cmp.w   #3,r3               ; r3=3 ?
    jgtu    U_Version_store     ; jump Version_store at r3>3
    mov.w   r3,a0               ; r3 --> a0
    mov.b   r0l,addr_l[a0]      ; Store program size
    mov.w   #0,a0               ; a0 initialize
    cmp.w   #3,r3               ; r3 = 3 ?
    jne     U_Download_loop     ; No, jump U_Download_loop
    cmp.w   #0,addr_m           ; program size = 0 ?
    jz      Version_inf         ; jump to Version_inf at program size error
    jmp     U_Download_loop     ; jump Download_loop
;
U_Version_store:
    cmp.w   #11,r3              ; r3 = 11?
    jgtu    U_Program_store     ; jump  Program_store at r3 > 11
    mov.b   r0l,ver[a0]         ; version data store to RAM
    jmp     U_Program_store_1
;
U_Program_store:
    mov.b   r0l,Ram_progTOP-8[a0]   ; program data store to RAM
U_Program_store_1:
    add.b   r0l,a1              ; add data to a1
    add.w   #1,a0               ; a0(download offset)+1 increment
    cmp.w   addr_m,a0           ; a0 = program size (addr_m,h)?
    jltu    U_Download_loop     ; jump  Download_loop at a0< program size
SUM_Check:
    mov.w   a1,r0
    cmp.b   data,r0l            ; compare check sum
    bmeq    sr12                ; check sum flag set at data=r0l
    jne     Version_inf         ; jump Version_inf at check sum error
    bset    sr15                ; Download flag set
    jmp     Ram_progTOP         ; jump Ram_progTOP
;
;===========================================================
;+  Transfer Program -- clock synchronous serial I/O mode +
;+      (1) Main flow                                      +
;+      (2) Flash control program                          +
;+            Read,Program,Erase,All_erase,etc.            +
;+      (3) Other program                                  +
;+            ID_check,Download,Version_output etc.        +
;===========================================================
Trans_TOP1:
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     main program                                      +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Main:
    jsr     Initialize_2        ; Initialize_2
Loop_main:
    bset    ta0os               ; 300usec timer start
    mov.b   #0,ta0ic
?:
    btst    ir_ta0ic            ; 300usec?
    jz      ?-
    mov.b   #0,ta0ic            ; Interrupt request bit clear
    mov.b   #0ffh,r1l           ; #ffh --> r1l (transfer data);
    mov.b   r1l,u0tb            ; dummy data --> transfer buffer
    bclr    busy                ; BUSY  "L"
?:
    btst    ti_u0c1             ; Transmit buffer empty ?
    jz      ?-
    bset    ta0os               ; 300É sec timer start
?:
    btst    ir_ta0ic            ; 300É sec ?
    jc      Time_out            ; jump Time_out at time out
    btst    ri_u0c1             ; receive complete ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    mov.b   #0,tb0ic            ;
    bset    tb0s                ; 6usec timer start
    mov.b   #0,ta0ic            ; Interrupt request bit clear
    mov.w   u0rb,r0             ; receive data --> r0
    mov.b   #0ch,r0h            ; #00001100b sr10,11 mask data
    and.b   SRD1,r0h            ; sr10,11 pick up
    cmp.b   #0ch,r0h            ; ID check OK?
    jne     Command_check_2     ; jump Command_check_2 at ID uncheck
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 RBS      (71h)
    jeq     Read_RB
    cmp.b   #077h,r0l          ; RB program    (77h)
    jeq     Program_RB
    cmp.b   #07ah,r0l          ; RB enable     (7ah)
    jeq     Loop_main
    cmp.b   #075h,r0l          ; RB disable    (75h)
    jeq     Loop_main
    cmp.b   #0fah,r0l          ; Download      (fah)
    jeq     Download
    cmp.b   #0fch,r0l          ; Boot output   (fch)
    jeq     Boot_output
Command_check_2:
    cmp.b   #070h,r0l          ; Read SRD      (70h)
    jeq     Read_SRD
    cmp.b   #0f5h,r0l          ; ID check      (f5h)
    jeq     ID_check
    cmp.b   #0fbh,r0l          ; Version out   (fbh)
    jeq     Ver_output
Command_err:
    jsr     Initialize_21      ; Command error,UART reset
    jmp     Loop_main
;
;---------------------------------------------------------
;+     Read / Boot output                                +
;---------------------------------------------------------
Boot_output:
    bclr    fcon00             ; not CPU write mode
Read:
    mov.w   #0,r3              ; receive number (r3=0)
    mov.b   #0,addr_l          ; addr_l = 0
Read_loop:
    mov.b   r1l,u0tb           ; data transfer
    jsr     wait_loop          ; BUSY"H" 6usec check
    bclr    busy               ; BUSY "L"
    bset    ta0os              ; 300usec start
?:
    btst    ir_ta0ic           ; Time over check
    jc      Time_out           ; jump Time_out at time out
    btst    ri_u0c1            ; Receive complete ?
    jz      ?-
    bset    busy               ; BUSY  "H"
    bset    tb0s               ; 6usec timer start
    mov.w   u0rb,r0            ; receive data --> r0
    add.w   #1,r3              ; r3+1 increment
    cmp.w   #2,r3              ; r3 = 2 ?
    jgtu    Read_data          ; jump Read_data at r3>2
    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
    mov.b   #00h,fcmd          ; Read command
Read_data:
    lde.b   [a1a0],r1l         ; Flash memory read
    add.w   #1,a0              ; a0+1 increment
    cmp.w   #258,r3            ; r3 = 258 ?
    jne     Read_loop          ; jump Read_loop at r3<258
Read_end:
    bset    fcon00              ; CPU write mode
    jmp     Loop_main           ; jump Loop_main
;
;---------------------------------------------------------
;+     Program                                          +
;---------------------------------------------------------
Program:
    mov.w   #1,r3               ; receive number (r3=1)
    mov.b   #0,addr_l           ; addr_l = 0
Program_loop:
    mov.b   r1l,u0tb            ; data transfer
    jsr     wait_loop           ; BUSY "H" 6usec check
    bclr    busy                ; BUSY  "L"
    bset    ta0os               ; 300usec start
?:
    btst    ir_ta0ic            ; Time over check
    jc      Time_out            ; jump Time_out at time out
    btst    ri_u0c1             ; Receive complete ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    bset    tb0s                ; 6usec timer start
    mov.w   u0rb,r0             ; receive data --> r0
    mov.w   r3,a0               ; r3 --> a0
    mov.b   r0l,addr_l[a0]      ; Store address
    add.w   #1,r3               ; r3+1 increment
    cmp.w   #259,r3             ; r3 = 259 ?
    jltu    Program_loop        ; jump Program_loop at r3<259
    btst    fcon02              ; Vpp 12V input ?
    jz      Program_err
    mov.w   #0,r3               ; writing number (r3=0)
Address_check:
    mov.w   addr_m,a0           ; addr_m,h --> a0
    cmp.w   #0f40h,a0           ; compare f4000h
    jltu    Address_err         ; jump Address_err  at < f4000h
    cmp.w   #1000h,a0           ; compare 100000h
    jltu    Program_loop2       ; jump  Program_loop2
Address_err:
    bset    sr8                 ; address error
    jmp     Program_end         ; jump Program_end at address error
Program_loop2:
    mov.w   r3,a0               ; r3       --> a0
    mov.b   data[a0],r0l        ; data     --> r0l
    mov.w   addr_l,a0           ; addr_l,m --> a0
    mov.b   addr_h,a1           ; addr_h   --> a1
;
    mov.w   #25,r2              ; retry number
    bclr    ta0os               ; timer stop
    mov.w   #200-1,ta0          ; 20usec timer set
Byte_loop:
    mov.b   #40h,fcmd           ; Program command
    ste.b   r0l,[a1a0]          ; data write
    bset    ta0os               ; 20usec timer start
?:
    btst    ir_ta0ic
    jz      ?-
    mov.b   #0,ta0ic
    mov.b   #0c0h,fcmd          ; Verify command
    jsr     wait_6usec          ; wait 6usec
    lde.b   [a1a0],r0h          ; data read
    cmp.b   r0l,r0h             ; compare write and read data
    jeq     Byte_end            ; when equal,jump to Byte_end
    sbjnz.w #1,r2,Byte_loop     ; to Byte_loop
Program_err:
    bset    sr4                 ; error flag set
    jmp     Program_end
```

```
Byte_end:
    add.w   #1,addr_l           ; address increment
    add.w   #1,r3               ; write number increment
    cmp.w   #256,r3             ; 256 times writing ?
    jltu    Program_loop2       ; jump Program_loop2 at r3<256
Program_end:
    mov.w   #3000-1,ta0         ; 300 usec set
    mov.b   #0ffh,fcmd          ; reset command
    mov.b   #0ffh,fcmd
    mov.b   #00h,fcmd           ; Read command
    jmp     Loop_main           ; jump Loop_main
;
;-------------------------------------------------------
;+     All erase                                       +
;-------------------------------------------------------
All_erase:
    mov.b   r1l,u0tb            ; data transfer
    jsr     wait_loop           ; BUSY "H" 6sec check
    bclr    busy                ; BUSY  "L"
    bset    ta0os               ; 300usec start
?:
    btst    ir_ta0ic            ; Time over check
    jc      Time_out            ; jump Time_out at time out
    btst    ri_u0c1             ; Receive complete ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    mov.w   u0rb,r0             ; receive data --> r0
    cmp.b   #0d0h,r0l           ; Confirm command check
    jne     Erase_err           ; jump  Erase_end at Confirm command error
;
    mov.w   #4000h,a0           ; start address f4000h
    mov.w   #0fh,a1
    mov.w   a0,data
    mov.w   #0ffffh,addr_l      ; end address fffffh
;
Zero_clear:
    btst    fcon02              ; Vpp 12V input ?
    jz      Erase_err
    mov.b   #00h,r0l            ; write data "00h"
    mov.w   #25,r2              ; retry counter
    bclr    ta0os               ; timer stop
    mov.w   #200-1,ta0          ; 20usec set
Zero_clear1:
    mov.b   #40h,fcmd           ; Program command
    ste.b   r0l,[a1a0]          ; "00h" write
    bset    ta0os               ; 20usec timer start
?:
    btst    ir_ta0ic
    jz  ?-
    mov.b   #0,ta0ic
    mov.b   #0c0h,fcmd          ; program verify command
    jsr     wait_6usec          ; wait 6usec
    lde.b   [a1a0],r0h          ; data read
    cmp.b   r0l,r0h             ; compare write and read data
    jeq     Zero_clear2         ; when equal , jump to Zero_clear2
    sbjnz.w #1,r2,Zero_clear1   ; to Zero_clear1
    jmp     Erase_err           ; jump to Erase_err
Zero_clear2:
    mov.w   #25,r2              ; retry counter
    cmp.w   addr_l,a0           ; end address ?
    jeq     Erase_verify        ; jump to Erase_verify
    add.w   #1,a0
    jmp     Zero_clear1
;
Erase_verify:
    mov.w   #1000,r2            ; retry 1000 times
    bclr    ta0os               ; timer stop
    mov.w   #3000-1,ta0         ; 300usec set
Erase_verify2:
    mov.b   #20h,fcmd           ; Auto erase command
    mov.b   #20h,fcmd           ; Auto erase command
    mov.b   #0,ta0ic
    mov.w   #0,r1
    bset    ta0os               ; 300usec timer start
?:
    btst    ir_ta0ic
    jz      ?-
    add.b   #1,r1l
    cmp.b   #66,r1l             ; 20msec ?
    jeq     ?+
    mov.b   #0,ta0ic
    bset    ta0os
    jmp     ?-
?:
Erase_verify3:
    mov.b   #0a0h,fcmd          ; erase verify command
    jsr     wait_6usec          ; wait 6usec
    lde.b   [a1a0],r0h          ; data read
    cmp.b   #0ffh,r0h           ; "ffh" ?
    jeq     Erase_verify4       ; when equal , jump to Erase_verify2
    sbjnz.w #1,r2,Erase_verify2 ; to Erase_verify
    jmp     Erase_err           ; jump to Erase_err
Erase_verify4:
    cmp.w   data,a0             ; start address ?
    jeq     Erase_end           ; jump to Erase_end
    sub.w   #1,a0
    jmp     Erase_verify3
Erase_err:
    bset    sr5                 ; erase error flag set
Erase_end:
    mov.b   #0ffh,fcmd          ; reset command
    mov.b   #0ffh,fcmd
    mov.b   #00h,fcmd           ; read command
    jmp     Loop_main           ; jump Loop_main
;
;-------------------------------------------------------
;   Read SRD
;-------------------------------------------------------
Read_SRD:
    mov.w   #0,r3
    mov.b   SRD,r1l
    mov.b   SRD1,r1h
;
Read_SRD_loop:
    mov.b   r1l,u0tb            ; data transfer
    jsr     wait_loop           ; BUSY "H" 6usec check
    bclr    busy                ; BUSY "L"
    bset    ta0os               ; 300usec start
?:
    btst    ir_ta0ic            ; Time over check
    jc      Time_out            ; jump Time_out at time out
    btst    ri_u0c1             ; Receive complete ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    bset    tb0s                ; 6usec timer start
    mov.w   u0rb,r0             ; receive data --> r0
    add.w   #1,r3
    mov.b   r1h,r1l
    cmp.w   #2,r3
    jne     Read_SRD_loop
```

```
Read_SRD_end:
    jmp     Loop_main
;
;-------------------------------------------------------
;+      Clear SRD                                      +
;-------------------------------------------------------
Clear_SRD:
    mov.b   #80h,SRD            ; SRD clear
    and.b   #9ch,SRD1           ; SRD1 clear
    jmp     Loop_main
;
;-------------------------------------------------------
;+      Block Erase / Read Rock Bit                    +
;+      / Program Rock Bit (dummy)                     +
;-------------------------------------------------------
Erase:
Read_RB:
Program_RB:
    mov.w   #0,r3
    mov.b   #0ffh,r1l
Read_RB_loop:
    mov.b   r1l,u0tb            ; data transfer
    jsr     wait_loop           ; BUSY "H" 6usec check
    bclr    busy                ; BUSY "L"
    bset    ta0os               ; 300usec start
?:
    btst    ir_ta0ic            ; Time over check
    jc      Time_out            ; jump Time_out at time out
    btst    ri_u0c1             ; Receive complete ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    bset    tb0s                ; 6usec timer start
    mov.w   u0rb,r0             ; receive data --> r0
    add.w   #1,r3
    cmp.w   #3,r3
    jltu    Read_RB_loop
Read_RB_end:
    jmp     Loop_main
;
;-------------------------------------------------------
;   ID check
;-------------------------------------------------------
ID_check:
    mov.w   #0,r3               ; receive number (r3=0)
    mov.w   #0ffh,a1            ; ID size (dummy data = ffh)
ID_data_store:
    cmp.w   a1,r3               ; r3=a1(ID size)
    jeq     ID_address_check    ; jump ID_address_check at r3=ID size
    mov.b   r1l,u0tb            ; data transfer
    jsr     wait_loop           ; BUSY "H" 6usec check
    bclr    busy                ; BUSY "L"
    bset    ta0os               ; 300usec start
?:
    btst    ir_ta0ic            ; Time over check
    jc      Time_out            ; jump Time_out at time out
    btst    ri_u0c1             ; Receive complete ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    bset    tb0s                ; 6usec timer start
    mov.w   u0rb,r0             ; receive data --> 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 ?
    jne     ID_data_store       ; jump ID_data_store at r3 not= 4
    mov.b   data,a1             ; ID size --> a1
    add.w   #4,a1               ; a1=a1+4
    jmp     ID_data_store       ; jump ID_data_store
ID_address_check:
    btst    blank               ; blank flag check
    jc      ID_check_end        ; jump ID_check_end at blank
    cmp.w   #0ffdfh,addr_l      ; lower ID address check
    jne     ID_error            ; jump ID_error at ID address error
    cmp.w   #0070fh,addr_h      ; higher ID address check
    jne     ID_error            ; jump ID_error at ID address error
ID_data_check:
    mov.w   #0000fh,a1          ; ID higher address --> a1
    mov.w   #0ffdfh,r1          ; ID lower address --> r1
    mov.w   #1,r3               ; check loop number (r3=1)
ID_check_loop:
    mov.w   r1,a0               ; r1 --> a0
    lde.b   [a1a0],r0l          ; ID data read from Flash memory
    mov.w   r3,a0               ; r3 --> a0
    cmp.b   r0l,data[a0]        ; compare ID data
    jne     ID_error            ; jump ID_error at ID error
    add.w   #4,r1               ; r1+4 increment (next ID address)
    cmp.w   #0ffe7h,r1          ; r1=0ffefh ?
    jne     ?+                  ; jump ? at not equal
    mov.w   #0ffebh,r1          ; r1=0ffeb at equal
?:
    add.w   #1,r3               ; r3 +1 increment
    cmp.w   #8,r3               ; r3=8 ?
    jltu    ID_check_loop       ; jump ID_check_loop at r3<8
ID_OK:
    bset    sr10
    bset    sr11                ; ID check OK (sr11=1,sr10=1)
    jmp     ID_check_end        ; jump  ID_check_end
ID_error:
    bset    sr10
    bclr    sr11                ; ID error (sr11=0,sr10=1)
ID_check_end:
    jmp     Loop_main           ; jump Loop_main
;
;-------------------------------------------------------
;     Download
;-------------------------------------------------------
Download:
    bclr    fcon00              ; not CPU write mode
    jmp.a   Download_program    ; jump Download_program
;
;--------------------------------------------------------
;+      Version output                                  +
;--------------------------------------------------------
Ver_output:
    mov.w   #0,a0
Ver_output_loop:
    lde.b   ver[a0],r1l
    mov.b   r1l,u0tb            ; data transfer
    jsr     wait_loop           ; BUSY "H" 6usec check
    bclr    busy                ; BUSY  "L"
    bset    ta0os               ; 300usec start
?:
    btst    ir_ta0ic            ; Time over check
    jc      Time_out            ; jump Time_out at time out
    btst    ri_u0c1             ; Receive complete ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    bset    tb0s                ; 6usec timer start
    mov.w   u0rb,r0             ; receive data --> r0
    add.w   #1,a0
    cmp.w   #8,a0
    jltu    Ver_output_loop
Ver_output_end:
    jmp     Loop_main
;
;--------------------------------------------------------
;+      6usec timer wait                               +
;--------------------------------------------------------
wait_6usec:
    bset    tb0s
wait_loop:
    btst    ir_tb0ic
    jz      wait_loop
    bclr    tb0s
    mov.b   #0,tb0ic
    rts
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Subroutine : Initialize_2                         +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Initialize_2:
    bset    fcon00              ; CPU write mode
    bset    fcon05              ; F4000h-FFFFFh select
    bclr    fcon04
    lde.w   0ffffch,r0          ; Reset vector read --> r0
    lde.w   0ffffeh,r1          ; Reset vector read --> r1
    and.w   r1,r0               ; r0 & r1
    cmp.w   #0ffffh,r0          ; Blank check
    jne     Blank_check_end     ; jump Blank_check_end at Blank error
    bset    sr10                ; check complete
    bset    sr11                ;
    bset    blank               ; blank flag set
Blank_check_end:
;
;--------------------------------------------------------
;+     UART0                                           +
;--------------------------------------------------------
Initialize_21:
;----- UART0 transmit/receive mode register
;
    mov.b   #0,u0c1             ; UART0 reset
    mov.b   #0,u0mr
    mov.b   #0,u0c0
;
    mov.b   #00001001b,u0mr
;            |||||+++------------ clock synchronous SI/O
;            ||||+--------------- external clock
;            ++++---------------- fixed
;
;----- UART0 transmit/receive control register 0
;
    mov.b   #00000100b,u0c0
;            |||| |++------------ 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
;
;----- UART0 transmit/receive control register 1
;
    mov.b   #00000101b,u0c1
;            |||| | +------------ Transmission enabled
;            |||| +-------------- Reception enabled
;            +++++--------------- fixed
;
;---------------------------------------------------------
;+     Timer                                             +
;---------------------------------------------------------
    mov.b   #02h,ta0mr          ; f1 select,one-shot mode
    mov.b   #0,ta0ic            ; Interrupt flag clear
    mov.w   #3000-1,ta0         ; 300usec at 10 MHz
    bset    ta0s
;
    mov.b   #00h,tb0mr          ; f1 select
    mov.w   #60-1,tb0           ; 6usec at 10 MHz
;
    rts
;
;-----------------------------------------------------------
;+     Time_out                                        +
;-----------------------------------------------------------
Time_out:
    bset    busy                ; BUSY "H"
    bset    sr9                 ; SRD1 time out flag set
    jmp     Command_err         ; jump Command_err at time out
;
Trans_END1:
;-----------------------------------------------------------
;===========================================================
;+  Transfer Program -- UART mode                          +
;+       (1) Main flow                                     +
;+       (2) Flash control program                         +
;+             Read,Program,All_erase,Read_SRD,Clear_SRD  +
;+       (3) Other program                                 +
;+             ID_check                                    +
;===========================================================
;
Trans_TOP2:
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     main program                                      +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_Main:
    bclr    freq_set0           ; freq set flag clear
    bclr    freq_set1
    bclr    freq_set2
    mov.b   #64,data            ; 9600bps for 10MHz
    jsr     U_Initialize_2      ; Initialize_2
    bset    re_u0c1             ; Reception enabled
    bclr    busy                ; BUSY  "L"
?:
    btst    ri_u0c1             ; receive complete ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    mov.w   u0rb,r0             ; receive data --> r0
    cmp.b   #0b0h,r0l
    jne     U_Freq_Get
;
;------ 10MHz --------------;
```

```
        mov.b   #64,baud            ; 9600bps
        mov.b   #32,baud+1          ; 19200bps
        mov.b   #15,baud+2          ; 38400bps
        mov.b   #10,baud+3          ; 57600bps
        mov.b   baud,data
        mov.b   data,u0brg          ; Transmission late
        bset    freq_set0           ; "B0h" get flag set
        jsr     U_BPS_B0
        jmp     U_Loop_main
;
U_Freq_Get:
        mov.b   #80h,data
        mov.b   #01000000b,r1l      ; counbter1,2 reset
        mov.b   #10000000b,r1h
        mov.b   data,u0brg          ; Transmission late
;
;---------------------------;
U_Loop_main:
        btst    txept_u0c0          ; Transmit register empty ?
        jnc     U_Loop_main
;
        bclr    te_u0c1             ; Transmission disabled
        bset    re_u0c1             ; Reception enabled
        bclr    busy                ; BUSY  "L"
?:
        btst    sum_u0rb            ; Error sum flag check
        jc      U_RESET
U_Loop_bak:
        btst    ri_u0c1             ; receive complete ?
        jz      ?-
        bset    busy                ; BUSY  "H"
        mov.w   u0rb,r0             ; receive data --> r0
        btst    freq_set2           ; freq fixed ?
        jc      U_Command_check     ; jump Command_check_2 at data
        btst    freq_set0
        jz      U_Freq_check        ; jump U_Freq_check
        cmp.b   #00h,r0l            ; "00h" get?
        bmgtu   freq_set2
        jne     U_Command_check     ; jump U_Freq_check
        bclr    freq_set0
        mov.b   #0ffh,r0l           ; dummy data set
        mov.b   #01000000b,r1l      ; counbter1,2 reset
        mov.b   #10000000b,r1h
        mov.b   #80h,data
        jmp     U_Freq_check
;
U_Command_check:
        mov.b   #0ch,r0h            ; #00001100b sr10,11 mask data
        and.b   SRD1,r0h            ; sr10,11 pick up
        cmp.b   #0ch,r0h            ; ID check OK?
        jne     U_Command_check_2   ; jump Command_check_2 at ID uncheck
U_Command_check_1:
        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 RBS      (71h)
        jeq     U_Read_RB
        cmp.b   #077h,r0l           ; RB program    (77h)
```

```
    jeq     U_Program_RB
    cmp.b   #07ah,r0l           ; RB enable     (7ah)
    jeq     U_Loop_main
    cmp.b   #075h,r0l           ; RB disable    (75h)
    jeq     U_Loop_main
    cmp.b   #0fah,r0l           ; Download      (fah)
    jeq     U_Download
    cmp.b   #0fch,r0l           ; Boot output   (fch)
    jeq     U_Boot_output
U_Command_check_2:
    cmp.b   #070h,r0l           ; Read SRD      (70h)
    jeq     U_Read_SRD
    cmp.b   #0f5h,r0l           ; ID check      (f5h)
    jeq     U_ID_check
    cmp.b   #0fbh,r0l           ; Version out   (fbh)
    jeq     U_Ver_output
    cmp.b   #0b0h,r0l           ; Baud rate 9600bp (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
U_Command_err:
    jsr     U_Initialize_21     ; command error,UART Initialize
    jmp     U_Loop_main
;
U_RESET:
    mov.b   #0,u0mr             ; u0mr reset
    mov.b   #00000101b,u0mr
    jmp     U_Loop_bak
;---------------------------------------------------------
;+     UART Read / Boot output                          +
;---------------------------------------------------------
U_Boot_output:
    bclr    fcon00              ; not CPU write mode
U_Read:
    mov.w   #0,r3               ; receive number (r3=0)
    mov.b   #0,addr_l           ; addr_l = 0
U_Read_loop:
    bclr    busy                ; BUSY "L"
?:
    btst    ri_u0c1             ; Receive complete ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    mov.w   u0rb,r0             ; receive data --> r0
    add.w   #1,r3               ; r3+1 increment
    cmp.w   #2,r3               ; r3 = 2 ?
    jgtu    U_Read_data           ; jump U_Read_data at r3>2
    mov.w   r3,a0               ; r3 --> a0
    mov.b   r0l,addr_l[a0]      ; Store address
    cmp.w   #2,r3               ; r3 = 2 ?
    jltu    U_Read_loop           ; jump U_Read_loop at r3<2
    mov.w   addr_l,a0           ; addr_l,m --> a0
    mov.b   addr_h,a1           ; addr_h   --> a1
    mov.b   #00h,fcmd           ; Read command
    bclr    re_u0c1             ; Reception disabled
    bset    te_u0c1             ; Transmission enabled

U_Read_data:
    cmp.w   #258,r3             ; r3 = 258 ?
    jz     U_Read_end           ; jump U_Read_loop at r3=258
    lde.b   [a1a0],r1l          ; Flash memory read
    mov.b   r1l,u0tb            ; r1l --> transmit buffer register
```

```
?:
    btst    ti_u0c1             ; transmit buffer empty ?
    jnc     ?-
    add.w   #1,a0               ; address increment
    add.w   #1,r3               ; counter increment
    jmp     U_Read_data           ; jump Read_data
;
U_Read_end:
    bset    fcon00              ; CPU write mode
    jmp     U_Loop_main         ; jump Loop_main
;
;---------------------------------------------------------
;+     Program                                           +
;---------------------------------------------------------
U_Program:
    mov.w   #1,r3               ; receive number (r3=1)
    mov.b   #0,addr_l           ; addr_l = 0
U_Program_loop:
    bclr    busy                ; BUSY  "L"
?:
    btst    ri_u0c1             ; Receive complete ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    mov.w   u0rb,r0             ; receive data --> r0
    mov.w   r3,a0               ; r3 --> a0
    mov.b   r0l,addr_l[a0]      ; Store address
    add.w   #1,r3               ; r3+1 increment
    cmp.w   #259,r3             ; r3 = 259 ?
    jltu    U_Program_loop      ; jump Program_loop at r3<259
    btst    fcon02              ; Vpp 12V input ?
    jz      U_Program_err
    mov.w   #0,r3               ; writing number (r3=0)
U_Address_check:
    mov.w   addr_m,a0           ; addr_m,h --> a0
    cmp.w   #0f40h,a0           ; compare f4000h
    jltu    U_Address_err       ; jump U_Address_err at < f4000h
    cmp.w   #1000h,a0           ; compare 100000h
    jltu    U_Program_loop2     ; jump U_Program_loop2
U_Address_err:
    bset    sr8                 ; address error
    jmp     U_Program_end       ; jump U_Program_end at address error
U_Program_loop2:
    mov.w   r3,a0               ; r3       --> a0
    mov.b   data[a0],r0l        ; data     --> r0l
    mov.w   addr_l,a0           ; addr_l,m --> a0
    mov.b   addr_h,a1           ; addr_h   --> a1
;
    mov.w   #25,r2              ; retry number
    mov.w   #200-1,ta0          ; 20usec timer set
U_Byte_loop:
    mov.b   #40h,fcmd           ; Program command
    ste.b   r0l,[a1a0]          ; data write
    bset    ta0os               ; 20usec timer start
?:
    btst    ir_ta0ic
    jz      ?-
    mov.b   #0,ta0ic
    mov.b   #0c0h,fcmd          ; Verify command
    jsr     U_wait_6usec        ; wait 6usec
    lde.b   [a1a0],r0h          ; data read
    cmp.b   r0l,r0h             ; compare write and read data
    jeq     U_Byte_end          ; when equal,jump to Byte_end
    sbjnz.w #1,r2,U_Byte_loop   ; to Byte_loop
U_Program_err:
    bset    sr4                 ; error flag set
    jmp     U_Program_end
U_Byte_end:
    add.w   #1,addr_l           ; address increment
    add.w   #1,r3               ; write number increment
    cmp.w   #256,r3             ; 256 times writing ?
    jltu    U_Program_loop2     ; jump Program_loop2 at r3<256
U_Program_end:
    mov.b   #0ffh,fcmd          ; reset command
    mov.b   #0ffh,fcmd
    mov.b   #00h,fcmd           ; Read command
    jmp     U_Loop_main         ; jump Loop_main
;
;---------------------------------------------------------
;+     Erase : Block Erase/ Rock Bit Program (dummy)    +
;---------------------------------------------------------
U_Erase:
U_Program_RB:
    mov.w   #1,r3               ; receive number (r3=1)
U_Erase_loop:
    bclr    busy                ; BUSY  "L"
?:
    btst    ri_u0c1             ; Receive complete ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    mov.w   u0rb,r0             ; receive data --> r0
    mov.w   r3,a0               ; r3 --> a0
    add.w   #1,r3               ; r3+1 increment
    cmp.w   #4,r3               ; r3 = 4 ?
    jltu    U_Erase_loop        ; jump Erase_loop at r3<4
    jmp     U_Loop_main         ; jump U_Loop_main
;
;---------------------------------------------------------
;+     All erase                                        +
;---------------------------------------------------------
U_All_erase:
    bclr    busy                ; BUSY  "L"
?:
    btst    ri_u0c1             ; Receive complete ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    mov.w   u0rb,r0             ; receive data --> r0
    cmp.b   #0d0h,r0l           ; Confirm command check
    jne     U_Erase_err         ; jump  Erase_end at Confirm command error
;
     mov.w   #4000h,a0          ; start address f4000h
     mov.w   #0fh,a1
     mov.w   a0,data
     mov.w   #0ffffh,addr_l     ; end address fffffh
;
U_Zero_clear:
    btst    fcon02              ; Vpp 12V input ?
    jz      U_Erase_err
    mov.b   #00h,r0l            ; write data "00h"
    mov.w   #25,r2              ; retry counter
    bclr    ta0os               ; timer stop
    mov.w   #200-1,ta0          ; 20usec set
U_Zero_clear1:
    mov.b   #40h,fcmd           ; Program command
    ste.b   r0l,[a1a0]          ; "00h" write
    bset    ta0os               ; 20usec timer start
?:
    btst    ir_ta0ic
    jz  ?-
    mov.b   #0,ta0ic
    mov.b   #0c0h,fcmd          ; program verify command
    jsr     U_wait_6usec        ; wait 6usec
    lde.b   [a1a0],r0h          ; data read
    cmp.b   r0l,r0h             ; compare write and read data
    jeq     U_Zero_clear2       ; when equal , jump to Zero_clear2
    sbjnz.w #1,r2,U_Zero_clear1 ; to Zero_clear1
    jmp     U_Erase_err         ; jump to Erase_err
U_Zero_clear2:
    mov.w   #25,r2              ; retry counter
    cmp.w   addr_l,a0           ; end address ?
    jeq     U_Erase_verify      ; jump to Erase_verify
    add.w   #1,a0
    jmp     U_Zero_clear1
;
U_Erase_verify:
    mov.w   #1000,r2            ; retry 1000 times
    mov.w   #3000-1,ta0         ; 300u set
U_Erase_verify2:
    mov.b   #20h,fcmd           ; Auto erase command
    mov.b   #20h,fcmd           ; Auto erase command
    mov.b   #0,ta0ic
    mov.w   #0,r1
    bset    ta0os               ; 300usec timer start
?:
    btst    ir_ta0ic
    jz      ?-
    add.b   #1,r1l
    cmp.b   #66,r1l             ; 20msec ?
    jeq     ?+
    mov.b   #0,ta0ic
    bset    ta0os
    jmp     ?-
?:
U_Erase_verify3:
    mov.b   #0a0h,fcmd          ; erase verify command
    jsr     U_wait_6usec        ; wait 6usec
    lde.b   [a1a0],r0h          ; data read
    cmp.b   #0ffh,r0h           ; "ffh" ?
    jeq     U_Erase_verify4     ; when equal , jump to Erase_verify2
    sbjnz.w #1,r2,U_Erase_verify2 ; to Erase_verify
    jmp     U_Erase_err         ; jump to Erase_err
U_Erase_verify4:
    cmp.w   data,a0             ; start address ?
    jeq     U_Erase_end         ; jump to Erase_end
    sub.w   #1,a0
    jmp     U_Erase_verify3
U_Erase_err:
    bset    sr5                 ; erase error flag set
U_Erase_end:
    mov.b   #0ffh,fcmd          ; reset command
    mov.b   #0ffh,fcmd
    mov.b   #00h,fcmd           ; read command
    jmp     U_Loop_main           ; jump Loop_main
;
;-------------------------------------------------------
;   Read SRD                                           +
;-------------------------------------------------------
U_Read_SRD:
    bclr    re_u0c1             ; Reception disabled
    mov.w   #0,r3
    mov.b   SRD,r1l
    mov.b   SRD1,r1h
    bset    te_u0c1             ; Transmission enable
;
```

```
U_Read_SRD_loop:
    mov.b   r1l,u0tb            ; r1l --> transmit buffer register
?:
    btst    ti_u0c1             ; transmit  buffer empty ?
    jz      ?-
    add.w   #1,r3
    mov.b   r1h,r1l
    cmp.w   #2,r3
    jne     U_Read_SRD_loop
    jmp     U_Loop_main
;
;-------------------------------------------------------
;+      Clear SRD                                      +
;-------------------------------------------------------
U_Clear_SRD:
    mov.b   #80h,SRD            ; SRD clear
    and.b   #9ch,SRD1           ; SRD1 clear
    jmp     U_Loop_main
;
;-------------------------------------------------------
;+      Read Rock Bit (dummy)                           +
;-------------------------------------------------------
U_Read_RB:
    mov.w   #0,r3
    mov.b   #0ffh,r1l
U_Read_RB_loop:
    bclr    busy                ; BUSY  "L"
?:
    btst    ri_u0c1             ; transmit  buffer empty ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    mov.w   u0rb,r0             ; receive data ---> r0
    add.w   #1,r3               ; r3+1 increment
    cmp.w   #2,r3               ; r3 = 2 ?
    jltu    U_Read_RB_loop      ; jump U_Read_RB_loop at r3<2
    bclr    re_u0c1             ; Reception disabled
    bset    te_u0c1             ; Transmission enabled
    mov.b   r1l,u0tb            ; dummy --> Transmit buffer register
?:
    btst    ti_u0c1             ; transmit buffer empty?
    jnc     ?-

    jmp     U_Loop_main
;
;-------------------------------------------------------
;   ID check
;-------------------------------------------------------
U_ID_check:
    mov.w   #0,r3               ; receive number (r3=0)
    mov.w   #0ffh,a1            ; ID size (dummy data = ffh)
U_ID_data_store:
    cmp.w   a1,r3               ; r3=a1(ID size)
    jeq     U_ID_address_check  ; jump ID_address_check at r3=ID size
    bclr    busy                ; BUSY  "L"
?:
    btst    ri_u0c1             ; Receive complete ?
    jz      ?-
    bset    busy                ; BUSY  "H"
    mov.w   u0rb,r0             ; receive data --> 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 ?
    jne     U_ID_data_store     ; jump ID_data_store at r3 not= 4
```

```
        mov.b   data,a1             ; ID size --> a1
        add.w   #4,a1               ; a1=a1+4
        jmp     U_ID_data_store     ; jump ID_data_store
U_ID_address_check:
        btst    blank               ; blank flag check
        jc      U_ID_check_end      ; jump ID_check_end at blank
        cmp.w   #0ffdfh,addr_l      ; lower ID address check
        jne     U_ID_error          ; jump ID_error at ID address error
        cmp.w   #0070fh,addr_h      ; higher ID address check
        jne     U_ID_error          ; jump ID_error at ID address error
U_ID_data_check:
        mov.w   #0000fh,a1          ; ID higher address --> a1
        mov.w   #0ffdfh,r1          ; ID lower address --> r1
        mov.w   #1,r3               ; check loop number (r3=1)
U_ID_check_loop:
        mov.w   r1,a0               ; r1 --> a0
        lde.b   [a1a0],r0l          ; ID data read from Flash memory
        mov.w   r3,a0               ; r3 --> a0
        cmp.b   r0l,data[a0]        ; compare ID data
        jne     U_ID_error            ; jump ID_error at ID error
        add.w   #4,r1               ; r1+4 increment (next ID address)
        cmp.w   #0ffe7h,r1          ; r1=0ffefh ?
        jne     ?+                  ; jump ? at not equal
        mov.w   #0ffebh,r1          ; r1=0ffeb at equal
?:
        add.w   #1,r3               ; r3 +1 increment
        cmp.w   #8,r3               ; r3=8 ?
        jltu    U_ID_check_loop     ; jump ID_check_loop at r3<8
U_ID_OK:
        bset    sr10
        bset    sr11                ; ID check OK (sr11=1,sr10=1)
        jmp     U_ID_check_end      ; jump  ID_check_end
U_ID_error:
        bset    sr10
        bclr    sr11                ; ID error (sr11=0,sr10=1)
U_ID_check_end:
        jmp     U_Loop_main         ; jump Loop_main
;
;-------------------------------------------------------
;     Download
;-------------------------------------------------------
U_Download:
        bclr    fcon00              ; not CPU write mode
        jmp.a   U_Download_program  ; jump Download_program
;
;--------------------------------------------------------
;+      Version output                                 +
;--------------------------------------------------------
U_Ver_output:
        mov.w   #0,a0
        bclr    re_u0c1             ; Reception disabled
        bset    te_u0c1             ; Transmission enabled
U_Ver_output_loop:
        lde.b   ver[a0],u0tb        ; Version data transfer
?:
        btst    ti_u0c1             ; transmit buffer empty ?
        jz      ?-
        add.w   #1,a0
        cmp.w   #8,a0
        jltu    U_Ver_output_loop
        jmp     U_Loop_main
;
;----------------------------------------------------------
;+     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
U_BPS_SET:
    bclr    re_u0c1             ; Reception disabled
    bset    te_u0c1             ; Transmission enabled
    mov.b   r0l,u0tb            ; r1l --> transmit buffer register
?:
    btst    ti_u0c1             ; transmit buffer empty ?
    jnc     ?-
?:
    btst    txept_u0c0
    jnc     ?-
    bclr    te_u0c1             ; Transmission disabled
    jsr     U_Initialize_20     ; UART mode Initialize
    jmp     U_Loop_main         ; jump Loop_main
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Freq check  - UART mode -                          +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_Freq_check:
    bclr    re_u0c1             ; Reception disabled
    btst    8,r1                ; counter = 8 times
    jc      U_Freq_check_4
;
    btst    freq_set1
    jc      U_Freq_check_1
    cmp.b   #00h,r0l            ; "00h"?
    jeq     U_Freq_check_3
    jmp     U_Freq_check_2
U_Freq_check_1:
    btst    13,r0               ; fer_u1rb
    jz      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           ; Min-Baud get ?
    jc      U_Freq_set_1        ; Yes , finished
    bset    freq_set1
    cmp.b   #00h,r0l            ; "00h"?
    jeq     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   #01111111b,data     ; Reset
U_Freq_check_6:
    mov.b   data,u0brg          ; Transmission late
?:
    btst    p5_1
    jz      ?-
    jmp     U_Loop_main
;
U_Freq_set_1:
    btst    13,r0               ; fer_u1rd
    jz      U_Freq_set_2
    xor.b   data,r1h
    mov.b   r1h,data
U_Freq_set_2:
    bset    freq_set2
    mov.b   data+1,r1l
    sub.b   data,r1l
    shl.b   #-1,r1l
    add.b   data,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,data
    mov.b   #0b0h,r0l           ; "B0h" set
    mov.b   data,u0brg          ; Transmission late
    jmp     U_BPS_SET
;
;-------------------------------------------------------
;+      6usec timer wait                               +
;-------------------------------------------------------
U_wait_6usec:
    bset    tb0s
?:
    btst    ir_tb0ic
    jz      ?-
    bclr    tb0s
    mov.b   #0,tb0ic
    rts
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Subroutine : U_Initialize_2                       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_Initialize_2:
    bset    fcon00              ; CPU write mode
    bset    fcon05              ; F4000h-FFFFFh select
    bclr    fcon04
    lde.w   0ffffch,r0          ; Reset vector read --> r0
    lde.w   0ffffeh,r1          ; Reset vector read --> r1
    and.w   r1,r0               ; r0 & r1
    cmp.w   #0ffffh,r0          ; Blank check
    jne     U_Blank_check_end   ; jump Blank_check_end at Blank error
    bset    sr10                ; check complete
    bset    sr11                ;
    bset    blank               ; blank flag set
U_Blank_check_end:
;
```

```
;---------------------------------------------------------
;+     UART0                                              +
;---------------------------------------------------------
;
U_Initialize_20:
;----- UART0 init late  generater 1
    mov.b   data,u0brg          ; Transmission late
;
U_Initialize_21:
;----- UART0 transmit/receive mode register
;
    mov.b   #0,u0mr             ; u0mr reset
    mov.b   #00000101b,u0mr
;            |||||+++------------ transfer data 8 bit long
;            ||||+--------------- Internal clock
;            |||+---------------- one stop bit
;            ||+----------------- parity disabled
;            |+------------------ sleep mode deselected
;
;----- UART0 transmit/receive control register 0
;
    mov.b   #00001000b,u0c0
;            |||| |++------------ 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
;

;----- UART transmit/received control register 1
;
    mov.b   #00000000b,u0c1
;            |||||||+------------ Transmission disabled
;            ||||||+------------- Transmission enabled
;            |||||+-------------- Reception disabled
;            ||||+--------------- Reception enabled
;            ++++---------------- fixed
;
;---------------------------------------------------------
;+     Timer                                              +
;---------------------------------------------------------
    mov.b   #02h,ta0mr          ; f1 select,one-shot mode
    mov.b   #0,ta0ic            ; Interrupt flag clear
    mov.w   #3000-1,ta0         ; 300usec at 10 MHz
    bset    ta0s
;
    mov.b   #00h,tb0mr          ; f1 select
    mov.w   #60-1,tb0           ; 6usec at 10 MHz
;
    rts
;
Trans_END2:
;
    .end
```

Header

```
;********************************************************
;*                                                      *
;*  file name   : definition of M16C/20 Flash           *
;*                                                      *
;*  Version     : 0.07 ( 1999-8-5)                      *
;*              : for Boot Ver 1.03                     *
;********************************************************
;
;-------------------------------------------------------
;   BUSY output
;-------------------------------------------------------
busy        .btequ  3,03E9h     ; p5_3
busy_d      .btequ  3,03EBh     ; pd5_3
;
;-------------------------------------------------------
;    define of symbols
;-------------------------------------------------------
Ram_TOP     .equ    000400h
Ram_END     .equ    000bffh     ;
Istack      .equ    000c00h     ;
;
Version     .equ    0dfbf0h
Boot_TOP    .equ    0df000h
Boot_END    .equ    0dffffh
Vector      .equ    0fffdch
;
SB_base     .equ    000400h
Ram_progTOP .equ    000600h
;
Download_program    .equ    0df100h
U_Download_program  .equ    0df1a0h
;
    .section    memory,data
    .org        Ram_TOP
;
SRD:        .blkb   1
SRD1:       .blkb   1
ver:        .blkb   10
SF:         .blkb   1
unuse:      .blkb   3
addr_l:     .blkb   1
addr_m:     .blkb   1
addr_h:     .blkb   1
data:       .blkb   256
ID:         .blkb   11
buff:       .blkb   20
ID_err:     .blkb   1
baud:       .blkb   4
;
sr0         .btequ  0,SRD       ;
sr1         .btequ  1,SRD       ;
sr2         .btequ  2,SRD       ;
sr3         .btequ  3,SRD       ; Block status after program ( 0=OK 1=ERR )
sr4         .btequ  4,SRD       ; Program status ( 0=OK 1=ERR )
sr5         .btequ  5,SRD       ; Erase status ( 0=OK 1=ERR )
sr6         .btequ  6,SRD       ;
sr7         .btequ  7,SRD       ; Write state machine status ( 0=BUSY 1=READY )
sr8         .btequ  0,SRD1      ; Block address error
sr9         .btequ  1,SRD1      ; Time out ( 0=OK 1=TIME OUT )
sr10        .btequ  2,SRD1      ; ID collation
sr11        .btequ  3,SRD1      ; (00= no check 01= error 10= --  11= OK )
```

```
sr12        .btequ  4,SRD1      ; Check sum ( 0= error 1= ok )
sr13        .btequ  5,SRD1      ;
sr14        .btequ  6,SRD1      ;
sr15        .btequ  7,SRD1      ; Download flag
;
ram_check   .btequ  0,SF
blank       .btequ  1,SF
s_mode      .btequ  2,SF
old_mode    .btequ  3,SF
freq_set0   .btequ  4,SF
freq_set1   .btequ  5,SF
freq_set2   .btequ  6,SF
;
```

2.4 Precautions

This section describes precautions to be observed when controlling the M16C/20's internal flash memory.

Handling of Vpp Power Supply

In addition to the operating Vcc power supply, the flash memory requires a high-voltage (12 V) Vpp power supply for program/erase operations. We recommend that the Vpp power supply be 12 V only when you need to program/erase the flash memory, and 0 V otherwise.
When using the Vpp power supply, pay attention to the following:

(1) Do not apply an overvoltage to the Vpp pin. If the flash memory's Vpp voltage exceeds the absolute maximum rated voltage of 14 V, the device may be damaged.

(2) When turning the Vpp voltage on or off, make sure the Vcc power supply is turned on. Before accessing the device, wait until the power supply stabilizes after being turned on.

(3) Set the current capacity of the Vpp power supply by considering the device's power consumption in the same way as for the Vcc power supply. The Vpp current (Ipp) during programming/erasing reaches the maximum value when program or erase operation is executed internally in the device after entering the command. At this time, be careful that the Vpp voltage applied to the M16C/20 will not drop.

(4) Connect a bypass capacitor to the Vpp power supply pin as close to the Vpp pin as possible, as for the Vcc power supply pin. To prevent a transient drop of the Vpp voltage in (3) above, connect a bypass capacitor as close to the Vpp pin as possible. Although the value of this bypass capacitor varies with the operating current, the appropriate value normally is 0.1 to 1 µF per device.

(5) Do not enter a low signal to the WE pin while you are applying 12 V to the Vpp pin. If the WE input is pulled low while the Vpp pin has 12 V applied to it, the flash memory may receive the data pin status at that point in time as a command. Therefore, if the WE input is pulled low by noise or for other reasons while writing data, the flash memory may be erroneously programmed/erased.

Additional programming inhibited

Additional programming means writing data to a byte again after once writing to it (the byte that passed verification).
Additional programming causes a high voltage to be applied to memory cells of the flash memory repeatedly, which may result in reduced or lost margins for memory data cell readout or degraded data retention characteristics.
Therefore, when programming the flash memory, be careful not to additionally write to the bytes that have passed test by Program Verify. However, an exception is that only when you wrote "00_{16}" to all bytes of flash memory before erasing, you can additionally write data "00_{16}" to each byte once. This is because writing "00_{16}" to all bytes of flash memory before erasing is indispensable to prevent overerase.
Table 2.4.1 shows inhibited additional programming and acceptable additional programming.

Table 2.4.1 Inhibited Additional Programming and Acceptable Additional Programming

Inhibited additional programming	(1) Writing the same data to the verified bytes again. Example: Writing data 25 times per byte without verifying programmed data. (2) Writing new data to already programmed bytes without erasing them.
Acceptable additional programming	Writing "00_{16}" to all bytes before erasing.

Chapter 3

M16C/62 Group

3.1 Outline of Hardware

The M16C/62 group contains DINOR-type flash memory.
This section shows hardware information about the M16C/62 group which we think is necessary to create a boot program.

Internal Flash Memory Outline

Table 3.1.1 shows the outline performance of M30624FG and M30624FGL of the M16C/62 group.

Table 3.1.1. Outline Performance of M30624FG and M30624FGL

Item		Performance
Power supply voltage		5V version: 2.7V to 5.5 V (f(XIN)=16MHz, without wait, 4.2V to 5.5V, f(XIN)=10MHz, with one wait, 2.7V to 5.5V) 3V version: 2.4V to 3.6 V (f(XIN)=10MHz, without wait, 2.7V to 3.6V, f(XIN)=7MHz, without wait, 2.4V to 3.6V)
Program/erase voltage		5V version: 4.2V to 5.5 V (f(XIN)=12.5MHz, with one wait, f(XIN)=6.25MHz, without wait) 3V version: 2.7V to 3.6 V (f(XIN)=10MHz, with one wait, f(XIN)=6.25MHz, without wait)
Flash memory operation mode		Three modes (parallel I/O, standard serial I/O, CPU rewrite)
Erase block division	User ROM area	See Figure 3.1.1
	Boot ROM area	One division (8 Kbytes) (Note)
Program method		In units of pages (in units of 256 bytes)
Erase method		Collective erase/block erase
Program/erase control method		Program/erase control by software command
Protect method		Protected for each block by lock bit
Number of commands		8 commands
Program/erase count		100 times
ROM code protect		Parallel I/O and standard serial modes are supported.

Note: The boot ROM area contains a standard serial I/O mode control program which is stored in it when shipped from the factory.This area can be erased and programmed in only parallel I/O mode.

Memory Map

The user ROM of M30624FG has seven blocks as block 0 to block 6. Figure 3.1.1 shows the memory map.

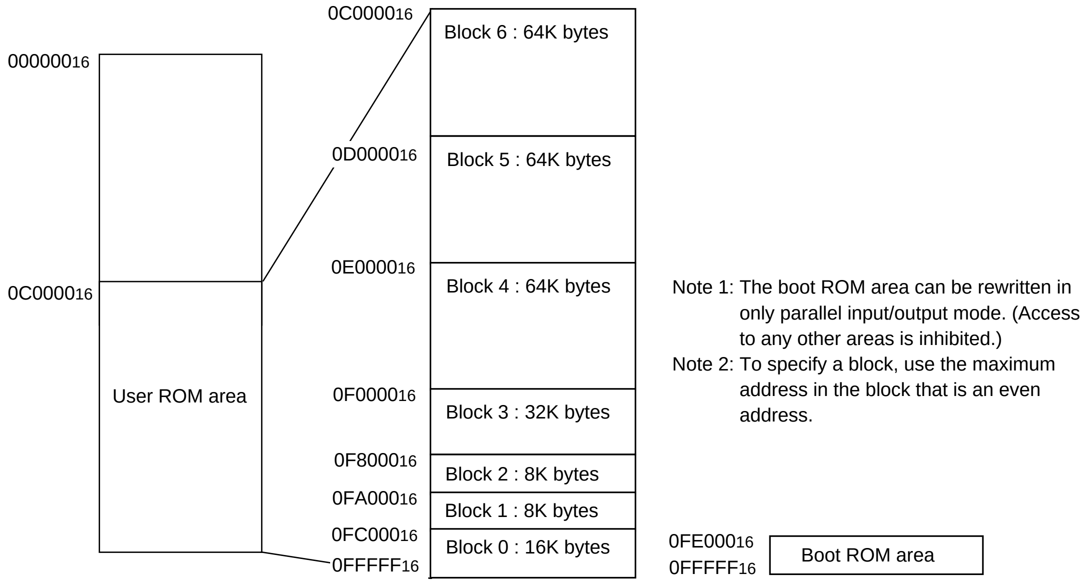


Figure 3.1.1 M30624FG memory Map

Related Register Configuration

Figure 3.1.2 shows related registers for making user boot program.

Flash memory control register 0

b7	b6	b5	b4	b3	b2	b1	b0
╳	╳		0				

Symbol	Address	When reset
FMR0	$03B7_{16}$	$XX000001_2$

Bit symbol	Bit name	Function	R	W
FMR00	RY/$\overline{BY}$ status flag	0: Busy (being written or erased) 1: Ready	○	×
FMR01	CPU rewrite mode select bit (Note 1)	0: Normal mode (Software commands invalid) 1: CPU rewrite mode (Software commands acceptable)	○	○
FMR02	Lock bit disable bit (Note 2)	0: Block lock by lock bit data is enabled 1: Block lock by lock bit data is disabled	○	○
FMR03	Flash memory reset bit (Note 3)	0: Normal operation 1: Reset	○	○
Reserved bit		Must always be set to “0”	○	○
FMR05	User ROM area select bit (Note 4) (Effective in only boot mode)	0: Boot ROM area is accessed 1: User ROM area is accessed	○	○
Nothing is assigned. When write, set "0". When read, values are indeterminate.			—	—

Note 1: For this bit to be set to “1”, the user needs to write a “0” and then a “1” to it in succession. When it is not this procedure, it is not enacted in “1”. This is necessary to ensure that no interrupt or DMA transfer will be executed during the interval. Use the control program except in the internal flash memory for write to this bit.
Note 2: For this bit to be set to “1”, the user needs to write a “0” and then a “1” to it in succession when the CPU rewrite mode select bit = “1”. When it is not this procedure, it is not enacted in “1”. This is necessary to ensure that no interrupt or DMA transfer will be executed during the interval.
Note 3: Effective only when the CPU rewrite mode select bit = 1. Set this bit to 0 subsequently after setting it to 1 (reset).
Note 4: Use the control program except in the internal flash memory for write to this bit.

Flash memory control register 1

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0		0	0	0

Symbol	Address	When reset
FMR1	$03B6_{16}$	$XXXX0XXX_2$

Bit symbol	Bit name	Function	R	W
Reserved bit		Must always be set to “0”	—	○
FMR13	Flash memory power supply-OFF bit (Note)	0: Flash memory power supply is connected 1: Flash memory power supply-off	○	○
Reserved bit		Must always be set to “0”	—	○

Note : For this bit to be set to “1”, the user needs to write a “0” and then a “1” to it in succession. When it is not this procedure, it is not enacted in “1”. This is necessary to ensure that no interrupt or DMA transfer will be executed during the interval. Use the control program except in the internal flash memory for write to this bit.
During parallel I/O mode,programming,erase or read of flash memory is not controlled by this bit,only by external pins.

Figure 3.1.2 Related Register Configuration

Flash Control Circuit

The M16C/62's flash control circuit controls the block erase and page program operations performed on the internal flash memory. Operation modes are selected by entering software commands to the flash control circuit. The status shows the status of the flash control circuit, as well as the status of program and block erase operations performed by the flash control circuit.
To enter commands to the flash control circuit, write the command to flash memory address.

Software Commands

Flash memory operations are selected by writing a software command to the flash control circuit. The table below lists the operations performed by software commands.

Table 3.1.2 Software Command List

Command	First bus cycle			Second bus cycle			Third bus cycle		
	Mode	Address	Data (D_0 to D_7)	Mode	Address	Data (D_0 to D_7)	Mode	Address	Data (D_0 to D_7)
Read array	Write	X (Note 6)	FF_{16}						
Read status register	Write	X	70_{16}	Read	X	SRD (Note 2)			
Clear status register	Write	X	50_{16}						
Page program (Note 3)	Write	X	41_{16}	Write	WA0 (Note 3)	WD0 (Note 3)	Write	WA1	WD1
Block erase	Write	X	20_{16}	Write	BA (Note 4)	$D0_{16}$			
Erase all unlock block	Write	X	$A7_{16}$	Write	X	$D0_{16}$			
Lock bit program	Write	X	77_{16}	Write	BA	$D0_{16}$			
Read lock bit status	Write	X	71_{16}	Read	BA	D_6 (Note 5)			

Note 1: When a software command is input, the high-order byte of data (D_8 to D_{15}) is ignored.
Note 2: SRD = Status Register Data
Note 3: WA = Write Address, WD = Write Data
WA and WD must be set sequentially from 00_{16} to FE_{16} (byte address; however, an even address). The page size is 256 bytes.
Note 4: BA = Block Address (Enter the maximum address of each block that is an even address.)
Note 5: D_6 corresponds to the block lock status. Block not locked when D_6 = 1, block locked when D_6 = 0.
Note 6: X denotes a given address in the user ROM area (that is an even address).

Flash Memory Address

The table below shows the flash memory capacity of each block (address space, number of pages) and the block addresses of each block.

Table 3.1.3 Flash Memory Address

	Size	Page No.	Address	Block address
Block 6	64 Kbytes	256	$C0000_{16}$–$CFFFF_{16}$	$CFFFE_{16}$
Block 5	64 Kbytes	256	$D0000_{16}$–$DFFFF_{16}$	$DFFFE_{16}$
Block 4	64 Kbytes	256	$E0000_{16}$–$EFFFF_{16}$	$EFFFE_{16}$
Block 3	32 Kbytes	128	$F0000_{16}$–$F7FFF_{16}$	$F7FFE_{16}$
Block 2	8 Kbytes	32	$F8000_{16}$–$F9FFF_{16}$	$F9FFE_{16}$
Block 1	8 Kbytes	32	$FA000_{16}$–$FBFFF_{16}$	$FBFFE_{16}$
Block 0	16 Kbytes	64	$FC000_{16}$–$FFFFF_{16}$	$FFFFE_{16}$

Read Array Command (FF_{16})

The read array mode is entered by writing the command code "FF_{16}" in the first bus cycle. When an even address to be read is input in one of the bus cycles that follow, the content of the specified address is read out at the data bus (D0–D15), 16 bits at a time. The read array mode is retained intact until another command is written.

Read Status Register Command (70_{16})

When the command code "70_{16}" is written in the first bus cycle, the content of the status register is read out at the data bus (D0–D7) by a read in the second bus cycle.
The status register is explained in the next section.

Clear Status Register Command (50_{16})

This command is used to clear the bits SR3 to 5 of the status register after they have been set. These bits indicate that operation has ended in an error. To use this command, write the command code "50_{16}" in the first bus cycle.

Page Program Command (41_{16})

Page program allows for high-speed programming in units of 256 bytes. Page program operation starts when the command code "41_{16}" is written in the first bus cycle. In the second bus cycle through the 129th bus cycle, the write data is sequentially written 16 bits at a time. At this time, the addresses A0-A7 need to be increased by 2 from "00_{16}" to "FE_{16}." When the system finishes loading the data, it starts an auto write operation (data program and verify operation).
Whether the auto write operation is completed can be confirmed by reading the status register or the flash memory control register 0. At the same time the auto write operation starts, the read status register mode is automatically entered.
After the auto write operation is completed, the status register can be read out to know the result of the auto write operation. For details, refer to the section where the status register is detailed.
The status register bit 7 (SR7) is set to 0 at the same time the auto write operation starts and is returned to 1 upon completion of the auto write operation. In this case, the read status register mode remains active until the Read Array command (FF_{16}) or Read Lock Bit Status command (71_{16}) is written or the flash memory is reset using its reset bit.
The RY/$\overline{BY}$ status flag of the flash memory control register 0 is 0 during auto write operation and 1 when the auto write operation is completed as is the status register bit 7.
Figure 3.1.3 shows an example of a page program flowchart.
Each block of the flash memory can be write protected by using a lock bit. For details, refer to the section where the data protect function is detailed.
Additional writes to the already programmed pages are prohibited.

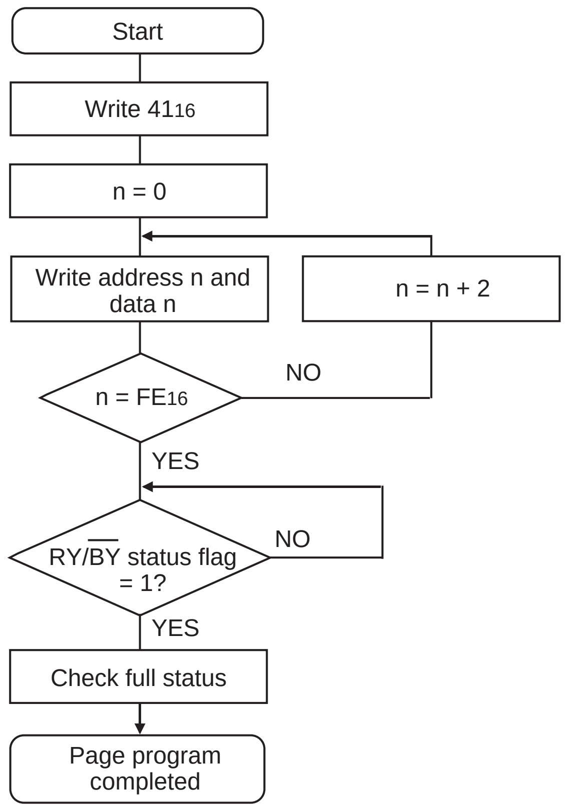


Figure 3.1.3 Page Program Flowchart

Block Erase Command (20_{16}/$D0_{16}$)

By writing the command code “20_{16}” in the first bus cycle and the confirm command code “$D0_{16}$” in the second bus cycle that follows to the block address of a flash memory block, the system initiates an auto erase (erase and erase verify) operation.
Whether the auto erase operation is completed can be confirmed by reading the status register or the flash memory control register 0. At the same time the auto erase operation starts, the read status register mode is automatically entered, so the content of the status register can be read out. The status register bit 7 (SR7) is set to 0 at the same time the auto erase operation starts and is returned to 1 upon completion of the auto erase operation. In this case, the read status register mode remains active until the Read Array command (FF_{16}) or Read Lock Bit Status command (71_{16}) is written or the flash memory is reset using its reset bit.
The RY/$\overline{\text{BY}}$ status flag of the flash memory control register 0 is 0 during auto erase operation and 1 when the auto erase operation is completed as is the status register bit 7.
After the auto erase operation is completed, the status register can be read out to know the result of the auto erase operation. For details, refer to the section where the status register is detailed.
Figure 3.1.4 shows an example of a block erase flowchart.
Each block of the flash memory can be protected against erasure by using a lock bit. For details, refer to the section where the data protect function is detailed.

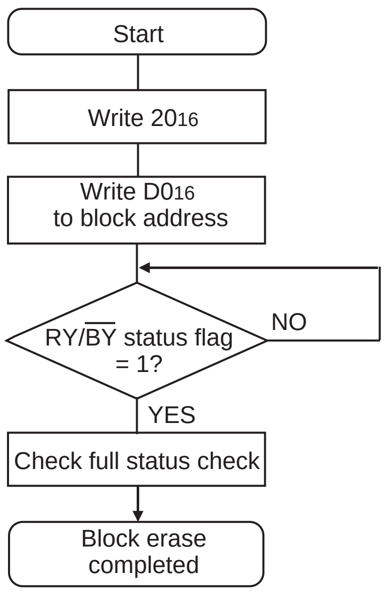


Figure 3.1.4 Block Erase Flowchart

Erase All Unlock Blocks Command (A7$_{16}$/D0$_{16}$)

By writing the command code “A7$_{16}$” in the first bus cycle and the confirm command code “D0$_{16}$” in the second bus cycle that follows, the system starts erasing blocks successively.
Whether the Erase All Unlock Blocks command is terminated can be confirmed by reading the status register or the flash memory control register 0, in the same way as for block erase. Also, the status register can be read out to know the result of the auto erase operation.
When the lock bit disable bit of the flash memory control register 0 = 1, all blocks are erased no matter how the lock bit is set. On the other hand, when the lock bit disable bit = 0, the function of the lock bit is effective and only unlocked blocks (where lock bit data = 1) are erased.

Lock Bit Program Command (77$_{16}$/D0$_{16}$)

By writing the command code “77$_{16}$” in the first bus cycle and the confirm command code “D0$_{16}$” in the second bus cycle that follows to the block address of a flash memory block, the system sets the lock bit for the specified block to 0 (locked).
Figure 3.1.5 shows an example of a lock bit program flowchart. The status of the lock bit (lock bit data) can be read out by a Read Lock Bit Status command.
Whether the lock bit program command is terminated can be confirmed by reading the status register or the flash memory control register 0, in the same way as for page program.
For details about the function of the lock bit and how to reset the lock bit, refer to the section where the data protect function is detailed.

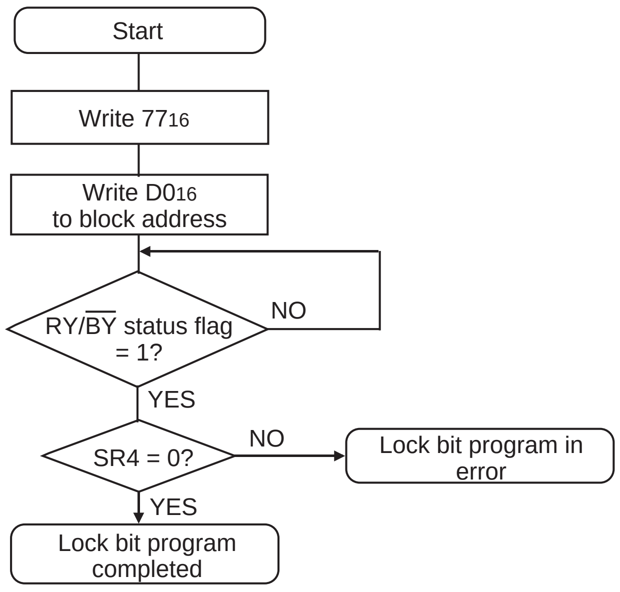
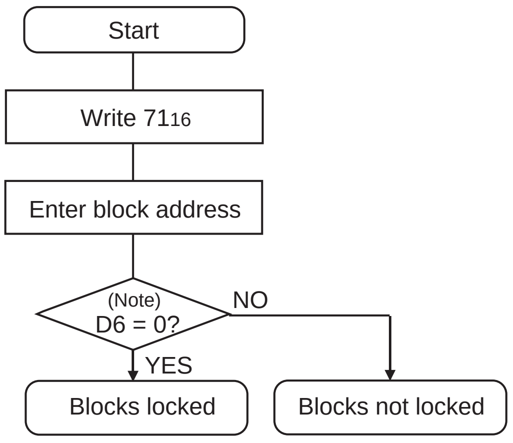

Start
Write 77$_{16}$
Write D0$_{16}$ to block address
RY/$\overline{BY}$ status flag = 1?
NO
YES
SR4 = 0?
NO
Lock bit program in error
YES
Lock bit program completed

Figure 3.1.5 Lock Bit Program Flowchart

Read Lock Bit Status Command (71_{16})

By writing the command code “71_{16}” in the first bus cycle and then the block address of a flash memory block in the second bus cycle that follows, the system reads out the status of the lock bit of the specified block on to the data (D6).
Figure 3.1.6 shows an example of a read lock bit program flowchart.



Note: Data bus bit 6.

Figure 3.1.6 Read Lock Bit Program Flowchart

Data Protect Function (Block Lock)

Each block in Figure 3.1.1 has a nonvolatile lock bit to specify that the block be protected (locked) against erase/write. The Lock Bit Program command is used to set the lock bit to 0 (locked). The lock bit of each block can be read out using the Read Lock Bit Status command.
Whether block lock is enabled or disabled is determined by the status of the lock bit and how the flash memory control register 0’s lock bit disable bit is set.

(1) When the lock bit disable bit = 0, a specified block can be locked or unlocked by the lock bit status (lock bit data). Blocks whose lock bit data = 0 are locked, so they are disabled against erase/write.
On the other hand, the blocks whose lock bit data = 1 are not locked, so they are enabled for erase/write.

(2) When the lock bit disable bit = 1, all blocks are unlocked regardless of the lock bit data, so they are enabled for erase/write. In this case, the lock bit data that is 0 (locked) is set to 1 (unlocked) after erasure, so that the lock bit-actuated lock is removed.

Status Register

The status register indicates the operating status of the flash memory and whether an erase or program operation has terminated normally or in an error. The content of this register can be read out by only writing the read status register command (70_{16}). Table 3.1.3 details the status register.
The status register is cleared by writing the Clear Status Register command (50_{16}).
After a reset, the status register is set to "80_{16}."
Each bit in this register is explained below.

Write State Machine (WSM) Status (SR7)

After power-on, the write state machine (WSM) status is set to 1.
The write state machine (WSM) status indicates the operating status of the device, as for output on the RY/$\overline{\text{BY}}$ pin. This status bit is set to 0 during auto write or auto erase operation and is set to 1 upon completion of these operations.

Erase Status (SR5)

The erase status informs the operating status of auto erase operation to the CPU. When an erase error occurs, it is set to 1.
The erase status is reset to 0 when cleared.

Program Status (SR4)

The program status informs the operating status of auto write operation to the CPU. When a write error occurs, it is set to 1.
The program status is reset to 0 when cleared.
When an erase command is in error (which occurs if the command entered after the block erase command (20_{16}) is not the confirm command ($D0_{16}$), both the program status and erase status (SR5) are set to 1.
When the program status or erase status = 1, the following commands entered by command write are not accepted.
Also, in one of the following cases, both SR4 and SR5 are set to 1 (command sequence error):
(1) When the valid command is not entered correctly
(2) When the data entered in the second bus cycle of lock bit program (77_{16}/$D0_{16}$), block erase (20_{16}/$D0_{16}$), or erase all unlock blocks ($A7_{16}$/$D0_{16}$) is not the $D0_{16}$ or FF_{16}. However, if FF_{16} is entered, read array is assumed and the command that has been set up in the first bus cycle is canceled.

Block Status After Program (SR3)

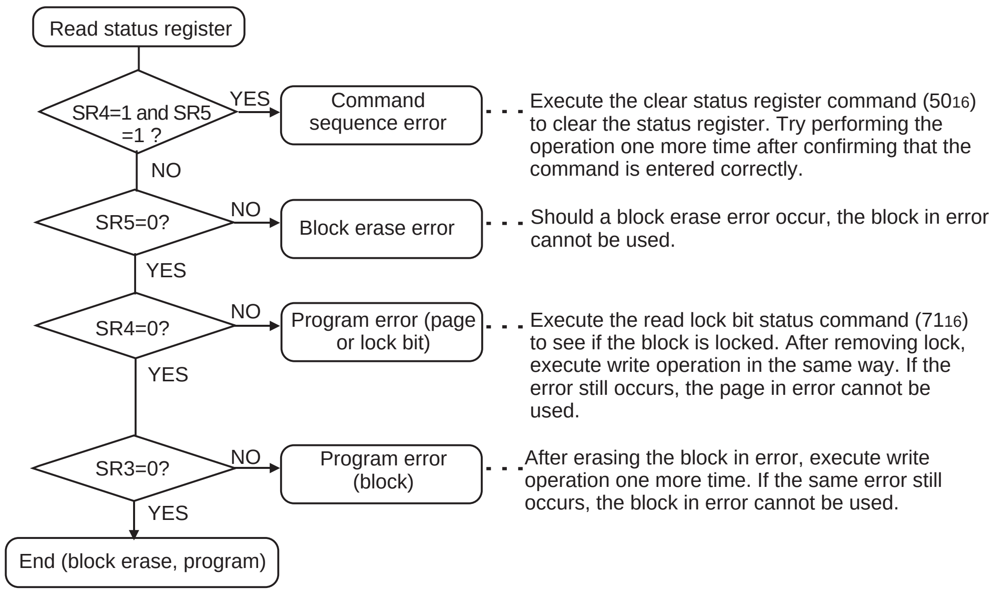
If excessive data is written (phenomenon whereby the memory cell becomes depressed which results in data not being read correctly), "1" is set for the program status after-program at the end of the page write operation. In other words, when writing ends successfully, "80_{16}" is output; when writing fails, "90_{16}" is output; and when excessive data is written, "88_{16}" is output.

Table 3.1.4 Definition of Each Bit in Status Register

Each bit of SRD	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	-	-

Full Status Check

By performing full status check, it is possible to know the execution results of erase and program operations. Figure 3.1.7 shows a full status check flowchart and the action to be taken when each error occurs.



Execute the clear status register command (50_{16}) to clear the status register. Try performing the operation one more time after confirming that the command is entered correctly.

Should a block erase error occur, the block in error cannot be used.

Execute the read lock bit status command (71_{16}) to see if the block is locked. After removing lock, execute write operation in the same way. If the error still occurs, the page in error cannot be used.

After erasing the block in error, execute write operation one more time. If the same error still occurs, the block in error cannot be used.

Note: When one of SR5 to SR3 is set to 1, none of the page program, block erase, erase all unlock blocks and lock bit program commands is accepted. Execute the clear status register command (50_{16}) before executing these commands.

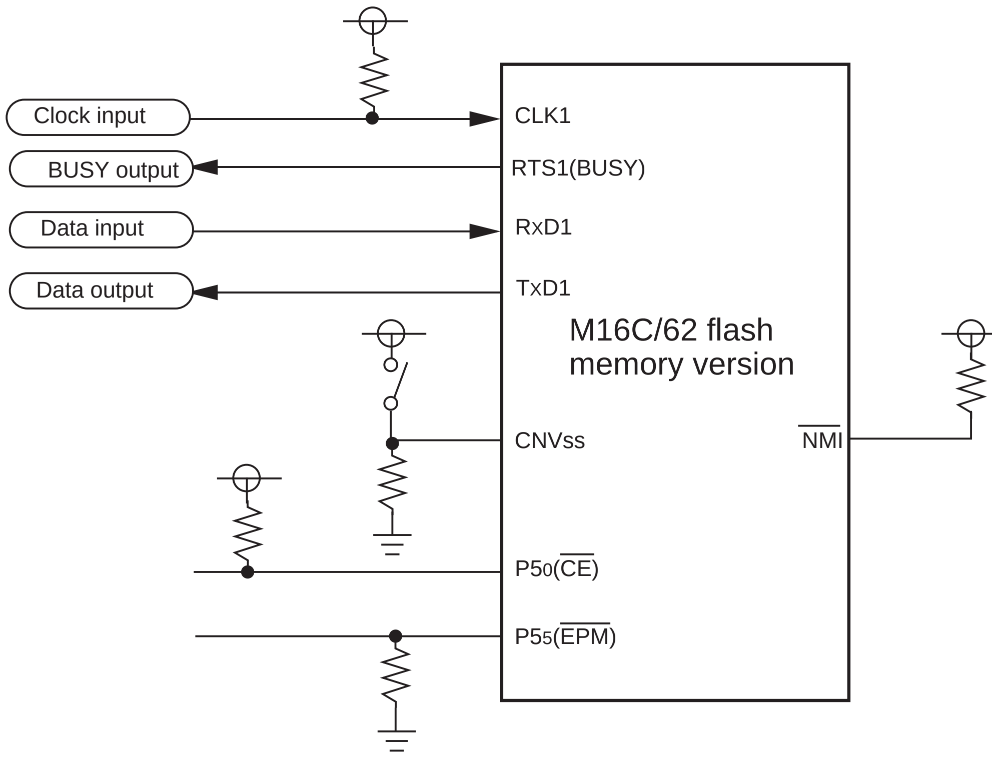
Figure 3.1.7 Full Status Check Flowchart and Remedial Procedure for Errors

3.2 Developing The Boot Program

The standard boot program that was built into the boot ROM area of the flash microcomputer when shipped from the factory can be used to program/erase the flash memory. In this case, the hardware resources (internal functions) used for control are fixed. Therefore, if you want to control flash memory in the way suitable for your system, you need to create a boot program for yourself.
This section shows an algorithm for the boot program (e.g., for erase and program) that you must at least have in order to control the flash memory of the M16C/62 group.

System Example

By using the internal peripheral function of UART1 and a serial programmer to control flash memory, the following shows an example of device connections is shown in Figure 3.2.1. Assignments of internal peripheral functions are listed in Table 3.2.1.



(1) Control pins and external circuitry will vary according to peripheral unit (programmer). For more information, see the peripheral unit (programmer) manual.
(2) In this example, the microprocessor mode and standard serial I/O mode are switched via a switch

Figure 3.2.1 Example of Device Connection

Table 3.2.1 Assignments of Internal Peripheral Functions

Peripheral function	Usage	Setting example
UART1	Used for transfer/receive of serial programmer and data	• Clock synchronous serial I/O • External clock is used
Timer A0	Used for time-over judgment of serial transfer/receive	• One-shot timer mode • 300 µs(when 20MHz)

Flow of The Main Processing

Figure 3.2.2 shows a flow of the main processing.
After initializing the CPU, transfer the write control program to RAM. When transfer is finished, jump to RAM and execute the write control program from RAM.

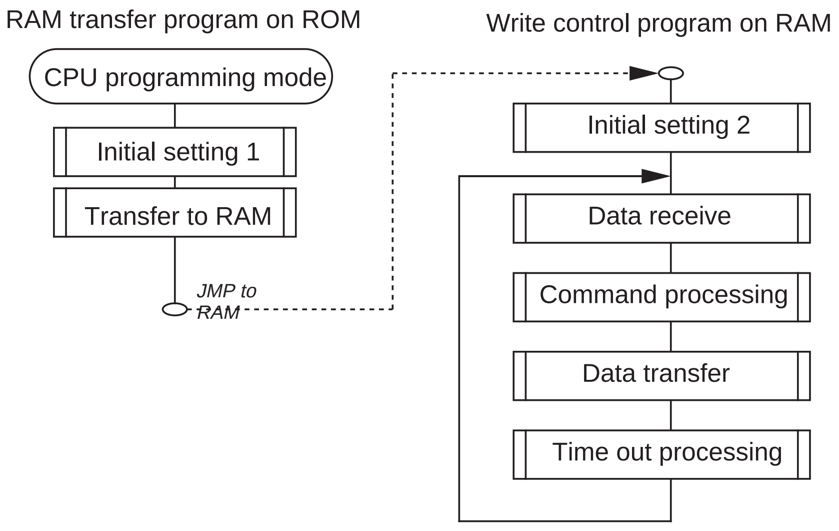


Figure 3.2.2 Flow of The Main Processing

Initialization 1 (CPU, Memory)

The CPU and RAM are initialized. Figure 3.2.3 shows a flow of initialization 1. To clear RAM, use of string instructions will prove effective.

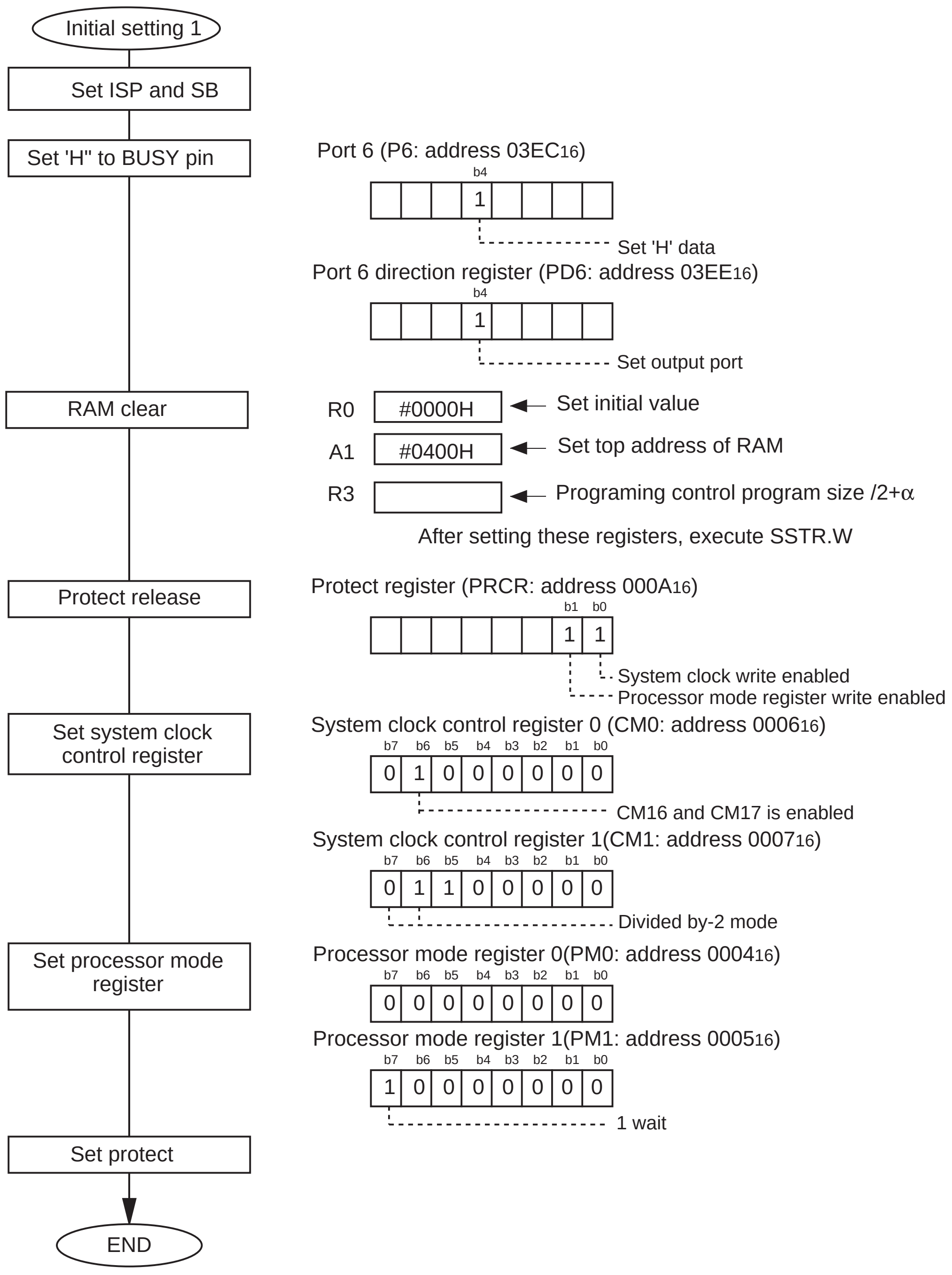

Initial setting 1
Set ISP and SB
Set 'H" to BUSY pin
RAM clear
Protect release
Set system clock control register
Set processor mode register
Set protect
END
Port 6 (P6: address 03EC16)
Set 'H' data
Port 6 direction register (PD6: address 03EE16)
Set output port
R0 #0000H Set initial value
A1 #0400H Set top address of RAM
R3 Programing control program size /2+α
After setting these registers, execute SSTR.W
Protect register (PRCR: address 000A16)
System clock write enabled
Processor mode register write enabled
System clock control register 0 (CM0: address 000616)
0 1 0 0 0 0 0 0
CM16 and CM17 is enabled
System clock control register 1(CM1: address 000716)
0 1 1 0 0 0 0 0
Divided by-2 mode
Processor mode register 0(PM0: address 000416)
0 0 0 0 0 0 0 0
Processor mode register 1(PM1: address 000516)
1 0 0 0 0 0 0 0
1 wait

Figure 3.2.3 Initialization 1

Transfer to RAM Area

The version information of write program and write control program are transferred to RAM. After transferring, jump to write control program on RAM. To transfer, use of string instructions will prove effective.
Figure 3.2.4 shows the algorithm.

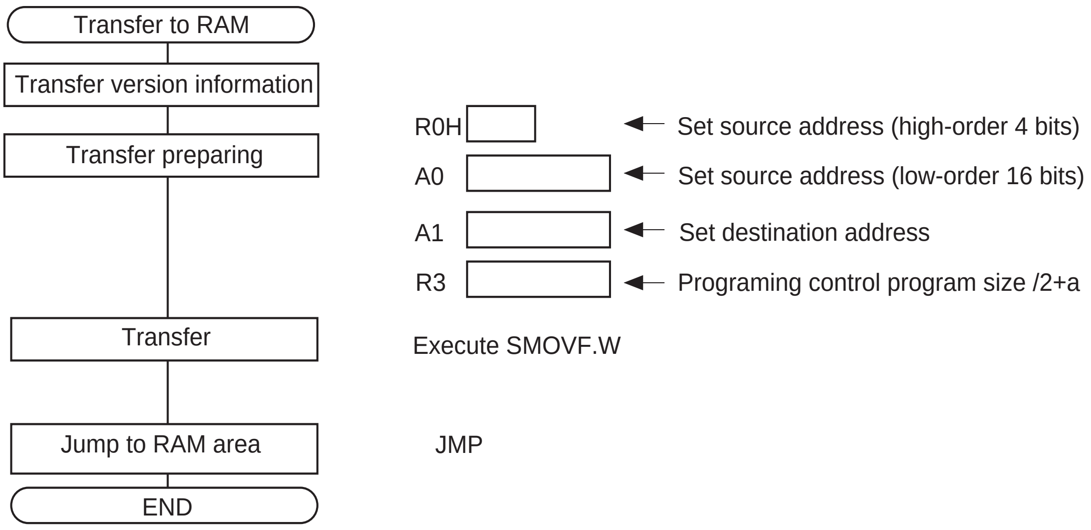

Transfer to RAM
Transfer version information
Transfer preparing
Transfer
Jump to RAM area
END
R0H
A0
A1
R3
Set source address (high-order 4 bits)
Set source address (low-order 16 bits)
Set destination address
Programing control program size /2+a
Execute SMOVF.W
JMP

Figure 3.2.4 Transfer to RAM Area

Initialization 2

Set of write to Flash memory and initialization of serial communication are executed. To switch erase/write mode, clear the flash entry bit (bit 1 of address 3B7$_{16}$), then set 1.
Figure 3.2.5 shows a algorithm.

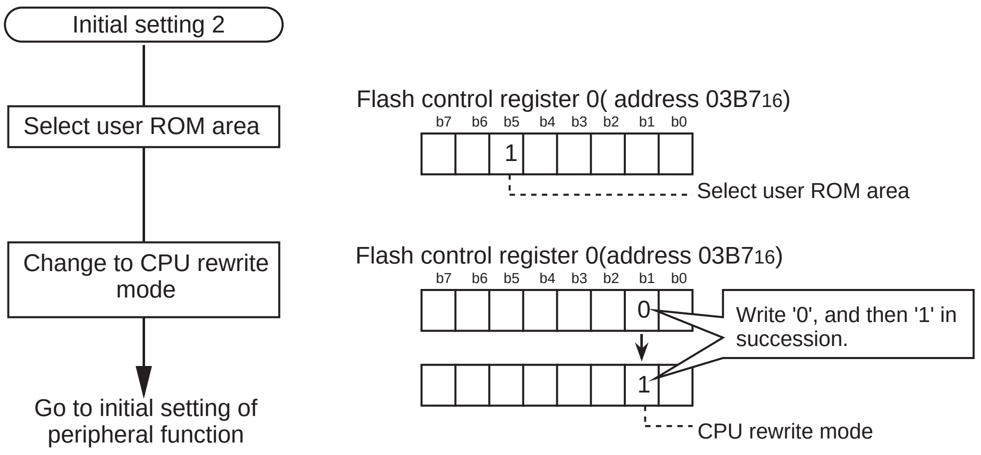

Initial setting 2
Select user ROM area
Change to CPU rewrite mode
Go to initial setting of peripheral function
Flash control register 0(address 03B716)
b7 b6 b5 b4 b3 b2 b1 b0
1
Select user ROM area
Flash control register 0(address 03B716)
b7 b6 b5 b4 b3 b2 b1 b0
0
1
Write '0', and then '1' in succession.
CPU rewrite mode

Figure 3.2.5 Initialization 2

Initialization 2 (Peripheral Function)

The peripheral functions used for programming flash memory is initialized. Figure 3.2.6 shows initialization of UART1 for data transmit and timer A0 for time-out calculation.

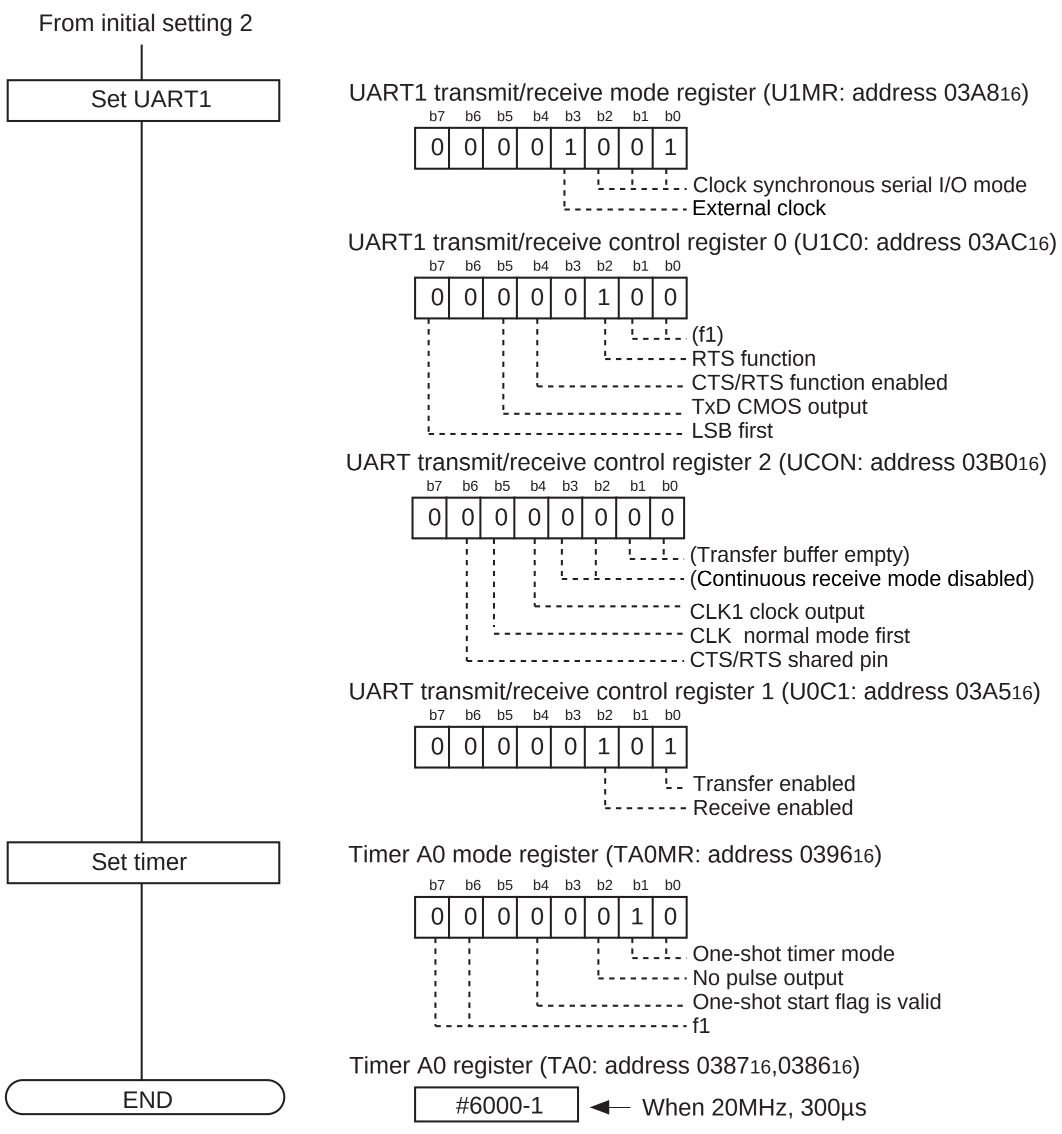


Figure 3.2.6 Initialization 2 (Peripheral Function)

Receiving Commands

Commands are received from the serial programmer.
Write dummy data to the transmit buffer, enable reception (the BUSY signal = low), and wait for data from the serial programmer. At the timing of start reception (the BUSY signal = high), the timer used to check data reception time-out is started. When data is not received within 300 msec, a time-out error is judged and time-out processing flag is set.
When command reception flag is set (cmd_flg = "1"), processing jumps to data reception cycle number check processing. When it is not set (cmd_flg = "0"), command reception flag is set. After that, jump address is set based on the received serial command and processing jumps to the corresponding process. When the serial command is not matched, serial initialization flag is set and processing is ended. When the number of receive cycle matches to the prescribed number of serial reception command, command reception flag is initialized (cmd_flg = "0") and processing is ended.
Figure 2.2.7 shows a processing flow.

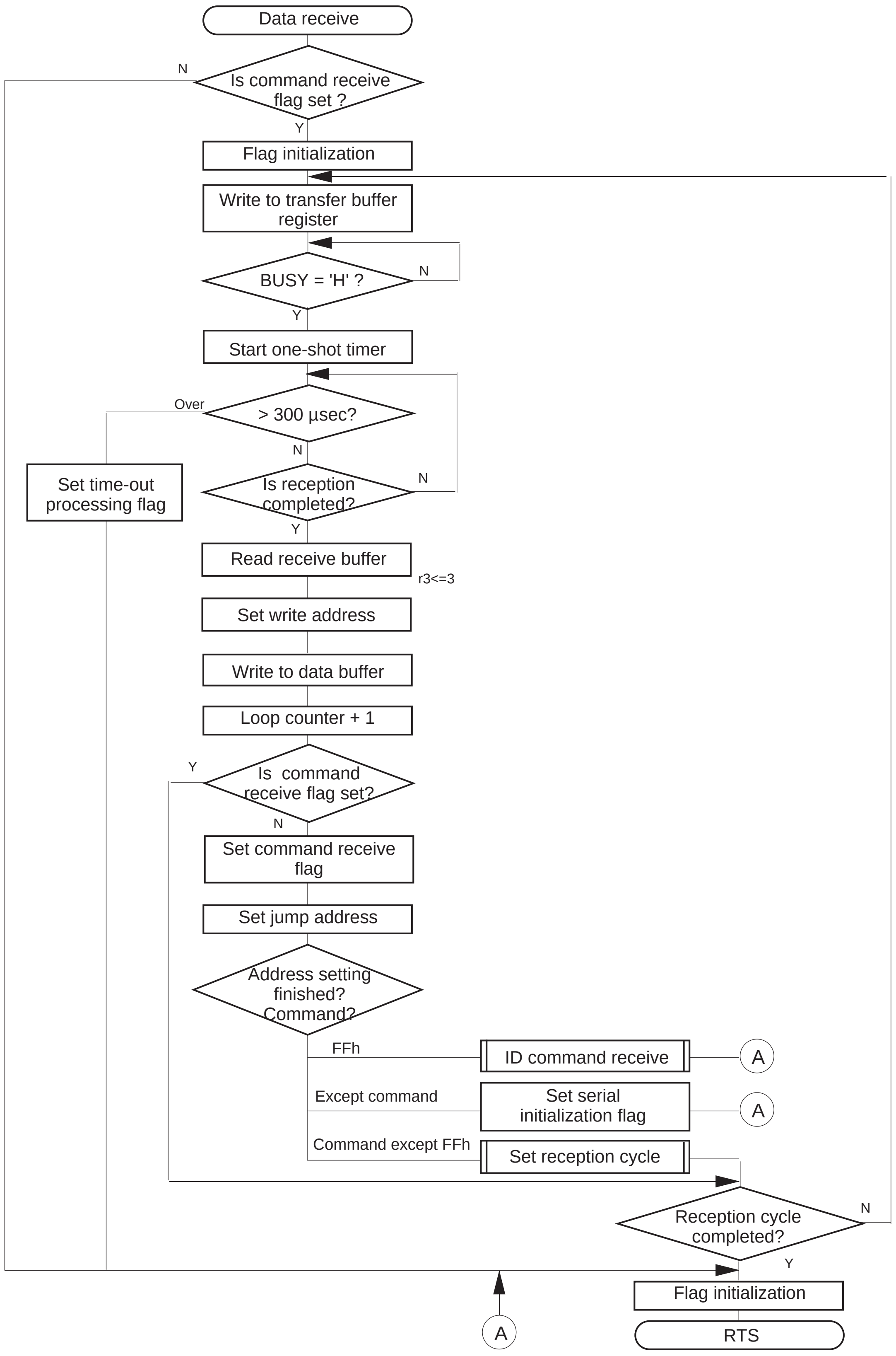


Figure 3.2.7 Data Reception

ID Check Receive Processing

ID check data is received. Transferred ID data is saved to RAM.
Figure 3.2.8 shows a processing flow.

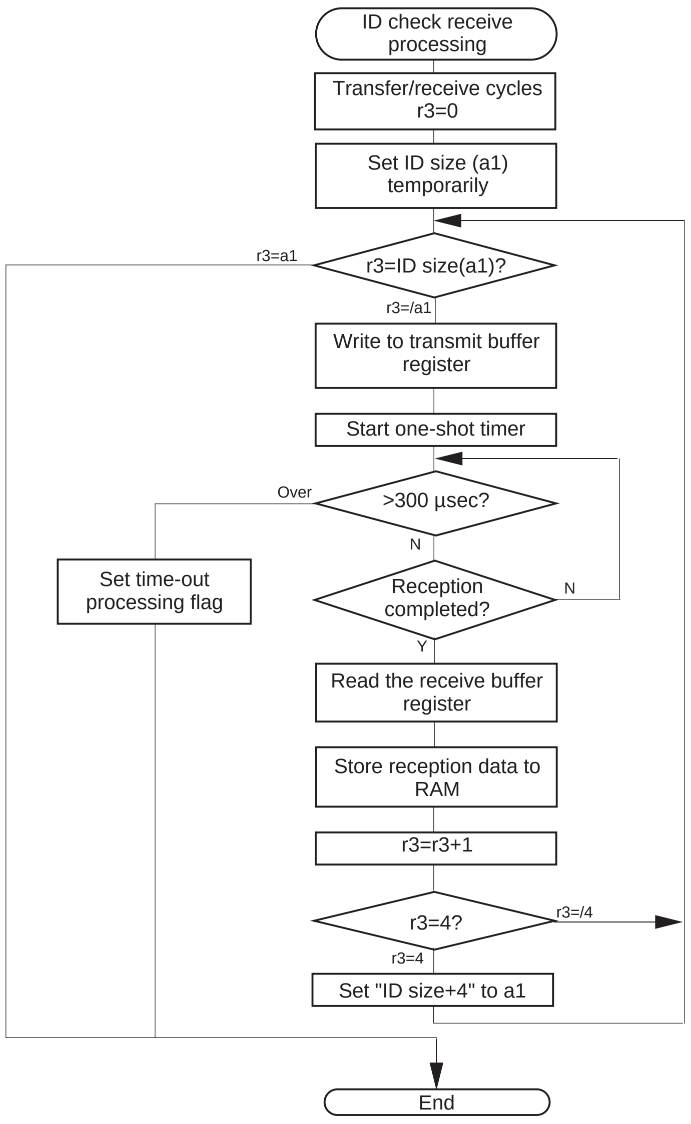

ID check receive processing
Transfer/receive cycles r3=0
Set ID size (a1) temporarily
r3=a1
r3=ID size(a1)?
r3=/a1
Write to transmit buffer register
Start one-shot timer
Over
>300 µsec?
N
Set time-out processing flag
Reception completed?
N
Y
Read the receive buffer register
Store reception data to RAM
r3=r3+1
r3=4?
r3=/4
r3=4
Set "ID size+4" to a1
End

Figure 3.2.8 ID Data Receive Process

Receive Cycle Setting Processing

Data receive cycle is set by referring to transferred serial command.
Figure 3.2.9 shows processing flow.

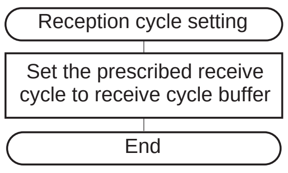


Figure 3.2.9 Receive Cycle Setting Processing

Command Processing

Flash control command is written into memory by referring to received serial programmer command.
The ID check is checked as to whether it has been completed or not. (ID check completed bits: SR10 = 1, SR11 = 1) When the ID check has been completed, decisions are made on commands such as page read and page program, and processing branches to the process in the match commands.
When the ID check has not been completed, decisions are made on 3 types of commands such as ID processing, and processing jumps to the process in the match commands. With mismatch commands, processing returns to main part.
Figure 3.2.10 shows processing flow.

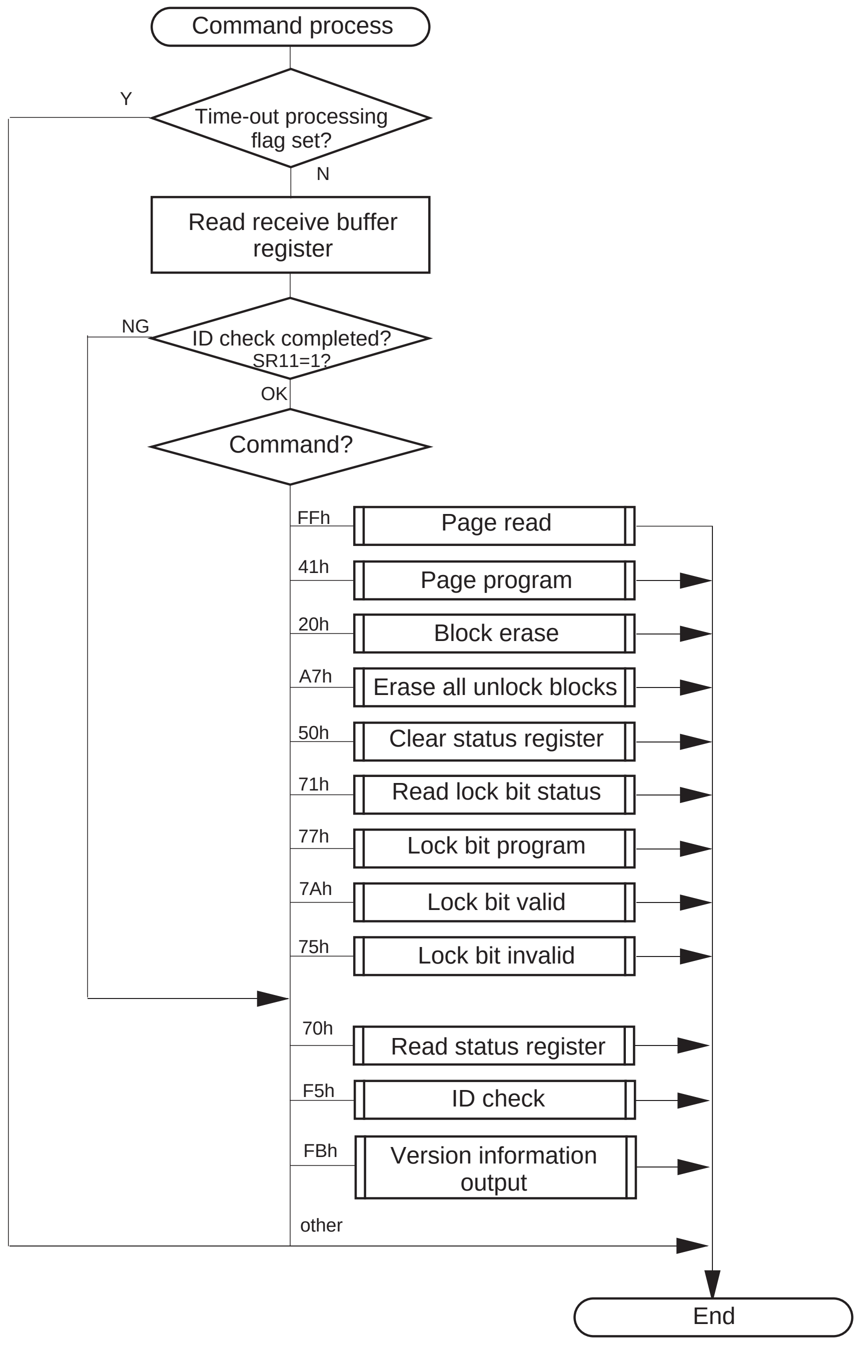


Figure 3.2.10 Command Process

Page Read

To read data from the user area in blocks of 256 bytes, read address is stored to RAM and Read Array command (FF_{16}) is written. The address of the read area is changed from $xxx00_{16}$ to $xxxFF_{16}$, and the data following $xxx00_{16}$ is transferred in succession.
Figure 3.2.11 shows processing flowchart.

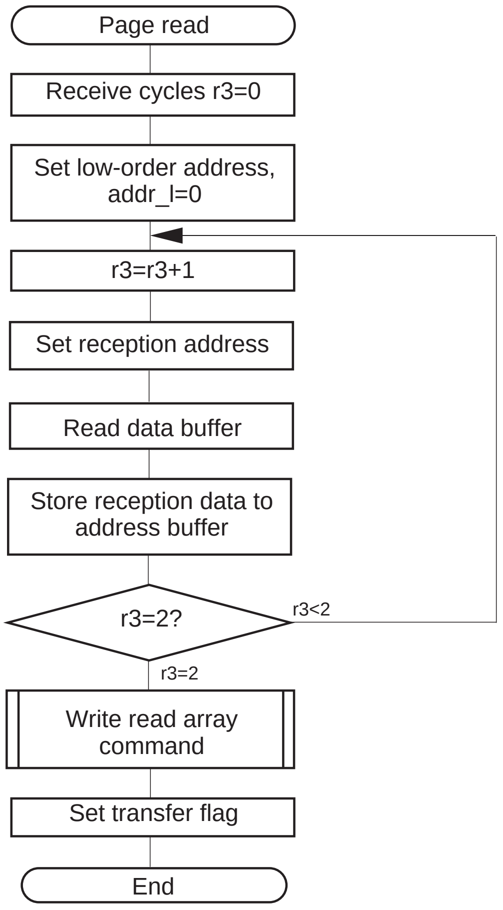


Figure 3.2.11 Page Read

Page Program

Data is written into the user area in blocks of 256 bytes.
Read 258 bytes data from RAM: 2 bytes of address and 256 bytes of write data received from serial programer. Status data is read from the flash memory. The read status is checked. When it is under error state, processing does not write but returns to the main part.
When it is not under error state, the page program command (41_{16}) is written in the flash memory, then 256 bytes of data is written. After data has been written, the read array command (FF_{16}) is written and processing returns to the main part.
Figure 3.2.12 shows processing flow.

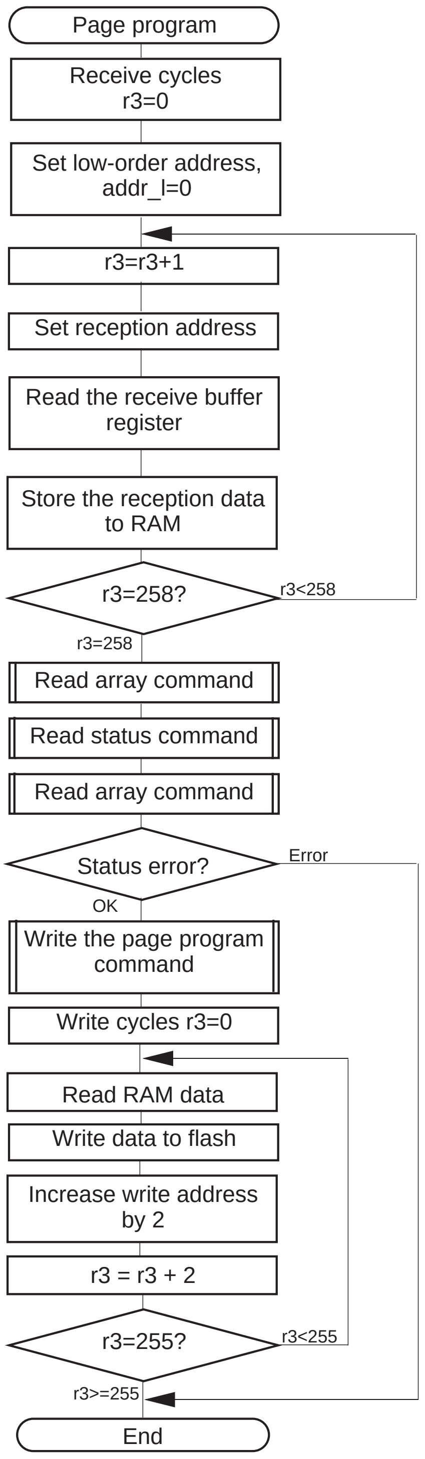


Figure 3.2.12 Page Program

Block Erase

A specified block area of the flash memory is erased. However, blocks that are locked by Lock Bit Program command cannot be erased. To erase these blocks, you need to disable the lock bit.
After confirming the two bytes of address and one byte of confirm command ($D0_{16}$) received from the serial programmer and stored in RAM, write Block Erase command (20_{16}) and confirm command ($D0_{16}$) to the area specified by the received address for block erase processing.
If the received confirm command is incorrect, block erase processing cannot be performed. In this case, write Read Array command (FF_{16}) to the flash memory to return the processing to the main routine.
Figure 3.2.13 shows a processing flow.

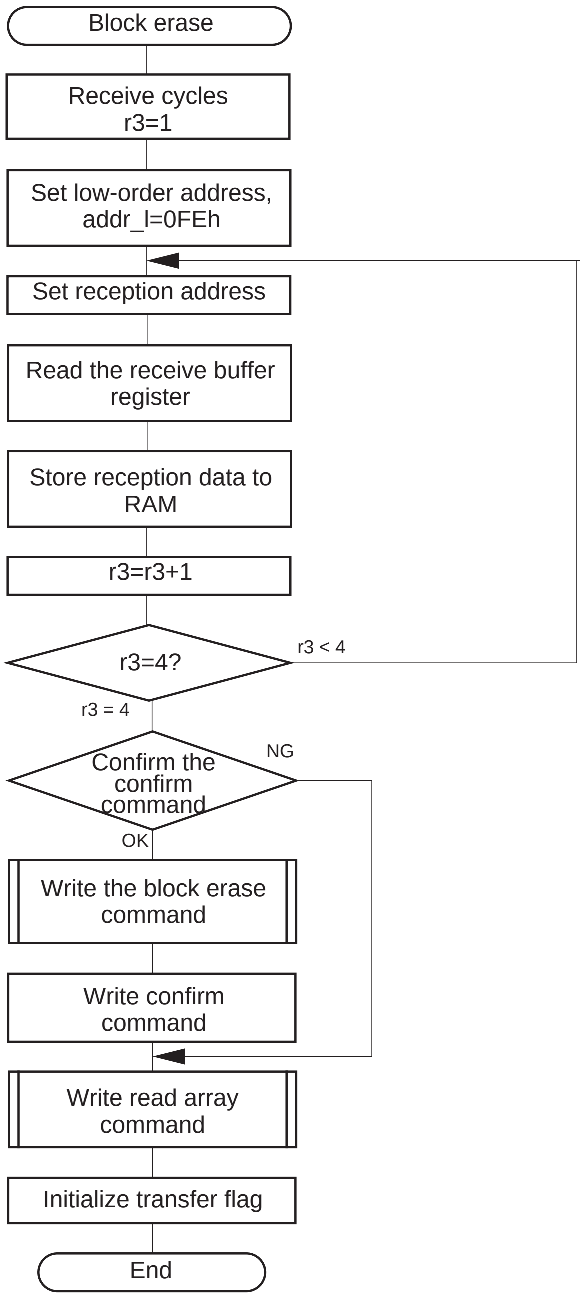


Figure 3.2.13 Block Erase

Erase All Unlock Block

A specified block area of the flash memory is erased. However, blocks that are locked by Lock Bit Program command cannot be erased. To erase these blocks, you need to disable the lock bit.
When the all erase command is received from a serial programmer, receive more 1 byte data in succession.
After the second data is checked to see if it is confirm command ($D0_{16}$), write Erase All Unlock command (20_{16}) and confirm command ($D0_{16}$) to the area specified by the received address for erase all unlock block processing.
If the received confirm command is incorrect, erase all unlock block processing cannot be performed. In this case, write Read Array command (FF_{16}) to the flash memory to end the processing.
Figure 3.2.14 shows a processing flow.

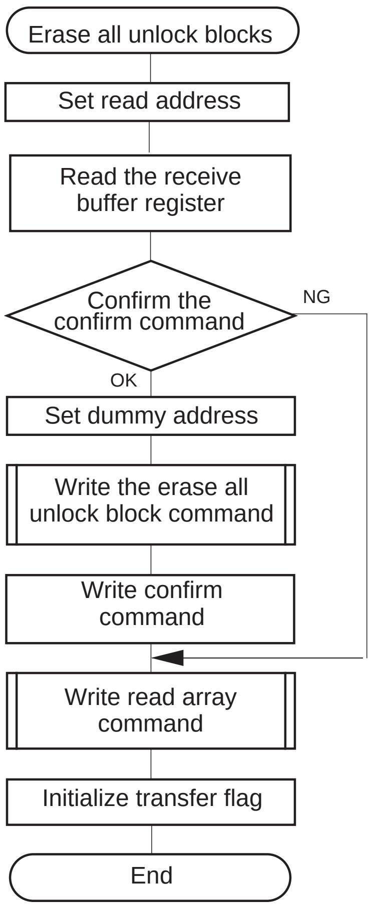


Figure 3.2.14 Erase All Unlock Block

Read Status Register

Two bytes of status data indicating the flash memory's operating status is stored to RAM to transmit via serial I/O.
Write the Read Array command (FF_{16}) to the flash memory, then write the Read Status command (70_{16}).
After status register reception, write the Read Array command and return to the main routine.
Figure 3.2.15 shows a processing flow.

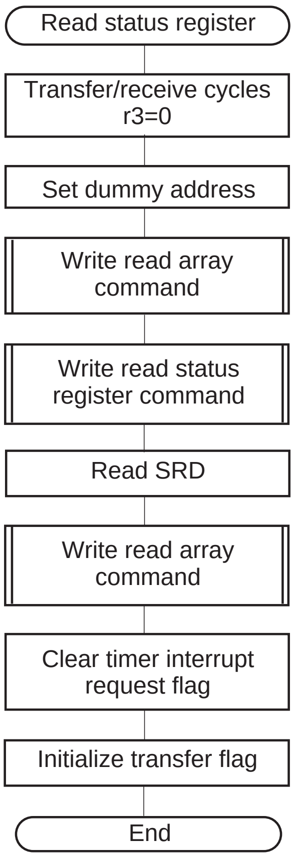


Figure 3.2.15 Read Status Register

Clear Status Register

Status register error information is cleared.
The Read Array command (FF16), Clear Status command (5016) and Read Array command (FF16) are written into the flash memory in succession.
The logic sum for the status register 1 (SRD1) is obtained on #9C16 and the error flag is cleared. Processing returns to the main part.
Figure 3.2.16 shows a processing flow.

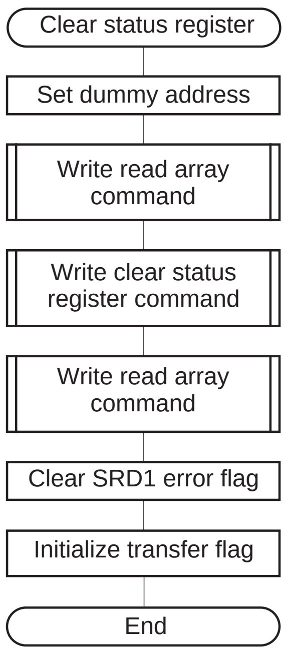


Figure 3.2.16 Clear Status Register

Read Lock Bit Status

One byte of data indicating the lock status of each individual block in the flash memory is saved via serial I/O. Of the 1-byte data, the 6th bit indicates lock status. When "1", the block is unlocked. When "0", the block is locked.
After receiving two byte of data indicating address, store specified address in the address buffer. At this time, set #FE16 to the low order address.
The Read Array command (FF16) and the Read Lock Bit command (7116) are written and, there after, the lock bit information is read from the flash memory. After the lock bit information has been read, the read array command (FF16) is written again. Processing then returns to main part.
Figure 3.2.17 shows a processing flow.

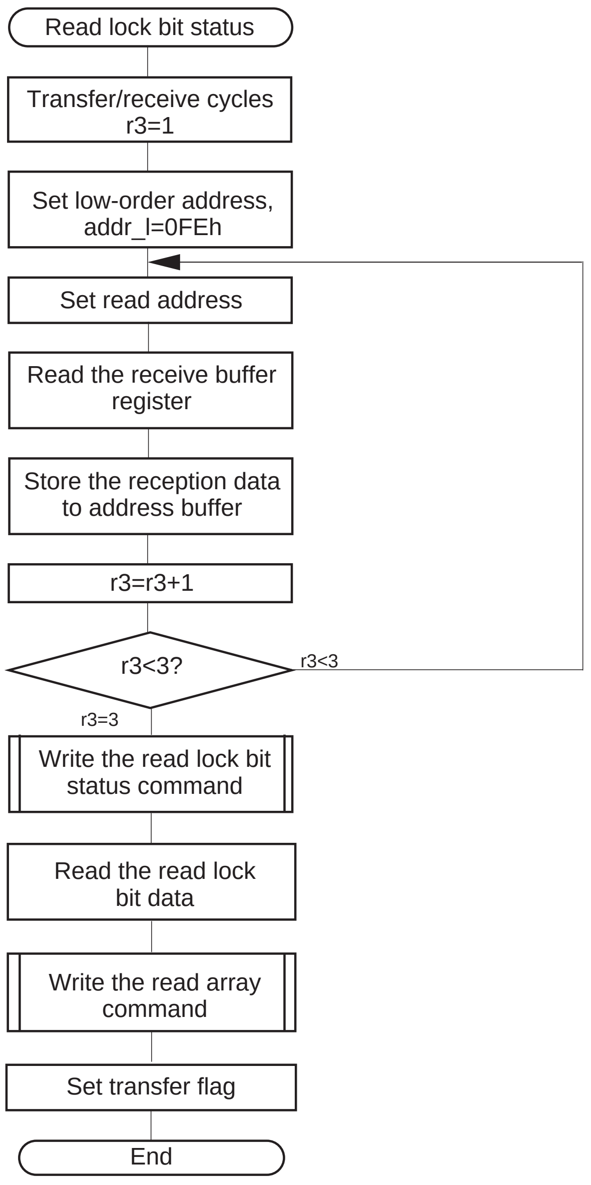

Read lock bit status
Transfer/receive cycles r3=1
Set low-order address, addr_l=0FEh
Set read address
Read the receive buffer register
Store the reception data to address buffer
r3=r3+1
r3<3?
r3<3
r3=3
Write the read lock bit status command
Read the read lock bit data
Write the read array command
Set transfer flag
End

Figure 3.2.17 Read Lock Bit Status

Lock Bit Program

Blocks in the flash memory is locked. Locked block areas cannot be erased.
After receiving two byte of data indicating address, store specified address in the address buffer. At this time, set #FE16 to the low order address.
If the received confirm command is incorrect, lock bit program processing cannot be performed. If correct, for lock bit program processing, write the Lock Bit Program command (7716) to the flash memory and the Confirm command (D016) in succession. Write Read Array command (FF16) and processing returns to the main part.
Figure 3.2.18 shows a processing flow.

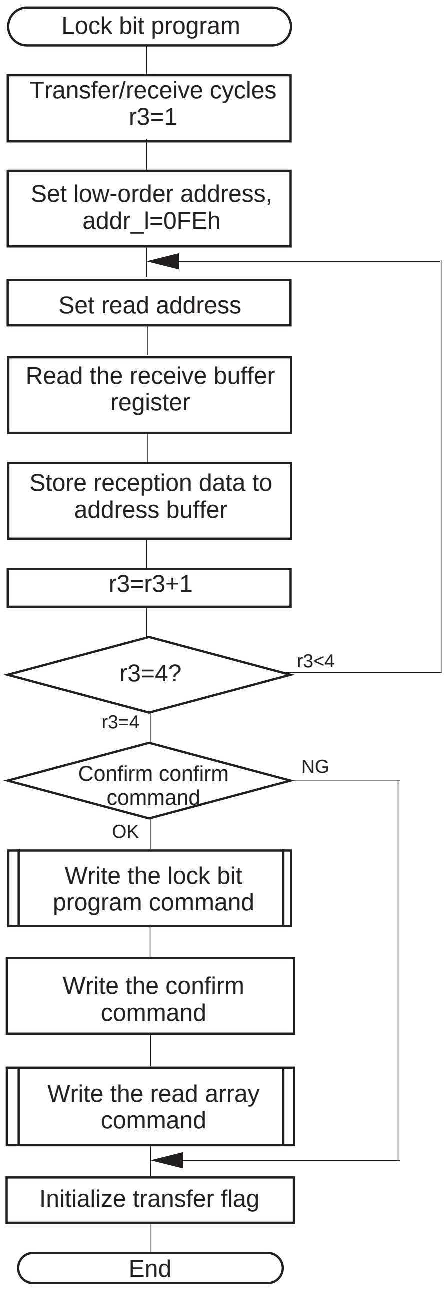

Lock bit program
Transfer/receive cycles r3=1
Set low-order address, addr_l=0FEh
Set read address
Read the receive buffer register
Store reception data to address buffer
r3=r3+1
r3=4?
r3<4
r3=4
Confirm confirm command
NG
OK
Write the lock bit program command
Write the confirm command
Write the read array command
Initialize transfer flag
End

Figure 3.2.18 Lock Bit Program

Lock Bit Enable/Disable

Enables/disables the lock bit function of flash memory. The lock bit disable command cancels the lock on all blocks.
To enable the lock bit, "0" is written for the lock bit cancel bit. To disable the lock bit, "0" followed by "1" is written for the lock bit cancel bit.
Figure 3.2.19 shows a processing flow.

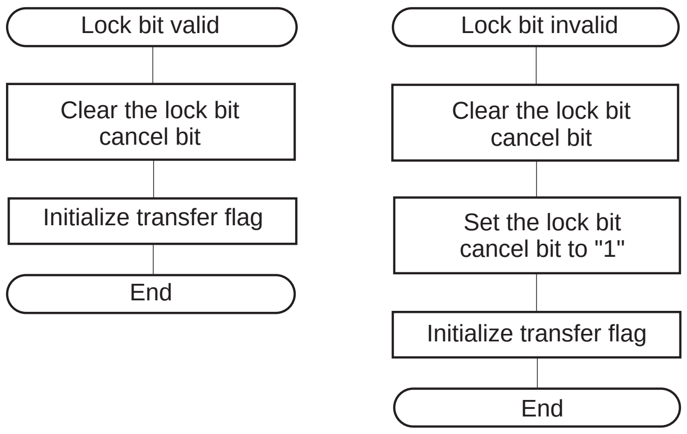

Lock bit valid
Clear the lock bit cancel bit
Initialize transfer flag
End
Lock bit invalid
Clear the lock bit cancel bit
Set the lock bit cancel bit to "1"
Initialize transfer flag
End

Figure 3.2.19 Lock Bit Enable/Disable

ID Check

The ID data stored in the flash memory is compared with the data received by serial I/O. This process judges whether the flash memory is blank or not. When blank, the ID check is ended and processing returns to the main part. When something is written in the ROM, the received ID address, the ID data size and ID data contents are checked. When mismatch, ID check error is generated (SR10 = 1, SR11 = 0) and processing returns to the main part. When match, the ID check is ended (SR10 = 1, SR11 = 1) and processing returns to the main part.
Figure 3.2.20 shows a processing flow.

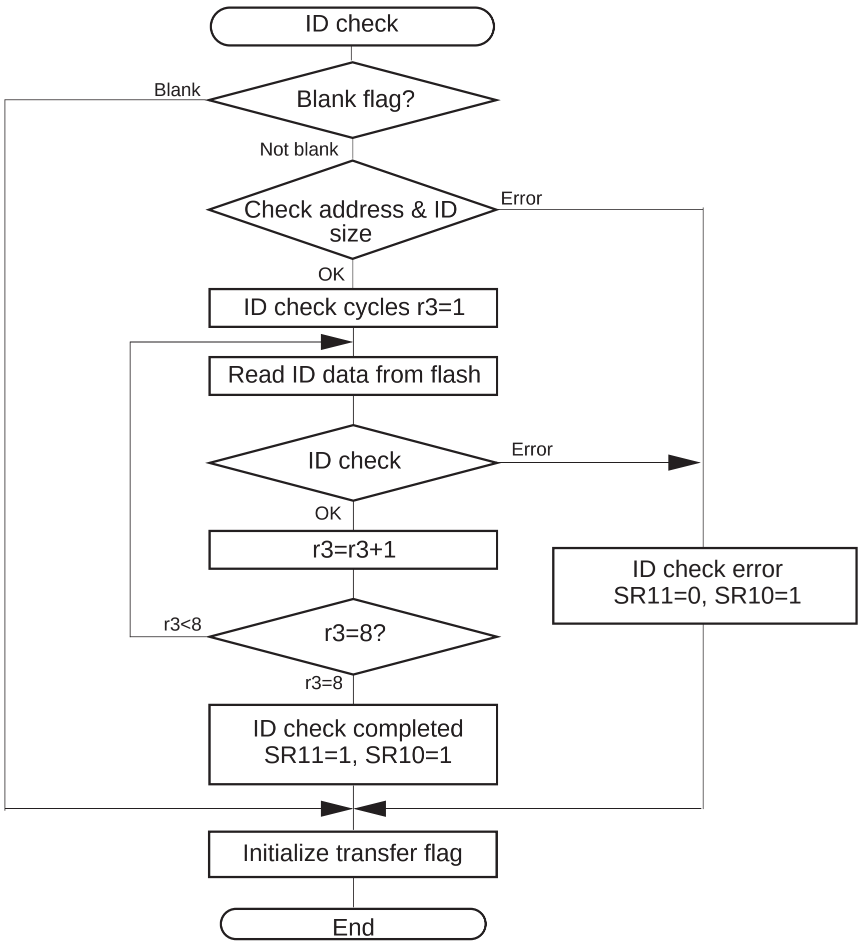


Figure 3.2.20 ID Check

Version Information Output

Transfer flag is set to transfer the version information of the boot program via serial I/O.
Figure 3.2.21 shows a processing flow.

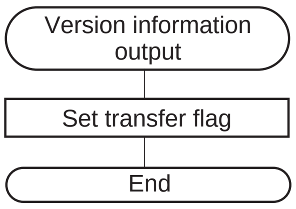


Figure 3.2.21 Version Information Output

Data Transfer Processing

The result of process after receiving a control command from serial programer is transfered via serial I/O. When transfer flag is 0, or time-out flag is 1, the processing returns to the main part. Otherwise next process is executed. Command buffer is read, the serial command is compared, and processing branches to the process in the match commands. After processing, initialize the transfer flag and return to main part. With mismatch command, initialize the transfer flag and return to main part.
Figure 3.2.22 shows a processing flow.

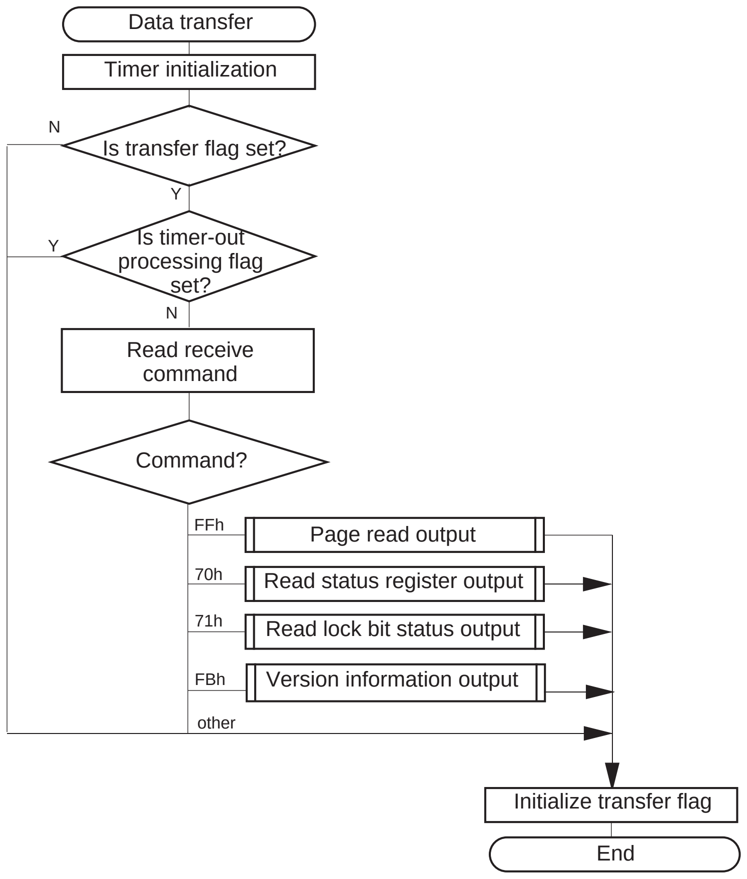


Figure 3.2.22 Data Transfer

Page Read Transfer Processing

Data from the user area in blocks of 256 bytes is read and the read data is sent via serial I/O.
Data is read from the flash memory and set to transfer buffer register. The timer used to check data reception time-out is started. When data is not received within 300 msec, a time-out error is judged, time-out processing flag is set and processing jumps to data transfer processing. After 256 bytes of data is sent, processing jumps to data transfer processing.
Figure 3.2.23 shows a processing flow.

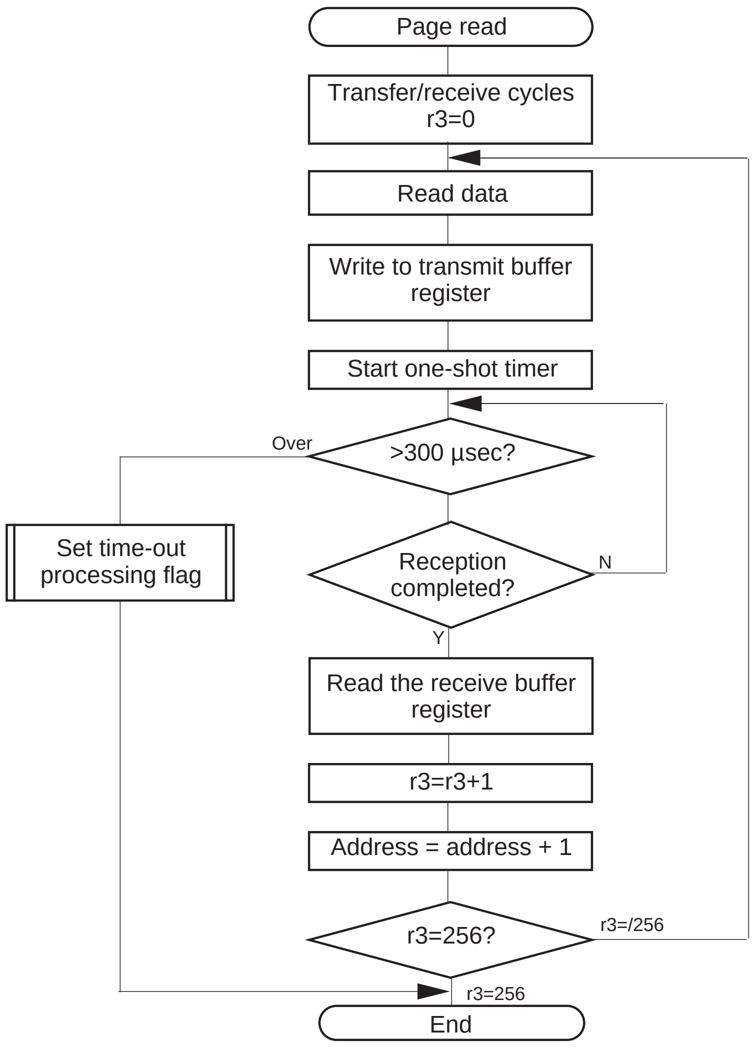


Figure 3.2.23 Page Read Transfer Processing

Read Status Register Transfer Processing

The two-byte status data (SRD: status register and SRD1: status register 1) that indicates flash memory operating status is sent via serial I/O.
The SRD is read from flash memory and written into transmit buffer register. The timer used to check data reception time-out is started. When data is not received within 300 msec, a time-out error is judged, time-out processing flag is set and processing jumps to data transfer processing. After data reception is completed, receive buffer register is read.
The SRD1 is read from flash memory and written into transmit buffer register. The timer used to check data reception time-out is started. When data is not received within 300 msec, a time-out error is judged, time-out processing flag is set and processing jumps to data transfer processing. After data reception is completed, reception buffer register is read and processing returns to data transfer processing.
Figure 3.2.24 shows a processing flow.

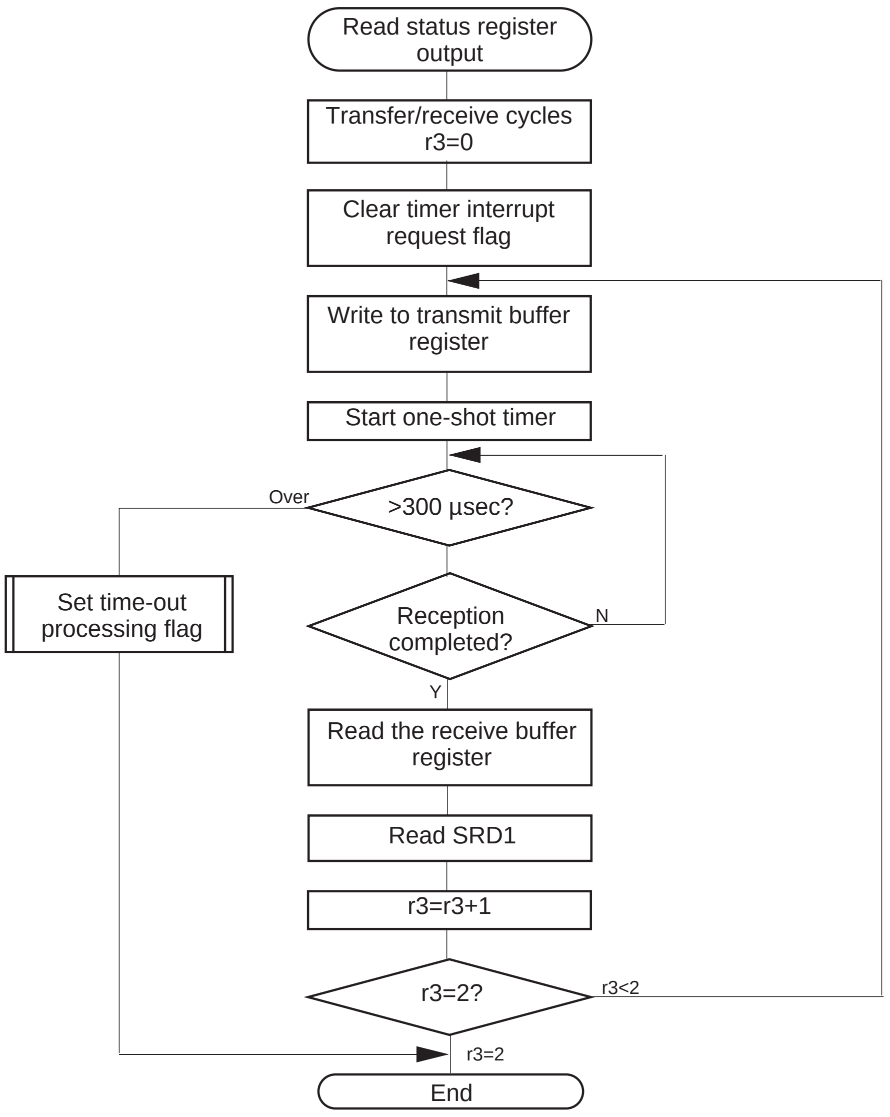

Read status register output
Transfer/receive cycles r3=0
Clear timer interrupt request flag
Write to transmit buffer register
Start one-shot timer
Over
>300 µsec?
Set time-out processing flag
Reception completed?
N
Y
Read the receive buffer register
Read SRD1
r3=r3+1
r3=2?
r3<2
r3=2
End

Figure 3.2.24 Read Status Register Transfer Processing

Read Lock Bit Status Transfer Processing

The lock bit status that set in command processing is sent via serial I/O.
The lock bit status data that set in command processing is read from RAM and written into transmit buffer register. The timer used to check data reception time-out is started. When data is not received within 300 msec, a time-out error is judged, time-out processing flag is set and processing jumps to data transfer processing. After data reception is completed, processing jumps to data transfer processing.
Figure 3.2.25 shows a processing flow.

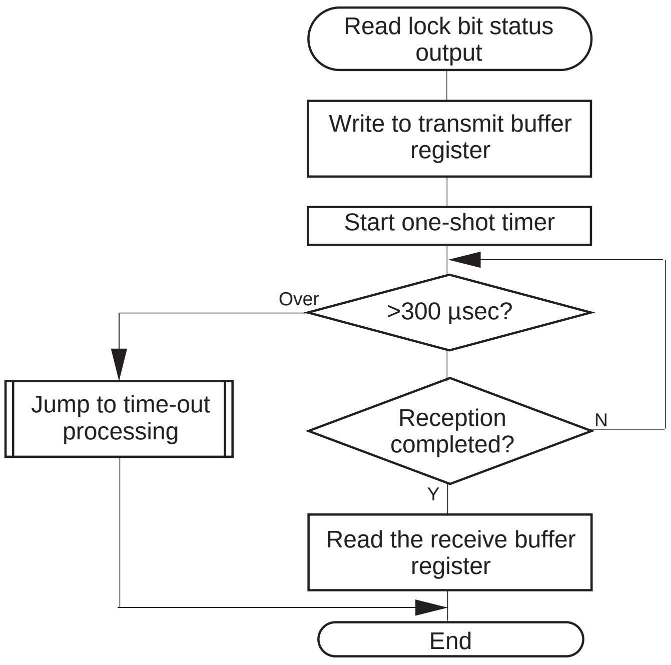

Read lock bit status output
Write to transmit buffer register
Start one-shot timer
Over
>300 µsec?
Jump to time-out processing
Reception completed?
N
Y
Read the receive buffer register
End

Figure 3.2.25 Read Lock Bit Data Transfer Processing

Version Information Output Processing

The version information of boot program is sent via serial I/O.
Version information is read and written in the transmit buffer register.
The timer used to check data reception time-out is started. When data is not received within 300 msec, a time-out error is judged, time-out processing flag is set and processing jumps to data transfer processing.
After all version information is send, processing jumps to data transfer processing.
Figure 3.2.26 shows a processing flow.

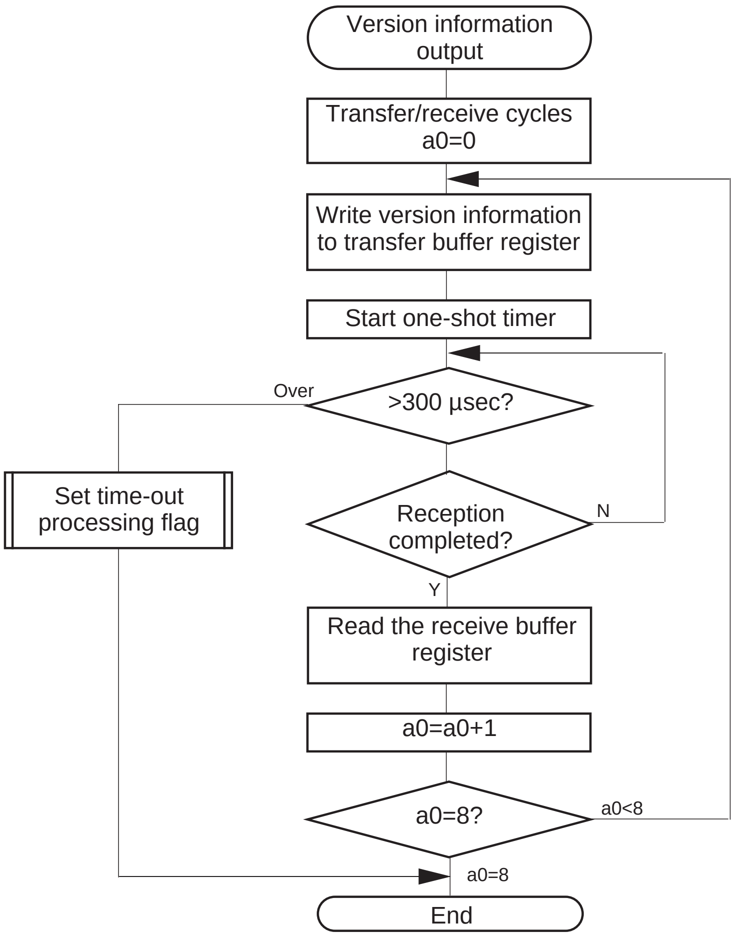

Version information output
Transfer/receive cycles a0=0
Write version information to transfer buffer register
Start one-shot timer
Over
>300 µsec?
Set time-out processing flag
Reception completed?
N
Y
Read the receive buffer register
a0=a0+1
a0=8?
a0<8
a0=8
End

Figure 3.2.26 Version Information Output Processing

Time-Out Processing

When time-out flag is set, serial I/O and time-out flag are initialized.
Figure 3.2.27 shows a processing flow.

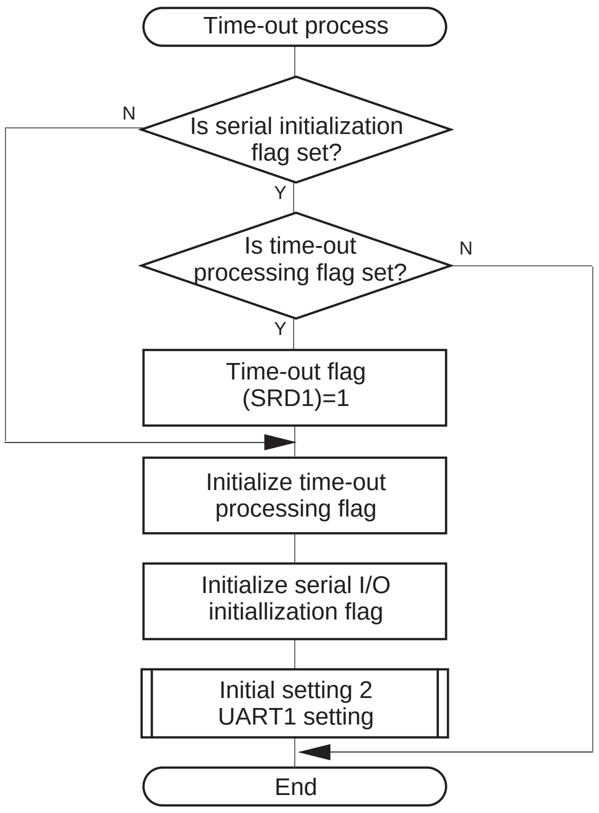

Time-out process
Is serial initialization flag set?
N
Y
Is time-out processing flag set?
N
Y
Time-out flag (SRD1)=1
Initialize time-out processing flag
Initialize serial I/O initiallization flag
Initial setting 2 UART1 setting
End

Figure 3.2.27 Time-Out Processing

Command Write

Commands are written in the flash memory. Commands are accepted when the flash memory is in the ready state (RY/BY signal status flag [bit 0 in address 03B7$_{16}$ of the flash memory control register] is "1").
Figure 3.2.28 shows a processing flow.

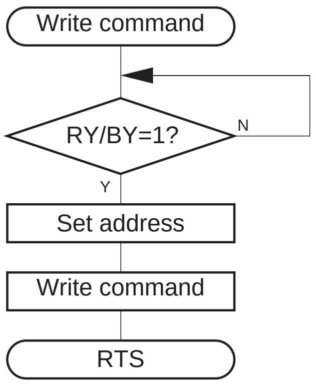


Figure 3.2.28 Command Write

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 writing the Read Status Register command (70_{16}). Also, the status register is cleared by writing the Clear Status Register command (50_{16}). Table 3.2.2 shows the definition of each status register bit. After clearing the reset, the status register outputs "80_{16}".

Table 3.2.2 Status Register (SRD)

Each bit of SRD	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". When the erase status is cleared, it 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". When the program status is cleared, it is set to "0".

Block Status After Program (SR3)

If excessive data is written (phenomenon whereby the memory cell becomes depressed which results in data not being read correctly), "1" is set for the block status after-program at the end of the page write operation. In other words, when writing ends successfully, "80_{16}" is output; when writing fails, "90_{16}" is output; and when excessive data is written, "88_{16}" is output.
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}) and clear the status register.

Status Register 1 (SRD1)

Status register 1 indicates the status of serial communications, results from ID checks and results from check sum comparisons. It can be read after the SRD by writing the Read Status Register command (70_{16}). Also, status register 1 is cleared by writing the Clear Status Register command (50_{16}).
Table 3.2.3 gives the definition of each status register 1 bit. "00_{16}" is output when power is turned ON and the flag status is maintained even after the reset.

Table 3.2.3 Status Register 1 (SRD1)

Each bit of SRD1	Status name	Definition	
		"1"	"0"
SR15 (bit7)	Boot update completed bit	Update completed	Not update
SR14 (bit6)	Reserved	-	-
SR13 (bit5)	Reserved	-	-
SR12 (bit4)	Checksum match bit	Match	Mismatch
SR11 (bit3) SR10 (bit2)	ID check completed bits	00 Not verified 01 Verification mismatch 10 Reserved 11 Verified	
SR9 (bit1)	Data receive time out	Time out	Normal operation
SR8 (bit0)	Reserved	-	-

Boot Update Completed Bit (SR15)

This flag indicates whether the control program was downloaded to the RAM or not, using the download function.

Checksum Match Bit (SR12)

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

ID Check Completed Bits (SR11 and SR10)

These flags indicate the result of ID checks. Some commands cannot be accepted without an ID check.

Data Reception Time Out (SR9)

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

3.3 Sample List

This section shows a sample list of the program described in Section 3.2.
In addition to the processing explained in Section 3.2, the sample shown below includes the programmer command processing used by a synchronous serial programmer and the command processing used by an asynchronous serial communication programmer (M16C Flash Start).

Source

```
;;*********************************************************
;*   System Name    : Sample Program for M16C/62 Flash     *
;*   File Name      : M624SAMP.a30                         *
;*   MCU            : M30624FG(L)FP/GP                     *
;*                  : M30625FG(L)GP                        *
;*   Xin            : 2MHz - 16MHz (for UART mode)         *
;*   Assembler      : AS30 ver 3.00                        *
;*   Linker         : LN30 ver 3.00                        *
;*-------------------------------------------------------*
;*      Copyright,1999 MITSUBISHI ELECTRIC CORPORATION     *
;*      AND MITSUBISHI SEMICONDUCTOR SYSTEM CORPORATION    *
;*-------------------------------------------------------*
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Include file                                        +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    .list       off
    .include    sfr62.inc
    .include    flash624.inc
    .list       on
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Version table                                     +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
    .section    rom,code
    .org        Version
    .byte       'VER.0.00(VER.2.02)'
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Program section start                               +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    .section    prog,code
    .org        Boot_TOP
    .sb         SB_base
    .sbsym      SRD
    .sbsym      SRD1
    .sbsym      ver
    .sbsym      SF
    .sbsym      addr_l
    .sbsym      addr_m
    .sbsym      addr_h

;=========================================================
;+     Boot program start                                  +
;=========================================================
Reset:
;-------------------------------------
;+     Initialize_1                  +
;-------------------------------------
    ldc     #Istack,ISP         ; stack pointer set
    ldc     #SB_base,SB         ; SB register set

    bset    busy
```

```
        bset    busy_d              ; BUSY "H"output
        bclr    s_mode_d            ; Serial mode select input
    ;
    ;---------------------------------------
    ;+     Hot start & RAM clear           +
    ;---------------------------------------
    ;----- RAM Check -----
    RAM_Check:
        mov.b   #0,r1l
        cmp.b   SRD1,SRD1_bak       ; check1
        jeq     RAM_Check2
        cmp.b   SRD1_bak,SRD1_bak+2 ; check2
        jeq     RAM_Check3
        cmp.b   SRD1,SRD1_bak+2     ;check3
        jne     CRC_Check
    ;
    RAM_Check2:
        mov.b   SRD1,r1l            ; r1l <- SRD1
        jmp     CRC_Check
    ;
    RAM_Check3:
        mov.b   SRD1_bak,r1l        ; r1l <- SRD1_bak
        jmp     CRC_Check
    ;
    ;---------------------------------------
    ;+      CRC Check                      +
    ;---------------------------------------
    ;;----- CRC Check -----
    CRC_Check:
        jsr     SUB_CRC             ; Ram data CRC
    ;
        mov.b   Ram_progTOP[a0],r0l ; old CRC code
        mov.b   R0L,crcin
        mov.b   Ram_progTOP+1[a0],r0l
        mov.b   r0l,crcin           ; CRC input data
        mov.w   crcd,r0
        cmp.w   #0,r0
        jne     RAM_clear           ; jump RAM clear
    ;
    ;--------------------------------------
    ;+  UPDATE Check                       +
    ;--------------------------------------
    ;
        bset    ram_check           ; RAM Check OK flag set
        jmp     CPU_set
    ;
    RAM_clear:
        mov.w   #0,r0
        mov.w   #(Ram_END+1-Ram_TOP)/2,r3
        mov.w   #Ram_TOP,a1
        sstr.w
        and.b   #0ch,r1l

    ;
    ;--------------------------------------
    ;+  Processor mode register          +
    ;+    & System clock control register +
    ;--------------------------------------
    CPU_set:
        mov.b   #3,prcr             ; Protect off
        mov.w   #8000h,pm0          ; 1 wait
        mov.w   #6008h,cm0          ; f2
        mov.b   #0,prcr             ; Protect on
    ;
```

```
Reload_chack:
    btst    sr15                ; Update ?
    jc      Transfer_end
    btst    ram_check           ; Reload ?
    jz      Version_inf         ; Yes
;
;---------------------------------------
;+  SI/O Mode Check                    +
;---------------------------------------
;
    btst    s_mode              ; SI/O Mode old = new ?
    bxor    old_mode
    jnc     Transfer_end        ; Yes, jump Transfer_end
;
;---------------------------------------
;+     Version information             +
;---------------------------------------
Version_inf:
    bclr    dwn_flg
    mov.w   #0,a0               ; a0=0
Ver_loop:
    lde.w   Version+9[a0],ver[a0] ; Version data store
    add.w   #2,a0               ; address increment
    cmp.w   #8,a0               ; a0=8 ?
    jltu    Ver_loop            ; jump Ver_loop at a0<8
;
;---------------------------------------
;+     Program_transfer                +
;---------------------------------------
    btst    s_mode              ; Serial I/O mode select
    jz      Transfmcr2          ; UART mode
;
Transfmcr1:
    bset    old_mode            ; clock synchronous mode
    mov.w   #(Trans_TOP1 & 0ffffh),a0   ; Transfer source address (Low)
    mov.b   #(Trans_TOP1 >> 16),r1h     ; Transfer source address (high)
    mov.w   #Ram_progTOP,a1             ; Transfer destination address
    mov.w   #(Trans_END1 - Trans_TOP1)/2,r3 ; Transfer number
    smovf.w                     ; String move
    jmp     Transfer_end0
;
Transfmcr2:
    bclr    old_mode            ; UART mode
    mov.w   #(Trans_TOP2 & 0ffffh),a0   ; Transfer source address (Low)
    mov.b   #(Trans_TOP2 >> 16),r1h     ; Transfer source address (high)
    mov.w   #Ram_progTOP,a1             ; Transfer destination address
    mov.w   #(Trans_END2 - Trans_TOP2)/2,r3 ; Transfer number
    smovf.w                     ; String move
;
Transfer_end0:
    jsr     SUB_CRC             ; Transfer data CRC
    mov.w   crcd,r0             ; CRC code --> r0
    mov.w   r0,Ram_progTOP[a0]
;
Transfer_end:
;---------------------------------------
;+     Jump to RAM                     +
;---------------------------------------
    jmp     Ram_progTOP
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine  :   SUB_CRC                      +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++
SUB_CRC:
    mov.w   #0FFFFh,crcd        ; CRC data register set
```

```
        mov.w   #0,a0
?:
        mov.b   Ram_progTOP[a0],r0l ; Ram data --> rol
        mov.b   r0l,crcin           ; CRC input register
        inc.w   a0
        cmp.w   #Ram_progEND-Ram_progTOP-2,a0
        jne ?-
        rts
;
;---------------------------------------
;+      Download program               +
;---------------------------------------
        .org    Download_program
;
        jsr     set_TA0
;
        mov.w   #0,r3               ; receive number (r3=0)
        mov.w   #0,a1               ; sumcheck buffer
        bclr    sr15                ; Download flag reset
        bclr    sr12                ; Check sum flag reset
Download_loop:
        jsr     SIO_D_rcv
        btst    tout_flg            ; time out error ?
        jc      Download_err        ; jump Download_err at time out
        mov.w   rcv_d,r0            ; receive data read --> r0
        add.w   #1,r3               ; r3 +1 increment
        cmp.w   #3,r3               ; r3=3 ?
        jgtu    Version_store       ; jump Version_store at r3>3
        mov.w   r3,a0               ; r3 --> a0
        mov.b   r0l,addr_l[a0]      ; Store program size
        mov.w   #0,a0               ; a0 initialize
        cmp.w   #3,r3               ; r3 = 3?
        jne     Download_loop       ; No, jump Download_loop
        cmp.w   #0,addr_m           ; program size = 0 ?
        jz      Version_inf         ; jump to Version_inf at program size error
        jmp     Download_loop       ; jump Download_loop
Version_store:
        cmp.w   #11,r3              ; r3=11 ?
        jgtu    Program_store       ; jump Program_store at r3 >11
        mov.b   r0l,ver[a0]         ; version data store to RAM
        jmp     Program_store_1
;
Program_store:
        mov.b   r0l,Ram_progTOP-8[a0]   ; program data store to RAM
Program_store_1:
        add.b   r0l,a1              ; add data to a1
        add.w   #1,a0               ; a0(downloa0 offset) +1 increment
        cmp.w   addr_m,a0           ; a0 = program size (addr_m,h)?
        jltu    Download_loop       ; jump Download_loop at a0< program size
        jmp     Download_CRC        ; jump Download_CRC
;
Download_err:
        bset    busy                ; busy  "H"
        bset    busy_d              ; busy output
        mov.b   #0,u1c1             ; transmit/receive disable
        mov.b   #0,u1mr             ; u1mr reset
        jmp     Version_inf
;
;---------------------------------------
;+      Download program - UART mode - +
;---------------------------------------
        .org    U_Download_program
;
        mov.w   #0,r3               ; receive number (r3=0)
        mov.w   #0,a1               ; sumcheck buffer
```

```
    bclr    sr15                ; Download flag reset
    bclr    sr12                ; Check sum flag reset
U_Download_loop:
    jsr     U_SIO_D_rcv
    mov.w   rcv_d,r0
    add.w   #1,r3               ; r3 +1 increment
    cmp.w   #3,r3               ; r3=3 ?
    jgtu    U_Version_store     ; jump U_Version_store at r3>3
    mov.w   r3,a0               ; r3 --> a0
    mov.b   r0l,addr_l[a0]      ; Store program size
    mov.w   #0,a0               ; a0 initialize
    cmp.w   #3,r3               ; r3 = 3?
    jne     U_Download_loop     ; No, jump U_Download_loop
    cmp.w   #0,addr_m           ; program size = 0?
    jz      Version_inf         ; jump to Version_inf at program size error
    jmp     U_Download_loop
U_Version_store:
    cmp.w   #11,r3              ; r3=11 ?
    jgtu    U_Program_store     ; jump U_Program_store at r3 >11
    mov.b   r0l,ver[a0]         ; version data store to RAM
    jmp     U_Program_store_1
;
U_Program_store:
    mov.b   r0l,Ram_progTOP-8[a0]   ; program data store to RAM
U_Program_store_1:
    add.b   r0l,a1              ; add data to a1
    add.w   #1,a0               ; a0(downloa0 offset) +1 increment
    cmp.w   addr_m,a0           ; a0 = program size (addr_m,h)?
    jltu    U_Download_loop     ; jump  Download_loop at a0< program size
;

Download_CRC:
    mov.w   a1,r0
    cmp.b   data,r0l            ; compare check sum
    bmeq    sr12                ; check sum flag set at data=r0l
    jne     Version_inf         ; jump Version_inf at check sum error
    bset    sr15                ; Download flag set
;
    jsr     SUB_CRC             ; Download data CRC
    mov.w   crcd,r0
    mov.w   r0,Ram_progTOP[a0]
    jmp     Ram_progTOP         ; jump Ram_progTOP

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : a synchronized signal I/O receive dwn+
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SIO_D_rcv:
    mov.b   r1l,u1tb
    bset    ta0os               ; ta0 start
?:
    btst    ir_ta0ic            ; time out error ?
    bmc     sr9                 ; time out flag set
    jc      SIO_D_rcv_err       ; jump SIO_D_rcv_err
    btst    ri_u1c1             ; receive complete ?
    jnc ?-
    mov.w   u1rb,rcv_d          ; receive data read --> r0
SIO_D_rcv_end:
    rts

SIO_D_rcv_err:
    bset    tout_flg
    jmp     SIO_D_rcv_end
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : UART receive dwn                    +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_SIO_D_rcv:
    btst    ri_u1c1             ; receive complete ?
    jnc     U_SIO_D_rcv
    mov.w   u1rb,rcv_d          ; receive data read --> r0
    rts

;===========================================================
;+  Transfer Program -- clock synchronous serial I/O mode +
;+       (1) Main flow                                   +
;+       (2) Flash control program                       +
;+             Read,Program,Erase,All_erase,etc.         +
;+       (3) Other program                               +
;+             ID_check,Download,Version_output etc.     +
;===========================================================
    .section    dump,code
    .org        Trans_TOP1
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Main flow  - clock synchronous serial I/O mode -   +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Main:
    jsr     Initialize_2        ; clock synchronous serial I/O mode
    mov.b   #0,data
Loop_main:
    mov.b   SRD1,SRD1_bak       ; SRD1 back up
    mov.b   SRD1,SRD1_bak+2
;
    jsr     time_init
    jsr     SIO_rcv_first_data
    jsr     Flash_func
    jsr     SIO_send_data
    jsr     Time_out
    jmp     Loop_main
;
;---------------------------------------
;+     initialize       SIO           +
;---------------------------------------
time_init:
    bclr    tout_flg
    bclr    tint_flg
    bset    ta0os
    mov.b   #0,ta0ic
Loop_main1:
    btst    ir_ta0ic            ; 300 usec ?
    jz      Loop_main1
    bset    rcv_flg
    rts
;
;---------------------------------------
;+     SI/O time out                  +
;---------------------------------------
Time_out:
    btst    tint_flg
    jc      Time_out_init
    btst    tout_flg
    jnc     Time_out_end
    bset    sr9                 ; SRD1 time out flag set
    bclr    tout_flg
Time_out_init:
    bclr    tint_flg
    jsr     Initialize_21        ; command error,UART1 reset
Time_out_end:
    rts
;
;---------------------------------------
```

```
;+     SI/O recieve data              +
;---------------------------------------
SIO_rcv_first_data:
    mov.b   #0,cmd_d
    bclr    cmd_flg
    btst    rcv_flg
    jnc     SIO_rcv_end
    btst    tout_flg
    jc      SIO_rcv_end
    mov.b   #0,ta0ic
    mov.w   #0,r2
;
SIO_rcv_first_data_loop:
    mov.b   #0ffh,r1l           ; #ffh --> r1l (transfer data)
    mov.b   r1l,u1tb
    btst    cmd_flg
    jc      SIO_rcv_first_data_loop1
    bclr    busy_d              ; busy input
?:  btst    busy                ; Reception start?
    jz      ?-
SIO_rcv_first_data_loop1:
    bset    ta0os               ; 300 usec timer start
;
SIO_rcv_first_data_loop2:
    btst    ir_ta0ic            ; 300 usec ?
    jnc     ?+
    bset    tout_flg            ; time out
?:  btst    tout_flg
    jc      SIO_rcv_end
    btst    ri_u1c1             ; receive complete ?
    jz      SIO_rcv_first_data_loop2
    mov.w   u1rb,r0             ; receive data --> r0
    mov.w   r2,a0
    mov.b   r0l,data[a0]
    add.w   #1,r2
;
    btst    cmd_flg
    jc      SIO_loop_chk
    bset    cmd_flg
    mov.b   r0l,cmd_d
;
    mov.w   #15,a0
SIO_rcv_command_chk:
    lde.b   Index_tbl-Trans_TOP1+Ram_progTOP-1[a0],r0h
    cmp.b   r0h,r0l
    jeq     SIO_cmd_jmp_2
    sbjnz.w  #1,a0,SIO_rcv_command_chk
    jmp     SIO_rcv_end_1
;
SIO_cmd_jmp_2:
    shl.w   #1,a0
    lde.w   jmp_tbl_2-Trans_TOP1+Ram_progTOP-2[a0],r0
SIO_cmd_jmp_2_1:
    jmpi.w  r0
;
SIO_2:
    mov.w   #2,loop_cnt
    jmp     SIO_loop_chk
SIO_259:
    mov.w   #259,loop_cnt
    jmp     SIO_loop_chk
SIO_4:
    mov.w   #4,loop_cnt
    jmp     SIO_loop_chk
SIO_3:
        mov.w   #3,loop_cnt
        jmp     SIO_loop_chk
;
;---------------------------------------
;+     ID check     SI/O              +
;---------------------------------------
SIO_rcv_ID_check:
        mov.w   #0,r3               ; receive number (r3=0)
        mov.w   #0ffh,a1            ; ID size (dummy data = ffh)
        mov.b   #0,ta0ic
SIO_ID_data_store:
        cmp.w   a1,r3               ; r3=a1(ID size)
        jeq     SIO_ID_address_check; jump ID_address_check at r3=ID size
        mov.b   r1l,u1tb            ; data transfer
        bset    ta0os               ; ta0 start
SIO_ID_data_loop:
        btst    ir_ta0ic            ; 300 usec ?
        jnc     ?+
        bset    tout_flg            ; time out
?:      btst    tout_flg
        jc      SIO_ID_address_check
        btst    ri_u1c1             ; receive complete ?
        jnc     SIO_ID_data_loop
        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 ?
        jne     SIO_ID_data_store   ; jump ID_data_store at r3 not= 4
        mov.b   data,a1             ; ID size --> a1
        add.w   #4,a1               ; a1=a1+4
        jmp     SIO_ID_data_store   ; jump ID_data_store
SIO_ID_address_check:
        jmp     SIO_rcv_end
;
SIO_rcv_end_1:
        bset    tint_flg
        jmp     SIO_rcv_end

SIO_loop_chk:
        cmp.w   loop_cnt,r2
        jltu    SIO_rcv_first_data_loop

SIO_rcv_end:
        bclr    cmd_flg
        bclr    rcv_flg
        rts
;
;---------------------------------------
;+     SIO_send data                  +
;---------------------------------------
SIO_send_data:
        jsr     set_TA0
        btst    send_flg
        jnc     SIO_send_data_end
        btst    tout_flg
        jc      SIO_send_data_end
        mov.b   cmd_d,r1h
;
        cmp.b   #0ffh,r1h           ; Read(ffh)
        jeq     Read_data
        cmp.b   #070h,r1h           ; Read SRD (70h)
        jeq     Read_SRD_data
        cmp.b   #071h,r1h           ; Read LB (71h)
        jeq     Read_LB_data
        cmp.b   #0fbh,r1h           ; Version_output(fbh)
        jeq     Ver_output_data
        cmp.b   #0fdh,r1h           ; Read_check(fdh)
        jeq     Read_check_data
        cmp.b   #0fch,r1h           ; Boot_check(fch)
        jeq     Boot_data
        jmp     SIO_send_func
;
Read_check_data:
        mov.w   #0,r3
        mov.w   sum,r1
Read_check_data_loop:
        mov.b   r1l,u1tb
        bset    ta0os               ; ta0 start
Read_check_data_check:
        btst    ir_ta0ic
        jnc     ?+
        bset    tout_flg
?:
        btst    tout_flg
        jc      SIO_send_data_end
        btst    ri_u1c1             ; receive complete ?
        jnc     Read_check_data_check
        mov.w   u1rb,r0             ; receive data read --> r0
        mov.b   r1h,r1l
        add.w   #1,r3
        cmp.w   #2,r3
        jltu    Read_check_data_loop
Read_check_data_end:
        mov.w   #0,sum              ; reset
        jmp     SIO_send_data_end
;
Read_data:
        mov.w   #0,r3
Read_data_loop:
        lde.b   [a1a0],r1l          ; Flash memory read
        mov.b   r1l,u1tb
        bset    ta0os               ; ta0 start
Read_data_chk:
        btst    ir_ta0ic            ; 300 usec ?
        jnc     ?+
        bset    tout_flg            ; time out
?:
        btst    tout_flg
        jc      Read_data_end
        btst    ri_u1c1             ; receive complete ?
        jnc     Read_data_chk
        mov.w   u1rb,r0             ; receive data read --> r0
        add.w   #1,r3
        add.w   #1,a0
        cmp.w   #256,r3             ; r3 = 256 ?
        jne     Read_data_loop
Read_data_end:
        jmp     SIO_send_data_end
;
Ver_output_data:
        mov.w   #0,a0               ; Version address offset (a0=0)
Ver_output_data_loop:
        mov.b   ver[a0],u1tb        ;send_data set
        bset    ta0os               ; ta0 start
Ver_output_data_check:
        btst    ir_ta0ic            ; 300 usec ?
        jnc     ?+
        bset    tout_flg            ; time out
?:
        btst    tout_flg
        jc      Ver_output_data_end
        btst    ri_u1c1             ; receive complete ?
        jnc     Ver_output_data_check
        mov.w   u1rb,r0             ; receive data read --> r0
        add.w   #1,a0
        cmp.w   #8,a0               ; a0 = 8 ?
        jne     Ver_output_data_loop
Ver_output_data_end:
        jmp     SIO_send_data_end
;
Read_SRD_data:
        mov.w   #0,r3
Read_SRD_data_loop:
        bclr    tout_flg            ; clear time out
        mov.b   #0,ta0ic            ; clear time out
        mov.b   r1l,u1tb            ; data transfer
        bset    ta0os               ; ta0 start ; test
Read_SRD_data_check:
        btst    ir_ta0ic            ; 300 usec ?
        jnc     ?+
        bset    tout_flg            ; time out
?:  btst    tout_flg
        jc      Read_SRD_data_end
        btst    ri_u1c1             ; receive complete ?
        jnc     Read_SRD_data_check
        mov.w   u1rb,r0             ; receive data read --> r0
        mov.b   SRD1,r1l            ; SRD1 data --> r1l
        add.w   #1,r3
        cmp.w   #2,r3               ; r3 = 2 ?
        jltu    Read_SRD_data_loop  ; jump Read_SRD_loop at r3<2
Read_SRD_data_end:
        jmp     SIO_send_data_end
;
Read_LB_data:
Read_LB_data_loop:
        mov.b   r1l,u1tb            ; data transfer
        bset    ta0os               ; ta0 start
Read_LB_data_check:
        btst    ir_ta0ic            ; 300 usec ?
        jnc     ?+
        bset    tout_flg            ; time out
?:
        btst    tout_flg
        jc      Read_LB_data_end
        btst    ri_u1c1             ; receive complete ?
        jnc     Read_LB_data_check
        mov.w   u1rb,r0             ; receive data read --> r0
Read_LB_data_end:
        jmp     SIO_send_data_end
;
Boot_data:
        bclr    fmcr5
        mov.w   addr_l,a0
        mov.b   addr_h,a1
        mov.w   #0,r3
Boot_data_loop:
        lde.b   [a1a0],r1l          ; Flash memory read
        mov.b   r1l,u1tb
        bset    ta0os               ; ta0 start
Boot_data_chk:
        btst    ir_ta0ic            ; 300 usec ?
        jnc     ?+
        bset    tout_flg            ; time out
?:
        btst    tout_flg
        jc      Boot_data_end
        btst    ri_u1c1             ; receive complete ?
        jnc     Boot_data_chk
        mov.w   u1rb,r0             ; receive data read --> r0
        add.w   #1,r3
        add.w   #1,a0
        cmp.w   #256,r3             ; r3 = 256 ?
        jne     Boot_data_loop
Boot_data_end:
        bset    fmcr5
        jmp     SIO_send_data_end
;
SIO_send_func:
        mov.w   start_cnt,r3
SIO_send_data_loop:
        mov.w   r3,a0
        mov.b   data[a0],r1l
        mov.b   r1l,u1tb            ; data transfer
        bset    ta0os               ; ta0 start
SIO_send_chk:
        btst    ir_ta0ic            ; 300 usec ?
        jnc     ?+
        bset    tout_flg            ; time out
?:      btst    tout_flg
        jc      SIO_send_data_end
        btst    ri_u1c1             ; receive complete ?
        jnc     SIO_send_chk
        mov.w   u1rb,r0             ; receive data read --> r0
        add.w   #1,r3
        cmp.w   send_cnt,r3         ; r3 = send_cnt ?
        jne     SIO_send_data_loop
        mov.w   r3,r0
SIO_send_data_end:
        bclr    send_flg
        rts
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : Time_over_flg                         +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Time_over_flg:
        bset    tout_flg
        rts

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      jump table for Flash_func                          +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
jmp_tbl:
        .word   Read - cmd_jmp
        .word   Program - cmd_jmp
        .word   Erase - cmd_jmp
        .word   All_erase - cmd_jmp
        .word   Clear_SRD - cmd_jmp
        .word   Read_LB - cmd_jmp
        .word   Program_LB - cmd_jmp
        .word   LB_enable - cmd_jmp
        .word   LB_disable - cmd_jmp
        .word   Download - cmd_jmp
        .word   Boot_output - cmd_jmp
        .word   Read_check - cmd_jmp

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      jump table for SIO_rcv_first_data                  +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
jmp_tbl_2:
        .word   SIO_3 - SIO_cmd_jmp_2_1         ; Read
        .word   SIO_259 - SIO_cmd_jmp_2_1       ; Program
        .word   SIO_4 - SIO_cmd_jmp_2_1         ; erase
        .word   SIO_2 - SIO_cmd_jmp_2_1         ; All erase
        .word   SIO_rcv_end - SIO_cmd_jmp_2_1   ; Clear SRD
        .word   SIO_3 - SIO_cmd_jmp_2_1         ; Read LB
        .word   SIO_4 - SIO_cmd_jmp_2_1         ; LB Program
        .word   SIO_rcv_end - SIO_cmd_jmp_2_1   ; LB enable
        .word   SIO_rcv_end - SIO_cmd_jmp_2_1   ; LB disable
        .word   SIO_rcv_end - SIO_cmd_jmp_2_1   ; Download
        .word   SIO_3 - SIO_cmd_jmp_2_1         ; Boot output
        .word   SIO_rcv_end - SIO_cmd_jmp_2_1   ; Read check
        .word   SIO_rcv_end - SIO_cmd_jmp_2_1   ; Read SRD
        .word   SIO_rcv_ID_check - SIO_cmd_jmp_2_1 ; ID check
        .word   SIO_rcv_end - SIO_cmd_jmp_2_1   ;  Version out
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     serch table for Flash_func,SIO_rcv_first_data      +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Index_tbl:
        .byte   0ffh                ; Read(ffh)
        .byte   041h                ; Program(41h)
        .byte   020h                ; Erase(20h)
        .byte   0a7h                ; All_erase(a7h)
        .byte   050h                ; Clear SRD(50h)
        .byte   071h                ; Read LBS(71h)
        .byte   077h                ; LB program(77h)
        .byte   07ah                ; LB enable (7ah)
        .byte   075h                ; LB disable(75h)
        .byte   0fah                ; Download (fah)
        .byte   0fch                ; Boot output(fch)
        .byte   0fdh                ; Read check(fdh)
        .byte   070h                ; Read SRD(70h)
        .byte   0f5h                ; ID check(f5h)
        .byte   0fbh                ; Version output(fbh)
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Subroutine : Initialize_2                          +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Initialize_2:
;-------------------------------------
;+     Flash mode set                +
;-------------------------------------
;
        bset    fmcr5               ; User ROM select
        bclr    fmcr1               ; Flash entry bit clear
        bset    fmcr1               ; Flash entry bit set (E/W mode)
;
;-------------------------------------
;+     Blank check                   +
;-------------------------------------
        lde.w   0ffffch,r0          ; Reset vector read
        lde.w   0ffffeh,r1          ; Reset vector read
        and.w   r1,r0               ; r0 & r1
        cmp.w   #0ffffh,r0          ; r0=ffffh ?
        jne     blank_end
        bset    sr10                ; check complete at r0=ffffh
        bset    sr11
        bset    blank               ; blank flag set
blank_end:
;-------------------------------------
;+     UART1                         +
;-------------------------------------
Initialize_21:
;----- UART1 transmit/receive mode register
```

```
;
    bset    busy                ; busy  "H"
    bset    busy_d              ; busy output
    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 A1                       +
;--------------------------------------
set_TA0:
;----- Timer A1 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
;
    rts
;
;--------------------------------------
;+     FLASH function main            +
;--------------------------------------
Flash_func:
    btst    tout_flg
    jc      Flash_func_end
    bclr    ta0s
    mov.b   cmd_d,r0l           ; receive data --> r0l
;
```

```
    mov.b   #0ch,r0h            ; #00001100b sr10,11 mask data
    and.b   SRD1,r0h            ; sr10,11 pick up
    cmp.b   #0ch,r0h            ; ID check OK?
    jne     Command_check_2     ; jump Command_check_2 at ID unchecked
    mov.w   #12,a0
;
Command_check:
    lde.b   Index_tbl-Trans_TOP1+Ram_progTOP-1[a0],r0h
    cmp.b   r0h,r0l
    jeq     cmd_jmp_1
    sbjnz.w  #1,a0,Command_check
    jmp     Command_check_2
;
cmd_jmp_1:
    shl.w   #1,a0
    lde.w   jmp_tbl-Trans_TOP1+Ram_progTOP-2[a0],r0
cmd_jmp:
    jmpi.w  r0
;
Command_check_2:
?:  cmp.b   #070h,r0l           ; Read SRD  (70h)
    jne     ?+
    jmp     Read_SRD
?:  cmp.b   #0f5h,r0l           ; ID check  (f5h)
    jne     ?+
    jmp     ID_check
?:  cmp.b   #0fbh,r0l           ; Version out   (fbh)
    jne     Flash_func_end
    jmp     Ver_output
;
Flash_func_end:
    rts
;
;-------------------------------------
;+     Read                         +
;-------------------------------------
Read:
    mov.w   #0,r3               ; receive number
    mov.b   #0,addr_l           ; addr_l = 0
Read_loop:
    add.w   #1,r3               ; r3 +1 increment
    mov.w   r3,a0               ; r3 --> a0
    mov.w   data[a0],r0
    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   #00ffh,r2           ; Read array command
    jsr     Command_write       ; command_write
    bset    send_flg
    mov.w   #3,start_cnt
    mov.w   #258,send_cnt
    jmp     Flash_func_end      ; jump  Flash_func_end
;
;-------------------------------------
;+     Program                      +
;-------------------------------------
Program:
    mov.w   #0,r3               ; receive number
    mov.b   #0,addr_l           ; addr_l = 0
    mov.w   sum,crcd            ; for Read check command
Program_loop_1:
    add.w   #1,r3               ; r3 +1 increment
    mov.w   r3,a0               ; r3 --> a0
    mov.b   data[a0],r0l
```

```
        mov.b   r0l,addr_l[a0]      ; Store address
        cmp.w   #259,r3             ; r3 = 259 ?
        jltu    Program_loop_1      ; jump Program_loop_1 at r3<258
    ;
        mov.w   #00ffh,r2           ; Read array command
        jsr     Command_write       ; command_write
        mov.w   #0070h,r2           ; Read SRD command
        jsr     Command_write       ; Command write
        lde.w   [a1a0],r1           ; SRD read
        mov.w   #00ffh,r2           ; Read array command
        jsr     Command_write       ; command_write
        cmp.b   #80h,r1l            ; error check
        jne     Program_end
    ;
        mov.w   #0041h,r2           ; Page program command
        jsr     Command_write       ; command_write
        mov.w   #0,r3               ; writing number (r3=0)
        mov.b   addr_h,a1           ; addr_h   --> a1
    Program_loop_2:
        mov.w   r3,a0               ; r3       --> a0
        mov.w   data[a0],r1         ; data     --> r1
        mov.w   addr_l,a0           ; addr_l,m --> a0
        ste.w   r1,[a1a0]           ; 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    Program_loop_2      ; jump Program_loop_2 at r3<255
    Program_end:
        mov.w   crcd,sum            ; for Read check command
        bclr    send_flg
        mov.w   #0,send_cnt
        mov.w   #0,start_cnt
        jmp     Flash_func_end      ; jump Flash_func_end
    ;
    ;-------------------------------------
    ;+     Block erase                   +
    ;-------------------------------------
    Erase:
        mov.w   #1,r3               ; receive number (r3=1)
        mov.b   #0feh,addr_l        ; addr_l = ffh
    Erase_loop:
        mov.w   r3,a0               ; r3 --> a0
        mov.b   data[a0],r0l
        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
        mov.w   #0020h,r2           ; Erase command
        jsr     Command_write       ; command write
        mov.w   #00d0h,r2           ; Confirm command
        ste.w   r2,[a1a0]           ; command write
    Erase_end:
        mov.w   #00ffh,r2           ; Read array command
        jsr     Command_write       ; command_write
        bclr    send_flg
        mov.w   #0,send_cnt
        mov.w   #0,start_cnt
        jmp     Flash_func_end      ; jump Flash_func_end
    ;
```

```
;-------------------------------------
;+      All erase ( unlock block )     +
;-------------------------------------
All_erase:
    mov.w   #1,a0
    mov.b   data[a0],r0l        ; receive data read --> r0
    cmp.b   #0d0h,r0l           ; Confirm command check
    jne     All_erase_end       ; jump All_erase_end at Confirm command error
    mov.w   #0000h,addr_l       ; 0f0000h --> addr
    mov.b   #000fh,addr_h
    mov.w   #00a7h,r2           ; All erase command
    jsr     Command_write       ; command write
    mov.w   #00d0h,r2           ; Confirm command
    ste.w   r2,[a1a0]           ; command write
All_erase_end:
    mov.w   #00ffh,r2           ; Read array command
    jsr     Command_write       ; command_write
    bclr    send_flg
    mov.w   #0,send_cnt
    mov.w   #0,start_cnt
    jmp     Flash_func_end      ; jump Flash_func_end
;
;-------------------------------------
;+     Read SRD                       +
;-------------------------------------
Read_SRD:
    mov.w   #0,r3               ; receive number (r3=0)
    mov.w   #0000h,addr_l       ; 0f0000h --> addr
    mov.b   #000fh,addr_h
    mov.w   #00ffh,r2           ; Read array command
    jsr     Command_write       ; command_write
    mov.w   #0070h,r2           ; Read SRD command
    jsr     Command_write       ; command write
    lde.w   [a1a0],r1           ; SRD read
    mov.w   #00ffh,r2           ; Read array command
    jsr     Command_write       ; command_write
    mov.w   #1,start_cnt
    mov.w   #3,send_cnt
    bset    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;-------------------------------------
;+     Clear SRD                      +
;-------------------------------------
Clear_SRD:
    mov.w   #0000h,addr_l       ; 0f0000h --> addr
    mov.b   #000fh,addr_h
    mov.w   #00ffh,r2           ; Read array command
    jsr     Command_write       ; command write
    mov.w   #0050h,r2           ; Clear SRD command
    jsr     Command_write       ; command write
    mov.w   #00ffh,r2           ; Read array command
    jsr     Command_write       ; command write
    and.b   #10011100b,SRD1     ; SRD1 clear
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;-------------------------------------
;+     Read Lock Bit                  +
;-------------------------------------
Read_LB:
    mov.w   #1,r3               ; receive number (r3=1)
    mov.b   #0feh,addr_l        ; addr_l = ffh
```

```
Read_LB_loop:
    mov.w   r3,a0               ; r3 --> a0
    mov.b   data[a0],r0l
    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
    mov.w   #0071h,r2           ; Read LB command
    jsr     Command_write       ; command write
    lde.w   [a1a0],r1           ; read LB
    mov.w   #00ffh,r2           ; Read array command
    jsr     Command_write       ; command write
Read_LB_end:
    mov.w   #1,start_cnt
    mov.w   #1,send_cnt
    bset    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;---------------------------------------
;+     Program Lock Bit                +
;---------------------------------------
Program_LB:
    mov.w   #1,r3               ; receive number (r3=1)
    mov.b   #0feh,addr_l        ; addr_l = ffh
Program_LB_loop:
    mov.w   r3,a0               ; r3 --> a0
    mov.b   data[a0],r0l
    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
    mov.w   #0077h,r2           ; Program LB command
    jsr     Command_write       ; command write
    mov.w   #00d0h,r2           ; Confirm command
    ste.w   r2,[a1a0]           ; command write
    mov.w   #00ffh,r2           ; Read array command
    jsr     Command_write       ; command write
Program_LB_end:
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;---------------------------------------
;+     Lock Bit enable                 +
;---------------------------------------
LB_enable:
    bclr    fmcr2               ; Lock disable bit = 0
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;---------------------------------------
;+     Lock Bit disable                +
;---------------------------------------
LB_disable:
    bclr    fmcr2               ; Lock disable bit = 0
    bset    fmcr2               ; Lock disable Bit = 1
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
```

```
;
;-------------------------------------
;+     ID check                      +
;-------------------------------------
ID_check:
    btst    blank               ; blank flag check
    jc      ID_check_end        ; jump ID_check_end at blank
    cmp.w   #0ffdfh,addr_l      ; lower ID address check
    jne     ID_error            ; jump ID_error at ID address error
    cmp.w   #0070fh,addr_h      ; higher ID address check
    jne     ID_error            ; jump ID_error at ID address error
ID_data_check:
    mov.w   #0000fh,a1          ; ID higher address --> a1
    mov.w   #0ffdfh,r1          ; ID lower address --> r1
    mov.w   #1,r3               ; check loop number (r3=1)
ID_check_loop:
    mov.w   r1,a0               ; r1 --> a0
    lde.b   [a1a0],r0l          ; ID data read from Flash memory
    mov.w   r3,a0               ; r3 --> a0
    cmp.b   r0l,data[a0]        ; compare ID data
    jne     ID_error            ; jump ID_error at ID error
    add.w   #4,r1               ; r1 +4 increment (next ID address)
    cmp.w   #0ffe7h,r1          ; r1=0ffefh ?
    jne     ?+                  ; jump ? at not equal
    mov.w   #0ffebh,r1          ; r1=0ffeb at equal
?:
    add.w   #1,r3               ; r3 +1 increment
    cmp.w   #8,r3               ; r3=8 ?
    jltu    ID_check_loop       ; jump ID_check_loop at r3<8
ID_OK:
    bset    sr10
    bset    sr11                ; ID check OK (sr11=1,sr10=1)
    jmp     ID_check_end        ; jump  ID_check_end
ID_error:
    bset    sr10
    bclr    sr11                ; ID error (sr11=0,sr10=1)
ID_check_end:
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;-------------------------------------
;+     Boot output                   +
;-------------------------------------
Boot_output:
    bclr    fmcr5               ; Boot ROM select
    mov.w   #0,r3               ; receive number
    mov.b   #0,addr_l           ; addr_l = 0
Boot_loop:
    add.w   #1,r3               ; r3 +1 increment
    mov.w   r3,a0               ; r3 --> a0
    mov.w   data[a0],r0
    mov.b   r0l,addr_l[a0]      ; Store address
    cmp.w   #2,r3               ; r3 = 2 ?
    jltu    Boot_loop           ; jump Read_loop at r3<2
    bset    send_flg
    mov.w   #3,start_cnt
    mov.w   #258,send_cnt
    jmp     Flash_func_end      ; jump  Flash_func_end
;
;-------------------------------------
;+     Read check                    +
;-------------------------------------
Read_check:
    mov.w   #0,start_cnt
    mov.w   #2,send_cnt
    bset    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;---------------------------------------
;+      Download                       +
;---------------------------------------
Download:
    bclr    fmcr5               ; Boot ROM select
    jmp.a   Download_program    ; jump Download_program
;
;---------------------------------------
;+      Version output                 +
;---------------------------------------
Ver_output:
    mov.w   #0,start_cnt
    mov.w   #8,send_cnt
    bset    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+       Subroutine : Command write                        +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Command_write:
    btst    fmcr0               ; RY/BY status check
    jz      Command_write
    mov.w   addr_l,a0           ; addr_l,m --> a0
    mov.b   addr_h,a1           ; addr_h   --> a1
    ste.w   r2,[a1a0]           ; command write
    rts
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+       Subroutine : a synchronized signal I/O receive data+
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SIO_rcv_data:
    jsr     set_TA0
SIO_rcv_data_1:
    btst    ir_ta0ic            ; time out error ?
    jnc     ?+
    jsr     Time_over_flg       ; jump Time_over at time out
?:
    btst    ri_u1c1             ; receive complete ?
    jnc     SIO_rcv_data_1
    mov.w   u1rb,rcv_d          ; receive data read --> r0
    rts
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+       Subroutine : a synchronized signal I/O receive data+
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SIO_rcv_data_rom:
    jsr     set_TA0
SIO_rcv_data_rom_1:
    btst    ir_ta0ic            ; time out error ?
    bmc     fmcr5               ; time out, User ROM select
    jnc     ?+
    jsr     Time_over_flg       ; jump Time_over at time out
?:
    btst    ri_u1c1             ; receive complete ?
    jnc     SIO_rcv_data_rom_1
    mov.w   u1rb,rcv_d          ; receive data read --> r0
    rts
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+       Subroutine : a synchronized signal I/O send       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SIO_send:
    jsr     set_TA0
    jsr     SIO_send_data
    jsr     SIO_rcv_data
    rts
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : a synchronized signal I/O send       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SIO_send_rom:
    jsr     set_TA0
    jsr     SIO_send_data
    jsr     SIO_rcv_data_rom
    rts
;
;===========================================================
;+  Transfer Program -- UART mode                          +
;+       (1) Main flow                                     +
;+       (2) Flash control program                         +
;+             Read,Program,All_erase,Read_SRD,Clear_SRD  +
;+       (3) Other program                                 +
;+             ID_check                                    +
;===========================================================
;
    .org        Trans_TOP2
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Main flow  - UART mode -                            +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_Main:
    jmp     U_SIO_init_first
;
U_Loop_main:
    mov.b   SRD1,SRD1_bak       ; SRD1 back up
    mov.b   SRD1,SRD1_bak+2
;
    jsr     U_SIO_rcv
    mov.w   rcv_d,r0
    mov.b   r0l,cmd_d
    mov.w   #0,r2
    mov.w   r2,a0
    mov.b   r0l,data[a0]
    bclr    cmd_flg
;
    jmp     U_SIO_freq

    jsr     U_time_init

    jmp     U_SIO_rcv_first_data
U_Flash_set:
    jmp     U_Flash_func
U_Flash_send:
    jmp     U_SIO_send_data
U_Flash_int:
    btst    tint_flg
    jnc     U_Main_end
    jsr     Initialize_31       ; command error,UART mode Initialize
;
U_Main_end:
    jmp     U_Loop_main         ; jump U_Loop_main
;
;-------------------------------------
;+    initialize       SIO           +
;-------------------------------------
```

```
U_time_init:
    bset    rcv_flg
    bclr    tint_flg
    rts
;
;---------------------------------------
;+     SI/O recieve data              +
;---------------------------------------
U_SIO_rcv_first_data:
    btst    rcv_flg
    jnc     U_SIO_rcv_end
    jc      U_SIO_rcv_first_data_set

U_SIO_rcv_first_data_loop:
    jsr     U_SIO_rcv_only
    mov.w   rcv_d,r0            ; receive data --> r0
U_SIO_rcv_first_data_set:
    mov.w   r2,a0
    mov.b   r0l,data[a0]
    add.w   #1,r2
;
    btst    cmd_flg
    jc      U_SIO_loop_chk
    bset    cmd_flg
    mov.b   r0l,cmd_d
;
    mov.w   #19,a0
U_SIO_rcv_command_chk:
    lde.b   U_Index_tbl-Trans_TOP2+Ram_progTOP-1[a0],r0h
    cmp.b   r0h,r0l
    jeq     U_SIO_cmd_jmp_2
    sbjnz.w  #1,a0,U_SIO_rcv_command_chk
    jmp     U_SIO_rcv_end

U_SIO_cmd_jmp_2:
    shl.w   #1,a0
    lde.w   U_jmp_tbl_2-Trans_TOP2+Ram_progTOP-2[a0],r0
U_SIO_cmd_jmp_2_1:
    jmpi.w  r0

U_SIO_2:
    mov.w   #2,loop_cnt
    jmp     U_SIO_loop_chk
U_SIO_259:
    mov.w   #259,loop_cnt
    jmp     U_SIO_loop_chk
U_SIO_4:
    mov.w   #4,loop_cnt
    jmp     U_SIO_loop_chk
U_SIO_3:
    mov.w   #3,loop_cnt
    jmp     U_SIO_loop_chk
;
;---------------------------------------
;+     ID check     SI/O              +
;---------------------------------------
U_SIO_rcv_ID_check:
    mov.w   #0,r3               ; receive number (r3=0)
    mov.w   #0ffh,a1            ; ID size (dummy data = ffh)
    mov.b   #0,ta0ic
U_SIO_ID_data_store:
    cmp.w   a1,r3               ; r3=a1(ID size)
    jeq     U_SIO_ID_address_check; jump ID_address_check at r3=ID size
    jsr     U_SIO_rcv_only
    mov.w   rcv_d,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 ?
    jne     U_SIO_ID_data_store ; jump ID_data_store at r3 not= 4
    mov.b   data,a1             ; ID size --> a1
    add.w   #4,a1               ; a1=a1+4
    jmp     U_SIO_ID_data_store ; jump ID_data_store
U_SIO_ID_address_check:
    jmp     U_SIO_rcv_end
;
U_SIO_rcv_end_1:
    bset    tint_flg
    jmp     U_SIO_rcv_end

U_SIO_loop_chk:
    cmp.w   loop_cnt,r2
    jltu    U_SIO_rcv_first_data_loop
U_SIO_rcv_end:
    bclr    cmd_flg
    bclr    rcv_flg
    jmp     U_Flash_set
;
;-------------------------------------
;+     SIO_send data                  +
;-------------------------------------
U_SIO_send_data:
    btst    send_flg
    jnc     U_SIO_send_data_end
    mov.b   cmd_d,r1h

    cmp.b   #0ffh,r1h           ; Read(ffh)
    jeq     U_Read_data
    cmp.b   #070h,r1h           ; Read SRD (70h)
    jeq     U_Read_SRD_data
    cmp.b   #071h,r1h           ; Read LB (71h)
    jeq     U_Read_LB_data
    cmp.b   #0fbh,r1h           ; Version_output(fbh)
    jeq     U_Ver_output_data
    cmp.b   #0fdh,r1h           ; Read_check(fdh)
    jeq     U_Read_check_data
    cmp.b   #0fch,r1h           ; Boot_check(fch)
    jeq     U_Boot_data
    cmp.b   #0b0h,r1h           ; BPS SET(b0h)
    jeq     U_BPS_B0_data
    cmp.b   #0b1h,r1h           ; BPS SET(b1h)
    jeq     U_BPS_B1_data
    cmp.b   #0b2h,r1h           ; BPS SET(b2h)
    jeq     U_BPS_B2_data
    cmp.b   #0b3h,r1h           ; BPS SET(b3h)
    jeq     U_BPS_B3_data
    jmp     U_SIO_send_func

U_Read_check_data:
    mov.w   #0,r3
    mov.w   sum,r1
U_Read_check_data_loop:
    mov.b   r1l,send_d
    jsr     U_SIO_send
    mov.b   r1h,r1l
    add.w   #1,r3
    cmp.w   #2,r3
    jltu    U_Read_check_data_loop
U_Read_check_data_end:
    mov.w   #0,sum              ; reset
```

```
        jsr     U_SIO_exit
        jmp     U_SIO_send_data_end

    U_Read_data:
        mov.w   #0,r3
    U_Read_data_loop:
        lde.b   [a1a0],r1l          ; Flash memory read
        mov.b   r1l,send_d
        jsr     U_SIO_send
        add.w   #1,r3
        add.w   #1,a0
        cmp.w   #256,r3             ; r3 = 256 ?
        jne     U_Read_data_loop
    U_Read_data_end:
        jsr     U_SIO_exit
        jmp     U_SIO_send_data_end

    U_Ver_output_data:
        mov.w   #0,a0               ; Version address offset (a0=0)
    U_Ver_output_data_loop:
        mov.b   ver[a0],send_d      ; send_data set
        jsr     U_SIO_send
        add.w   #1,a0
        cmp.w   #8,a0               ; a0 = 8 ?
        jne     U_Ver_output_data_loop
    U_Ver_output_data_end:
        jsr     U_SIO_exit
        jmp     U_SIO_send_data_end

    U_Read_SRD_data:
        mov.w   #0,r3
    U_Read_SRD_data_loop:
        mov.b   r1l,send_d          ; data transfer
        jsr     U_SIO_send
        mov.b   SRD1,r1l            ; SRD1 data --> r1l
        add.w   #1,r3
        cmp.w   #2,r3               ; r3 = 2 ?
        jltu    U_Read_SRD_data_loop; jump Read_SRD_loop at r3<2
    U_Read_SRD_data_end:
        jsr     U_SIO_exit
        jmp     U_SIO_send_data_end

    U_Read_LB_data:
        mov.b   r1l,send_d          ; data transfer
        jsr     U_SIO_send
    U_Read_LB_data_end:
        jsr     U_SIO_exit
        jmp     U_SIO_send_data_end

    U_Boot_data:
        bclr    fmcr5
        mov.w   addr_l,a0
        mov.b   addr_h,a1
        mov.w   #0,r3
    U_Boot_data_loop:
        lde.b   [a1a0],r1l          ; Flash memory read
        mov.b   r1l,send_d
        jsr     U_SIO_send
        add.w   #1,r3
        add.w   #1,a0
        cmp.w   #256,r3             ; r3 = 256 ?
        jne     U_Boot_data_loop
    U_Boot_data_end:
        bset    fmcr5
        jsr     U_SIO_exit
        jmp     U_SIO_send_data_end

    U_BPS_B0_data:
        mov.b   buff,data_BPS           ; Baud rate 9600bps
        jmp     U_BPS_SET_data
    U_BPS_B1_data:
        mov.b   buff+1,data_BPS         ; Baud rate 19200bps
        jmp     U_BPS_SET_data
    U_BPS_B2_data:
        mov.b   buff+2,data_BPS         ; Baud rate 38400bps
        jmp     U_BPS_SET_data
    U_BPS_B3_data:
        mov.b   buff+3,data_BPS         ; Baud rate 57600bps
    U_BPS_SET_data:
        mov.b   r0l,send_d
        jsr     U_SIO_send
        jsr     U_SIO_exit
        jsr     U_blank_end         ; UART mode Initialize
        jmp     U_SIO_send_data_end
    ;
    U_SIO_send_func:
        mov.w   start_cnt,r3
    U_SIO_send_data_loop:
        mov.w   r3,a0
        mov.b   data[a0],r1l
        mov.b   r1l,send_d
        jsr     U_SIO_send
        add.w   #1,r3
        cmp.w   send_cnt,r3         ; r3 = send_cnt ?
        jne     U_SIO_send_data_loop
        mov.w   r3,r0
    U_SIO_send_data_end:
        bclr    send_flg
        jmp     U_Flash_int
    ;
    ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    ;+     jump table for Flash_func                           +
    ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    U_jmp_tbl:
        .word   U_Read - U_cmd_jmp
        .word   U_Program - U_cmd_jmp
        .word   U_Erase - U_cmd_jmp
        .word   U_All_erase - U_cmd_jmp
        .word   U_Clear_SRD - U_cmd_jmp
        .word   U_Read_LB - U_cmd_jmp
        .word   U_Program_LB - U_cmd_jmp
        .word   U_LB_enable - U_cmd_jmp
        .word   U_LB_disable - U_cmd_jmp
        .word   U_Download - U_cmd_jmp
        .word   U_Boot_output - U_cmd_jmp
        .word   U_Read_check - U_cmd_jmp
    ;
    ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    ;+     jump table for SIO_rcv_first_data                   +
    ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    U_jmp_tbl_2:
        .word   U_SIO_3 - U_SIO_cmd_jmp_2_1         ; Read
        .word   U_SIO_259 - U_SIO_cmd_jmp_2_1       ; Program
        .word   U_SIO_4 - U_SIO_cmd_jmp_2_1         ; erase
        .word   U_SIO_2 - U_SIO_cmd_jmp_2_1         ; All erase
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ; Clear SRD
        .word   U_SIO_3 - U_SIO_cmd_jmp_2_1         ; Read LB
        .word   U_SIO_4 - U_SIO_cmd_jmp_2_1         ; LB Program
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ; LB enable
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ; LB disable
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ; Download
        .word   U_SIO_3 - U_SIO_cmd_jmp_2_1         ; Boot output
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ; Read check
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ; Read SRD
        .word   U_SIO_rcv_ID_check - U_SIO_cmd_jmp_2_1 ; ID check
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ;  Version out
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ;  U_BPS_B0
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ;  U_BPS_B1
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ;  U_BPS_B2
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ;  U_BPS_B3
    ;
    ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    ;+     serch table for Flash_func,SIO_rcv_first_data      +
    ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    U_Index_tbl:
        .byte   0ffh                ; Read(ffh)
        .byte   041h                ; Program(41h)
        .byte   020h                ; Erase(20h)
        .byte   0a7h                ; All_erase(a7h)
        .byte   050h                ; Clear SRD(50h)
        .byte   071h                ; Read LBS(71h)
        .byte   077h                ; LB program(77h)
        .byte   07ah                ; LB enable (7ah)
        .byte   075h                ; LB disable(75h)
        .byte   0fah                ; Download (fah)
        .byte   0fch                ; Boot output(fch)
        .byte   0fdh                ; Read check(fdh)
        .byte   070h                ; Read SRD(70h)
        .byte   0f5h                ; ID check(f5h)
        .byte   0fbh                ; Version output(fbh)
        .byte   0b0h                ; BPS_SET 9600 (b0h)
        .byte   0b1h                ; BPS_SET 19200 (b1h)
        .byte   0b2h                ; BPS_SET 38400(b2h)
        .byte   0b3h                ; BPS_SET 57600(b3h)
    ;
    ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    ;+     Subroutine : Initialize_3 - UART mode              +
    ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    Initialize_3:
    ;-------------------------------------
    ;+     Flash mode set                +
    ;-------------------------------------
    ;
        bset    fmcr5               ; User ROM select
        bclr    fmcr1               ; Flash entry bit clear
        bset    fmcr1               ; Flash entry bit set (E/W mode)
    ;
    ;-------------------------------------
    ;+     Blank check                   +
    ;-------------------------------------
        lde.w   0ffffch,r0          ; Reset vector read
        lde.w   0ffffeh,r1          ; Reset vector read
        and.w   r1,r0               ; r0 & r1
        cmp.w   #0ffffh,r0          ; r0=ffffh ?
        jne     U_blank_end
        bset    sr10                ; check complete at r0=ffffh
        bset    sr11
        bset    blank               ; blank flag set
    U_blank_end:
    ;-------------------------------------
    ;+      UART1                        +
    ;-------------------------------------
    ;----- UART nit rate generator
    ;
```

```
    mov.w   data_BPS,u1brg
;
Initialize_31:
;
;----- 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
;
;---------------------------------------
;+     FLASH function main            +
;---------------------------------------
U_Flash_func:
    mov.b   cmd_d,r0l           ; receive data --> r0l

    mov.b   #0ch,r0h            ; #00001100b sr10,11 mask data
    and.b   SRD1,r0h            ; sr10,11 pick up
    cmp.b   #0ch,r0h            ; ID check OK?
    jne     U_Command_check_2   ; jump Command_check_2 at ID unchecked
    mov.w   #12,a0

U_Command_check:
    lde.b   U_Index_tbl-Trans_TOP2+Ram_progTOP-1[a0],r0h
    cmp.b   r0h,r0l
    jeq     U_cmd_jmp_1
    sbjnz.w  #1,a0,U_Command_check
    jmp     U_Command_check_2

U_cmd_jmp_1:
    shl.w   #1,a0
    lde.w   U_jmp_tbl-Trans_TOP2+Ram_progTOP-2[a0],r0
U_cmd_jmp:
    jmpi.w  r0

U_Command_check_2:
?:  cmp.b   #070h,r0l           ; Read SRD  (70h)
    jne     ?+
    jmp     U_Read_SRD
?:  cmp.b   #0f5h,r0l           ; ID check  (f5h)
    jne     ?+
    jmp     U_ID_check
?:  cmp.b   #0b0h,r0l           ; BPS_SET 9600  (b0h)
    jne     ?+
    jmp     U_BPS_B0
?:  cmp.b   #0b1h,r0l           ; BPS_SET 19200  (b1h)
    jne     ?+
    jmp     U_BPS_B1
?:  cmp.b   #0b2h,r0l           ; BPS_SET 38400  (b2h)
    jne     ?+
    jmp     U_BPS_B2
?:  cmp.b   #0b3h,r0l           ; BPS_SET 57600  (b3h)
    jne     ?+
    jmp     U_BPS_B3
?:  cmp.b   #0fbh,r0l           ; Version out   (fbh)
    jne     U_Flash_func_end
    jmp     U_Ver_output
;
U_Flash_func_end:
    jmp     U_Flash_send
;
;---------------------------------------
;+     Read                            +
;---------------------------------------
U_Read:
    mov.w   #0,r3               ; receive number
    mov.b   #0,addr_l           ; addr_l = 0
U_Read_loop:
    add.w   #1,r3               ; r3 +1 increment
    mov.w   r3,a0               ; r3 --> a0
    mov.w   data[a0],r0
    mov.b   r0l,addr_l[a0]      ; Store address
    cmp.w   #2,r3               ; r3 = 2 ?
    jltu    U_Read_loop         ; jump Read_loop at r3<2
    mov.w   #00ffh,r2           ; Read array command
    jsr     U_Command_write     ; command_write
    bset    send_flg
    mov.w   #3,start_cnt
    mov.w   #258,send_cnt
    jmp     U_Flash_func_end    ; jump  Flash_func_end
;
;---------------------------------------
;+     Program                         +
;---------------------------------------
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_1:
    add.w   #1,r3               ; r3 +1 increment
    mov.w   r3,a0               ; r3 --> a0
    mov.b   data[a0],r0l
    mov.b   r0l,addr_l[a0]      ; Store address
    cmp.w   #259,r3             ; r3 = 259 ?
    jltu    U_Program_loop_1    ; jump Program_loop_1 at r3<258
;
    mov.w   #00ffh,r2           ; Read array command
    jsr     U_Command_write     ; command_write
    mov.w   #0070h,r2           ; Read SRD command
    jsr     U_Command_write     ; Command write
    lde.w   [a1a0],r1           ; SRD read
    mov.w   #00ffh,r2           ; Read array command
    jsr     U_Command_write     ; command_write
    cmp.b   #80h,r1l            ; error check
    jne     U_Program_end
;
    mov.w   #0041h,r2           ; Page program command
    jsr     U_Command_write     ; command_write
    mov.w   #0,r3               ; writing number (r3=0)
    mov.b   addr_h,a1           ; addr_h   --> a1
U_Program_loop_2:
    mov.w   r3,a0               ; r3       --> a0
    mov.w   data[a0],r1         ; data     --> r1
    mov.w   addr_l,a0           ; addr_l,m --> a0
    ste.w   r1,[a1a0]           ; 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 Program_loop_2 at r3<255
U_Program_end:
    mov.w   crcd,sum            ; for Read check command
    bclr    send_flg
    mov.w   #0,send_cnt
    mov.w   #0,start_cnt
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-------------------------------------
;+     Block erase                   +
;-------------------------------------
U_Erase:
    mov.w   #1,r3               ; receive number (r3=1)
    mov.b   #0feh,addr_l        ; addr_l = ffh
U_Erase_loop:
    mov.w   r3,a0               ; r3 --> a0
    mov.b   data[a0],r0l
    mov.b   r0l,addr_l[a0]      ; Store address
    add.w   #1,r3               ; r3 +1 increment
    cmp.w   #4,r3               ; r3=4 ?
    jltu    U_Erase_loop        ; jump Erase_loop at r3<4
    cmp.b   #0d0h,data          ; Confirm command check
    jne     U_Erase_end         ; jump Erase_end at Confirm command error
    mov.w   #0020h,r2           ; Erase command
    jsr     U_Command_write     ; command write
    mov.w   #00d0h,r2           ; Confirm command
    ste.w   r2,[a1a0]           ; command write
U_Erase_end:
    mov.w   #00ffh,r2           ; Read array command
    jsr     U_Command_write     ; command_write
    bclr    send_flg
    mov.w   #0,send_cnt
    mov.w   #0,start_cnt
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-------------------------------------
```

```
;+      All erase ( unlock block )    +
;-------------------------------------
U_All_erase:
    mov.w   #1,a0
    mov.b   data[a0],r0l        ; receive data read --> r0
    cmp.b   #0d0h,r0l           ; Confirm command check
    jne     U_All_erase_end     ; jump All_erase_end at Confirm command error
    mov.w   #0000h,addr_l       ; 0f0000h --> addr
    mov.b   #000fh,addr_h
    mov.w   #00a7h,r2           ; All erase command
    jsr     U_Command_write     ; command write
    mov.w   #00d0h,r2           ; Confirm command
    ste.w   r2,[a1a0]           ; command write
U_All_erase_end:
    mov.w   #00ffh,r2           ; Read array command
    jsr     U_Command_write     ; command_write
    bclr    send_flg
    mov.w   #0,send_cnt
    mov.w   #0,start_cnt
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-------------------------------------
;+     Read SRD                       +
;-------------------------------------
U_Read_SRD:
    mov.w   #0,r3               ; receive number (r3=0)
    mov.w   #0000h,addr_l       ; 0f0000h --> addr
    mov.b   #000fh,addr_h
    mov.w   #00ffh,r2           ; Read array command
    jsr     U_Command_write     ; command_write
    mov.w   #0070h,r2           ; Read SRD command
    jsr     U_Command_write     ; command write
    lde.w   [a1a0],r1           ; SRD read
    mov.w   #00ffh,r2           ; Read array command
    jsr     U_Command_write     ; command_write
    mov.w   #1,start_cnt
    mov.w   #3,send_cnt
    bset    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-------------------------------------
;+     Clear SRD                      +
;-------------------------------------
U_Clear_SRD:
    mov.w   #0000h,addr_l       ; 0f0000h --> addr
    mov.b   #000fh,addr_h
    mov.w   #00ffh,r2           ; Read array command
    jsr     U_Command_write     ; command write
    mov.w   #0050h,r2           ; Clear SRD command
    jsr     U_Command_write     ; command write
    mov.w   #00ffh,r2           ; Read array command
    jsr     U_Command_write     ; command write
    and.b   #10011100b,SRD1     ; SRD1 clear
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-------------------------------------
;+     Read Lock Bit                  +
;-------------------------------------
U_Read_LB:
    mov.w   #1,r3               ; receive number (r3=1)
    mov.b   #0feh,addr_l        ; addr_l = ffh
```

```
U_Read_LB_loop:
    mov.w   r3,a0               ; r3 --> a0
    mov.b   data[a0],r0l
    mov.b   r0l,addr_l[a0]      ; Store address
    add.w   #1,r3               ; r3 +1 increment
    cmp.w   #3,r3               ; r3=3 ?
    jltu    U_Read_LB_loop      ; jump Read_LB_loop at r3<3
    mov.w   #0071h,r2           ; Read LB command
    jsr     U_Command_write     ; command write
    lde.w   [a1a0],r1           ; read LB
    mov.w   #00ffh,r2           ; Read array command
    jsr     U_Command_write     ; command write
U_Read_LB_end:
    mov.w   #1,start_cnt
    mov.w   #1,send_cnt
    bset    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-------------------------------------
;+     Program Lock Bit               +
;-------------------------------------
U_Program_LB:
    mov.w   #1,r3               ; receive number (r3=1)
    mov.b   #0feh,addr_l        ; addr_l = ffh
U_Program_LB_loop:
    mov.w   r3,a0               ; r3 --> a0
    mov.b   data[a0],r0l
    mov.b   r0l,addr_l[a0]      ; Store address
    add.w   #1,r3               ; r3 +1 increment
    cmp.w   #4,r3               ; r3=4 ?
    jltu    U_Program_LB_loop   ; jump   Program_LB_loop at r3<4
    cmp.b   #0d0h,data          ; Confirm command check
    jne     U_Program_LB_end    ; jump Program_LB_end at Confirm command error
    mov.w   #0077h,r2           ; Program LB command
    jsr     U_Command_write     ; command write
    mov.w   #00d0h,r2           ; Confirm command
    ste.w   r2,[a1a0]           ; command write
    mov.w   #00ffh,r2           ; Read array command
    jsr     U_Command_write     ; command write
U_Program_LB_end:
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-------------------------------------
;+     Lock Bit enable                +
;-------------------------------------
U_LB_enable:
    bclr    fmcr2               ; Lock disable bit = 0
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-------------------------------------
;+     Lock Bit disable               +
;-------------------------------------
U_LB_disable:
    bclr    fmcr2               ; Lock disable bit = 0
    bset    fmcr2               ; Lock disable Bit = 1
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-------------------------------------
;+     ID check                      +
;-------------------------------------
U_ID_check:
    btst    blank               ; blank flag check
    jc      U_ID_check_end      ; jump ID_check_end at blank
    cmp.w   #0ffdfh,addr_l      ; lower ID address check
    jne     U_ID_error          ; jump ID_error at ID address error
    cmp.w   #0070fh,addr_h      ; higher ID address check
    jne     U_ID_error          ; jump ID_error at ID address error
U_ID_data_check:
    mov.w   #0000fh,a1          ; ID higher address --> a1
    mov.w   #0ffdfh,r1          ; ID lower address --> r1
    mov.w   #1,r3               ; check loop number (r3=1)
U_ID_check_loop:
    mov.w   r1,a0               ; r1 --> a0
    lde.b   [a1a0],r0l          ; ID data read from Flash memory
    mov.w   r3,a0               ; r3 --> a0
    cmp.b   r0l,data[a0]        ; compare ID data
    jne     U_ID_error          ; jump ID_error at ID error
    add.w   #4,r1               ; r1 +4 increment (next ID address)
    cmp.w   #0ffe7h,r1          ; r1=0ffefh ?
    jne     ?+                  ; jump ? at not equal
    mov.w   #0ffebh,r1          ; r1=0ffeb at equal
?:
    add.w   #1,r3               ; r3 +1 increment
    cmp.w   #8,r3               ; r3=8 ?
    jltu    U_ID_check_loop     ; jump ID_check_loop at r3<8
U_ID_OK:
    bset    sr10
    bset    sr11                ; ID check OK (sr11=1,sr10=1)
    jmp     U_ID_check_end      ; jump  ID_check_end
U_ID_error:
    bset    sr10
    bclr    sr11                ; ID error (sr11=0,sr10=1)
U_ID_check_end:
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-------------------------------------
;+     Boot output                   +
;-------------------------------------
U_Boot_output:
    bclr    fmcr5               ; Boot ROM select
    mov.w   #0,r3               ; receive number
    mov.b   #0,addr_l           ; addr_l = 0
U_Boot_loop:
    add.w   #1,r3               ; r3 +1 increment
    mov.w   r3,a0               ; r3 --> a0
    mov.w   data[a0],r0
    mov.b   r0l,addr_l[a0]      ; Store address
    cmp.w   #2,r3               ; r3 = 2 ?
    jltu    U_Boot_loop         ; jump Read_loop at r3<2
    bset    send_flg
    mov.w   #3,start_cnt
    mov.w   #258,send_cnt
    jmp     U_Flash_func_end    ; jump  Flash_func_end
;
;-------------------------------------
;+     Read check                    +
;---------------------------------------
U_Read_check:
    mov.w   #0,start_cnt
    mov.w   #2,send_cnt
    bset    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;---------------------------------------
;+     Download                      +
;---------------------------------------
U_Download:
    bclr    fmcr5               ; Boot ROM select
    jmp.a   U_Download_program  ; jump Download_program
;
;---------------------------------------
;+     Version output                +
;---------------------------------------
U_Ver_output:
    mov.w   #0,start_cnt
    mov.w   #8,send_cnt
    bset    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : Command write                         +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_Command_write:
    btst    fmcr0               ; RY/BY status check
    jz      U_Command_write
    mov.w   addr_l,a0           ; addr_l,m --> a0
    mov.b   addr_h,a1           ; addr_h   --> a1
    ste.w   r2,[a1a0]           ; command write
    rts
;
;---------------------------------------
;+     Main Init first - UART mode -  +
;---------------------------------------
U_SIO_init_first:
    bclr    freq_set0           ; freq set flag clear
    bclr    freq_set1
    bclr    freq_set2
    mov.b   #100,data_BPS           ; 9600bps for 16MHz
;
    jsr     Initialize_3        ; UART mode Initialize
    jsr     U_SIO_rcv
;
    mov.w   rcv_d,r0
    cmp.b   #0b0h,r0l
    jeq     U_Freq_Get1
    cmp.b   #0f4h,r0l
    jne     U_Freq_Get3
;------ 10MHz ------------------;
    mov.b   #64,buff            ; 9600bps
    mov.b   #32,buff+1          ; 19200bps
    mov.b   #15,buff+2          ; 38400bps
    mov.b   #10,buff+3          ; 57600bps
    mov.b   #0b0h,r0l
    jmp     U_Freq_Get2
;------ 16MHz ------------------;
U_Freq_Get1:
    mov.b   #103,buff           ; 9600bps
    mov.b   #50,buff+1          ; 19200bps
    mov.b   #25,buff+2          ; 38400bps
    mov.b   #17,buff+3          ; 57600bps
```

```
;
U_Freq_Get2:
    mov.b   #0b0h,r0l
    mov.b   buff,data_BPS
    jsr     U_blank_end         ; UART mode Initialize
    bset    freq_set0           ; "B0h" get flag set
    mov.b   #0b0h,cmd_d
    jmp     U_Flash_set
;
U_Freq_Get3:
    mov.b   #80h,data_BPS
    mov.b   #01000000b,r1l      ; counbter1,2 reset
    mov.b   #10000000b,r1h
    jsr     U_blank_end
    jmp     U_Loop_main
;
;-------------------------------------
;+  SIO Init - UART mode -           +
;-------------------------------------
U_SIO_freq:
    btst    freq_set2           ; freq fixed ?
    jc      U_SIO_rcv_first_data_set ;  jump Command_check_2 at data
    btst    freq_set0
    jz      U_Freq_check        ; jump U_Freq_check
    cmp.b   #00h,r0l            ; "00h" get?
    bmgtu    freq_set2
    jne     U_SIO_rcv_first_data_set    ;  jump U_Freq_check
    bclr    freq_set0
    mov.b   #0ffh,r0l           ; dummy data set
    mov.b   #01000000b,r1l      ; counbter1,2 reset
    mov.b   #10000000b,r1h
    mov.b   #80h,data_BPS
    jmp     U_Freq_check

;-------------------------------------
;+     Baud rate change - UART mode   +
;-------------------------------------
U_BPS_B0:
U_BPS_B1:
U_BPS_B2:
U_BPS_B3:
    mov.w   #0,start_cnt
    mov.w   #1,send_cnt
    bset    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Freq check  - UART mode -                         +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_Freq_check:
    bclr    re_u1c1             ; Reception disabled
    btst    8,r1                ; counter = 8 times
    jc      U_Freq_check_4
;
    btst    freq_set1
    jc      U_Freq_check_1
    cmp.b   #00h,r0l            ; "00h"?
    jeq     U_Freq_check_3
    jmp     U_Freq_check_2
U_Freq_check_1:
    btst    13,r0               ; fer_u1rb
    jz      U_Freq_check_3
U_Freq_check_2:
    or.b    r1h,r1l             ; r1l = counter1 or counter2
```

```
U_Freq_check_3:
    xor.b   data_BPS,r1l            ; Baud = Baud xor r1l
    mov.b   r1l,data_BPS            ; 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           ; Min-Baud get ?
    jc      U_Freq_set_1        ; Yes , finished
    bset    freq_set1
    cmp.b   #00h,r0l            ; "00h"?
    jeq     U_Freq_check_5
    xor.b   data_BPS,r1h
    mov.b   r1h,data_BPS
U_Freq_check_5:
    mov.b   data_BPS,data_BPS+1         ; Min Baud --> data+1
    mov.b   #01000000b,r1l      ; counter reset
    mov.b   #10000000b,r1h
    mov.b   #01111111b,data_BPS     ; Reset
U_Freq_check_6:
    jsr     U_blank_end         ; UART mode Initialize
?:
    btst    p6_6
    jz      ?-
    bset    re_u1c1             ; Reception enabled
    jmp     U_Loop_main
;
U_Freq_set_1:
    btst    13,r0               ; fer_u1rd
    jz      U_Freq_set_2
    xor.b   data_BPS,r1h
    mov.b   r1h,data_BPS
U_Freq_set_2:
    bset    freq_set2
    mov.b   data_BPS+1,r1l
    sub.b   data_BPS,r1l
    shl.b   #-1,r1l
    add.b   data_BPS,r1l
;
    mov.b   r1l,buff            ; 9600bps
    shl.b   #-1,r1l             ; 19200bps
    mov.b   r1l,buff+1
    shl.b   #-1,r1l             ; 38400bps
    mov.b   r1l,buff+2
    mov.b   buff,r0l            ; 57600bps
    mov.b   #0,r0h
    divu.b  #6
    mov.b   r0l,buff+3
    mov.b   buff,data_BPS
    mov.b   #0b0h,r0l           ; "B0h" set
    jsr     U_blank_end         ; UART mode Initialize
    jmp     U_BPS_SET_data
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : serial I/O send    - UART mode      +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_SIO_send:
    bclr    re_u1c1
    bset    te_u1c1
    mov.b   send_d,u1tb         ; transmit buffer register
?:
    btst    ti_u1c1             ; transmit buffer empty?
    jnc     ?-
    rts
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : serial I/O send    - UART mode       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_SIO_send_only:
    mov.b   send_d,u1tb         ; transmit buffer register
?:
    btst    ti_u1c1             ; transmit buffer empty?
    jnc     ?-
    rts

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : serial I/O receive - UART mode       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_SIO_rcv:
    bclr    te_u1c1
    bset    re_u1c1
?:
    btst    ri_u1c1             ; receive complete?
    jnc     ?-
    mov.w   u1rb,rcv_d
    rts
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : serial I/O receive - UART mode       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_SIO_rcv_only:
?:
    btst    ri_u1c1             ; receive complete?
    jnc     ?-
    mov.w   u1rb,rcv_d
    rts
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : serial I/O receive - UART mode       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_SIO_exit:
    btst    txept_u1c0
    jnc     U_SIO_exit
    rts
;
;===========================================================
;+     Vector Table                                         +
;===========================================================
    .section    inter,romdata
    .org        Vector+(5*4)
    .lword      Reset|0ff000000h    ; WDT
;
    .org        Vector+(7*4)
    .lword      Reset|0ff000000h    ; NMI
    .lword      Reset               ; Reset
;
    .end
```

Header

```
;*******************************************************
;*  file name   : definition of M16C/62 Flash          *
;*******************************************************
;------------------------------------------------------
;   BUSY output
;------------------------------------------------------
busy        .btequ  4,03ECh     ; p6_4
busy_d      .btequ  4,03EEh     ; pd6_4
;
;------------------------------------------------------
;   Serial I/O select bit
;------------------------------------------------------
s_mode      .btequ  5,03ECh     ; p6_5
s_mode_d    .btequ  5,03EEh     ; pd6_5
;
;------------------------------------------------------
;    define of symbols
;------------------------------------------------------
Ram_TOP     .equ    000400h
Ram_END     .equ    00fffh
Istack      .equ    003000h
;
Version     .equ    0fe000h
Boot_TOP    .equ    0fe020h
Trans_TOP1  .equ    0fe310h
Trans_END1  .equ    0feb90h
Trans_TOP2  .equ    0fed00h
Trans_END2  .equ    0ff630h
Vector      .equ    0fffdch
;
Download_program    .equ    0fe200h
U_Download_program  .equ    0fe270h
;
SB_base     .equ    000400h
Ram_progTOP .equ    000600h
Ram_progEND .equ    000e00h
;
    .section    memory,data
    .org        Ram_TOP
;
SRD:        .blkb   1
SRD1:       .blkb   1
ver:        .blkb   10
SF:         .blkb   1
unuse:      .blkb   4
addr_l:     .blkb   1
addr_m:     .blkb   1
addr_h:     .blkb   1
data:       .blkb   300
buff:       .blkb   20
ID_err:     .blkb   1
sum:        .blkb   2
;
rcv_d:      .blkb   2
send_d:     .blkb   1
t_flg:      .blkb   1
cmd_d:      .blkb   1
loop_cnt:   .blkw   1
send_cnt:   .blkw   1
start_cnt:  .blkw   1
SRD1_bak:   .blkb   3
data_BPS:   .blkb   2
```

```
;
sr0     .btequ  0,SRD
sr1     .btequ  1,SRD
sr2     .btequ  2,SRD
sr3     .btequ  3,SRD
sr4     .btequ  4,SRD
sr5     .btequ  5,SRD
sr6     .btequ  6,SRD
sr7     .btequ  7,SRD
sr8     .btequ  0,SRD1
sr9     .btequ  1,SRD1
sr10        .btequ  2,SRD1
sr11        .btequ  3,SRD1
sr12        .btequ  4,SRD1
sr13        .btequ  5,SRD1
sr14        .btequ  6,SRD1
sr15        .btequ  7,SRD1
;
ram_check   .btequ  0,SF
blank       .btequ  1,SF
old_mode    .btequ  2,SF
freq_set0   .btequ  3,SF
freq_set1   .btequ  4,SF
freq_set2   .btequ  5,SF
;
tout_flg    .btequ  0,t_flg
dwn_flg     .btequ  1,t_flg
cmd_flg     .btequ  2,t_flg
send_flg    .btequ  3,t_flg
rcv_flg     .btequ  4,t_flg
tint_flg    .btequ  5,t_flg
;
```

3.4 Precautions

This section describes precautions to be observed when controlling the M16C/62's internal flash memory.

When Powering On/Off

When powering on/off, pay attention to the following:

(1) Be careful that noise will not get into the control pins (WE, CE, OE). If a noise pulse is applied to the control pins when turning the power on or off, a program/erase error will occur, which in the worst case may destroy the memory data.

(2) A finite wait time is required before you can start read or program/erase operation after power-on. Specifically, a wait time of 2 µs is required before read or program/erase operation can be started after Vcc reached Vccmin (3.0 V).

Chapter 4

M16C/80 Group

4.1 Outline of Hardware

The M16C/80 group contains DINOR-type flash memory.
This section shows hardware information about the M16C/80 group which we think is necessary to create a boot program.

Internal Flash Memory Outline

Table 4.1.1 shows the outline performance of M30800FC/M30802FC of the M16C/80 group.

Table 4.1.1. Outline Performance of M30800FC and M30802FC

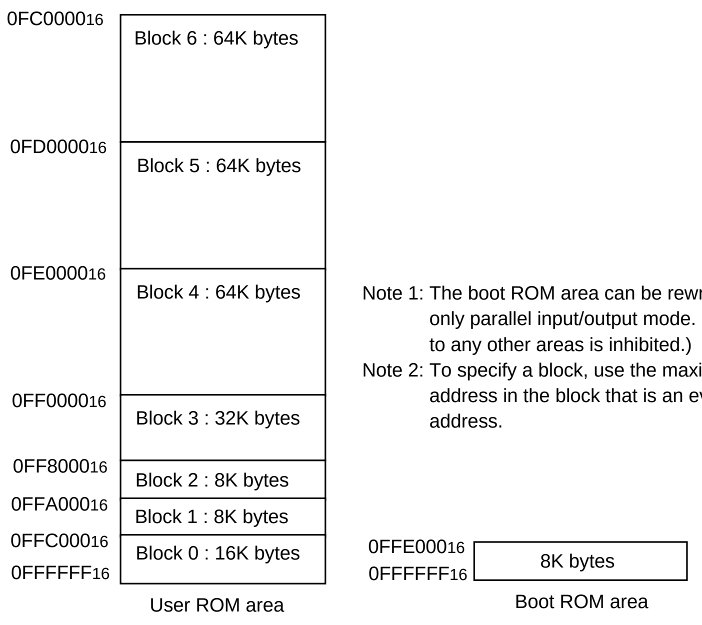
Item		Performance
Power supply voltage		5V version: f(XIN)=20MHz, without wait, 4.2V to 5.5V f(XIN)=10MHz, without wait, 2.7V to 5.5V (under planning)
Program/erase voltage		5V version: 4.2V to 5.5 V f(XIN)=12.5MHz, with one wait f(XIN)=6.25MHz, without wait, 2.7 to 5.5V
Flash memory operation mode		Three modes (parallel I/O, standard serial I/O, CPU rewrite)
Erase block division	User ROM area	See Figure 4.1.1
	Boot ROM area	One division (8 Kbytes) (Note 1)
Program method		In units of pages (in units of 256 bytes)
Erase method		Collective erase/block erase
Program/erase control method		Program/erase control by software command
Protect method		Protected for each block by lock bit
Number of commands		8 commands
Program/erase count		100 times
ROM code protect		Parallel I/O and standard serial modes are supported.

Note: The boot ROM area contains a standard serial I/O mode control program which is stored in it when shipped from the factory. This area can be erased and programmed in only parallel I/O mode.

Memory Map

The user ROM of M30800FC has six blocks as block 0 to block 5 and that of M30803FC has seven blocks as block 0 to block 6. Figure 4.1.1 shows the memory map.

Flash memory size	Flash memory start address
128Kbytes	0FE0000$_{16}$
256Kbytes	0FC0000$_{16}$



Note 1: The boot ROM area can be rewritten in only parallel input/output mode. (Access to any other areas is inhibited.)
Note 2: To specify a block, use the maximum address in the block that is an even address.

Figure 4.1.1 Memory Map

Related Register Configuration

Figure 4.1.2 shows related registers for making user boot program.

Flash memory control register 0

b7	b6	b5	b4	b3	b2	b1	b0
×	×		0				

Symbol	Address	When reset
FMR0	0377_{16}	$XX000001_2$

Bit symbol	Bit name	Function	R	W
FMR00	RY/$\overline{\text{BY}}$ status flag	0: Busy (being written or erased) 1: Ready	○	×
FMR01	CPU rewrite mode select bit (Note 1)	0: Normal mode (Software commands invalid) 1: CPU rewrite mode (Software commands acceptable)	○	○
FMR02	Lock bit disable bit (Note 2)	0: Block lock by lock bit data is enabled 1: Block lock by lock bit data is disabled	○	○
FMR03	Flash memory reset bit (Note 3)	0: Normal operation 1: Reset	○	○
Reserved bit		Must always be set to “0”	○	○
FMR05	User ROM area select bit (Note 4) (Effective in only boot mode)	0: Boot ROM area is accessed 1: User ROM area is accessed	○	○
Nothing is assigned. When write, set "0". When read, values are indeterminate.			—	—

Note 1: For this bit to be set to “1”, the user needs to write a “0” and then a “1” to it in succession. When it is not this procedure, it is not enacted in “1”. This is necessary to ensure that no interrupt or DMA transfer will be executed during the interval. Use the control program except in the internal flash memory for write to this bit.
Note 2: For this bit to be set to “1”, the user needs to write a “0” and then a “1” to it in succession when the CPU rewrite mode select bit = “1”. When it is not this procedure, it is not enacted in “1”. This is necessary to ensure that no interrupt or DMA transfer will be executed during the interval.
Note 3: Effective only when the CPU rewrite mode select bit = 1. Set this bit to 0 subsequently after setting it to 1 (reset).
Note 4: Use the control program except in the internal flash memory for write to this bit.

Flash memory control register 1

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0		0	0	0

Symbol	Address	When reset
FMR1	0376_{16}	$XXXX0XXX_2$

Bit symbol	Bit name	Function	R	W
Reserved bit		Must always be set to “0”	—	○
FMR13	Flash memory power supply-OFF bit (Note)	0: Flash memory power supply is connected 1: Flash memory power supply-off	○	○
Reserved bit		Must always be set to “0”	—	○

Note : For this bit to be set to “1”, the user needs to write a “0” and then a “1” to it in succession. When it is not this procedure, it is not enacted in “1”. This is necessary to ensure that no interrupt or DMA transfer will be executed during the interval. Use the control program except in the internal flash memory for write to this bit.
During parallel I/O mode,programming,erase or read of flash memory is not controlled by this bit,only by external pins.

Figure 4.1.2 Related Register Configuration

Flash Control Circuit

The M16C/80's flash control circuit controls the block erase and page program operations performed on the internal flash memory. Operation modes are selected by entering software commands to the flash control circuit. The status shows the status of the flash control circuit, as well as the status of program and block erase operations performed by the flash control circuit.
To enter commands to the flash control circuit, write the command to flash memory address.

Software commands

Flash memory operations are selected by writing a software command to the flash control circuit. The table below lists the operations performed by software commands.

Table 4.1.2 Software Command List

Command	First bus cycle			Second bus cycle			Third bus cycle		
	Mode	Address	Data (D_0 to D_7)	Mode	Address	Data (D_0 to D_7)	Mode	Address	Data (D_0 to D_7)
Read array	Write	X (Note 6)	FF_{16}						
Read status register	Write	X	70_{16}	Read	X	SRD (Note 2)			
Clear status register	Write	X	50_{16}						
Page program (Note 3)	Write	X	41_{16}	Write	WA0 (Note 3)	WD0 (Note 3)	Write	WA1	WD1
Block erase	Write	X	20_{16}	Write	BA (Note 4)	$D0_{16}$			
Erase all unlock block	Write	X	$A7_{16}$	Write	X	$D0_{16}$			
Lock bit program	Write	X	77_{16}	Write	BA	$D0_{16}$			
Read lock bit status	Write	X	71_{16}	Read	BA	D_6 (Note 5)			

Note 1: When a software command is input, the high-order byte of data (D_8 to D_{15}) is ignored.
Note 2: SRD = Status Register Data
Note 3: WA = Write Address, WD = Write Data
WA and WD must be set sequentially from 00_{16} to FE_{16} (byte address; however, an even address). The page size is 256 bytes.
Note 4: BA = Block Address (Enter the maximum address of each block that is an even address.)
Note 5: D_6 corresponds to the block lock status. Block not locked when D_6 = 1, block locked when D_6 = 0.
Note 6: X denotes a given address in the user ROM area (that is an even address).

Flash memory address

The table below shows the flash memory capacity of each block (address space, number of pages) and the block addresses of each block.

Table 4.1.3 Flash Memory Address

	Size	Page No.	Address	Block address
Block 6	64 Kbytes	256	$FC0000_{16}$–$FCFFFF_{16}$	$FCFFFE_{16}$
Block 5	64 Kbytes	256	$FD0000_{16}$–$FDFFFF_{16}$	$FDFFFE_{16}$
Block 4	64 Kbytes	256	$FE0000_{16}$–$FEFFFF_{16}$	$FEFFFE_{16}$
Block 3	32 Kbytes	128	$FF0000_{16}$–$FF7FFF_{16}$	$FF7FFE_{16}$
Block 2	8 Kbytes	32	$FF8000_{16}$–$FF9FFF_{16}$	$FF9FFE_{16}$
Block 1	8 Kbytes	32	$FFA000_{16}$–$FFBFFF_{16}$	$FFBFFE_{16}$
Block 0	16 Kbytes	64	$FFC000_{16}$–$FFFFFF_{16}$	$FFFFFE_{16}$

Read Array Command (FF_{16})

The read array mode is entered by writing the command code "FF_{16}" in the first bus cycle. When an even address to be read is input in one of the bus cycles that follow, the content of the specified address is read out at the data bus (D0–D15), 16 bits at a time. The read array mode is retained intact until another command is written.

Read Status Register Command (70_{16})

When the command code "70_{16}" is written in the first bus cycle, the content of the status register is read out at the data bus (D0–D7) by a read in the second bus cycle.
The status register is explained in the next section.

Clear Status Register Command (50_{16})

This command is used to clear the bits SR3 to 5 of the status register after they have been set. These bits indicate that operation has ended in an error. To use this command, write the command code "50_{16}" in the first bus cycle.

Page Program Command (41_{16})

Page program allows for high-speed programming in units of 256 bytes. Page program operation starts when the command code "41_{16}" is written in the first bus cycle. In the second bus cycle through the 129th bus cycle, the write data is sequentially written 16 bits at a time. At this time, the addresses A0-A7 need to be increased by 2 from "00_{16}" to "FE_{16}." When the system finishes loading the data, it starts an auto write operation (data program and verify operation).
Whether the auto write operation is completed can be confirmed by reading the status register or the flash memory control register 0. At the same time the auto write operation starts, the read status register mode is automatically entered.
After the auto write operation is completed, the status register can be read out to know the result of the auto write operation. For details, refer to the section where the status register is detailed.
The status register bit 7 (SR7) is set to 0 at the same time the auto write operation starts and is returned to 1 upon completion of the auto write operation. In this case, the read status register mode remains active until the Read Array command (FF_{16}) or Read Lock Bit Status command (71_{16}) is written or the flash memory is reset using its reset bit.
The RY/$\overline{\text{BY}}$ status flag of the flash memory control register 0 is 0 during auto write operation and 1 when the auto write operation is completed as is the status register bit 7.
Figure 4.1.3 shows an example of a page program flowchart.
Each block of the flash memory can be write protected by using a lock bit. For details, refer to the section where the data protect function is detailed.
Additional writes to the already programmed pages are prohibited.

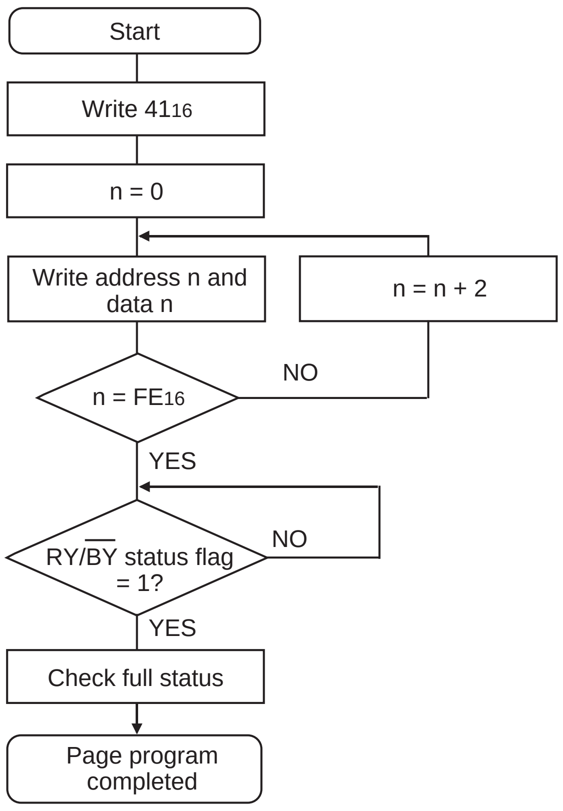


Figure 4.1.3 Page Program Flowchart

Block Erase Command (20_{16}/$D0_{16}$)

By writing the command code “20_{16}” in the first bus cycle and the confirm command code “$D0_{16}$” in the second bus cycle that follows to the block address of a flash memory block, the system initiates an auto erase (erase and erase verify) operation.
Whether the auto erase operation is completed can be confirmed by reading the status register or the flash memory control register 0. At the same time the auto erase operation starts, the read status register mode is automatically entered, so the content of the status register can be read out. The status register bit 7 (SR7) is set to 0 at the same time the auto erase operation starts and is returned to 1 upon completion of the auto erase operation. In this case, the read status register mode remains active until the Read Array command (FF_{16}) or Read Lock Bit Status command (71_{16}) is written or the flash memory is reset using its reset bit.
The RY/$\overline{\text{BY}}$ status flag of the flash memory control register 0 is 0 during auto erase operation and 1 when the auto erase operation is completed as is the status register bit 7.
After the auto erase operation is completed, the status register can be read out to know the result of the auto erase operation. For details, refer to the section where the status register is detailed.
Figure 4.1.4 shows an example of a block erase flowchart.
Each block of the flash memory can be protected against erasure by using a lock bit. For details, refer to the section where the data protect function is detailed.

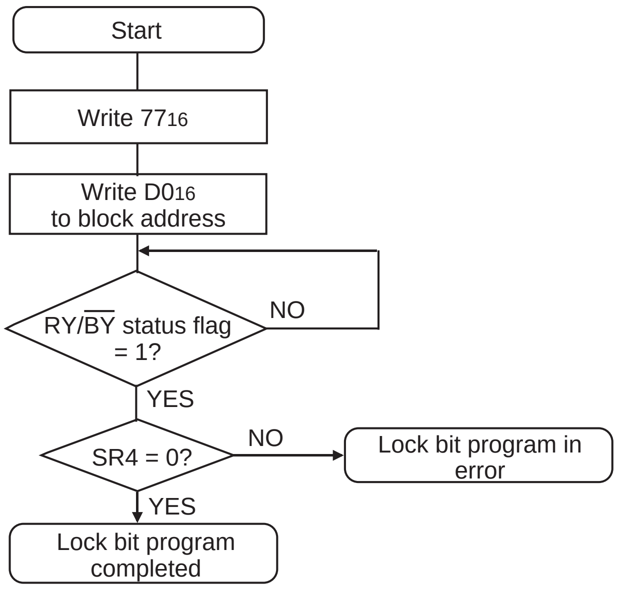


Figure 4.1.4 Block Erase Flowchart

Erase All Unlock Blocks Command ($A7_{16}$/$D0_{16}$)

By writing the command code "$A7_{16}$" in the first bus cycle and the confirm command code "$D0_{16}$" in the second bus cycle that follows, the system starts erasing blocks successively.
Whether the Erase All Unlock Blocks command is terminated can be confirmed by reading the status register or the flash memory control register 0, in the same way as for block erase. Also, the status register can be read out to know the result of the auto erase operation.
When the lock bit disable bit of the flash memory control register 0 = 1, all blocks are erased no matter how the lock bit is set. On the other hand, when the lock bit disable bit = 0, the function of the lock bit is effective and only unlocked blocks (where lock bit data = 1) are erased.

Lock Bit Program Command (77_{16}/$D0_{16}$)

By writing the command code "77_{16}" in the first bus cycle and the confirm command code "$D0_{16}$" in the second bus cycle that follows to the block address of a flash memory block, the system sets the lock bit for the specified block to 0 (locked).
Figure 4.1.5 shows an example of a lock bit program flowchart. The status of the lock bit (lock bit data) can be read out by a Read Lock Bit Status command.
Whether the lock bit program command is terminated can be confirmed by reading the status register or the flash memory control register 0, in the same way as for page program.
For details about the function of the lock bit and how to reset the lock bit, refer to the section where the data protect function is detailed.

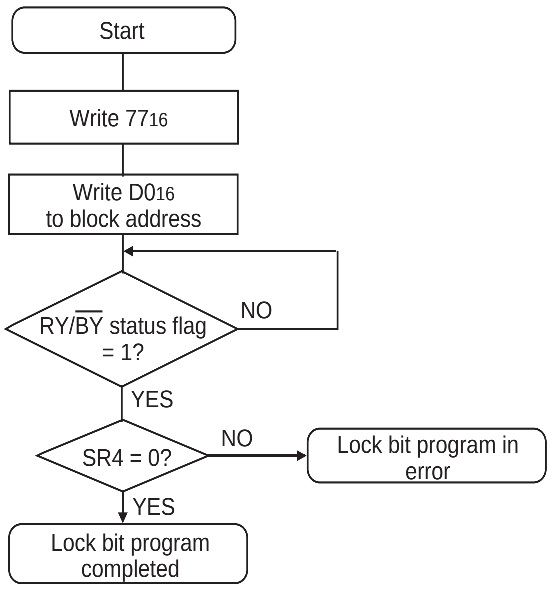
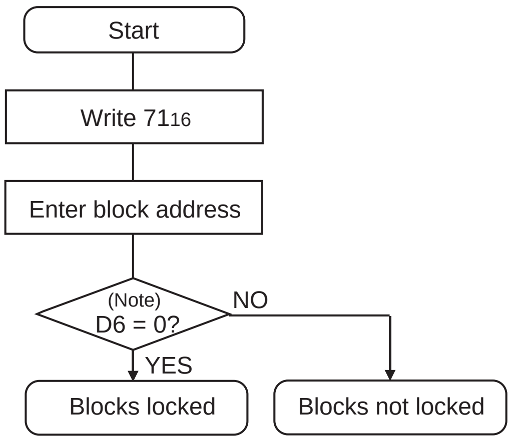


Figure 4.1.5   Lock Bit Program Flowchart

Read Lock Bit Status Command (71_{16})

By writing the command code "71_{16}" in the first bus cycle and then the block address of a flash memory block in the second bus cycle that follows, the system reads out the status of the lock bit of the specified block on to the data (D6).
Figure 4.1.6 shows an example of a read lock bit program flowchart.



Note: Data bus bit 6.

Figure 4.1.6 Read Lock Bit Program Flowchart

Data Protect Function (Block Lock)

Each block in Figure 4.1.1 has a nonvolatile lock bit to specify that the block be protected (locked) against erase/write. The Lock Bit Program command is used to set the lock bit to 0 (locked). The lock bit of each block can be read out using the Read Lock Bit Status command.
Whether block lock is enabled or disabled is determined by the status of the lock bit and how the flash memory control register 0's lock bit disable bit is set.

(1) When the lock bit disable bit = 0, a specified block can be locked or unlocked by the lock bit status (lock bit data). Blocks whose lock bit data = 0 are locked, so they are disabled against erase/write.
On the other hand, the blocks whose lock bit data = 1 are not locked, so they are enabled for erase/write.

(2) When the lock bit disable bit = 1, all blocks are unlocked regardless of the lock bit data, so they are enabled for erase/write. In this case, the lock bit data that is 0 (locked) is set to 1 (unlocked) after erasure, so that the lock bit-actuated lock is removed.

Status Register

The status register indicates the operating status of the flash memory and whether an erase or program operation has terminated normally or in an error. The content of this register can be read out by only writing the read status register command (70_{16}). Table 4.1.3 details the status register.
The status register is cleared by writing the Clear Status Register command (50_{16}).
After a reset, the status register is set to “80_{16}.”
Each bit in this register is explained below.

Write state machine (WSM) status (SR7)

After power-on, the write state machine (WSM) status is set to 1.
The write state machine (WSM) status indicates the operating status of the device, as for output on the RY/$\overline{\text{BY}}$ pin. This status bit is set to 0 during auto write or auto erase operation and is set to 1 upon completion of these operations.

Erase status (SR5)

The erase status informs the operating status of auto erase operation to the CPU. When an erase error occurs, it is set to 1.
The erase status is reset to 0 when cleared.

Program status (SR4)

The program status informs the operating status of auto write operation to the CPU. When a write error occurs, it is set to 1.
The program status is reset to 0 when cleared.
When an erase command is in error (which occurs if the command entered after the block erase command (20_{16}) is not the confirm command ($D0_{16}$), both the program status and erase status (SR5) are set to 1.
When the program status or erase status = 1, the following commands entered by command write are not accepted.
Also, in one of the following cases, both SR4 and SR5 are set to 1 (command sequence error):
(1) When the valid command is not entered correctly
(2) When the data entered in the second bus cycle of lock bit program (77_{16}/$D0_{16}$), block erase (20_{16}/$D0_{16}$), or erase all unlock blocks ($A7_{16}$/$D0_{16}$) is not the $D0_{16}$ or FF_{16}. However, if FF_{16} is entered, read array is assumed and the command that has been set up in the first bus cycle is canceled.

Block status after program (SR3)

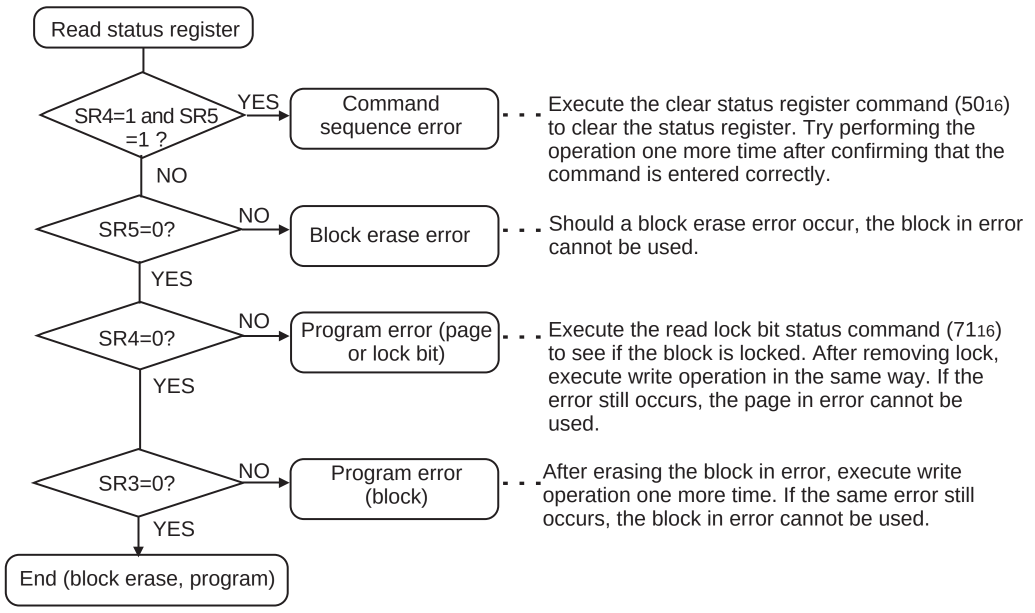
If excessive data is written (phenomenon whereby the memory cell becomes depressed which results in data not being read correctly), “1” is set for the program status after-program at the end of the page write operation. In other words, when writing ends successfully, “80_{16}” is output; when writing fails, “90_{16}” is output; and when excessive data is written, “88_{16}” is output.

Table 4.1.4 Definition of Each Bit in Status Register

Each bit of SRD	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	-	-

Full Status Check

By performing full status check, it is possible to know the execution results of erase and program operations. Figure 4.1.7 shows a full status check flowchart and the action to be taken when each error occurs.


Read status register
SR4=1 and SR5 =1 ?
YES
Command sequence error
Execute the clear status register command (50$_{16}$) to clear the status register. Try performing the operation one more time after confirming that the command is entered correctly.
NO
SR5=0?
NO
Block erase error
Should a block erase error occur, the block in error cannot be used.
YES
SR4=0?
NO
Program error (page or lock bit)
Execute the read lock bit status command (71$_{16}$) to see if the block is locked. After removing lock, execute write operation in the same way. If the error still occurs, the page in error cannot be used.
YES
SR3=0?
NO
Program error (block)
After erasing the block in error, execute write operation one more time. If the same error still occurs, the block in error cannot be used.
YES
End (block erase, program)

Note: When one of SR5 to SR3 is set to 1, none of the page program, block erase, erase all unlock blocks and lock bit program commands is accepted. Execute the clear status register command (50$_{16}$) before executing these commands.

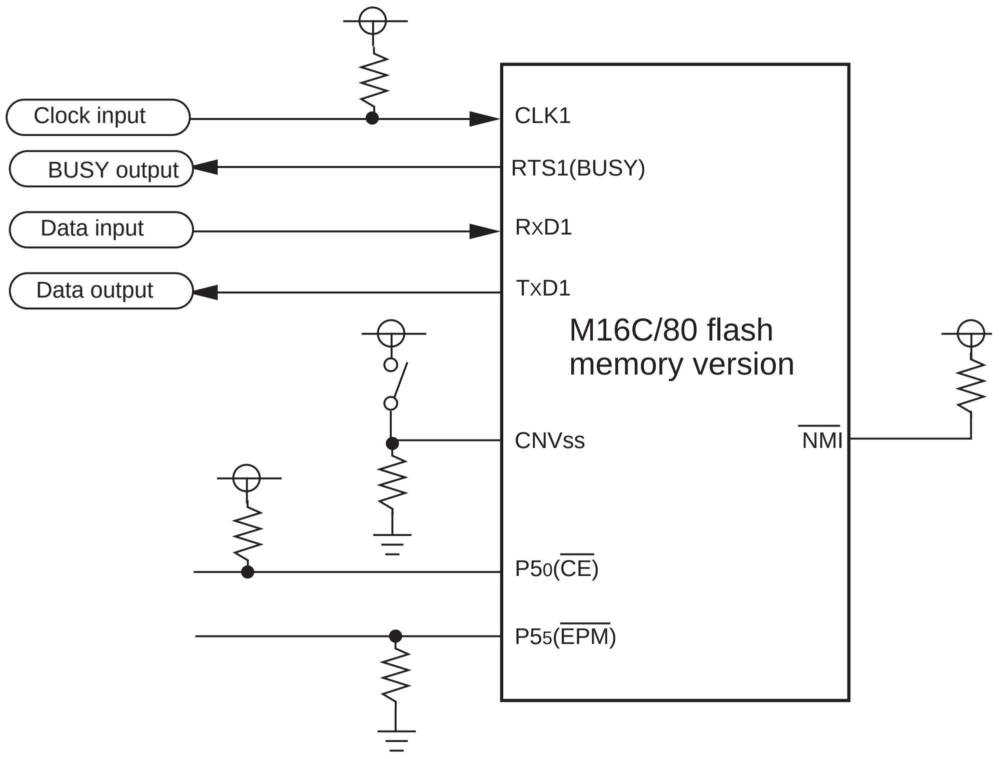
Figure 4.1.7 Full Status Check Flowchart and Remedial Procedure for Errors

4.2 Developing The Boot Program

The standard boot program that was built into the boot ROM area of the flash microcomputer when shipped from the factory can be used to program/erase the flash memory. In this case, the hardware resources (internal functions) used for control are fixed. Therefore, if you want to control flash memory in the way suitable for your system, you need to create a boot program for yourself.
This section shows an algorithm for the boot program (e.g., for erase and program) that you must at least have in order to control the flash memory of the M16C/80 group.

System Example

By using the internal peripheral function of UART0 and a serial programmer to control flash memory, the following shows an example of device connections is shown in Figure 4.2.1. Assignments of internal peripheral functions are listed in Table 4.2.1.



(1) Control pins and external circuitry will vary according to peripheral unit (programmer). For more information, see the peripheral unit (programmer) manual.
(2) In this example, the microprocessor mode and standard serial I/O mode are switched via a switch

Figure 4.2.1 Example of Device Connection

Table 4.2.1 Assignments of Internal Peripheral Functions

Peripheral function	Usage	Setting example
UART1	Used for transfer/receive of serial programmer and data	• Clock synchronous serial I/O • External clock is used
Timer A0	Used for time-over judgment of serial transfer/receive	• One-shot timer mode • 300 µs(when 20MHz)

Flow of The Main Processing

Figure 4.2.2 shows a flow of the main processing.
After initializing the CPU, transfer the write control program to RAM. When transfer is finished, jump to RAM and execute the write control program from RAM.

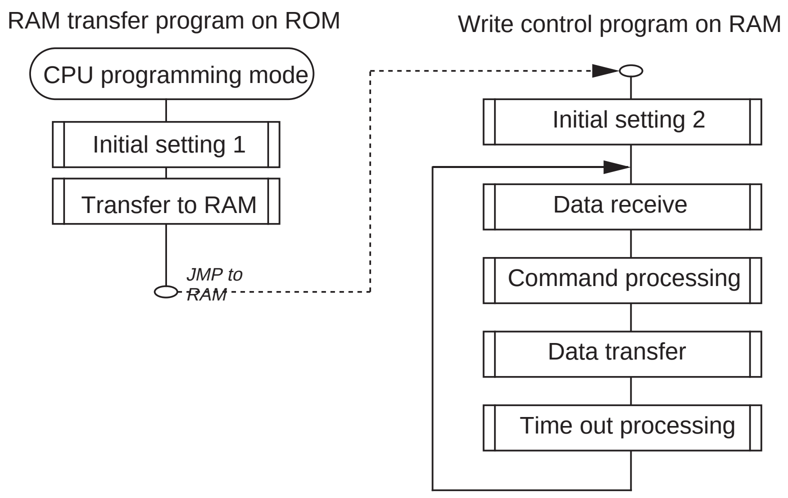


Figure 4.2.2 Flow of The Main Processing

Initialization 1 (CPU, Memory)

The CPU and RAM are initialized. Figure 4.2.3 shows a flow of initialization 1. To clear RAM, use of string instructions (e.g., SSTR.W) will prove effective.

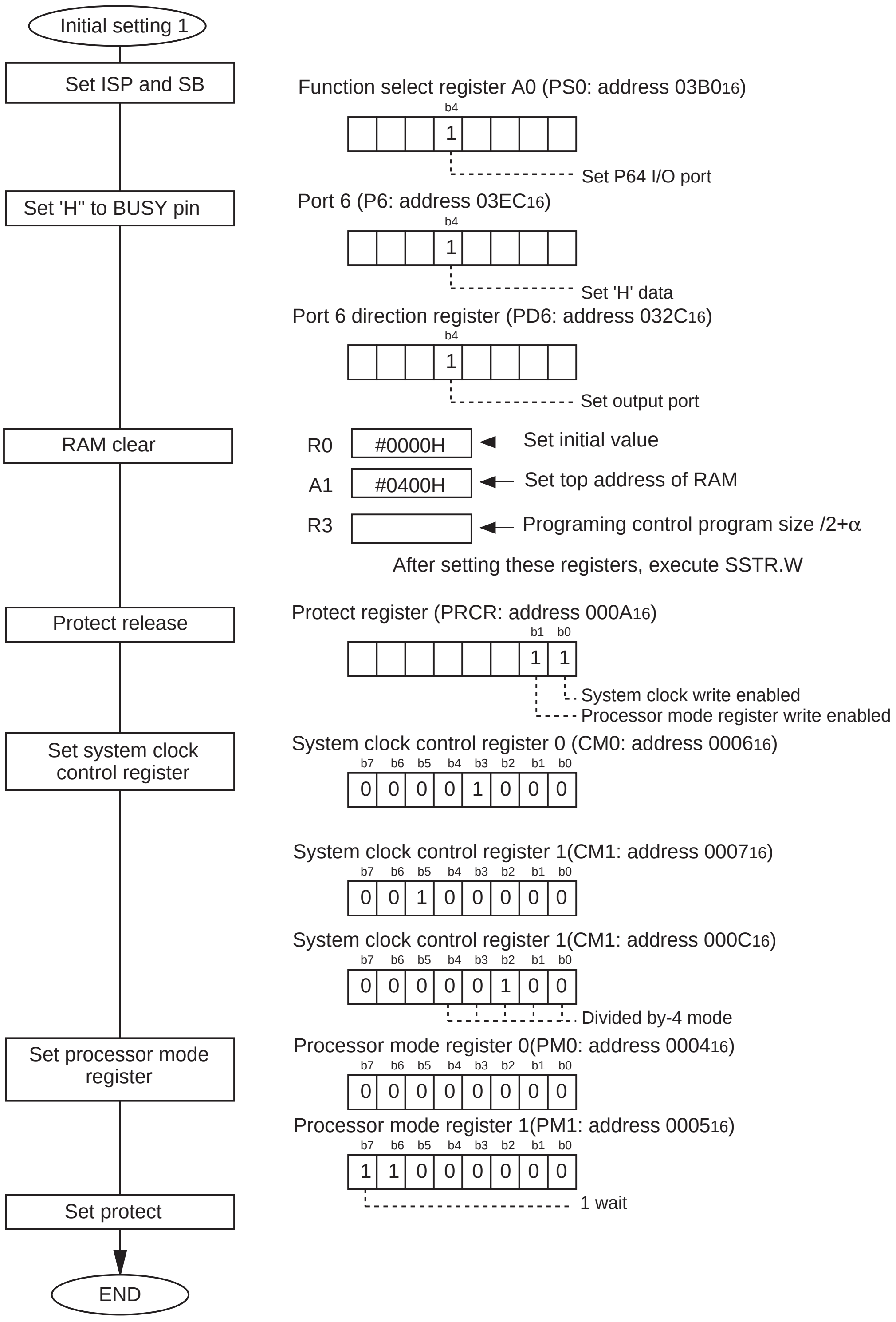

Initial setting 1
Set ISP and SB
Set 'H" to BUSY pin
RAM clear
Protect release
Set system clock control register
Set processor mode register
Set protect
END
Function select register A0 (PS0: address 03B016)
b4
1
Set P64 I/O port
Port 6 (P6: address 03EC16)
Set 'H' data
Port 6 direction register (PD6: address 032C16)
Set output port
R0 #0000H Set initial value
A1 #0400H Set top address of RAM
R3 Programing control program size /2+α
After setting these registers, execute SSTR.W
Protect register (PRCR: address 000A16)
b1 b0 1 1
System clock write enabled
Processor mode register write enabled
System clock control register 0 (CM0: address 000616)
b7 b6 b5 b4 b3 b2 b1 b0
0 0 0 0 1 0 0 0
System clock control register 1(CM1: address 000716)
0 0 1 0 0 0 0 0
System clock control register 1(CM1: address 000C16)
0 0 0 0 0 1 0 0
Divided by-4 mode
Processor mode register 0(PM0: address 000416)
0 0 0 0 0 0 0 0
Processor mode register 1(PM1: address 000516)
1 1 0 0 0 0 0 0
1 wait

Figure 4.2.3 Initialization 1

Transfer to RAM Area

The version information of write program and write control program are transferred to RAM. After transferring, jump to write control program on RAM. To transfer, use of string instructions will prove effective.
Figure 4.2.4 shows the algorithm.

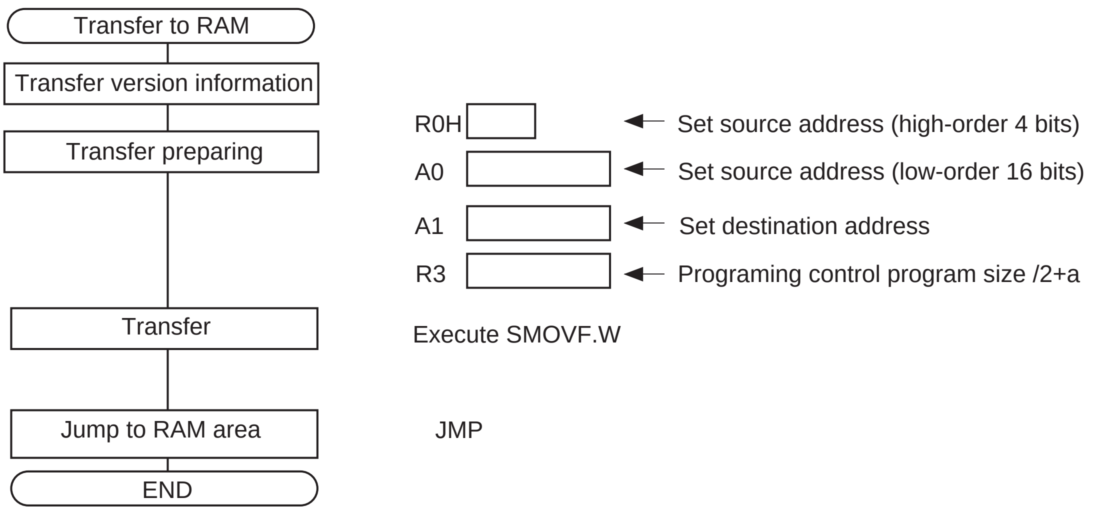

Transfer to RAM
Transfer version information
Transfer preparing
Transfer
Jump to RAM area
END
R0H
A0
A1
R3
Set source address (high-order 4 bits)
Set source address (low-order 16 bits)
Set destination address
Programing control program size /2+a
Execute SMOVF.W
JMP

Figure 4.2.4 Transfer to RAM Area

Initialization 2

Set of write to Flash memory and initialization of serial communication are executed. To switch erase/write mode, clear the flash entry bit (bit 1 of address 377_{16}), then set 1.
Figure 4.2.5 shows a algorithm.

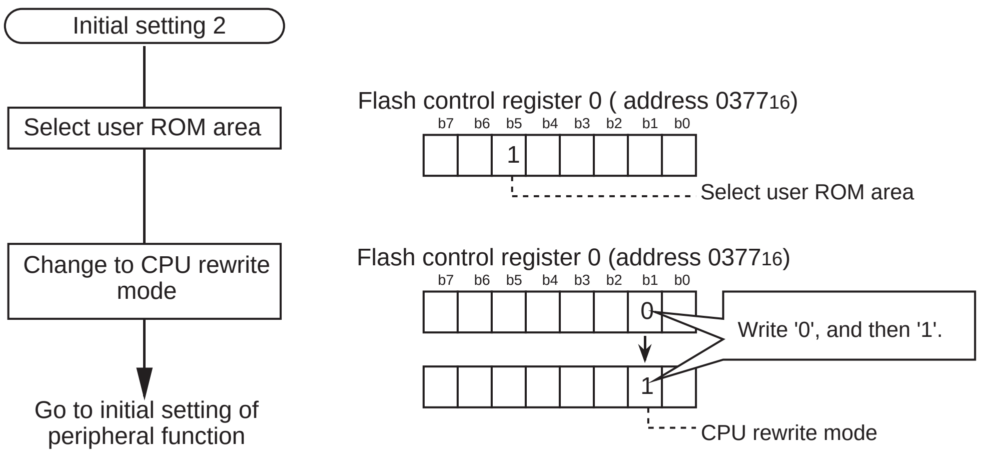

Initial setting 2
Select user ROM area
Change to CPU rewrite mode
Go to initial setting of peripheral function
Flash control register 0 (address 0377₁₆)
b7 b6 b5 b4 b3 b2 b1 b0
1
Select user ROM area
Flash control register 0 (address 0377₁₆)
b7 b6 b5 b4 b3 b2 b1 b0
0
1
Write '0', and then '1'.
CPU rewrite mode

Figure 4.2.5 Initialization 2

Initialization 2 (Peripheral Function)

The peripheral functions used for programming flash memory is initialized. Figure 4.2.6 shows initialization of UART1 for data transmit and timer A0 for time-out calculation.

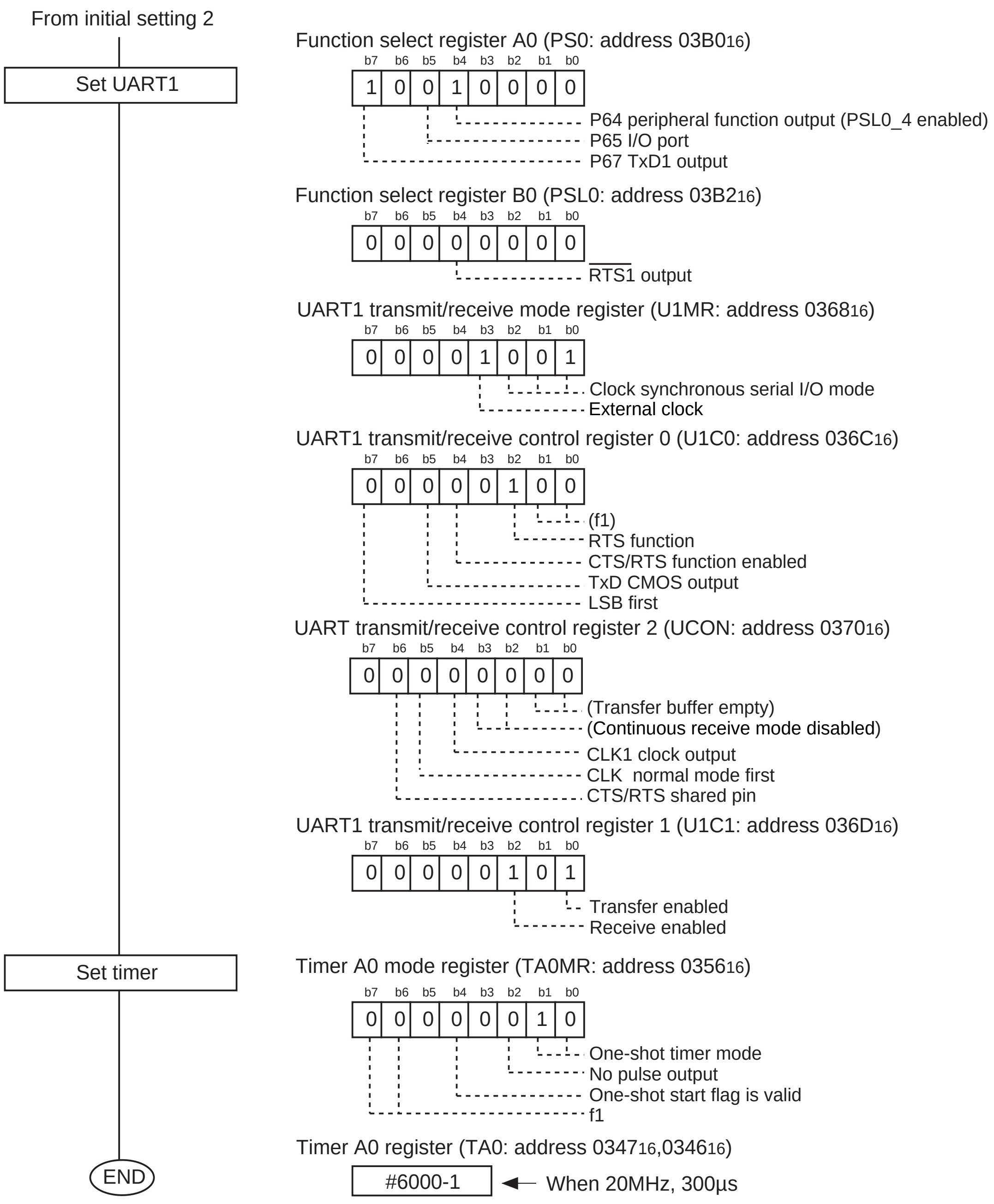

From initial setting 2
Set UART1
Function select register A0 (PS0: address 03B016)
b7 b6 b5 b4 b3 b2 b1 b0
1 0 0 1 0 0 0 0
P64 peripheral function output (PSL0_4 enabled)
P65 I/O port
P67 TxD1 output
Function select register B0 (PSL0: address 03B216)
0 0 0 0 0 0 0 0
RTS1 output
UART1 transmit/receive mode register (U1MR: address 036816)
0 0 0 0 1 0 0 1
Clock synchronous serial I/O mode
External clock
UART1 transmit/receive control register 0 (U1C0: address 036C16)
0 0 0 0 0 1 0 0
(f1)
RTS function
CTS/RTS function enabled
TxD CMOS output
LSB first
UART transmit/receive control register 2 (UCON: address 037016)
0 0 0 0 0 0 0 0
(Transfer buffer empty)
(Continuous receive mode disabled)
CLK1 clock output
CLK normal mode first
CTS/RTS shared pin
UART1 transmit/receive control register 1 (U1C1: address 036D16)
0 0 0 0 0 1 0 1
Transfer enabled
Receive enabled
Set timer
Timer A0 mode register (TA0MR: address 035616)
0 0 0 0 0 0 1 0
One-shot timer mode
No pulse output
One-shot start flag is valid
f1
Timer A0 register (TA0: address 034716,034616)
#6000-1
When 20MHz, 300µs
END

Figure 4.2.6 Initialization 2 (Peripheral Function)

Receiving Commands

Commands are received from the serial programmer.
Write dummy data to the transmit buffer, enable reception (the BUSY signal = low), and wait for data from the serial programmer. At the timing of start reception (the BUSY signal = high), the timer used to check data reception time-out is started. When data is not received within 300 µsec, a time-out error is judged and time-out processing flag is set.
When command reception flag is set (cmd_flg = "1"), processing jumps to data reception cycle number check processing. When it is not set (cmd_flg = "0"), command reception flag is set. After that, jump address is set based on the received serial command and processing jumps to the corresponding process. When the serial command is not matched, serial initialization flag is set and processing is ended.
When the number of receive cycle matches to the prescribed number of serial reception command, command reception flag is initialized (cmd_flg = "0") and processing is ended.
Figure 4.2.7 shows a processing flow.

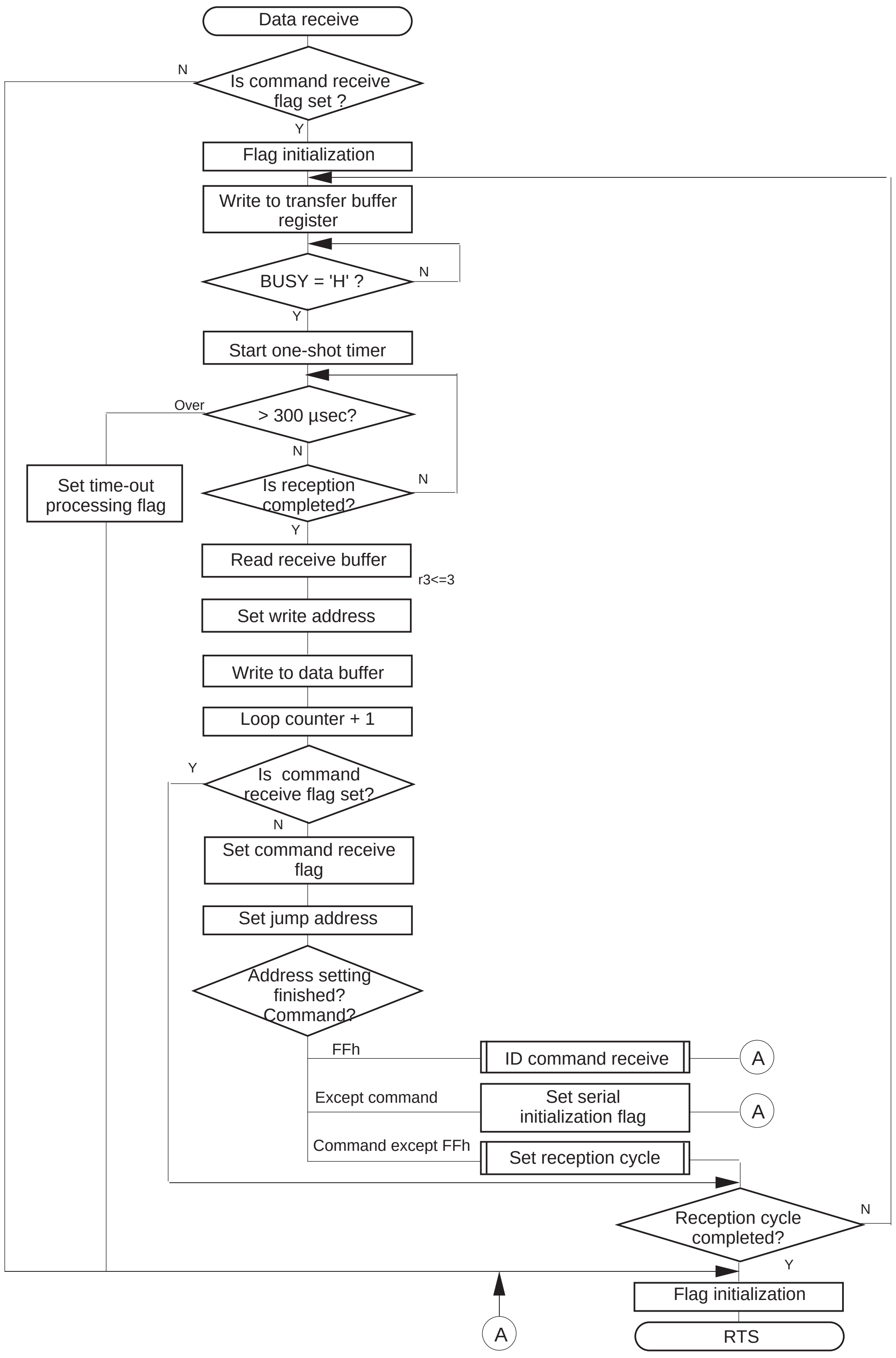


Figure 4.2.7 Data Reception

ID Check Receive Process

ID check data is received. Transferred ID data is saved to RAM.
Figure 4.2.8 shows a process flow.

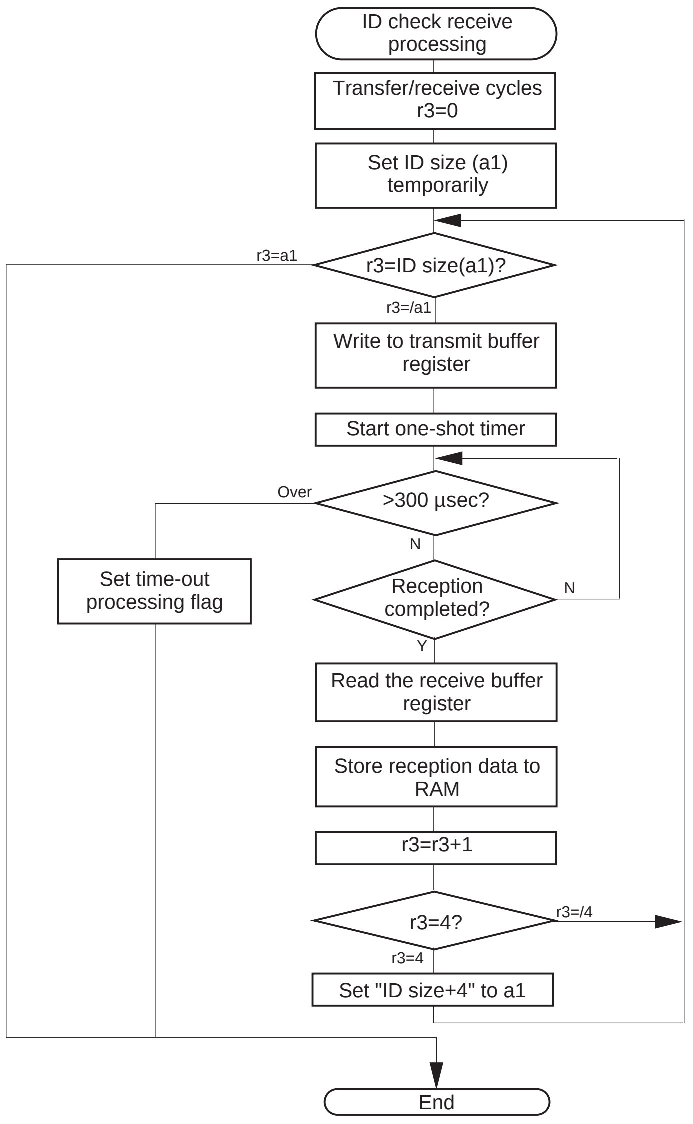

ID check receive processing
Transfer/receive cycles r3=0
Set ID size (a1) temporarily
r3=a1
r3=ID size(a1)?
r3=/a1
Write to transmit buffer register
Start one-shot timer
Over
>300 µsec?
N
Set time-out processing flag
Reception completed?
N
Y
Read the receive buffer register
Store reception data to RAM
r3=r3+1
r3=4?
r3=/4
r3=4
Set "ID size+4" to a1
End

Figure 4.2.8 ID Data Receive Process

Receive Cycle Setting Process

Data receive cycle is set by referring to transferred serial command.
Figure 4.2.9 shows process flow.

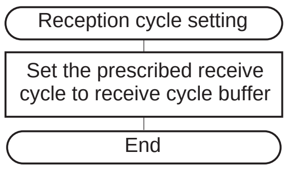


Figure 4.2.9   Receive Cycle Setting Process

Command Processing

Flash control command is written into memory by referring to received serial programmer command.
The ID check is checked as to whether it has been completed or not. (ID check completed bits: SR10 = 1, SR11 = 1) When the ID check has been completed, decisions are made on commands such as page read and page program, and processing branches to the process in the match commands.
When the ID check has not been completed, decisions are made on 3 types of commands such as ID processing, and processing jumps to the process in the match commands. With mismatch commands, processing returns to main part.
Figure 4.2.10 shows processing flow.

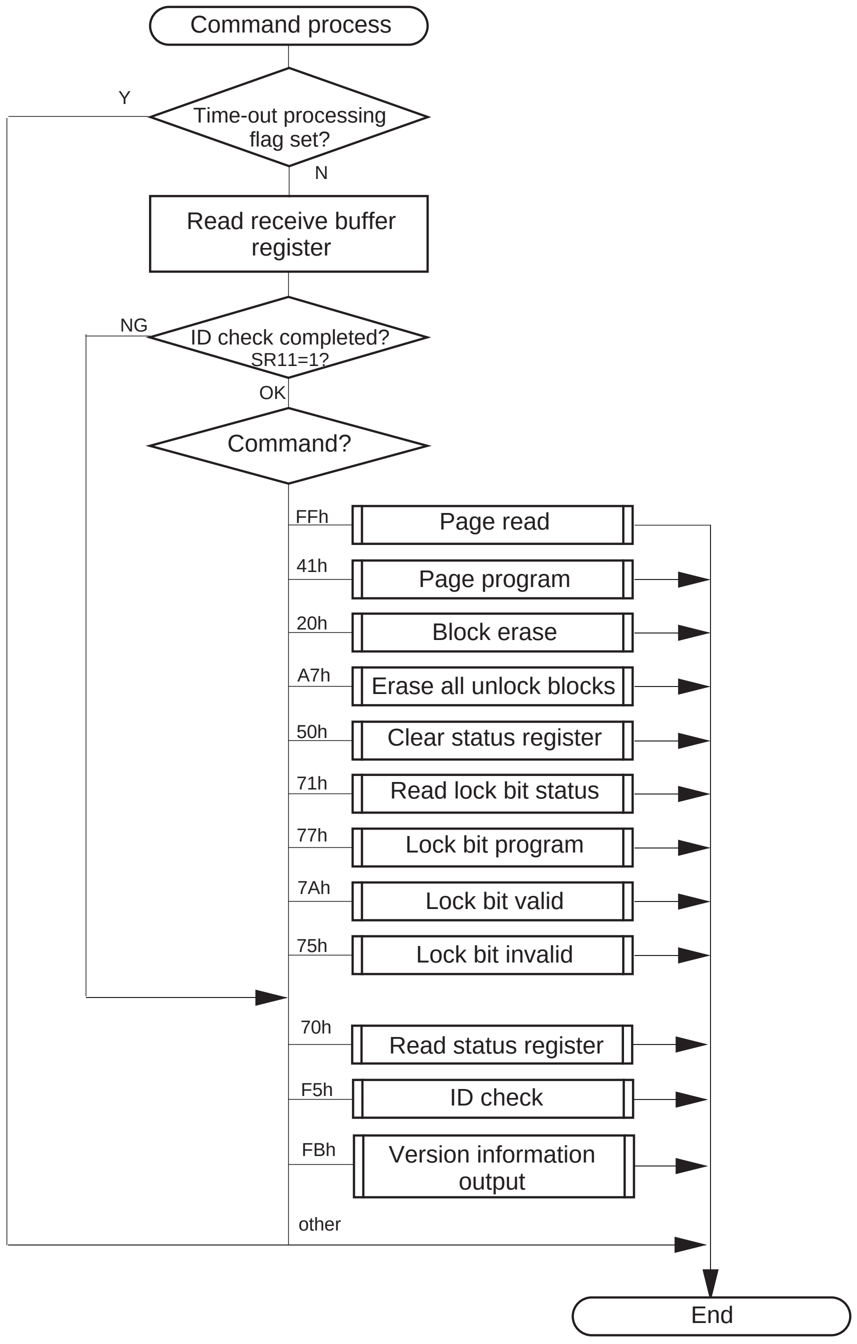


Figure 4.2.10 Command Processing

Page Read

To read data from the user area in blocks of 256 bytes, read address is stored to RAM and Read Array command (FF_{16}) is written. The address of the read area is changed from $xxxx00_{16}$ to $xxxxFF_{16}$, and the data following $xxxx00_{16}$ is transferred in succession.
Figure 4.2.11 shows processing flowchart.

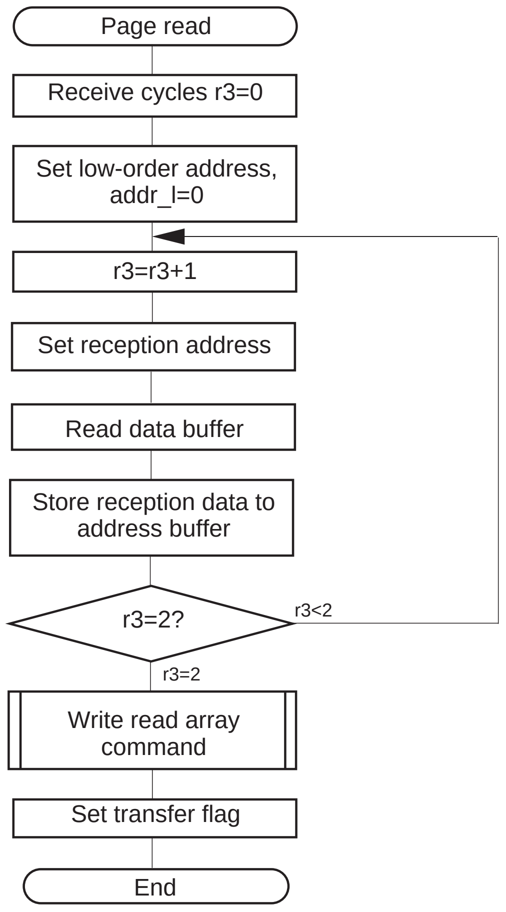


Figure 4.2.11 Page Read

Page Program

Data is written into the user area in blocks of 256 bytes.
Read 258 bytes data from RAM: 2 bytes of address and 256 bytes of write data received from serial programer. Status data is read from the flash memory. The read status is checked. When it is under error state, processing does not write but returns to the main part.
When it is not under error state, the page program command (41_{16}) is written in the flash memory, then 256 bytes of data is written. After data has been written, the read array command (FF_{16}) is written and processing returns to the main part.
Figure 4.2.12 shows processing flow.

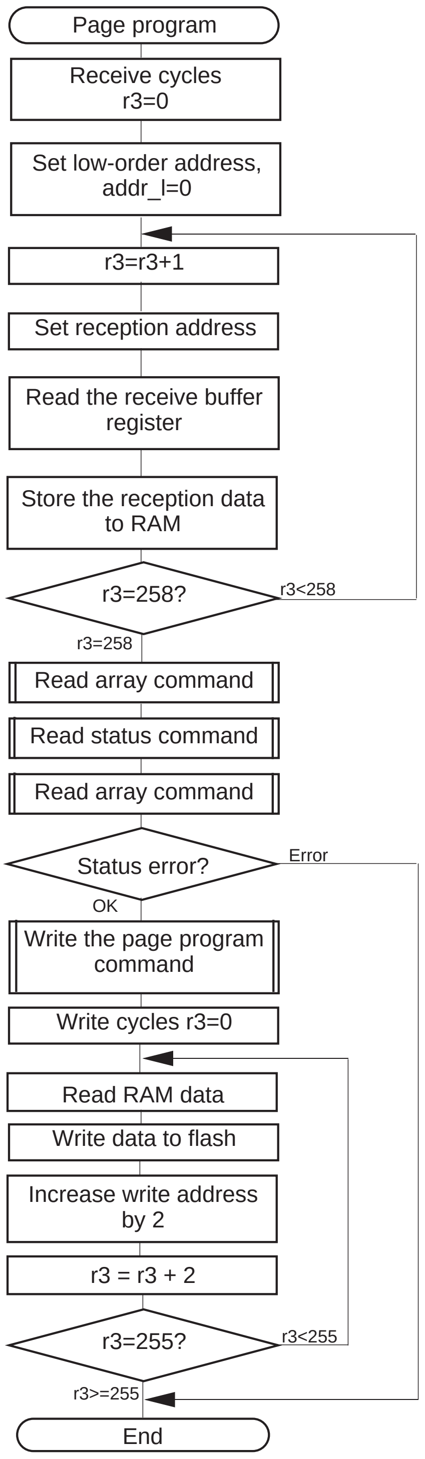


Figure 4.2.12 Page Program

Block Erase

A specified block area of the flash memory is erased. However, blocks that are locked by Lock Bit Program command cannot be erased. To erase these blocks, you need to disable the lock bit.
After confirming the two bytes of address and one byte of confirm command ($D0_{16}$) received from the serial programmer and stored in RAM, write Block Erase command (20_{16}) and confirm command ($D0_{16}$) to the area specified by the received address for block erase processing.
If the received confirm command is incorrect, block erase processing cannot be performed. In this case, write Read Array command (FF_{16}) to the flash memory to return the processing to the main routine.
Figure 4.2.13 shows a processing flow.

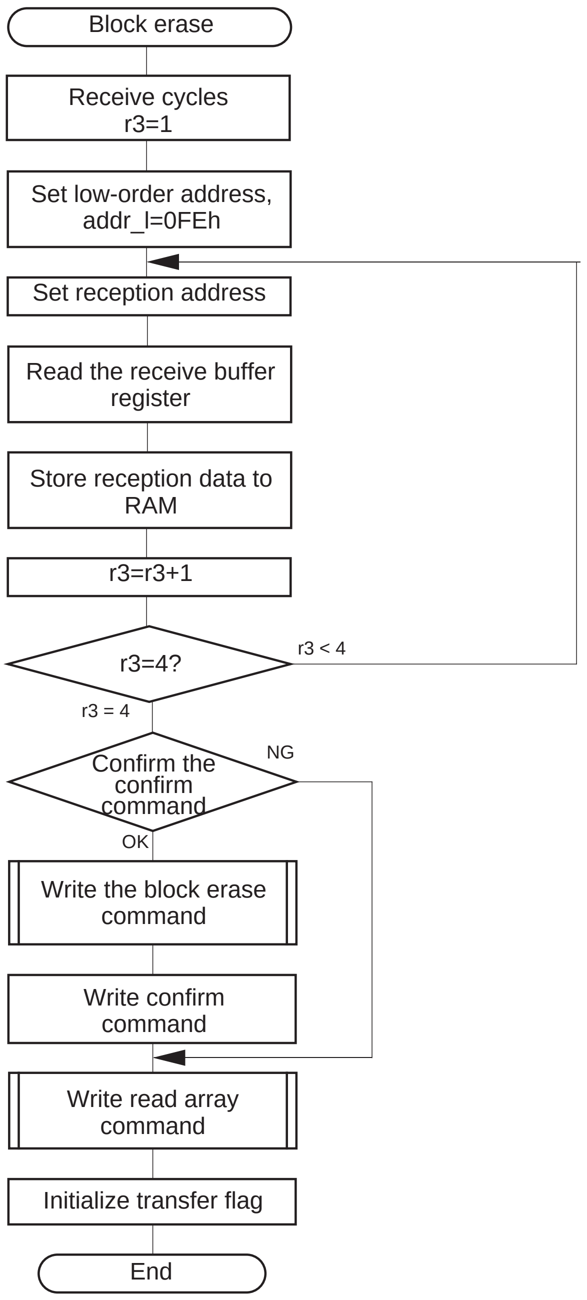


Figure 4.2.13 Block Erase

Erase All Unlock Blocks

A specified block area of the flash memory is erased. However, blocks that are locked by Lock Bit Program command cannot be erased. To erase these blocks, you need to disable the lock bit.
When the all erase command is received from a serial programmer, receive more 1 byte data in succession. After the second data is checked to see if it is confirm command ($D0_{16}$), write Erase All Unlock command (20_{16}) and confirm command ($D0_{16}$) to the area specified by the received address for erase all unlock blocks processing.
If the received confirm command is incorrect, erase all unlock blocks processing cannot be performed. In this case, write Read Array command (FF_{16}) to the flash memory to end the processing.
Figure 4.2.14 shows a processing flow.

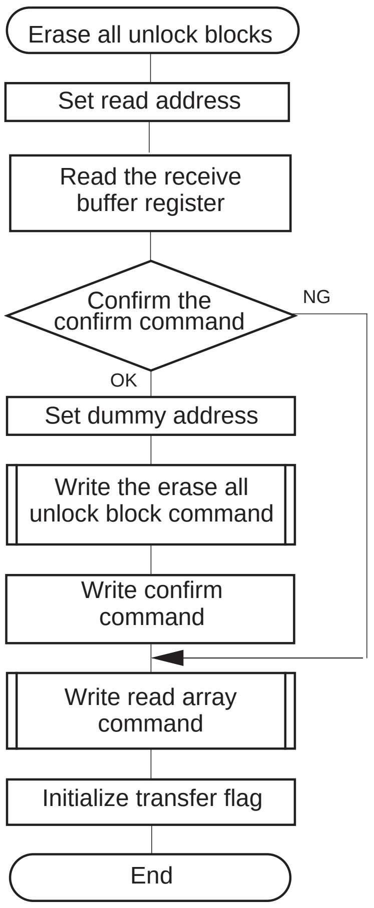


Figure 4.2.14 Erase All Unlock Block

Read Status Register

Two bytes of status data indicating the flash memory's operating status is stored to RAM to transmit via serial I/O.
Write the Read Array command (FF_{16}) to the flash memory, then write the Read Status command (70_{16}).
After status register reception, write the Read Array command and return to the main routine.
Figure 4.2.15 shows a processing flow.

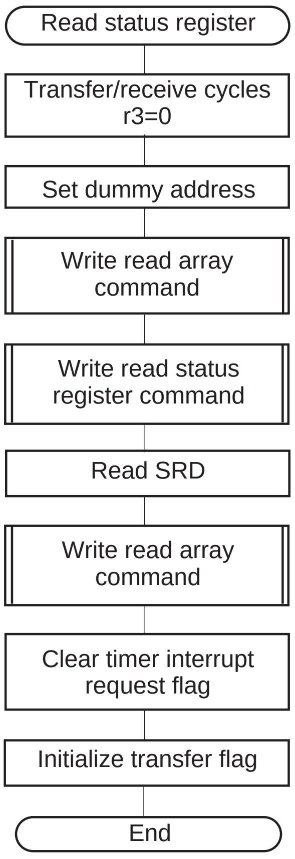


Figure 4.2.15 Read Status Register

Clear Status Register

Status register error information is cleared.
The Read Array command (FF_{16}), Clear Status command (50_{16}) and Read Array command (FF_{16}) are written into the flash memory in succession.
The logic sum for the status register 1 (SRD1) is obtained on $9C_{16}$ and the error flag is cleared. Processing returns to the main part.
Figure 4.2.16 shows a processing flow.

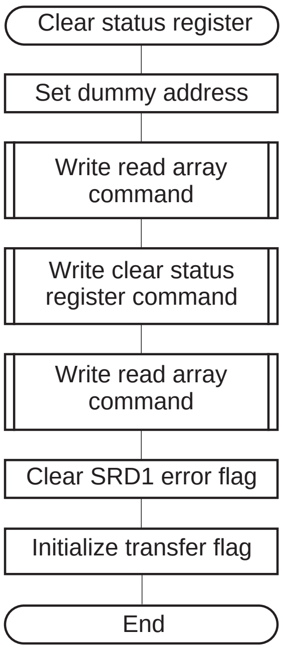


Figure 4.2.16 Clear Status Register

Read Lock Bit Status

One byte of data indicating the lock status of each individual block in the flash memory is saved via serial I/O. Of the 1-byte data, the 6th bit indicates lock status. When "1", the block is unlocked. When "0", the block is locked.
After receiving two byte of data indicating address, store specified address in the address buffer. At this time, set #FE$_{16}$ to the low order address.
The Read Array command (FF$_{16}$) and the Read Lock Bit command (71$_{16}$) are written and, there after, the lock bit information is read from the flash memory. After the lock bit information has been read, the Read Array command (FF$_{16}$) is written again. Processing then returns to main part.
Figure 4.2.17 shows a processing flow.

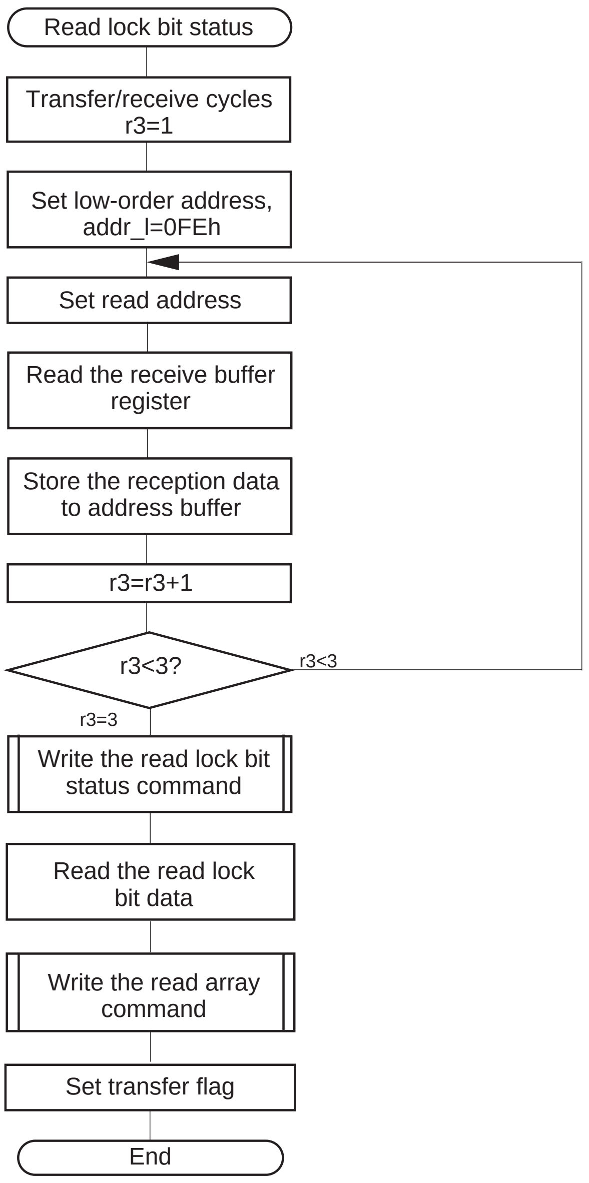

Read lock bit status
Transfer/receive cycles r3=1
Set low-order address, addr_l=0FEh
Set read address
Read the receive buffer register
Store the reception data to address buffer
r3=r3+1
r3<3?
r3<3
r3=3
Write the read lock bit status command
Read the read lock bit data
Write the read array command
Set transfer flag
End

Figure 4.2.17 Read Lock Bit Status

Lock Bit Program

Blocks in the flash memory is locked. Locked block areas cannot be erased.
After receiving two byte of data indicating address, store specified address in the address buffer. At this time, set #FE_{16} to the low order address.
If the Received Confirm command is incorrect, lock bit program processing cannot be performed. If correct, for lock bit program processing, write the Lock Bit Program command (77_{16}) to the flash memory and the Confirm command ($D0_{16}$) in succession. Write Read Array command (FF_{16}) and processing returns to the main part.
Figure 4.2.18 shows a processing flow.

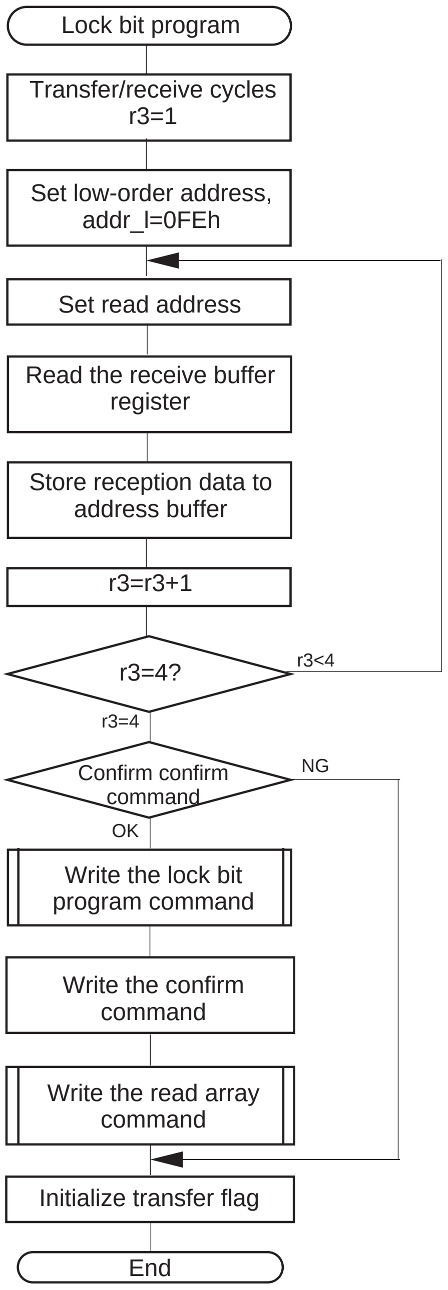

Lock bit program
Transfer/receive cycles r3=1
Set low-order address, addr_l=0FEh
Set read address
Read the receive buffer register
Store reception data to address buffer
r3=r3+1
r3=4?
r3<4
r3=4
Confirm confirm command
NG
OK
Write the lock bit program command
Write the confirm command
Write the read array command
Initialize transfer flag
End

Figure 4.2.18 Lock Bit Program

Lock Bit Enable/Disable

Enables/disables the lock bit function of flash memory. The lock bit disable command cancels the lock on all blocks.
To enable the lock bit, "0" is written for the lock bit cancel bit. To disable the lock bit, "0" followed by "1" is written for the lock bit cancel bit.
Figure 4.2.19 shows a processing flow.

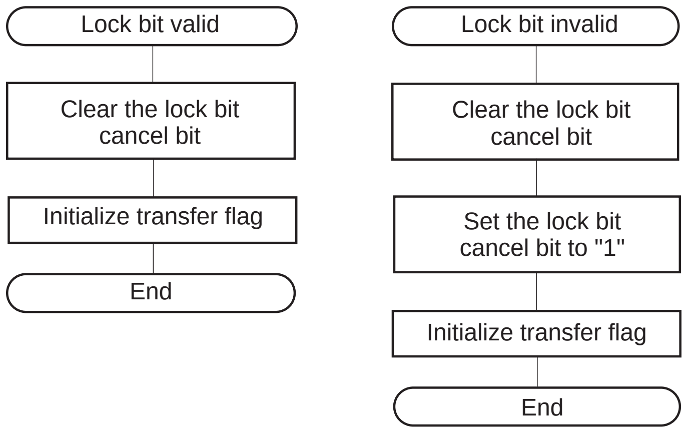

Lock bit valid
Clear the lock bit cancel bit
Initialize transfer flag
End
Lock bit invalid
Clear the lock bit cancel bit
Set the lock bit cancel bit to "1"
Initialize transfer flag
End

Figure 4.2.19 Lock Bit Enable/Disable

ID Check

The ID data stored in the flash memory is compared with the data received by serial I/O. This process judges whether the flash memory is blank or not. When blank, the ID check is ended and processing returns to the main part. When something is written in the ROM, the received ID address, the ID data size and ID data contents are checked. When mismatch, ID check error is generated (SR10 = 1, SR11 = 0) and processing returns to the main part. When match, the ID check is ended (SR10 = 1, SR11 = 1) and processing returns to the main part.
Figure 4.2.20 shows a processing flow.

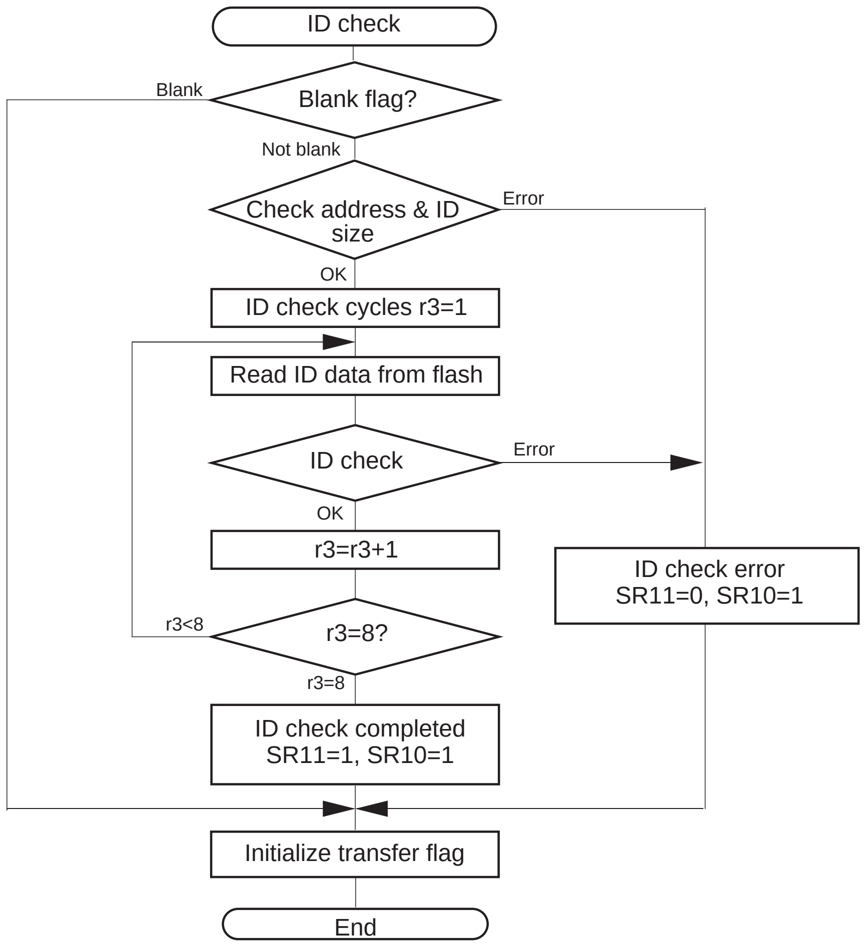


Figure 4.2.20 ID Check

Version Information Output

Transfer flag is set to transfer the version information of the boot program via serial I/O.
Figure 4.2.21 shows a processing flow.

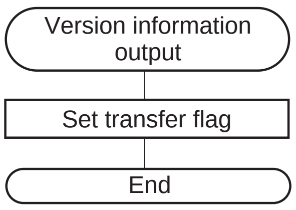


Figure 4.2.21 Version Information Output

Data Transfer Processing

The result of process after receiving a control command from serial programer is transfered via serial I/O. When transfer flag is 0, or time-out flag is 1, the processing returns to the main part. Otherwise next process is executed. Command buffer is read, the serial command is compared, and processing branches to the process in the match commands. After processing, initialize the transfer flag and return to main part. With mismatch command, initialize the transfer flag and return to main part.
Figure 4.2.22 shows a processing flow.

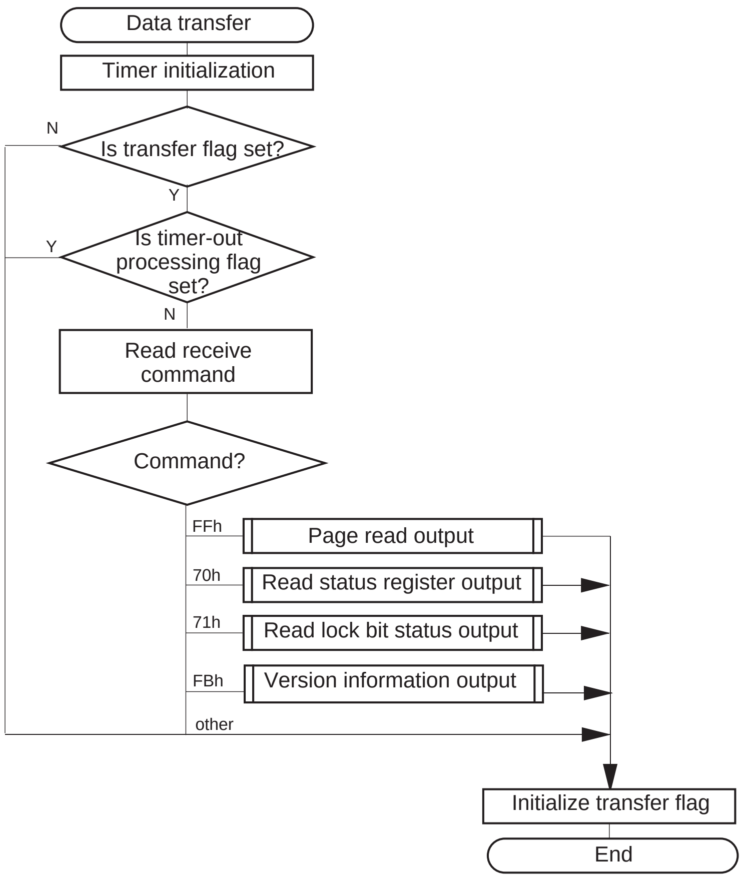


Figure 4.2.22 Data Transfer

Page Read Transfer Processing

Data from the user area in blocks of 256 bytes is read and the read data is sent via serial I/O.
Data is read from the flash memory and set to transfer buffer register. The timer used to check data reception time-out is started. When data is not received within 300 msec, a time-out error is judged, time-out processing flag is set and processing jumps to data transfer processing. After 256 bytes of data is received, processing jumps to data transfer processing.
Figure 4.2.23 shows a processing flow.

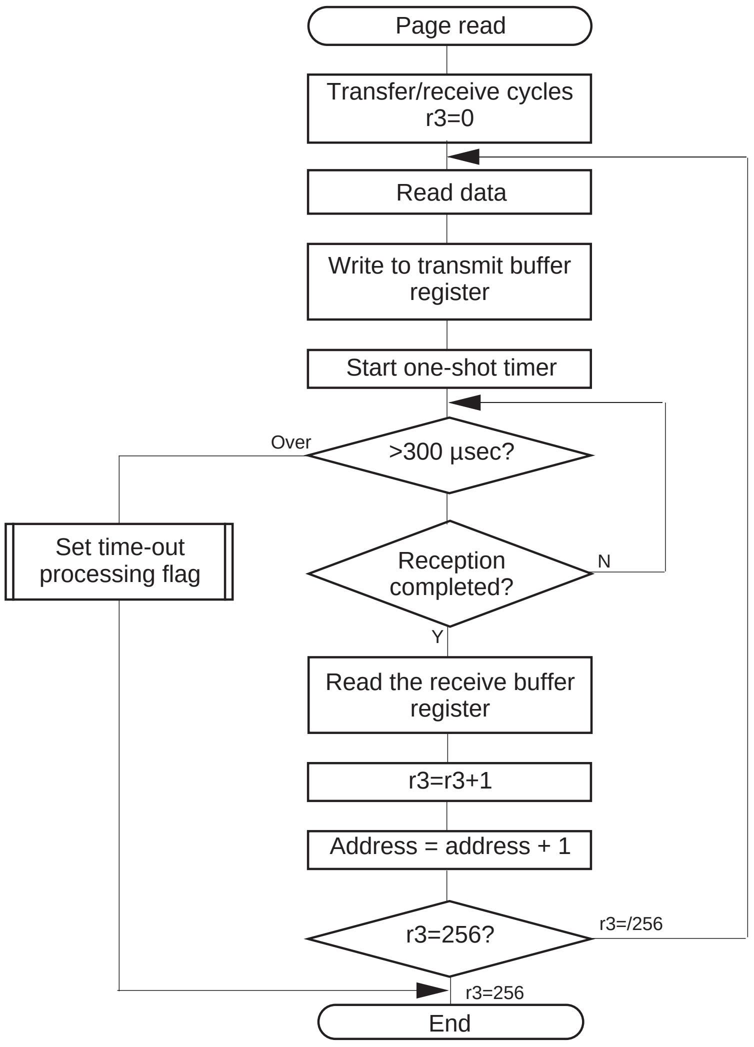


Figure 4.2.23 Page Read Transfer Processing

Read Status Register Transfer Processing

The two-byte status data (SRD: status register and SRD1: status register 1) that indicates flash memory operating status is sent via serial I/O.
The SRD is read from flash memory and written into transmit buffer register. The timer used to check data reception time-out is started. When data is not received within 300 msec, a time-out error is judged, time-out processing flag is set and processing jumps to data transfer processing. After data reception is completed, receive buffer register is read.
The SRD1 is read from flash memory and written into transmit buffer register. The timer used to check data reception time-out is started. When data is not received within 300 msec, a time-out error is judged, time-out processing flag is set and processing jumps to data transfer processing. After data reception is completed, reception buffer register is read and processing returns to data transfer processing.
Figure 4.2.24 shows a processing flow.

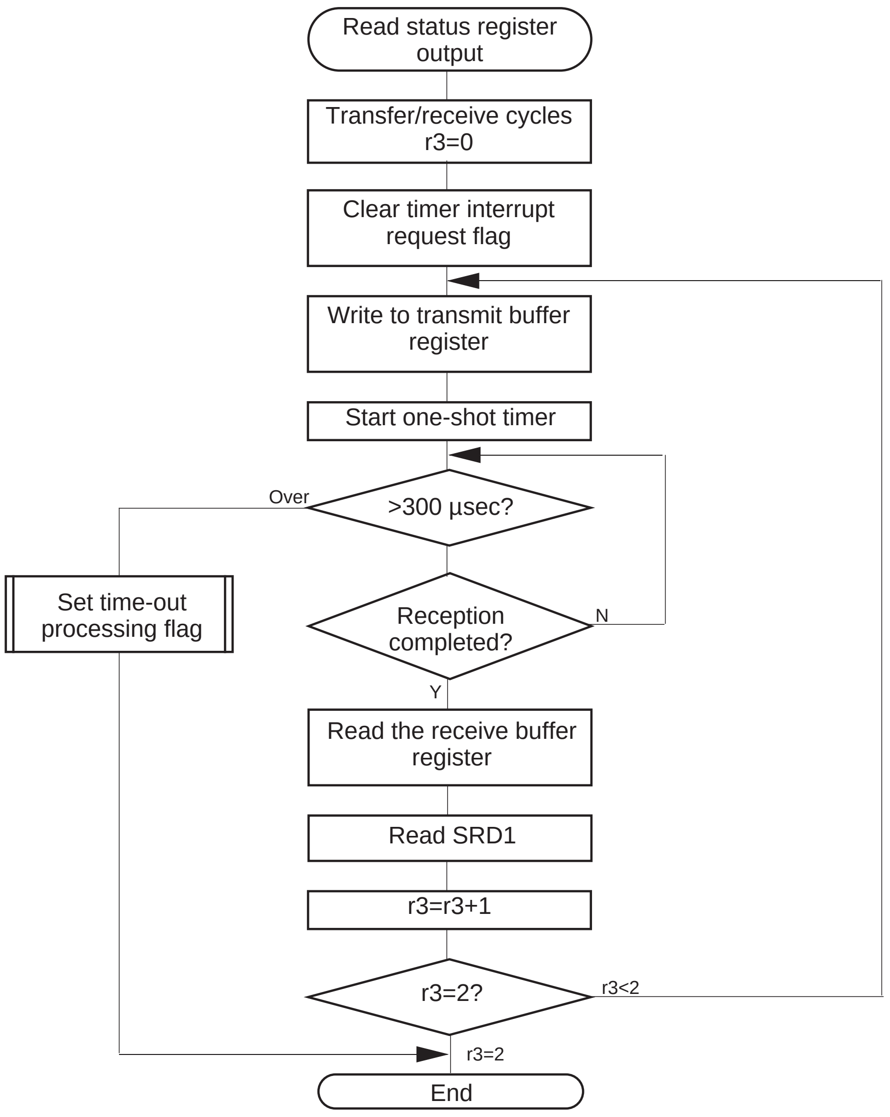

Read status register output
Transfer/receive cycles r3=0
Clear timer interrupt request flag
Write to transmit buffer register
Start one-shot timer
Over
>300 µsec?
Set time-out processing flag
Reception completed?
N
Y
Read the receive buffer register
Read SRD1
r3=r3+1
r3=2?
r3<2
r3=2
End

Figure 4.2.24 Read Status Register Transfer Processing

Read Lock Bit Status Transfer Processing

The lock bit status that set in command processing is sent via serial I/O.
The lock bit status data that set in command processing is read from RAM and written into transmit buffer register. The timer used to check data reception time-out is started. When data is not received within 300 msec, a time-out error is judged, time-out processing flag is set and processing jumps to data transfer processing. After data reception is completed, processing jumps to data transfer processing.
Figure 4.2.25 shows a processing flow.

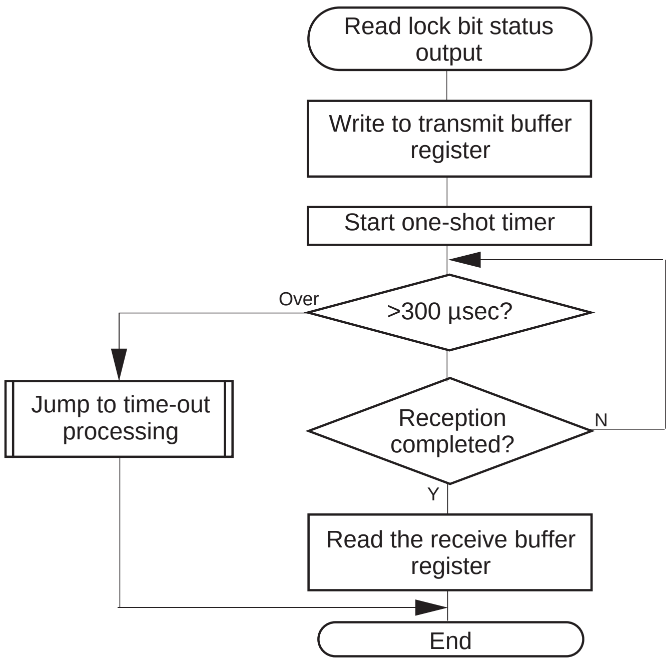

Read lock bit status output
Write to transmit buffer register
Start one-shot timer
Over
>300 µsec?
Jump to time-out processing
Reception completed?
N
Y
Read the receive buffer register
End

Figure 4.2.25 Read Lock Bit Data Transfer Processing

Version Information Output Processing

The version information of boot program is sent via serial I/O.
Version information is read and written in the transmit buffer register.
The timer used to check data reception time-out is started. When data is not received within 300 msec, a time-out error is judged, time-out processing flag is set and processing jumps to data transfer processing.
After all version information is send, processing jumps to data transfer processing.
Figure 4.2.26 shows a processing flow.

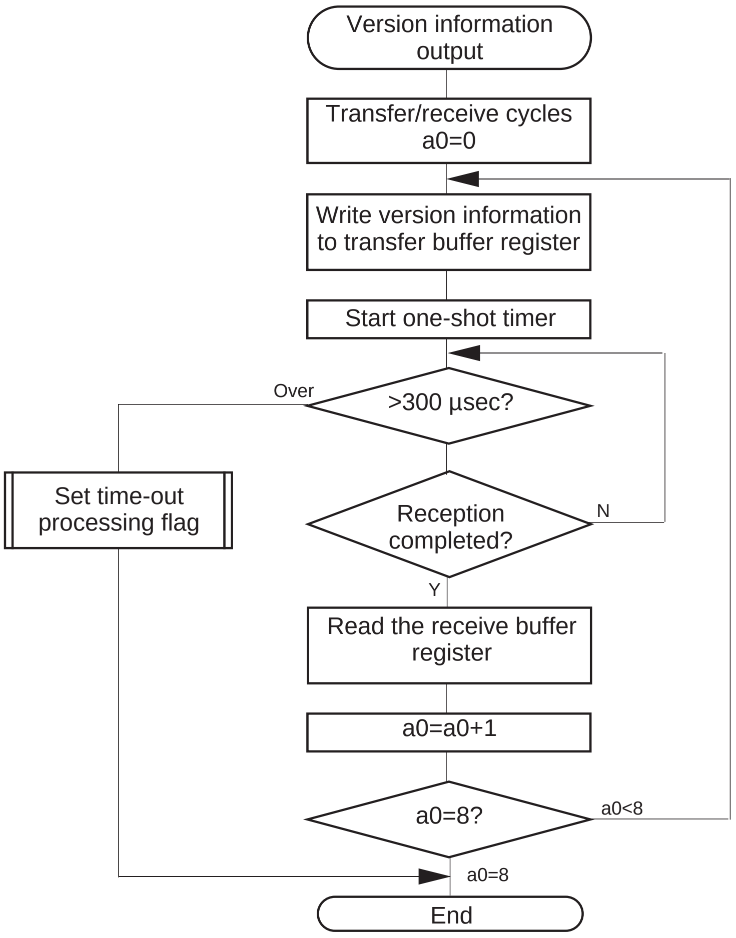

Version information output
Transfer/receive cycles a0=0
Write version information to transfer buffer register
Start one-shot timer
Over
>300 µsec?
Set time-out processing flag
Reception completed?
N
Y
Read the receive buffer register
a0=a0+1
a0=8?
a0<8
a0=8
End

Figure 4.2.26 Version Information Output Processing

Time-Out Processing

When time-out flag is set, serial I/O and time-out flag are initialized.
Figure 4.2.27 shows a processing flow.

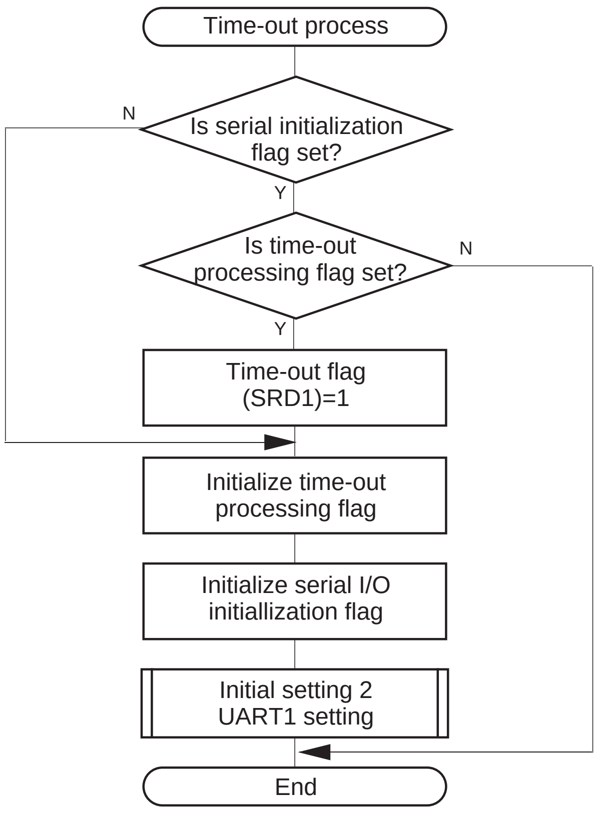

Time-out process
Is serial initialization flag set?
N
Y
Is time-out processing flag set?
N
Y
Time-out flag (SRD1)=1
Initialize time-out processing flag
Initialize serial I/O initiallization flag
Initial setting 2 UART1 setting
End

Figure 4.2.27 Time-Out Processing

Command Write

Commands are written in the flash memory. Commands are accepted when the flash memory is in the ready state (RY/BY signal status flag [bit 0 in address 03B7$_{16}$ of the flash memory control register] is "1").
Figure 4.2.28 shows a processing flow.

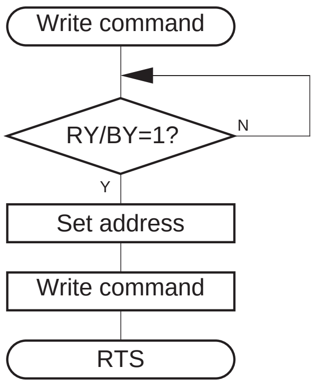


Figure 4.2.28 Command Write

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 writing the read status register command (70_{16}). Also, the status register is cleared by writing the clear status register command (50_{16}). Table 4.2.2 shows the definition of each status register bit. After clearing the reset, the status register outputs "80_{16}".

Table 4.2.2 Status Register (SRD)

Each bit of SRD	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". When the erase status is cleared, it 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". When the program status is cleared, it is set to "0".

Block Status After Program (SR3)

If excessive data is written (phenomenon whereby the memory cell becomes depressed which results in data not being read correctly), "1" is set for the block status after-program at the end of the page write operation. In other words, when writing ends successfully, "80_{16}" is output; when writing fails, "90_{16}" is output; and when excessive data is written, "88_{16}" is output.
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}) and clear the status register.

Status Register 1 (SRD1)

Status register 1 indicates the status of serial communications, results from ID checks and results from check sum comparisons. It can be read after the SRD by writing the Read Status Register command (70_{16}).
Also, status register 1 is cleared by writing the Clear Status Register command (50_{16}).
Table 4.2.3 gives the definition of each status register 1 bit. "00_{16}" is output when power is turned ON and the flag status is maintained even after the reset.

Table 4.2.3 Status Register 1 (SRD1)

Each bit of SRD1	Status name	Definition	
		"1"	"0"
SR15 (bit7)	Boot update completed bit	Update completed	Not update
SR14 (bit6)	Reserved	-	-
SR13 (bit5)	Reserved	-	-
SR12 (bit4)	Checksum match bit	Match	Mismatch
SR11 (bit3) SR10 (bit2)	ID check completed bits	00 Not verified 01 Verification mismatch 10 Reserved 11 Verified	
SR9 (bit1)	Data receive time out	Time out	Normal operation
SR8 (bit0)	Reserved	-	-

Boot Update Completed Bit (SR15)

This flag indicates whether the control program was downloaded to the RAM or not, using the download function.

Checksum Match Bit (SR12)

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

ID Check Completed Bits (SR11 and SR10)

These flags indicate the result of ID checks. Some commands cannot be accepted without an ID check.

Data Reception Time Out (SR9)

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

4.3 Sample List

This section shows a sample list of the program described in Section 4.2.
In addition to the processing explained in Section 4.2, the sample shown below includes the programmer command processing used by a synchronous serial programmer and the command processing used by an asynchronous serial communication programmer (M16C Flash Start).

Source

```
;**********************************************************
;*   System Name  : Sample Program for M16C/80 Flash      *
;*   MCU          : M30800FCFP                            *
;*   Xin          : 2M-20MHz (for UART mode )             *
;*   Assembler    : AS308 ver 1.00.00                     *
;*   Linker       : LN308 ver 1.00.00                     *
;*   Converter    : LMC308 ver 1.00.01                    *
;*-------------------------------------------------------*
;*   Copyright,1999 MITSUBISHI ELECTRIC CORPORATION       *
;*      AND MITSUBISHI SEMICONDUCTOR SYSTEM CORPORATION   *
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Include file                                       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    .list       off
    .include    sfr800.inc
    .include    flash800.inc
    .list       on
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Version table                                    +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
    .section    rom,code
    .org        0ffe000h
    .byte       'VER.0.00(VER.1.00)'
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Program section start                              +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    .section    prog,code
    .org        Boot_TOP
    .sb         SB_base
    .sbsym      SRD
    .sbsym      SRD1
    .sbsym      ver
    .sbsym      SF
    .sbsym      addr_l
    .sbsym      addr_m
    .sbsym      addr_h
;
;=========================================================
;+     Boot program start                                 +
;=========================================================
Reset:
;-----------------------------------------------
;+     Initialize_1                             +
;-----------------------------------------------
    ldc     #Istack,ISP     ; stack pointer set
    ldc     #SB_base,SB     ; SB register set
;
    bset    busy
    bset    busy_d          ; BUSY "H"output
    bclr    s_mode_d        ; Serial mode select input
;
;-------------------------------------
;+     Hot start & RAM clear          +
;-------------------------------------
;
    mov.w   #0,a0
Start_check:
    cmp.w   #55aah,buff[a0]
    jne     RAM_clear
    add.w   #2,a0
    cmp.w   #18,a0
    jltu    Start_check
    bset    ram_check       ; RAM check OK flag set
    jmp     CPU_set
;
RAM_clear:
    mov.w   #0,r0
    mov.w   #(Ram_END+1-Ram_TOP)/2,r3
    mov.w   #Ram_TOP,a1
    sstr.w
;
    mov.w   #0,a0
Buff_set:
    mov.w   #55aah,buff[a0]
    add.w   #2,a0
    cmp.w   #18,a0
    jltu    Buff_set
;
;-------------------------------------
;+  Processor mode register           +
;+    & System clock control register +
;-------------------------------------
CPU_set:
    mov.b   #3,prcr         ; Protect off
    mov.w   #0000h,pm0      ; wait off
    mov.b   #04h,mcd        ; f4
    mov.b   #20h,cm1
    mov.b   #08h,cm0        ;
    mov.b   #0,prcr         ; Protect on
;
Reload_chack:
    btst    sr15            ; Update ?
    jc      Transfer_end
    btst    ram_check       ; Reload ?
    jz      Version_inf      ;
    btst    s_mode
    bxor    old_mode
    jnc     Transfer_end
;
;-------------------------------------
;+     Version information            +
;-------------------------------------
Version_inf:
    mov.w   #0,a0           ; a0=0
Ver_loop:
    mov.w   0ffe000h+9[a0],ver[a0] ; Version data store
    add.w   #2,a0           ; address increment
    cmp.w   #8,a0           ; a0=8 ?
    jltu    Ver_loop        ; jump Ver_loop at a0<8
;
;-------------------------------------------------
;+     Program_transfer                          +
;-------------------------------------------------
```

```
    btst    s_mode          ; Serial I/O mode select
    jz      Transfer2       ; UART mode
;
Transfer1:
    bset    old_mode        ; clock synchronous mode
    mov.l   #Trans_TOP1,a0  ; Transfer source address
    mov.w   #Ram_progTOP,a1 ; Transfer destination address
    mov.w   #(Trans_END1 - Trans_TOP1)/2,r3 ; Transfer number
    smovf.w                 ; String move
    jmp     Transfer_end
;
Transfer2:
    bclr    old_mode        ; UART mode
    mov.l   #Trans_TOP2 ,a0 ; Transfer source address
    mov.w   #Ram_progTOP,a1 ; Transfer destination address
    mov.w   #(Trans_END2 - Trans_TOP2)/2,r3 ; Transfer number
    smovf.w                 ; String move
Transfer_end:
;-------------------------------------------------
;+     Jump to RAM                               +
;-------------------------------------------------
    jmp Ram_progTOP

;-------------------------------------------------
;+     Download program                          +
;-------------------------------------------------
    .org    Download_program
;
    jsr     set_TA0

    mov.w   #0,r3           ; receive number (r3=0)
    mov.w   #0,a1           ; sumcheck buffer
    bclr    sr15            ; Download flag reset
    bclr    sr12            ; Check sum flag reset
Download_loop:
    jsr     SIO_D_rcv
    btst    tout_flg        ; time out error ?
    jc      Download_err    ; jump Download_err at time out
    mov.w   rcv_d,r0        ; receive data read --> r0
    add.w   #1,r3
    cmp.w   #3,r3           ; r3=3 ?
    jgtu    Version_store   ; jump Version_store at r3>3
    mov.w   r3,a0           ; r3 --> a0
    mov.b   r0l,addr_l[a0]  ; Store program size
    mov.w   #0,a0           ; a0 initialize
    cmp.w   #3,r3           ; r3 = 3 ?
    jne     Download_loop   ; No, jump to Download_loop
    cmp.w   #0,addr_m       ; program size = 0 ?
    jz      Version_inf     ; jump to Version_inf at program size error
    jmp     Download_loop   ; jump Download_loop
Version_store:
    cmp.w   #11,r3          ; r3=11 ?
    jgtu    Program_store   ; jump Program_store at r3 >11
    mov.b   r0l,ver[a0]     ; version data store to RAM
    jmp     Program_store_1
;
Program_store:
    mov.b   r0l,Ram_progTOP-8[a0]   ; program data store to RAM
Program_store_1:
    add.b   r0l,a1          ; add data to a1
    add.w   #1,a0           ; a0(downloa0 offset) +1 increment
    cmp.w   addr_m,a0       ; a0 = program size (addr_m,h)?
    jltu    Download_loop   ; jump Download_loop at a0< program size
    jmp     SUM_Check       ; jump SUM Check
```

```
;
Download_err:
    bset    busy            ; busy "H"
    bset    busy_d          ; busy output
    mov.b   #0,u1c1         ; transmit/receive disable
    mov.b   #0,u1mr         ; u1mr reset
    jmp     Version_inf
;
;-------------------------------------------------
;+     Download program - UART mode -           +
;-------------------------------------------------
    .org    U_Download_program
;
    mov.w   #0,r3           ; receive number (r3=0)
    mov.w   #0,a1           ; sumcheck buffer
    bclr    sr15            ; Download flag reset
    bclr    sr12            ; Check sum flag reset
U_Download_loop:
    jsr     U_SIO_D_rcv
    mov.w   rcv_d,r0
    add.w   #1,r3           ; r3 +1 increment
    cmp.w   #3,r3           ; r3=3 ?
    jgtu    U_Version_store ; jump U_Version_store at r3>3
    mov.w   r3,a0           ; r3 --> a0
    mov.b   r0l,addr_l[a0]  ; Store program size
    mov.w   #0,a0           ; a0 initialize
    cmp.w   #3,r3           ; r3 = 3 ?
    jne     U_Download_loop ; No, jump U_Download_loop
    cmp.w   #0,addr_m       ; program size = 0 ?
    jz      Version_inf     ; jump Version_inf at program size error
    jmp     U_Download_loop
U_Version_store:
    cmp.w   #11,r3          ; r3=11 ?
    jgtu    U_Program_store   ; jump U_Program_store at r3 >11
    mov.b   r0l,ver[a0]     ; version data store to RAM
    jmp     U_Program_store_1
;
U_Program_store:
    mov.b   r0l,Ram_progTOP-8[a0]   ; program data store to RAM
U_Program_store_1:
    add.b   r0l,a1          ; add data to a1
    add.w   #1,a0           ; a0(downloa0 offset) +1 increment
    cmp.w   addr_m,a0       ; a0 = program size (addr_m,h)?
    jltu    U_Download_loop   ; jump  Download_loop at a0< program size
;
SUM_Check:
    mov.w   a1,r0
    cmp.b   data,r0l        ; compare check sum
    bmeq    sr12            ; check sum flag set at data=r0l
    jne     Version_inf     ; jump Version_inf at check sum error
    bset    sr15            ; Download flag set
    jmp     Ram_progTOP     ; jump Ram_progTOP
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : a synchronized signal I/O receive dwn+
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SIO_D_rcv:
    mov.b   r1l,u1tb
    bset    ta0os               ; ta0 start
?:
    btst    ir_ta0ic            ; time out error ?
    bmc     sr9                 ; time out flag set
    jc      SIO_D_rcv_err       ; jump SIO_D_rcv_err
    btst    ri_u1c1             ; receive complete ?
    jnc ?-
    mov.w   u1rb,rcv_d          ; receive data read --> r0
SIO_D_rcv_end:
    rts

SIO_D_rcv_err:
    bset    tout_flg
    jmp     SIO_D_rcv_end
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : UART receive dwn                      +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_SIO_D_rcv:
    btst    ri_u1c1             ; receive complete ?
    jnc     U_SIO_D_rcv
    mov.w   u1rb,rcv_d          ; receive data read --> r0
    rts
;
;===========================================================
;+  Transfer Program -- clock synchronous serial I/O mode +
;+       (1) Main flow                                     +
;+       (2) Flash control program                         +
;+             Read,Program,Erase,All_erase,etc.           +
;+       (3) Other program                                 +
;+             ID_check,Download,Version_output etc.       +
;===========================================================
    .section    dump,code
    .org        Trans_TOP1
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Main flow  - clock synchronous serial I/O mode -   +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Main:
    jsr     Initialize_2        ; clock synchronous serial I/O mode
    mov.b   #0,data
Loop_main:
    mov.b   SRD1,SRD1_bak       ; SRD1 back up
    mov.b   SRD1,SRD1_bak+2
;
    jsr     time_init
    jsr     SIO_rcv_first_data
    jsr     Flash_func
    jsr     SIO_send_data
    jsr     Time_out
    jmp     Loop_main
;
;---------------------------------------
;+     initialize       SIO           +
;---------------------------------------
time_init:
    bclr    tout_flg
    bclr    tint_flg
    bset    ta0os
    mov.b   #0,ta0ic
Loop_main1:
    btst    ir_ta0ic            ; 300 usec ?
    jz      Loop_main1
    bset    rcv_flg
    rts
;
;---------------------------------------
;+     SI/O time out                  +
;---------------------------------------
Time_out:
    btst    tint_flg
    jc      Time_out_init
```

```
        btst    tout_flg
        jnc     Time_out_end
        bset    sr9                 ; SRD1 time out flag set
        bclr    tout_flg
    Time_out_init:
        bclr    tint_flg
        jsr     Initialize_21        ; command error,UART1 reset
    Time_out_end:
        rts
    ;
    ;---------------------------------------
    ;+     SI/O recieve data              +
    ;---------------------------------------
    SIO_rcv_first_data:
        mov.b   #0,cmd_d
        bclr    cmd_flg
        btst    rcv_flg
        jnc     SIO_rcv_end
        btst    tout_flg
        jc      SIO_rcv_end
        mov.b   #0,ta0ic
        mov.w   #0,r2
    ;
    SIO_rcv_first_data_loop:
        mov.b   #0ffh,r1l           ; #ffh --> r1l (transfer data)
        mov.b   r1l,u1tb
        btst    cmd_flg
        jc      SIO_rcv_first_data_loop1
        bclr    busy_d              ; busy input
    ?:  btst    busy                ; Reception start?
        jz      ?-
    SIO_rcv_first_data_loop1:
        bset    ta0os               ; 300 usec timer start
    ;
    SIO_rcv_first_data_loop2:
        btst    ir_ta0ic            ; 300 usec ?
        jnc     ?+
        bset    tout_flg            ; time out
    ?:  btst    tout_flg
        jc      SIO_rcv_end
        btst    ri_u1c1             ; receive complete ?
        jz      SIO_rcv_first_data_loop2
        mov.w   u1rb,r0             ; receive data --> r0
        mov.w   r2,a0
        mov.b   r0l,data[a0]
        add.w   #1,r2
    ;
        btst    cmd_flg
        jc      SIO_loop_chk
        bset    cmd_flg
        mov.b   r0l,cmd_d
    ;
        mov.w   #15,a0
    SIO_rcv_command_chk:
        mov.b   Index_tbl-Trans_TOP1+Ram_progTOP-1[a0],r0h
        cmp.b   r0h,r0l
        jeq     SIO_cmd_jmp_2
        sbjnz.w  #1,a0,SIO_rcv_command_chk
        jmp     SIO_rcv_end_1
    ;
    SIO_cmd_jmp_2:
        shl.w   #1,a0
        mov.w   jmp_tbl_2-Trans_TOP1+Ram_progTOP-2[a0],r0
    SIO_cmd_jmp_2_1:
        jmpi.w  r0
```

```
;
SIO_2:
    mov.w   #2,loop_cnt
    jmp     SIO_loop_chk
SIO_259:
    mov.w   #259,loop_cnt
    jmp     SIO_loop_chk
SIO_4:
    mov.w   #4,loop_cnt
    jmp     SIO_loop_chk
SIO_3:
    mov.w   #3,loop_cnt
    jmp     SIO_loop_chk
;
;-------------------------------------
;+     ID check     SI/O             +
;-------------------------------------
SIO_rcv_ID_check:
    mov.w   #0,r3               ; receive number (r3=0)
    mov.w   #0ffh,a1            ; ID size (dummy data = ffh)
    mov.b   #0,ta0ic
SIO_ID_data_store:
    cmp.w   a1,r3               ; r3=a1(ID size)
    jeq     SIO_ID_address_check; jump ID_address_check at r3=ID size
    mov.b   r1l,u1tb            ; data transfer
    bset    ta0os               ; ta0 start
SIO_ID_data_loop:
    btst    ir_ta0ic            ; 300 usec ?
    jnc     ?+
    bset    tout_flg            ; time out
?:  btst    tout_flg
    jc      SIO_ID_address_check
    btst    ri_u1c1             ; receive complete ?
    jnc     SIO_ID_data_loop
    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 ?
    jne     SIO_ID_data_store   ; jump ID_data_store at r3 not= 4
    mov.b   data,a1             ; ID size --> a1
    add.w   #4,a1               ; a1=a1+4
    jmp     SIO_ID_data_store   ; jump ID_data_store
SIO_ID_address_check:
    jmp     SIO_rcv_end
;
SIO_rcv_end_1:
    bset    tint_flg
    jmp     SIO_rcv_end

SIO_loop_chk:
    cmp.w   loop_cnt,r2
    jltu    SIO_rcv_first_data_loop

SIO_rcv_end:
    bclr    cmd_flg
    bclr    rcv_flg
    rts
;
;-------------------------------------
;+     SIO_send data                 +
;-------------------------------------
SIO_send_data:
    jsr     set_TA0
    btst    send_flg
    jnc     SIO_send_data_end
    btst    tout_flg
    jc      SIO_send_data_end
    mov.b   cmd_d,r1h
;
    cmp.b   #0ffh,r1h           ; Read(ffh)
    jeq     Read_data
    cmp.b   #070h,r1h           ; Read SRD (70h)
    jeq     Read_SRD_data
    cmp.b   #071h,r1h           ; Read LB (71h)
    jeq     Read_LB_data
    cmp.b   #0fbh,r1h           ; Version_output(fbh)
    jeq     Ver_output_data
    cmp.b   #0fdh,r1h           ; Read_check(fdh)
    jeq     Read_check_data
    cmp.b   #0fch,r1h           ; Boot_check(fch)
    jeq     Boot_data
    jmp     SIO_send_func
;
Read_check_data:
    mov.w   #0,r3
    mov.w   sum,r1
Read_check_data_loop:
    mov.b   r1l,u1tb
    bset    ta0os               ; ta0 start
Read_check_data_check:
    btst    ir_ta0ic
    jnc     ?+
    bset    tout_flg
?:
    btst    tout_flg
    jc      SIO_send_data_end
    btst    ri_u1c1             ; receive complete ?
    jnc     Read_check_data_check
    mov.w   u1rb,r0             ; receive data read --> r0
    mov.b   r1h,r1l
    add.w   #1,r3
    cmp.w   #2,r3
    jltu    Read_check_data_loop
Read_check_data_end:
    mov.w   #0,sum              ; reset
    jmp     SIO_send_data_end
;
Read_data:
    mov.w   #0,r3

Read_data_loop:
    mov.b   [a1],r1l          ; Flash memory read
    mov.b   r1l,u1tb
    bset    ta0os               ; ta0 start
Read_data_chk:
    btst    ir_ta0ic            ; 300 usec ?
    jnc     ?+
    bset    tout_flg            ; time out
?:
    btst    tout_flg
    jc      Read_data_end
    btst    ri_u1c1             ; receive complete ?
    jnc     Read_data_chk
    mov.w   u1rb,r0             ; receive data read --> r0
    add.w   #1,r3
    add.l   #1,a1
    cmp.w   #256,r3             ; r3 = 256 ?
    jne     Read_data_loop
Read_data_end:
    jmp     SIO_send_data_end
;
Ver_output_data:
    mov.w   #0,a0               ; Version address offset (a0=0)
Ver_output_data_loop:
    mov.b   ver[a0],u1tb        ;send_data set
    bset    ta0os               ; ta0 start
Ver_output_data_check:
    btst    ir_ta0ic            ; 300 usec ?
    jnc     ?+
    bset    tout_flg            ; time out
?:
    btst    tout_flg
    jc      Ver_output_data_end
    btst    ri_u1c1             ; receive complete ?
    jnc     Ver_output_data_check
    mov.w   u1rb,r0             ; receive data read --> r0
    add.w   #1,a0
    cmp.w   #8,a0               ; a0 = 8 ?
    jne     Ver_output_data_loop
Ver_output_data_end:
    jmp     SIO_send_data_end
;
Read_SRD_data:
    mov.w   #0,r3
Read_SRD_data_loop:
    bclr    tout_flg            ; clear time out
    mov.b   #0,ta0ic            ; clear time out
    mov.b   r1l,u1tb            ; data transfer
    bset    ta0os               ; ta0 start ; test
Read_SRD_data_check:
    btst    ir_ta0ic            ; 300 usec ?
    jnc     ?+
    bset    tout_flg            ; time out
?:  btst    tout_flg
    jc      Read_SRD_data_end
    btst    ri_u1c1             ; receive complete ?
    jnc     Read_SRD_data_check
    mov.w   u1rb,r0             ; receive data read --> r0
    mov.b   SRD1,r1l            ; SRD1 data --> r1l
    add.w   #1,r3
    cmp.w   #2,r3               ; r3 = 2 ?
    jltu    Read_SRD_data_loop  ; jump Read_SRD_loop at r3<2
Read_SRD_data_end:
    jmp     SIO_send_data_end
;
Read_LB_data:
Read_LB_data_loop:
    mov.b   r1l,u1tb            ; data transfer
    bset    ta0os               ; ta0 start
Read_LB_data_check:
    btst    ir_ta0ic            ; 300 usec ?
    jnc     ?+
    bset    tout_flg            ; time out
?:
    btst    tout_flg
    jc      Read_LB_data_end
    btst    ri_u1c1             ; receive complete ?
    jnc     Read_LB_data_check
    mov.w   u1rb,r0             ; receive data read --> r0
Read_LB_data_end:
    jmp     SIO_send_data_end
;
Boot_data:
    bclr    fmr05
    mov.w   addr_l,a0
    mov.b   addr_h,a1
    mov.w   #0,r3
    sha.l   #16,a1
    add.l   a0,a1
Boot_data_loop:
    mov.b   [a1],r1l        ; Boot data read
    mov.b   r1l,u1tb
    bset    ta0os               ; ta0 start
Boot_data_chk:
    btst    ir_ta0ic            ; 300 usec ?
    jnc     ?+
    bset    tout_flg            ; time out
?:
    btst    tout_flg
    jc      Boot_data_end
    btst    ri_u1c1             ; receive complete ?
    jnc     Boot_data_chk
    mov.w   u1rb,r0             ; receive data read --> r0
    add.w   #1,r3
    add.l   #1,a1
    cmp.w   #256,r3             ; r3 = 256 ?
    jne     Boot_data_loop
Boot_data_end:
    bset    fmr05
    jmp     SIO_send_data_end
;
SIO_send_func:
    mov.w   start_cnt,r3
SIO_send_data_loop:
    mov.w   r3,a0
    mov.b   data[a0],r1l
    mov.b   r1l,u1tb            ; data transfer
    bset    ta0os               ; ta0 start
SIO_send_chk:
    btst    ir_ta0ic            ; 300 usec ?
    jnc     ?+
    bset    tout_flg            ; time out
?:  btst    tout_flg
    jc      SIO_send_data_end
    btst    ri_u1c1             ; receive complete ?
    jnc     SIO_send_chk
    mov.w   u1rb,r0             ; receive data read --> r0
    add.w   #1,r3
    cmp.w   send_cnt,r3         ; r3 = send_cnt ?
    jne     SIO_send_data_loop
    mov.w   r3,r0
SIO_send_data_end:
    bclr    send_flg
    rts
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : Time_over_flg                         +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Time_over_flg:
    bset    tout_flg
    rts

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      jump table for Flash_func                          +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
jmp_tbl:
    .word   Read - cmd_jmp
    .word   Program - cmd_jmp
    .word   Erase - cmd_jmp
    .word   All_erase - cmd_jmp
    .word   Clear_SRD - cmd_jmp
    .word   Read_LB - cmd_jmp
    .word   Program_LB - cmd_jmp
    .word   LB_enable - cmd_jmp
    .word   LB_disable - cmd_jmp
    .word   Download - cmd_jmp
    .word   Boot_output - cmd_jmp
    .word   Read_check - cmd_jmp

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     jump table for SIO_rcv_first_data                  +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
jmp_tbl_2:
    .word   SIO_3 - SIO_cmd_jmp_2_1         ; Read
    .word   SIO_259 - SIO_cmd_jmp_2_1       ; Program
    .word   SIO_4 - SIO_cmd_jmp_2_1         ; erase
    .word   SIO_2 - SIO_cmd_jmp_2_1         ; All erase
    .word   SIO_rcv_end - SIO_cmd_jmp_2_1   ; Clear SRD
    .word   SIO_3 - SIO_cmd_jmp_2_1         ; Read LB
    .word   SIO_4 - SIO_cmd_jmp_2_1         ; LB Program
    .word   SIO_rcv_end - SIO_cmd_jmp_2_1   ; LB enable
    .word   SIO_rcv_end - SIO_cmd_jmp_2_1   ; LB disable
    .word   SIO_rcv_end - SIO_cmd_jmp_2_1   ; Download
    .word   SIO_3 - SIO_cmd_jmp_2_1         ; Boot output
    .word   SIO_rcv_end - SIO_cmd_jmp_2_1   ; Read check
    .word   SIO_rcv_end - SIO_cmd_jmp_2_1   ; Read SRD
    .word   SIO_rcv_ID_check - SIO_cmd_jmp_2_1 ; ID check
    .word   SIO_rcv_end - SIO_cmd_jmp_2_1   ;  Version out
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     serch table for Flash_func,SIO_rcv_first_data      +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Index_tbl:
    .byte   0ffh                ; Read(ffh)
    .byte   041h                ; Program(41h)
    .byte   020h                ; Erase(20h)
    .byte   0a7h                ; All_erase(a7h)
    .byte   050h                ; Clear SRD(50h)
    .byte   071h                ; Read LBS(71h)
    .byte   077h                ; LB program(77h)
    .byte   07ah                ; LB enable (7ah)
    .byte   075h                ; LB disable(75h)
    .byte   0fah                ; Download (fah)
    .byte   0fch                ; Boot output(fch)
    .byte   0fdh                ; Read check(fdh)
    .byte   070h                ; Read SRD(70h)
    .byte   0f5h                ; ID check(f5h)
    .byte   0fbh                ; Version output(fbh)
;

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Subroutine : Initialize_2                          +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Initialize_2:
;-------------------------------------
;+     Flash mode set                +
;-------------------------------------
Flash_mode:
    bset    fmr05            ; User ROM select
    bclr    fmr01            ; Flash entry bit clear
    bset    fmr01            ; Flash entry bit set (E/W mode)
;
;-------------------------------------
;+     Blank check                   +
;-------------------------------------
    mov.w   0fffffch,r0      ; Reset vector read
    mov.w   0fffffeh,r1      ; Reset vector read
    and.w   r1,r0           ; r0 & r1
    cmp.w   #0ffffh,r0      ; r0=ffffh ?
    jne     blank_end
;
    bset    sr10            ; check complete at r0=ffffh
    bset    sr11
    bset    blank           ; blank flag set
blank_end:
;
;-------------------------------------
;+     UART1                       +
;-------------------------------------
Initialize_21:
;
    bclr    pd6_2           ; RxD-input
;
;----- 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                    +
;---------------------------------------
set_TA0:
;----- 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
;
    rts
;
;---------------------------------------
;+     FLASH function main            +
;---------------------------------------
Flash_func:
    btst    tout_flg
    jc      Flash_func_end
    bclr    ta0s
    mov.b   cmd_d,r0l           ; receive data --> r0l
;
    mov.b   #0ch,r0h            ; #00001100b sr10,11 mask data
    and.b   SRD1,r0h            ; sr10,11 pick up
    cmp.b   #0ch,r0h            ; ID check OK?
    jne     Command_check_2     ; jump Command_check_2 at ID unchecked
    mov.w   #12,a0
;
Command_check:
    mov.b   Index_tbl-Trans_TOP1+Ram_progTOP-1[a0],r0h
    cmp.b   r0h,r0l
    jeq     cmd_jmp_1
    sbjnz.w  #1,a0,Command_check
    jmp     Command_check_2
;
cmd_jmp_1:
    shl.w   #1,a0
    mov.w   jmp_tbl-Trans_TOP1+Ram_progTOP-2[a0],r0
cmd_jmp:
    jmpi.w  r0
;
Command_check_2:
?:  cmp.b   #070h,r0l           ; Read SRD  (70h)
    jne     ?+
    jmp     Read_SRD
?:  cmp.b   #0f5h,r0l           ; ID check  (f5h)
    jne     ?+
    jmp     ID_check
?:  cmp.b   #0fbh,r0l           ; Version out   (fbh)
    jne     Flash_func_end
    jmp     Ver_output
;
Flash_func_end:
    rts
;
;-----------------------------------------------------------
;+     Read                                                +
;-----------------------------------------------------------
Read:
    mov.w   #0,r3           ; receive number
    mov.b   #0,addr_l       ; addr_l = 0
```

```
Read_loop:
    add.w   #1,r3           ; r3 +1 increment
    mov.w   r3,a0           ; r3 --> a0
    mov.w   data[a0],r0
    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   #00ffh,r2       ; Read array command
    jsr     Command_write   ; command_write
    bset    send_flg
    mov.w   #3,start_cnt
    mov.w   #258,send_cnt
    jmp     Flash_func_end      ; jump  Flash_func_end
;

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : Command write                       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Command_write:
    btst    fmr00            ; RY/BY status check
    jz      Command_write
    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
    mov.w   r2,[a1]       ; command write
    rts
;
;---------------------------------------------------------
;+     Program                                           +
;---------------------------------------------------------
Program:
    mov.w   #0,r3           ; receive number
    mov.b   #0,addr_l       ; addr_l = 0
    mov.w   sum,crcd        ; for Read check command
Program_loop_1:
    add.w   #1,r3           ; r3 +1 increment
    mov.w   r3,a0           ; r3 --> a0
    mov.b   data[a0],r0l
    mov.b   r0l,addr_l[a0]  ; Store address
    cmp.w   #259,r3         ; r3 = 259 ?
    jltu    Program_loop_1  ; jump Program_loop_1 at r3<258
;
    mov.w   #00ffh,r2       ; Read array command
    jsr     Command_write   ; command_write
    mov.w   #0070h,r2       ; Read SRD command
    jsr     Command_write   ; Command_write
    mov.w   [a1],r1         ; SRD read
    mov.w   #00ffh,r2       ; Read array command
    jsr     Command_write   ; command_write
    cmp.b   #80h,r1l        ; error check
    jne     Program_end
;
    mov.w   #0041h,r2       ; Page program command
    jsr     Command_write   ; command_write
    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
    mov.w   r1,[a1]         ; data write
;
```

```
        mov.b   r1l,crcin       ; for Read check command
        mov.b   r1h,crcin       ; for Read check command
;
        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:
        mov.w   crcd,sum
        bclr    send_flg
        mov.w   #0,send_cnt
        mov.w   #0,start_cnt
        jmp     Flash_func_end      ; jump Flash_func_end
;

;-----------------------------------------------------------
;+     Block erase                                         +
;-----------------------------------------------------------
Erase:
        mov.w   #1,r3           ; receive number (r3=1)
        mov.b   #0feh,addr_l    ; addr_l = feh
Erase_loop:
        mov.w   r3,a0           ; r3 --> a0
        mov.b   data[a0],r0l
        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
        mov.w   #0020h,r2       ; Erase command
        jsr     Command_write   ; command write
        mov.w   #00d0h,r2       ; Confirm command
        mov.w   r2,[a1]         ; command write
Erase_end:
        mov.w   #00ffh,r2       ; Read array command
        jsr     Command_write   ; command_write
        bclr    send_flg
        mov.w   #0,send_cnt
        mov.w   #0,start_cnt
        jmp     Flash_func_end  ; jump Flash_func_end
;
;-----------------------------------------------------------
;+      All erase ( unlock block )                         +
;-----------------------------------------------------------
All_erase:
        mov.w   #1,a0
        mov.b   data[a0],r0l    ; receive data read --> r0
        cmp.b   #0d0h,r0l       ; Confirm command check
        jne     All_erase_end   ; jump All_erase_end at Confirm command error
        mov.w   #0000h,addr_l   ; 0fe0000h --> addr
        mov.b   #00feh,addr_h
        mov.w   #00a7h,r2       ; All erase command
        jsr     Command_write   ; command write
        mov.w   #00d0h,r2       ; Confirm command
        mov.w   r2,[a1]         ; command write
All_erase_end:
        mov.w   #00ffh,r2       ; Read array command
        jsr     Command_write   ; command_write
        bclr    send_flg
        mov.w   #0,send_cnt
        mov.w   #0,start_cnt
        jmp     Flash_func_end      ; jump Flash_func_end
;
;-----------------------------------------------------------
```

```
;+     Read SRD                                           +
;----------------------------------------------------------
Read_SRD:
    mov.w   #0,r3           ; receive number (r3=0)
    mov.w   #0000h,addr_l   ; 0fe0000h --> addr
    mov.b   #00feh,addr_h   ;
    mov.w   #00ffh,r2       ; Read array command
    jsr     Command_write   ; command_write
    mov.w   #0070h,r2       ; Read SRD command
    jsr     Command_write   ; command write
    mov.w   [a1],r1         ; SRD read
    mov.w   #00ffh,r2       ; Read array command
    jsr     Command_write   ; command_write
    mov.w   #1,start_cnt
    mov.w   #3,send_cnt
    bset    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;

;----------------------------------------------------------
;+     Clear SRD                                          +
;----------------------------------------------------------
Clear_SRD:
    mov.w   #0000h,addr_l   ; 0fe0000h --> addr
    mov.b   #00feh,addr_h   ;
    mov.w   #00ffh,r2       ; Read array command
    jsr     Command_write   ; command write
    mov.w   #0050h,r2       ; Clear SRD command
    jsr     Command_write   ; command write
    mov.w   #00ffh,r2       ; Read array command
    jsr     Command_write   ; command write
    and.b   #10011100b,SRD1 ; SRD1 clear
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;----------------------------------------------------------
;+     Read Lock Bit                                      +
;----------------------------------------------------------
Read_LB:
    mov.w   #1,r3           ; receive number (r3=1)
    mov.b   #0feh,addr_l    ; addr_l = feh
Read_LB_loop:
    mov.w   r3,a0           ; r3 --> a0
    mov.b   data[a0],r0l
    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
    mov.w   #0071h,r2       ; Read LB command
    jsr     Command_write   ; command write
    mov.w   [a1],r1         ; read LB
    mov.w   #00ffh,r2       ; Read array command
    jsr     Command_write   ; command write
Read_LB_end:
    mov.w   #1,start_cnt
    mov.w   #1,send_cnt
    bset    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
```

```
;-----------------------------------------------------------
;+     Program Lock Bit                                    +
;-----------------------------------------------------------
Program_LB:
    mov.w   #1,r3           ; receive number (r3=1)
    mov.b   #0feh,addr_l    ; addr_l = feh
Program_LB_loop:
    mov.w   r3,a0           ; r3 --> a0
    mov.b   data[a0],r0l
    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
    mov.w   #0077h,r2       ; Program LB command
    jsr     Command_write   ; command write
    mov.w   #00d0h,r2       ; Confirm command
    mov.w   r2,[a1]         ; command write
    mov.w   #00ffh,r2       ; Read array command
    jsr     Command_write   ; command write
Program_LB_end:
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;---------------------------------------
;+     Lock Bit enable                 +
;---------------------------------------
LB_enable:
    bclr    fmr02               ; Lock disable bit = 0
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;---------------------------------------
;+     Lock Bit disable                +
;---------------------------------------
LB_disable:
    bclr    fmr02               ; Lock disable bit = 0
    bset    fmr02               ; Lock disable Bit = 1
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;-----------------------------------------------------------
;+     ID check                                            +
;-----------------------------------------------------------
ID_check:
    btst    blank           ; blank flag check
    jc      ID_check_end    ; jump ID_check_end at blank
    cmp.w   #0ffdfh,addr_l  ; lower ID address check
    jne     ID_error        ; jump ID_error at ID address error
    cmp.w   #007ffh,addr_h  ; higher ID address check
    jne     ID_error        ; jump ID_error at ID address error
ID_data_check:
    mov.w   #0ffdfh,r1      ; ID lower address --> r1
    mov.w   #1,r3           ; check loop number (r3=1)
ID_check_loop:
    mov.w   r1,a0           ; r1 --> a0
    mov.w   #000ffh,a1      ; ID higher address --> a1
    sha.l   #16,a1
    add.l   a0,a1
    mov.b   [a1],r0l        ; ID data read from Flash memory
    mov.w   r3,a0           ; r3 --> a0
    cmp.b   r0l,data[a0]    ; compare ID data
    jne     ID_error        ; jump ID_error at ID error
    add.w   #4,r1           ; r1 +4 increment (next ID address)
    cmp.w   #0ffe7h,r1      ; r1=0ffefh ?
    jne     ?+              ; jump ? at not equal
    mov.w   #0ffebh,r1      ; r1=0ffeb at equal
?:
    add.w   #1,r3           ; r3 +1 increment
    cmp.w   #8,r3           ; r3=8 ?
    jltu    ID_check_loop   ; jump ID_check_loop at r3<8
ID_OK:
    bset    sr10
    bset    sr11            ; ID check OK (sr11=1,sr10=1)
    jmp     ID_check_end    ; jump  ID_check_end
ID_error:
    bset    sr10
    bclr    sr11            ; ID error (sr11=0,sr10=1)
ID_check_end:
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;-----------------------------------------------------------
;+     Boot output                                         +
;-----------------------------------------------------------
Boot_output:
    bclr    fmr05            ; Boot ROM select
    mov.w   #0,r3           ; receive number (r3=1)
    mov.w   #0,addr_l       ; addr_l = 0
Boot_loop:
    add.w   #1,r3           ; r3 +1 increment
    mov.w   r3,a0           ; r3 --> a0
    mov.w   data[a0],r0
    mov.b   r0l,addr_l[a0]  ; Store address
    cmp.w   #2,r3           ; r3 = 3 ?
    jltu    Boot_loop; jump Boot_output_loop at r3<3
    bset    send_flg
    mov.w   #3,start_cnt
    mov.w   #258,send_cnt
    jmp     Flash_func_end      ; jump  Flash_func_end
;
;-----------------------------------------------------------
;+     Read check                                          +
;-----------------------------------------------------------
Read_check:
    mov.w   #0,start_cnt
    mov.w   #2,send_cnt
    bset    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;---------------------------------------
;+     Download                        +
;---------------------------------------
Download:
    bclr    fmr05               ; Boot ROM select
    jmp.a   Download_program    ; jump Download_program
;
;---------------------------------------
;+     Version output                  +
;---------------------------------------
```

```
Ver_output:
    mov.w   #0,start_cnt
    mov.w   #8,send_cnt
    bset    send_flg
    jmp     Flash_func_end      ; jump Flash_func_end
;
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : a synchronized signal I/O receive data+
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SIO_rcv_data:
    jsr     set_TA0
SIO_rcv_data_1:
    btst    ir_ta0ic            ; time out error ?
    jnc     ?+
    jsr     Time_over_flg       ; jump Time_over at time out
?:
    btst    ri_u1c1             ; receive complete ?
    jnc     SIO_rcv_data_1
    mov.w   u1rb,rcv_d          ; receive data read --> r0
    rts
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : a synchronized signal I/O receive data+
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SIO_rcv_data_rom:
    jsr     set_TA0
SIO_rcv_data_rom_1:
    btst    ir_ta0ic            ; time out error ?
    bmc     fmr05               ; time out, User ROM select
    jnc     ?+
    jsr     Time_over_flg       ; jump Time_over at time out
?:
    btst    ri_u1c1             ; receive complete ?
    jnc     SIO_rcv_data_rom_1
    mov.w   u1rb,rcv_d          ; receive data read --> r0
    rts
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : a synchronized signal I/O send       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SIO_send:
    jsr     set_TA0
    jsr     SIO_send_data
    jsr     SIO_rcv_data
    rts
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : a synchronized signal I/O send       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SIO_send_rom:
    jsr     set_TA0
    jsr     SIO_send_data
    jsr     SIO_rcv_data_rom
    rts
;
;===========================================================
;+  Transfer Program -- UART mode                         +
;+      (1) Main flow                                     +
;+      (2) Flash control program                         +
;+            Read,Program,All_erase,Read_SRD,Clear_SRD  +
;+      (3) Other program                                 +
;+            ID_check                                    +
;===========================================================
;
```

```
    .org        Trans_TOP2
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Main flow  - UART mode -                           +
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_Main:
    jmp     U_SIO_init_first
;
U_Loop_main:
    mov.b   SRD1,SRD1_bak       ; SRD1 back up
    mov.b   SRD1,SRD1_bak+2
;
    jsr     U_SIO_rcv
    mov.w   rcv_d,r0
    mov.b   r0l,cmd_d
    mov.w   #0,r2
    mov.w   r2,a0
    mov.b   r0l,data[a0]
    bclr    cmd_flg
;
    jmp     U_SIO_freq

U_Flash_init:
    jsr     U_time_init

    jmp     U_SIO_rcv_first_data
U_Flash_set:
    jmp     U_Flash_func
U_Flash_send:
    jmp     U_SIO_send_data
U_Flash_int:
    btst    tint_flg
    jnc     U_Main_end
    jsr     Initialize_31       ; command error,UART mode Initialize
;
U_Main_end:
    jmp     U_Loop_main         ; jump U_Loop_main
;
;--------------------------------------
;+     initialize       SIO           +
;--------------------------------------
U_time_init:
    bset    rcv_flg
    bclr    tint_flg
    rts
;
;--------------------------------------
;+     SI/O recieve data              +
;--------------------------------------
U_SIO_rcv_first_data:
    btst    rcv_flg
    jnc     U_SIO_rcv_end
    jc      U_SIO_rcv_first_data_set

U_SIO_rcv_first_data_loop:
    jsr     U_SIO_rcv_only
    mov.w   rcv_d,r0            ; receive data --> r0
U_SIO_rcv_first_data_set:
    mov.w   r2,a0
    mov.b   r0l,data[a0]
    add.w   #1,r2
;
    btst    cmd_flg
    jc      U_SIO_loop_chk
    bset    cmd_flg
    mov.b   r0l,cmd_d
;
    mov.w   #20,a0
U_SIO_rcv_command_chk:
    mov.b   U_Index_tbl-Trans_TOP2+Ram_progTOP-1[a0],r0h
    cmp.b   r0h,r0l
    jeq     U_SIO_cmd_jmp_2
    sbjnz.w  #1,a0,U_SIO_rcv_command_chk
    jmp     U_SIO_rcv_end

U_SIO_cmd_jmp_2:
    shl.w   #1,a0
    mov.w   U_jmp_tbl_2-Trans_TOP2+Ram_progTOP-2[a0],r0
U_SIO_cmd_jmp_2_1:
    jmpi.w  r0

U_SIO_2:
    mov.w   #2,loop_cnt
    jmp     U_SIO_loop_chk
U_SIO_259:
    mov.w   #259,loop_cnt
    jmp     U_SIO_loop_chk
U_SIO_4:
    mov.w   #4,loop_cnt
    jmp     U_SIO_loop_chk
U_SIO_3:
    mov.w   #3,loop_cnt
    jmp     U_SIO_loop_chk
;
;---------------------------------------
;+     ID check     SI/O              +
;---------------------------------------
U_SIO_rcv_ID_check:
    mov.w   #0,r3              ; receive number (r3=0)
    mov.w   #0ffh,a1           ; ID size (dummy data = ffh)
    mov.b   #0,ta0ic
U_SIO_ID_data_store:
    cmp.w   a1,r3              ; r3=a1(ID size)
    jeq     U_SIO_ID_address_check; jump ID_address_check at r3=ID size
    jsr     U_SIO_rcv_only
    mov.w   rcv_d,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 ?
    jne     U_SIO_ID_data_store ; jump ID_data_store at r3 not= 4
    mov.b   data,a1            ; ID size --> a1
    add.w   #4,a1              ; a1=a1+4
    jmp     U_SIO_ID_data_store ; jump ID_data_store
U_SIO_ID_address_check:
    jmp     U_SIO_rcv_end
;
U_SIO_rcv_end_1:
    bset    tint_flg
    jmp     U_SIO_rcv_end

U_SIO_loop_chk:
    cmp.w   loop_cnt,r2
    jltu    U_SIO_rcv_first_data_loop
U_SIO_rcv_end:
    bclr    cmd_flg
    bclr    rcv_flg
    jmp     U_Flash_set
;
;---------------------------------------
```

```
;+     SIO_send data                  +
;-------------------------------------
U_SIO_send_data:
    btst    send_flg
    jnc     U_SIO_send_data_end
    mov.b   cmd_d,r1h

    cmp.b   #0ffh,r1h           ; Read(ffh)
    jeq     U_Read_data
    cmp.b   #070h,r1h           ; Read SRD (70h)
    jeq     U_Read_SRD_data
    cmp.b   #071h,r1h           ; Read LB (71h)
    jeq     U_Read_LB_data
    cmp.b   #0fbh,r1h           ; Version_output(fbh)
    jeq     U_Ver_output_data
    cmp.b   #0fdh,r1h           ; Read_check(fdh)
    jeq     U_Read_check_data
    cmp.b   #0fch,r1h           ; Boot_check(fch)
    jeq     U_Boot_data
    cmp.b   #0b0h,r1h           ; BPS SET(b0h)
    jeq     U_BPS_B0_data
    cmp.b   #0b1h,r1h           ; BPS SET(b1h)
    jeq     U_BPS_B1_data
    cmp.b   #0b2h,r1h           ; BPS SET(b2h)
    jeq     U_BPS_B2_data
    cmp.b   #0b3h,r1h           ; BPS SET(b3h)
    jeq     U_BPS_B3_data
    cmp.b   #0b4h,r1h           ; BPS SET(b4h)
    jeq     U_BPS_B4_data
    jmp     U_SIO_send_func

U_Read_check_data:
    mov.w   #0,r3
    mov.w   sum,r1
U_Read_check_data_loop:
    mov.b   r1l,send_d
    jsr     U_SIO_send
    mov.b   r1h,r1l
    add.w   #1,r3
    cmp.w   #2,r3
    jltu    U_Read_check_data_loop
U_Read_check_data_end:
    mov.w   #0,sum              ; reset
    jsr     U_SIO_exit
    jmp     U_SIO_send_data_end

U_Read_data:
    mov.w   #0,r3
U_Read_data_loop:
    mov.b   [a1],r1l          ; Flash memory read
    mov.b   r1l,send_d
    jsr     U_SIO_send
    add.w   #1,r3
    add.l   #1,a1
    cmp.w   #256,r3             ; r3 = 256 ?
    jne     U_Read_data_loop
U_Read_data_end:
    jsr     U_SIO_exit
    jmp     U_SIO_send_data_end

U_Ver_output_data:
    mov.w   #0,a0               ; Version address offset (a0=0)
U_Ver_output_data_loop:
    mov.b   ver[a0],send_d      ; send_data set
    jsr     U_SIO_send
```

```
        add.w   #1,a0
        cmp.w   #8,a0               ; a0 = 8 ?
        jne     U_Ver_output_data_loop
    U_Ver_output_data_end:
        jsr     U_SIO_exit
        jmp     U_SIO_send_data_end

    U_Read_SRD_data:
        mov.w   #0,r3
    U_Read_SRD_data_loop:
        mov.b   r1l,send_d          ; data transfer
        jsr     U_SIO_send
        mov.b   SRD1,r1l            ; SRD1 data --> r1l
        add.w   #1,r3
        cmp.w   #2,r3               ; r3 = 2 ?
        jltu    U_Read_SRD_data_loop; jump Read_SRD_loop at r3<2
    U_Read_SRD_data_end:
        jsr     U_SIO_exit
        jmp     U_SIO_send_data_end

    U_Read_LB_data:
        mov.b   r1l,send_d          ; data transfer
        jsr     U_SIO_send
    U_Read_LB_data_end:
        jsr     U_SIO_exit
        jmp     U_SIO_send_data_end

    U_Boot_data:
        bclr    fmr05
        mov.w   addr_l,a0
        mov.b   addr_h,a1
        mov.w   #0,r3
        sha.l   #16,a1
        add.l   a0,a1
    U_Boot_data_loop:
        mov.b   [a1],r1l          ; Flash memory read
        mov.b   r1l,send_d
        jsr     U_SIO_send
        add.w   #1,r3
        add.w   #1,a1
        cmp.w   #256,r3             ; r3 = 256 ?
        jne     U_Boot_data_loop
    U_Boot_data_end:
        bset    fmr05
        jsr     U_SIO_exit
        jmp     U_SIO_send_data_end

    U_BPS_B0_data:
        mov.b   baud,data_BPS           ; Baud rate 9600bps
        jmp     U_BPS_SET_data
    U_BPS_B1_data:
        mov.b   baud+1,data_BPS         ; Baud rate 19200bps
        jmp     U_BPS_SET_data
    U_BPS_B2_data:
        mov.b   baud+2,data_BPS         ; Baud rate 38400bps
        jmp     U_BPS_SET_data
    U_BPS_B3_data:
        mov.b   baud+3,data_BPS         ; Baud rate 57600bps
        jmp     U_BPS_SET_data
    U_BPS_B4_data:
        mov.b   baud+4,data_BPS         ; Baud rate 115200bps
    U_BPS_SET_data:
        mov.b   r0l,send_d
        jsr     U_SIO_send
        jsr     U_SIO_exit
        jsr     U_blank_end         ; UART mode Initialize
        jmp     U_SIO_send_data_end
    ;
    U_SIO_send_func:
        mov.w   start_cnt,r3
    U_SIO_send_data_loop:
        mov.w   r3,a0
        mov.b   data[a0],r1l
        mov.b   r1l,send_d
        jsr     U_SIO_send
        add.w   #1,r3
        cmp.w   send_cnt,r3         ; r3 = send_cnt ?
        jne     U_SIO_send_data_loop
        mov.w   r3,r0
    U_SIO_send_data_end:
        bclr    send_flg
        jmp     U_Flash_int
    ;
    ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    ;+     jump table for Flash_func                            +
    ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    U_jmp_tbl:
        .word   U_Read - U_cmd_jmp
        .word   U_Program - U_cmd_jmp
        .word   U_Erase - U_cmd_jmp
        .word   U_All_erase - U_cmd_jmp
        .word   U_Clear_SRD - U_cmd_jmp
        .word   U_Read_LB - U_cmd_jmp
        .word   U_Program_LB - U_cmd_jmp
        .word   U_LB_enable - U_cmd_jmp
        .word   U_LB_disable - U_cmd_jmp
        .word   U_Download - U_cmd_jmp
        .word   U_Boot_output - U_cmd_jmp
        .word   U_Read_check - U_cmd_jmp
    ;
    ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    ;+     jump table for SIO_rcv_first_data                    +
    ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    U_jmp_tbl_2:
        .word   U_SIO_3 - U_SIO_cmd_jmp_2_1         ; Read
        .word   U_SIO_259 - U_SIO_cmd_jmp_2_1       ; Program
        .word   U_SIO_4 - U_SIO_cmd_jmp_2_1         ; erase
        .word   U_SIO_2 - U_SIO_cmd_jmp_2_1         ; All erase
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ; Clear SRD
        .word   U_SIO_3 - U_SIO_cmd_jmp_2_1         ; Read LB
        .word   U_SIO_4 - U_SIO_cmd_jmp_2_1         ; LB Program
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ; LB enable
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ; LB disable
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ; Download
        .word   U_SIO_3 - U_SIO_cmd_jmp_2_1         ; Boot output
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ; Read check
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ; Read SRD
        .word   U_SIO_rcv_ID_check - U_SIO_cmd_jmp_2_1 ; ID check
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ;  Version out
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ;  U_BPS_B0
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ;  U_BPS_B1
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ;  U_BPS_B2
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ;  U_BPS_B3
        .word   U_SIO_rcv_end - U_SIO_cmd_jmp_2_1   ;  U_BPS_B4
    ;
    ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    ;+     serch table for Flash_func,SIO_rcv_first_data        +
    ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    U_Index_tbl:
        .byte   0ffh                ; Read(ffh)
        .byte   041h                ; Program(41h)
        .byte   020h                ; Erase(20h)
        .byte   0a7h                ; All_erase(a7h)
        .byte   050h                ; Clear SRD(50h)
        .byte   071h                ; Read LBS(71h)
        .byte   077h                ; LB program(77h)
        .byte   07ah                ; LB enable (7ah)
        .byte   075h                ; LB disable(75h)
        .byte   0fah                ; Download (fah)
        .byte   0fch                ; Boot output(fch)
        .byte   0fdh                ; Read check(fdh)
        .byte   070h                ; Read SRD(70h)
        .byte   0f5h                ; ID check(f5h)
        .byte   0fbh                ; Version output(fbh)
        .byte   0b0h                ; BPS_SET 9600 (b0h)
        .byte   0b1h                ; BPS_SET 19200 (b1h)
        .byte   0b2h                ; BPS_SET 38400(b2h)
        .byte   0b3h                ; BPS_SET 57600(b3h)
        .byte   0b4h                ; BPS_SET 115200(b4h)
    ;
    ;
    ;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    ;+     Subroutine : Initialize_3 - UART mode              +
    ;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    Initialize_3:
    ;---------------------------------------
    ;+     Flash mode set                 +
    ;---------------------------------------
    ;
        bset    fmr05            ; User ROM select
        bclr    fmr01            ; Flash entry bit clear
        bset    fmr01            ; Flash entry bit set (E/W mode)
    ;
    ;---------------------------------------
    ;+     Blank check                    +
    ;---------------------------------------
        mov.w   0ffffffch,r0      ; Reset vector read
        mov.w   0ffffffeh,r1      ; Reset vector read
        and.w   r1,r0           ; r0 & r1
        cmp.w   #0ffffh,r0      ; r0=ffffh ?
        jne     U_blank_end
        bset    sr10            ; check complete at r0=ffffh
        bset    sr11
        bset    blank           ; blank flag set
    U_blank_end:
    ;
    ;---------------------------------------
    ;+     UART1                          +
    ;---------------------------------------
    ;----- UART init rate generator 1
    ;
        mov.w   data_BPS,u1brg
    ;
    Initialize_31:
    ;
        bclr    pd6_2           ; RxD input
    ;
    ;----- 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   #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
;
;-------------------------------------
;+     FLASH function main           +
;-------------------------------------
U_Flash_func:
    mov.b   cmd_d,r0l           ; receive data --> r0l

    mov.b   #0ch,r0h            ; #00001100b sr10,11 mask data
    and.b   SRD1,r0h            ; sr10,11 pick up
    cmp.b   #0ch,r0h            ; ID check OK?
    jne     U_Command_check_2   ; jump Command_check_2 at ID unchecked
    mov.w   #12,a0

U_Command_check:
    mov.b   U_Index_tbl-Trans_TOP2+Ram_progTOP-1[a0],r0h
    cmp.b   r0h,r0l
    jeq     U_cmd_jmp_1
    sbjnz.w  #1,a0,U_Command_check
    jmp     U_Command_check_2

U_cmd_jmp_1:
    shl.w   #1,a0
    mov.w   U_jmp_tbl-Trans_TOP2+Ram_progTOP-2[a0],r0
U_cmd_jmp:
    jmpi.w  r0
```

```
U_Command_check_2:
?:  cmp.b   #070h,r0l           ; Read SRD  (70h)
    jne     ?+
    jmp     U_Read_SRD
?:  cmp.b   #0f5h,r0l           ; ID check  (f5h)
    jne     ?+
    jmp     U_ID_check
?:  cmp.b   #0b0h,r0l           ; BPS_SET 9600  (b0h)
    jne     ?+
    jmp     U_BPS_B0
?:  cmp.b   #0b1h,r0l           ; BPS_SET 19200  (b1h)
    jne     ?+
    jmp     U_BPS_B1
?:  cmp.b   #0b2h,r0l           ; BPS_SET 38400  (b2h)
    jne     ?+
    jmp     U_BPS_B2
?:  cmp.b   #0b3h,r0l           ; BPS_SET 57600  (b3h)
    jne     ?+
    jmp     U_BPS_B3
?:  cmp.b   #0b4h,r0l           ; BPS_SET 115200  (b4h)
    jne     ?+
    jmp     U_BPS_B4
?:  cmp.b   #0fbh,r0l           ; Version out   (fbh)
    jne     U_Flash_func_end
    jmp     U_Ver_output
;
U_Flash_func_end:
    jmp     U_Flash_send
;

;-----------------------------------------------------------
;+     Read - UART mode -                                  +
;-----------------------------------------------------------
U_Read:
    mov.w   #0,r3           ; receive number
    mov.b   #0,addr_l       ; addr_l = 0
U_Read_loop:
    add.w   #1,r3           ; r3 +1 increment
    mov.w   r3,a0           ; r3 --> a0
    mov.w   data[a0],r0
    mov.b   r0l,addr_l[a0]  ; Store address
    cmp.w   #2,r3           ; r3 = 2 ?
    jltu    U_Read_loop     ; jump Read_loop at r3<2
    mov.w   #00ffh,r2       ; Read array command
    jsr     U_Command_write ; command_write
    bset    send_flg
    mov.w   #3,start_cnt
    mov.w   #258,send_cnt
    jmp     U_Flash_func_end    ; jump  Flash_func_end
;
;-----------------------------------------------------------
;+     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_1:
    add.w   #1,r3           ; r3 +1 increment
    mov.w   r3,a0           ; r3 --> a0
    mov.b   data[a0],r0l
    mov.b   r0l,addr_l[a0]  ; Store address
    cmp.w   #259,r3         ; r3 = 258 ?
    jltu    U_Program_loop_1  ; jump U_Program_loop at r3<258
```

```
;
    mov.w   #00ffh,r2       ; Read array command
    jsr     U_Command_write ; command_write
    mov.w   #0070h,r2       ; Read SRD command
    jsr     U_Command_write ; command_write
    mov.w   [a1],r1         ; SRD read
    mov.w   #00ffh,r2       ; Read array command
    jsr     U_Command_write ; command_write
    cmp.b   #80h,r1l        ; error check
    jne     U_Program_end
;
    mov.w   #0041h,r2       ; Page program command
    jsr     U_Command_write ; command_write
    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
    mov.w   r1,[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
    bclr    send_flg
    mov.w   #0,send_cnt
    mov.w   #0,start_cnt
    jmp     U_Flash_func_end    ; jump Flash_func_end
;

;----------------------------------------------------------
;+     Block erase - UART mode -                          +
;----------------------------------------------------------
U_Erase:
    mov.w   #1,r3           ; receive number (r3=1)
    mov.b   #0feh,addr_l    ; addr_l = feh
U_Erase_loop:
    mov.w   r3,a0           ; r3 --> a0
    mov.b   data[a0],r0l
    mov.b   r0l,addr_l[a0]  ; Store address
    add.w   #1,r3           ; r3 +1 increment
    cmp.w   #4,r3           ; r3=4 ?
    jltu    U_Erase_loop    ; jump  at r3<4
    cmp.b   #0d0h,data      ; Confirm command check
    jne     U_Erase_end     ; jump Erase_end at Confirm command error
    mov.w   #0020h,r2       ; Erase command
    jsr     U_Command_write ; command write
    mov.w   #00d0h,r2       ; Confirm command
    mov.w   r2,[a1]         ; command write
U_Erase_end:
    mov.w   #00ffh,r2       ; Read array command
    jsr     U_Command_write ; U_command_write
    bclr    send_flg
    mov.w   #0,send_cnt
    mov.w   #0,start_cnt
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
```

```
;-----------------------------------------------------------
;+      All erase ( unlock block ) - UART mode -          +
;-----------------------------------------------------------
U_All_erase:
    mov.w   #1,a0
    mov.b   data[a0],r0l    ;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
    mov.w   #0000h,addr_l   ; 0fe0000h --> addr
    mov.b   #00feh,addr_h
    mov.w   #00a7h,r2       ; All erase command
    jsr     U_Command_write ; command write
    mov.w   #00d0h,r2       ; Confirm command
    mov.w   r2,[a1]         ; command write
U_All_erase_end:
    mov.w   #00ffh,r2       ; Read array command
    jsr     U_Command_write ; command_write
    bclr    send_flg
    mov.w   #0,send_cnt
    mov.w   #0,start_cnt
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-----------------------------------------------------------
;+     Read SRD - UART mode                               +
;-----------------------------------------------------------
U_Read_SRD:
    mov.w   #0,r3           ; receive number (r3=0)
    mov.w   #0000h,addr_l   ; 0fe0000h --> addr
    mov.b   #00feh,addr_h
    mov.w   #00ffh,r2       ; Read array command
    jsr     U_Command_write ; command_write
    mov.w   #0070h,r2       ; Read SRD command
    jsr     U_Command_write ; command write
    mov.w   [a1],r1         ; SRD read
    mov.w   #00ffh,r2       ; Read array command
    jsr     U_Command_write ; command_write
    mov.w   #1,start_cnt
    mov.w   #3,send_cnt
    bset    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-----------------------------------------------------------
;+     Clear SRD - UART mode                              +
;-----------------------------------------------------------
U_Clear_SRD:
    mov.w   #0000h,addr_l   ; 0fe0000h --> addr
    mov.b   #00feh,addr_h
    mov.w   #00ffh,r2       ; Read array command
    jsr     U_Command_write ; command write
    mov.w   #0050h,r2       ; Clear SRD command
    jsr     U_Command_write ; command write
    mov.w   #00ffh,r2       ; Read array command
    jsr     U_Command_write ; command write
    and.b   #10011100b,SRD1 ; SRD1 clear
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-----------------------------------------------------------
;+     Read Lock Bit - UART mode -                        +
;-----------------------------------------------------------
U_Read_LB:
    mov.w   #1,r3           ; receive number (r3=1)
    mov.b   #0feh,addr_l    ; addr_l = feh
```

```
U_Read_LB_loop:
    mov.w   r3,a0           ; r3 --> a0
    mov.b   data[a0],r0l
    mov.b   r0l,addr_l[a0]  ; Store address
    add.w   #1,r3           ; r3 +1 increment
    cmp.w   #3,r3           ; r3=3 ?
    jltu    U_Read_LB_loop  ; jump at r3<3
    bclr    re_u1c1         ; Reception disabled
    mov.w   #0071h,r2       ; Read LB command
    jsr     U_Command_write   ; command write
    mov.w   [a1],r1         ; read LB
    mov.w   #00ffh,r2       ; Read array command
    jsr     U_Command_write ; U_command write
U_Read_LB_end:
    mov.w   #1,start_cnt
    mov.w   #1,send_cnt
    bset    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;----------------------------------------------------------
;+     Program Lock Bit - UART mode -                     +
;----------------------------------------------------------
U_Program_LB:
    mov.w   #1,r3           ; receive number (r3=1)
    mov.b   #0feh,addr_l    ; addr_l = feh
U_Program_LB_loop:
    mov.w   r3,a0           ; r3 --> a0
    mov.b   data[a0],r0l
    mov.b   r0l,addr_l[a0]  ; Store address
    add.w   #1,r3           ; r3 +1 increment
    cmp.w   #4,r3           ; r3=4 ?
    jltu    U_Program_LB_loop  ; jump at r3<4
    cmp.b   #0d0h,data      ; Confirm command check
    jne     U_Program_LB_end ; jump U_Program_LB_end at Confirm command error
    mov.w   #0077h,r2       ; Program LB command
    jsr     U_Command_write ; command write
    mov.w   #00d0h,r2       ; Confirm command
    mov.w   r2,[a1]         ; command write
    mov.w   #00ffh,r2       ; Read array command
    jsr     U_Command_write ; command write
U_Program_LB_end:
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;----------------------------------------------------------
;+     Lock Bit enable - UART mode -                      +
;----------------------------------------------------------
U_LB_enable:
    bclr    fmr02            ; Lock disable bit = 0
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;
;----------------------------------------------------------
;+     Lock Bit disable - UART mode -                     +
;----------------------------------------------------------
U_LB_disable:
    bclr    fmr02            ; Lock disable bit = 0
    bset    fmr02            ; Lock disable Bit = 1
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-----------------------------------------------------------
;+     ID check - UART mode                               +
;-----------------------------------------------------------
U_ID_check:
    btst    blank           ; blank flag check
    jc      U_ID_check_end  ; jump U_ID_check_end at blank
    cmp.w   #0ffdfh,addr_l  ; lower U_ID address check
    jne     U_ID_error      ; jump U_ID_error at ID address error
    cmp.w   #007ffh,addr_h  ; higher ID address check
    jne     U_ID_error      ; jump U_ID_error at ID address error
U_ID_data_check:
    mov.w   #0ffdfh,r1      ; ID lower address --> r1
    mov.w   #1,r3           ; check loop number (r3=1)
U_ID_check_loop:
    mov.w   #00ffh,a1       ; ID higher address --> a1
    sha.l   #16,a1
    mov.w   r1,a0           ; r1 --> a0
    add.l   a0,a1
    mov.b   [a1],r0l        ; ID data read from Flash memory
    mov.w   r3,a0           ; r3 --> a0
    cmp.b   r0l,data[a0]    ; compare ID data
    jne     U_ID_error      ; jump U_ID_error at ID error
    add.w   #4,r1           ; r1 +4 increment (next ID address)
    cmp.w   #0ffe7h,r1      ; r1=0ffefh ?
    jne     ?+              ; jump ? at not equal
    mov.w   #0ffebh,r1      ; r1=0ffeb at equal
?:
    add.w   #1,r3           ; r3 +1 increment
    cmp.w   #8,r3           ; r3=8 ?
    jltu    U_ID_check_loop ; jump U_ID_check_loop at r3<8
U_ID_OK:
    bset    sr10
    bset    sr11            ; ID check OK (sr11=1,sr10=1)
    jmp     U_ID_check_end  ; jump  U_ID_check_end
U_ID_error:
    bset    sr10
    bclr    sr11            ; ID error (sr11=0,sr10=1)
U_ID_check_end:
    mov.w   #0,start_cnt
    mov.w   #0,send_cnt
    bclr    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-----------------------------------------------------------
;+     Boot output - UART mode -                           +
;-----------------------------------------------------------
U_Boot_output:
    bclr    fmr05            ; Boot ROM select
    mov.w   #0,r3           ; receive number (r3=1)
    mov.w   #0,addr_l       ; addr_l = 0
U_Boot_loop:
    add.w   #1,r3           ; r3 +1 increment
    mov.w   r3,a0           ; r3 --> a0
    mov.w   data[a0],r0
    mov.b   r0l,addr_l[a0]  ; Store address
    cmp.w   #2,r3           ; r3 = 3 ?
    jltu    U_Boot_loop     ; jump at r3<3
    bset    send_flg
    mov.w   #3,start_cnt
    mov.w   #258,send_cnt
    jmp     U_Flash_func_end    ; jump  Flash_func_end
;
```

```
;-----------------------------------------------------------
;+     Read check                                          +
;-----------------------------------------------------------
U_Read_check:
    mov.w   #0,start_cnt
    mov.w   #2,send_cnt
    bset    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;-----------------------------------------------------------
;+     Download - UART mode -                               +
;-----------------------------------------------------------
U_Download:
    bclr    fmr05            ; Boot ROM select
    jmp.a   U_Download_program    ; jump U_Download_program
;
;-----------------------------------------------------------
;+     Version output - UART mode -                        +
;-----------------------------------------------------------
U_Ver_output:
    mov.w   #0,start_cnt
    mov.w   #8,send_cnt
    bset    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : Command write  - UART mode            +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_Command_write:
    btst    fmr00            ; RY/BY status check
    jz      U_Command_write
    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
    mov.w   r2,[a1]       ; command write
    rts
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     Main init first   - UART mode -                            +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_SIO_init_first:
    bclr    freq_set1       ; freq set flag clear
    bclr    freq_set2
    mov.b   #01111111b,data_BPS ; Initialize Baud rate
;    mov.b   #129,data_BPS ; Initialize Baud rate 9600bps for 20MHz
    jsr     Initialize_3    ; UART mode Initialize
    mov.b   #01000000b,r1l  ; counbter1,2 reset
    mov.b   #10000000b,r1h

    jsr     U_SIO_rcv
;
    mov.w   rcv_d,r0        ; receive data --> r0
    btst    freq_set2
    jz      U_Freq_check
    jmp     U_Loop_main
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     SIO init  - UART mode -                             +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_SIO_freq:
    btst   freq_set2
    jz     U_Freq_check
    jmp    U_Flash_init
```

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+     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_BPS,r1l        ; Baud = Baud xor r1l
    mov.b   r1l,data_BPS        ; 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_BPS,r1h
    mov.b   r1h,data_BPS
U_Freq_check_5:
    mov.b   data_BPS,data_BPS+1     ; Min Baud --> data+1
    mov.b   #01000000b,r1l  ; counter reset
    mov.b   #10000000b,r1h
    mov.b   #10000000b,data_BPS ; 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_BPS,r1h
    mov.b   r1h,data_BPS
U_Freq_set_2:
    bset    freq_set2
    mov.b   data_BPS,r1l        ; Max Baud --> data
    sub.b   data_BPS+1,r1l
    shl.b   #-1,r1l
    add.b   data_BPS+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_BPS
    mov.b   #0b0h,r0l       ; "B0h" set
    jsr     U_blank_end     ; UART mode Initialize
    jmp     U_BPS_SET_data
;
;---------------------------------------
;+     Baud rate change - UART mode   +
;---------------------------------------
U_BPS_B0:
U_BPS_B1:
U_BPS_B2:
U_BPS_B3:
U_BPS_B4:
    mov.w   #0,start_cnt
    mov.w   #1,send_cnt
    bset    send_flg
    jmp     U_Flash_func_end    ; jump Flash_func_end
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : serial I/O send    - UART mode       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_SIO_send:
    bclr    re_u1c1
    bset    te_u1c1
    mov.b   send_d,u1tb         ; transmit buffer register
?:
    btst    ti_u1c1             ; transmit buffer empty?
    jnc     ?-
    rts
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : serial I/O send    - UART mode       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_SIO_send_only:
    mov.b   send_d,u1tb         ; transmit buffer register
?:
    btst    ti_u1c1             ; transmit buffer empty?
    jnc     ?-
    rts
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : serial I/O receive - UART mode       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_SIO_rcv:
    bclr    te_u1c1
    bset    re_u1c1
?:
    btst    ri_u1c1             ; receive complete?
    jnc     ?-
    mov.w   u1rb,rcv_d
    rts
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : serial I/O receive - UART mode       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_SIO_rcv_only:
?:
    btst    ri_u1c1             ; receive complete?
    jnc     ?-
    mov.w   u1rb,rcv_d
    rts
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;+      Subroutine : serial I/O receive - UART mode       +
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
U_SIO_exit:
    btst    txept_u1c0
    jnc     U_SIO_exit
    rts
;

;===========================================================
;+     Vector Table                                        +
;===========================================================
    .section    inter,romdata
    .org        Vector
;
    .lword      Reset       ; UNO
    .lword      Reset       ; INTO
    .lword      Reset       ; BRK
    .lword      Reset       ; address matchnig
    .lword      Reset       ;
    .lword      Reset       ; WDT
    .lword      Reset       ;
    .lword      Reset       ; NMI
    .lword      Reset       ; Reset
;
    .end
```

Header

```
;*******************************************************
;*                                                    *
;*  file name   : definition of M16C/80 Flash         *
;*                                                    *
;*  Version     : 0.04 ( 1999- 4-28 )                 *
;*                for Boot Ver.1.00                   *
;*******************************************************
;
;------------------------------------------------------
;   BUSY output
;------------------------------------------------------
busy        .btequ  4,03C0h     ; p6_4
busy_d      .btequ  4,03C2h     ; pd6_4
;
;------------------------------------------------------
;   Serial I/O select bit
;------------------------------------------------------
s_mode      .btequ  5,03C0h     ; p6_5
s_mode_d    .btequ  5,03C2h     ; pd6_5
;
;------------------------------------------------------
;    define of symbols
;------------------------------------------------------
Ram_TOP     .equ    000400h
Ram_END     .equ    000bffh
Istack      .equ    000c00h
;
Version     .equ    0ffe000h
Boot_TOP    .equ    0ffe020h
Trans_TOP1  .equ    0ffe200h
Trans_END1  .equ    0ffe700h
Trans_TOP2  .equ    0ffe800h
Trans_END2  .equ    0ffed80h
Vector      .equ    0ffffdch
;
Download_program    .equ    0ffe0f0h
U_Download_program  .equ    0ffe170h
;
SB_base     .equ    000400h
Ram_progTOP .equ    000600h
Ram_progEND .equ    000B80h
;
    .section    memory,data
    .org        Ram_TOP
;
SRD:        .blkb   1
SRD1:       .blkb   1
ver:        .blkb   10
SF:         .blkb   1
unuse:      .blkb   4
addr_l:     .blkb   1
addr_m:     .blkb   1
addr_h:     .blkb   1
data:       .blkb   300
buff:       .blkb   20
ID_err:     .blkb   1
sum:        .blkb   2
baud:       .blkb   5
;
sr0         .btequ  0,SRD
sr1         .btequ  1,SRD
```

```
sr2         .btequ  2,SRD
sr3         .btequ  3,SRD
sr4         .btequ  4,SRD
sr5         .btequ  5,SRD
sr6         .btequ  6,SRD
sr7         .btequ  7,SRD
sr8         .btequ  0,SRD1
sr9         .btequ  1,SRD1
sr10        .btequ  2,SRD1
sr11        .btequ  3,SRD1
sr12        .btequ  4,SRD1
sr13        .btequ  5,SRD1
sr14        .btequ  6,SRD1
sr15        .btequ  7,SRD1
;
ram_check   .btequ  0,SF
blank       .btequ  1,SF
old_mode    .btequ  2,SF
freq_set1   .btequ  3,SF
freq_set2   .btequ  4,SF
;
;
```

4.4 Precautions

This section describes precautions to be observed when controlling the M16C/80's internal flash memory.

When Powering On/Off

When powering on/off, pay attention to the following:

(1) Be careful that noise will not get into the control pins (WE, CE, OE). If a noise pulse is applied to the control pins when turning the power on or off, a program/erase error will occur, which in the worst case may destroy the memory data.

(2) A finite wait time is required before you can start read or program/erase operation after power-on. Specifically, a wait time of 2 µs is required before read or program/erase operation can be started after Vcc reached Vccmin (3.0 V).

Chapter 5

Internal Flash Memory Rewrite Inhibit Function

5.1 ID Code

To prevent illegal leakage of a program, the M16C flash memory allows ID data to be set (called the "ID code"). This section describes how to inhibit the internal flash memory against rewriting by using ID data.

What Is The ID Code?

The ID code is the ID data that is written into the internal flash memory beforehand in order to inhibit the flash memory against rewriting.
When exercising control, enter the ID from the programmer newly again and only when it matches the ID data stored in the flash memory, you can control program or read operation.
The ID code for the M16C/20 and 62's flash memory is fixed to 7 bytes in length. The areas in which to store the ID code are address $FFFDF_{16}$, address $FFFE3_{16}$, address $FFFEB_{16}$, address $FFFEF_{16}$, address $FFFF3_{16}$, address $FFFF7_{16}$, and address $FFFFB_{16}$.
The ID code for the M16C/80's flash memory is fixed to 7 bytes in length. The areas in which to store the ID code are address $FFFFDF_{16}$, address $FFFFE3_{16}$, address $FFFFEB_{16}$, address $FFFFEF_{16}$, address $FFFFF3_{16}$, address $FFFFF7_{16}$, and address $FFFFFB_{16}$.

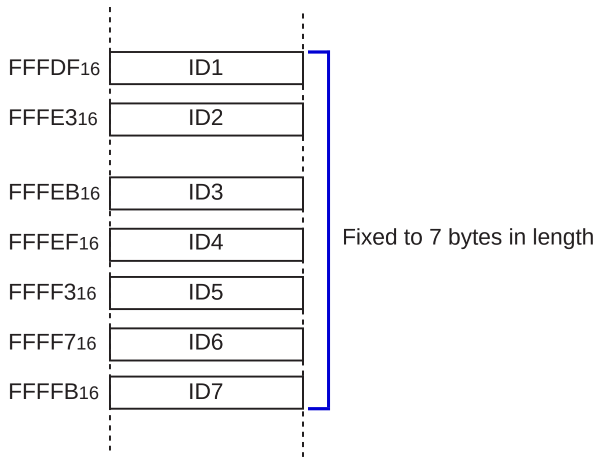


Figure 5.1.1 ID Codes of The M16C/20 and 62

Processing Flow

Figure 5.1.2 shows a flow of the main program using ID code.

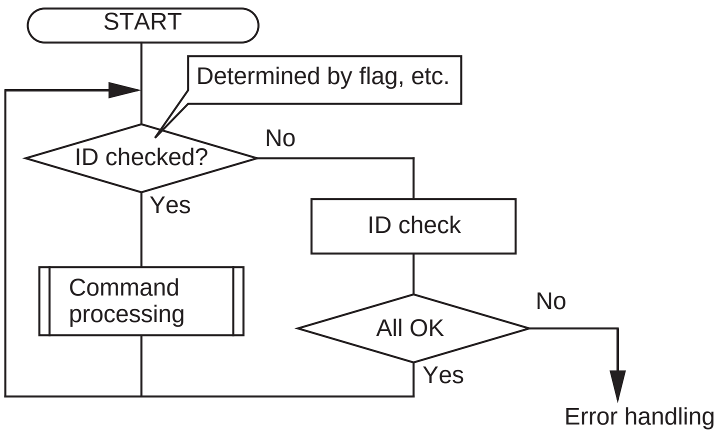
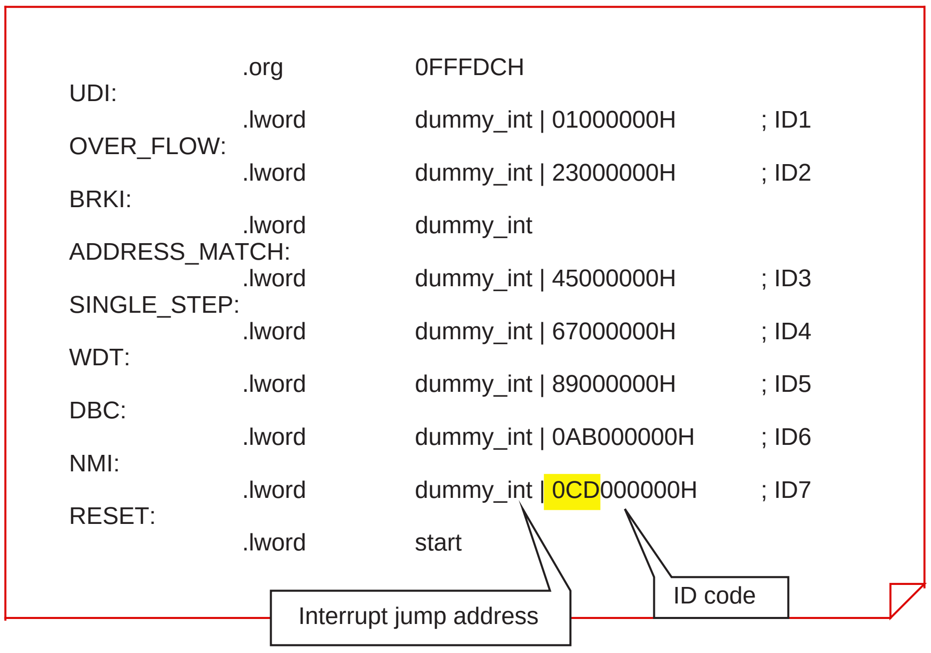


Figure 5.1.2 Flow of The Main Program Using ID Code

How to Set ID Code

The addresses at which ID code is set overlaps the fixed vector area. Therefore, set the logical sum of each interrupt's jump address and the ID code as fixed vector. A description example is shown in Figure 5.1.3.



```
                  .org         0FFFDCH
UDI:
                  .lword       dummy_int | 01000000H      ; ID1
OVER_FLOW:
                  .lword       dummy_int | 23000000H      ; ID2
BRKI:
                  .lword       dummy_int
ADDRESS_MATCH:
                  .lword       dummy_int | 45000000H      ; ID3
SINGLE_STEP:
                  .lword       dummy_int | 67000000H      ; ID4
WDT:
                  .lword       dummy_int | 89000000H      ; ID5
DBC:
                  .lword       dummy_int | 0AB000000H     ; ID6
NMI:
                  .lword       dummy_int | 0CD000000H     ; ID7
RESET:
                  .lword       start
```

Interrupt jump address
ID code

Figure 5.1.3 Description Example for ID Code Setting (M16C/20, 62)

Setting ID Code by lmc30, lmc308

The load module converters (lmc30, lmc308) included with the assemblers for Mitsubishi M16C (AS30, AS308) allow any ID code to be set in a load module when generating the load module.
The following shows an example of command settings necessary to set ID code when generating load modules. For details about the load module converters (lmc30, lmc308), refer to the AS30 User's Manual or AS308 User's Manual.

Example 1: lmc30 -ID Code No1 sample ("CodeNo1" specified using ASCII code)
ID code : 436F64654E6F31

Table 5.1.1 ID Code Setting Example

	ID1	ID2	ID3	ID4	ID5	ID6	ID7
Address	$FFFDF_{16}$	$FFFE3_{16}$	$FFFEB_{16}$	$FFFEF_{16}$	$FFFF3_{16}$	$FFFF7_{16}$	$FFFFB_{16}$
Data	43_{16}	$6F_{16}$	64_{16}	65_{16}	$4E_{16}$	$6F_{16}$	31_{16}

Example 2: lmc30 -ID Code sample ("Code" specified using ASCII code)
ID code : 436F6465000000

Example 3: lmc30 -ID1234567 sample ("1234567" specified using ASCII code)
ID code : 31323334353637

Example 4: lmc30 -ID#49562137856132 sample ("49562137856132" specified using HEX code)
ID code : 49562137856132

Example 5: lmc30 -ID1234567 sample ("1234567" specified using HEX code)
ID code : 12345670000000

Example 6: lmc30 -ID sample
ID code : FFFFFFFFFFFFFF

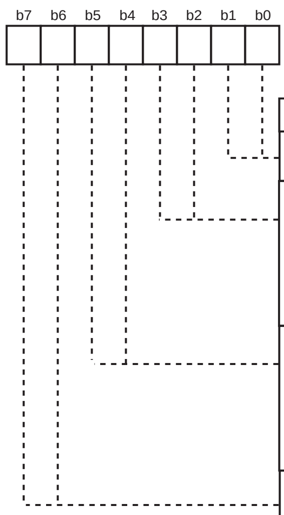
5.2 ROM Code Protect Function

To prevent illegal leakage of a program, the M16C flash memory allows you to limit rewriting of ROM code (called the "ROM code protect"). This section describes how to limit rewriting of the internal flash memory by using the ROM code protect function.

What is The ROM Code Protect Function?

The ROM code protect function reading out or modifying the contents of the flash memory version by using the ROM code protect control address ($0FFFFF_{16}$) during parallel I/O mode. Figure 5.2.1 shows the ROM code protect control address ($0FFFFF_{16}$). (This address exists in the user ROM area.)
If one of the pair of ROM code protect bits is set to 0, ROM code protect is turned on, so that the contents of the flash memory version are protected against readout and modification. ROM code protect is implemented in two levels. If level 2 is selected, the flash memory is protected even against readout by a shipment inspection LSI tester, etc. When an attempt is made to select both level 1 and level 2, level 2 is selected by default.
If both of the two ROM code protect reset bits are set to “00,” ROM code protect is turned off, so that the contents of the flash memory version can be read out or modified. Once ROM code protect is turned on, the contents of the ROM code protect reset bits cannot be modified in parallel I/O mode. Use the serial I/O or some other mode to rewrite the contents of the ROM code protect reset bits.

ROM code protect control address



Symbol	Address	When reset
ROMCP	$0FFFFF_{16}$ (M16C/20, 60) $FFFFFF_{16}$ (M16C/80)	FF_{16}

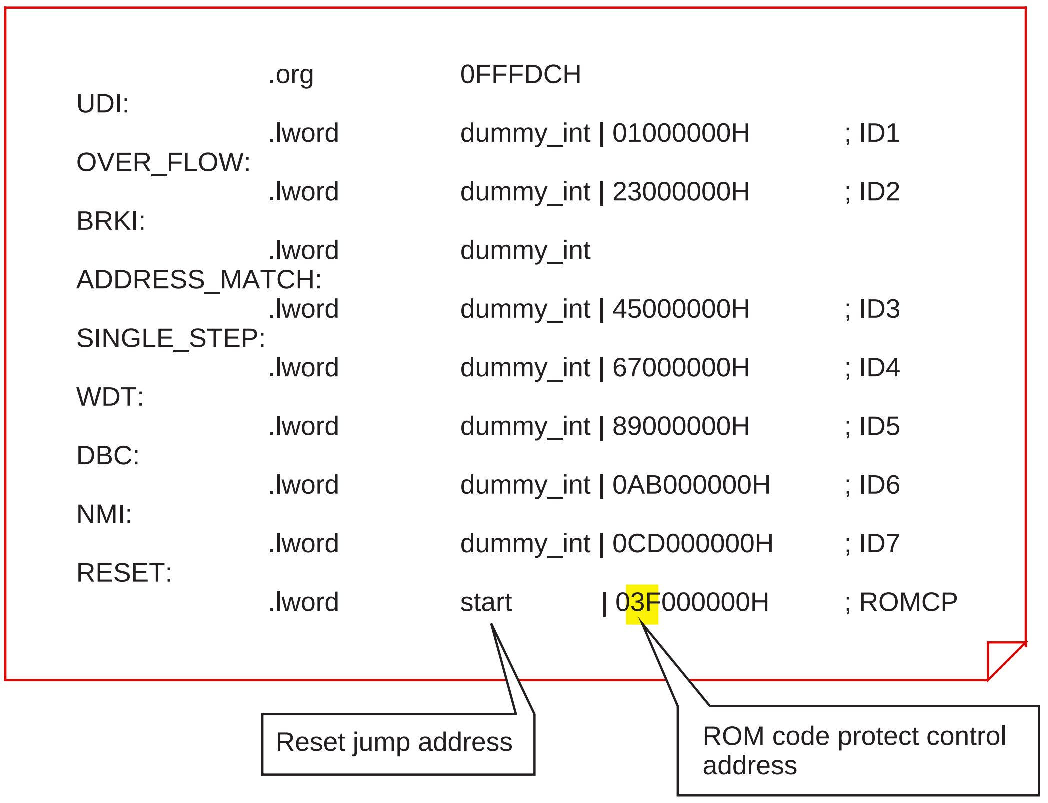
Bit symbol	Bit name	Function
Reserved bit		Always set this bit to 1.
ROMCP2	ROM code protect level 2 set bit (Note 1, 2)	b3 b2 00: Protect enabled 01: Protect enabled 10: Protect enabled 11: Protect disabled
ROMCR	ROM code protect reset bit (Note 3)	b5 b4 00: Protect removed 01: Protect set bit effective 10: Protect set bit effective 11: Protect set bit effective
ROMCP1	ROM code protect level 1 set bit (Note 1)	b7 b6 00: Protect enabled 01: Protect enabled 10: Protect enabled 11: Protect disabled

Note 1: When ROM code protect is turned on, the on-chip flash memory is protected against readout or modification in parallel input/output mode.
Note 2: When ROM code protect level 2 is turned on, ROM code readout by a shipment inspection LSI tester, etc. also is inhibited.
Note 3: The ROM code protect reset bits can be used to turn off ROM code protect level 1 and ROM code protect level 2. However, since these bits cannot be changed in parallel input/output mode, they need to be rewritten in serial input/output or some other mode.
Note 4: This bit is defined as the reserved bit in M16C/20 group. Must set "1" to it.

Figure 5.2.1 ROM Code Protect Control Address

How to Set ROM Code Protect Control Addresses

The addresses at which ROM code protect is set overlaps the fixed vector area. Therefore, set the logical sum of the reset jump address and the set value of ROM code protect bit as fixed vector. A description example is shown in Figure 5.2.2.



```
                .org            0FFFDCH
UDI:
                .lword          dummy_int | 01000000H       ; ID1
OVER_FLOW:
                .lword          dummy_int | 23000000H       ; ID2
BRKI:
                .lword          dummy_int
ADDRESS_MATCH:
                .lword          dummy_int | 45000000H       ; ID3
SINGLE_STEP:
                .lword          dummy_int | 67000000H       ; ID4
WDT:
                .lword          dummy_int | 89000000H       ; ID5
DBC:
                .lword          dummy_int | 0AB000000H      ; ID6
NMI:
                .lword          dummy_int | 0CD000000H      ; ID7
RESET:
                .lword          start     | 03F000000H      ; ROMCP
```

Figure 5.2.2 Description Example for ROM Code Protect Control Address (M16C/20,62)

Setting ROM Code Protect by lmc30, lmc308

The load module converters (lmc30, lmc308) included with the assemblers for Mitsubishi M16C (AS30, AS308) allow any ROM code protect function to be set in a load module when generating the load module. The following shows an example of command settings necessary to set ROM code protect function when generating load modules. For details about the load module converters (lmc30, lmc308), refer to the AS30 User's Manual or AS308 User's Manual.

Example 1: lmc30 -protect 1 sample (Set ROM code protect function level 1)
Protect code : $3F_{16}$

Example 2: lmc30 -protect 2 sample (Set ROM code protect function level 2)
Protect code : $F3_{16}$

Example 3: lmc30 sample (Without ROM code protect function setting)
Protect code : Data described in source program

MITSUBISHI SINGLE-CHIP MICROCOMPUTERS
M16C/60, M16C/20, M16C/80 Series
Application Note <Flash Memory>

December First Edition 1999
Editioned by
Committee of editing of Mitsubishi Semiconductor
Published by
Mitsubishi Electric Corp., Kitaitami Works