NEC

Preliminary Application Note

78K/0, 78K/0S Series

8-Bit Single-Chip Microcontrollers

Flash Memory Write

For entire 78K/0 Series
For entire 78K/0S Series

Document No. U14458EJ1V0AN00 (1st edition)
Date Published November 1999 N CP(K)

[MEMO]

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.l.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 91-504-2787
Fax: 91-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
United Square, Singapore 1130
Tel: 65-253-8311
Fax: 65-250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

NEC do Brasil S.A.
Electron Devices Division
Rodovia Presidente Dutra, Km 214
07210-902-Guarulhos-SP Brasil
Tel: 55-11-6465-6810
Fax: 55-11-6465-6829

J99.1

[MEMO]

INTRODUCTION

Target Readers	These application notes are intended for users who wish to understand the functions of the 78K/0 and 78K/0S Series products and concerns the flash programmer used to write programs in flash memory versions.
Purpose	These application notes are intended for users to understand how to write to 78K/0, 78K/0S Series flash memory versions by providing application sample programs for these products.
How to Use This Manual	In these application notes, it is assumed that the reader has general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.

- To know the 78K/0 Series instruction function in detail:
  → See **78K/0 SERIES USER'S MANUAL INSTRUCTIONS (U12326E)**
- To know the 78K/0S Series instruction function in detail:
  → See **78K/0S SERIES USER'S MANUAL INSTRUCTIONS (U11047E)**

Conventions

Data significance:	Higher digits on the left and lower on the right
Active low representation:	$\overline{\times\times\times}$ (over score over pin or signal name)
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numeric representation:	Binary… ×××× or ××××B
	Decimal… ××××
	Hexadecimal… ××××H

Related documents The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

78K/0 Series

Documents Related to Development Tools (User's Manual)

Document Name		Document No.	
		English	Japanese
RA78K0 Assembler Package	Operation	U11802E	U11802J
	Assembly Language	U11801E	U11801J
	Structured Assembly language	U11789E	U11789J
RA78K Series Structured Assembler Preprocessor		U12323E	U12323J
CC78K0 C Compiler	for Operation	U11517E	U11517J
	for Language	U11518E	U11518J
CC78K0 C Compiler Application Notes	Programming Know-How	U13034E	U13034J
IE-78K0-NS		U13731E	U13731J
IE-78001-R-A		To be prepared	To be prepared
IE-78K0-R-EX1		To be prepared	To be prepared
SM78K0 System Simulator Windows™ Based	Reference	U10181E	U10181J
SM78K Series System Simulator	External Part User Open Interface Specifications	U10092E	U10092J
ID78K0-NS Integrated Debugger Windows Based	Reference	U12900E	U12900J
ID78K0 Integrated Debugger EWS Based	Reference	U11151E	U11151J
ID78K0 Integrated Debugger PC Based	Reference	U11539E	U11539J
ID78K0 Integrated Debugger Windows Based	Guide	U11649E	U11649J

Documents Related to Embedded Software (User's Manual)

Document Name		Document No.	
		English	Japanese
78K/0 Series Real-Time OS	Fundamentals	U11537E	U11537J
	Installation	U11536E	U11536J
78K/0 Series OS MX78K0	Fundamental	U12257E	U12257J

Documents Related to Development Tools (User's Manual)

Document Name		Document No.	
		English	Japanese
RA78K0S Assembler Package	Operation	U11622E	U11622J
	Assembly Language	U11599E	U11599J
	Structured Assembly language	U11623E	U11623J
CC78K0S C Compiler	Operation	U11816E	U11816J
	Language	U11817E	U11817J
SM78K0S System Simulator Windows Based	Reference	U11489E	U11489J
SM78K Series System Simulator	External Part User Open Interface Specifications	U10092E	U10092J
ID78K0S-NS Integrated Debugger Windows Based	Reference	U12901E	U12901J
IE-78K0S-NS In-circuit Emulator		U13549E	U13549J

Documents Related to Embedded Software (User's Manual)

Document Name		Document No.	
		English	Japanese
78K/0S Series OS MX78K0S	Basics	U12938E	U12938J

Common to 78K/0 and 78K/0S Series

Other Documents

Document Name	Document No.	
	English	Japanese
SEMICONDUCTORS SELECTION GUIDE Products & Packages (CD-ROM)	X13769X	
Semiconductor Device Mounting Technology Manual	C10535E	C10535J
Quality Grades on NEC Semiconductor Devices	C11531E	C11531J
NEC Semiconductor Device Reliability/Quality Control System	C10983E	C10983J
Guide to Prevent Damage for Semiconductor Devices by Electrostatic Discharge (ESD)	C11892E	C11892J
Guide to Microcomputer-Related Products by Third Party	U11416E	U11416J

Caution The above documents are subject to change without prior notice. be sure to use the latest document for designing.

[MEMO]

CONTENTS

LIST OF FIGURES (1/2)

LIST OF FIGURES (2/2)

LIST OF TABLES

CHAPTER 1 GENERAL

These application notes describe how to create a flash memory write tool (called a "flash programmer") for 78K/0 and 78K/0S Series microcontrollers that feature on-chip flash memory (below, these microcontrollers are called "flash microcontrollers").

To write to a flash microcontroller, certain commands must be executed for the flash microcontroller in a certain predetermined order. See **CHAPTER 2 BASIC FORMAT** for a description of the flash microcontroller control commands used to write to flash memory.

3-wire serial communications, IIC communications, UART communications, or pseudo 3-wire serial communications can be selected as the communication method for transmitting control commands and write data to a flash microcontroller. See **CHAPTER 3 WRITE SEQUENCE** for a description of the flash microcontroller communication timing and write sequence for each communication method.

Also, see **CHAPTER 4 SAMPLE PROGRAMS** and **CHAPTER 5 INTERFACE EXAMPLES** for a description of sample programs that write to flash memory.

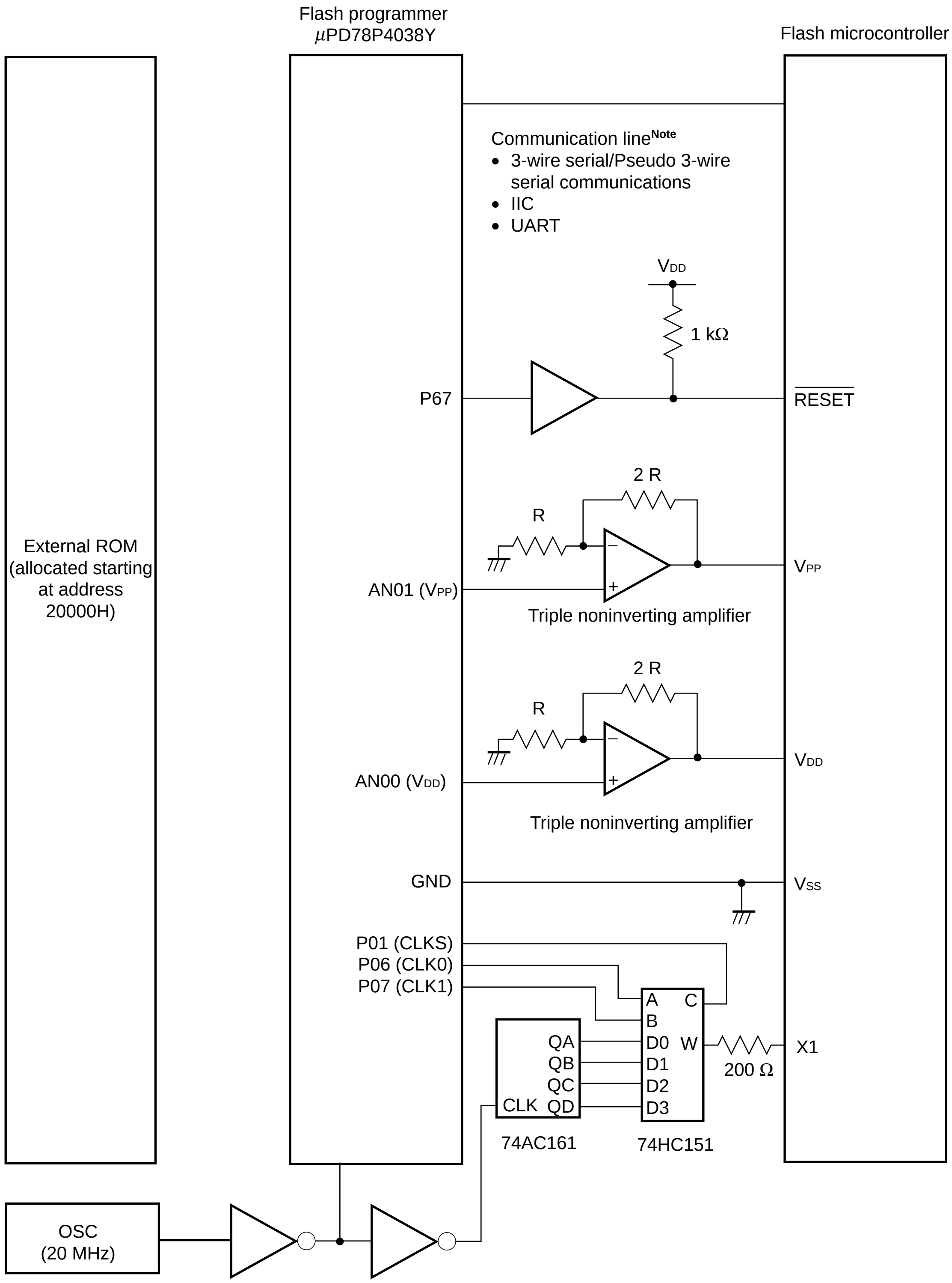
1.1 System Configuration

A μPD78P4038Y is used as the control chip for the flash programmer. Write data that is sent to the flash microcontroller is allocated and stored in external ROM starting at address 20000H in the μPD78P4038Y's external memory space. Data that has been stored in external ROM is transferred to the flash microcontroller when the flash microcontroller is accessed for write and verify operations.

The flash programmer supplies the V_{DD} and V_{PP} voltage and the operating clock for the flash microcontroller.

Figures 1-1 and 1-2 illustrate the flash programmer's system configuration. In Figure 1-2, the pins that are used (for communications) vary depending on the communication method. Table 1-1 lists the correspondences of communication methods and used pins.

Figure 1-1. System Configuration Diagram



Note See **Figure 1-2. Communication Line Connection Diagram** for an illustration of the communication line connections.

Figure 1-2. Communication Line Connection Diagram

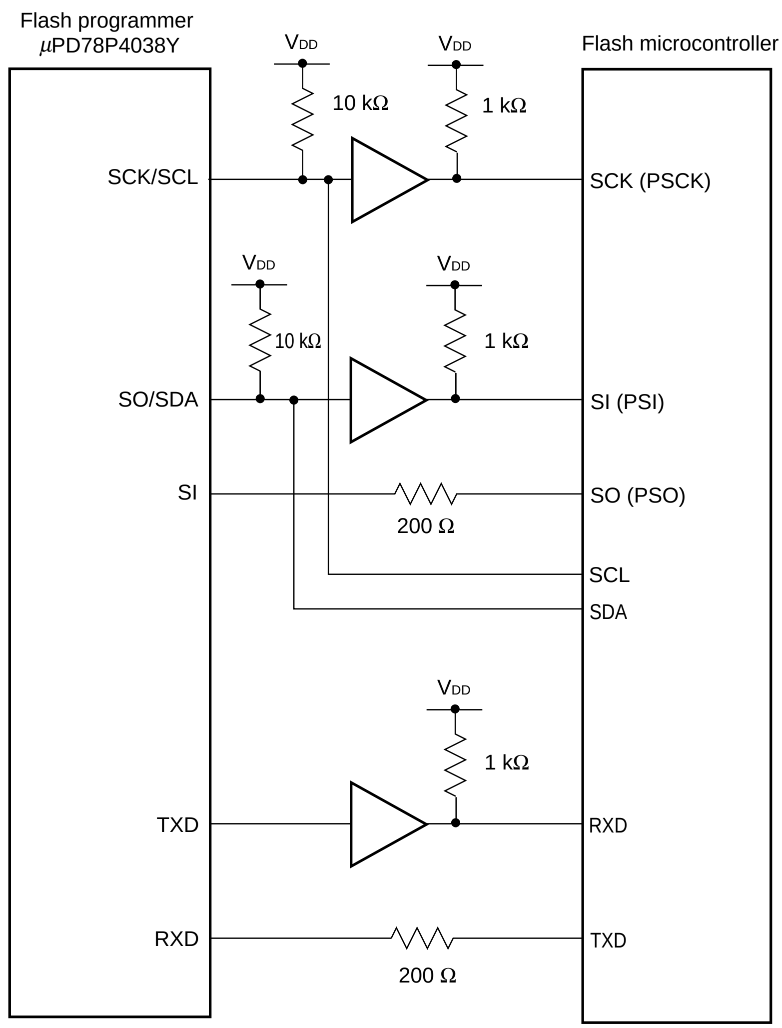


Table 1-1. Communication Line Connections

Communication Method	Pins Used for Communications	
	Flash Programmer Side	Flash Microcontroller Side
3-wire serial/Pseudo 3-wire serial communications	SCK	SCK/PSCK[Note]
	SO	SI/PSI[Note]
	SI	SO/PSO[Note]
IIC	SCL	SCL
	SDA	SDA
UART	TXD	RXD
	RXD	TXD

Note PSCK, PSI, PSO: Ports used for the flash microcontroller's pseudo 3-wire serial communications.

1.2 Differences between the μPD78F0xxx and the μPD78F9xxx

The differences between writing to the 78K/0 Series flash microcontroller (μPD78F0xxx) and writing to the 78K/0S Series flash microcontroller (μPD78F9xxx) are listed below.

- Size of write data transmitted using high-speed or continuous write command
  (Size range is 1 to 256 bytes for μPD78F0xxx and 1 to 128 bytes for μPD78F9xxx)
- Size of one verify data transfer using verify command
  (Size is 256 bytes for μPD78F0xxx and 128 bytes for μPD78F9xxx)
- Number of wait clocks used to adjust communication timing for each communication method (3-wire serial communications, IIC communications, UART communications, or pseudo 3-wire serial communications)
- Number of wait clocks for flash microcontroller's internal processing when executing various commands.

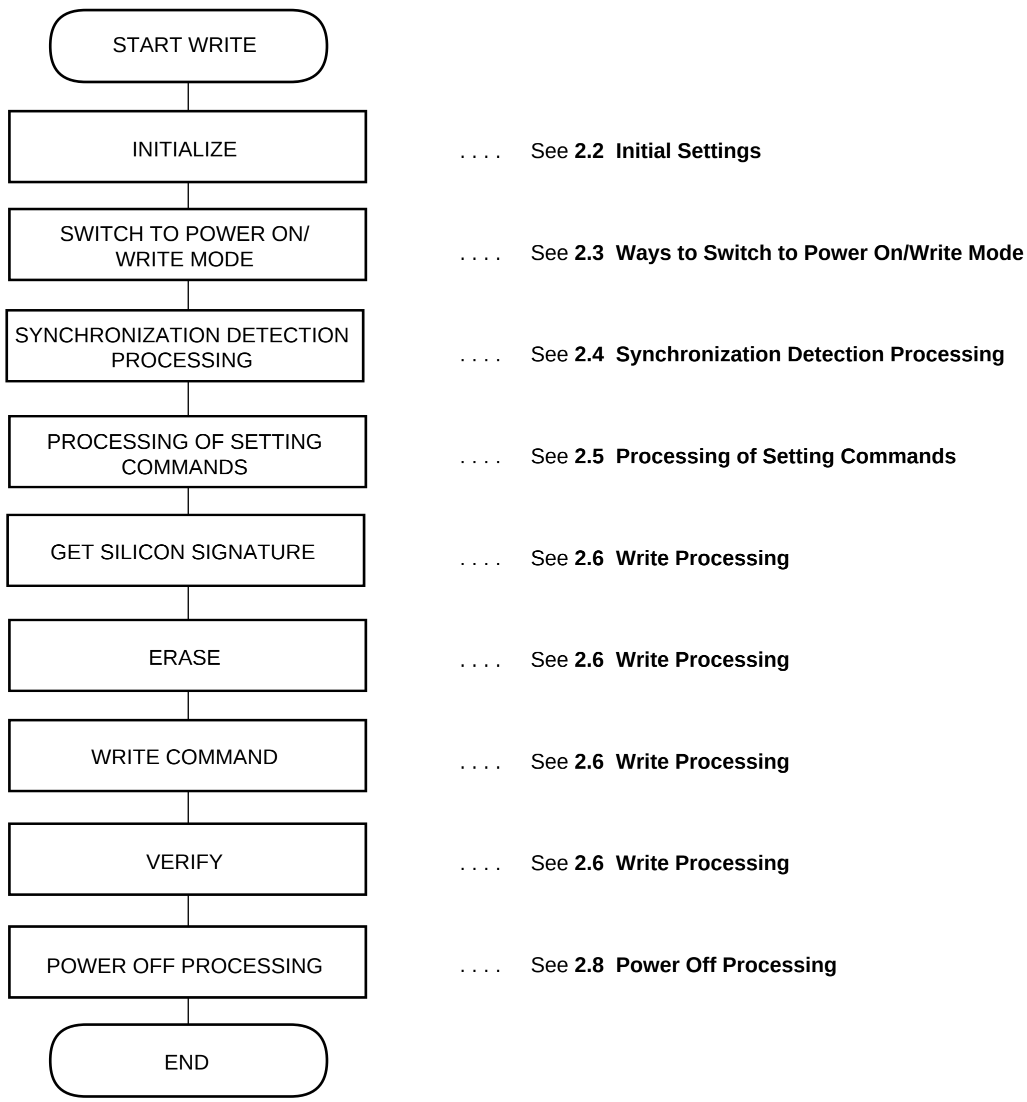
For details, see **CHAPTER 4 SAMPLE PROGRAMS**.

CHAPTER 2 BASIC FORMAT

2.1 Flow Chart of Write Operation

The operation of writing to the flash microcontroller proceeds via predetermined steps.
A basic flow chart of the steps required when writing to flash memory is shown below in Figure 2-1.

Figure 2-1. Basic Flow Chart



2.2 Initial Settings

The following initial settings must be made before writing to the flash microcontroller.

(1) Initial settings for flash programmer's controller

Typically, a microcontroller is used as the controller for the flash programmer. Therefore, before writing to the flash microcontroller, initial settings must be made for the flash programmer's controller.

An example of initial settings when using the μPD78P4038Y (an NEC 16-bit microcontroller) as the controller is shown in **CHAPTER 4 SAMPLE PROGRAMS**.

(2) Setting of parameters required for write data and write control

The write data (program data) to be written to the flash microcontroller, along with the parameters required for write control, must prepared for the flash programmer.

The parameters required for controlling write operations to the flash microcontroller are listed below.

- Target series: Select either 78K/0 Series or 78K/0S Series
- Erase time (time required by the flash microcontroller to erase the data in the flash memory)
- Write start address
- Write end address
- CPU clock source: Select the method for supplying the flash microcontroller's operating clock from the flash programmer
- CPU clock speed: Set the speed of the flash microcontroller's operating clock
- V_{PP} pulse count: Select the communication method to be used with the flash microcontroller
- CSI communication clock speed: Select speed of communication clock (clock used for 3-wire serial communication or pseudo 3-wire serial communication between flash programmer and flash microcontroller)
- Baud rate selection: Select the communication baud rate for UART communication between the flash programmer and the flash microcontroller
- Slave address: Slave address for flash microcontroller during IIC communications

For details of the above parameters, see **4.3 Description of RAM**.

2.3 Ways to Switch to Power On/Write Mode

To erase and write to the flash microcontroller, the flash programmer sets the flash microcontroller to the flash memory write mode. You must also select the communication method to be used by the flash programmer (for details, see **Table 2-1. Selection of Communication Method for Write Operation**). Select the communication method immediately after turning on the power to the flash microcontroller. For details, see **2.3.1 Switching to power on/write mode** below.

2.3.1 Switching to power on/write mode

Applying 10-V voltage to the flash microcontroller's V_{PP} pin switches the flash microcontroller from normal operation mode to flash memory write mode.

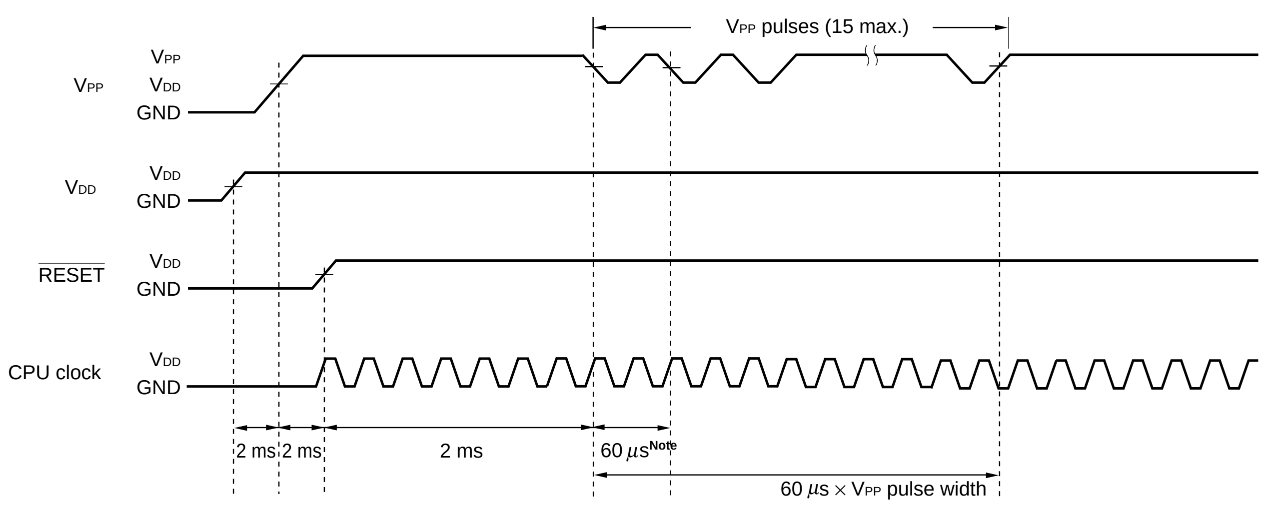
The power-on sequence is described below.

<1> Apply power to the V_{DD} pin after the flash microcontroller's $\overline{RESET}$ pin is connected to a GND potential.
<2> Apply 10-V voltage to the flash microcontroller's V_{PP} pin.
<3> Connect the flash microcontroller's $\overline{RESET}$ pin to a V_{DD} potential (to clear reset).
<4> Supply a CPU clock to the flash microcontroller.
<5> Send a pulse to the flash microcontroller's V_{PP} pin to select the write communication method.
<6> Maintain application of 10-V voltage to the flash microcontroller's V_{PP} pin.

The flash programmer is supported as a CPU clock supply source. To use this option, do not supply the CPU clock until after the rise of the V_{DD} signal.

The following is a timing chart of the switch to power on/write mode. The timing for switching to power on/write mode that is shown in Figure 2-2 is the timing that is used in the sample programs in Chapter 4. For details, see **CHAPTER 4 SAMPLE PROGRAMS**.

Figure 2-2. Timing of Switch to Power On/Write Mode



Note Detected at the falling edge of the V_{PP} pulse

The flash microcontroller selects the communication method according to the V_{PP} pulse that is sent from the flash programmer (i.e., the communication method selected according to the V_{PP} pulse is used to send or receive communications command and data to and from the flash microcontroller). The V_{PP} pulse counts listed in Table 2-1 are used to select the communication method. However, the list of supported communication methods varies depending on which flash microcontroller is used.

Table 2-1. Selection of Communication Method for Write Operation

Communication method	V_{PP} Pulse Count
3-wire serial I/O (channel 0)	0
3-wire serial I/O (channel 1)	1
3-wire serial I/O (channel 2)	2
3-wire serial I/O (channel 3)	3 (Handshaking support) (Not supported in example shown in **CHAPTER 4 SAMPLE PROGRAMS**).
IIC communications (Channel 0)	4
IIC communications (Channel 1)	5
IIC communications (Channel 2)	6
IIC communications (Channel 3)	7
UART communications (Channel 0)	8
UART communications (Channel 1)	9
UART communications (Channel 2)	10
UART communications (Channel 3)	11
Pseudo 3-wire serial I/O (Port A)	12
Pseudo 3-wire serial I/O (Port B)	13
Pseudo 3-wire serial I/O (Port C)	14 (Handshaking support) (Not supported in example shown in **CHAPTER 4 SAMPLE PROGRAMS**).

2.4 Synchronization Detection Processing

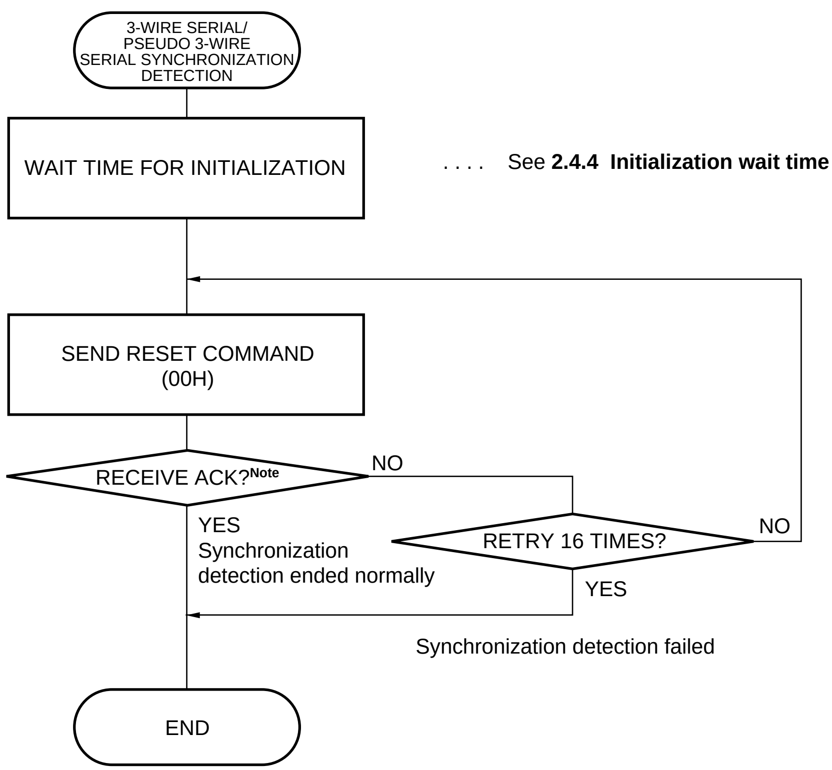
Synchronization detection processing is a type of processing whose purpose is to confirm whether or not the flash microcontroller can operate normally after it has been switched to flash memory write mode. The flash programmer sends a reset command to the flash microcontroller and check whether or not an ACK response is returned. The maximum number of retries is 16; if an ACK is issued from the flash microcontroller in 16 or fewer tries, it is determined that the flash microcontroller is in a programmable mode. The format of this reset command is described in **CHAPTER 3 WRITE SEQUENCE**.

The synchronization detection method differs depending on the communication method (3-wire serial, pseudo 3-wire serial, IIC, or UART) selected by the V_{PP} pulse. This means that synchronization detection processing is required for each communication method.

2.4.1 Synchronization detection processing for 3-wire serial and pseudo 3-wire serial communication methods

Figure 2-3 shows the flow of synchronization detection processing for the 3-wire serial and pseudo 3-wire serial communication methods. For details, see **3.1.1 Reset command** and **4.5.6 Synchronization detection processing**.

Figure 2-3. Flow of Synchronization Detection Processing for 3-Wire Serial and Pseudo 3-Wire Serial Communication Methods



Note ACK: Acknowledge

This signal (3CH) indicates when the flash microcontroller's processing ends normally.

A different signal "NACK" (FFH) indicates when a processing fault has occurred (even if NACK is not FFH, a "NACK" judgement is made whenever a value other than "3CH" is returned at the timing for receiving the ACK signal).

2.4.2 Synchronization detection processing for UART communication method

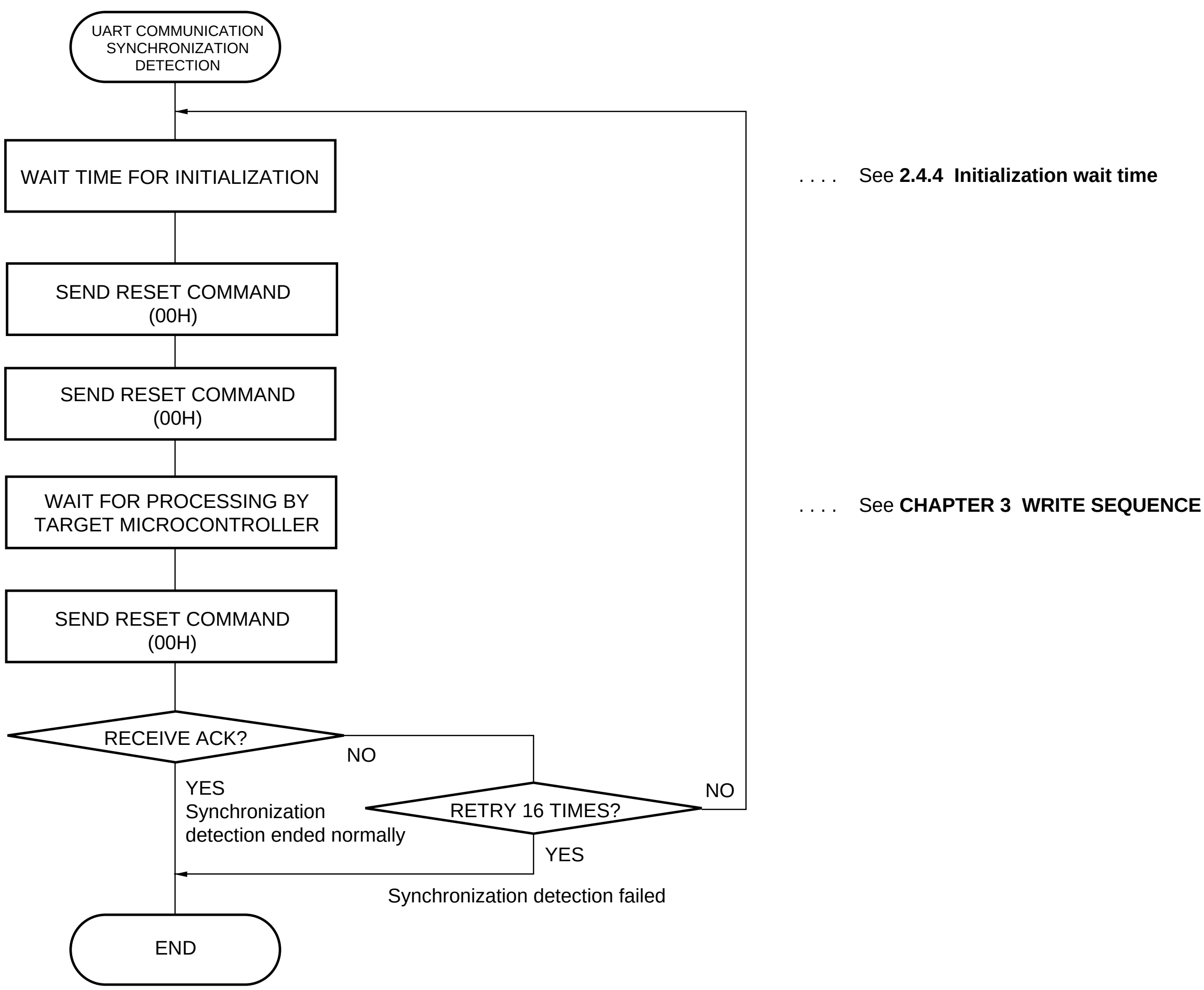
Synchronization detection for UART communication confirms whether or not an ACK response is received after a reset command has been sent three times. At that time, the flash microcontroller predicts its own operating frequency based on the low width of the first two reset commands (00H). It uses this predicted operating frequency to set a communication baud rate of 9,600 bps. If the third reset command (00H) is received correctly, an ACK is returned. If the reset command (00H) cannot be received, a NACK (FFH) is returned instead.

Table 2-2. UART Communication Conditions

Communication baud rate	9,600 bps
Parity bit	None
Data length	8 bits
Stop bits	1 bit

Figure 2-4 shows the flow of synchronization detection for UART communications.

Figure 2-4. Flow of Synchronization Detection Processing for UART Communication Method



2.4.3 Synchronization detection processing for IIC communication method

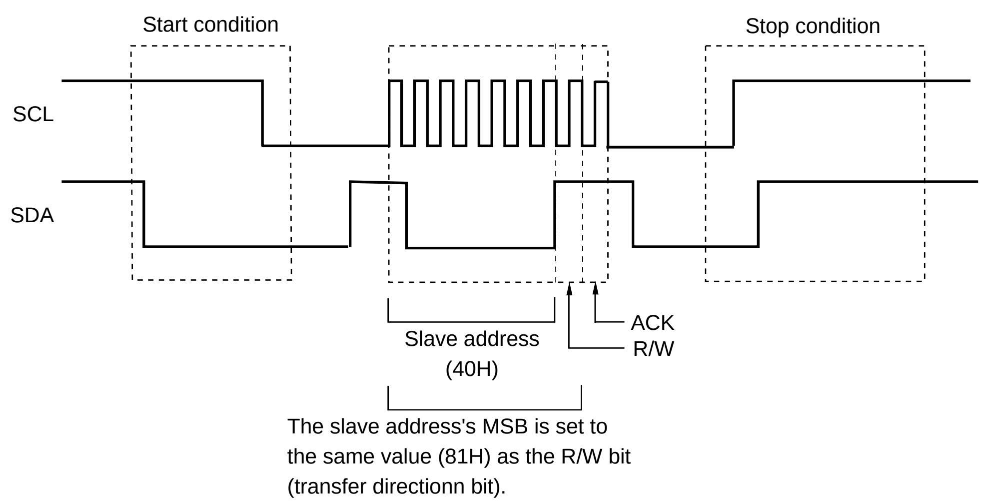
Synchronization detection for IIC communication requires that the flash microcontroller's own slave address be sent to the flash microcontroller.

Typically, when performing IIC communications, a slave address is required so that the master side can designate a slave address for the remote side. However, when you switch to write mode, the flash microcontroller's slave address becomes undefined after the power is turned on. Therefore, when performing actual communications, the slave address to be set from the flash programmer to the flash microcontroller is sent and the flash microcontroller's slave address must be determined. The range of specifiable slave address values and the sequence for determining the flash microcontroller's slave address are described below.

(1) Range of specifiable slave address values: 08H to 77H (data error occurs when out-of-range value is specified)
(2) Method for setting slave address:

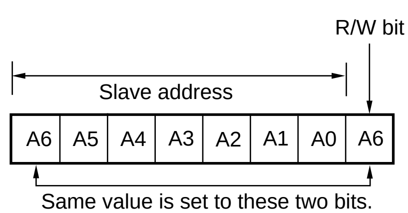
The communication protocol is illustrated in Figure 2-5.

Figure 2-5. Communication Protocol



Usually, the high-order seven bits specify the slave address and the eighth bit is the R/W (transfer direction) bit. However, when setting the slave address, the slave address's MSB is set to the same value as the eighth (R/W) bit (see **Figure 2-6. Method for Setting Slave Address**). In the above example, the slave address is 40H, so the MSB value is "1". Consequently, "1" is also set to the eighth (R/W) bit.

Figure 2-6. Method for Setting Slave Address



After the slave address has been set, the slave address is set as usual to the high-order seven bits and the R/W (transfer direction) bit value is specified for the eighth bit, after which synchronization detection processing is performed. The synchronization detection method is the same as for 3-wire serial or pseudo 3-wire serial communication, namely that a reset command is sent and the system checks to see whether or not an ACK response is returned. For details, see **3.2.1 Reset command** and **4.5.6 Synchronization detection processing**.

2.4.4 Initialization wait time

After the mode has been switched to flash memory write mode, the flash programmer must wait until the initialization wait period has elapsed before the write-related firmware in the flash microcontroller can be operated. This initialization wait time must be at least as long as the flash microcontroller's oscillation wait time and the time that write-related firmware must wait while the flash microcontroller self-initializes. After this initialization wait time has elapsed, synchronization detection processing is performed. In the sample programs shown in Chapter 4, a margin is added to the initialization wait time so that the total wait time is 100 ms.

2.5 Processing of Setting Commands

After synchronization detection processing has been completed with the flash microcontroller, the operating frequency and erase time must be sent to the flash microcontroller. If UART has been selected as the communication method for the flash microcontroller, the communication baud rate can be changed by issuing a baud rate setting command.

2.5.1 Oscillation frequency setting command

This command sets the flash microcontroller's operating frequency to the flash microcontroller. The oscillation frequency data consists of four bytes. The format of this data is shown in Table 2-3.

Table 2-3. Oscillation Frequency Data Format

Offset	Description
+0	First column (unpacked BCD)
+1	Second column (unpacked BCD)
+2	Third column (unpacked BCD)
+3	Exponent portion (signed integer; one byte)

Oscillation frequency (kHz) = (0.1 $\times$ first column + 0.01 $\times$ second column + 0.001 $\times$ third column) $\times 10^{\text{exponent}}$
Range of specifiable values: 1 MHz to 10 MHz

Example: When oscillation frequency is 5 MHz

Oscillation frequency data to be sent: (4 bytes) [05] [00] [00] [04] : 0.500 $\times 10^{4}$ kHz

2.5.2 Erase time setting command

This command sets the flash microcontroller's erase time to the flash microcontroller. The erase time data consists of four bytes. The format of this data is shown in Table 2-4.

Table 2-4. Erase Time Data Format

Offset	Description
+0	First column (unpacked BCD)
+1	Second column (unpacked BCD)
+2	Third column (unpacked BCD)
+3	Exponent portion (signed integer; one byte)

Time (s) = (0.1 $\times$ first column + 0.01 $\times$ second column + 0.001 $\times$ third column) $\times 10^{\text{exponent}}$

Example: When erase time is 2 seconds

Erase time data to be sent: (4 bytes) [02] [00] [00] [01] : 0.200 $\times 10^{1}$ s

2.5.3 Baud rate setting command

The baud rate setting command is valid only for UART communications.

Before the baud rate setting command is executed and UART communications are started, communications uses a rate of 9,600 bps. The baud rate setting command changes this communication rate. The baud rate setting command is expressed as a single-byte numerical value. The format of this command is shown in Table 2-5.

Once you have used the baud rate setting command to change the communication rate, use the reset command to double-check that communications will use the newly set baud rate. If this confirmation yields a negative result (i.e., if ACK is not received), a communication error has occurred and communications are no longer enabled. For details, see **3.3.4 Baud rate setting command**.

Table 2-5. Format of Baud Rate Setting Data

Setting Data	Baud Rate (bps)
2	4,800
3	9,600
4	19,200
5	31,250
6	38,400
7	76,800
Other	Data error

2.6 Write Processing

The following three commands are the basic commands used for writing to the flash microcontroller.

(1) Erase command

This command is used to erase the flash microcontroller's flash memory. Before issuing the erase command, issue the prewrite command to prepare for erasure.

(2) Write command

This command is used to write to the flash microcontroller's flash memory. After the write command has been executed, issue the internal verify command to check the depth of the write level.

There are two types of write commands, which provide different levels of efficiency during the write operation.

(a) High-speed write command

This command specifies the write size (the number of transferred bytes of write data: for the 78K/0 Series, the maximum number is 256 bytes (00H), for the 78K/0S Series it is 128 bytes (80H)) and the start address for writing, then performs the write operation[Note].

Note Write size is indicated by 1-byte data (00H to FFH), and start address is 3-byte data (000000H to 00EFFFH). When the write size is "00H", it indicates 256 bytes.

(b) Continuous write command

This command performs the write operation for a write size specified by the high-speed write command. Data is written to the next address after the address last written to by either the high-speed write command or the continuous write command.

Note Write size is indicated by 1-byte data (00H to FFH), and start address is 3-byte data (000000H to 00EFFFH). When the write size is "00H", it indicates 256 bytes.

(3) Verify command

This command is used to verify the contents of the flash microcontroller's flash memory and the contents of data sent from the flash programmer (the data transfer size is fixed at 256 bytes for the 78K/0 Series and 128 bytes for the 78K/0S Series).

In addition to the three basic commands described above, there are also the following five types of commands.

(a) Blank check command

This command is used to confirm that the flash microcontroller's flash memory has been erased.

(b) Prewrite command

This command clears the flash memory contents to "00H" to prepare for erasure by the flash microcontroller. This command must be executed before executing the erase command.

(c) Internal verify command

This command checks the depth of the write level. This command must be executed after executing the write command.

(d) Silicon signature command

This command is used to get the flash microcontroller's silicon signature. The meaning of the silicon signature data is shown in Table 2-6. The silicon signature data when using the μPD78F9197 as the target is shown as an example.

The silicon signature data's MSB (bit 7) is the parity (odd parity) bit.

Table 2-6. Meaning of Silicon Signature Data

SIGNETURE DATA			Meaning
HEX	BIN		
10H	0	001 0000	Vendor Code (NEC)
7FH	0	111 1111	Single chip μcom ID code
49H	0	100 1001	Electrical information
7FH	0	111 1111	Last Address: 05FFFH Flash memory: 24 KB
BFH	1	011 1111	
01H	1	000 0001	
C4H	1	100 0100	"D"
37H	0	011 0111	"7"
38H	0	011 1000	"8"
46H	0	100 0110	"F"
39H	1	011 1001	"9"
31H	0	011 0001	"1"
39H	1	011 1001	"9"
37H	0	011 0111	"7"
20H	0	010 0000	"Space"
20H	0	010 0000	"Space"
00H	0	000 0000	"00" is information without block divisions

(e) Status check command

This command is used to query the flash microcontroller's internal command execution status. Table 2-7 lists the command execution status corresponding to each bit.

Table 2-7. Meaning of Status and Data Bits in Status Check Command

Bit position	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Flag	Erase mode	Write mode	Verify mode	Blank check mode	Erase error	Write error	Verify error	Blank check error

Erase mode	Description
0	Not erasing flash memory
1	Erasing flash memory

Write mode	Description
0	Not writing
1	Writing

Verify mode	Description
0	Not verifying
1	Verifying

Blank check mode	Description
0	Not performing blank check of flash memory
1	Performing blank check of flash memory

Erase error	Description
0	No flash memory erase error
1	Flash memory erase error has occurred

Write error	Description
0	No write error
1	Write error has occurred

Verify error	Description
0	No verify error
1	Verify error has occurred

Blank check error	Description
0	No flash memory blank check error
1	Flash memory blank check error has occurred

For timing charts and sample programs related to these commands, see **CHAPTER 3 WRITE SEQUENCE** and **CHAPTER 4 SAMPLE PROGRAMS**.

2.7 List of Commands

The flash microcontroller's control commands are listed in Table 2-8.

Table 2-8. List of Commands

Command Code	Command Name	Description of Processing
00H	Reset command	This command initializes the flash microcontroller's mode and confirms synchronization detection as part of synchronization detection processing.
90H	Oscillation frequency setting command	This command notifies the flash microcontroller concerning its operating clock. The flash microcontroller internally receives this frequency value as the basic frequency value for calculating the write time, erase time, etc.
95H	Erase time setting command	This command sets the flash microcontroller's erase time to the flash microcontroller.
9AH	Baud rate setting command	This command is used to change the communication rate when UART communications has been selected.
48H	Prewrite command	This command clears the flash microcontroller's program area (flash memory area) to "00H" to prepare for erasure before the erase command can be used.
20H	Erase command	This command erases the flash microcontroller's program area (flash memory).
40H	High-speed write command	This command writes data to the flash microcontroller's program area (flash memory). It is used in combination with the status check command to check for write failures while the write operation is in progress.
44H	Continuous write command	This command is used to execute another write operation after the high-speed write command has been issued. This format eliminates the need to transfer the "write start address" and "transfer byte count" from the high-speed write command.
18H	Internal verify command	This command checks the depth of the write level after a write command has been executed.
11H	Verify command	This command compares the contents of the flash microcontroller's program area (flash memory) with the data received by the flash microcontroller.
30H	Blank check command	This command checks whether or not the flash microcontroller's program area (flash memory) has been erased.
C0H	Silicon signature command	This command gets the flash microcontroller's device information.
70H	Status check command	This command gets the flash microcontroller's internal command execution status.

2.8 Power Off Processing

After the write operation to the flash microcontroller has been completed, the power to the flash microcontroller is turned off. The power off sequence is described below.

<1> Set the flash microcontroller's reset pin to low level
<2> Turn off V_{PP} voltage
<3> Turn off V_{DD} voltage

For details of power off processing, see **CHAPTER 4 SAMPLE PROGRAMS**.

[MEMO]

CHAPTER 3 WRITE SEQUENCE

This command indicates the communication timing and processing time related to the execution of commands. Here, the processing time refers to the flash microcontroller's processing time. The commands and data described below cannot be sent normally until a wait time consisting of the following number of clocks (i.e., time) has elapsed.

The clock referred to below is the flash microcontroller's operating clock. The waits that are executed in the examples shown in **CHAPTER 4 SAMPLE PROGRAMS** are calculated using wait time based on this number of clocks.

The following commands are issued after completion of the steps described earlier, namely "Switching to power on/write mode" and "Synchronization Detection Processing" (which uses the reset command). The four supported communication methods are 3-wire serial, IIC, UART, and pseudo 3-wire serial. For description of how the communication method is selected, see **2.3 Ways to Switch to Power On/Write Mode**.

The communication format for 3-wire serial and pseudo 3-wire serial communications has an 8-bit data length with MSB first. The serial clock is supplied from the flash programmer side.

The communication format for IIC communications has an 8-bit data length with MSB first. The flash programmer performs the master operations and the serial clock and slave address are both output from the flash programmer side.

The communication format for UART communications is shown in Table 3-1.

Table 3-1. Communication Format for UART Communications

Communication baud rate (bps)	4,800, 9,600, 19,200, 31,250, 38,400, 76,800[Note]
Parity bit	None
Data length	8 bits
Stop bits	1 bit

Note Be sure to use 9600 bps for synchronization detection processing.

The write sequence for 3-wire serial and pseudo 3-wire serial communications is described in **3.1 Write Sequence for 3-Wire Serial and Pseudo 3-Wire Serial Communications**, that for IIC communications is described in **3.2 Write Sequence for IIC Communications**, and that for UART communications is described in **3.3 Write Sequence for UART Communications**.

3.1 Write Sequence for 3-Wire Serial and Pseudo 3-Wire Serial Communications

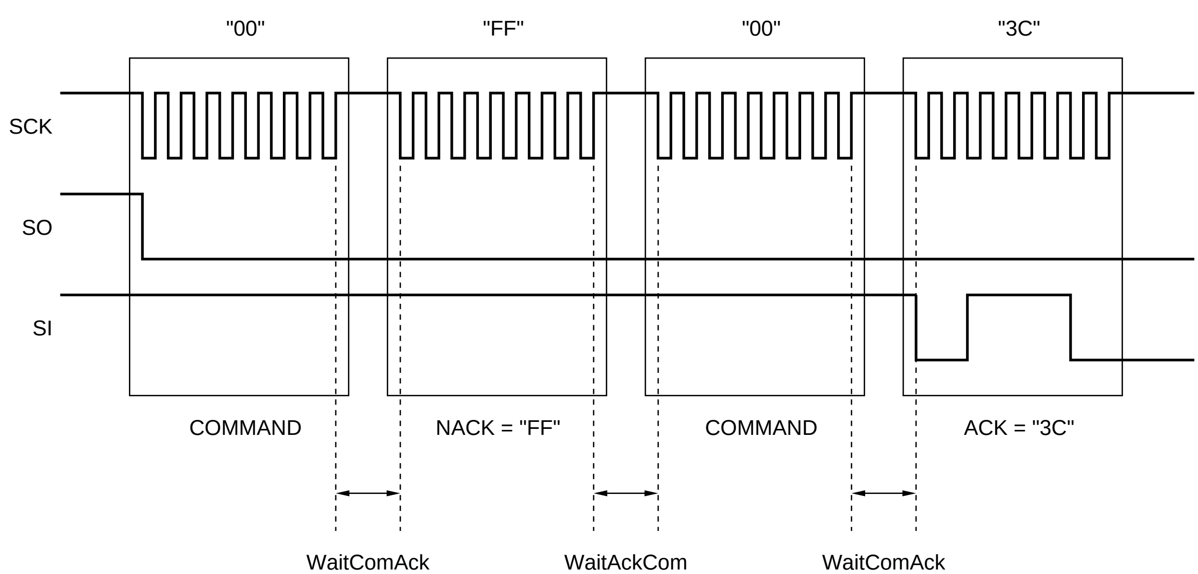
The write sequence for 3-wire serial and pseudo 3-wire serial communications is shown below.
The number of wait clocks is represented as "A (B) / C (D)" on the following pages.

A: Number of wait clocks when the target microcontroller is a 78K/0 Series product and the communication method is the 3-wire serial method
B: Number of wait clocks when the target microcontroller is a 78K/0 Series product and the communication method is the pseudo 3-wire serial method
C: Number of wait clocks when the target microcontroller is a 78K/0S Series product and the communication method is the 3-wire serial method
D: Number of wait clocks when the target microcontroller is a 78K/0S Series product and the communication method is the pseudo 3-wire serial method

3.1.1 Reset command

This command is used to confirm synchronization detection as part of synchronization detection processing.

Figure 3-1. Timing of Reset Command



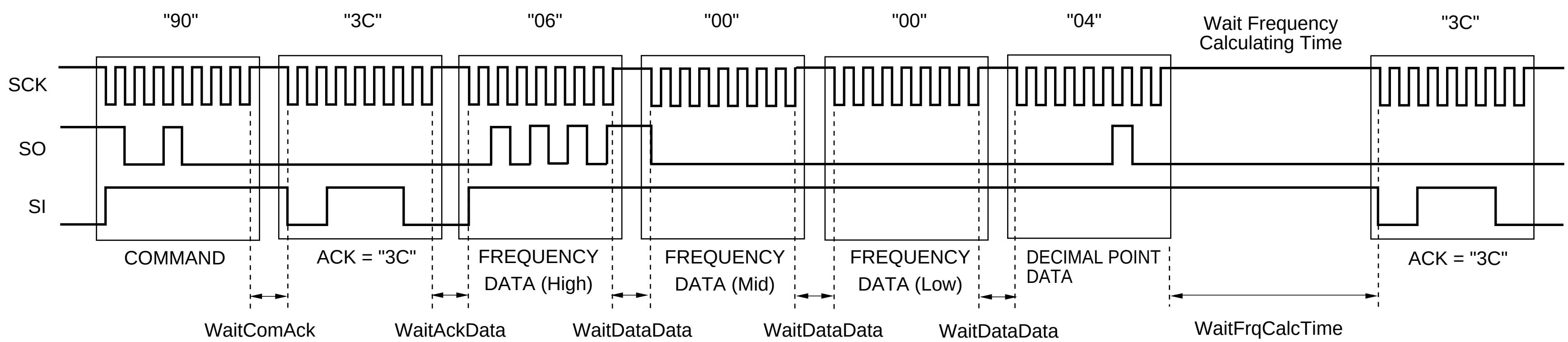
WaitComAck: The number of wait clocks is at least 900 (1630) / 1040 (2580) CPU clocks
The wait time is the time between issuing a command and receiving an ACK signal.
WaitAckCom: The number of wait clocks is at least 170 (790) / 210 (820) CPU clocks
The wait time is the time between receiving an ACK signal and issuing a command.

Caution Once a NACK signal is returned, retries are performed until an ACK signal is returned. The maximum number of retries is 16. A communication error occurs if 17 or more retries are attempted. For details, see CHAPTER 4 SAMPLE PROGRAMS.

3.1.2 Oscillation frequency setting command

This command notifies the flash microcontroller concerning its operating clock. The flash microcontroller internally uses this frequency value as the basic frequency value for calculating the write time, erase time, etc.

Figure 3-2. Timing of Oscillation Frequency Setting Command



WaitComAck: The number of wait clocks is at least 900 (1,630)/1,040 (2,580) CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 230 (640)/190 (700) CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

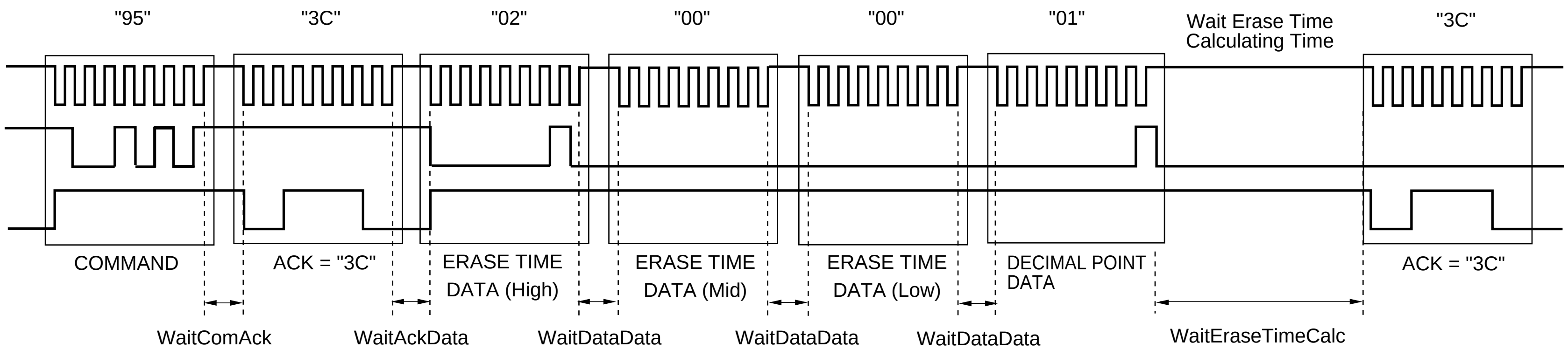
WaitDataData: The number of wait clocks is at least 300 (860)/360 (1,560) CPU clocks. The wait time is the time between receiving two sets of data.

WaitFrqCalcTime: The number of wait clocks is at least 2,200 (3,380)/31,600 (44,200) CPU clocks. The wait time is the time used to calculate the oscillation frequency setting.

3.1.3 Erase time setting command

This command sets the flash microcontroller's erase time to the flash microcontroller's program area (flash memory).

Figure 3-3. Timing of Erase Time Setting Command



WaitComAck: The number of wait clocks is at least 900 (1,630)/1,040 (2,580) CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 230 (640)/190 (700) CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

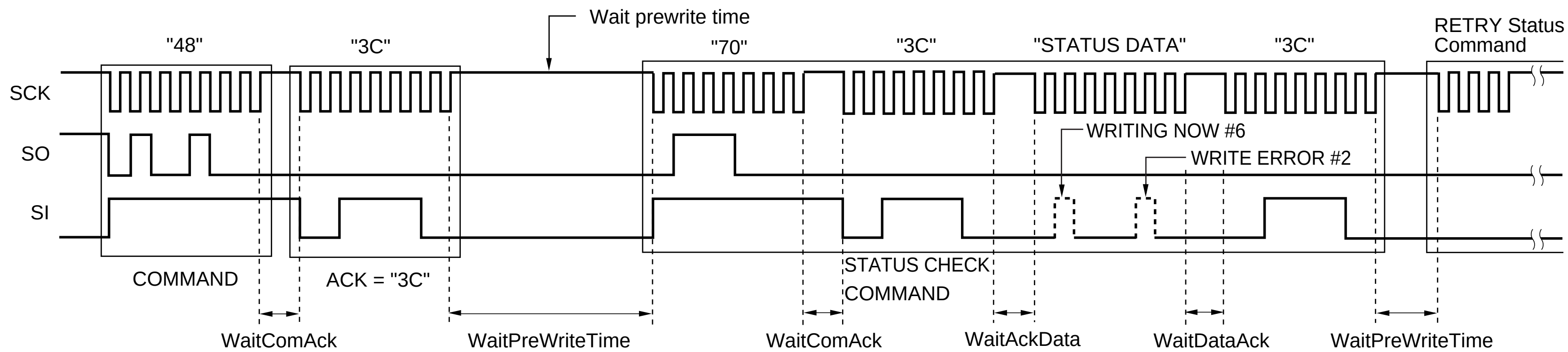
WaitDataData: The number of wait clocks is at least 300 (860)/360 (1,560) CPU clocks. The wait time is the time between receiving erase time data (high) and receiving erase time data (low).

WaitEraseTimeCalc: The number of wait clocks is at least 1,200 (1,690)/2,000 (27,600) CPU clocks. The wait time is the time used to calculate the erase time setting.

3.1.4 Prewrite command

This command must be used to clear the flash microcontroller's program area (flash memory area) to "00H" to prepare for erasure before the erase command can be used.

Figure 3-4. Timing of Prewrite Command



WaitComAck: The number of wait clocks is at least 900 (1,630)/1,040 (2,580) CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 230 (640)/190 (700) CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

WaitDataAck: The number of wait clocks is at least 350 (960)/320 (1,600) CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

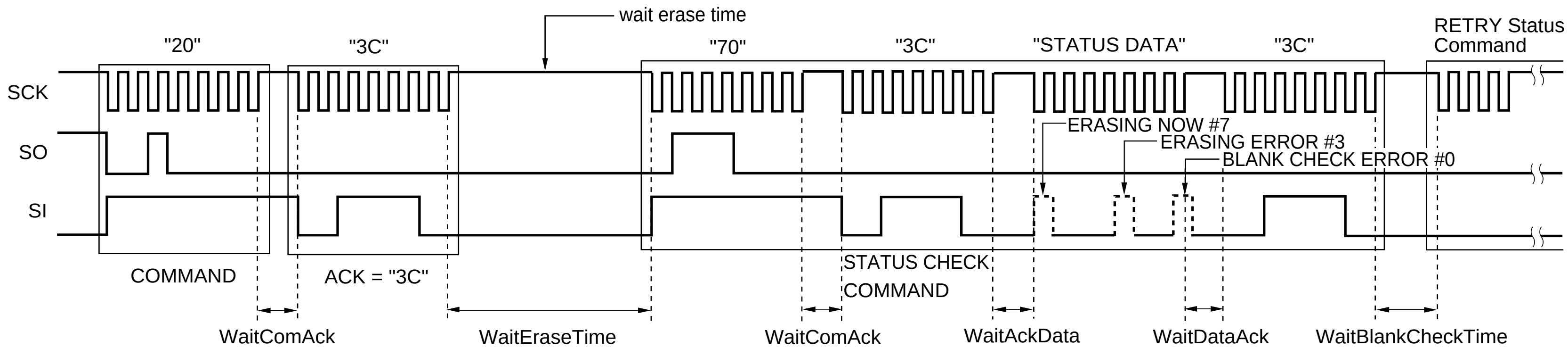
WaitPreWriteTime: The number of wait clocks is at least (230 (330)/216 (340) CPU clocks + flash memory write time[Note]) × flash memory capacity (bytes).

Note See **CHAPTER 4 SAMPLE PROGRAMS**.

3.1.5 Erase command

This command erases the flash microcontroller's program area (flash memory).

Figure 3-5. Timing of Erase Command



WaitComAck:	The number of wait clocks is at least 900 (1,630)/1,040 (2,580) CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.
WaitAckData:	The number of wait clocks is at least 230 (640)/190 (700) CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.
WaitDataAck:	The number of wait clocks is at least 350 (960)/320 (1,600) CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.
WaitEraseTime:	The number of wait clocks is at least the erase time set via the erase time setting command + (690 (840)/175 (235) CPU clocks × flash memory capacity (bytes)). The wait time is equal to the erase time.
WaitBlankCheckTime:	The number of wait clocks is at least 690 (840)/175 (235) CPU clocks × flash memory capacity (bytes).

3.1.6 Write commands

This command writes data to the flash microcontroller's program area (flash memory). It is used in combination with the status check command to check for write failures while the write operation is in progress.

Figure 3-6. Timing of High-Speed Write Command

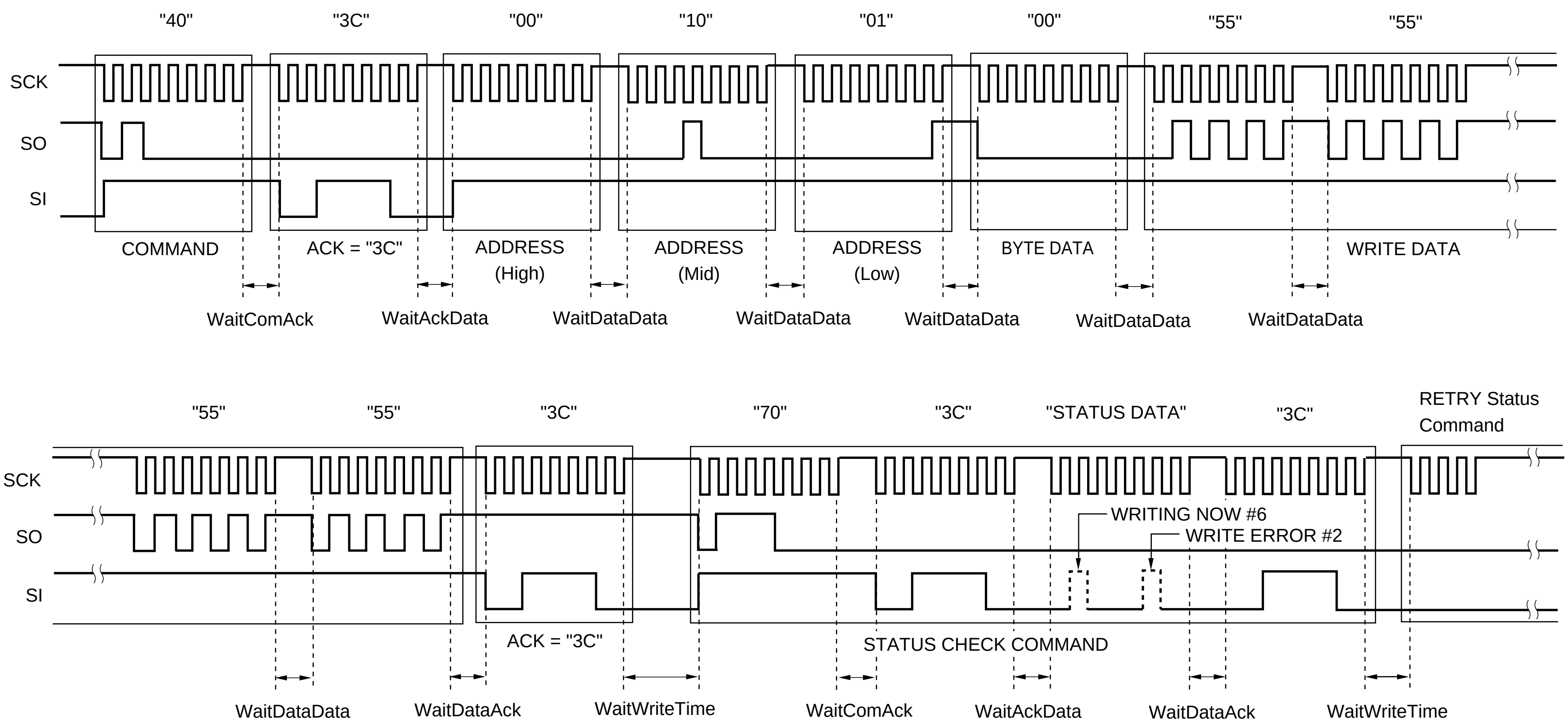
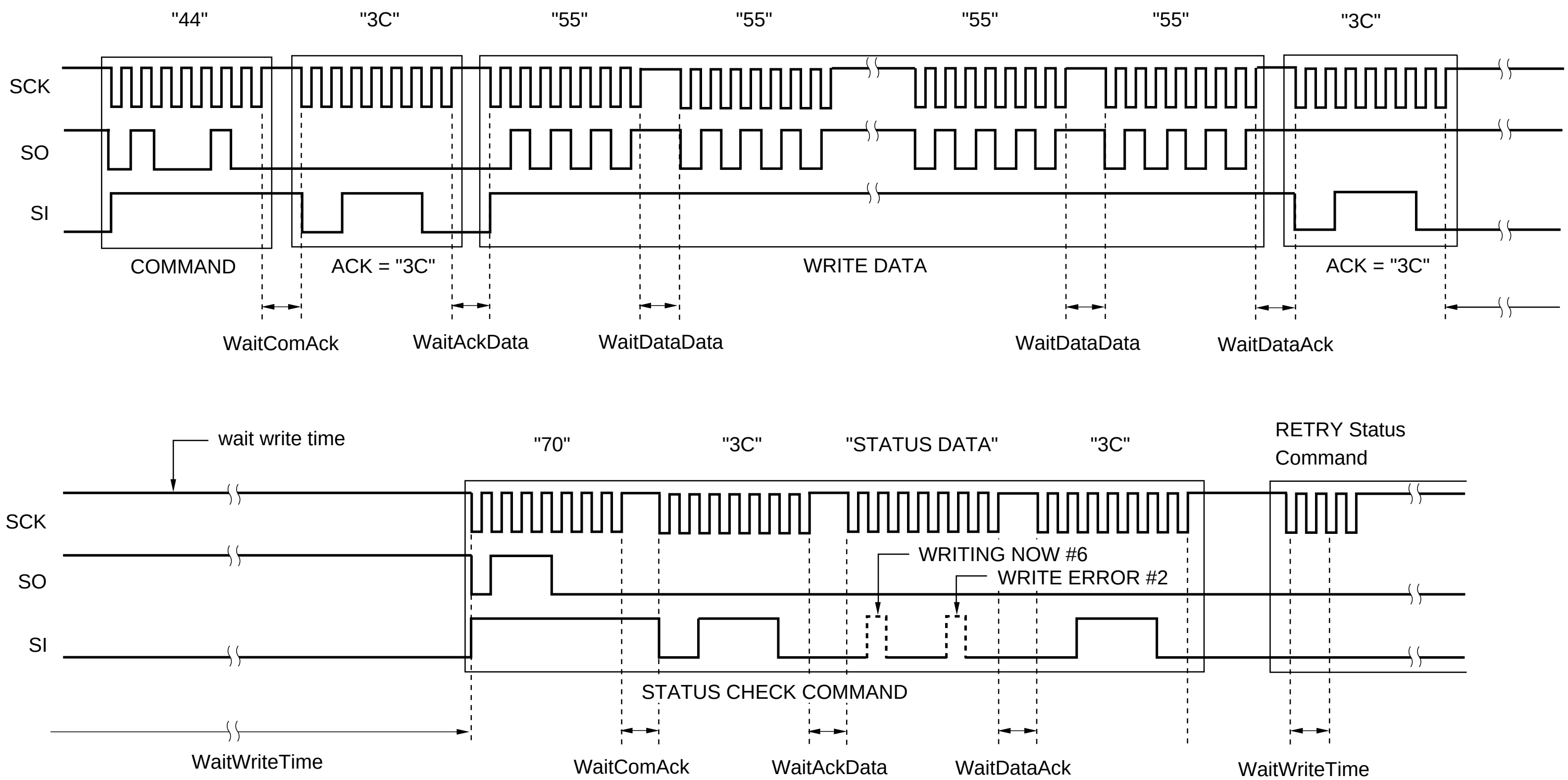


Figure 3-7. Timing of Continuous Write Command



WaitComAck: The number of wait clocks is at least 900 (1,630)/1,040 (2,580) CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 230 (640)/190 (700) CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

WaitDataAck: The number of wait clocks is at least 350 (960)/320 (1,600) CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

WaitDataData: The number of wait clocks is at least 300 (860)/360 (1,560) CPU clocks. The wait time is the time between receiving two sets of data.

WaitWriteTime: The number of wait clocks is at least (1,010 (1,010)/275 (440) CPU clocks × flash memory write time[Note 1] × write data size (bytes))[Note 2].

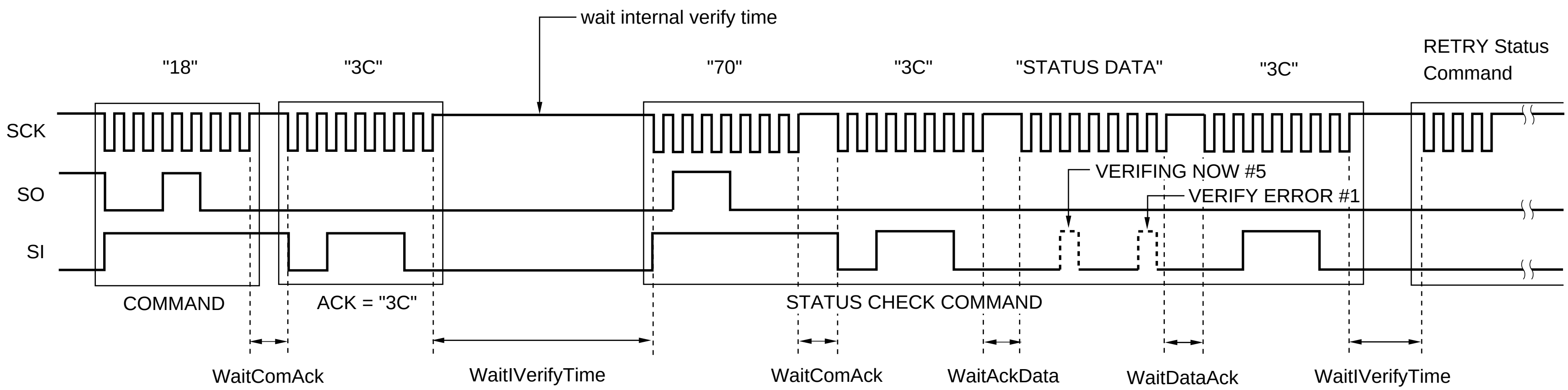
Notes 1. See **CHAPTER 4 SAMPLE PROGRAMS**.

2. Write data size: 1 to 256 bytes (for 78K/0) or 1 to 128 bytes (for 78K/0S)

3.1.7 Internal verify command

This command is used after the write command has been executed to check the depth of the write level.

Figure 3-8. Timing of Internal Verify Command



WaitComAck: The number of wait clocks is at least 900 (1,630)/1,040 (2,580) CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 230 (640)/190 (700) CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

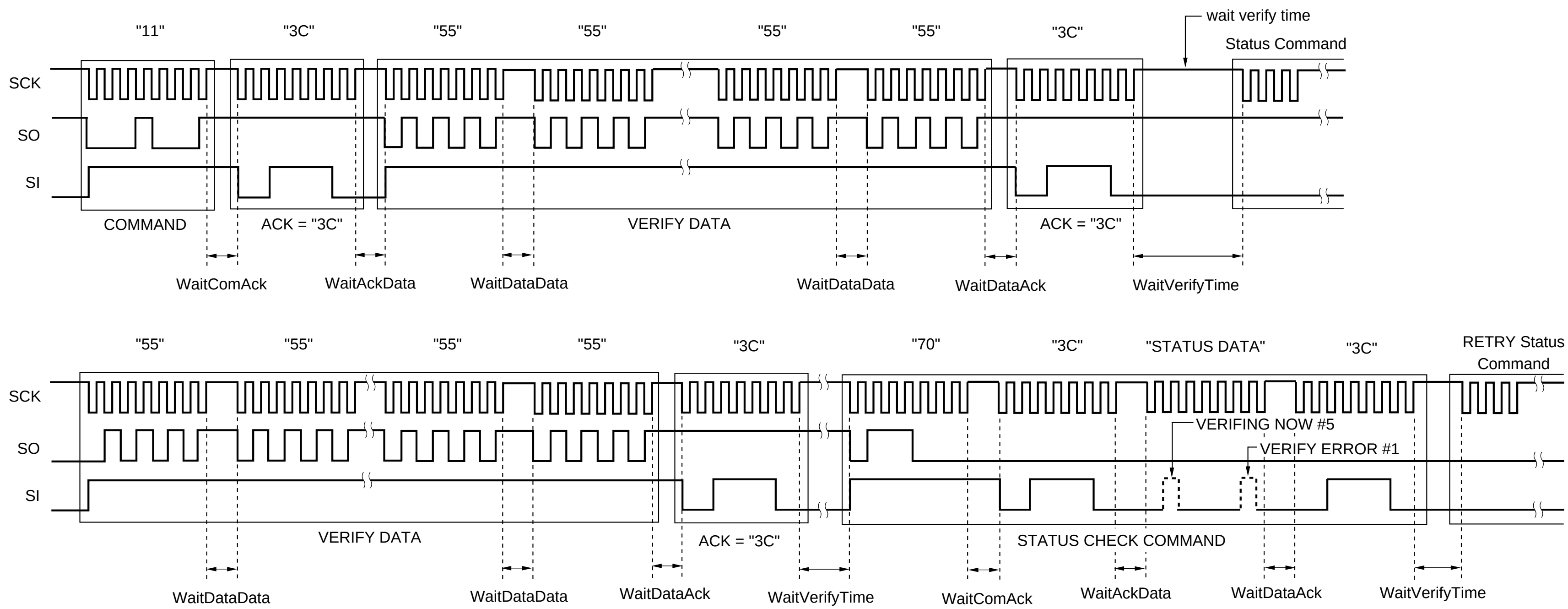
WaitDataAck: The number of wait clocks is at least 350 (960)/320 (1,600) CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

WaitVerifyTime: The number of wait clocks is at least 840 (840)/230 (325) CPU clocks × flash memory capacity (bytes).

3.1.8 Verify command

This command compares the contents of the flash microcontroller's program area (flash memory) with the data received by the flash microcontroller.

Figure 3-9. Timing of Verify Command



WaitComAck: The number of wait clocks is at least 900 (1,630)/1,040 (2,580) CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 230 (640)/190 (700) CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

WaitDataAck: The number of wait clocks is at least 350 (960)/320 (1,600) CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

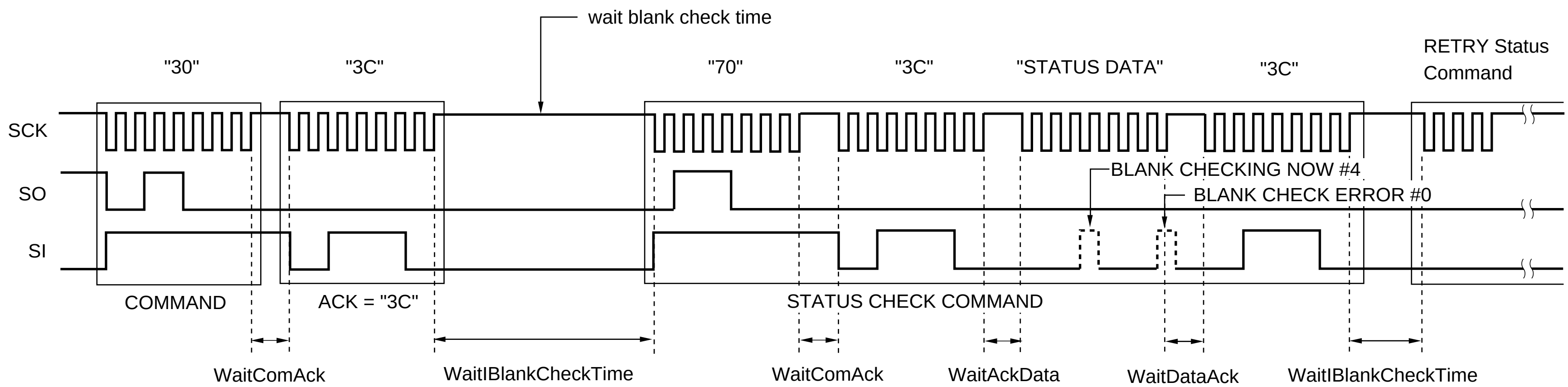
WaitDataData: The number of wait clocks is at least 300 (860)/360 (1,560) CPU clocks. The wait time is the time between receiving two sets of data.

WaitVerifyTime: The number of wait clocks is at least 258,600 (258,600)/29,400 (41,800) CPU clocks.

3.1.9 Blank check command

This command checks whether or not the flash microcontroller's program area (flash memory) has been erased.

Figure 3-10. Timing of Blank Check Command



WaitComAck: The number of wait clocks is at least 900 (1,630)/1,040 (2,580) CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 230 (640)/190 (700) CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

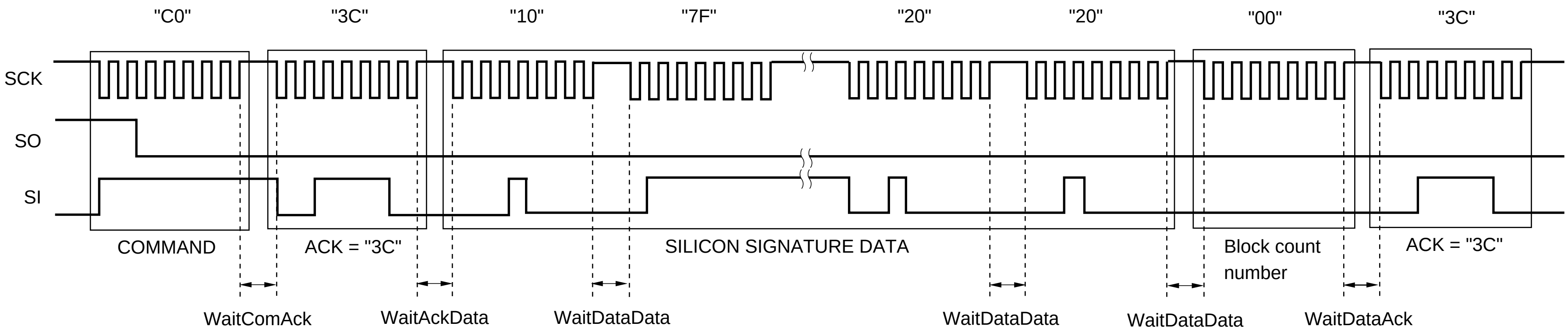
WaitDataAck: The number of wait clocks is at least 350 (960)/320 (1,600) CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

WaitBlankCheckTime: The number of wait clocks is at least 690 (840)/175 (235) CPU clocks × flash memory capacity (bytes).

3.1.10 Silicon signature command

This command gets the flash microcontroller's device information (silicon signature). For description of the silicon signature data, see **Table 2-6. Meaning of Silicon Signature Data**.

Figure 3-11. Timing of Silicon Signature Command



WaitComAck: The number of wait clocks is at least 900 (1,630)/1,040 (2,580) CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 230 (640)/190 (700) CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

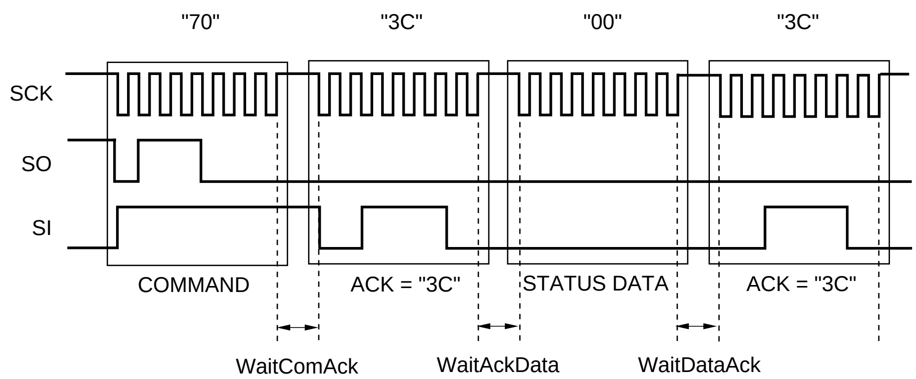
WaitDataData: The number of wait clocks is at least 300 (860)/360 (1,560) CPU clocks. The wait time is the time between receiving two sets of data.

WaitDataAck: The number of wait clocks is at least 350 (960)/320 (1,600) CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

3.1.11 Status check command

This command gets the flash microcontroller's internal command execution status and also gets the execution results. The status check command can be executed any number of times after any command is executed. The status data is 8-bit data in which the bits are assigned values indicating the command's execution status and execution results. For description of the status data, see **Table 2-7. Meaning of Status and Data Bits in Status Check Command**.

Figure 3-12. Timing of Status Check Command



WaitComAck: The number of wait clocks is at least 900 (1,630)/1,040 (2,580) CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 230 (640)/190 (700) CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

WaitDataAck: The number of wait clocks is at least 350 (960)/320 (1,600) CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

3.2 Write Sequence for IIC Communications

The following are timing charts for each command used during IIC communications.

The number of wait clocks indicated below are represented as:

(Number of wait clocks when the target microcontroller is a 78K/0 Series product)/(Number of wait clocks when the target microcontroller is a 78K/0S Series product).

In the examples, the slave address is indicated as "40H" (when the transfer direction bit is included, the value is "80H" when sending and "81H" when receiving).

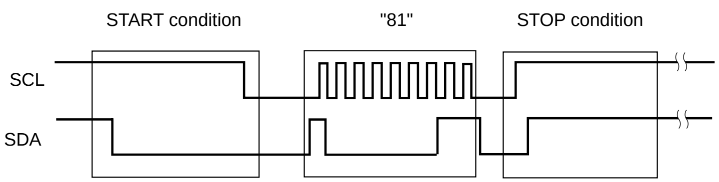
For details of the slave address, see **2.4.3 Synchronization detection processing for IIC communication method**.

3.2.1 Reset command

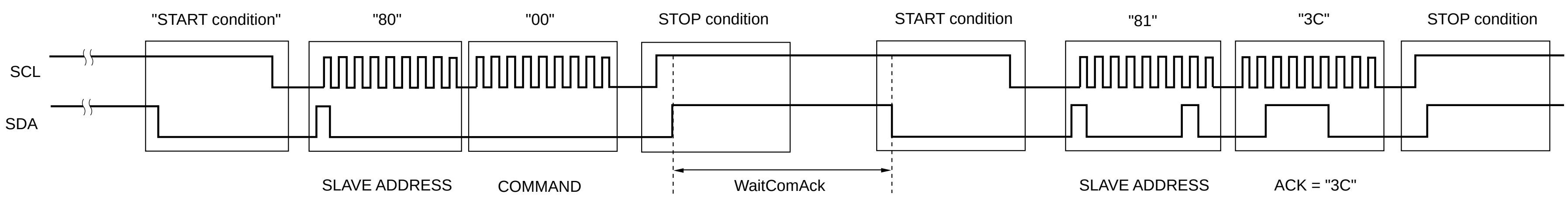
This command is used to confirm synchronization detection as part of synchronization detection processing.

Before sending a reset command, the flash microcontroller's slave address must be set. For a description of how to set the slave address, see **2.4.3 Synchronization detection processing for IIC communication method**.

Figure 3-13. Timing of Reset Command



Set the flash microcontroller's slave address before sending the reset command.
After that, normal communications are performed.



WaitComAck: The number of wait clocks is at least 1,030/1,240 CPU clocks. The wait time is the time between a command and an ACK signal.

Caution Once a NACK signal is returned, retries are performed until an ACK signal is returned. The maximum number of retries is 16. A communication error occurs if 17 or more retries are attempted. For details, see CHAPTER 4 SAMPLE PROGRAMS.

3.2.2 Oscillation frequency setting command

This command notifies the flash microcontroller concerning its operating clock. The flash microcontroller internally uses this frequency value as the basic frequency value for calculating the write time, erase time, etc.

Figure 3-14. Timing of Oscillation Frequency Setting Command (1/2)

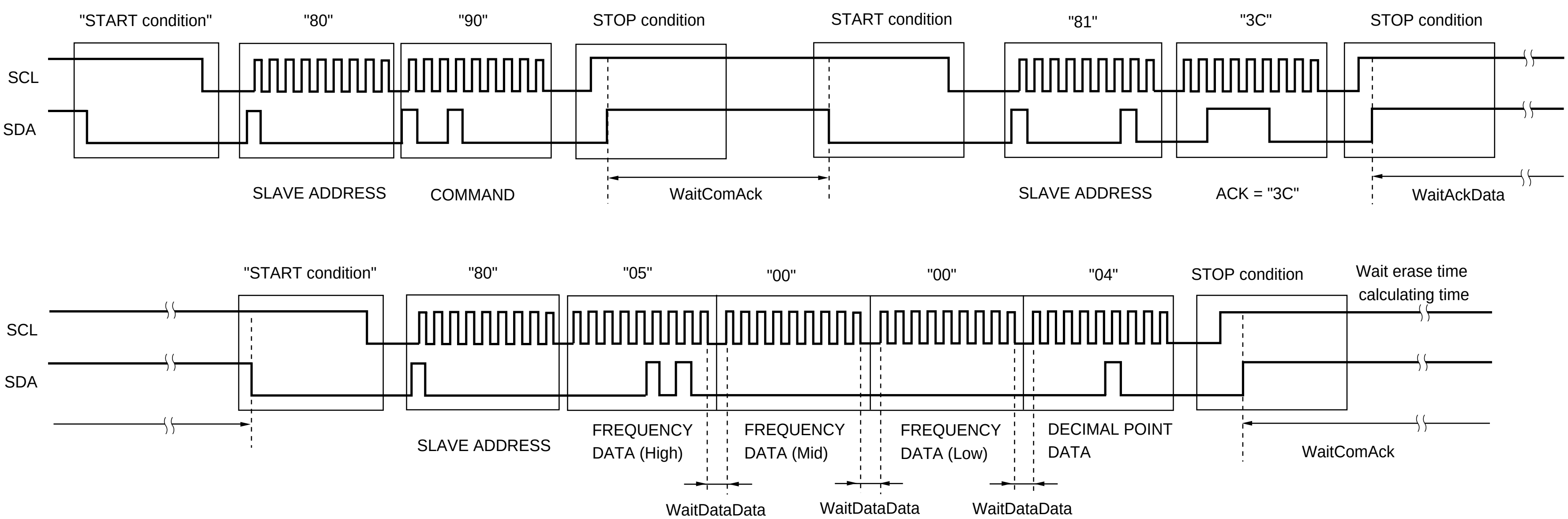
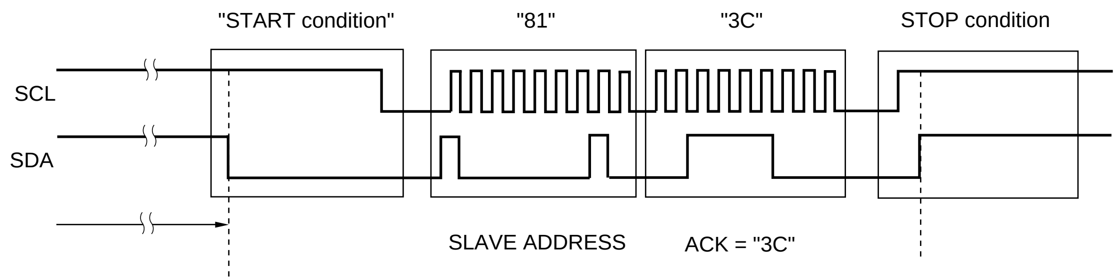


Figure 3-14. Timing of Oscillation Frequency Setting Command (2/2)



WaitComAck: The number of wait clocks is at least 1,030/1,240 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 50/640 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

WaitDataData: The number of wait clocks is at least 70/530 CPU clocks. The wait time is the time between receiving frequency data (high) and receiving frequency data (low).

WaitFrqCalcTime: The number of wait clocks is at least 2,350/65,000 CPU clocks. The wait time is the time used to calculate the oscillation frequency setting.

3.2.3 Erase time setting command

This command sets the flash microcontroller's erase time to the flash microcontroller's program area (flash memory).

Figure 3-15. Timing of Erase Time Setting Command (1/2)

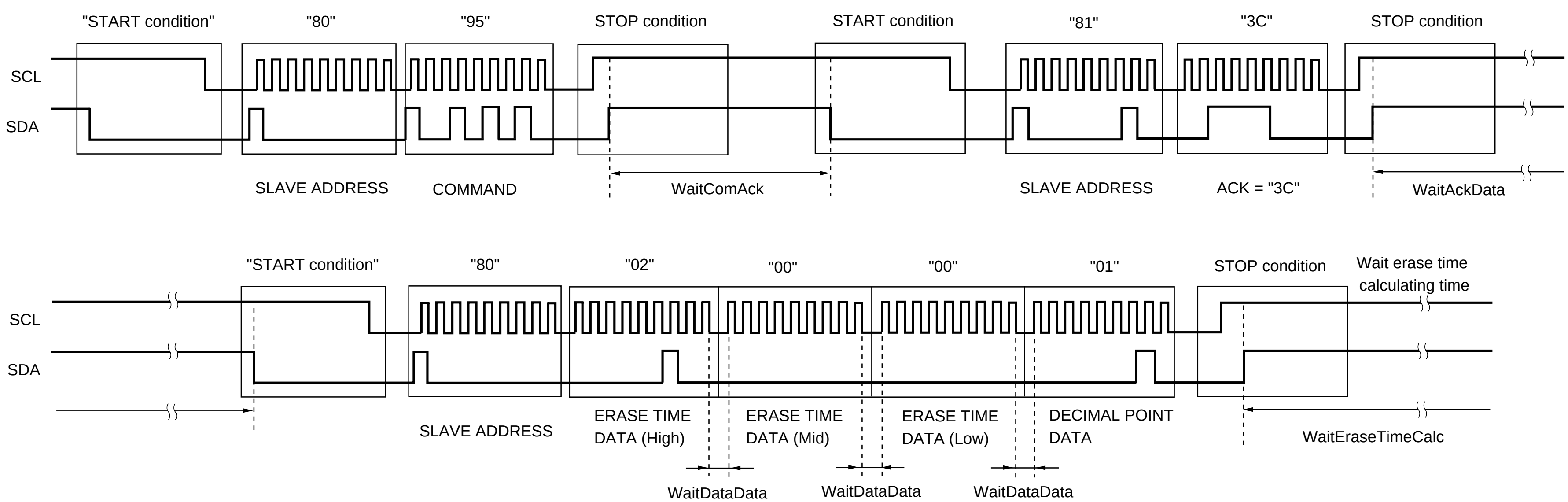
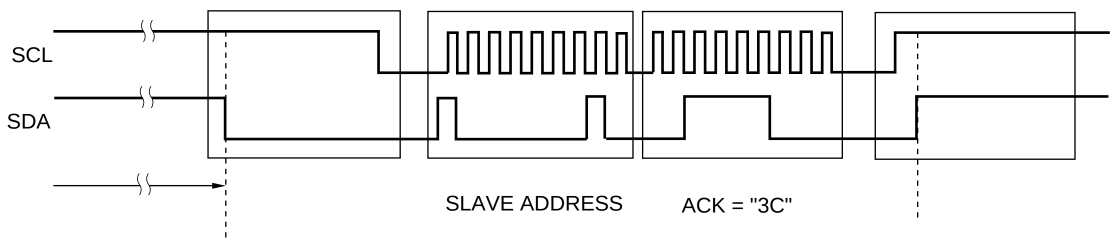


Figure 3-15. Timing of Erase Time Setting Command (2/2)

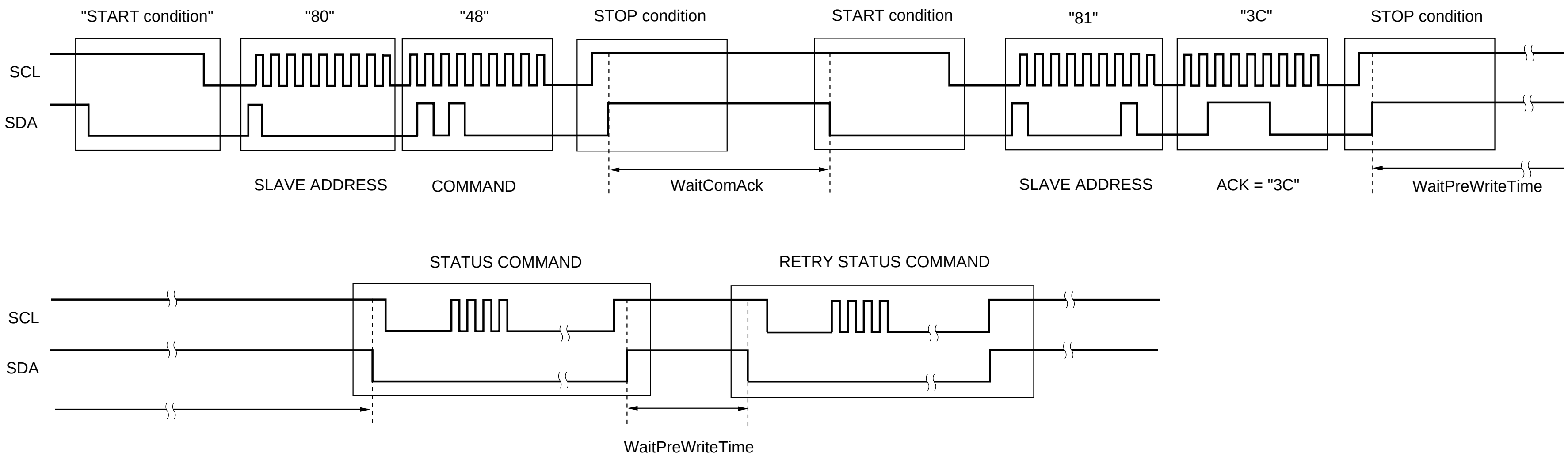


WaitComAck:	The number of wait clocks is at least 1,030/1,240 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.
WaitAckData:	The number of wait clocks is at least 50/640 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.
WaitDataData:	The number of wait clocks is at least 70/530 CPU clocks. The wait time is the time between receiving erase time data (high) and receiving erase time data (low).
WaitEraseTimeCalc:	The number of wait clocks is at least 1,200/20,000 CPU clocks. The wait time is the time used to calculate the erase time setting.

3.2.4 Prewrite command

This command must be used to clear the flash microcontroller's program area (flash memory area) to "00H" to prepare for erasure before the erase command can be used.

Figure 3-16. Timing of Prewrite Command



WaitComAck: The number of wait clocks is at least 1,030/1,240 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

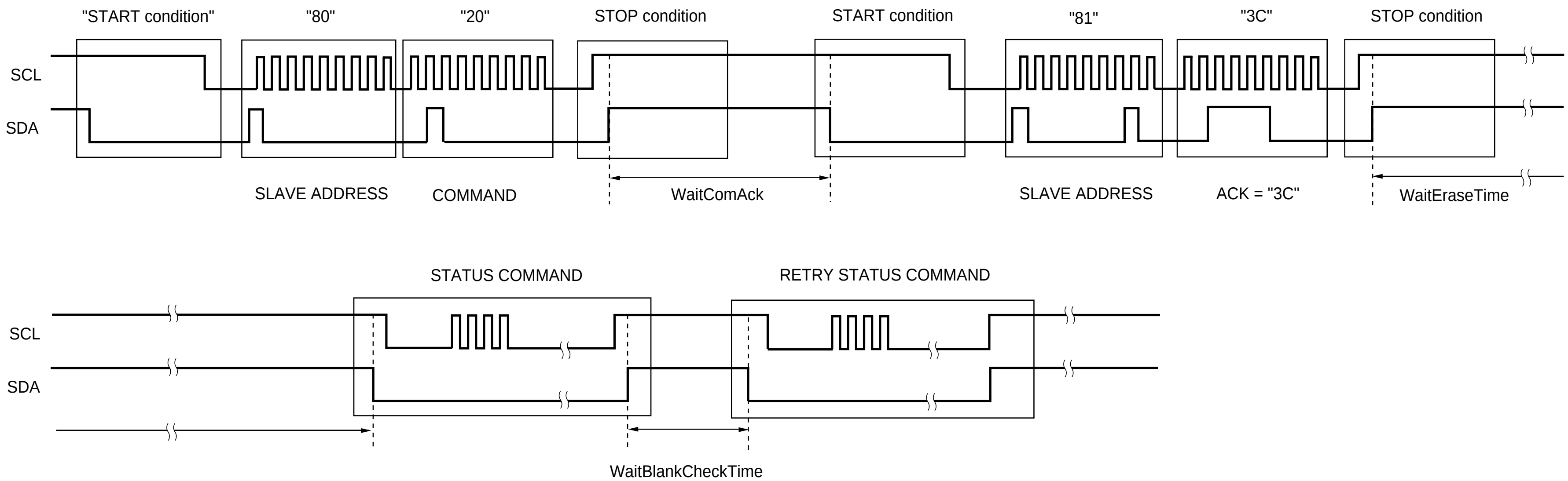
WaitPreWriteTime: The number of wait clocks is at least (230/216 CPU clocks + flash memory write time[Note]) × flash memory capacity (bytes).

Note See **CHAPTER 4 SAMPLE PROGRAMS**.

3.2.5 Erase command

This command erases the flash microcontroller's program area (flash memory).

Figure 3-17. Timing of Erase Command



WaitComAck:	The number of wait clocks is at least 1,030/1,240 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.
WaitEraseTime:	The number of wait clocks is at least the erase time set via the erase time setting command + (690/175 CPU clocks × flash memory capacity (bytes)). The wait time is equal to the erase time.
WaitBlankCheckTime:	The number of wait clocks is at least 690/175 CPU clocks × flash memory capacity (bytes).

3.2.6 Write commands

This command writes data to the flash microcontroller's program area (flash memory). It is used in combination with the status check command to check for write failures while the write operation is in progress.

Figure 3-18. Timing of High-Speed Write Command (1/2)

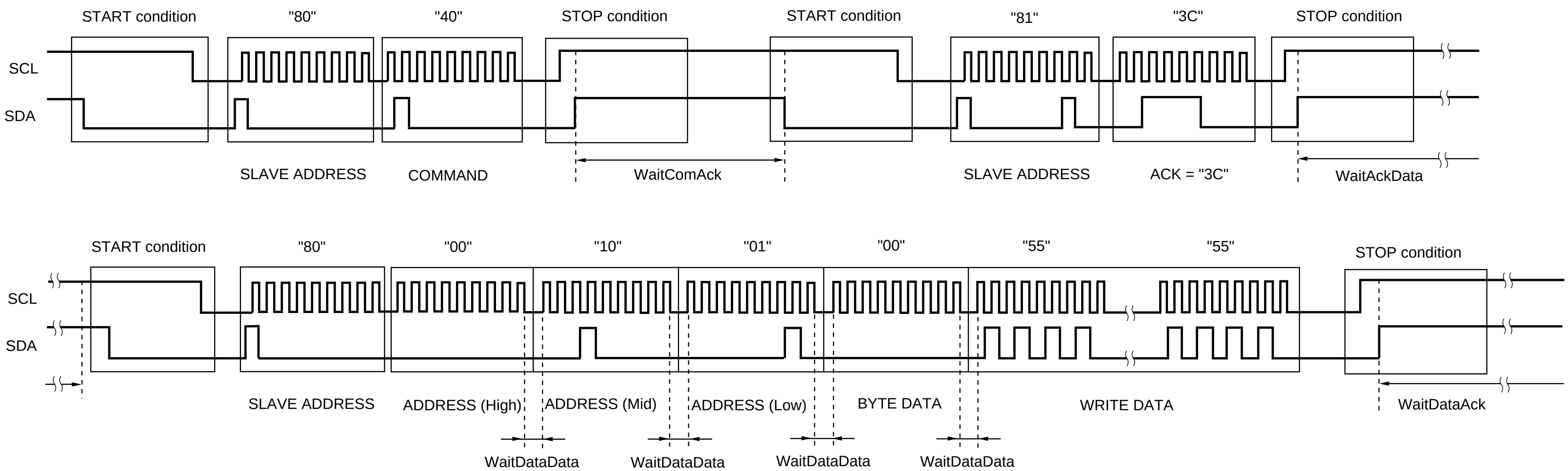


Figure 3-18. Timing of High-Speed Write Command (2/2)

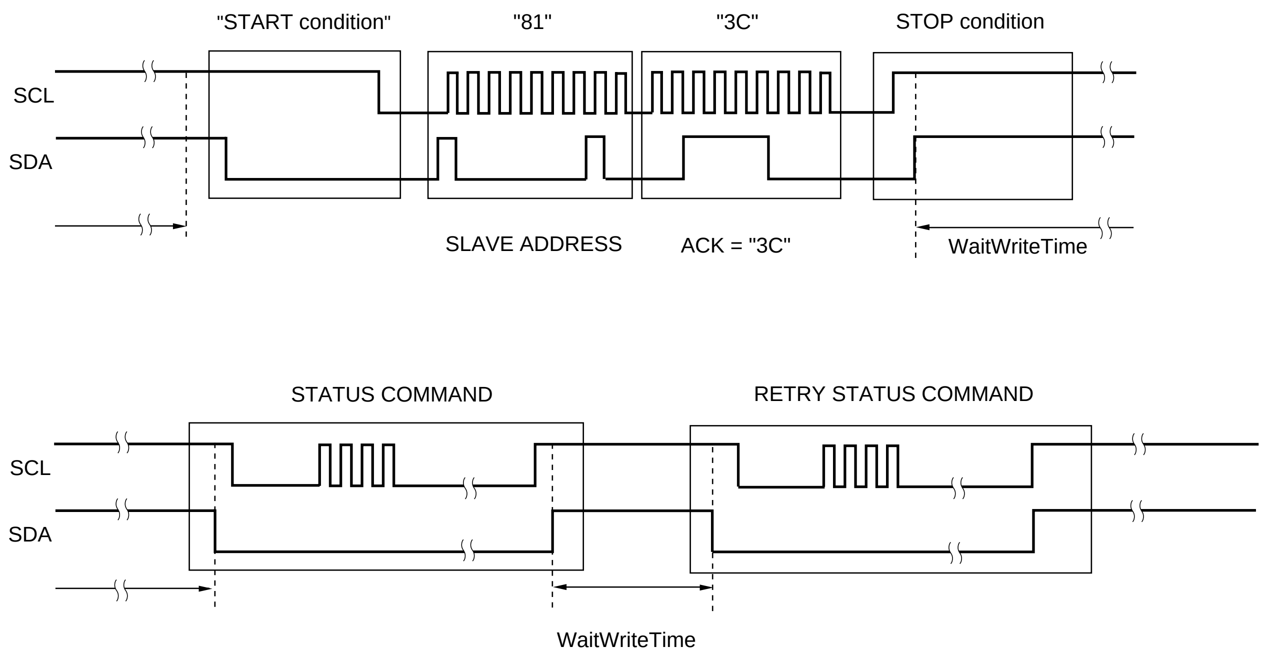


Figure 3-19. Timing of Continuous Write Command (1/2)

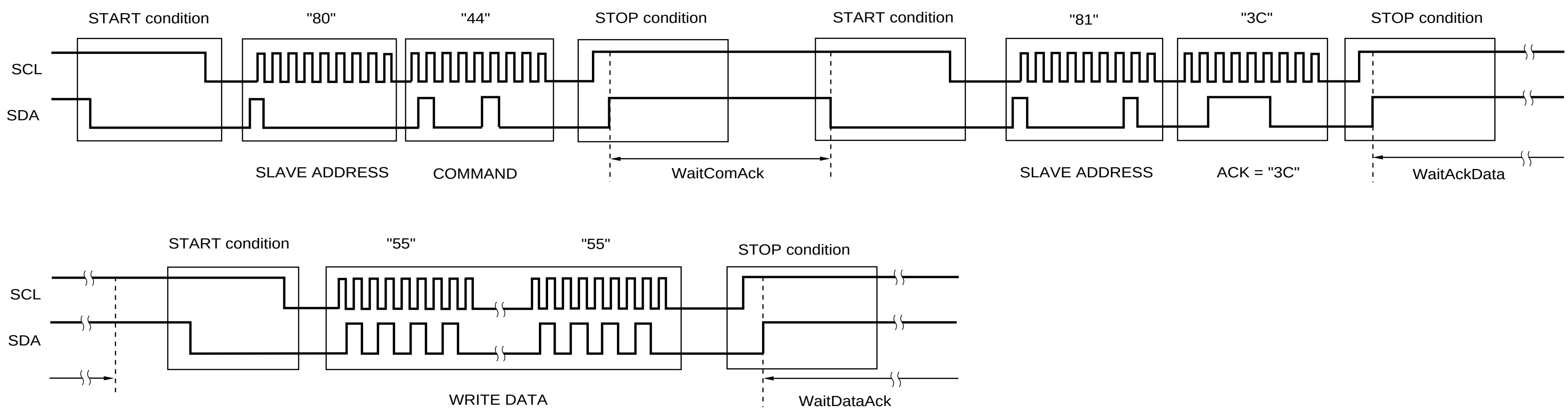
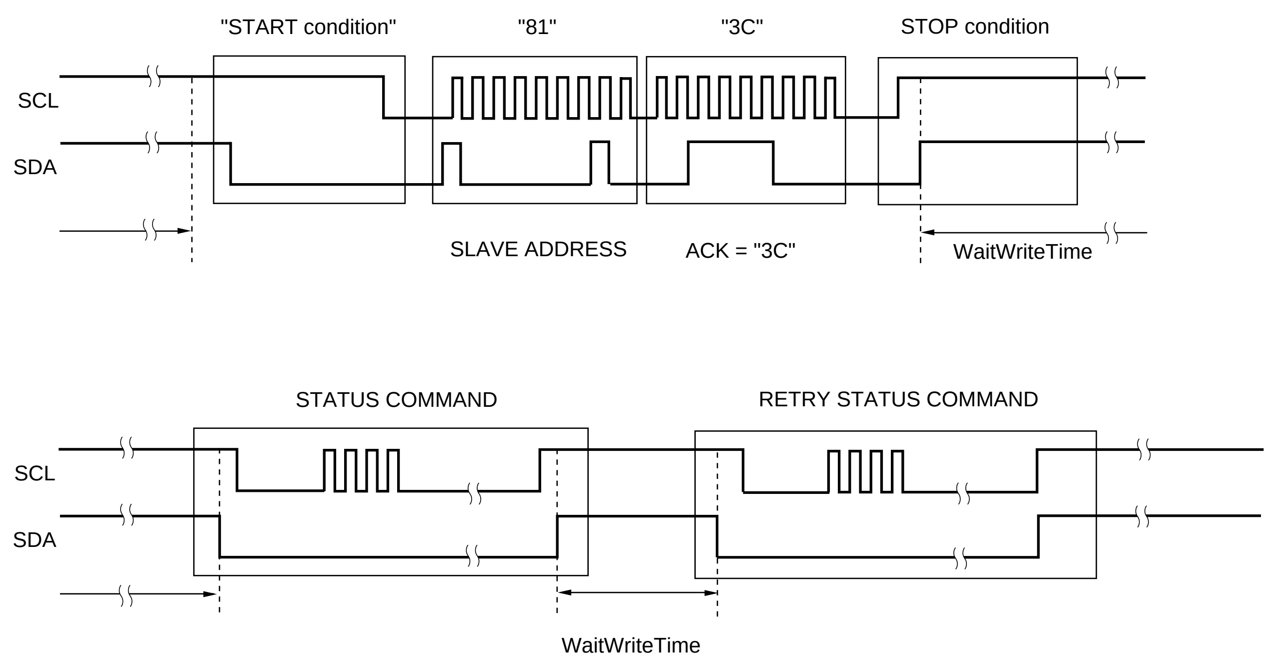


Figure 3-19. Timing of Continuous Write Command (2/2)



WaitComAck: The number of wait clocks is at least 1,030/1,240 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 50/640 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

WaitDataData: The number of wait clocks is at least 70/530 CPU clocks. The wait time is the time between receiving two sets of data.

WaitDataAck: The number of wait clocks is at least 70/530 CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

WaitWriteTime: The number of wait clocks is at least (1,010/275 CPU clocks + flash memory write time[Note 1]) × write data size (bytes)[Note 2].

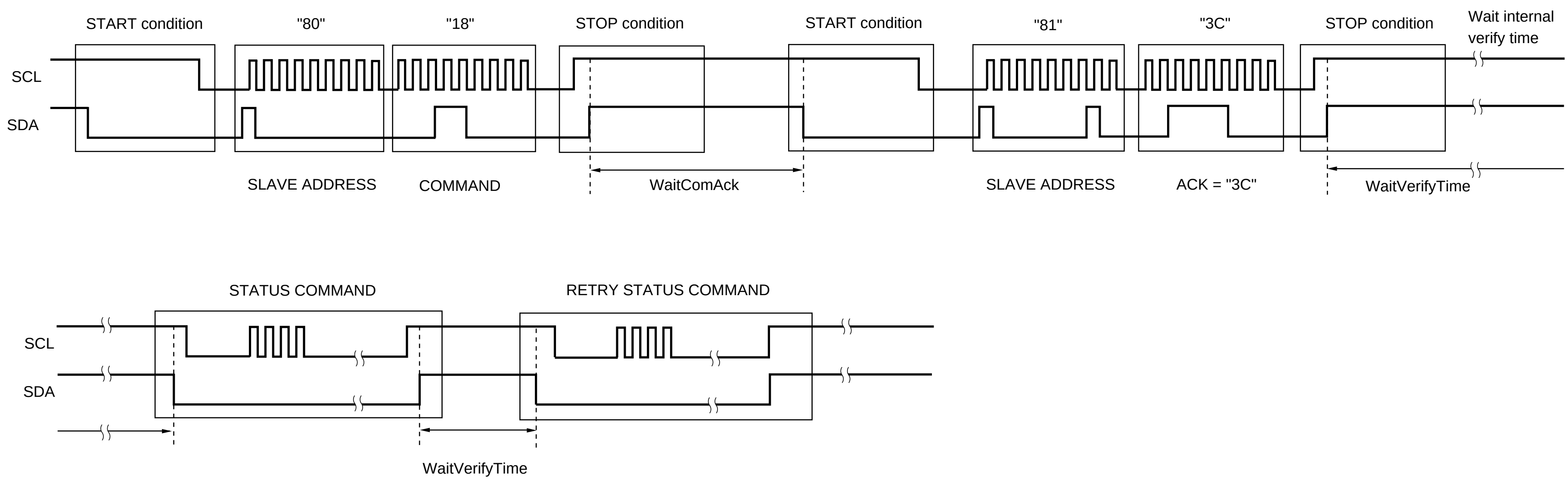
Notes 1. See **CHAPTER 4 SAMPLE PROGRAMS**.

2. Write data size: 1 to 256 bytes (for 78K/0) or 1 to 128 bytes (for 78K/0S)

3.2.7 Internal verify command

This command is used after the write command has been executed to check the depth of the write level.

Figure 3-20. Timing of Internal Verify Command



WaitComAck: The number of wait clocks is at least 1,030/1,240 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitVerifyTime: The number of wait clocks is at least 840/230 CPU clocks $\times$ flash memory capacity (bytes).

3.2.8 Verify command

This command compares the contents of the flash microcontroller's program area (flash memory) with the data received by the flash microcontroller.

Figure 3-21. Timing of Verify Command (1/2)

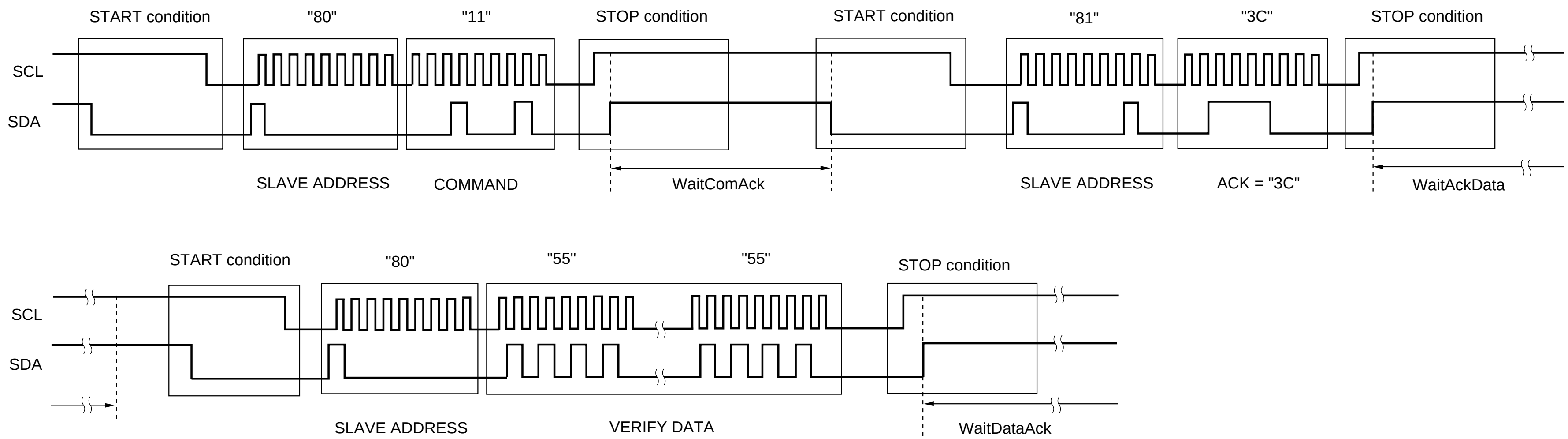
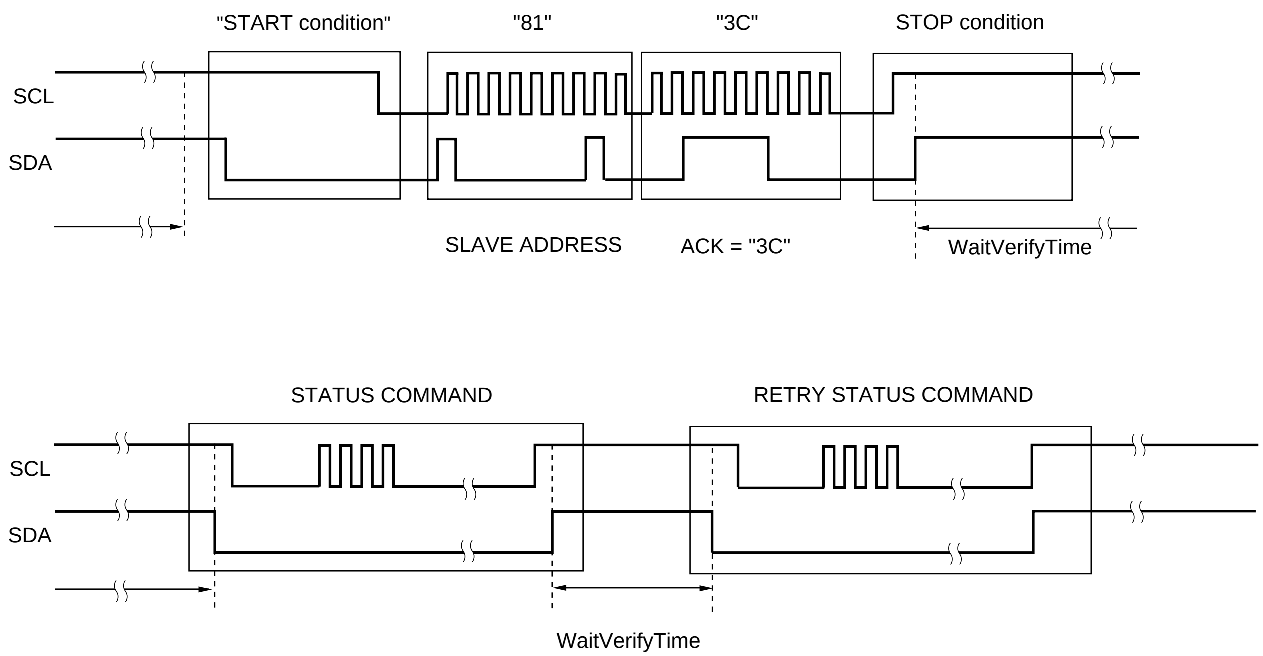


Figure 3-21. Timing of Verify Command (2/2)



WaitComAck: The number of wait clocks is at least 1,030/1,240 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 50/640 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

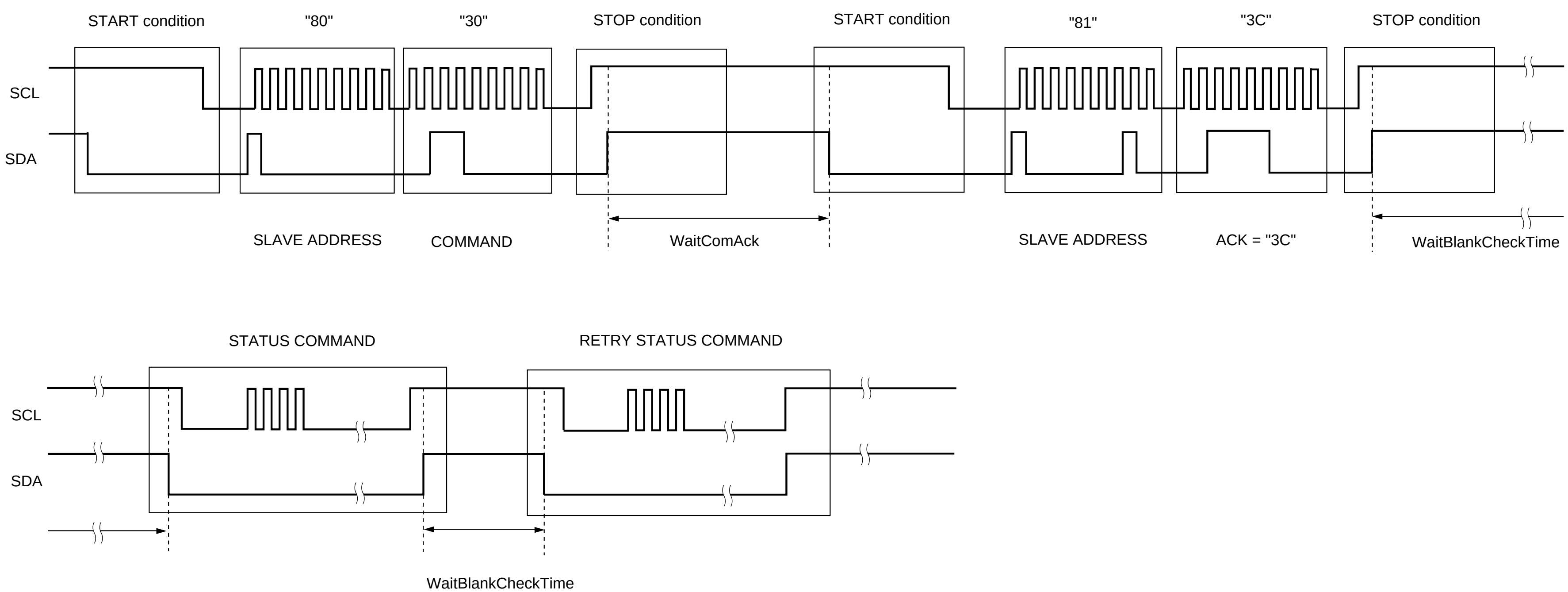
WaitDataAck: The number of wait clocks is at least 70/530 CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

WaitVerifyTime: The number of wait clocks is at least 258,600/29,400 CPU clocks.

3.2.9 Blank check command

This command checks whether or not the flash microcontroller's program area (flash memory) has been erased.

Figure 3-22. Timing of Blank Check Command



WaitComAck: The number of wait clocks is at least 1,030/1,240 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitBlankCheckTime: The number of wait clocks is at least 690/175 CPU clocks × flash memory capacity (bytes).

3.2.10 Silicon signature command

This command gets the flash microcontroller's device information (silicon signature). For description of the silicon signature data, see **Table 2-6. Meaning of Silicon Signature Data**.

Figure 3-23. Timing of Silicon Signature Command (1/2)

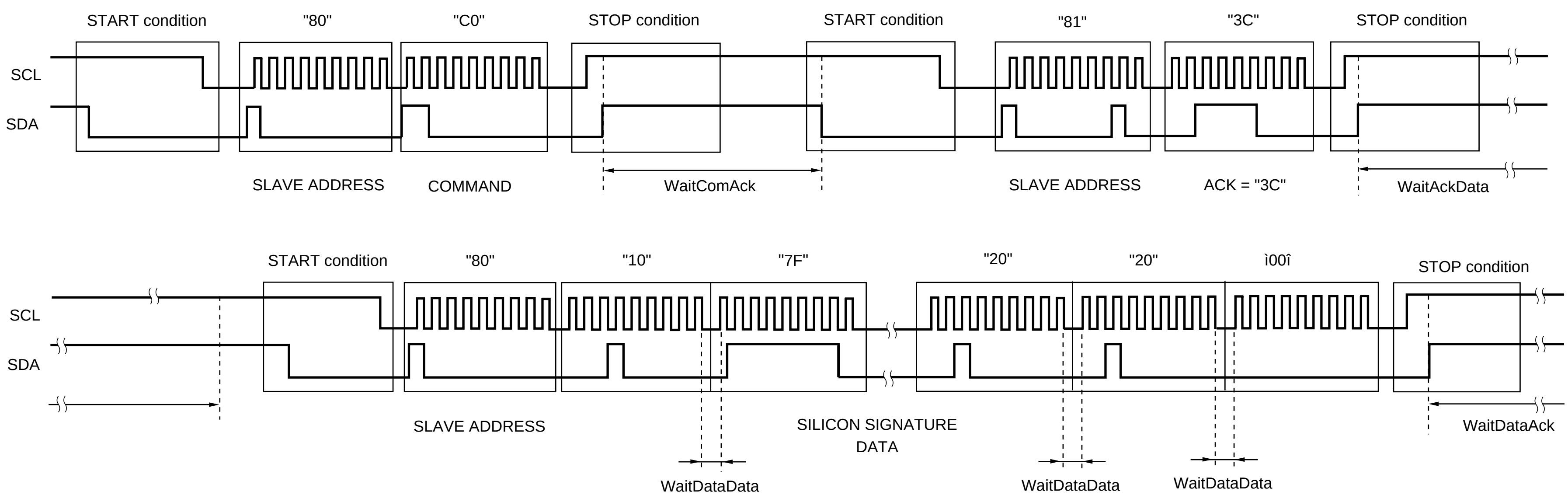
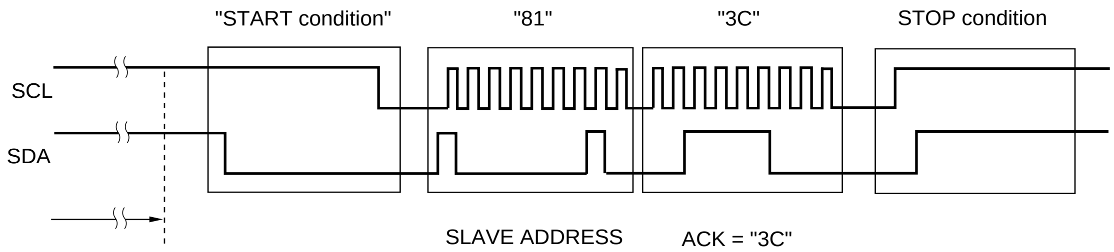


Figure 3-23. Timing of Silicon Signature Command (2/2)



WaitComAck: The number of wait clocks is at least 1,030/1,240 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 50/640 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

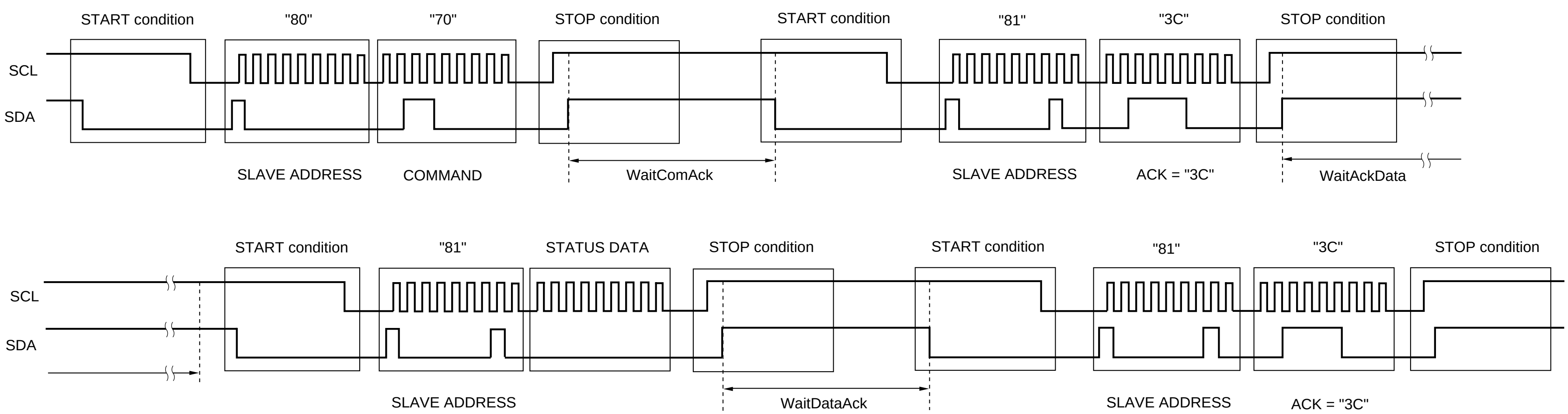
WaitDataData: The number of wait clocks is at least 70/530 CPU clocks. The wait time is the time between receiving two sets of data.

WaitDataAck: The number of wait clocks is at least 70/530 CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

3.2.11 Status check command

This command gets the flash microcontroller's internal command execution status and gets the execution results. The status check command can be executed any number of times after any command is executed. The status data is 8-bit data in which the bits are assigned values indicating the command's execution status and execution results. For description of the status data, see **Table 2-7. Meaning of Status and Data Bits in Status Check Command**.

Figure 3-24. Timing of Status Check Command



WaitComAck: The number of wait clocks is at least 1,030/1,240 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.
WaitAckData: The number of wait clocks is at least 50/640 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.
WaitDataAck: The number of wait clocks is at least 70/530 CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

3.3 Write Sequence for UART Communications

The following are timing charts for each command used during UART communications.

The number of wait clocks indicated below are represented as:

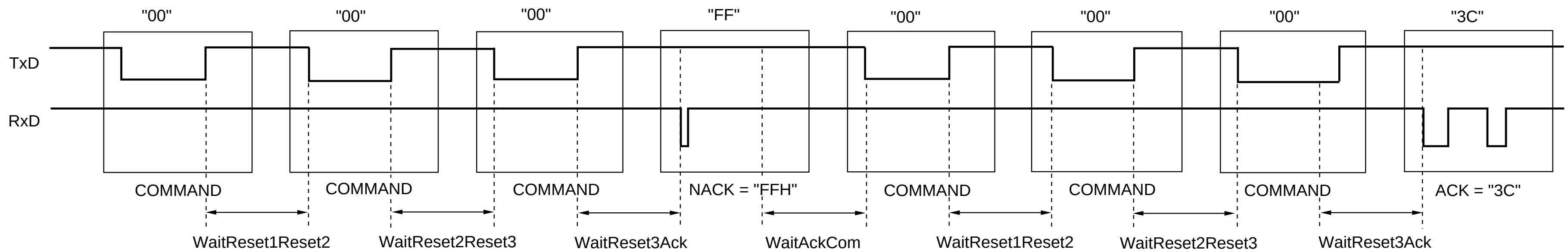
(Number of wait clocks when the target microcontroller is a 78K/0 Series product)/(Number of wait clocks when the target microcontroller is a 78K/0S Series product).

3.3.1 Reset command

This command is used to confirm synchronization detection as part of synchronization detection processing.

The timing of the reset command during synchronization detection is shown below.

Figure 3-25. Timing of Reset Command



WaitComAck: The number of wait clocks is at least 1,870/1,900 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckCom: The number of wait clocks is at least 170/190 CPU clocks. The wait time is the time between receiving an ACK signal and issuing a command.

WaitReset1Reset2: The number of wait clocks is at least 260/320 CPU clocks. The wait time is the time between the first reset command and the second reset command.

WaitReset2Reset3: The number of wait clocks is at least 180/230 CPU clocks. The wait time is the time between the second reset command and the third reset command.

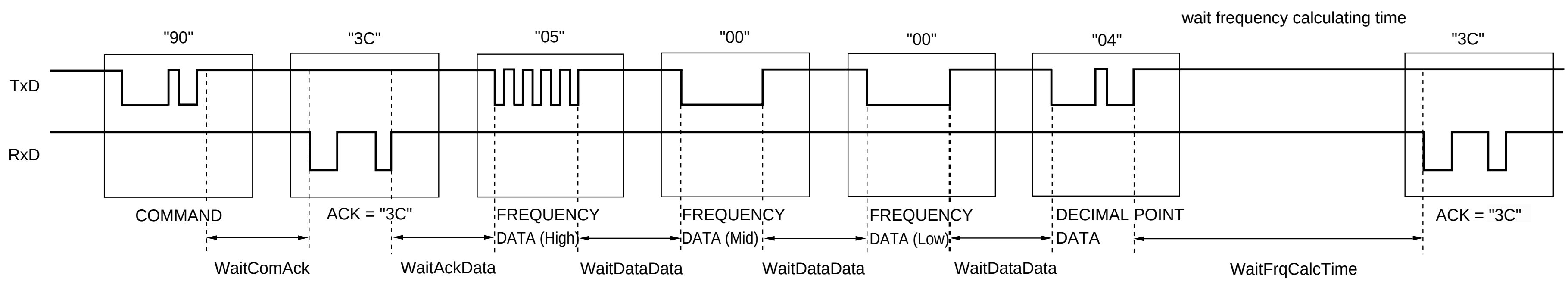
WaitReset3Ack: The number of wait clocks is at least 4,100/14,700 CPU clocks. The wait time is the time between the third reset command and an ACK signal.

Caution Once a NACK signal is returned, retries are performed until an ACK signal is returned. The maximum number of retries is 16. A communication error occurs if 17 or more retries are attempted. For details, see CHAPTER 4 SAMPLE PROGRAMS.

3.3.2 Oscillation frequency setting command

This command notifies the flash microcontroller concerning its operating clock. The flash microcontroller internally uses this frequency value as the basic frequency value for calculating the write time, erase time, etc.

Figure 3-26. Timing of Oscillation Frequency Setting Command

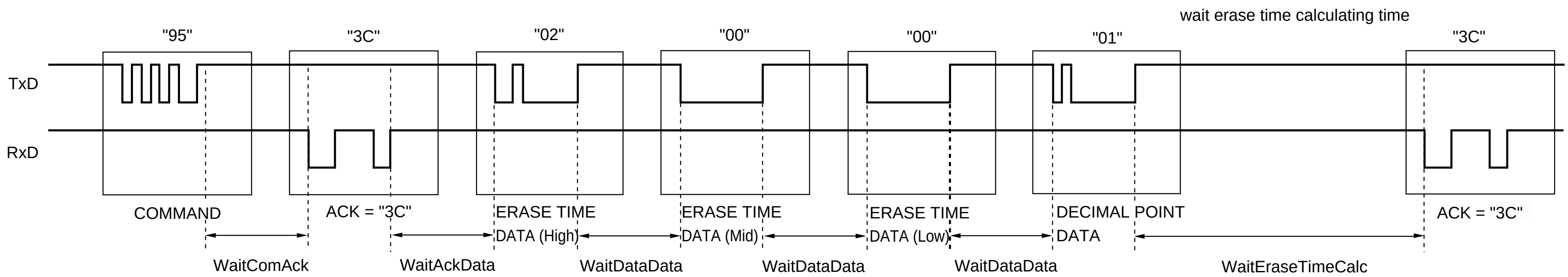


WaitComAck:	The number of wait clocks is at least 1,870/1,900 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.
WaitAckData:	The number of wait clocks is at least 240/180 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.
WaitDataData:	The number of wait clocks is at least 650/690 CPU clocks. The wait time is the time between receiving frequency data (high) and receiving frequency data (low).
WaitFrqCalcTime:	The number of wait clocks is at least 5,260/46,600 CPU clocks. The wait time is the time used to calculate the oscillation frequency setting.

3.3.3 Erase time setting command

This command sets the flash microcontroller's erase time to the flash microcontroller's program area (flash memory).

Figure 3-27. Timing of Erase Time Setting Command



WaitComAck: The number of wait clocks is at least 1,870/1,900 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 240/180 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

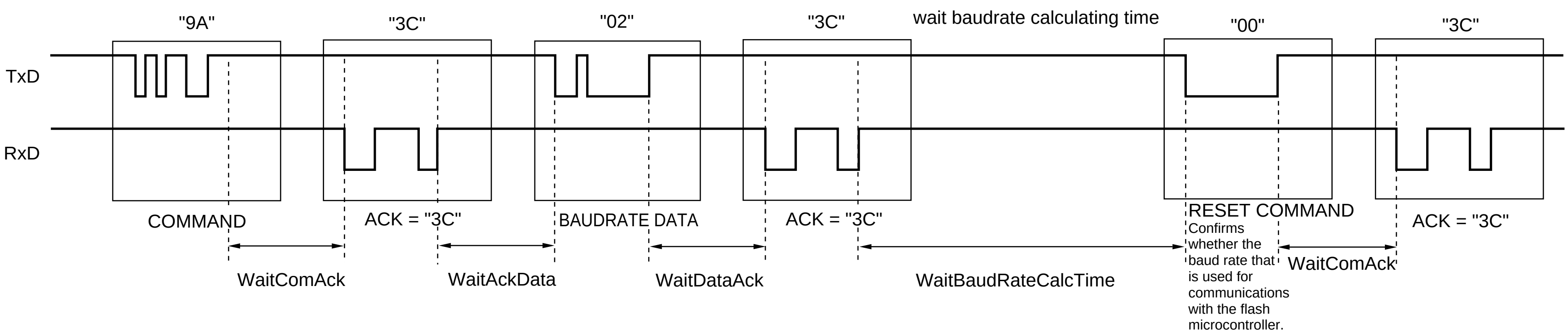
WaitDataData: The number of wait clocks is at least 650/690 CPU clocks. The wait time is the time between receiving erase time data (high) and receiving erase time data (low).

WaitEraseTimeCalc: The number of wait clocks is at least 1,450/276,000 CPU clocks. The wait time is the time used to calculate the erase time setting.

3.3.4 Baud rate setting command

This changes the baud rate that is used for communications with the flash microcontroller.

Figure 3-28. Timing of Baud Rate Setting Command

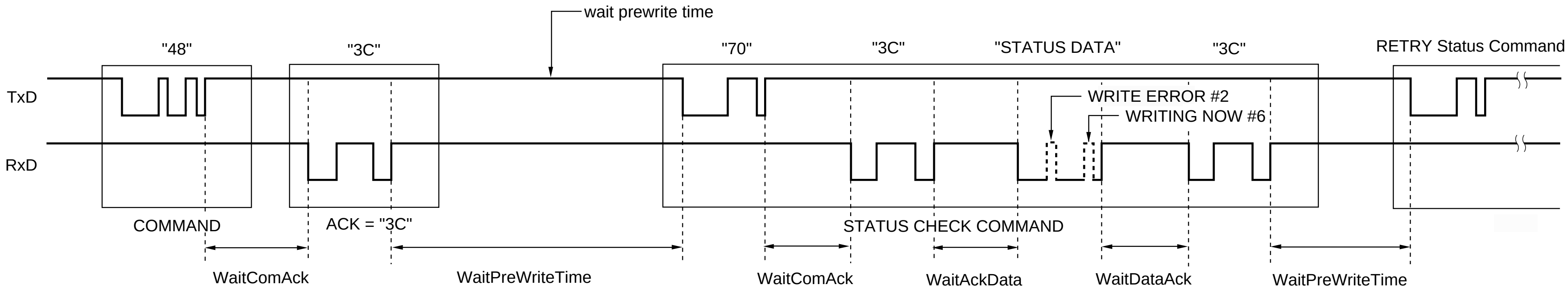


WaitComAck:	The number of wait clocks is at least 1,870/1,900 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.
WaitAckData:	The number of wait clocks is at least 240/180 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.
WaitDataAck:	The number of wait clocks is at least 700/660 CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.
WaitBaudRateCalcTime:	The number of wait clocks is at least 3,820/27,000 CPU clocks. The wait time is the time used to calculate the baud rate setting.

3.3.5 Prewrite command

This command must be used to clear the flash microcontroller's program area (flash memory area) to "00H" to prepare for erasure before the erase command can be used.

Figure 3-29. Timing of Prewrite Command



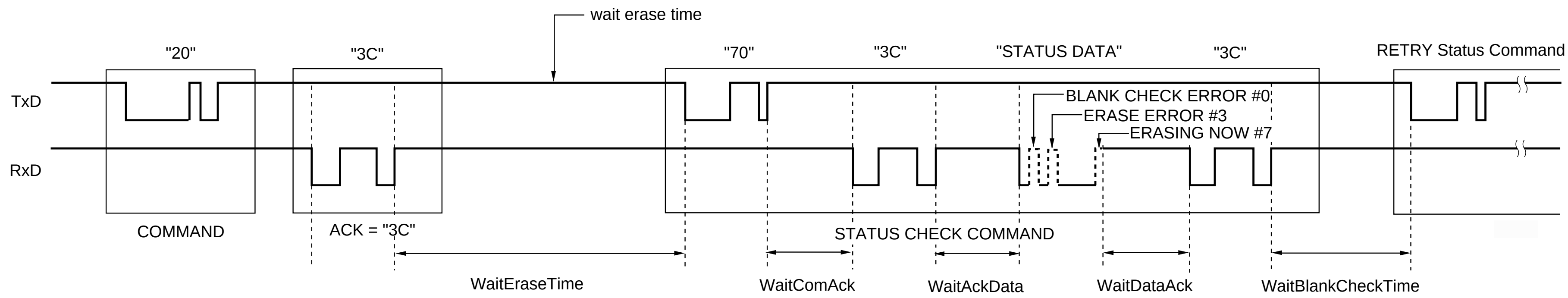
WaitComAck:	The number of wait clocks is at least 1,870/1,900 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.
WaitAckData:	The number of wait clocks is at least 240/180 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.
WaitDataAck:	The number of wait clocks is at least 700/660 CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.
WaitPreWriteTime:	The number of wait clocks is at least (230/216 CPU clocks + flash memory write time[Note]) × flash memory capacity (bytes).

Note See **CHAPTER 4 SAMPLE PROGRAMS**.

3.3.6 Erase command

This command erases the flash microcontroller's program area (flash memory).

Figure 3-30. Timing of Erase Command



WaitComAck: The number of wait clocks is at least 1,870/1,900 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 240/180 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

WaitDataAck: The number of wait clocks is at least 700/660 CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

WaitEraseTime: The number of wait clocks is at least the erase time set via the erase time setting command + (690/175 CPU clocks × flash memory capacity (bytes)). The wait time is equal to the erase time.

WaitBlankCheckTime: The number of wait clocks is at least 690/175 CPU clocks × flash memory capacity (bytes).

3.3.7 Write commands

This command writes data to the flash microcontroller's program area (flash memory). It is used in combination with the status check command to check for write failures while the write operation is in progress.

Figure 3-31. Timing of High-Speed Write Command

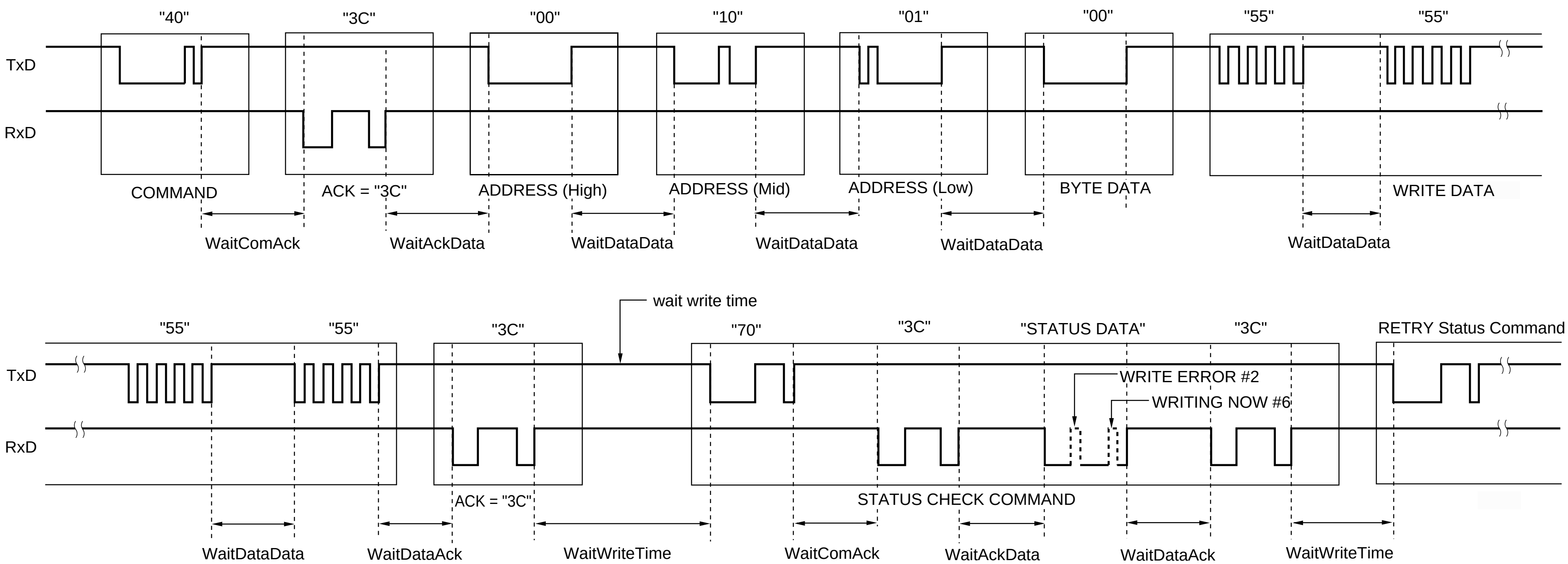
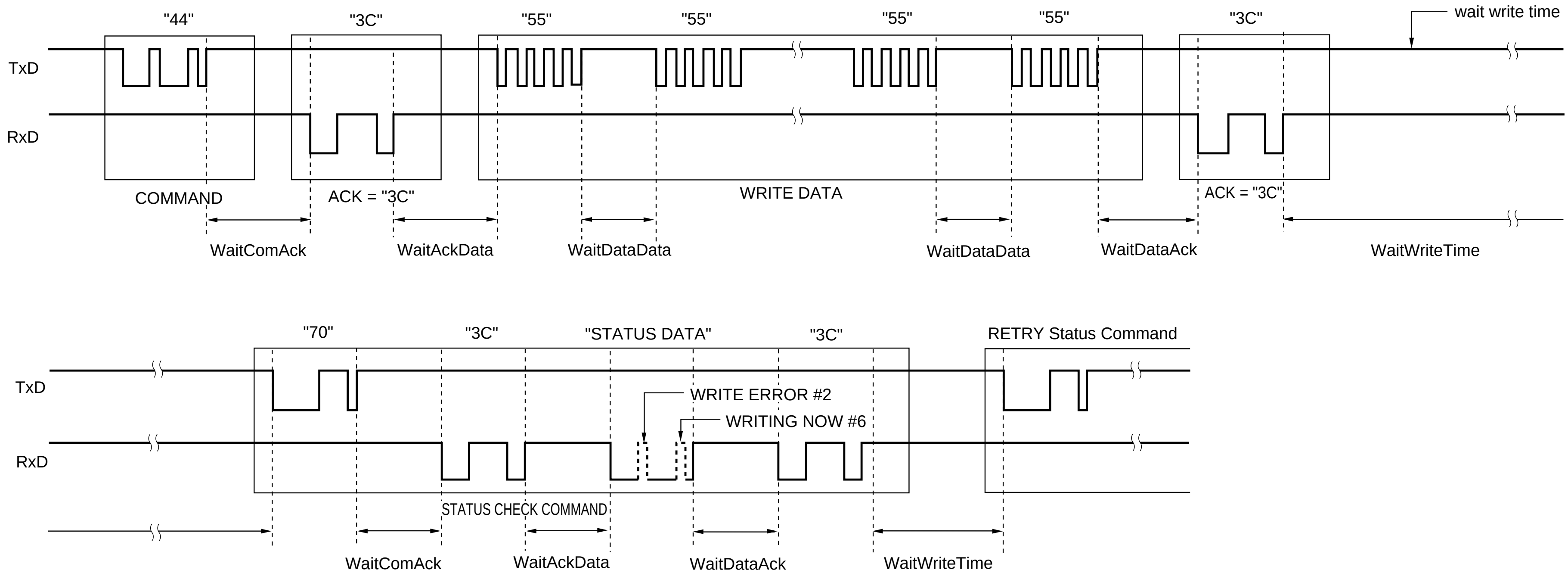


Figure 3-32. Timing of Continuous Write Command



WaitComAck:	The number of wait clocks is at least 1,870/1,900 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.
WaitAckData:	The number of wait clocks is at least 240/180 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.
WaitDataAck:	The number of wait clocks is at least 700/660 CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.
WaitDataData:	The number of wait clocks is at least 650/690 CPU clocks. The wait time is the time between receiving two sets of data.
WaitWriteTime:	The number of wait clocks is at least (1,010/275 CPU clocks + flash memory write time[Note 1]) × write data size (bytes)[Note 2].

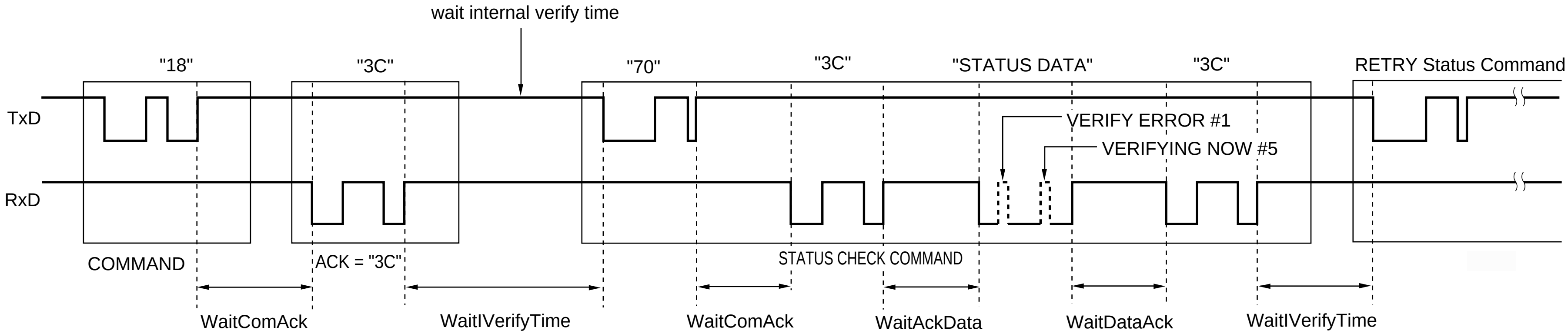
Notes 1. See CHAPTER 4 SAMPLE PROGRAMS.

2. Write data size: 1 to 256 bytes (for 78K/0) or 1 to 128 bytes (for 78K/0S)

3.3.8 Internal verify command

This command is used after the write command has been executed to check the depth of the write level.

Figure 3-33. Timing of Internal Verify Command



WaitComAck: The number of wait clocks is at least 1,870/1,900 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 240/180 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

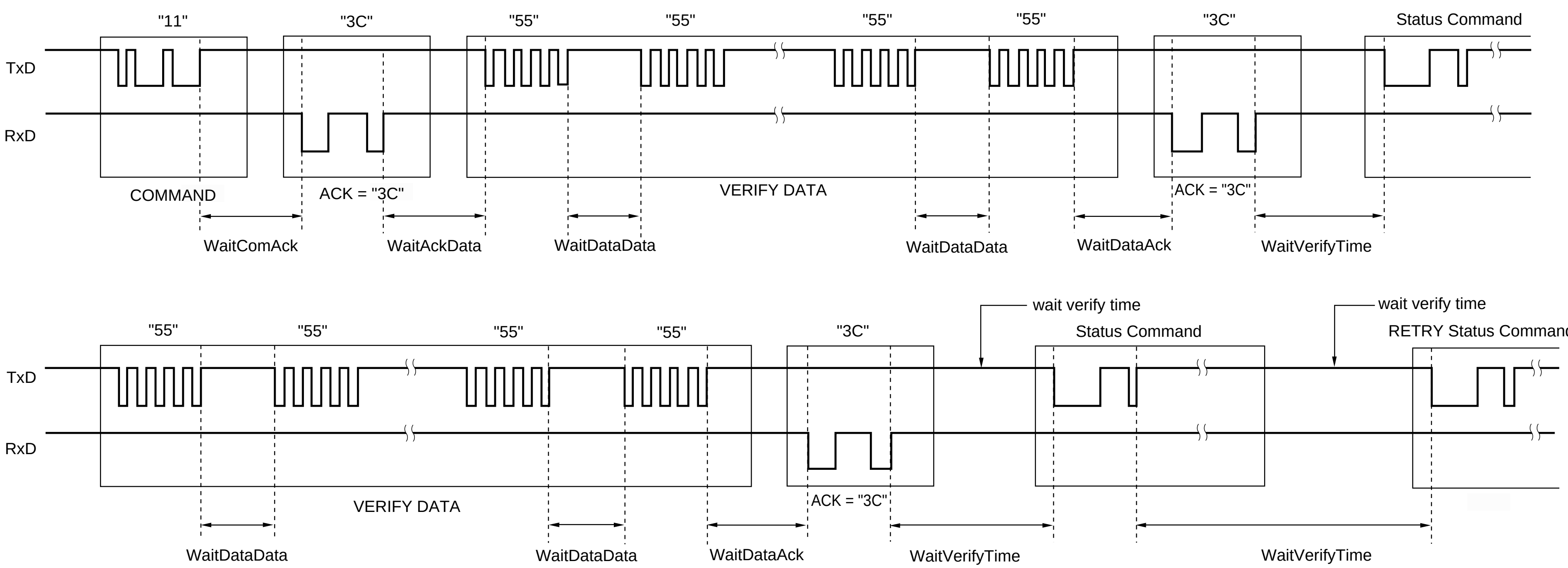
WaitDataAck: The number of wait clocks is at least 700/660 CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

WaitVerifyTime: The number of wait clocks is at least 840/230 CPU clocks × flash memory capacity (bytes).

3.3.9 Verify command

This command compares the contents of the flash microcontroller's program area (flash memory) with the data received by the flash microcontroller.

Figure 3-34. Timing of Verify Command

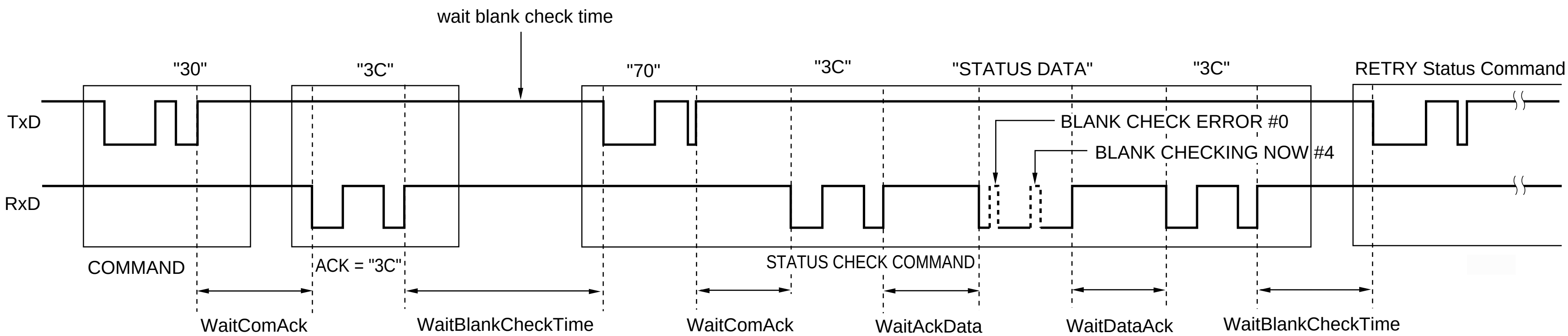


WaitComAck:	The number of wait clocks is at least 1,870/1,900 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.
WaitAckData:	The number of wait clocks is at least 240/180 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.
WaitDataAck:	The number of wait clocks is at least 700/660 CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.
WaitDataData:	The number of wait clocks is at least 650/690 CPU clocks. The wait time is the time between receiving two sets of data.
WaitVerifyTime:	The number of wait clocks is at least 258,560/29,400 CPU clocks.

3.3.10 Blank check command

This command checks whether or not the flash microcontroller's program area (flash memory) has been erased.

Figure 3-35. Timing of Blank Check Command

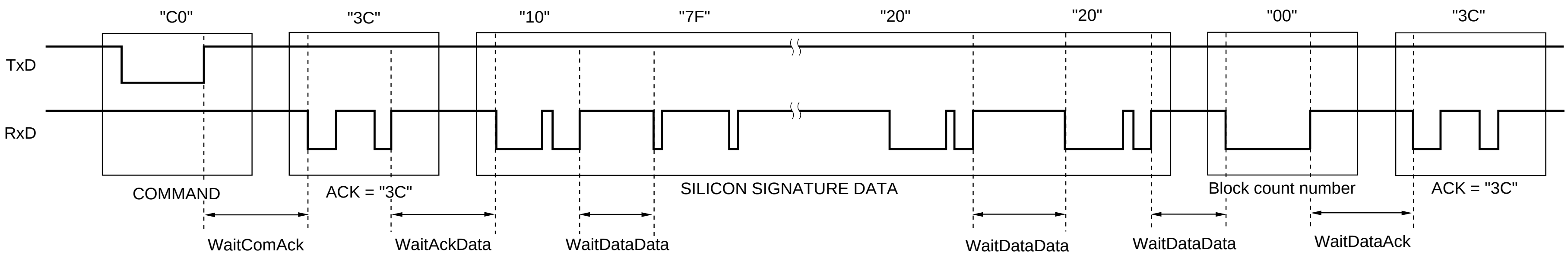


WaitComAck:	The number of wait clocks is at least 1,870/1,900 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.
WaitAckData:	The number of wait clocks is at least 240/180 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.
WaitDataAck:	The number of wait clocks is at least 700/660 CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.
WaitBlankCheckTime:	The number of wait clocks is at least 690/175 CPU clocks × flash memory capacity (bytes).

3.3.11 Silicon signature command

This command gets the flash microcontroller's device information (silicon signature). For description of the silicon signature data, see **Table 2-6. Meaning of Silicon Signature Data**.

Figure 3-36. Timing of Silicon Signature Command



WaitComAck: The number of wait clocks is at least 1,870/1,900 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.

WaitAckData: The number of wait clocks is at least 240/180 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.

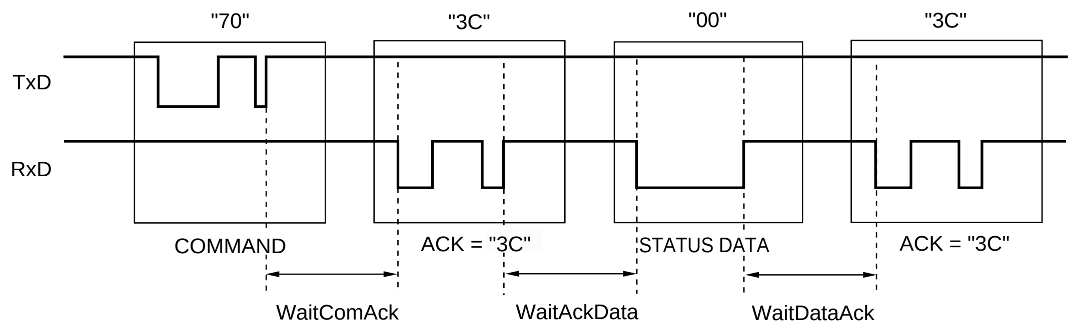
WaitDataAck: The number of wait clocks is at least 700/660 CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

WaitDataData: The number of wait clocks is at least 650/690 CPU clocks. The wait time is the time between receiving two sets of data.

3.3.12 Status check command

This command gets the flash microcontroller's internal command execution status and also gets the command execution results. The status check command can be executed any number of times after any command is executed. The status data is 8-bit data in which values indicating the command's execution status and execution results are assigned to each bit. For description of the status data, see **Table 2-7. Meaning of Status and Data bits in Status Check Command**.

Figure 3-37. Timing of Status Check Command



WaitComAck: The number of wait clocks is at least 1,870/1,900 CPU clocks. The wait time is the time between issuing a command and receiving an ACK signal.
WaitAckData: The number of wait clocks is at least 240/180 CPU clocks. The wait time is the time between receiving an ACK signal and receiving data.
WaitDataAck: The number of wait clocks is at least 700/660 CPU clocks. The wait time is the time between receiving data and receiving an ACK signal.

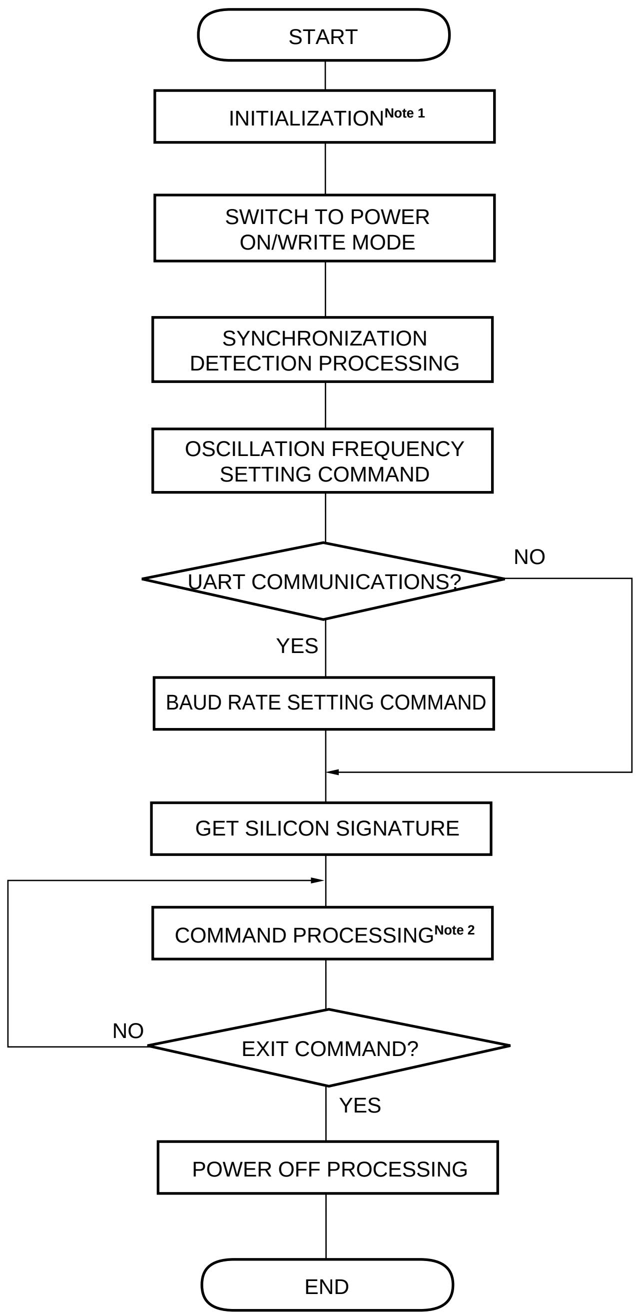
CHAPTER 4 SAMPLE PROGRAMS

This chapter describes an example of software that writes to the flash microcontroller using a μPD784038Y as the flash programmer's controller.

4.1 Description of Configuration for Processing

The overall program processing flow when developing an actual program is shown below.

Figure 4-1. Overall Flow of Program



Note 1. This initializes the I/O ports and sets parameters.

Note 2. This refers to the next command (one of the following)

- Erase command
- Write command
- Verify command
- Blank check command
- Prewrite command
- Internal verify command

4.2 Description of ROM

Table 4-1 lists the areas of ROM usage.

Table 4-1. ROM Map

ROM Address	Size (Bytes)	Description
000000H to 00003FH	40H	Vector entry table
000040H to 00007FH	40H	CALLT table area (not used)
000080H to 000121H	A2H	Startup routine
000122H to 000261H	320 bytes	ROM table (name of table: WaitDataTable) This stores the number of wait clocks during the flash microcontroller's processing.
000262H to 0007FFH	59EH	Unused area
000800H to 001EAEH	16AFH	Area for processing commands and subroutines
001EAFH to 00EDFFH	CF51H	Unused area
20000H to 2FFFFH	64 Kbytes	External memory area In the sample programs, this area is allocated in external ROM (64 Kbytes) and is used to store data to be written to the flash microcontroller.

4.3 Description of RAM

The RAM areas are described in Table 4-2 below. For description of constant values that are stored in RAM, see **4.6.5 List of constant values**. Also, see the corresponding source file for description of variables (local variables) that are used only within modules.

Table 4-2. RAM Specifications (1/3)

RAM Name	Address	Used Bytes	Name	Description
aSendBuffer[ ]	EE00H	256	Send buffer	This is the area where data is stored when being transmitted to the flash microcontroller.
aRecieveBuffer[ ]	EF00H	256	Receive buffer	This is the area where data received from the flash microcontroller is stored.
STBEG	F000H	3,360	Stack area	The stack is used to temporarily store addresses and register data during subroutine calls, etc.
dwParStartAddress	FD20H	4	Write start address	This is the flash microcontroller's write start address. When using the high-speed write command, the low-order three bytes of data are sent to the flash microcontroller as the write start address.

Table 4-2. RAM Specifications (2/3)

RAM Name	Address	Used Bytes	Name	Description
dwParEndAddress	FD24H	4	Write end address	This is the flash microcontroller's write end address.
wParCpuClockSpeed	FD28H	2	CPU clock speed	This is the flash microcontroller's operating clock. A clock value is stored in 10-kHz units, then data is processed and transferred to the flash microcontroller. This clock value is used to calculate the flash microcontroller's processing wait time.
wParCsiClockSpeed	FD2AH	2	CSI communication clock speed	This is the communications clock speed for 3-wire serial or pseudo 3-wire serial communications. Data is stored at a frequency of 100 Hz.
wParEraseTime	FD2CH	2	Erase time	This is the flash microcontroller's erase time. The erase time is stored using 10-ms units.
cParTargetSeries	FD2EH	1	Target series	This is used to determine the series (78K/0 or 78K/0S) of the microcontroller that will perform the write operation.
cParVppPulse	FD2FH	1	V_{PP} pulse count	This stores the number of V_{PP} pulses to be sent to the flash microcontroller. The communication method is selected based on the contents of this area.
cParBaudRate	FD30H	1	Baud rate select	This area stores data used to change the baud rate for UART communications.
cParCpuClockSource	FD31H	1	CPU clock source	This selects the source for supplying the flash microcontroller's operating clock.
cParSlaveAddress	FD32H	1	Slave address data	This area stores the flash microcontroller's slave address used during IIC communications.
wSendSize	FD33H	2	Send size	This area stores the size of the data to be sent to the flash microcontroller.
cCommunicationMethod	FD35H	1	Communication method	The value set to this area is determined based on the value stored in cParVppPulse. The communication method to be used with the flash microcontroller is stored as data and is used for branch decisions in the program's processing that depends on a specified communication method.
cSendData	FD36H	1	Send data	This area stores the data to be sent to the flash microcontroller.
cRecieveData	FD37H	1	Receive data	This area stores the data to be received from the flash microcontroller.
cSendFlag	FD38H	1	Send flag	This flag is used when sending data to the flash microcontroller.
cRecieveFlag	FD39H	1	Receive flag	This flag is used when receiving data from the flash microcontroller.
wWaitTimeVppCom	FD3AH	2	V_{PP}-COM wait time	This area stores the amount of wait time required between outputting a V_{PP} pulse and sending a command.
wWaitTimeComAck	FD3CH	2	COM-ACK wait time	This area stores the amount of wait time required between sending a command and receiving an ACK signal.
wWaitTimeAckCom	FD3EH	2	ACK-COM wait time	This area stores the amount of wait time required between receiving an ACK signal and sending a command.

Table 4-2. RAM Specifications (3/3)

RAM Name		Address	Used Bytes	Name	Description
wWaitTimeAckData		FD40H	2	ACK-DAT wait time	This area stores the amount of wait time required between receiving an ACK signal and sending data.
wWaitTimeDataData		FD42H	2	DAT-DAT wait time	This area stores the amount of wait time required between sending two sets of data.
wWaitTimeDataAck		FD44H	2	DAT-ACK wait time	This area stores the amount of wait time required between sending data and receiving an ACK signal.
cTargetStatus		FD46H	1	Target status	This area stores the flash microcontroller's command execution status that is received via the status command.
cRetryCounter		FD47H	1	Retry counter	This counter counts the number of retries by each module.
cErrorStatus		FD48H	1	Error status	This area stores an indicator of the type of error that has occurred while executing a command (when "0" is stored in this area, it means there is no error).
cEnterCommand		FD49H	1	Enter command	This area stores a command that is captured via a SW (in the sample programs, this area is used since certain commands have been determined to be commands that are captured by SW and executed). This area is used only for the "main" and "MGetCom" modules.
cTimerFlag		FD4AH	1	Timer flag	This flag provides notification of the start of wait processing. It also provides notification when waiting processing is finished.
cWaitClockSelect		FD4BH	1	Wait clock select	The value in this area is set based on the values in the cParTargetSeries area and cParVppPulse area. This area is used to select the number of wait clocks used to calculate the wait time for executing various commands that differ according to the target series and communication method.
sSig	cSigVendorCode	FD4CH	1	Vendor code	This area stores the silicon signature data's "vendor code".
	cSigIdCode	FD42H	1	ID code	This area stores the silicon signature data's "ID code".
	cSigElectInf	FD43H	1	Electrical information	This area stores the silicon signature data's "electrical information".
	dwSigLastAddress	FD44H	4	Flash end address	This area stores the silicon signature data's "flash end address".
	aSigDeviceName[]	FD48H	10	Device name	This area stores the silicon signature data's "device name".
	cSigBlockInf	FD52H	1	Block division information	This area stores the silicon signature data's "block division information".

4.3.1 Nomenclature related to processing and RAM

In the program description for this system, the following process names and RAM names are used to improve operational efficiency. The meanings of these names are explained below.

(1) Names of modules and subroutines

M****: Module that combines various processing, such as power supply processing and command processing.

S****: Module that is called from another module as a subroutine.

(2) Definitions of RAM and ROM data types

The following three data type definitions are used for RAM and ROM data types in programs. The definition for each data type is stored in the file "DatType.h". Therefore, when using these sample programs, the DatType.h must be included. For further description of DatType.h, see **4.3.2 Data type definition file**.

Byte: Defines RAM and ROM as one byte of unsigned data.
Word: Defines RAM and ROM as two bytes of unsigned data.
DWord: Defines RAM and ROM as four bytes of unsigned data.

(3) RAM and ROM names

The size or data type of an area is indicated by one or two lower-case letters added to the start of the RAM name or ROM name.

c***:	Area whose size is one byte	Example: cParTargetSeries
w***:	Area whose size is two bytes	Example: wParCpuClockSpeed
dw***:	Area whose size is four bytes	Example: dwParStartAddress
a***:	Area defined as an array	Example: aSendBuffer[256]
s***:	Area defined as a structure	Example: sSig

The characters that follow the one or two characters (in the examples above: c, w, dw, a, and s) that indicate the area size or data type describe the area itself.

*Par***: Name of an area that is used as a parameter (a user-defined value)
Examples: cParTargetSeries, wParEraseTime, dwParEndAddress

sSig.***: Name of an area that is used to store silicon signature information
Examples: sSig.cSigVendorCode, sSig.aSigDeviceName

*Wait***: Name of an area that is used to store the number of wait clocks, wait time, or other wait-related data
Examples: cWaitClockSelect, wWaitTimeComAck

4.3.2 Data type definition file

Be sure to include the following declarations when using the sample programs.

```
typedef unsigned char   Byte;    // Defined as an unsigned one-byte area.
typedef unsigned short  Word;    // Defined as an unsigned two-byte area.
typedef unsigned long   DWord;   // Defined as an unsigned four-byte area.
```

4.4 Description of Modules

The processing corresponding to each module is described in Table 4-3 below.

Table 4-3. Description of Modules (1/2)

Module Name	Full Name	Description
cstartrn	Startup routine	The startup routine branches to the main routine (main) after performing hardware settings, RAM clearing, I/O port settings, etc.
hdwinit	Hardware initialization	This module is called from the startup routine to perform hardware settings and I/O port settings.
main	Main routine	This is the flash programmer's main module. Commands to be executed are controlled by this module.
RamIni	Variable initialization	This module sets hardware settings for variables to be used as parameters and sets other variables that are used by programs.
MGetCom	Enter command	This module selects the command to be executed after input via a SW and stores the command code to be executed in cEnterCommand. Also, if an error occurred during execution of the previous command, an LED indicates the error.
MPowerOn	Switch to power on/write mode	This module switches the flash microcontroller to power on mode and write mode.
MSyncChek	Synchronization detection	This module uses the reset command to check synchronization of the communication with the flash microcontroller.
MFrequencySetUp	Oscillation frequency setting	This module sets the flash microcontroller's operating clock to the flash microcontroller.
MEraseTimeSetUp	Erase time setting	This module sets the flash microcontroller's erase time to the flash microcontroller.
MBaudRate	Baud rate setting	This module changes the baud rate for UART communications. This module is called from the main routine only when UART has been selected as the communication method to be used with the flash microcontroller.
MGetSiliconeSignature	Get silicon signature	This module fetches the flash microcontroller's silicon signature (device information).
MPreWrite	Prewrite	This module is called from MErase and performs prewrite in preparation for erasure.
MErase	Erase	This module erases the flash microcontroller's program area (flash memory).
MProgram	Write	This module writes data to the flash microcontroller's program area (flash memory).
MInternalVerify	Internal verify	This module is called from Mprogram and is used to check the depth of the write level in the flash microcontroller's program area (flash memory) after a write operation.
MVerify	Verify	This module compares the contents of the flash microcontroller's program area (flash memory) with the data received by the flash microcontroller.
MBlankChek	Blank check	This module checks whether or not the flash microcontroller's program area (flash memory) has been erased.
MGetStatus	Get status	This module gets the flash microcontroller's internal command execution status.
MPowerOff	Power off processing	This module turns off the power to the flash microcontroller.
SWaitMicroSec	Microsecond wait	This module inserts a wait period in 1-μs units during processing.

Table 4-3. Description of Modules (2/2)

Module Name	Full Name	Description
SWaitMiliSec	Millisecond wait	This module inserts a wait period in 1-ms units. Meanwhile, the cTimerFlag module can be used to perform other processing during the wait time.
SWaitTimeCalcFlMemSize	Calculate wait time per flash memory size	This module calculates the wait time for the blank check and internal verify operations.
SWriteWaitTimeCalc	Calculate write/prewrite time	This module calculates the write wait time or the prewrite wait time.
SWait	Wait processing	This module inserts a wait period in 1-ms units.
SWait30us	30-μs wait	This module inserts a 30-μs wait period.
SByteDataSend	Send single-byte data	This module sends one byte of data.
SDataSend	Send data	This module uses the SByteDataSend module to send the specified number of data bytes.
SSlaveAddressSend	Send slave address	This module sends slave addressees.
SbyteDataReceive	Receive single-byte data	This module receives one byte of data.
SdataReceive	Receive data	This module uses the SByteDataReceive module to receive the specified number of data bytes.
ScsiIni	Set 3-wire serial/pseudo 3-wire serial communications	This module sets either 3-wire serial or pseudo 3-wire serial communications.
SIicIni	Set IIC communications	This module sets IIC communications.
SUartIni	Set UART communications	This module sets UART communications.

4.5 Sample Programs

(1) In the following sample programs, the μPD78P4038Y is used as the flash programmer's controller.

(2) In these sample programs, compiler or assembler options specify "LOCATION0" as the location and the "large model" as the memory model. During actual coding, "LOCATION0" must be specified as the location and "large model" must be specified as the memory model.

(3) The constant value definitions stored in the program's subroutines, variables (RAM), and ROM table are listed in **4.6.5 List of constant value definitions**.

(4) In these sample programs, the "MGetCom" module[Note 1] performs a key scan and stores the code for the command (erase, write, verify, or erase, write, & verify) that is executed for the variable "cEnterCommand"[Note 2] and selects the command that determines and executes the contents of cEnterCommand in the main routine. During actual coding, we recommend modifying the main routine to suit the interface to be used.

(5) Although an error may be detected in any of these modules, if an error is detected in a sample program, the variable "cErrorStatus" stores the error description and the "MGetCom" module[Note 1] uses an LED to indicate the error. During actual coding, we recommend performing error notification in a way that suits the interface to be used. For a list of errors that may be detected in these sample programs, see **4.7 Error Code List**. Also, the processing that sets the type of error to the "cErrorStatus" variable and the part that determines the contents of cErrorStatus and branches processing are not shown in the sample program flow chart.

Notes 1. See **CHAPTER 5 INTERFACE EXAMPLES** for description of the MGetCom module. Since MGetCom is a module that performs processing of the interface part (key scan and LED display) in these sample programs, it does not affect flash memory write operations. Consequently, when the interface has been changed, there is no need to call MGetCom from the main routine or from another module.

2. cEnterCommand is not used in any modules other than main (the main routine) and MGetCom. Accordingly, there is no need to use this variable when the main routine or interface has been changed.

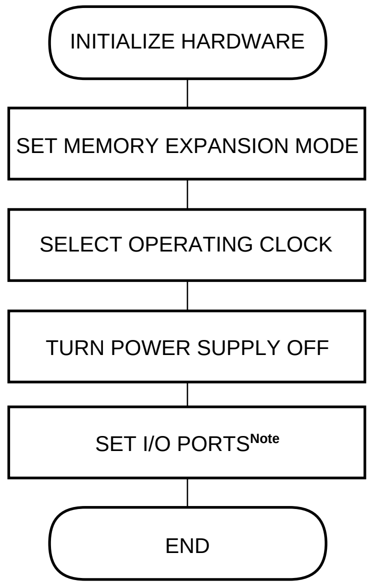
4.5.1 Startup routine

In the following sample programs, the file "cStartrn.asm" which is supplied with the C compiler (CC78K4) is used in the startup routine. When referring to these sample programs, you must use cStartrn.asm. In the startup routine, processing branches to the main routine (main) after the hardware settings, I/O port settings, and clearing of RAM have been performed. The hardware settings and I/O port settings are performed by calling the function "hdwinit" from within the startup routine. The startup routine (cStartrn.asm) can be used without modification if the function name for the hardware settings has been set as "hdwinit" and the function name "main" is used as the main routine. In the following sample programs, the C compiler (CC78K4)'s default file (cStartrn.asm) is used without modification.

4.5.2 Hardware initialization processing

This processing initializes the flash programmer's hardware and set I/O ports.

(1) Flow chart



Note Set the ports to suit the system (peripheral circuit) to be used.

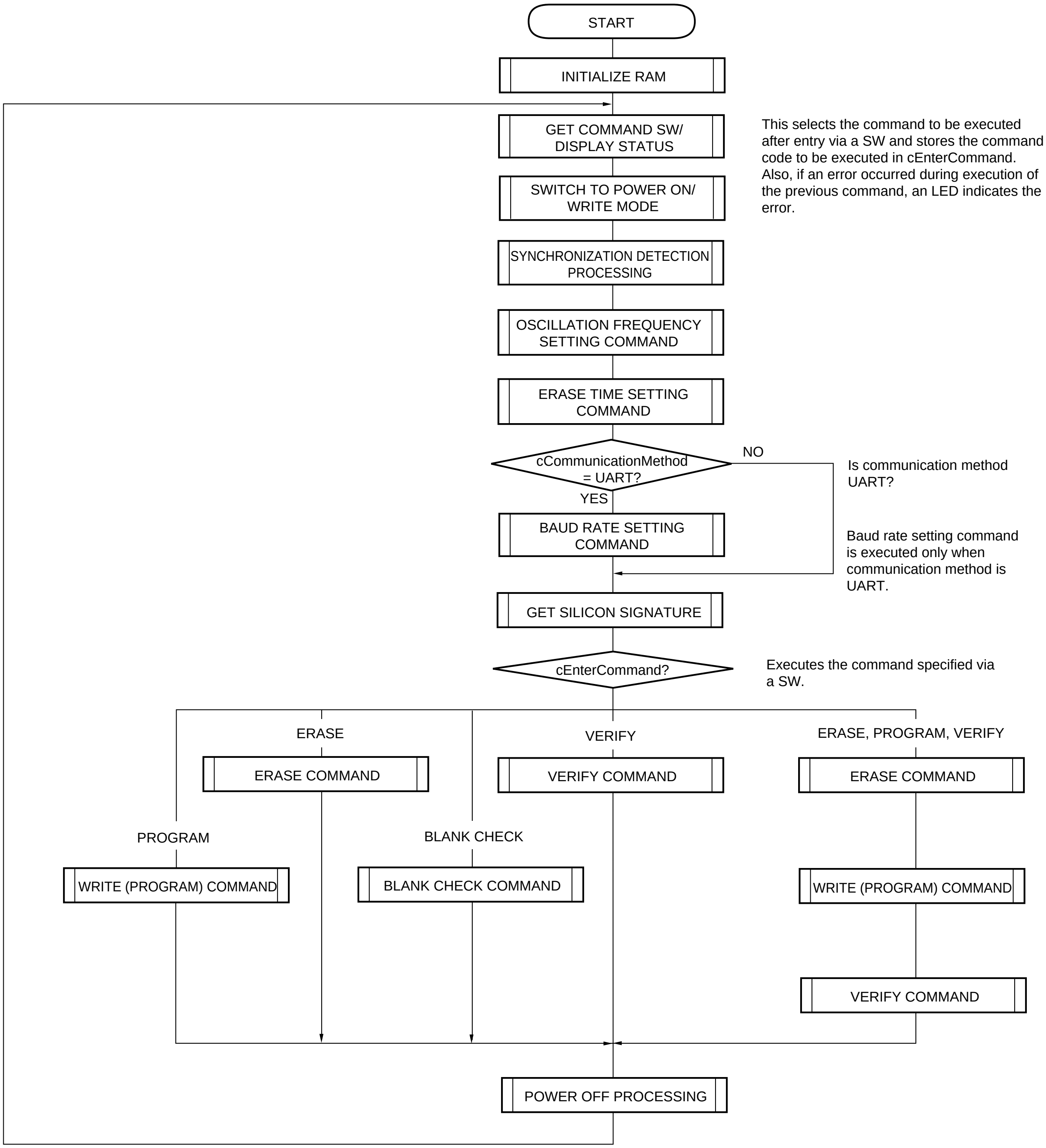
(2) Sample program

```
#pragma sfr                                   //Uses sfr area

#include "DATTYPE.H"                          //Data type definition file
#include "sram.h"                             //RAM external access definition file
#include "constant.h"                         //Constant value definition file

/****************************************************
*                Hardware initialization                    *
*   Initializes D/A converter, port output latch, and port mode.   *
*   This routine is called by the startup routine.            *
****************************************************/
void hdwinit( void ){

   MM = 0x29;                                 //1 MByte expansion mode
   STBC= 0x00;                                //fCLK/2

/***** Initialize D/A converter *****/
   DACS1 = 0;                                 //VPP = 0 V
   DACS0 = 0;                                 //VDD = 0 V

/***** Set port output latch and port mode *****/
   P0 = 0b00000010;                           //Selects CPU clock
                                              //P0.1:  High (no clock supply)
                                              //P0.6:  Low [1.25 MHz]
                                              //P0.7:  Low [1.25 MHz]
   P1 = 0b00001111;                           //Status lamp (low active)
                                              //P1.0:  BLANK CHEK
                                              //P1.1:  VERIFY
                                              //P1.2:  PROGRAM
                                              //P1.3:  ERASE
   P3 = 0b00001000;                           //P3.0 [RXD]
                                              //P3.1 [TXD]
                                              //P3.2:  [SCK/SCL] (low active)
                                              //P3.3:  [SO/SDA]
   P6 = 0b00000000;                           //P6.7 Reset target (low active)
   P7 = 0b00000000;
   PM0 = 0b00000000;                          //Output mode
   PM1 = 0b00000000;                          //Output mode
   PM3 = 0b00000001;                          //P3.0 [RXD]
                                              //P3.1 [TXD]
                                              //P3.2 [SCK] (low active)
                                              //P3.3 [SO]
   PM6 = 0b00000000;                          //P6.7 Reset target (low active)
   PM7 = 0b00011111;                          //Command SW
                                              //P7.0 BLANK CHEK
                                              //P7.1 VERIFY
                                              //P7.2 PROGRAM
                                              //P7.3 ERASE
                                              //P7.4 E.P.V

   PMC1 = 0b00000000;                         //Set general-purpose port
   PMC3 = 0b00000000;                         //High during initialization of communication method
   PUO = 0b00000100;                          //P2.[2-6] On-chip pull-up resistor connection (not used)
}
```

4.5.3 Main processing

The following sample shows SW selection of five types of processing: erase, write, verify, blank check, and erase/write/verify. NEC recommends changing the main processing to suit the target system before using it.

(1) Flow chart



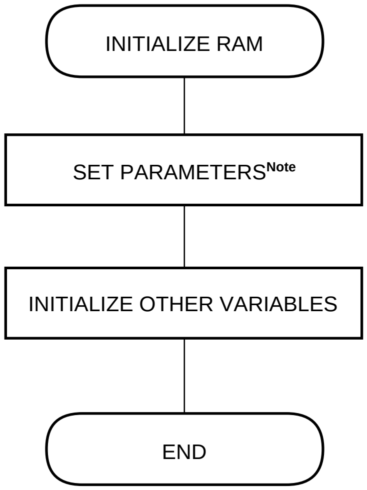
(2) Sample program

```
#pragma sfr                                         //Uses sfr area

#include "DATTYPE.H"                                //Data type definition file
#include "sram.h"                                   //RAM external access definition file
#include "constant.h"                               //Constant value definition file

/*-----------------------------------------
     FUNCTION PROTOTYPE DECLARATION
-----------------------------------------*/
void RamIni( void );                                //Initializes RAM (parameters, etc.)
void MGetCom( void );                               //Gets command
void MPowerOn( void );                              //Switches to power on/write mode
void MSyncChek( void );                             //Synchronization detection
void MFrequencySetUp( void );                       //Sets oscillation frequency
void MEraseTimeSetUp( void );                       //Sets erase time
void MBaudRate( void );                             //Sets baud rate
void MGetSiliconeSignature( void );                 //Gets silicon signature
void MErase( void );                                //Erase
void MProgram( void );                              //Write
void MVerify( void );                               //Verify
void MBlankChek( void );                            //Blank check
void MPowerOff( void );                             //Power off

/*************************************************************
*                         Main routine                       *
*     Global variables:  cErrorStatus          Error status  *
*                        cCommunicationMethod  Communication method *
*************************************************************/
void main( void ){
   RamIni();                                        //Initializes RAM

   while(1){
      MGetCom();                                    //Gets command

      MpowerOn();                                   //Switches to power on/write mode

      MSyncChek();                                  //Synchronization detection
      if( cErrorStatus == NO_ERROR ){               //Any errors?

          MFrequencySetUp();                        //Sets oscillation frequency
          if( cErrorStatus == NO_ERROR ){           //Any errors?
              if( UART == cCommunicationMethod ){
                                                    //UART communication method?
                  MBaudRate();                      //If yes, sets baud rate
              }
              if( cErrorStatus == NO_ERROR ){
                                                    //Any errors?
                  MEraseTimeSetUp();                //Sets erase time
                  if( cErrorStatus == NO_ERROR ){
                                                    //Any errors?
                      MGetSiliconeSignature();
                                                    //Gets silicon signature
                      if( cErrorStatus == NO_ERROR ){
                                                    //Any errors?
                          switch( cEnterCommand ){
                                                    //Erase, write, and verify
                             case ENTER_EPV:
                                P1.3 = 0;
                                                    //Displays status
```

```
                                        MErase();
                                                                //Erase
                                        if( cErrorStatus != NO_ERROR )break;
                                                                //Any errors?
                                        P1.2 = 0;
                                                                //Displays status
                                        MProgram();
                                                                //Write
                                        if( cErrorStatus != NO_ERROR )break;
                                                                //Any errors?
                                        P1.1 = 0;
                                                                //Displays status
                                        MVerify();break;
                                                                //Verify
                                    case ENTER_ERA:    MErase();     break;
                                                                //Erase
                                    case ENTER_PRG:    MProgram();   break;
                                                                //Write
                                    case ENTER_VRF:    MVerify();    break;
                                                                //Verify
                                    case ENTER_BLN:    MBlankChek(); break;
                                                                //Blank check
                                }
                            }
                        }
                    }
                }
            }
            MPowerOff();                                        //Power OFF
    }
}
```

4.5.4 RAM initialization

(1) Flow chart



Note The parameters required for the write operation must be set.

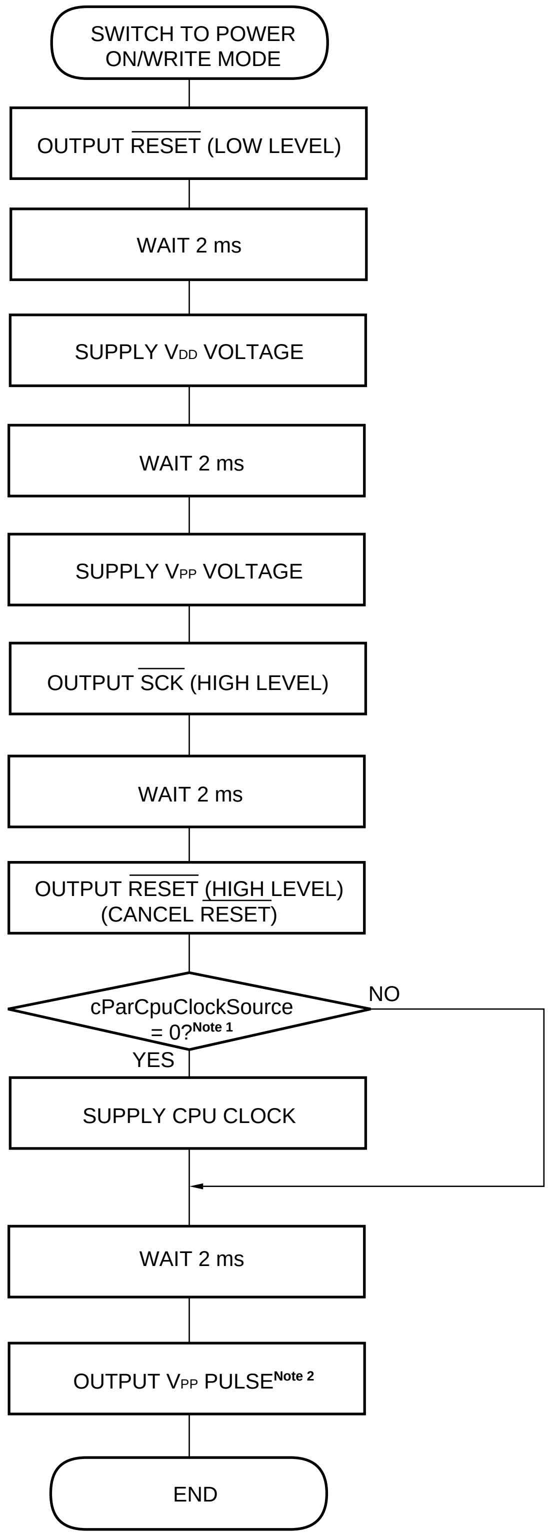
(2) Sample program

```
#pragma sfr                                    //Uses sfr area

#include "DATTYPE.H"                           //Data type definition file
#include "sram.h"                              //RAM external access definition file
#include "constant.h"                          //Constant value definition file

/*---------------------------------------------
       FUNCTION PROTOTYPE DECLARATION
---------------------------------------------*/
Word  SWaitTimeCalc( Word wWaitClock );        //Calculates communication wait time

/*******************************************************************************
*                          RAM initialization                                  *
*                                                                              *
*  Initializes parameter settings and other RAM to be used as global variables *
*******************************************************************************/
void RamIni( void ){


/****     Parameter settings     ****/

    cParTargetSeries = K0S;                    //Selects 78K/0S as target series

    dwParStartAddress = 0x00000000;            //Write start address
    dwParEndAddress = 0x00007fff;              //Write end address
                                               //Stores write program size
                                               //Set a value that does not exceed the capacity of
                                               //the target microcontroller's flash memory.

    wParEraseTime = 200;                       //Erase time (in 10-ms units)
                                               //2.0 s is set in the sample program.

    cParCpuClockSource = IN_FLASHWRITER;       //Sets source for supply of CPU clock to target
                                               //microcontroller
                                               //Select the method for supplying a CPU clock from
                                               //this flash programmer.

    wParCpuClockSpeed = 500;                   //Oscillation frequency (in 10-kHz units)
                                               //Speed of CPU clock supplied to target
                                               //microcontroller
                                               //5 MHz is set in sample program
                                               //Setting range: 100 to 1,000 (1 MHz to 10 MHz)

    cParVppPulse = SIO_CH0;                    //VPP pulse count (0 to 14)
                                               //3-wire serial, channel 0 is selected in sample
                                               //program
                                               //SIO_CH0 = 0)
                                               //See 4.6.5 List of constant value definitions for
                                               //correspondence between VPP pulse count and
                                               //selected communication method

    wParCsiClockSpeed = 10000;                 //Serial clock speed (3-wire serial or pseudo 3-wire
                                               //serial communication method), 100-Hz units
                                               //1 MHz is set in sample program
                                               // (For pseudo 3-wire serial communication method,
                                               //set to 1 kHz or less)
    cParBaudRate = BPS9600;                    //Baud rate for UART communications
                                               //9,600 bps is selected in sample program
```

```
    cParSlaveAddress = 0x10;                //Slave address setting (required for IIC
                                            //communications)
                                            //Setting range: (08H to 77H) Communication will
                                            //not be possible if it is set outside of this range.

/****    End setting of parameters    ****/

    cCommunicationMethod = cParVppPulse / 4;    //RAM setting to select communication method
                                            //(used to speed up communication processing)
                                            //  Value    Communication method
                                            //    0      3-wire serial
                                            //    1      IIC
                                            //    2      UART
                                            //    3      Pseudo 3-wire serial

    cWaitClockSelect = cParVppPulse / 4;    //Sets element number of structure array (wait data
    if( cParTargetSeries == K0S ){          //table) used to select the number of wait clocks for
        cWaitClockSelect += 4;              //each communication method and target series
}

/***** Calculates each communication wait time and stores value in RAM (to speed up communications) *****/
    wWaitTimeVppCom = SWaitTimeCalc( WaitDataTable[cWaitClockSelect].wWaitVppCom );
                                            //Wait time between VPP and command [µs units]
    wWaitTimeComAck = SWaitTimeCalc( WaitDataTable[cWaitClockSelect].wWaitComAck );
                                            //Wait time between command and ACK [µs units]
    wWaitTimeAckCom = SWaitTimeCalc( WaitDataTable[cWaitClockSelect].wWaitAckCom );
                                            //Wait time between ACK and command [µs units]
    wWaitTimeAckData = SWaitTimeCalc( WaitDataTable[cWaitClockSelect].wWaitAckData );
                                            //Wait time between ACK and data [µs units]
    wWaitTimeDataData = SWaitTimeCalc( WaitDataTable[cWaitClockSelect].wWaitDataData );
                                            //Wait time between two data sets [µs units]
    wWaitTimeDataAck = SWaitTimeCalc( WaitDataTable[cWaitClockSelect].wWaitDataAck );
                                            //Wait time between data and ACK [µs units]

    cTargetStatus = READY;                  //Initializes target microcontroller's status
    cRetryCounter = 0;                      //Initializes retry counter
    cErrorStatus = NO_ERROR;                //Initializes error status (no errors)
    cEnterCommand = ENTER_NOTHING;          //Initializes input command (no entered command)
    cTimerFlag = WAIT_START;                //Initializes timer flag

}
```

4.5.5 Switch to power on/write mode

(1) Flow chart



Note 1. 0: Supply CPU clock from flash programmer
1: Use target board's CPU clock

Note 2. Stop output of V_{PP} pulse within 20 ms after starting supply of CPU clock.

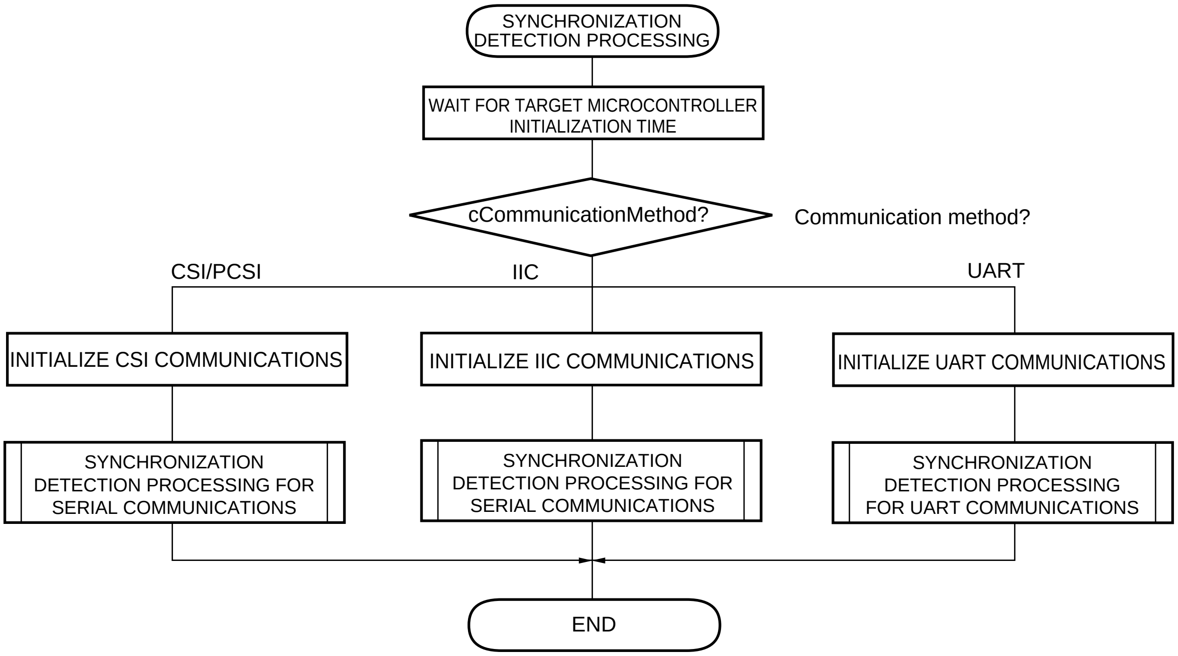
(2) Sample program

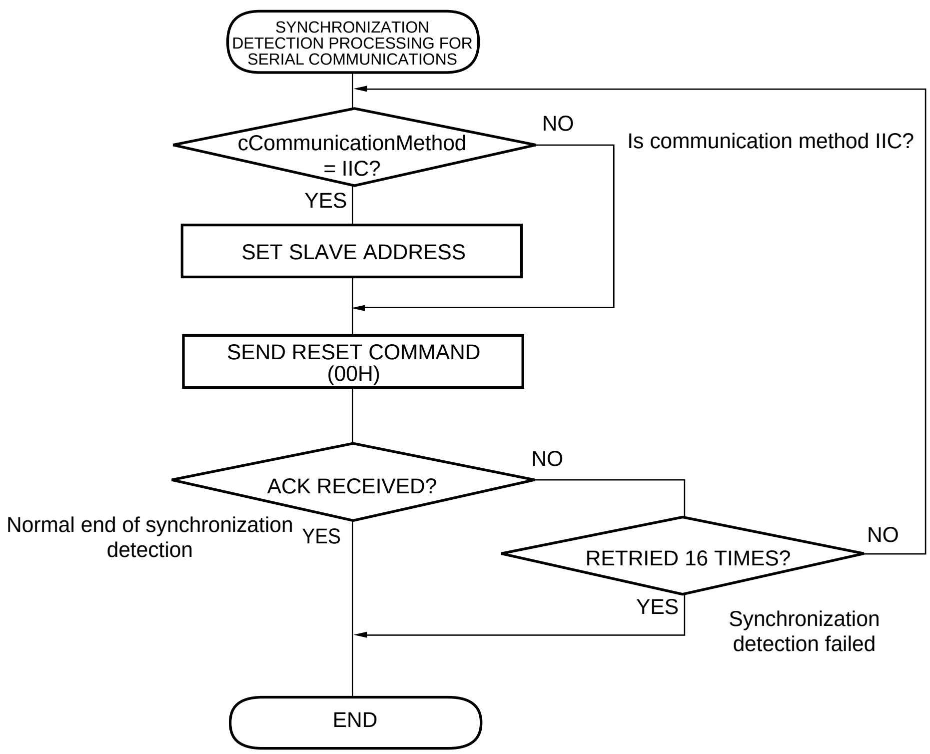
```
#pragma sfr                                          //Uses sfr area

#include "DATTYPE.H"                                 //Data type definition file
#include "sram.h"                                    //RAM external access definition file
#include "constant.h"                                //Constant value definition file

/*-----------------------------------------------
        FUNCTION PROTOTYPE DECLARATION
-----------------------------------------------*/
void SWaitMiliSec( Word wWaitTime );                 //Wait time (1-ms units)
void SWait30us( void );                              //30-µs wait time

/**********************************************************************
*             Switch to power on/write mode                           *
*     Global variables:  cParCpuClockSource   CPU clock source        *
*                        wParCpuClockSpeed    CPU clock speed         *
*                        cErrorStatus         Error status            *
*                        cTimerFlag           Timer flag              *
*                                                                     *
*     Local variable:    cVppPulseCounter     VPP pulse counter       *
*                                                                     *
**********************************************************************/
void MPowerOn(void){
   register Byte cVppPulseCounter;                   //VPP pulse counter

   P6.7 = 0;                                         //Low-level output of RESET signal
   cTimerFlag = WAIT_START;
   do{
      SWaitMiliSec( 2 );                             //2-ms wait time
   }while( cTimerFlag == WAIT_NOW );

   DAM = 0x03;                                       //Enables D/A converter output
   DACS0 = 85;                                       //VDD = 5-V power supply
                                                     //(5/256) V × 85 × 3 = 4.98 V
   do{
      SWaitMiliSec( 2 );                             //2-ms wait time
   }while( cTimerFlag == WAIT_NOW );

   DACS1 = 171;                                      //VPP = 10-V power supply
                                                     //(5/256) V × 171 × 3 = 10.02 V

   P3.2 = 1;                                         //High-level output of SCK signal
   do{
      SWaitMiliSec( 2 );                             ///2-ms wait time
   }while( cTimerFlag == WAIT_NOW );

   P6.7 = 1;                                         //High-level output of RESET signal

*****   Select CPU clock    *****/
   if( cParCpuClockSource == IN_FLASHWRITER ){ //When supplying a CPU clock from the flash
                                                     //programmer
       switch( wParCpuClockSpeed / 100 ){
          case 1:   P0.6 = 0;                        //1.25 MHz
             P0.7 = 0; break;
          case 2:   P0.6 = 1;                        //2.5 MHz
             P0.7 = 0; break;
          case 5:   P0.6 = 0;                        //5.0 MHz
             P0.7 = 1; break;
          case 10:P0.6 = 1;                          //10 MHz
             P0.7 = 1; break;
```

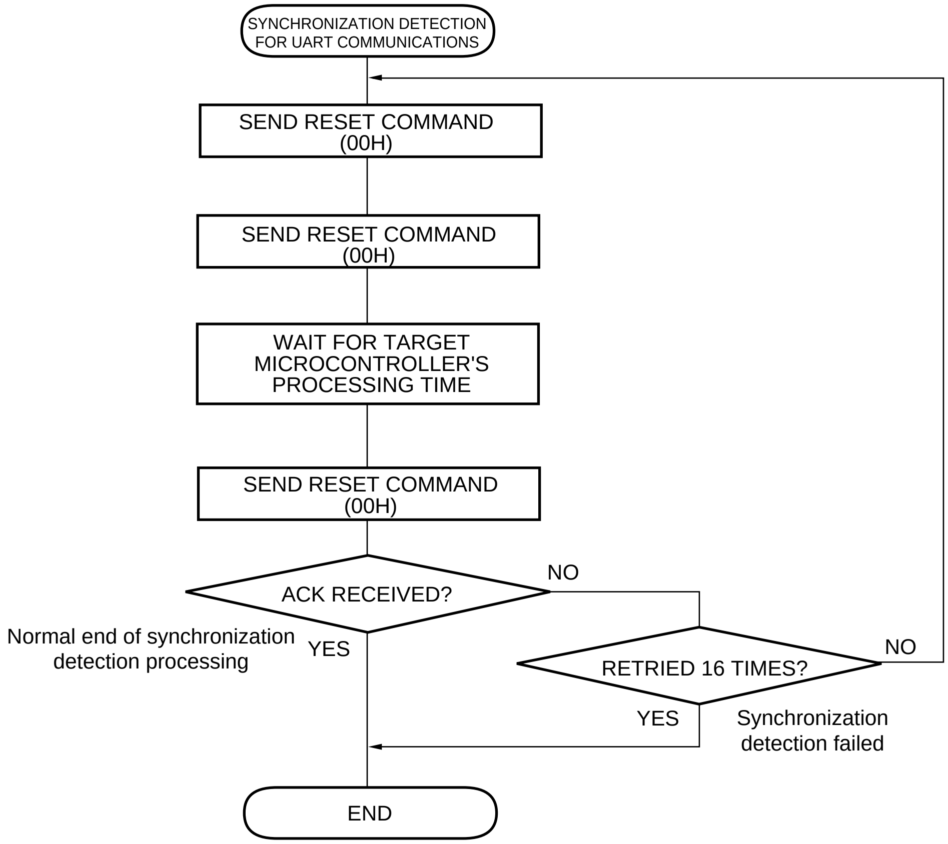
```
          default:
             cErrorStatus = PARAMETER_OUT_OF_RANGE;
                                                           //Parameter is out of range
       }

       P0.1 = 0;                                           //Clock supply starts when P0.1 = L
   }
   do{                                                     //Output timing of VPP pulse after high-level output of
                                                           //RESET signal:
       SWaitMiliSec( 2 );                                  //Within 20 ms for 78K/0 series (@8.38-MHz
                                                           //operation)
   }while( cTimerFlag == WAIT_NOW );                       //Within 20 ms for 78K/0S series (@10-MHz
                                                           //operation)

: /***** VPP pulse output *****/                          //Notifies target flash microcontroller of
                                                           //communication method
   for( cVppPulseCounter = cParVppPulse ; 0 < cVppPulseCounter ; cVppPulseCounter-- ){
      DACS1 = 85;                                          //VPP pin, VDD level output (5 V)
      SWait30us();                                         //30-µs wait time
      DACS1 = 171;                                         //VPP pin, VPP level output (10 V)
      SWait30us();                                         //30-µs wait time
   }
}
```

4.5.6 Synchronization detection processing

(1) Flow chart






SYNCHRONIZATION DETECTION
FOR UART COMMUNICATIONS
SEND RESET COMMAND
(00H)
SEND RESET COMMAND
(00H)
WAIT FOR TARGET
MICROCONTROLLER'S
PROCESSING TIME
SEND RESET COMMAND
(00H)
ACK RECEIVED?
NO
YES
Normal end of synchronization
detection processing
RETRIED 16 TIMES?
NO
YES
Synchronization
detection failed
END

(2) Sample program

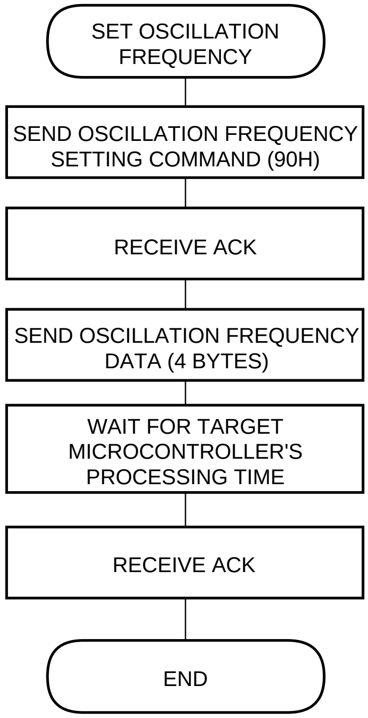
```
#pragma sfr                                  //Uses sfr area

#include "DATTYPE.H"                         //Data type definition file
#include "sram.h"                            //RAM external access definition file
#include "constant.h"                        //Constant value definition file

/*-----------------------------------------------
        FUNCTION PROTOTYPE DECLARATION
-----------------------------------------------*/
void SDataSend( Word SendSize , Byte *SendDataAddress );
                                             //Sends data
void SDataRecieve( Word wRecieveDataSize );  //Receives data
void SWaitMiliSec( Word wWaitTime );         //Wait time (1-ms units)
void SWaitMicroSec( Word wCrRegData );       //Wait time (1-µs units)
void SWait( DWord dwWaitClock );             //Wait
void SCsiIni( void );                        //Initializes 3-wire serial/pseudo 3-wire serial
                                             //communications
void SUartIni( void );                       //Initializes UART communications
void SIicIni( void );                        //Initializes IIC communications
void SSyncChekCsiOrIic( void );              //Synchronization detection for 3-wire serial/pseudo
                                             //3-wire serial/IIC communications
void SSyncChekUart( void );                  //Synchronization detection for UART
                                             //communications
void SSlaveAddressSend( Byte cSendOrRecieve ); //Slave address transmit processing

/*******************************************************************
*                  Synchronization detection                       *
*      Global variables:  cCommunicationMethod   Communication method  *
*                         cTimerFlag             Timer flag            *
*******************************************************************/
void MSyncChek( void ){

/****** Wait for target microcontroller's initialization time *****/
   cTimerFlag = WAIT_START;                  //Wait for flash microcontroller's initialization time
   do{                                       // (Wait time for flash microcontroller's oscillation
                                             //stabilization + wait time for initialization of flash
                                             //microcontroller)
       SWaitMiliSec( 100 );                  //In the sample program, this time is set as 100 ms.
   }while( cTimerFlag == WAIT_NOW );

/***** Synchronization detection for each communication method *****/
   switch( cCommunicationMethod ){           //Synchronization detection for each communication
                                             //method
      case CSI:                              //3-wire serial
      case PCSI:                             //Pseudo 3-wire serial
               SCsiIni( );                   //Initializes 3-wire serial/pseudo 3-wire serial
                                             //communications
               SSyncChekCsiOrIic( );         //3-wire serial/pseudo 3-wire serial/IIC
                                             //synchronization detection
               break;
      case IIC:                              //IIC
               SIicIni( );                   //Initializes IIC communications
               SSyncChekCsiOrIic( );         //3-wire serial/pseudo 3-wire serial/IIC
                                             //synchronization detection
               break;
      case UART:                             //UART
               SUartIni( );                  //Initializes UART communications
               SSyncChekUart( );             //UART synchronization detection processing
               break;
```

```
      default:    cErrorStatus = PARAMETER_OUT_OF_RANGE;
                                                          //Parameter is out of range
                  return;
   }
}

/*********************************************************
*     3-wire serial/pseudo 3-wire serial/IIC synchronization detection       *
*    Global variables:   cSendData            Send data                      *
*                        cRecieveData         Receive data                   *
*                        cErrorStatus         Error status                   *
*                        cRetryCounter        Retry counter                  *
*                        wWaitTimeComAck      COM-ACK wait time              *
*                        wWaitTimeAckCom      ACK-COM wait time              *
*********************************************************/
void SSyncChekCsiOrIic( void ){
   for( cRetryCounter = 0 ; cRetryCounter < 16 ; cRetryCounter++ ) {
                                               //Maximum of 16 retries for cRetryCounter
      switch(cCommunicationMethod){            //Communication method:  IIC
         case IIC:                             //Notifies the target microcontroller concerning its
                                               //slave address
           if(( cParSlaveAddress < 0x08 )||    //Slave address is out of range
                ( cParSlaveAddress > 0x77 )){  //(Valid range is 0x08 to 0x77)
                   cErrorStatus = PARAMETER_OUT_OF_RANGE;
                                               //Parameter is out of range
                   return;
                }
                SSlaveAddressSend( ((cParSlaveAddress & 0b01000000) >> 6) );
                                               //Sets transfer direction bit to A6 (same value as
                                               //bit6 in address)
                SPT = 1;                       //Outputs stop condition
           }
           cErrorStatus = NO_ERROR;            //Sets error status to "no errors"

/***** Send reset command  *****/
       cSendData = CMD_RESET;
       SDataSend( 1, &cSendData );             //Sends reset command
       if( cErrorStatus != NO_ERROR )continue; //Retries if error has occurred

       SWaitMicroSec( wWaitTimeComAck );       //Wait time between sending command and
                                               //receiving an ACK signal

/***** Receive ACK *****/
       SDataRecieve( 1 );                      //Receives ACK signal
       if( cErrorStatus != NO_ERROR )continue; //Any errors?
       SWaitMicroSec( wWaitTimeAckCom );       //Wait for time between receiving ACK signal and
                                               //sending command
       if( cRecieveData != ACK )     continue; //Is receive data an ACK signal?

       return;                                 //Normal end of synchronization detection
   }
   cErrorStatus = INITIALISE_ERROR;            //Synchronization detection failed
}
```

```
/*******************************************************************
*              UART synchronization detection                       *
*     Global variables:  cSendData          Send data              *
*                        cRecieveData       Receive data           *
*                        cErrorStatus       Error status           *
*                        cRetryCounter      Retry counter          *
*                        wWaitTimeComAck    COM-ACK wait time      *
*                        wWaitTimeAckCom    ACK-COM wait time      *
*******************************************************************/
void SSyncChekUart( void ){
   for( cRetryCounter = 0 ; cRetryCounter < 16 ; cRetryCounter++ ){
                                                //Maximum of 16 retries for cRetryCounter
      cErrorStatus = NO_ERROR;                  //Sets error status as "no errors"
/***** Send reset command *****/
      cSendData = CMD_RESET;
      SDataSend( 1, &cSendData );               //Sends reset command (first time)
      if( cErrorStatus != NO_ERROR )continue;   //Any errors?
          cRetryCounter++;

      SWait(( DWord )WaitDataTable[cWaitClockSelect].wWaitRst1 );
                                                //Wait time after first reset command is sent

/***** Send reset command *****/
      cSendData = CMD_RESET;
      SDataSend( 1, &cSendData );               //Sends reset command (second time)
      if( cErrorStatus != NO_ERROR )continue;   //Any errors?
          cRetryCounter++;

      SWait(( DWord )WaitDataTable[cWaitClockSelect].wWaitRst2 );
                                                //Wait time after second reset command is sent
/***** Sends reset command *****/
      cSendData = CMD_RESET;
      SDataSend( 1, &cSendData );               //Sends reset command (third time)
      if( cErrorStatus != NO_ERROR )continue;   //Any errors?

      SWait(( DWord )WaitDataTable[cWaitClockSelect].wWaitRst3 );
                                                //Wait time after third reset command is sent
/***** Receive ACK *****/
      SDataRecieve( 1 );                        //Receives ACK signal
      if( cErrorStatus != NO_ERROR )continue;   //Any errors?
      SWaitMicroSec( wWaitTimeAckCom );         //Wait for time between receiving ACK signal and
                                                //sending command
      if( cRecieveData != ACK )continue;        //Is receive data an ACK signal?

      return;                                   //Normal end of synchronization detection
   }
   cErrorStatus = INITIALISE_ERROR;             //Synchronization detection failed
}
```

4.5.7 Oscillation frequency setting command

(1) Flow chart



Example: The following is indicated for 5.00 MHz operation.
Since 5.00 MHz = 0.500×10^4 [kHz], the oscillation frequency data is "05 00 00 04".

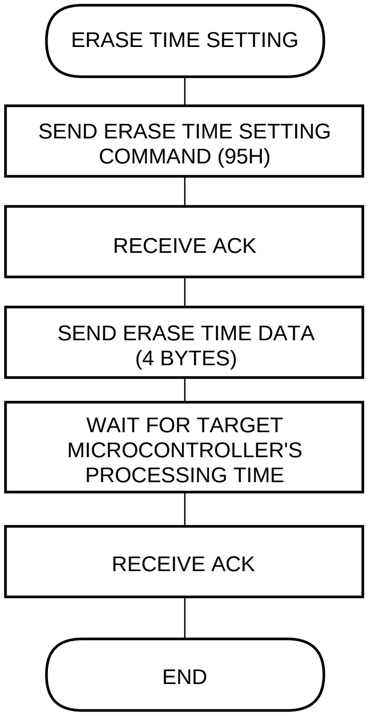
(2) Sample program

```
#pragma sfr                                         //Uses sfr area

#include "DATTYPE.H"                                //Data type definition file
#include "sram.h"                                   //RAM external access definition file
#include "constant.h"                               //Constant value definition file

/*-----------------------------------------------
        FUNCTION PROTOTYPE DECLARATION
-----------------------------------------------*/
void SDataSend( Word SendSize , Byte *SendDataAddress );
                                                    //Sends data
void SDataRecieve( Word wRecieveDataSize );         //Receives data
void SWait( DWord dwWaitClock );                    //Wait
void SWaitMicroSec( Word wCrRegData );              //Wait time (1-µs units)

/*********************************************************
*             Oscillation frequency setting command             *
*     Global variables:   cSendData          Send data          *
*                         cRecieveData       Receive data       *
*                         cErrorStatus       Error status       *
*                         wWaitTimeComAck    COM-ACK wait time  *
*                         wWaitTimeAckData   ACK-COM wait time  *
*                                                               *
*     Local variable:     wWork1             Work area 1        *
*                         cWork2             Work area 2        *
*                         cWork3             Work area 3        *
*                                                               *
*********************************************************/
void MFrequencySetUp( void ){
   register Word wWork1;                            //Work area 1
   register Byte cWork2;                            //Work area 2
   register Byte cWork3;                            //Work area 3

/***** Send command *****/
   cSendData = CMD_FRQ_SET;
   SDataSend( 1, &cSendData );                      //Sends oscillation frequency setting command
   if( cErrorStatus != NO_ERROR )  return;          //Any errors?

   SWaitMicroSec( wWaitTimeComAck );                //Wait for time between sending command and
                                                    //receiving ACK signal
/***** Receive ACK *****/
   SDataRecieve( 1 );                               //Receives ACK signal
   if( cErrorStatus != NO_ERROR )  return;          //Any errors?
   if( cRecieveData != ACK ){                       //Is receive data an ACK signal?
       cErrorStatus = TARGET_RETURN_ERROR; return;
   }
   SWaitMicroSec( wWaitTimeAckData );               //Wait for time between receiving ACK signal and
                                                    //sending data


/***** CPU clock range judgement (valid range: 1 MHz to 10 MHz) *****/
   if(!(( 100 <= wParCpuClockSpeed ) && ( 1000 >= wParCpuClockSpeed ))){
        cErrorStatus = PARAMETER_OUT_OF_RANGE;
        return;                                     //Invalid value was set to parameter
   }
```

```
/**** Send four bytes of oscillation frequency data ****/
   wWork1 = wParCpuClockSpeed;
   cWork2 = 4;                                  //Exponent: 10^4 (10-kHz units)
   if( 1000 == wParCpuClockSpeed ){            //When oscillation frequency is 10 MHz
       wWork1 /= 10;                            //Multiplies by 0.1 and adds 1 to exponent
       cWork2++;
   }
   aSendBuffer[3] = cWork2;                     //Exponent

   for( cWork3 = 0 ; 100 <= wWork1 ; cWork3++ ){
        wWork1 -= 100;                          //Example:  when wParCpuClockSpeed (10-kHz
                                                //          units) = 500 (5 MHz)
   }                                            //Send data Hi Mid Low
   aSendBuffer[0] = cWork3;                     //          5   0   0
   for( cWork3 = 0 ; 10 <= wWork1 ; cWork3++ ){
        wWork1 -= 10;
   }
   aSendBuffer[1] = cWork3;
   for( cWork3 = 0 ; 1 <= wWork1 ; cWork3++ ){
        wWork1 -= 1;
   }
   aSendBuffer[2] = cWork3;

   SDataSend( 4 , aSendBuffer );                //Sends buffer contents (4 bytes)
   if( cErrorStatus != NO_ERROR )  return;      //Any errors?

   SWait(( DWord )WaitDataTable[cWaitClockSelect].wWaitFrequencySet );
                                                //Wait for target microcontroller's processing time
/***** Receive ACK *****/
   SDataRecieve( 1 );                           //Receives ACK signal
   if( cErrorStatus != NO_ERROR )  return;      //Any errors?
   if( cRecieveData != ACK ){                   //Is receive data an ACK signal?
       cErrorStatus = TARGET_RETURN_ERROR; return;
   }
}
```

4.5.8 Erase time setting command

(1) Flow chart



Example: The following is indicated for 2 s erase time. Since 2 s = 0.200×10^1 (s), the erase time data is "02 00 00 01".

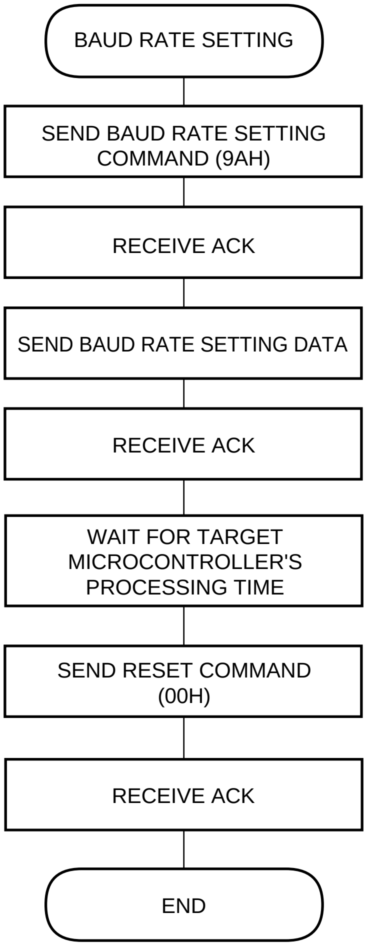
(2) Sample program

```
#pragma sfr                                         //Uses sfr area

#include "DATTYPE.H"                                //Data type definition file
#include "sram.h"                                   //RAM external access definition file
#include "constant.h"                               //Constant value definition file

/*---------------------------------------------
        FUNCTION PROTOTYPE DECLARATION
-----------------------------------------------*/
void SDataSend( Word SendSize , Byte *SendDataAddress );
                                                    //Sends data
void SDataRecieve( Word wRecieveDataSize );         //Receives data
void SWait( DWord dwWaitClock );                    //Wait
void SWaitMicroSec( Word wCrRegData );              //Wait time (1-µs units)

/**********************************************************
*                       Erase time setting                *
*      Global variables:  cSendData         Send data     *
*                         cRecieveData      Receive data  *
*                         cErrorStatus      Error status  *
*                         wWaitTimeComAck   COM-ACK wait time   *
*                         wWaitTimeAckData  ACK-DATA wait time  *
*                                                         *
*      Local variable:    wWork1            Work area 1   *
*                         cWork2            Work area 2   *
*                         cWork3            Work area 3   *
***********************************************************/
void MEraseTimeSetUp( void ){
   register Word wWork1;                            //Work area 1
   register Byte cWork2;                            //Work area 2
   register Byte cWork3;                            //Work area 3

/***** Send command *****/
   cSendData = CMD_ERT_SET;
   SDataSend( 1, &cSendData );                      //Sends erase time setting command
   if( cErrorStatus != NO_ERROR )  return;          //Any errors?

   SWaitMicroSec( wWaitTimeComAck );                //Wait for time between sending command and
                                                    //receiving ACK signal

/***** Receive ACK *****/
   SDataRecieve( 1 );                               //Receives ACK signal
   if( cErrorStatus != NO_ERROR )  return;          //Any errors?
   if( cRecieveData != ACK ){                       //Is receive data an ACK signal?
       cErrorStatus = TARGET_RETURN_ERROR; return;
   }

   SWaitMicroSec( wWaitTimeAckData );               //Wait for time between receiving ACK signal and
                                                    //sending data

/**** Erase time range judgement (valid range: 0.5 s to 20 s) ****/
   if(!(( 50 <= wParEraseTime ) && ( 2000 >= wParEraseTime ))){
       cErrorStatus = PARAMETER_OUT_OF_RANGE;return;
                                                    //Invalid value was set to parameter
   }

/**** Send four bytes of erase time data ****/
   wWork1 = wParEraseTime;
   cWork2 = 1;                                      //Exponent: 10^1 (10-ms units)
   if( 1000 <= wParEraseTime ){                     //When erase time is 10 s or longer
```

```
        wWork1 /= 10;                              //Multiplies by 0.1 and adds 1 to exponent
        cWork2++;
    }
    aSendBuffer[3] = cWork2;                       //Exponent

    for( cWork3 = 0 ; 100 <= wWork1 ; cWork3++ ){
         wWork1 -= 100;                            //Example:  when wParEraseTime (10-ms units) =
                                                   //          200 (2 s)
    }                                              //Send data    Hi    Mid    Low
    aSendBuffer[0] = cWork3;                       //             2     0      0
    for( cWork3 = 0 ; 10 <= wWork1 ; cWork3++ ){
         wWork1 -= 10;
    }
    aSendBuffer[1] = cWork3;
    for( cWork3 = 0 ; 1 <= wWork1 ; cWork3++ ){
         wWork1 -= 1;
    }
    aSendBuffer[2] = cWork3;

    SDataSend( 4 , aSendBuffer );                  //Sends buffer contents (4 bytes)
    if( cErrorStatus != NO_ERROR )  return;        //Any errors?

    SWait(( DWord )WaitDataTable[cWaitClockSelect].wWaitEraseTimeSet );
                                                   //Wait for target microcontroller's processing time
    SDataRecieve( 1 );                             //Receives ACK
    if( cErrorStatus != NO_ERROR )  return;        //Any errors?
    if( cRecieveData != ACK )                      //Is receive data an ACK signal?
        cErrorStatus = TARGET_RETURN_ERROR;
}
```

4.5.9 Baud rate setting command

(1) Flow chart



Example: Baud rates values corresponding to baud rate settings are indicated as follows.
When value = 2: 4,800 bps, 3: 9,600bps, 4: 19,200 bps
5: 31,250 bps, 6: 38,400 bps, 7: 76,800 bps
If value is other than above, a data error occurs.

Sends a reset command using the new baud rate

Confirms that normal communications have been executed using the new baud rate

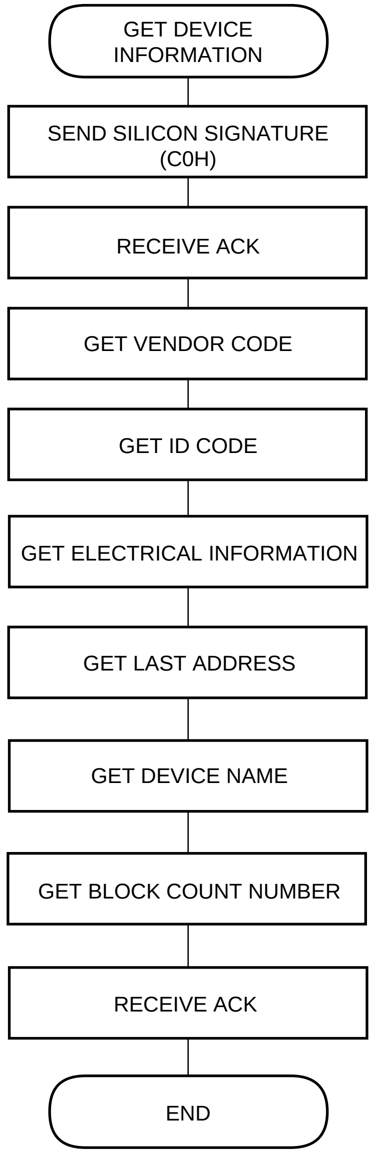
(2) Sample program

```
#pragma sfr                                   //Uses sfr area

#include "DATTYPE.H"                          //Data type definition file
#include "sram.h"                             //RAM external access definition file
#include "constant.h"                         //Constant value definition file

/*-----------------------------------------------
        FUNCTION PROTOTYPE DECLARATION
-----------------------------------------------*/
void SDataSend( Word SendSize , Byte *SendDataAddress );
void SDataRecieve( Word wRecieveDataSize );
void SWait( DWord dwWaitClock );
void SWaitMicroSec( Word wCrRegData );        //Wait for communications

/*******************************************************
*       Baud rate setting command (for UART communications only)        *
*       Global variables:   cSendData           Send data               *
*                           cRecieveData        Receive data            *
*                           cParBaudRate        Baud rate setting data  *
*                           cErrorStatus        Error status            *
*                           wWaitTimeComAck     COM-ACK wait time       *
*                           wWaitTimeAckData    ACK-DATA wait time      *
*                           wWaitTimeDataAck    DATA-ACK wait time      *
*******************************************************/
void MBaudRate( void ){

/***** Send command *****/
   cSendData = CMD_BAUDRATE;
   SDataSend( 1, &cSendData );                //Sends baud rate setting command
   if( cErrorStatus != NO_ERROR )  return;    //Any errors?

   SWaitMicroSec( wWaitTimeComAck );          //Wait for time between sending command and
                                              //receiving ACK signal

/***** Receive ACK *****/
   SDataRecieve( 1 );                         //Receives ACK signal
   if( cErrorStatus != NO_ERROR )  return;    //Any errors?
   if( cRecieveData != ACK ){                 //Is receive data an ACK signal?
       cErrorStatus = TARGET_RETURN_ERROR;return;
   }

   SWaitMicroSec( wWaitTimeAckData );         //Wait for time between receiving ACK signal and
                                              //sending data

/***** Send baud rate setting data *****/
   cSendData = cParBaudRate;
   SDataSend( 1, &cSendData );                //Sends baud rate setting data
                                              //cParBaudRate(bps)
                                              //0:  1,200    1:  2,400    2:  4,800    3:  9,600
                                              //4:  19,200   5:  31,250   6:  38,400   7:  76,800

   if( cErrorStatus != NO_ERROR )return;      //Any errors?

   SWaitMicroSec( wWaitTimeDataAck );         //Wait for time between sending data and receiving
                                              //ACK signal
```

```
/***** Receive ACK *****/
   SDataRecieve( 1 );                           //Receives ACK signal
   if( cErrorStatus != CMD_RESET )return;       //Any errors?
   if( cRecieveData != ACK ){                   //Is receive data an ACK signal?
       cErrorStatus = TARGET_RETURN_ERROR;return;
   }

   RXE = 0;                                     //Receive inhibit
   TXE = 0;                                     //Transmit inhibit

/***** Set new baud rate to BRGC register *****/
   switch( cParBaudRate ){                      //Sets new baud rate to BRGC register
      case BPS4800:  BRGC = BRGC4800;  break;   //4,800 bps
      case BPS9600:  BRGC = BRGC9600;  break;   //9,600 bps
      case BPS19200: BRGC = BRGC19200; break;   //19,200 bps
      case BPS31250: BRGC = BRGC31250; break;   //31,250 bps
      case BPS38400: BRGC = BRGC38400; break;   //38,400 bps
      case BPS76800: BRGC = BRGC76800; break;   //76,800 bps
      default:  cErrorStatus = PARAMETER_OUT_OF_RANGE;
                                                //Parameter is out of range
   }

   SWait(( DWord )WaitDataTable[cWaitClockSelect].wWaitBaudRateCalc );
                                                //Wait for baud rate calculation time

   TXE = 1;                                     //Transmit enabled
   RXE = 1;                                     //Receive enabled

/***** Send reset command at new baud rate *****/
   cSendData = CMD_RESET;
   SDataSend( 1, &cSendData );                  //Sends reset command at the new baud rate
   if( cErrorStatus != NO_ERROR )return;        //Any errors?

   SWaitMicroSec( wWaitTimeComAck );            //Wait for time between sending command and
                                                //receiving ACK signal

/***** Receive ACK *****/                       //Confirms that baud rate setting was executed
                                                //normally.
   SDataRecieve( 1 );                           //Receives ACK signal
   if( cErrorStatus != NO_ERROR )return;        //Any errors?
   if( cRecieveData != ACK )                    //Is receive data an ACK signal?
       cErrorStatus = TARGET_RETURN_ERROR;
}
```

4.5.10 Get device information command

(1) Flow chart



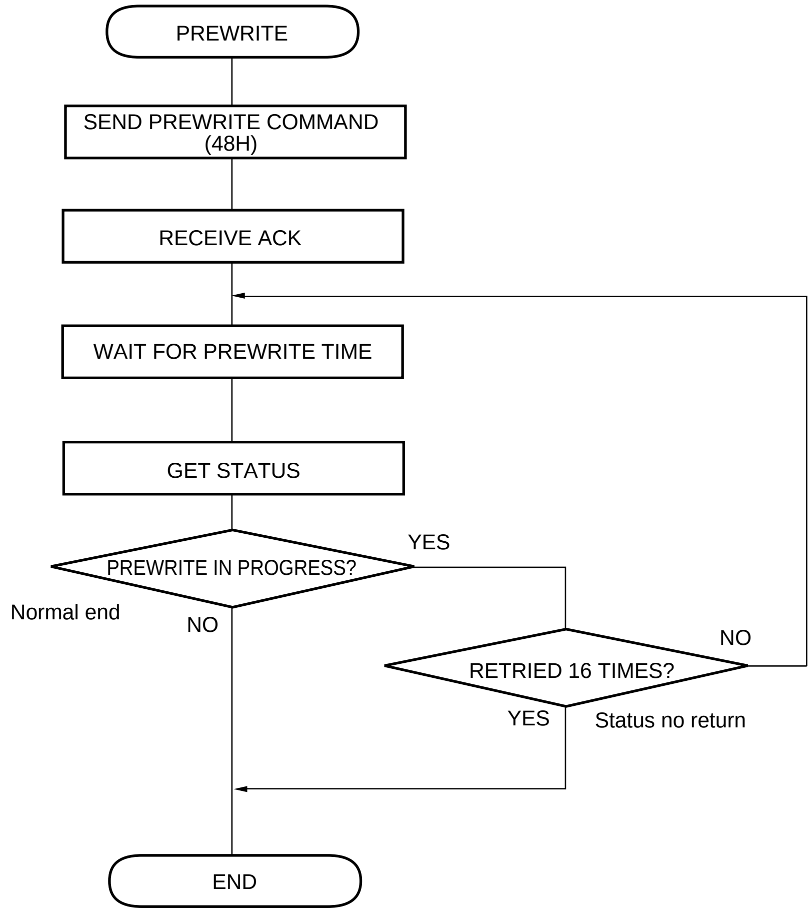
(2) Sample program

```
#pragma sfr                                       //Uses sfr area
#pragma NOP

#include "DATTYPE.H"                              //Data type definition file
#include "sram.h"                                 //RAM external access definition file
#include "constant.h"                             //Constant value definition file

/*-------------------------------------------
        FUNCTION PROTOTYPE DECLARATION
--------------------------------------------*/
void SDataSend( Word SendSize , Byte *SendDataAddress );
                                                  //Sends data
void SDataRecieve( Word wRecieveDataSize );       //Receives data
void SWait( DWord dwWaitClock );                  //Wait
void SWaitMicroSec( Word wCrRegData );            //Wait time (1-µs units)

/*********************************************************
*               Get device information command           *
*     Global variables:   cSendData         Send data    *
*                         cRecieveData      Receive data *
*                         cErrorStatus      Error status *
*                         cTargetStatus     Target status *
*                         cRetryCounter     Retry counter *
*                         wWaitTimeComAck   COM-ACK wait time *
*                         wWaitTimeAckData  ACK-DATA wait time *
*                         wWaitTimeDataAck  DATA-ACK wait time *
*                                                        *
*     Local variable:     cWork             Work         *
*********************************************************/
void MGetSiliconeSignature( void ){
   register Byte cWork;                           //Work

/***** Send silicon signature command *****/
   cSendData = CMD_SIGNATURE;
   SDataSend( 1, &cSendData );                    //Sends silicon signature command
   if( cErrorStatus != NO_ERROR )  return;        //Any errors?

   if(cCommunicationMethod != UART ){             //If other than UART communications
       SWaitMicroSec( wWaitTimeComAck );          //Wait time between sending command and
                                                  //receiving ACK signal
   }

/***** Receive ACK *****/
   SDataRecieve( 1 );                             //Receives ACK signal
   if( cErrorStatus != NO_ERROR )  return;        //Any errors?
   if( cRecieveData != ACK ){                     //Is receive data an ACK signal?
       cErrorStatus = TARGET_RETURN_ERROR;  return;
   }

   if( cCommunicationMethod != UART ){            //If other than UART communications
       SWaitMicroSec( wWaitTimeAckData );         //Wait time between receiving ACK signal and
                                                  //sending data
   }

/***** Get device information *****/
   SDataRecieve( 17 );                            //Receives silicon signature data (17 bytes)
   if( cErrorStatus != NO_ERROR )  return;        //Any errors?
```

```
   if( cCommunicationMethod != UART ){          //If other than UART communications
       SWaitMicroSec( wWaitTimeDataAck );       //Wait time between receiving data and receiving
                                                //ACK signal
   }

   for( cWork = 0 ; cWork < 17 ; cWork++ ){     //Stores receive data to RAM
       switch( cWork ){
          case 0:   sSig.cSigVendorCode = aRecieveBuffer[ cWork ];
                                                //Gets vendor code
                    break;
          case 1:   sSig.cSigIdCode = aRecieveBuffer[ cWork ];
                                                //Gets ID code
                    break;
          case 2:   sSig.cSigElectInf = aRecieveBuffer[ cWork ];
                                                //Gets electrical information
                    break;
          case 3:   sSig.dwSigLastAddress = ((( DWord )aRecieveBuffer[ cWork ]) &
                    0x0000007f );
                                                //Gets last address (low)
                    break;
          case 4:   sSig.dwSigLastAddress |= (((( DWord )aRecieveBuffer[ cWork ]) &
                    0x0000007f) << 7 );
                                                //Gets last address (mid)
                    break;
          case 5:   sSig.dwSigLastAddress |= (((( DWord )aRecieveBuffer[ cWork ]) &
                    0x0000007f ) << 14 );
                                                //Gets last address (high)
                    break;
          case 6:                               //Gets device name (10 bytes)
          case 7:
          case 8:
          case 9:
          case 10:
          case 11:
          case 12:
          case 13:
          case 14:
          case 15:
                   sSig.aSigDeviceName[ (cWork -6) ] = aRecieveBuffer[ cWork];
                   break;
          case 16:
                   sSig.cSigBlockInf = aRecieveBuffer[ cWork ];
                                                //Gets block information
                   break;
      }
   }

/***** Receive ACK *****/
   SDataRecieve( 1 );                           //Receives ACK signal
   if( cErrorStatus != NO_ERROR )  return;      //Any errors?
       if( cRecieveData != ACK ){               //Is receive data an ACK signal?
           cErrorStatus = TARGET_RETURN_ERROR; return;
      }
}
```

4.5.11 Prewrite command

(1) Flow chart



(2) Sample program

```
#pragma sfr                                       //Uses sfr area

#include "DATTYPE.H"                              //Data type definition file
#include "sram.h"                                 //RAM external access definition file
#include "constant.h"                             //Constant value definition file

/*-----------------------------------------------
        FUNCTION PROTOTYPE DECLARATION
-----------------------------------------------*/
void MGetStatus( void );                          //Gets status
void SDataSend( Word SendSize , Byte *SendDataAddress );
                                                  //Sends data
void SDataRecieve( Word wRecieveDataSize );       //Receives data
void SWait( DWord dwWaitClock );                  //Wait
Word SWriteWaitTimeCalc( Byte cWriteOrPreWrite , Word wWaitClock);
                                                  //Calculates write time
void SWaitMiliSec( Word wWaitTime );              //Wait time (1-ms units)
void SWaitMicroSec( Word wCrRegData );            //Wait time (1-µs units)

/*********************************************************
*                     Prewrite command                   *
*     Global variables:   cSendData        Send data         *
*                         cRecieveData     Receive data      *
*                         cErrorStatus     Error status      *
*                         cTargetStatus    Target status     *
*                         cTimerFlag       Timer flag        *
*                         cRetryCounter    Retry counter     *
*                         wWaitTimeComAck  COM-ACK wait time *
*                                                            *
*     Local variables:    wSec             Wait time (1-s units)  *
*                         wMiliSec         Wait time (1-ms units) *
*********************************************************/
void MPreWrite( void ){
   register Word wSec;                            //Wait time (1-s units)
   register Word wMiliSec;                        //Wait time (1-ms units)

/***** Send command *****/
       cSendData = CMD_PRE_WRITE;
       SDataSend( 1, &cSendData );                //Sends prewrite command
   if( cErrorStatus != NO_ERROR )  return;        //Any errors?

   SWaitMicroSec( wWaitTimeComAck );              //Wait time between sending command and
                                                  //receiving ACK signal

/***** Receive ACK *****/
   SDataRecieve( 1 );                             //Receives ACK signal
   if( cErrorStatus != NO_ERROR )  return;        //Any errors?
   if( cRecieveData != ACK ){                     //Is receive data an ACK signal?
       cErrorStatus = TARGET_RETURN_ERROR; return;
}

/***** Wait for prewrite time *****/
   for( cRetryCounter = 0 ; cRetryCounter < 16 ; cRetryCounter++ ){
       wSec = SWriteWaitTimeCalc( 1, WaitDataTable[cWaitClockSelect].wWaitPreWrite );
                                                  //Calculates prewrite time (1-ms units)
       wMiliSec = ( wSec % 1000 );                //Wait time (1-ms units)
       wSec = wSec / 1000;                        //Wait time (1-µs units)
```

```
      cTimerFlag = WAIT_START;
      do{                                       //Waits up to 1 second
         SWaitMiliSec( wMiliSec );
      }while( cTimerFlag == WAIT_NOW );

      for( cTimerFlag = WAIT_START ; 0 < wSec ; wSec-- ){
                                                //Any wait beyond 1 second?
         do{
            SWaitMiliSec( 1000 );               //Waits in 1-s units
         }while( cTimerFlag == WAIT_NOW);
      }

/***** Get status *****/
      MGetStatus();
      if( cErrorStatus != NO_ERROR )  break;    //Any errors?

      if( cTargetStatus == PRE_WRITING_NOW )continue;
                                                //Prewrite in progress?  YES
      else  return;                             //Ends prewrite
   }
   cErrorStatus = STATUS_NO_RETURN;             //Retries = 16 times
                                                //Status no return
}
```

4.5.12 Erase command

(1) Flow chart

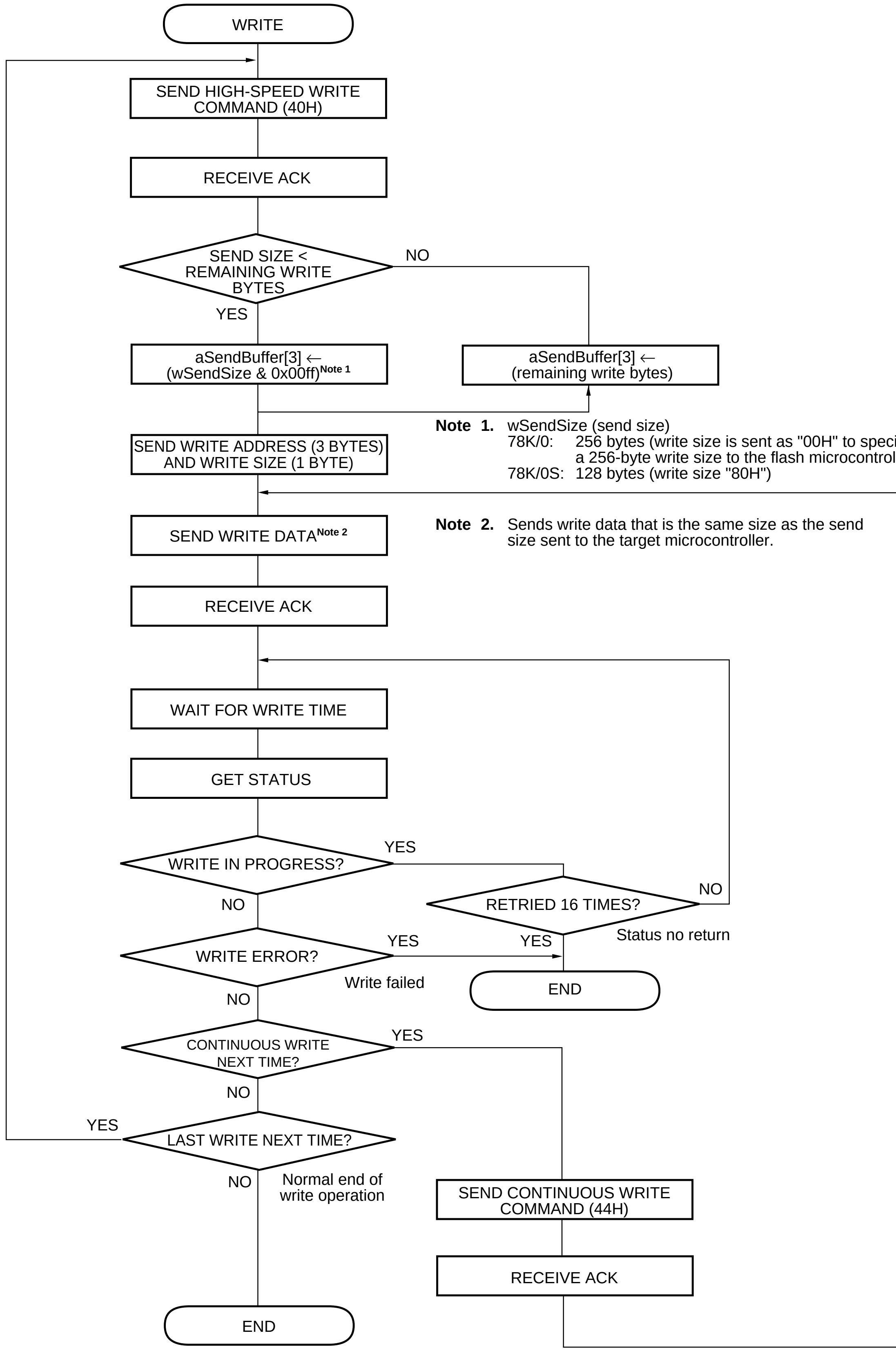
(2) Sample program

```
#pragma sfr                                             //Uses sfr area

#include "DATTYPE.H"                                    //Data type definition file
#include "sram.h"                                       //RAM external access definition file
#include "constant.h"                                   //Constant value definition file

/*-----------------------------------------------
        FUNCTION PROTOTYPE DECLARATION
-----------------------------------------------*/
void MPreWrite( void );                                 //Prewrites
void MGetStatus( void );                                //Gets status
void SDataSend( Word SendSize , Byte *SendDataAddress );
                                                        //Sends data
void SDataRecieve( Word wRecieveDataSize );             //Receives data
void SWait( DWord dwWaitClock );                        //Wait
Word SWaitTimeCalcFlMemSize( Word wWaitClock );//Calculates wait time
void SWaitMiliSec( Word wWaitTime );                    //Wait time (1-ms units)
void SWaitMicroSec( Word wCrRegData );                  //Wait time (1-μs units)

/**********************************************************
*                       Erase command                     *
*       Global variables:   cSendData       Send data         *
*                           cRecieveData    Receive data      *
*                           cErrorStatus    Error status      *
*                           cTargetStatus   Target status     *
*                           cRetryCounter   Retry counter     *
*                           cTimerFlag      Timer flag        *
*                           cRetryCounter   Retry counter     *
*                           wWaitTimeComAck COM-ACK wait time *
*                                                             *
*       Local variables:    wTotalEraseTime Total erase time  *
*                           wSec            Wait time (1-s units)  *
*                           wMiliSec        Wait time (1-ms units) *
**********************************************************/
void MErase( void ){

   register Word wTotalEraseTime;                       //Total erase time
   register Word wSec;                                  //Wait time (1-s units)
   register Word wMiliSec;                              //Wait time (1-ms units)

/***** Send prewrite command *****/
   MPreWrite();                                         //Prewrite command
   if( cErrorStatus != NO_ERROR )  return;              //Any errors?

   SWait(( DWord ) WaitDataTable[cWaitClockSelect].wWaitComToCom );
                                                        //Wait time between commands

   for( wTotalEraseTime = 0 ;                           //Initializes total erase time
        wTotalEraseTime <= 2000 ;                       //Total erase time up to 20 seconds (10-ms units)
        wTotalEraseTime += wParEraseTime ){
```

```
/***** Send command *****/
      cSendData = CMD_CHIP_ERASE;
      SDataSend( 1, &cSendData );                  //Sends erase command
      if( cErrorStatus != NO_ERROR )  return;      //Any errors?

      SWaitMicroSec( wWaitTimeComAck );            //Wait time between sending command and
                                                   //receiving ACK signal

/***** Receive ACK *****/
      SDataRecieve( 1 );                           //Receives ACK signal
      if( cErrorStatus != NO_ERROR )  return;      //Any errors?
      if( cRecieveData != ACK ){                   //Is receive data an ACK signal?
          cErrorStatus = TARGET_RETURN_ERROR;
          return;
      }

/***** Wait for erase time + blank check time *****/

      for( cRetryCounter = 0 ; cRetryCounter < 16 ; cRetryCounter++){
                                                   //Maximum of 16 retries
          wSec = SWaitTimeCalcFlMemSize(  WaitDataTable[cWaitClockSelect].wWaitErase );
                                                   //Calculates blank check time (1-ms units)
          if( 0 == cRetryCounter ){                //Retries do not include erase time (blank check
             wSec += (wParEraseTime * 10);         //time + (erase time × 10)) (ms)
          }                                        //wParEraseTime uses 10-ms units

          wMiliSec = wSec % 1000;                  //Wait time (1-ms units)
          wSec /= 1000;                            //Wait time (1-s units)

          cTimerFlag = WAIT_START;
          do{                                      //Waits up to 1 s
             SWaitMiliSec( wMiliSec );
          }while( cTimerFlag == WAIT_NOW );

          for( cTimerFlag = WAIT_START ; 0 < wSec ; wSec-- ){
                                                   //Any wait beyond 1 second?
             do{
                SWaitMiliSec( 1000 );              //Waits in 1-s units
             }while( cTimerFlag == WAIT_NOW);
          }

/***** Get status *****/
        MGetStatus();
        if( cErrorStatus != NO_ERROR )  return;    //Any errors?

        if( cTargetStatus == ERASING_NOW )continue;
                                                   //Erase in progress?  YES
        else if( cTargetStatus == BLANK_CHEK_FAILED )break;
                                                   //Any blank check errors?
        else if( cTargetStatus == READY )  return;
                                                   //Normal end of erase operation
      }                                            //else: erase in progress (retries getting status)
      if( cRetryCounter >= 16 ){                   //Retried 16 times?
          cErrorStatus = STATUS_NO_RETURN;         //Status no return
          return;
      }                                            //else:  retries blank check error erase
   }
   cErrorStatus = ERASE_FAILED;                    //Judged as erase error if blank check error occurs
                                                   //after total erase time exceeds 20 seconds
}
```

4.5.13 High-speed write/continuous write command

(1) Flow chart



Note 1. wSendSize (send size)
78K/0: 256 bytes (write size is sent as "00H" to specify a 256-byte write size to the flash microcontroller)
78K/0S: 128 bytes (write size "80H")

Note 2. Sends write data that is the same size as the send size sent to the target microcontroller.

(2) Sample program

```
#pragma sfr                                      //Uses sfr area

#include "DATTYPE.H"                             //Data type definition file
#include "sram.h"                                //RAM external access definition file
#include "constant.h"                            //Constant value definition file

/*-------------------------------------------
        FUNCTION PROTOTYPE DECLARATION
-------------------------------------------*/
void MInternalVerify( void );                    //Internal verify
void MGetStatus( void );                         //Gets status
void SDataSend( Word SendSize , Byte *SendDataAddress );
                                                 //Sends data
void SDataRecieve( Word wRecieveDataSize );      //Receives data
void SWait( DWord dwWaitClock );                 //Wait
void SWaitMiliSec( Word wWaitTime );             //Wait time (1-ms units)
Word SWriteWaitTimeCalc( Byte cWriteOrPreWrite , Word wWriteSize);
                                                 //Calculates write time
void SWaitMicroSec( Word wCrRegData );           //Wait time (1-μs units)

/*******************************************************************************
*               High-speed/continuous write command                            *
*     Global variables:  cSendData              Send data                      *
*                        cRecieveData           Receive data                   *
*                        cErrorStatus           Error status                   *
*                        dwParStartAddress      Write start address            *
*                        dwParEndAddress        Write end address              *
*                        cTargetStatus          Target status                  *
*                        cParTargetSeries       Target series                  *
*                        cTimerFlag             Timer flag                     *
*                        cRetryCounter          Retry counter                  *
*                        wSendSize              Send size                      *
*                        wWaitTimeComAck        COM-ACK wait time              *
*                        wWaitTimeDataAck       DATA-ACK wait time             *
*                                                                              *
*     Local variables:   *dwWriteDataAddress    Address where write data is stored *
*                        *dwWriteEndAddress     End address of write data      *
*                        dwWork                 Work                           *
*                        wWaitTime              Write wait time                *
*                        wWorkPointer           Work                           *
*                        cHighSpeedWriteFlag    High-speed write flag          *
*                                                                              *
*   In this sample program, data stored in external ROM is written to the flash microcontroller.  The *
*   external ROM's start address is 20000H (start address of example memory area in 78K4).  This *
*   address is declared and embedded in the program as follows in this sample program. *
*                                                                              *
*   #define USER_DATA_ADDRESS  0x20000     //Start address of storage area for write data *
*******************************************************************************/
void MProgram( void ){
   register Byte    *dwWriteDataAddress;         //Address where write data is stored
   register Byte    *dwWriteEndAddress;          //End address of write data
   register Dword   dwWork;                      //Work
   register Word    wWaitTime;                   //Write wait time
   register Word    wWorkPointer;                //Work
   register Byte    cHighSpeedWriteFlag;         //High-speed write flag

   cHighSpeedWriteFlag = 1;                      //Sets high-speed write flag
```

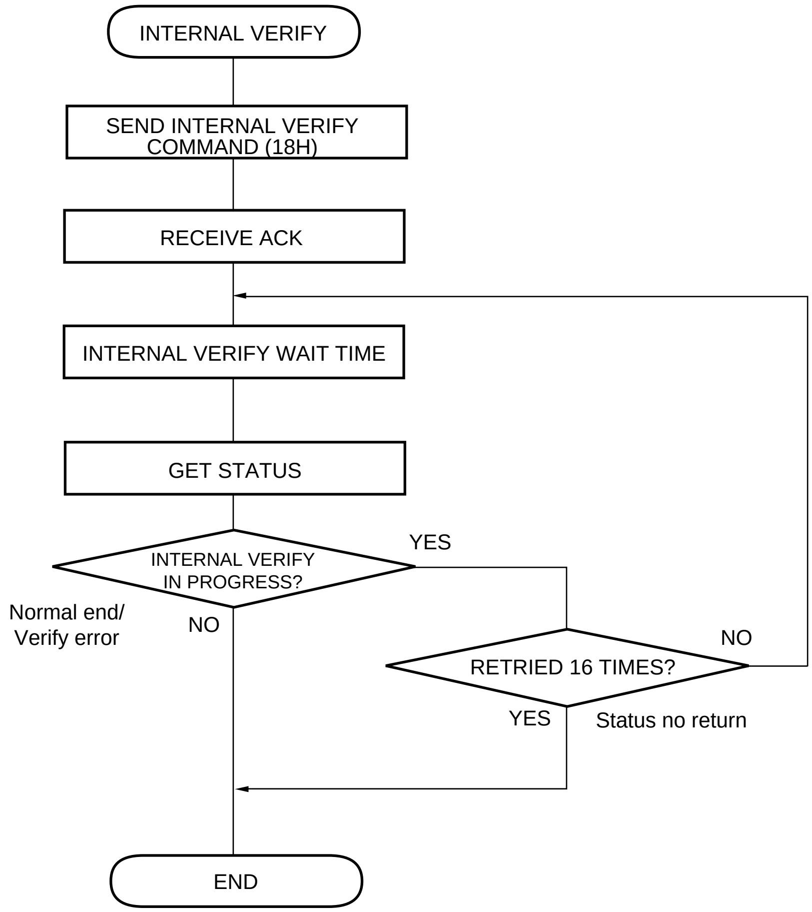
```
   dwWriteDataAddress = (Byte *)USER_DATA_ADDRESS;
                                                  //Start address of storage area for write data
   dwWriteDataAddress += dwParStartAddress;       //+ write start address
   dwWriteEndAddress = (Byte *)USER_DATA_ADDRESS;
                                                  //End address of storage area for write data
   dwWriteEndAddress += dwParEndAddress;          //+ write end address

   if ( cParTargetSeries == K0 ){                 //Specifies send size
       wSendSize = 0x0100;                        //78K/0:  256 bytes
   }else{
       wSendSize = 0x0080;                        //78K/0S:  128 bytes
   }

   while( dwWriteDataAddress <= dwWriteEndAddress ){
                                                  //Continue until write end address

/*****  Send high-speed write command  *****/
       cSendData = CMD_HIGH_SPEED_WRITE;
          SDataSend( 1, &cSendData );             //Sends high-speed write command
       if( cErrorStatus != NO_ERROR )  return;    //Any errors?

       SWaitMicroSec( wWaitTimeComAck );          //Wait time between sending command and
                                                  //receiving ACK signal

/*****  Receive ACK  *****/
       SDataRecieve( 1 );                         //Receives ACK signal
       if( cErrorStatus != NO_ERROR )  return;    //Any errors?
       if( cRecieveData != ACK ){                 //Is receive data an ACK signal?
          cErrorStatus = TARGET_RETURN_ERROR;
          return;
       }

       SWaitMicroSec( wWaitTimeAckData            //Wait time between receiving ACK signal and
                                                  //sending data

/*****  Send send size and write start address  *****/

       dwWork = dwWriteDataAddress - (Byte *)USER_DATA_ADDRESS;
                                                  //Sets write start address to send buffer
       aSendBuffer[ 2 ] =  (Byte)(dwWork & 0x000000ff);
                                                  //(high address)
       aSendBuffer[ 1 ] =  (Byte)((dwWork >>= 8) & 0x000000ff);
                                                  //(mid address)
       aSendBuffer[ 0 ] =  (Byte)((dwWork >>= 8) & 0x000000ff);
                                                  //(low address)

                                                  //Sets send size
       if (!( ( DWord )wSendSize < ( dwWriteEndAddress + 1 - dwWriteDataAddress ))){
           wSendSize = ( Word )(dwWriteEndAddress + 1 - dwWriteDataAddress );
       }
       aSendBuffer[ 3 ] = ( Byte )( wSendSize & 0x00ff );

       SDataSend( 4 , aSendBuffer );              //Sends address (3 bytes) and send size
       if( cErrorStatus != NO_ERROR )  return;    //Any errors?

       do {

/*****  Send write data  *****/
                                                  //Stores write data in send buffer
          for( wWorkPointer = 0 ; wWorkPointer < wSendSize ; wWorkPointer++ ){
              aSendBuffer[ wWorkPointer ] = *dwWriteDataAddress;
              dwWriteDataAddress++;
          }
```

```
        SDataSend( wSendSize , aSendBuffer ); //Send buffer contents
        if( cErrorStatus != NO_ERROR )  return;
                                             //Any errors?
        SWaitMicroSec( wWaitTimeDataAck );   //Wait time between sending data and receiving
                                             //ACK signal

/***** Receive ACK *****/
        SDataRecieve( 1 );                   //Receives ACK signal
        if( cErrorStatus != NO_ERROR )  return;
                                             //Any errors?
        if( cRecieveData != ACK ){           //Is receive data an ACK signal?
            cErrorStatus = TARGET_RETURN_ERROR;
            return;
        }


/***** Wait for write time *****/
        wWaitTime = SWriteWaitTimeCalc( 0, wSendSize );
                                             //Calculates write wait time (1-ms units)
        for( cRetryCounter = 0 ; cRetryCounter < 16 ; cRetryCounter++){
            cTimerFlag = WAIT_START;
            do{
               SWaitMiliSec( wWaitTime );    //Waits for write time
            }while( cTimerFlag == WAIT_NOW);

            /***** Get status *****/
            MGetStatus();
            if( cErrorStatus != NO_ERROR )   return;
                                             //Any errors?

            if( cTargetStatus == PROGRAMING_NOW )continue;
                                             //Write in progress?  YES
            else if( cTargetStatus == PROGRAM_FAILED ){
               cErrorStatus = PROGRAM_FAILED;
                                             //Program Failed
               return;
            }else if( cTargetStatus == READY ) break;
                                             //Normal end of write operation
        }                                    //else cTargetStatus = 0x40 (during write)
        if( cRetryCounter >= 16 ){           //Retried wait 16 times?
            cErrorStatus = STATUS_NO_RETURN; //Status no return
            return;
        }

        if ( wSendSize < ( dwWriteEndAddress + 1 - dwWriteDataAddress)){
            cHighSpeedWriteFlag = 0;         //Uses continuous write command next time
        }else{
            cHighSpeedWriteFlag = 1;         //Uses high-speed write command next time
        }

        if ( cHighSpeedWriteFlag == 0 ){     //Use continuous write command next time?

/***** Send continuous write command *****/
             cSendData = CMD_CONTINUE_WRITE; //No
             SDataSend( 1, &cSendData );     //Sends continuous write command
             if( cErrorStatus != NO_ERROR )  return;
                                             //Any errors?

             SWaitMicroSec( wWaitTimeComAck );
                                             //Wait time between sending command and receive
                                             //ACK signal
             SDataRecieve( 1 );              //Receives ACK signal
             if( cErrorStatus != NO_ERROR )  return;
                                             //Any errors?
```

```
                if( cRecieveData != ACK ){        //Is receive data an ACK signal?
                cErrorStatus = TARGET_RETURN_ERROR;
                return;
                }
                SWaitMicroSec( wWaitTimeAckData );
                                                  //Wait time between receiving ACK signal and
                                                  //sending data
           }
        }while( cHighSpeedWriteFlag == 0 );       //Use high-speed write command next time?  No
    }
    SWait(( DWord )WaitDataTable[cWaitClockSelect].wWaitComToCom );
                                                  //Wait time between commands

/***** Send internal verify command *****/
    MInternalVerify();                            //Internal verify command
}
```

4.5.14 Internal verify command

(1) Flow chart



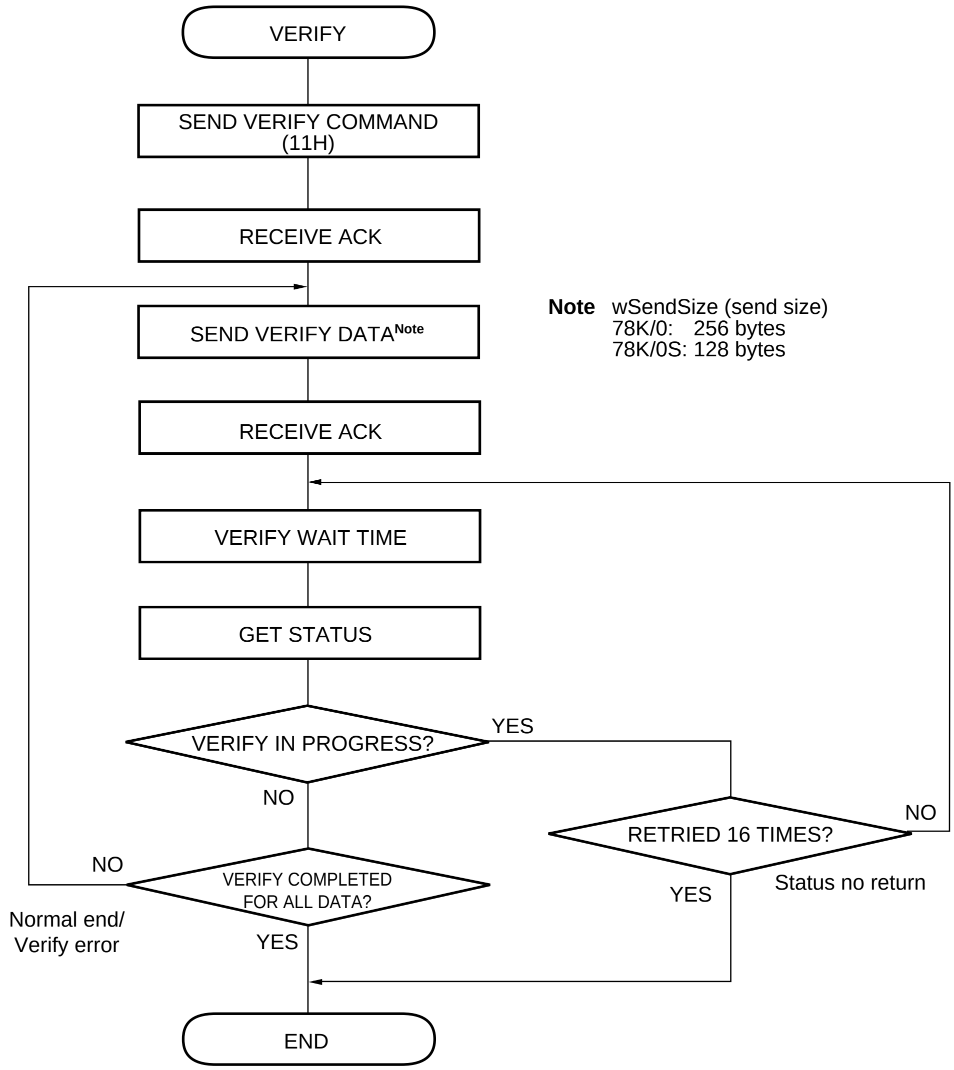
(2) Sample program

```
#pragma sfr                                           //Uses sfr area

#include "DATTYPE.H"                                  //Data type definition file
#include "sram.h"                                     //RAM external access definition file
#include "constant.h"                                 //Constant value definition file

/*-----------------------------------------------
        FUNCTION PROTOTYPE DECLARATION
-----------------------------------------------*/
void MGetStatus( void );                              //Gets status
void SDataSend( Word SendSize , Byte *SendDataAddress );
                                                      //Sends data
void SDataRecieve( Word wRecieveDataSize );           //Receives data
void SWait( DWord dwWaitClock );                      //Wait
Word SWaitTimeCalcFlMemSize( Word wWaitClock );       //Calculates wait time
void SWaitMiliSec( Word wWaitTime );                  //Wait time (1-ms units)
void SWaitMicroSec( Word wCrRegData );                //Wait time (1-μs units)

/**********************************************************
*                 Internal verify command                 *
*     Global variables:  cSendData        Send data       *
*                        cRecieveData     Receive data    *
*                        cErrorStatus     Error status    *
*                        cTargetStatus    Target status   *
*                        cTimerFlag       Timer flag      *
*                        cRetryCounter    Retry counter   *
*                        wWaitTimeComAck  COM-ACK wait time *
*                                                         *
*     Local variables:   wSec             Wait time (1-s units)  *
*                        wMiliSec         Wait time (1-ms units) *
**********************************************************/
void MInternalVerify( void ){

   register Word wSec;                                //Wait time (1-s units)
   register Word wMiliSec;                            //Wait time (1-ms units)

/***** Send command *****/
   cSendData = CMD_CHIP_IVRF;
   SDataSend( 1, &cSendData );                        //Sends internal verify command
   if( cErrorStatus != NO_ERROR )  return;            //Any errors?

   SWaitMicroSec( wWaitTimeComAck );                  //Wait time between sending command and
                                                      //receiving ACK signal

/***** Receive ACK *****/
   SDataRecieve( 1 );                                 //Receives ACK signal
   if( cErrorStatus != NO_ERROR )  return;            //Any errors?
   if( cRecieveData != ACK ){
       cErrorStatus = TARGET_RETURN_ERROR; return;
                                                      //Target return error
   }

/***** Wait for internal verify time *****/
   for( cRetryCounter = 0 ; cRetryCounter < 16 ; cRetryCounter++ ){
       wSec = SWaitTimeCalcFlMemSize(  WaitDataTable[cWaitClockSelect].
       wWaitInternalVerify );
                                                      //Calculates internal verify wait time (1-ms units)
       wMiliSec = wSec % 1000;                        //Wait time (1-ms units)
       wSec = wSec / 1000;                            //Wait time (1-s units)
```

```
        cTimerFlag = WAIT_START;
        do{                                      //Waits up to 1 second
           SWaitMiliSec( wMiliSec );
        }while( cTimerFlag == WAIT_NOW );

        for( cTimerFlag = WAIT_START ; 0 < wSec ; wSec-- ){
                                                 //Any wait beyond 1 second?
           do{
              SWaitMiliSec( 1000 );              //Waits in 1-s units
           }while( cTimerFlag == WAIT_NOW);
        }

/***** Get status *****/
        MGetStatus();
        if( cErrorStatus != NO_ERROR )  break;   //Any errors?

        if( cTargetStatus == VERIFYING_NOW )continue;
                                                 //Internal verify in progress?  YES
        else if( cTargetStatus == VERIFY_ERROR ){
                                                 //Any verify errors?
           cErrorStatus = VERIFY_ERROR;          //Sets verify error
              return;                            //Verify error
        }else return;                            //Normal end of verify operation
    }
    cErrorStatus = STATUS_NO_RETURN;             //Retries 16 times?
                                                 //Status no return
}
```

4.5.15 Verify command

(1) Flow chart



Note wSendSize (send size)
78K/0: 256 bytes
78K/0S: 128 bytes

(2) Sample program

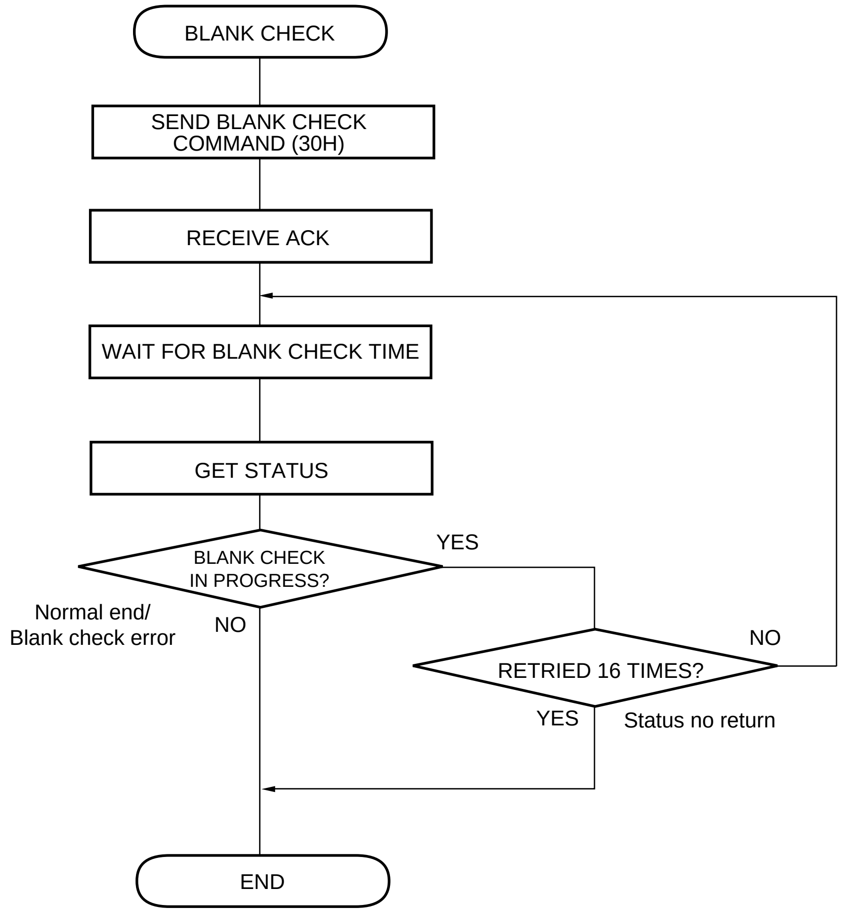
```
#pragma sfr                                             //Uses sfr area

#include "DATTYPE.H"                                    //Data type definition file
#include "sram.h"                                       //RAM external access definition file
#include "constant.h"                                   //Constant value definition file

/*----------------------------------------------
        FUNCTION PROTOTYPE DECLARATION
-----------------------------------------------*/
void MGetStatus( void );                                //Gets status
void SDataSend( Word SendSize , Byte *SendDataAddress );
                                                        //Sends data
void SDataRecieve( Word wRecieveDataSize );             //Receives data
void SWait( DWord dwWaitClock );                        //Wait
void SWaitMiliSec( Word wWaitTime );                    //Wait time (1-ms units)
void SWaitMicroSec( Word wCrRegData );                  //Wait time (1-μs units)

/****************************************************************************
*                               Verify command                              *
*    Global variables:  cSendData               Send data                   *
*                       cRecieveData            Receive data                *
*                       cErrorStatus            Error status                *
*                       cTargetStatus           Target status               *
*                       cParTargetSeries        Target series               *
*                       sSig.dwSigLastAddress   Flash memory end address    *
*                                               (fetched using silicon signature command) *
*                       cRetryCounter           Retry counter               *
*                       wWaitTimeComAck         COM-ACK wait time           *
*                       wWaitTimeDataAck        DATA-ACK wait time          *
*                                                                           *
*    Local variables:   *dwVerifyDataAddress    Address where verify data is stored *
*                       *dwVerifyEndAddress     End address for verify data *
*                       wWorkPointer            Work                        *
*                                                                           *
*   In this sample program, the data stored in external ROM and the contents of the flash *
*   microcontroller's flash memory are verified.                            *
*   The start address in external ROM is 20000H (which is the starting address of external memory *
*   area in 78K/4 devices).  In this sample program, the following declaration is included in the *
*   program.                                                                *
*                                                                           *
*    #define USER_DATA_ADDRESS    0x20000   //Start address of write data storage area *
****************************************************************************/
void MVerify( void ){
   register Byte *dwVerifyDataAddress;                  //Address where verify data is stored
   register Byte *dwVerifyEndAddress;                   //End address of verify data
   register Word wWorkPointer;                          //Work

/***** Send command *****/
   cSendData = CMD_CHIP_VRF;
   SDataSend( 1, &cSendData );                          //Sends verify command
   if( cErrorStatus != NO_ERROR )  return;              //Any errors?

   SWaitMicroSec( wWaitTimeComAck );                    //Wait time between sending command and
                                                        //receiving ACK signal

/***** Receive ACK *****/
   SDataRecieve( 1 );                                   //Receives ACK signal
   if( cErrorStatus != NO_ERROR )  return;              //Any errors?
   if( cRecieveData != ACK ){                           //Is receive data an ACK signal?
```

```
        cErrorStatus = TARGET_RETURN_ERROR;
        return;
    }

    SWaitMicroSec( wWaitTimeAckData );          //Wait time between receiving an ACK signal and
                                                //sending data

    dwVerifyDataAddress = ( Byte * )USER_DATA_ADDRESS;
                                                //Sets address where verify data is stored
    dwVerifyEndAddress = ( Byte * )USER_DATA_ADDRESS;
                                                //USER_DATA_ADDRESS = 20000H
    dwVerifyEndAddress += sSig.dwSigLastAddress; //Sets end address of verify data
                                                //The target for verification is the target device's
                                                //entire flash memory area.
                                                //End of verification is judged based on the last
                                                //address fetched by the silicon signature
                                                //command.


    if ( cParTargetSeries == K0 ){              //Send size
       wSendSize = 256;                         //78K/0:  256 bytes
    }else{
       wSendSize = 128;                         //78K/0S:  128 bytes
    }

    do {                                        //Continues until all verify data has been sent.
       for( wWorkPointer = 0 ; wWorkPointer < wSendSize ; wWorkPointer++ ){
       aSendBuffer[ wWorkPointer ] = *dwVerifyDataAddress;
                                                //Stores verify data in send buffer
       dwVerifyDataAddress++;
    }

/***** Send verify data *****/
    SDataSend( wSendSize , aSendBuffer );       //Sends contents of buffer
                                                //(78K/0:  256 bytes, 78K/0S:  128 bytes)
    if( cErrorStatus != NO_ERROR )  return;     //Any errors?

    SWaitMicroSec( wWaitTimeDataAck );          //Wait time between sending data and receiving an
                                                //ACK signal



/***** Receive ACK *****/
    SDataRecieve( 1 );                          //Receives ACK signal
    if( cErrorStatus != NO_ERROR )  return;     //Any errors?
    if( cRecieveData != ACK ){                  //Is receive data an ACK signal?
        cErrorStatus = TARGET_RETURN_ERROR;
        return;
    }
    for( cRetryCounter = 0 ; cRetryCounter < 16 ; cRetryCounter++){
                                                //Maximum of 16 retries
       SWait(  WaitDataTable[cWaitClockSelect].dwWaitVerify );
                                                //Waits for verify time

/***** Get status *****/
        MGetStatus();
        if( cErrorStatus != NO_ERROR )  return; //Any errors?

        if( cTargetStatus == VERIFYING_NOW )continue;
                                                //Verify in progress? YES
        else break;                             //Ends verification of send size
    }
    if( cRetryCounter >= 16 ){
        cErrorStatus = STATUS_NO_RETURN;        //Status no return
```

```
        return;
    }
    }while( dwVerifyDataAddress < dwVerifyEndAddress );
                                                        //Judges end of verification
    if( cTargetStatus == VERIFY_ERROR ){
        cErrorStatus = VERIFY_ERROR;                    //Verification error,
    }                                                   //else:  normal end of verification
}
```

4.5.16 Blank check command

(1) Flow chart



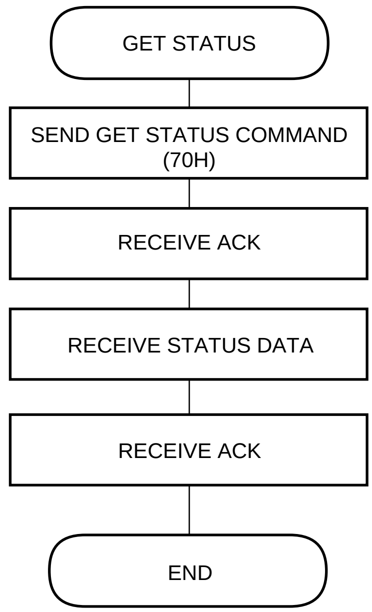
(2) Sample program

```
#pragma sfr                                   //Uses sfr area

#include "DATTYPE.H"                          //Data type definition file
#include "sram.h"                             //RAM external access definition file
#include "constant.h"                         //Constant value definition file

/*----------------------------------------------
        FUNCTION PROTOTYPE DECLARATION
----------------------------------------------*/
void MGetStatus( void );                      //Gets status
void SDataSend( Word SendSize , Byte *SendDataAddress );
                                              //Sends data
void SDataRecieve( Word wRecieveDataSize );   //Receives data
void SWaitMiliSec( Word wWaitTime );          //Wait time (1-ms units)
void SWaitMicroSec( Word wCrRegData );        //Wait time (1-μs units)
Word SWaitTimeCalcFlMemSize( Word wWaitClock ); //Calculates wait time


/*********************************************************
*                   Blank check command                  *
*  Global variables: cSendData        Send data          *
*                    cRecieveData     Receive data       *
*                    cErrorStatus     Error status       *
*                    cTargetStatus    Target status      *
*                    cTimerFlag       Timer flag         *
*                    cRetryCounter    Retry counter      *
*                    wWaitTimeComAck  COM-ACK wait time  *
*                                                        *
*  Local variables:  wSec             Wait time (1-s units)  *
*                    wMiliSec         Wait time (1-ms units) *
*                                                        *
*********************************************************/
void MBlankChek( void ){
   register Word wSec;                        //Wait time (1-s units)
   register Word wMiliSec;                    //Wait time (1-ms units)

/***** Send command *****/
   cSendData = CMD_CHIP_BLN;
   SDataSend( 1, &cSendData );                //Sends blank check command
   if( cErrorStatus != NO_ERROR )  return;    //Any errors?
   SWaitMicroSec( wWaitTimeComAck );          //Wait time between sending command and
                                              //receiving an ACK signal

/***** Receive ACK *****/
   SDataRecieve( 1 );                         //Receives ACK signal
   if( cErrorStatus != NO_ERROR )  return;    //Any errors?
   if( cRecieveData != ACK ){                 //Is receive data an ACK signal?
       cErrorStatus = TARGET_RETURN_ERROR;    //Sets error status
       return;
   }

/***** Wait for blank check time *****/
   for( cRetryCounter = 0 ; cRetryCounter < 16 ; cRetryCounter++ ){
                                              //Maximum of 16 retries
       wSec = SWaitTimeCalcFlMemSize( WaitDataTable[cWaitClockSelect]wWaitBlankChek );
                                              //Calculates blank check time (1-ms units)
       wMiliSec = wSec % 1000;                //1-ms units
       wSec = wSec / 1000;                    //1-s units
       cTimerFlag = WAIT_START;
       do{                                    //Waits up to 1 second
```

```
            SWaitMiliSec( wMiliSec );
        }while( cTimerFlag == WAIT_NOW );

        for( cTimerFlag = WAIT_START ; 0 < wSec ; wSec-- ){
                                                      //Any wait beyond 1 second?
            do{
                SWaitMiliSec( 1000 );                 //Waits in 1-s units
            }while( cTimerFlag == WAIT_NOW);
        }

/*****  Get status  *****/
        MGetStatus();
        if( cErrorStatus != NO_ERROR )  return;      //Any errors?

        if( cTargetStatus == BLANK_CHEK_NOW )  continue;
                                                      //Blank check in progress?  YES
        else if( cTargetStatus == BLANK_CHEK_FAILED ){
                                                      //Any blank check errors?
            cErrorStatus = BLANK_CHEK_FAILED;         //YES:  Sets error status
            return;                                   //Blank check error
        }else  return;                                //Normal end of blank check
    }
    cErrorStatus = STATUS_NO_RETURN;                  //Retry count = 16
                                                      //Status no return
}
```

4.5.17 Get status command

(1) Flow chart



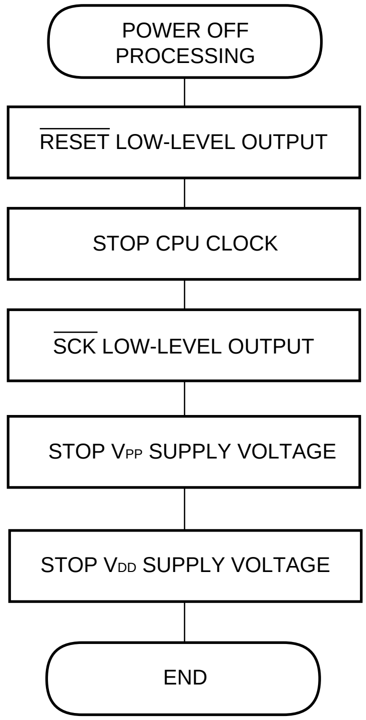
(2) Sample program

```
#pragma sfr                                           //Uses sfr area

#include "DATTYPE.H"                                  //Data type definition file
#include "sram.h"                                     //RAM external access definition file
#include "constant.h"                                 //Constant value definition file

/*-----------------------------------------------
        FUNCTION PROTOTYPE DECLARATION
-----------------------------------------------*/
void SDataSend( Word SendSize , Byte *SendDataAddress );
                                                      //Sends data
void SDataRecieve( Word wRecieveDataSize );           //Receives data
void SWait( Word wWaitClock );                        //Wait
void SWaitMicroSec( Word wCrRegData );                //Wait time (1-µs units)

/**********************************************************
*                    Get status command                   *
*     Global variables:  cSendData          Send data          *
*                        cRecieveData       Receive data       *
*                        cErrorStatus       Error status       *
*                        cTargetStatus      Target status      *
*                        cRetryCounter      Retry counter      *
*                        wWaitTimeComAck    COM-ACK wait time  *
*                        wWaitTimeAckData   ACK-DATA wait time *
*                        wWaitTimeDataAck   DATA-ACK wait time *
**********************************************************/
void MGetStatus( void ){

/***** Send get status command *****/
   cSendData = CMD_STATUS;
   SDataSend( 1, &cSendData );                        //Sends get status command (70H)
   if( cErrorStatus != NO_ERROR )  return;            //Any errors?

   if( cCommunicationMethod != UART ){
       SWaitMicroSec( wWaitTimeComAck );              //Wait time between sending command and
                                                      //receiving an ACK signal
   }

/***** Receive ACK *****/
   SDataRecieve( 1 );                                 //Receives ACK signal
   if( cErrorStatus != NO_ERROR )  return;            //Any errors?
   if( cRecieveData != ACK ){                         //Is receive data an ACK signal?
       cErrorStatus = TARGET_RETURN_ERROR;  return;
   }

   if( cCommunicationMethod != UART ){                //If other than UART communications
       SWaitMicroSec( wWaitTimeAckData );             //Wait time between receiving ACK signal and
                                                      //receiving data
   }

/***** Receive status data *****/
   SDataRecieve( 1 );                                 //Receives status data
   if( cErrorStatus != NO_ERROR )  return;            //Any errors?
   cTargetStatus = cRecieveData;                      //Stores status data

   if( cCommunicationMethod != UART ){                //If other than UART communications
       SWaitMicroSec( wWaitTimeDataAck );             //Wait time between receiving data and receiving an
                                                      //ACK signal
   }
```

```
/***** Receive ACK *****/
   SDataRecieve( 1 );                        //Receives ACK signal
   if( cErrorStatus != NO_ERROR )  return;   //Any errors?
   if( cRecieveData != ACK )                 //Is receive data an ACK signal?
       cErrorStatus = TARGET_RETURN_ERROR;
   }
```

4.5.18 Power off processing

(1) Flow chart



(2) Sample program

```
#pragma sfr                                   //Uses sfr area

#include "DATTYPE.H"                          //Data type definition file
#include "sram.h"                             //RAM external access definition file
#include "constant.h"                         //Constant value definition file

/*********************************************************
*                   Power off processing                 *
*********************************************************/
void MPowerOff(void){
   P6.7 = 0;                                  //Low-level output of RESET signal

   P0.1 = 1;                                  //Stops clock supply

   P3.2 = 0;                                  //High-level output of SCK signal

   DACS1 = 0;                                 //Stops VPP supply

   DACS0 = 0;                                 //Stops VDD supply

}
```

4.6 Other Sample Programs

4.6.1 Subroutines

```
#pragma sfr                                   //Uses sfr area
#pragma NOP

#include "DATTYPE.H"                          //Data type definition file
#include "sram.h"                             //RAM external access definition file
#include "constant.h"                         //Constant value definition file

/*---------------------------------------------/
                   Subroutine

   This is a subroutine that is used in the sample program.
   This subroutine must be included when using the sample
   program.
---------------------------------------------*/

/*****************************************************************
*              Wait processing in microsecond units              *
*  Local variable:  cWork Work                                   *
*                                                                *
*  IN:          wCrRegData (compare register setting data) = 1 – 65,535   *
*               A wait of (wCrRegData × 0.8) µs is performed within this routine.  *
*  OUT:         None                                             *
*****************************************************************/
void SWaitMicroSec( Word wCrRegData ){
   register Byte cWork;

      cWork = PRM0;                           //Selects count clock
      cWork &= 0xf0;                          //fxx/16 (0.8 µs )
      cWork |= 0x02;
      PRM0 = cWork;

      cWork = TMC0;                           //Stops operation of TM0
      cWork &= 0xf0;
      cWork |= 0x04;
      TMC0 = cWork;

      CIF00 = 0;                              //Clears interrupt flag

      CR00 = wCrRegData;                      //Sets compare register
      CE0 = 1;                                //Starts TM0


   while( CIF00 != 1 ){                       //Waits for interrupt flag
   }
   CIF00 = 0;                                 //Clears interrupt flag
}

/**************************************************************
*              Wait processing in millisecond units           *
*     Local variable:  cWork          Work                    *
*                                                             *
*     IN:              wWaitTime (wait time data) = 1 – 6,553 ms   *
*                      cTimerFlag = WAIT_START (wait start)   *
*     OUT:             cTimerFlag = WAIT_FINISH (wait end)    *
**************************************************************/
```

```
void SWaitMiliSec( Word wWaitTime ){
   register Byte cWork;

   if( cTimerFlag == WAIT_START ){               //Start wait?

      wWaitTime = wWaitTime * ( 1000 / 100 );   //Calculates compare register value

      cWork = PRM0;                              //Selects count clock
      cWork &= 0xf0;                             //fxx/2,048 (102.4 µs)
      cWork |= 0x09;
      PRM0 = cWork;

      cWork = TMC0;                              //Stops operation of TM0
      cWork &= 0xf0;
      cWork |= 0x04;
      TMC0 = cWork;

      CIF00 = 0;                                 //Clears interrupt flag

      if( wWaitTime == 0 ){
          wWaitTime++;
      }
      CR00 = wWaitTime;                          //Sets compare register
      CE0 = 1;                                   //Starts TM0

      cTimerFlag = WAIT_NOW;                     //Sets timer flag during wait time
   }

   if( CIF00 == 1 ){                             //Waits for interrupt flag
      CIF00 = 0;                                 //Clears interrupt flag
      cTimerFlag = WAIT_FINISH;                  //Ends wait time
   }
}


/**************************************************************************
*                Calculation of wait time for flash memory size          *
* This calculates the wait times during blank check and internal verify operations. *
*                                                                         *
* Number of wait clocks per byte × flash memory capacity / CPU clock (10-kHz unit) / 10 (ms) *
*     Local variables:  dwWaitTime (wait time)                            *
*                                                                         *
*     IN:       wWaitClock (wait clock count data) = 1 – 65,535 clocks    *
*     OUT:      dwWaitTime (wait time) = 1 – 65,535 ms                    *
**************************************************************************/
Word SWaitTimeCalcFlMemSize( Word wWaitClock ){
   register DWord dwWaitTime;                    //Wait time (ms units)

   dwWaitTime = ( DWord )wWaitClock;             //Number of wait clocks per byte
   dwWaitTime *= sSig.dwSigLastAddress;          //Number of wait clocks per byte × total memory
                                                 //size (bytes)
   dwWaitTime = dwWaitTime / ( DWord )wParCpuClockSpeed;
                                                 //CPU clocks (100-µs units)
   dwWaitTime = dwWaitTime / 10;                 //Sets 1-ms units
   dwWaitTime++;                                 //Truncates fraction values

   return ( Word )dwWaitTime;
}
```

```
/*******************************************************************************
*                         Write/Prewrite time calculation                        *
*         IN:    cWriteOrPreWrite (value 0: For write, 1: For prewrite)          *
*                wWriteSize (write size; for write command)                      *
*         OUT:   dwWaitTime (wait time for write/prewrite)                       *
* The formulas used to calculate the write time and prewrite time are shown below.  The wait times during *
* write processing and prewrite processing are based on these calculated write and prewrite times.       *
*                                                                                *
* Write time formula                                                             *
* [(Write processing wait clock count + a) × target write retry count            *
* + (α × 2)] × write byte count / oscillation frequency                          *
* = (((wWaitWrite + α) × cflashWriteRetry + α × 2) × wWriteSize) / wParCpuClockSpeed *
*                                                                                *
* Prewrite time formula                                                          *
* [(Prewrite processing wait clock count + α) × total memory size / oscillation frequency *
* = (PreWriteWait + α) × sSig.dwSigLastAddress / wParCpuClockSpeed               *
*                                                                                *
*     Calculation of value for α                                                 *
*           For 78K/0:     α1 = 2^ (n1 + m1) / fx [n1 = 1 to 4], [m1 = 5 (fixed)]]  *
*           For 78K/0S:    α2 = 2^ (n2 + m2) / fx [n2 = 2 to 5], [m2 = 4 (fixed)]   *
*                                                                                *
*                                                                                *
*      Relationship between target CPU's clock speed and value of n              *
*                                                                                *
*            Oscillation frequency          Value of n                           *
*                                           78K/0[n1]         78K/0S[n2]         *
*          1.00 MHz ≤ fx ≤ 1.28 MHz            1                  2              *
*          1.28 MHz < fx ≤ 2.56 MHz            2                  3              *
*          2.56 MHz < fx ≤ 5.12 MHz            3                  4              *
*          5.12 MHz < fx ≤ 10.0 MHz            4                  5              *
*                                                                                *
*     Given the above:                                                           *
*        α1 = α2 = α                                                             *
*        Therefore, the value of α is the same no matter whether the target is a 78K/0 Series product *
*        or a 78K/0S Series product.                                             *
*        Thus, in the sample program, the value of α can be determined using the formula for calculating *
*        the value of α regardless of whether the target is a 78K/0 Series product or a 78K/0S Series  *
*        product.                                                                *
*******************************************************************************/
Word SWriteWaitTimeCalc( Byte cWriteOrPreWrite , Word wWriteSize){
   register Byte n;                              //Value of n
   register Word wAlpha;                         //Value of α
   register Byte cTargetWriteRetry;              //Target microcontroller's write retry count
   register DWord dwWaitTime;                    //Wait time

   wAlpha = 1;                                   //Initializes α value
   cTargetWriteRetry = 1;                        //1 to 10 times:  In this sample program, the
                                                 //minimum value (1 time) is set.
                                                 //Select value of n (2^n)
   if(( 100 <= wParCpuClockSpeed ) && ( 128 >= wParCpuClockSpeed )){
                                                 //1.00 MHz ≤ fx ≤ 1.28 MHz
       n =  ( 2 + 4 );                           //2^(2 + 4)
   }else  if(( 128 < wParCpuClockSpeed ) && ( 256 >= wParCpuClockSpeed )){
                                                 //1.28 MHz < fx ≤ 2.56 MHz
       n =  ( 3 + 4 );                           //2^(3 + 4)
   }else if(( 256 < wParCpuClockSpeed ) && ( 512 >= wParCpuClockSpeed )){
                                                 //2.56 MHz < fx ≤ 5.12 MHz
       n =  ( 4 + 4 );                           //2^(4 + 4)
   }else  if(( 512 < wParCpuClockSpeed ) && ( 1000 >= wParCpuClockSpeed )){
                                                 //5.12 MHz < fx ≤ 10.0 MHz
```

```
        n =  ( 5 + 4 );                             //2^(5 + 4)
    }
    wAlpha <<= n;                                   //Calculates value of α

    if( cWriteOrPreWrite == 0 ){                    //When waiting for write command
        dwWaitTime = ( DWord )WaitDataTable[cWaitClockSelect].wWaitWrite;
        dwWaitTime += ( DWord )wAlpha;
        dwWaitTime *= cTargetWriteRetry;
        dwWaitTime += ( DWord )(wAlpha * 2 );
        dwWaitTime *= wWriteSize;                   //WriteSize  :  78K/0:  256 bytes
                                                    //           :  78K/0S:  128 bytes

    }else{                                          //During prewrite
        dwWaitTime = ( DWord )WaitDataTable[cWaitClockSelect].wWaitPreWrite;
                                                    //Wait clock count per byte
        dwWaitTime += ( DWord )wAlpha;              //Adds value of α
        dwWaitTime *= sSig.dwSigLastAddress;        //Multiplies flash memory size
    }
        dwWaitTime /= wParCpuClockSpeed;            //100-μs units, divided by oscillation frequency (10-
                                                    //kHz units)

    dwWaitTime /= 10;                               //Converts to ms units
    dwWaitTime++;                                   //Truncates fraction
    return ( Word )dwWaitTime;
}

/**********************************************************
*                           Wait                          *
*      Local variable:  dwWaitMiliSec (wait time in ms units)     *
*                                                         *
*      IN:              dwWaitClock (wait clock count) = 1 – 65,535    *
**********************************************************/
void SWait( DWord dwWaitClock ){
    register DWord dwWaitMiliSec;                   //Wait time in ms units

    dwWaitClock /= ( wParCpuClockSpeed / 100 );     //Converts to μs units
    dwWaitMiliSec = dwWaitClock / 1000;             //Stores ms units

    if( dwWaitMiliSec == 0 ){                       //Any wait beyond 1 ms?
        dwWaitMiliSec++;
    }
    cTimerFlag = WAIT_START;
    do{
        SWaitMiliSec(( Word )dwWaitMiliSec );       //Waits in 1-ms units
    }while(cTimerFlag == WAIT_NOW);
}

/**********************************************************
*                Calculate communication wait time             *
*           IN:     wWaitClock (wait clocks) = 1 – 65,535        *
*           OUT:    wWaitClock[μs × (5/4)] = 1 – 52,428         *
*                   (Used as parameter for SWaitMicroSec processing)   *
**********************************************************/
Word  SWaitTimeCalc( Word wWaitClock ){
    wWaitClock /= ( wParCpuClockSpeed / 100 );      //Converts to μs units
    wWaitClock = (( wWaitClock * 5 ) / 4) ;         //Calculates setting in compare register (μs units)
    if( wWaitClock == 0 ){
        wWaitClock++;
    }
    return wWaitClock;                              //μs units
}
```

```
/***************************************************************
*                   30-µs wait (measured value)                 *
*          Local variable:     cWork          Work              *
***************************************************************/
void SWait30us( void ){
   register Byte cWork;

   for( cWork = 2 ; 0 < cWork ; cWork-- ){
       NOP();
   }
}
```

```
#pragma sfr                                         //Uses sfr area
#pragma NOP

#include "DATTYPE.H"                                //Data type definition file
#include "sram.h"                                   //RAM external access definition file
#include "constant.h"                               //Constant value definition file

/*-----------------------------------------------
        FUNCTION PROTOTYPE DECLARATION
-----------------------------------------------*/
void SWaitMiliSec( Word wWaitTime );                //Wait time (1-ms units)
void SWaitMicroSec( Word wCrRegData );              //Wait time (1-µs units)
void SWait( DWord dwWaitClock );                    //Wait

/*******************************************************************
*                        Send 1 byte of data                               *
*      Global variables:  cSendFlag              Send flag                 *
*                         cCommunicationMethod   Communication method      *
*                         cSendData              Send data                 *
*******************************************************************/
void SByteDataSend( void )
{

   if( cSendFlag == SEND_START ){
       switch( cCommunicationMethod ){
          case UART:                                //Communication method:  UART
                     TXS = cSendData;               //Send
                     break;
          case IIC:                                 //Communication method:  IIC
                     SIO = cSendData;               //Send
                     break;
          case CSI:                                 //Communication method:  3-wire serial
          case PCSI:                                //Communication method:  pseudo 3-wire serial
                     SIO = cSendData;               //Send
          }
       cSendFlag = SEND_NOW;                        //Set to "send in progress"
       }
       switch( cCommunicationMethod ){
          case UART:                                //Communication method:  UART
                  if( STIF == 1){
                      STIF = 0;                     //Clear send finish flag
                      cSendFlag = SEND_FINISH;      //Finish one-byte transfer
                  }
                  break;
          case IIC:                                 //Communication method:  IIC
                  if( CSIIF == 1){
                      CSIIF = 0;                    //Clear send finish flag
                      if( ACKD == 0 ){              //Confirm ACK detection
                          cErrorStatus = IIC_NO_ACK;
                                                    //IIC_NO_ACK
                      }
                      cSendFlag = SEND_FINISH;      //Finish one-byte transfer
                  }
                  break;
          case CSI:                                 //Communication method:  3-wire serial
          case PCSI:                                //Communication method:  pseudo 3-wire serial
                  if( CSIIF == 1){
                      CSIIF = 0;                    //Clear send finish flag
                      cSendFlag = SEND_FINISH;      //Finish one-byte transfer
                  }
   }
}
```

```
/****************************************************************
*            Send slave address (during IIC communication only)            *
*   Global variables:   cSendFlag        Send flag                          *
*                       cTimerFlag       Timer flag                         *
*                       cErrorStatus     Send data                          *
*                                                                           *
*   IN:                 cSendOrRecieve (data transfer direction of slave address)   *
*                       (Transfer direction 0: Send, 1: Receive)           *
****************************************************************/
void SSlaveAddressSend( Byte cSendOrRecieve ){

   STT = 1;                                   //Outputs start condition
   NOP();                                     //Waits for one instruction

   if( cSendOrRecieve == 0 ){                 //slave address
       SIO = ( cParSlaveAddress << 1 );       //Transfer direction:  send (bit0 is low)
   }else{
       SIO = (( cParSlaveAddress << 1 ) | 0x01 );
                                              //Transfer direction:  receive (bit0 is high)
   }

   cTimerFlag = WAIT_START;                   //Timeout setting
   while( CSIIF == 0 ){                       //Waits until sending of slave address is finished
       SWaitMiliSec( 1000 );                  //Time out check
       if( cTimerFlag == WAIT_FINISH ){
           cErrorStatus = TARGET_IS_CLOSED;   //Send failed
           return;
      }
   }
   CSIIF = 0;                                 //Clears send finish flag
   if( ACKD == 0 ){                           //ACK detected?  No
       cErrorStatus = IIC_NO_ACK;             //IIC_NO_ACK
       cSendFlag = SEND_FINISH;               //Finish one-byte transfer
   }
}

/****************************************************************
*                          Send data                                        *
*    Global variables:   cSendFlag               Send flag                  *
*                        cTimerFlag              Timer flag                 *
*                        cErrorStatus            Error status               *
*                        cCommunicationMethod)   Communication method       *
*                                                                           *
*    IN:                 cSendDataSize (send size) = 1 – 256 bytes          *
*                        SendDataAddress (address where send data is stored)   *
****************************************************************/
void SDataSend( Word wSendDataSize , Byte *SendDataAddress ){
   register Word wSendCounter;

   switch(cCommunicationMethod){              //Communication method
       case UART:TXE = 1;                     //UART communication enabled
                 break;
       case IIC:                              //IIC communications
                 SSlaveAddressSend( 0 );      //Direction of slave address transfer   0:  Send
                 break;
       case CSI:                              //3-wire serial/pseudo 3-wire serial transmit enabled
       case PCSI:CTXE = 1;
   }

   for( wSendCounter = 0 ;
   (( wSendCounter < wSendDataSize ) && ( cErrorStatus == NO_ERROR));
   wSendCounter++ ){                          //Continues until send size has been sent
```

```
        cSendFlag = SEND_START;                   //Clears send flag
        cTimerFlag = WAIT_START;                  //Sets timeout
        cSendData = (*SendDataAddress);           //Send data setting
        SendDataAddress++;                        //Address refresh

        while( cSendFlag != SEND_FINISH ){        //Waits until one byte of data has been transmitted
           SWaitMiliSec( 1000 );                  //Time out check

           if( cTimerFlag == WAIT_FINISH ){
               cErrorStatus = TARGET_IS_CLOSED;   //Send failed
               break;
           }
           SByteDataSend();
        }
        if((( wSendCounter + 1 ) < wSendDataSize )
           && (cCommunicationMethod != PCSI)){    //Inserts a wait if communication method is pseudo
                                                  //3-wire serial (since SO has been high impedance)
           SWaitMicroSec( wWaitTimeDataData );    //Wait time between sending two sets of data
        }
    }

    switch(cCommunicationMethod){                 //Communication method
        case UART:TXE = 0;                        //UART communications disabled
        case IIC:                                 //IIC communications
                   SPT = 1;                       //Outputs stop condition
                   break;
        case CSI:
        case PCSI:
                   CTXE = 0;                      //CSI.PCSI transmit disabled
    }
}

/*******************************************************************
*                      Receive one byte of data                      *
*     Global variables:  cCommunicationMethod   Communication method  *
*                        cRecieveData           Receive data          *
*                        cRecieveFlag           Receive flag          *
*                        cErrorStatus           Error status          *
*******************************************************************/
void SByteDataRecieve( void ){

    if( cRecieveFlag == RECIEVE_START ){
        switch( cCommunicationMethod ){
            case UART:  break;                    //Communication method:  UART
            case IIC:                             //Communication method:  IIC
                        SIO = 0xff;               //Clock output for receiving
                        break;
            case CSI:                             //Communication method:  3-wire serial
            case PCSI:                            //Communication method:  pseudo 3-wire serial
                        CRXE = 0;                 //Receive disabled
                        CTXE = 0;                 //Transmit disabled
                        CRXE = 1;                 //Receive enabled
        }
        cRecieveFlag = RECIEVE_NOW;               //Sets receive flag while receiving
    }
    switch( cCommunicationMethod ){
        case UART:                                //Communication method:  UART
                   if( SRIF == 1 ){
                        cRecieveData = RXB;       //Reads receive data
                        SRIF = 0;
                        cRecieveFlag = RECIEVE_FINISH;
                   }
                   break;
```

```
        case IIC:                                       //Communication method:  IIC
                    if( CSIIF == 1 ){
                        cRecieveData = SIO;             //Reads receive data
                        CSIIF = 0;
                        cRecieveFlag = RECIEVE_FINISH;
                    }
                    break;
        case CSI:                                       //Communication method:  3-wire serial
        case PCSI:                                      //Communication method:  pseudo 3-wire serial
                    if( CSIIF == 1 ){
                        CRXE = 0;                       //Receive disabled
                        cRecieveData = SIO;             //Reads receive data
                        CSIIF = 0;
                        cRecieveFlag = RECIEVE_FINISH;
                    }
    }
}

/*****************************************************************
*                               Receive data                            *
*   Global variables:  cCommunicationMethod    Communication method     *
*                      cRecieveData            Receive data             *
*                      cRecieveFlag            Receive flag             *
*                      cTimerFlag              Timer flag               *
*                      cErrorStatus            Error status             *
*                      wWaitTimeDataData       Wait time between two sets of data  *
*                                                                       *
*   Local variable:    wRecieveCounter         Receive counter          *
*                                                                       *
*   IN:                wRecieveDataSize (receive data size) = 1 – 256 bytes  *
*****************************************************************/
void SDataRecieve( Word wRecieveDataSize ){
    register Word wRecieveCounter;

    if( cCommunicationMethod  == UART ){              //During UART reception
        for( wRecieveCounter = 0 ;
            wRecieveCounter < wRecieveDataSize ;
              wRecieveCounter++ ){
            cRecieveFlag = RECIEVE_START;             //Clears receive flag

            cTimerFlag = WAIT_START;                  //Sets timeout
            while( cRecieveFlag != RECIEVE_FINISH){
                                                      //Waits until one byte of data has been received
               SWaitMiliSec( 1000 );                  //Time out check
               if( cTimerFlag == WAIT_FINISH ){
                    cErrorStatus = STATUS_NO_RETURN;
                                                      //Reception failed
                    return;
            }
            if( SRIF == 1 ){
                cRecieveData = RXB;                   //Reads receive data
                SRIF = 0;
                  cRecieveFlag = RECIEVE_FINISH;
                }
            }
            aRecieveBuffer[ wRecieveCounter ] = cRecieveData;
                                                      //Stores receive data in buffer
        }
    }
    else{                                             //During 3-wire serial, IIC, or pseudo 3-wire serial
                                                      //communications
        switch(cCommunicationMethod){                 //Communication method:  only when IIC
           case IIC:
                     SSlaveAddressSend( 1 );          //Direction of slave address transmission  1:
                                                      //Receive
```

```
   }
   cTimerFlag = WAIT_START;                          //Sets timeout
   for( wRecieveCounter = 0 ;
       (( wRecieveCounter < wRecieveDataSize ) && ( cErrorStatus = = NO_ERROR )) ;
          wRecieveCounter++ ){

       cRecieveFlag = RECIEVE_START;                 //Clears receive flag
       while( cRecieveFlag != RECIEVE_FINISH){  //Waits until one byte of data has been received
          SWaitMiliSec( 1000 );                      //Time out check
          if( cTimerFlag == WAIT_FINISH ){
              cErrorStatus = STATUS_NO_RETURN;  //Reception failed
              break;
          }
          SByteDataRecieve();                        //Receives one byte of data
       }
       aRecieveBuffer[ wRecieveCounter ] = cRecieveData;
                                                     //Stores receive data in buffer
       if(( wRecieveCounter + 1 ) != wRecieveDataSize ){
                                                     //No wait during last transfer
          SWaitMicroSec( wWaitTimeDataData );   //Wait time between receiving two sets of data
      }
   }

   switch(cCommunicationMethod){                     //Communication method:  only when IIC
       case IIC:
                      SPT = 1;                       //Outputs stop condition
      }
   }
}

/*****************************************************************
*           Initialize 3-wire serial/pseudo 3-wire serial communications          *
*             Local variables:      wWork1           Work                          *
*                                   cWork2           Work                          *
*****************************************************************/
void SCsiIni( void ){
   register Word wWork1;                             //Work
   register Byte cWork2;                             //Work

   CSIM = 0x01;                                      //Send/receive disabled, MSB first, CLK = TM 3/2

   wWork1 = 12500;                                   //CR30W setting when 12,500:  fx = 20 MHz,
                                                     //sck = 100 Hz
   wWork1 /= wParCsiClockSpeed;                      //wSioClockSpeed:  100-Hz units
   wWork1--;
   CR30W = wWork1;                                   //CR30W = (12,500 / wSioClockSpeed) – 1

   cWork2 = PMC3;
   cWork2 &= 0xf3;                                   //PMC32:  SCK, PMC33:  SO0
   cWork2 |= 0x0c;
   PMC3 = cWork2;                                    //Control mode

   cWork2 = PRM0;                                    //bit[7-4]:  TM3's count clock specification
   cWork2 &= 0x0f;                                   //bit[3-0]:  TM0's count clock specification
   cWork2 |= 0x10;
   PRM0 = cWork2;                                    //TM3 = fxx/8, TM0 = fxx/2,048

   cWork2 = TMC0;                                    //TM3 16-bit operation mode, start of TM3 operation
   cWork2 &= 0x0f;
   cWork2 |= 0x90;
   TMC0 = cWork2;

   CSIIF = 0;                                        //Clears send/receive finish flag
}
```

```
/******************************************************************
*                    Initialize IIC communications                  *
*                    Local variable:   cWork          Work          *
******************************************************************/
void SIicIni( void ){
   register Byte cWork;
   CSIM = 0b00001110;                        //Transmit/receive disabled, internal clock, master,
                                             //SPRS
   cWork = PMC3;
   cWork &= 0xf3;                            //PMC32: SCK, PMC33: SO0
   cWork |= 0x0c;
   PMC3 = cWork;                             //Control mode

   IICC = 0x90;                              //Waits for nine clocks

   SPRM = 0x08;                              //duty: For standard mode
                                             //hold: 16 MHz < fxx < 32 MHz
                                             //clk: fxx/256 = 20 MHz / 256 = 78.125 kHz
   CTXE = 1;                                 //Transmit/receive enabled

   CSIIF = 0;                                //Clears send/receive finish flag
}

/******************************************************************
*                    Initialize UART communications                 *
*              Local variable:     cWork           Work             *
******************************************************************/
void SUartIni( void ){
   register Byte cWork;

   TXE = 0;                                  //Transmit disabled
   RXE = 0;                                  //Receive disabled

   cWork = PMC3;
   cWork &= 0xfc;                            //PMC30: RXD, PMC31: TXD
   cWork |= 0x03;
   PMC3 = cWork;                             //Control mode

   BRGC = BRGC9600;                          //9,600 bps when fx = 20 MHz

   ASIM = 0b11001011;                        //Internal clock is selected. Receive completion
                                             //interrupt inhibited when receive error has
                                             //occurred.
                                             //Stop bit: 1 bit, Characters: 8 bits
                                             //No parity, transmit/receive enabled

   STIF = 0;                                 //Clears send finish flag
   SRIF = 0;                                 //Clears receive finish flag
}
```

4.6.2 RAM definitions

```
#include "DATTYPE.H"                            //Data type definition file
#include "sram.h"                               //RAM external access definition file
#include "constant.h"                           //Constant value definition file

/*-----------------------------------------------------------/
                        RAM definition

   This is a variable that is used in the sample programs.
   The following declarations are required when using the sample programs.
-----------------------------------------------------------*/

Byte aSendBuffer[256];                          //Send buffer
Byte aRecieveBuffer[256];                       //Receive buffer

/***** Variables used as parameters *****/
sreg DWord dwParStartAddress;                   //Write start address
sreg DWord dwParEndAddress;                     //Write end address
sreg Word wParCpuClockSpeed;                    //Flash microcontroller CPU's clock speed, 10-kHz
                                                //units
sreg Word wParCsiClockSpeed;                    //Communication clock speed (100-Hz units) for 3-
                                                //wire serial or pseudo 3-wire serial communications
sreg Word wParEraseTime;                        //Erase time (10-ms units)

sreg Byte cParTargetSeries;                     //Selects 78K/0 or 78K/0S as target series
sreg Byte cParVppPulse;                         //VPP pulse count (valid range:  0 to 14)
sreg Byte cParBaudRate;                         //Communication baud rate
sreg Byte cParCpuClockSource;                   //Selects CPU clock source supplied to flash
                                                //microcontroller
sreg Byte cParSlaveAddress;                     //Slave address

/***** Other variables *****/
sreg Word wSendSize;                            //Buffer send size
sreg Byte cCommunicationMethod;                 //Communication method
sreg Byte cSendData;                            //Send data
sreg Byte cRecieveData;                         //Receive data
sreg Byte cSendFlag;                            //Send flag
sreg Byte cRecieveFlag;                         //Receive flag

sreg Word wWaitTimeVppCom;                      //Wait time (μs units) between VPP and COMMAND
sreg Word wWaitTimeComAck;                      //Wait time (μs units) between COMMAND and ACK
sreg Word wWaitTimeAckCom;                      //Wait time (μs units) between ACK and COMMAND
sreg Word wWaitTimeAckData;                     //Wait time (μs units) between ACK and DATA
sreg Word wWaitTimeDataData;                    //Wait time (μs units) between two sets of DATA
sreg Word wWaitTimeDataAck;                     //Wait time (μs units) between DATA and ACK

sreg Byte cTargetStatus;                        //Status of target microcontroller
sreg Byte cRetryCounter;                        //Retry counter
sreg Byte cErrorStatus;                         //Error status
sreg Byte cEnterCommand;                        //Enter command

sreg Byte cTimerFlag;                           //Timer flag

reg Byte cWaitClockSelect;                      //Element number of structure array (wait data table)

sreg struct SigType sSig;                       //Stores silicon signature
```

4.6.3 RAM declarations

```
#include "DATTYPE.H"                          //Data type definition file

/*-------------------------------------------------------------------/
                           RAM declaration

   These are variables that are used in the sample programs.
   The following types of declarations must be included when using the sample programs.
---------------------------------------------------------------------*/

/* This defines the data type of the area used to store the silicon signature data. */
struct SigType{                               //Type declaration for structure used to store silicon
                                              //signature data
   Byte cSigVendorCode;                       //Vendor Code
   Byte cSigIdCode;                           //ID Code
   Byte cSigElectInf;                         //Electrical Information
   DWord dwSigLastAddress;                    //Last Address
   Byte aSigDeviceName[10];                   //Device Name
   Byte cSigBlockInf;                         //Block Information//PARAMETER
};

/* This defines the data type of the wait clock count data table used to store the wait clock count. */
struct WaitData{                              //Type declaration of structure used to store wait
                                              //clock count (as ROM table)
   Word wWaitVppCom ;                         //0     Wait time between VPP and COMMAND
   Word wWaitComAck ;                         //1     Wait time between COMMAND and ACK
   Word wWaitAckCom ;                         //2     Wait time between ACK and COMMAND
   Word wWaitAckData;                         //3     Wait time between ACK and DATA
   Word wWaitDataData;                        //4     Wait time between two sets of DATA
   Word wWaitDataAck;                         //5     Wait time between DATA and ACK
   Word wWaitFrequencySet;                    //6     Wait time for calculation of oscillation
                                              //      frequency
   Word wWaitEraseTimeSet;                    //7     Wait time for calculation of erase time
   Word wWaitComToCom;                        //8     Wait time between two COMMANDS
   Word wWaitBaudRateCalc;                    //9     Wait time for calculation of baud rate
   Word wWaitPreWrite;                        //10    Wait time for prewrite
   Word wWaitErase;                           //11    Wait time for erase
   Word wWaitWrite;                           //12    Wait time for write
   Word wWaitInternalVerify;                  //13    Wait time for internal verify
   Word wWaitBlankChek;                       //14    Wait time for blank check
   Word wWaitRst1;                            //15    Wait time after sending first RESET
                                              //      command (for UART synchronization
                                              //      detection)
   Word wWaitRst2;                            //16    Wait time after sending second RESET
                                              //      command (for UART synchronization
                                              //      detection)
   Word wWaitRst3;                            //17    Wait time after sending third RESET
                                              //      command (for UART synchronization
                                              //      detection)
   Dword dwWaitVerify;                        //18    Wait time for verify
};

extern Byte aSendBuffer[256];                 //Send buffer
extern Byte aRecieveBuffer[256];              //Receive buffer
```

```
/***** Variables used as parameters *****/
extern sreg DWord dwParStartAddress;          //Write start address
extern sreg DWord dwParEndAddress;            //Write end address
extern sreg Word wParCpuClockSpeed;           //Flash microcontroller CPU's clock speed, 10-kHz
                                              //units
extern sreg Word wParCsiClockSpeed;           //Communication clock speed (100-Hz units) for 3-
                                              //wire serial or pseudo 3-wire serial
                                              //communications
extern sreg WordwParEraseTime;                //Erase time (10-ms units)
extern sreg Byte cParTargetSeries;            //Selects 78K/0 or 78K/0S as target series
extern sreg Byte cParVppPulse;                //VPP pulse count (valid range: 0 to 14)
extern sreg Byte cParBaudRate;                //Communication baud rate
extern sreg Byte cParCpuClockSource;          //Selects CPU clock source supplied to flash
                                              //microcontroller
extern sreg Byte cParSlaveAddress;            //Slave address

/***** Other variables *****/
extern sreg Word wSendSize;                   //Buffer send size
extern sreg Byte cCommunicationMethod;        //Communication method
extern sreg Byte cSendData;                   //Send data
extern sreg Byte cRecieveData;                //Receive data
extern sreg Byte cSendFlag;                   //Send flag
extern sreg Byte cRecieveFlag;                //Receive flag
extern sreg Word wWaitTimeVppCom;             //Wait time (μs units) between VPP and COMMAND
extern sreg Word wWaitTimeComAck;             //Wait time (μs units) between COMMAND and ACK
extern sreg Word wWaitTimeAckCom;             //Wait time (μs units) between ACK and COMMAND
extern sreg Word wWaitTimeAckData;            //Wait time (μs units) between ACK and DATA
extern sreg Word wWaitTimeDataData;           //Wait time (μs units) between two sets of DATA
extern sreg Word wWaitTimeDataAck;            //Wait time (μs units) between DATA and ACK
extern sreg Byte cTargetStatus;               //Status of target microcontroller
extern sreg Byte cRetryCounter;               //Retry counter
extern sreg Byte cErrorStatus;                //Error status
extern sreg Byte cEnterCommand;               //Enter command
extern sreg Byte cTimerFlag;                  //Timer flag
extern sreg Byte cWaitClockSelect;            //Element number of structure array (wait data table)
extern sreg struct SigType sSig;              //Stores silicon signature
extern sreg Byte cWaitClockSelect;            //Element number of structure array (wait data table)
extern const struct WaitData WaitDataTable[]; //ROM table wait clock count for each
                                              //communication method
```

4.6.4 Wait clock count data table definition

```
#pragma sfr                                        //Uses sfr area

#include "DATTYPE.H"                               //Data type definition file
#include "sram.h"                                  //RAM external access definition file
#include "constant.h"                              //Constant value definition file

/*---------------------------------------------------------------------------/
                          Wait clock count data table definition

  This defines the wait cock count data that is used in the sample programs.
  When using the sample programs, the following data must be defined and a wait data table area
  must be reserved.

  The wait clock counts are the number of clocks during the target microcontroller's processing.

  The wait clock counts for various commands, which differ according to the target series (78K/0 or
  78K/0S) and communication method, are stored to ROM as table data.

  When executing these commands, waits are performed using the required wait clock count
  based on the data stored in ROM.
---------------------------------------------------------------------------*/

/*---------------------------------------------
*       Wait clock count constant value definition
---------------------------------------------*/
#define DUMMY          1000                        //dummy (value is insignificant)

/***** Define wait clock count for 78K/0 series and 3-wire serial communications *****/
#define WAIT_K0_CSI_VPPCOM      150                //Wait clock count between VPP and COMMAND
#define WAIT_K0_CSI_COMACK      900                //Wait clock count between COMMAND and ACK
#define WAIT_K0_CSI_ACKCOM      170                //Wait clock count between ACK and COMMAND
#define WAIT_K0_CSI_ACKDAT      230                //Wait clock count between ACK and DATA
#define WAIT_K0_CSI_DATDAT      300                //Wait clock count between two sets of DATA
#define WAIT_K0_CSI_DATACK      350                //Wait clock count between DATA and ACK

#define WAIT_K0_CSI_FRQ         2200               //Wait clock count for calculation of oscillation
                                                   //frequency
#define WAIT_K0_CSI_ERT         1200               //Wait clock count for processing of erase time setting

/***** Define wait clock count for 78K/0 series and IIC communications *****/
#define WAIT_K0_IIC_VPPCOM      30                 //Wait clock count between VPP and COMMAND
#define WAIT_K0_IIC_COMACK      1030               //Wait clock count between COMMAND and ACK
#define WAIT_K0_IIC_ACKCOM      40                 //Wait clock count between ACK and COMMAND
#define WAIT_K0_IIC_ACKDAT      50                 //Wait clock count between ACK and DATA
#define WAIT_K0_IIC_DATDAT      70                 //Wait clock count between two sets of DATA
#define WAIT_K0_IIC_DATACK      70                 //Wait clock count between DATA and ACK

#define WAIT_K0_IIC_FRQ         2350               //Wait clock count for calculation of oscillation
                                                   //frequency
#define WAIT_K0_IIC_ERT         1200               //Wait clock count for processing of erase time setting

/***** Define wait clock count for 78K/0 series and UART communications *****/
#define WAIT_K0_UART_VPPCOM     290                //Wait clock count between VPP and COMMAND
#define WAIT_K0_UART_COMACK     1870               //Wait clock count between COMMAND and ACK
#define WAIT_K0_UART_ACKCOM     170                //Wait clock count between ACK and COMMAND
#define WAIT_K0_UART_ACKDAT     240                //Wait clock count between ACK and DATA
#define WAIT_K0_UART_DATDAT     650                //Wait clock count between two sets of DATA
#define WAIT_K0_UART_DATACK     700                //Wait clock count between DATA and ACK
```

```
#define WAIT_K0_UART_FRQ        5260        //Wait clock count for calculation of oscillation
                                            //frequency
#define WAIT_K0_UART_ERT        1450        //Wait clock count for processing of erase time setting

#define WAIT_K0_UART_BRGCALC    3820        //Wait clock count for wait during calculation of baud
                                            //rate setting

#define WAIT_K0_UART_RST1       260         //Wait clock count after sending first RESET
                                            //command
#define WAIT_K0_UART_RST2       180         //Wait clock count after sending second RESET
                                            //command
#define WAIT_K0_UART_RST3       4100        //Wait clock count after sending third RESET
                                            //command

/***** Define wait clock count for 78K/0 series and 3-wire serial, IIC, or UART communications *****/

#define WAIT_COMCOM             1000        //Wait clocks between two COMMANDS
#define WAIT_K0_PREW            230         //Wait clock count for per-byte prewrite processing
#define WAIT_K0_ERA             690         //Wait clock count for per-byte erase processing
#define WAIT_K0_PRG             1010        //Wait clock count for per-byte write processing
#define WAIT_K0_IVRF            840         //Wait clock count for per-byte internal verify
                                            //processing
#define WAIT_K0_BLN             690         //Wait clock count for per-byte blank check
                                            //processing
#define WAIT_K0_VRF             258560      //Wait clock count for verify processing per 256 bytes

/***** Define wait clock count for 78K/0 Series and pseudo 3-wire serial communications *****/
#define WAIT_K0_PCSI_VPPCOM     190         //Wait clock count between VPP and COMMAND
#define WAIT_K0_PCSI_COMACK     1630        //Wait clock count between COMMAND and ACK
#define WAIT_K0_PCSI_ACKCOM     790         //Wait clock count between ACK and COMMAND
#define WAIT_K0_PCSI_ACKDAT     640         //Wait clock count between ACK and DATA
#define WAIT_K0_PCSI_DATDAT     860         //Wait clock count between two sets of DATA
#define WAIT_K0_PCSI_DATACK     960         //Wait clock count between DATA and ACK

#define WAIT_K0_PCSI_FRQ        3380        //Wait clock count for calculation of oscillation
                                            //frequency
#define WAIT_K0_PCSI_ERT        1690        //Wait clock count for processing of erase time setting

#define WAIT_K0_PCSI_PREW       330         //Wait clock count for per-byte prewrite processing
#define WAIT_K0_PCSI_ERA        840         //Wait clock count for per-byte erase processing
#define WAIT_K0_PCSI_PRG        1010        //Wait clock count for per-byte write processing
#define WAIT_K0_PCSI_IVRF       900         //Wait clock count for per-byte internal verify
                                            //processing
#define WAIT_K0_PCSI_BLN        840         //Wait clock count for per-byte blank check
                                            //processing

/***** Define wait clock count for 78K/0S Series and pseudo 3-wire serial communications *****/
#define WAIT_K0S_CSI_VPPCOM     220         //Wait clock count between VPP and COMMAND
#define WAIT_K0S_CSI_COMACK     1040        //Wait clock count between COMMAND and ACK
#define WAIT_K0S_CSI_ACKCOM     210         //Wait clock count between ACK and COMMAND
#define WAIT_K0S_CSI_ACKDAT     190         //Wait clock count between ACK and DATA
#define WAIT_K0S_CSI_DATDAT     360         //Wait clock count between two sets of DATA
#define WAIT_K0S_CSI_DATACK     320         //Wait clock count between DATA and ACK

#define WAIT_K0S_CSI_FRQ        31600       //Wait clock count for calculation of oscillation
                                            //frequency
#define WAIT_K0S_ERT            20000       //Wait clock count for processing of erase time setting
```

```
#define WAIT_K0S_PREW          216       //Wait clock count for per-byte prewrite processing
#define WAIT_K0S_ERA           175       //Wait clock count for per-byte erase processing
#define WAIT_K0S_PRG           275       //Wait clock count for per-byte write processing
#define WAIT_K0S_IVRF          230       //Wait clock count for per-byte internal verify
                                         //processing
#define WAIT_K0S_BLN           175       //Wait clock count for per-byte blank check
                                         //processing

#define WAIT_K0S_VRF           29400     //Wait clock count for verify processing per 128 bytes

/***** Define wait clock count for 78K/0S Series and IIC communications *****/
#define WAIT_K0S_IIC_VPPCOM    220       //Wait clock count between VPP and COMMAND
#define WAIT_K0S_IIC_COMACK    1240      //Wait clock count between COMMAND and ACK
#define WAIT_K0S_IIC_ACKCOM    390       //Wait clock count between ACK and COMMAND
#define WAIT_K0S_IIC_ACKDAT    640       //Wait clock count between ACK and DATA
#define WAIT_K0S_IIC_DATDAT    530       //Wait clock count between two sets of DATA
#define WAIT_K0S_IIC_DATACK    530       //Wait clock count between DATA and ACK

#define WAIT_K0S_IIC_FRQ       65000     //Wait clock count for calculation of oscillation
                                         //frequency

/***** Define wait clock count for 78K/0S Series and UART communications *****/
#define WAIT_K0S_UART_VPPCOM   330       //Wait clock count between VPP and COMMAND
#define WAIT_K0S_UART_COMACK   1900      //Wait clock count between COMMAND and ACK
#define WAIT_K0S_UART_ACKCOM   190       //Wait clock count between ACK and COMMAND
#define WAIT_K0S_UART_ACKDAT   180       //Wait clock count between ACK and DATA
#define WAIT_K0S_UART_DATDAT   690       //Wait clock count between two sets of DATA
#define WAIT_K0S_UART_DATACK   660       //Wait clock count between DATA and ACK

#define WAIT_K0S_UART_FRQ      46600     //Wait clock count for calculation of oscillation
                                         //frequency
#define WAIT_K0S_UART_BRGCALC  27000     //Wait clock count for wait during calculation of
                                         //baud rate setting

#define WAIT_K0S_UART_RST1     320       //Wait clock count after sending first RESET
                                         //command
#define WAIT_K0S_UART_RST2     230       //Wait clock count after sending second RESET
                                         //command
#define WAIT_K0S_UART_RST3     14700     //Wait clock count after sending third RESET
                                         //command

/***** Define wait clock count for 78K/0S Series and pseudo 3-wire serial communications *****/
#define WAIT_K0S_PCSI_VPPCOM   210       //Wait clock count between VPP and COMMAND
#define WAIT_K0S_PCSI_COMACK   2580      //Wait clock count between COMMAND and ACK
#define WAIT_K0S_PCSI_ACKCOM   820       //Wait clock count between ACK and COMMAND
#define WAIT_K0S_PCSI_ACKDAT   700       //Wait clock count between ACK and DATA
#define WAIT_K0S_PCSI_DATDAT   1560      //Wait clock count between two sets of DATA
#define WAIT_K0S_PCSI_DATACK   1600      //Wait clock count between DATA and ACK

#define WAIT_K0S_PCSI_FRQ      44200     //Wait clock count for calculation of oscillation
                                         //frequency
#define WAIT_K0S_PCSI_ERT      27600     //Wait clock count for processing of erase time setting

#define WAIT_K0S_PCSI_PREW     340       //Wait clock count for per-byte prewrite processing
#define WAIT_K0S_PCSI_ERA      235       //Wait clock count for per-byte erase processing
#define WAIT_K0S_PCSI_PRG      440       //Wait clock count for per-byte write processing
#define WAIT_K0S_PCSI_BLN      235       //Wait clock count for per-byte blank check
                                         //processing
#define WAIT_K0S_PCSI_IVRF     325       //Wait clock count for per-byte internal verify
                                         //processing
```

```
#define WAIT_K0S_PCSI_VRF       41800            //Wait clock count for verify processing per 128 bytes

/*-------------------------------------------------------------------------------
                              Wait clock count data table
   The wait clock count data, which is used to adjust communication timing and in commands that differ
   according to the target series and communication method, is stored in a ROM table.  When executing
   these commands, wait processing is performed using the data stored in the ROM table.  The data stored
   in the ROM table is the data that is defined above.
---------------------------------------------------------------------------------*/
/**********************************************
*                 Wait data table
**********************************************/

const struct WaitData WaitDataTable[] = {
   /***** 78K/0, 3-wire serial *****/
   {   WAIT_K0_CSI_VPPCOM ,WAIT_K0_CSI_COMACK ,WAIT_K0_CSI_ACKCOM ,
       WAIT_K0_CSI_ACKDAT ,WAIT_K0_CSI_DATDAT ,WAIT_K0_CSI_DATACK ,
       WAIT_K0_CSI_FRQ ,WAIT_K0_CSI_ERT ,WAIT_COMCOM ,
       DUMMY ,WAIT_K0_PREW ,WAIT_K0_ERA ,
       WAIT_K0_PRG ,WAIT_K0_IVRF ,WAIT_K0_BLN ,
       DUMMY ,DUMMY ,DUMMY ,
       WAIT_K0_VRF },

   /***** 78K/0, IIC *****/
   {   WAIT_K0_IIC_VPPCOM,WAIT_K0_IIC_COMACK ,WAIT_K0_IIC_ACKCOM ,
       WAIT_K0_IIC_ACKDAT ,WAIT_K0_IIC_DATDAT,WAIT_K0_IIC_DATACK ,
       WAIT_K0_IIC_FRQ ,WAIT_K0_IIC_ERT , WAIT_COMCOM,
       DUMMY ,WAIT_K0_PREW ,WAIT_K0_ERA ,
       WAIT_K0_PRG ,WAIT_K0_IVRF , WAIT_K0_BLN ,
       DUMMY ,DUMMY ,DUMMY ,
       WAIT_K0_VRF },

   /***** 78K/0, UART *****/
   {   WAIT_K0_UART_VPPCOM ,WAIT_K0_UART_COMACK ,WAIT_K0_UART_ACKCOM ,
       WAIT_K0_UART_ACKDAT ,WAIT_K0_UART_DATDAT,WAIT_K0_UART_DATACK ,
       WAIT_K0_UART_FRQ ,WAIT_K0_UART_ERT  ,WAIT_COMCOM ,
       WAIT_K0_UART_BRGCALC ,WAIT_K0_PREW ,WAIT_K0_ERA ,
       WAIT_K0_PRG ,WAIT_K0_IVRF ,WAIT_K0_BLN ,
       WAIT_K0_UART_RST1 ,WAIT_K0_UART_RST2 ,WAIT_K0_UART_RST3 ,
       WAIT_K0_VRF },

   /***** 78K/0, pseudo 3-wire serial *****/
   {   WAIT_K0_PCSI_VPPCOM ,WAIT_K0_PCSI_COMACK ,WAIT_K0_PCSI_ACKCOM ,
       WAIT_K0_PCSI_ACKDAT ,WAIT_K0_PCSI_DATDAT,WAIT_K0_PCSI_DATACK ,
       WAIT_K0_PCSI_FRQ ,WAIT_K0_PCSI_ERT ,WAIT_COMCOM ,
       DUMMY ,WAIT_K0_PCSI_PREW ,WAIT_K0_PCSI_ERA ,
       WAIT_K0_PCSI_PRG ,WAIT_K0_PCSI_IVRF ,WAIT_K0_PCSI_BLN ,
       DUMMY ,DUMMY ,DUMMY ,
       WAIT_K0_VRF },

   /***** 78K/0S, 3-wire serial *****/
   {   WAIT_K0S_CSI_VPPCOM ,WAIT_K0S_CSI_COMACK ,WAIT_K0S_CSI_ACKCOM ,
       WAIT_K0S_CSI_ACKDAT ,WAIT_K0S_CSI_DATDAT ,WAIT_K0S_CSI_DATACK ,
       WAIT_K0S_CSI_FRQ ,WAIT_K0S_ERT ,WAIT_COMCOM ,
       DUMMY ,WAIT_K0S_PREW ,WAIT_K0S_ERA ,
       WAIT_K0S_PRG ,WAIT_K0S_IVRF ,WAIT_K0S_BLN ,
       DUMMY ,DUMMY ,DUMMY ,
       WAIT_K0S_VRF },
```

```
    /***** 78K/0S, IIC *****/
    {   WAIT_K0S_IIC_VPPCOM ,WAIT_K0S_IIC_COMACK ,WAIT_K0S_IIC_ACKCOM ,
        WAIT_K0S_IIC_ACKDAT ,WAIT_K0S_IIC_DATDAT ,WAIT_K0S_IIC_DATACK ,
        WAIT_K0S_IIC_FRQ ,WAIT_K0S_ERT ,WAIT_COMCOM ,
        DUMMY ,WAIT_K0S_PREW ,WAIT_K0S_ERA ,
        WAIT_K0S_PRG ,WAIT_K0S_IVRF ,WAIT_K0S_BLN ,
        DUMMY ,DUMMY ,DUMMY ,
        WAIT_K0S_VRF },

    /***** 78K/0S, UART *****/
    {   WAIT_K0S_UART_VPPCOM ,WAIT_K0S_UART_COMACK ,WAIT_K0S_UART_ACKCOM ,
        WAIT_K0S_UART_ACKDAT ,WAIT_K0S_UART_DATDAT ,WAIT_K0S_UART_DATACK ,
        WAIT_K0S_UART_FRQ ,WAIT_K0S_ERT ,WAIT_COMCOM ,
        WAIT_K0S_UART_BRGCALC ,WAIT_K0S_PREW ,WAIT_K0S_ERA ,
        WAIT_K0S_PRG ,WAIT_K0S_IVRF ,WAIT_K0S_BLN ,
        WAIT_K0S_UART_RST1 ,WAIT_K0S_UART_RST2 ,WAIT_K0S_UART_RST3 ,
        WAIT_K0S_VRF },

    /***** 78K/0S, pseudo 3-wire serial *****/
    {   WAIT_K0S_PCSI_VPPCOM ,WAIT_K0S_PCSI_COMACK ,WAIT_K0S_PCSI_ACKCOM ,
        WAIT_K0S_PCSI_ACKDAT ,WAIT_K0S_PCSI_DATDAT ,WAIT_K0S_PCSI_DATACK ,
        WAIT_K0S_PCSI_FRQ ,WAIT_K0S_PCSI_ERT ,WAIT_COMCOM ,
        DUMMY ,WAIT_K0S_PCSI_PREW ,WAIT_K0S_PCSI_ERA ,
        WAIT_K0S_PCSI_PRG ,WAIT_K0S_PCSI_IVRF ,WAIT_K0S_PCSI_BLN ,
        DUMMY ,DUMMY ,DUMMY ,
        WAIT_K0S_PCSI_VRF }
};
```

4.6.5 List of constant value definitions

```
/*----------------------------------------------------------------------/
                                Constant value definition

   These constant values are used in the sample programs
   The following types of declarations must be included when using the sample programs.
-----------------------------------------------------------------------*/

/*----------------------------------------------------------------------------------
                                Address where user data is stored

   In these sample programs, the data stored in external ROM is written to the flash microcontroller.
   The start address of external ROM is 20000H (start address of external memory area in 78K/4 products).
   The following are declared and included in the sample programs.
----------------------------------------------------------------------------------*/
#define USER_DATA_ADDRESS    0x020000        //Address where write and verify data is stored:  64
                                             //Kbytes starting from 20000H

/*---------------------------------------------------------
     Flash memory control command (variable to be set: cSendData)
---------------------------------------------------------*/
#define    CMD_RESET                 0x00     //Reset command
#define    CMD_CHIP_VRF              0x11     //Batch verify command
#define    CMD_CHIP_IVRF             0x18     //Batch internal verify command
#define    CMD_CHIP_ERASE            0x20     //Batch erase command
#define    CMD_CHIP_BLN              0x30     //Batch blank check command
#define    CMD_HIGH_SPEED_WRITE      0x40     //High-speed write command
#define    CMD_CONTINUE_WRITE        0x44     //Continuous write command
#define    CMD_PRE_WRITE             0x48     //Prewrite command
#define    CMD_STATUS                0x70     //Status command
#define    CMD_FRQ_SET               0x90     //Oscillation frequency setting command
#define    CMD_ERT_SET               0x95     //Erase time setting command
#define    CMD_BAUDRATE              0x9a     //Baud rate setting command
#define    CMD_SIGNATURE             0xC0     //Silicon signature read command

/*---------------------------------------------------
          Error status (variable to be set: cErrorStatus)
---------------------------------------------------*/
#define    NO_ERROR                  0x00     //No error
#define    BLANK_CHEK_FAILED         0x01     //Blank check error
#define    VERIFY_ERROR              0x02     //Verify error
#define    PROGRAM_FAILED            0x04     //Write failed
#define    ERASE_FAILED              0x08     //Erase failed
#define    INITIALISE_ERROR          0x09     //Synchronization detection failed
#define    TARGET_RETURN_ERROR       0x0a     //ACK not returned
#define    IIC_NO_ACK                0x0b     //ACK not detected during IIC communications
#define    STATUS_NO_RETURN          0x0c     //Reception failed
#define    PARAMETER_OUT_OF_RANGE    0X0d     //Parameter is out of range
#define    TARGET_IS_CLOSED          0x0e     //Send failed
#define    SYSTEM_ERROR              0x0f     //Unexpected error

/*---------------------------------------------------
        VPP pulse count (variable to be set: cParVppPulse)
---------------------------------------------------*/
#define    SIO_CH0                   0x00     //3-wire serial – ch0
#define    SIO_CH1                   0x01     //3-wire serial – ch1
#define    SIO_CH2                   0x02     //3-wire serial – ch2
#define    SIO_CH3                   0x03     //3-wire serial – ch3
#define    IIC_CH0                   0x04     //IIC – ch0
#define    IIC_CH1                   0x05     //IIC – ch1
```

```
#define    IIC_CH2                    0x06      //IIC – ch2
#define    IIC_CH3                    0x07      //IIC – ch3
#define    UART_CH0                   0x08      //UART – ch0
#define    UART_CH1                   0x09      //UART – ch1
#define    UART_CH2                   0x0a      //UART – ch2
#define    UART_CH3                   0x0b      //UART – ch3
#define    PSIO_A                     0x0c      //Pseudo 3-wire serial – A
#define    PSIO_B                     0x0d      //Pseudo 3-wire serial – B
#define    PSIO_C                     0x0e      //Pseudo 3-wire serial – C

/*--------------------------------------------------------
   Communication method (variable to be set:  cCommunicationMethod)
----------------------------------------------------------*/
#define    CSI                        0x00      //3-wire serial
#define    IIC                        0x01      //IIC
#define    UART                       0x02      //UART
#define    PCSI                       0x03      //Pseudo 3-wire serial
/*----------------------------------------------------
          Receive data (variable to be set: cReceiveData)
------------------------------------------------------*/
#define    ACK                        0x3c      //Acknowledge

/*----------------------------------------------------
          Status data (variable to be set: cTargetStatus)
------------------------------------------------------*/
#define    ERASING_NOW                0x80      //Erase in progress
#define    PROGRAMING_NOW             0x40      //Write in progress
#define    PRE_WRITING_NOW            0x40      //Prewrite in progress
#define    VERIFYING_NOW              0x20      //Verify in progress
#define    BLANK_CHEK_NOW             0x10      //Blank check in progress
//#define  ERASE_FAILED               0x08      //Erase failed         //shared with cErrorStatus
//#define  PROGRAM_FAILED             0x04      //Write failed         //shared with cErrorStatus
//#define  VERIFY_ERROR               0x02      //Verify error         //shared with cErrorStatus
//#define  BLANK_CHEK_FAILED          0x01      //Blank check error    //shared with cErrorStatus
#define    READY                      0x00      //Command processing completed or no error

/*----------------------------------------------------
   Communication baud rate (variable to be set: cParBaudRate)
------------------------------------------------------*/
#define    BPS4800                    0x02      //4,800 bps
#define    BPS9600                    0x03      //9,600 bps
#define    BPS19200                   0x04      //19,200 bps
#define    BPS31250                   0x05      //31,250 bps
#define    BPS38400                   0x06      //38,400 bps
#define    BPS76800                   0x07      //76,800 bps

/*---------------------------------------------------------
   Baud rate generator setting value (variable to be set: BRGC register)
-----------------------------------------------------------*/
#define    BRGC4800                   0x50      //4,800 bps
#define    BRGC9600                   0x40      //9,600 bps
#define    BRGC19200                  0x30      //19,200 bps
#define    BRGC31250                  0x24      //31,250 bps
#define    BRGC38400                  0x20      //38,400 bps
#define    BRGC76800                  0x10      //76,800 bps

/*---------------------------------------------------------
          Target series (variable to be set: cParTargetSeries)
-----------------------------------------------------------*/
#define    K0                         0x00      //Target series:  78K/0 Series
#define    K0S                        0x01      //Target series:  78K/0S Series
```

```
/*-------------------------------------------------------
        Select CPU clock source (variable to be set: cCpuClockSource)
---------------------------------------------------------*/
#define    IN_FLASHWRITER            0x00      //Supplied from flash programmer
#define    ON_TARGETBOARD            0x01      //Uses target board's clock

/*-------------------------------------------------------
            Enter command (variable to be set: cEnterCommand)
---------------------------------------------------------*/
#define    ENTER_EPV                 0x10      //Erase/write/verify command
#define    ENTER_ERA                 0x08      //Erase command
#define    ENTER_PRG                 0x04      //Write command
#define    ENTER_VRF                 0x02      //Verify command
#define    ENTER_BLN                 0x01      //Blank check command
#define    ENTER_NOTHING             0x00      //No entered command

/*-------------------------------------------------------
            Timer flag setting value (variable to be set: cTimerFlag)
---------------------------------------------------------*/
#define    WAIT_START                0x00      //Start of wait
#define    WAIT_FINISH               0x00      //End of wait
#define    WAIT_NOW                  0x01      //Wait in progress

/*-------------------------------------------------------
                 Send flag (variable to be set: cSendFlag)
---------------------------------------------------------*/
#define    SEND_START                0x00      //Start of send
#define    SEND_NOW                  0x01      //Send in progress
#define    SEND_FINISH               0xff      //End of send

/*-------------------------------------------------------
               Receive flag (variable to be set: cReceiveFlag)
---------------------------------------------------------*/
#define    RECIEVE_START             0x00      //Start of receive
#define    RECIEVE_NOW               0x01      //Receive in progress
#define    RECIEVE_FINISH            0xff      //End of receive
```

4.7 Error Code List

When an error occurs in any of these sample programs, a value such as those listed below entered as cErrorStatus (variable name).

cErrorStatus	Error
00H	No error
01H	Blank check error
02H	Verify error
04H	Write failed
08H	Erase failed
09H	Synchronization detection failed
0AH	ACK not returned
0BH	ACK not detected during IIC communications
0CH	Receive failed
0DH	Parameter is out of range
0EH	Send failed
0FH	Unexpected error

[MEMO]

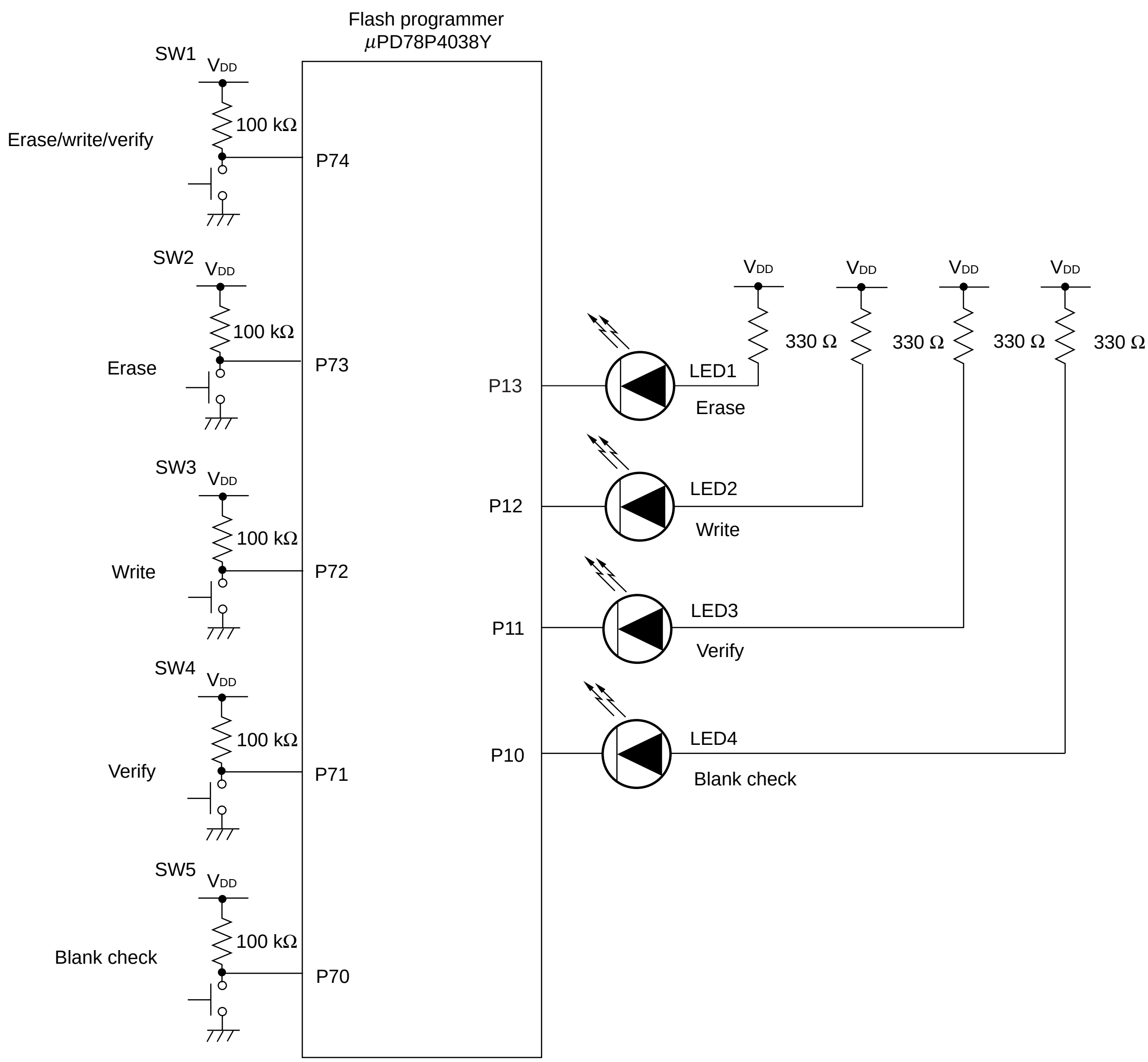
CHAPTER 5 SAMPLE INTERFACE

This chapter presents a sample program that uses keys (command entry) and LED (error indicators) in a flash programmer interface.

5.1 Connection Diagram

Figure 5-1 shows a connection diagram that includes the flash programmer and the interface keys and LEDs.

Figure 5-1. Connection Diagram



In this sample interface, commands entered via SW1 to SW5 are executed and the LEDs corresponding to the executed command is lit. If an error occurs during command execution, the corresponding LED blinks at a 0.5-second interval to notify the user of the error. Table 5-1 lists the correspondences among SW1 to SW5, LED1 to LED4, and the commands. Table 5-2 lists the types of errors corresponding to blinking LEDs.

Table 5-1. Correspondence among SWs, LEDs, and Commands

SW	Command Being Executed	Lit LED
SW1 is ON	Erase/write/verify	**Note**
SW2 is ON	Erase	LED1
SW3 is ON	Write	LED2
SW4 is ON	Verify	LED3
SW4 is ON	Blank check	LED4

Note The sequence of lit LEDs is LED1 → LED2 → LED3 corresponding to the sequence of command execution (erase → write → verify).

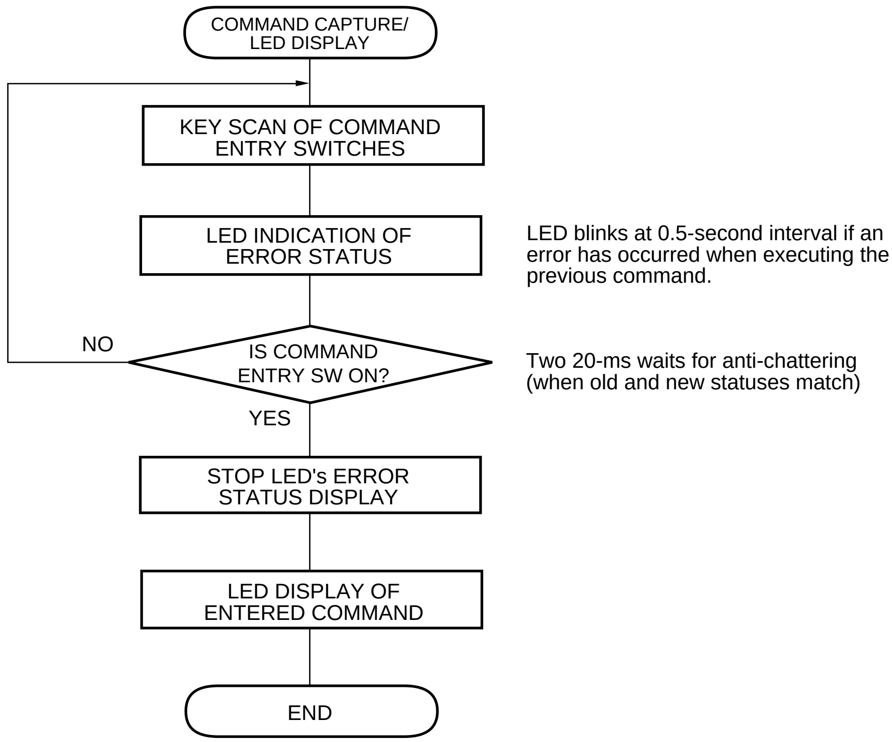
Table 5-2. Types of Errors Corresponding to Blinking LEDs

Blinking LED (blinks at 0.5-second interval)	Type of Error
LED1	Erase error
LED2	Write error
LED3	Verify error
LED4	Blank check error
LED1 and LED4	Synchronization detection error
LED1 and LED3	ACK not returned
LED1, LED3, and LED4	ACK not detected during IIC communications
LED1, LED2, and LED4	Parameter is out of range
LED1 and LED2	Receive failed
LED1, LED2, and LED3	Send failed
LED1, LED2, LED3, and LED4	Unexpected error

5.2 Sample Program

See **CHAPTER 4 SAMPLE PROGRAMS** for description of the variables used in this sample program.

(1) Flow chart



(2) Sample program

```
#pragma sfr                                         //Uses sfr area

#include "DATTYPE.H"                                //Data type definition file
#include "sram.h"                                   //RAM external access definition file
#include "constant.h"                               //Constant value definition file

/**************************************************************
*                Capture command/Display status               *
*     Global variables:  cErrorStatus          Error status         *
*                        cCommunicationMethod  Communication method *
*                                                                   *
*     Local variables:   cWork                 Work                 *
*                        cSwStatus             SW status            *
*                        cOldSwStatus          Old SW status        *
*                        c500msCounter         500-ms counter       *
**************************************************************/
void MGetCom( void ){
   register Byte cWork;                             //Work
   register Byte cSwStatus;                         //SW status
   register Byte cOldSwStatus;                      //Old SW status
   register Byte c500msCounter;                     //500-ms counter

   cSwStatus = 0;
   cOldSwStatus = 0;
   c500msCounter = 25;                              //25*20 ms = 500 ms

   PRM1 = 0x19;                                     //TM1 count clock = 2,048/fx
   CR11 = 195;                                      //195 × (2,048/20 MHz) = 20 ms
   CRC1 = 0x08;
   CE1 = 1;                                         //TM1 start

/***** Clear status display LED *****/
   cWork = P1;                                      //Clears status display LED
   cWork &= 0xf0;
   cWork |= 0x0f;
   P1 = cWork;

   while( P7 != 0x1f );                             //Waits until all switches are OFF

   for( cEnterCommand = ENTER_NOTHING ; cEnterCommand == ENTER_NOTHING ; ){
       if( CIF11 == 1 ){                            //Interrupt request flag
           CIF11 = 0;                               //Clears interrupt request flag

           c500msCounter--;

/***** Command capture (key scan) *****/
           cSwStatus = P7;
           if( cOldSwStatus == cSwStatus ){         //Anti-chattering (20-ms interval) when old and new
                                                    //statuses match
               cWork = cSwStatus ^ 0xff;
               cWork &= 0x1f;
               if(( cWork == ENTER_EPV ) || ( cWork == ENTER_ERA )
                    || (cWork == ENTER_PRG )
                    || ( cWork == ENTER_VRF ) || ( cWork == ENTER_BLN )){
                                                    //Is SW input valid?
                    cEnterCommand = cWork ;         //Sets command to be entered
               }
           }else{
               cOldSwStatus = cSwStatus;            //Stores SW status as old SW status
           }
       }
```

```
/***** Error status LED display *****/
       if(( cErrorStatus != NO_ERROR ) && ( c500msCounter == 0 )){
            c500msCounter = 25;                 //Initializes wait time (as 500 ms)
            cWork = P1;
            P1 = cWork ^ cErrorStatus;          //Blinks at 500-ms interval to indicate error status
       }
   }


/***** Error status LED OFF/LED ON corresponding to entered command *****/
   cErrorStatus = NO_ERROR;                     //Set for "no error"
   if( cEnterCommand  != ENTER_EPV ){           //Is captured command E.P.V.?
                                                //LED is ON in main routine during erase/write/verify
            cWork = cEnterCommand ^ 0xff;
            cWork &= 0x0f;
            cWork |= P1 & 0xf0;
            P1 = cWork;
   }
   else{                                        //During erase/write/verify
       cWork = P1;
       cWork &= 0xf0;
       cWork |= 0x0f;
       P1 = cWork;
   }
   CE1 = 0;                                     //Stops TM1

/***** Return to main routine (command execution) *****/
}
```

[MEMO]

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6465-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: 044-548-7900

I would like to report the following error/make the following suggestion:

Document title: ______________________

Document number: ______________________ Page number: ____________

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❏	❏	❏	❏
Technical Accuracy	❏	❏	❏	❏
Organization	❏	❏	❏	❏