NOTICE

No part of this material may be reproduced or duplicated in any form or by any means without the written permission of Seiko Epson. Seiko Epson reserves the right to make changes to this material without notice. Seiko Epson does not assume any liability of any kind arising out of any inaccuracies contained in this material or due to its application or use in any product or circuit and, further, there is no representation that this material is applicable to products requiring high level reliability, such as medical products. Moreover, no license to any intellectual property rights is granted by implication or otherwise, and there is no representation or warranty that anything made in accordance with this material will be free from any patent or copyright infringement of a third party. This material or portions thereof may contain technology or the subject relating to strategic products under the control of the Foreign Exchange and Foreign Trade Law of Japan and may require an export license from the Ministry of Economy, Trade and Industry or other approval from another government agency.

© SEIKO EPSON CORPORATION 2007, All rights reserved.
### Configuration of product number

<table>
<thead>
<tr>
<th>Devices</th>
<th>Model number</th>
<th>Package</th>
<th>Model name</th>
<th>Product classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>C</td>
<td>88104</td>
<td>F</td>
<td>0A01 00</td>
</tr>
</tbody>
</table>

- **Packing specifications**
  - 00: Besides tape & reel
  - 0A: TCP BL 2 directions
  - 0B: Tape & reel BACK
  - 0C: TCP BR 2 directions
  - 0D: TCP BT 2 directions
  - 0E: TCP BD 2 directions
  - 0F: Tape & reel FRONT
  - 0G: TCP BT 4 directions
  - 0H: TCP BD 4 directions
  - 0J: TCP SL 2 directions
  - 0K: TCP SR 2 directions
  - 0L: Tape & reel LEFT
  - 0M: TCP ST 2 directions
  - 0N: TCP SD 2 directions
  - 0P: TCP ST 4 directions
  - 0Q: TCP SD 4 directions
  - 0R: Tape & reel RIGHT
  - 99: Specs not fixed

### Development tools

<table>
<thead>
<tr>
<th>Development tools</th>
<th>Model number</th>
<th>Package</th>
<th>Model name</th>
<th>Product classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>S5U1</td>
<td>C</td>
<td>88348</td>
<td>D1</td>
<td>1 00</td>
</tr>
</tbody>
</table>

- **Packing specifications**
  - 00: standard packing

- **Version**
  - 1: Version 1

- **Tool type**
  - Hx: ICE
  - Ex: EVA board
  - Px: Peripheral board
  - Wx: Flash ROM writer for the microcomputer
  - Xx: ROM writer peripheral board
  - Cx: C compiler package
  - Ax: Assembler package
  - Dx: Utility tool by the model
  - Qx: Soft simulator

- **Corresponding model number**
  - 88348: for S1C88348

- **Tool classification**
  - C: microcomputer use

- **Product classification**
  - S5U1: development tool for semiconductor products
CONTENTS

– Contents –

1 Overview ......................................................................................................................... 1

2 Installation ...................................................................................................................... 2
   2.1 Items in the Package ................................................................................................. 2
   2.2 Working Environment ............................................................................................ 2
   2.3 How to Install the Library ..................................................................................... 3
   2.4 Installed Files ......................................................................................................... 5

3 Features of the Library .................................................................................................. 6
   3.1 Configuration of the Library Files ........................................................................... 6
   3.2 List of Library Facilities ........................................................................................ 6
   3.3 Library Size and Number of Processing Cycles .................................................... 7

4 Usage Directions for the Library .................................................................................. 8
   4.1 Adding Files into Project ....................................................................................... 8
   4.2 Locating the Library Object in the Memory ......................................................... 11

5 Creating a Program ........................................................................................................ 12
   5.1 Self-Program Processing Flow .............................................................................. 12
   5.2 Data Buffer ............................................................................................................ 13
   5.3 Error Structure spi88_err_str .............................................................................. 13
   5.4 Constant Definitions ............................................................................................. 13
   5.5 Programming Notes .............................................................................................. 14

6 Library Functions ......................................................................................................... 16
   6.1 Erase Sector Function (spi88_erase) ..................................................................... 16
   6.2 Program Function (spi88_write) .......................................................................... 18
   6.3 Verify Function (spi88_verify) ............................................................................. 20
   6.4 Blank Check Function (spi88_blank) ................................................................... 23

7 Precautions on Debugging ............................................................................................ 25

8 Restrictions ................................................................................................................... 26

Appendix Sample Programs ............................................................................................ 27
   A.1 List of Sample Programs ...................................................................................... 27
   A.2 Functions in the Sample Program ....................................................................... 29
      A.2.1 _start (Initialize Function) ............................................................................. 29
      A.2.2 main (Main Function) ................................................................................... 29
      A.2.3 spi88_wait (Wait Function) ......................................................................... 31
      A.2.4 spi88_setwritedat (Write Data Setup Function) .......................................... 31
      A.2.5 spi88_finish_proc (Termination Process Function) ................................... 32
1 Overview

The S5U1C8F626V4 is a program library for the Seiko Epson 8-bit microcomputer S1C8F626 and it provides the program modules allowing the application program to rewrite the program code and data stored in the Flash EEPROM built into the S1C8F626. The application program with the library linked can execute sector erase, program, verify, and blank check processes by calling the functions. This makes it possible to simply implement a self-programming feature into the S1C8F626 embedded applications.
2 Installation

2.1 Items in the Package
The S1C8F626 Self-Programming Library Package contains one CD-ROM in which the library files, installer and PDF manuals are included.

2.2 Working Environment
To use the S1C8F626 Self-Programming Library, the following conditions are necessary:

**Personal computer**
An IBM PC/AT or a compatible machine is required. Minimum operating conditions are a 200 MHz Pentium or a later model and 64M-byte RAM.
A PC which is equipped with a faster CPU than a 1 GHz and 256M bytes or more RAM is recommended.

**Hard disk drive and CD-ROM drive**
A CD-ROM drive and a hard disk drive (at least 10M bytes of free space) are required for installing the S1C8F626 Self-Programming Library.

**Display**
An SVGA (800 × 600 pixels) or larger display is required.

**System software**
The library and tools support Microsoft Windows 2000 Professional or Windows XP (English or Japanese version).

**Development software tool**
The S5U1C88000C1 (S1C88 Family Integrated Tool Package) is required.

**Development hardware tools**
The S5U1C88000H5, S5U1C88000P1, S5U1C88655P2, S5U1C8F626F1, and S5U1C8F626D4 tools are required.
2.3 How to Install the Library

To install the library, run the installer (Setup.exe) found on the CD-ROM provided. Before installing the S1C8F626 Self-Programming Library, make sure that the S5U1C88000C1 (S1C88 Family Integrated Tool Package) has been installed.

(1) Start Windows. If Windows is already running, close all other programs that are currently open.

(2) Insert the CD-ROM into the drive and open the root directory to display its contents.

(3) Double-click “Setup.exe” to launch the installer.

You will see the install wizard start screen.

(4) Click the [Next >] button to go to the next step.

Read the end user software license agreement displayed on the following screen.

(5) If you agree to the terms of the license, select “I accept the terms of the license agreement” and click the [Next >] button. If you do not agree, click the [Cancel] button to close the installer.

The screen displayed allows you to select the directory into which the S1C8F626 Self-Programming Library is to be installed.

(6) Check the destination directory in which the tool will be installed. To switch to a different directory, use the [Browse...] button to bring up a directory selection dialog box. From the list in this dialog box, select the directory in which you want to install the library. Click the [OK] button.

If you specify the directory in which an old version library exists, you are prompted to choose either uninstalling the old library or changing the install directory by a warning message displayed. The existing library may be left on the disk by specifying another directory.

(7) Click the [Next >] button.
2 INSTALLATION

This is the install start screen.

(8) Click the [Install] button to begin installing.

When installation is completed, a complete screen is displayed.

(9) Click the [Finish] button to quit the installer.

Canceling installation

All dialog boxes that appear during installation have a [Cancel] button. Click it to terminate the installer before installation has completed.

Uninstalling the library

To uninstall the library, use “Add/Remove Programs” on the Control Panel.
### 2.4 Installed Files

The following lists the configuration of directories and files after copying.

- **\EPSON\SPL88**
  - **ReadMe.txt**
    - readme file (Read this file first.)

- **\lib**
  - **\Large**
    - **selfprog.obj**
      - Self-programming object file
    - **spl88_def.inc**
      - External declaration/definition file (for assembler)
    - **spl88_def.h**
      - External declaration/definition file (for C)
  - **\CompactData**
    - Library directory for compact data memory model
    - **selfprog.obj**
      - Self-programming object file
    - **spl88_def.inc**
      - External declaration/definition file (for assembler)
    - **spl88_def.h**
      - External declaration/definition file (for C)
  - **\CompactCode**
    - Library directory for compact code memory model
    - **selfprog.obj**
      - Self-programming object file
    - **spl88_def.inc**
      - External declaration/definition file (for assembler)
    - **spl88_def.h**
      - External declaration/definition file (for C)
  - **\Small**
    - Library directory for small memory model
    - **selfprog.obj**
      - Self-programming object file
    - **spl88_def.inc**
      - External declaration/definition file (for assembler)
    - **spl88_def.h**
      - External declaration/definition file (for C)

- **\sample**
  - **\ASM**
    - Assembler sample directory
    - **\Large**
      - Self-programming sample (for large memory model)
    - **\CompactData**
      - Self-programming sample (for compact data memory model)
    - **\CompactCode**
      - Self-programming sample (for compact code memory model)
    - **\Small**
      - Self-programming sample (for small memory model)
  - **\C**
    - C sample directory
    - **\Large**
      - Self-programming sample (for large memory model)
    - **\CompactData**
      - Self-programming sample (for compact data memory model)
    - **\CompactCode**
      - Self-programming sample (for compact code memory model)
    - **\Small**
      - Self-programming sample (for small memory model)

- **\doc**
  - **\english**
    - English document directory
    - **manual_e.pdf**
      - Manual
    - **rel_note_e.txt**
      - Release note
  - **\japanese**
    - Japanese document directory
    - **manual_j.pdf**
      - Manual
    - **rel_note_j.txt**
      - Release note
3 Features of the Library

The library provides an object file that includes the functions required for self-programming of the S1C8F626 Flash EEPROM and header files in which various symbols are defined.

* Conditions on use
  1. The self-programming library is designed specifically for the EPSON 8-bit microcomputer S1C8F626.
  2. The library can be used for program development using the S5U1C88000C1 (S1C88 Family Integrated Tool Package).
  3. A 2.7 V or more power source voltage must be supplied to the S1C8F626 while the library functions are executed. (Refer to the “S1C8F626 Technical Manual.”)

3.1 Configuration of the Library Files

**selfprog.obj**: Object file
This object file contains the functions to process erasing, programming, verifying, and performing a blank check of the Flash EEPROM. Link this object to the application program to implement a self-programming facility.

**spl88_def.inc**: External declaration/definition file for assembler sources
This file contains the symbols decelerated with EXTERN used for calling the functions from an assembler source. Include this file when creating a self-programming module as an assembler source.

**spl88_def.h**: External declaration/definition file for C sources
This file contains the various definitions used for calling the functions from a C source. Include this file when creating a self-programming module as a C source.

The library provides different object files to support four memory models (large, compact code, compact data, and small), and they are installed with the external declaration/definition files into the directories for each different memory model (see Section 2.4). Select an appropriate object file according to the memory configuration of the application system.

3.2 List of Library Facilities

The object file provides the facilities listed below.

(1) **Erasing sector (function name: spl88_erase)**
This function erases a specified sector (4096 bytes) in the S1C8F626 Flash EEPROM.

(2) **Programming (function name: spl88_write)**
This function writes data stored in the RAM to the specified sector in the Flash EEPROM. Data size from 1 byte to 4096 bytes can be specified.

(3) **Verification (function name: spl88_verify)**
This function compares Flash EEPROM data in the specified sector with data stored in the RAM to verify the data that has been programmed. Data size from 1 byte to 4096 bytes can be specified.

(4) **Blank check (function name: spl88_blank)**
This function performs a blank check of the specified sector in the Flash EEPROM.

For details of the functions, see Chapter 6, “Library Functions.”
## 3.3 Library Size and Number of Processing Cycles

### Table 3.3.1 Library Size and Number of Processing Cycles

<table>
<thead>
<tr>
<th>Size/Number of cycles</th>
<th>Library memory model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Small, compact code</td>
</tr>
<tr>
<td>Object file size</td>
<td>834 bytes</td>
</tr>
<tr>
<td>Library work area size (RAM)</td>
<td>Stack: 34 bytes</td>
</tr>
<tr>
<td></td>
<td>Error structure: 6 bytes</td>
</tr>
<tr>
<td></td>
<td>Data buffer: Max. 4,096 bytes</td>
</tr>
<tr>
<td>Number of command processing cycles (per 1 sector)</td>
<td>Write: 151,836 cycles</td>
</tr>
<tr>
<td></td>
<td>Erase: 227 cycles</td>
</tr>
<tr>
<td></td>
<td>Verify: 73,999 cycles</td>
</tr>
<tr>
<td>Blank check</td>
<td>61,626 cycles</td>
</tr>
</tbody>
</table>
4 Usage Directions for the Library

4.1 Adding Files into Project

To use the S1C8F626 Self-Programming Library from an application program, the library files should be added to the project.

(1) Copying the files

Copy the object file and a header file into the folders shown below.

- **selfprog.obj**
  Copy this file into the OBJ folder for the project (`\<project name>\OBJ directory`). Use a “selfprog.obj” object file according to the memory model of the application system.

- **spl88_def.inc**
  Copy this file into the SRC folder for the project (`\<project name>\SRC directory`). (This file is necessary only when creating assembler sources for the self-programming module.)

- **spl88_def.h**
  Copy this file into the SRC folder for the project (`\<project name>\SRC directory`). (This file is necessary only when creating C sources for the self-programming module.)

(2) Specifying the include file

Not only can the header file (`spl88_def.inc` or `spl88_def.h`) be directly included in source files, but it can also be specified as an include file for the project using the work bench (WB88) as follows.

When including the header file into the project

1. Select [Insert file into Project] from the [Source] menu.

   ![Dialog box for selecting an include file](image)

   A dialog box for selecting an include file is displayed.

2. Change the [Files of type:] to “Include Files (*.h, *.inc)”, and select the file to be included.
   The selected file is included into the project by clicking [Open].


When the library functions are called from assembler sources

1. Open the [ASM Options] page in the option view.
2. Click the [Reference] button to display a file select dialog box and select the “spl88_def.inc” file that has been copied in Step (1) above. The file name will be inserted in the [Include Files] field. The file name can also be entered directly into the [Include Files] field.

When the library functions are called from C sources

1. Open the [C Options] page in the option view.
2. Click the [Reference] button to display a file select dialog box and select the “spl88_def.h” file that has been copied in Step (1) above. The file name will be inserted in the [Include Files] field. The file name can also be entered directly into the [Include Files] field.
4 USAGE DIRECTIONS FOR THE LIBRARY

(3) Specifying linker options

Select linker options so that the copied object file (selfprog.obj) will be linked. The following shows an operating procedure using WB88:

1. Open the [Linker Options] page in the option view.
2. Select the memory model to be used from the [Memory Model] list.

![Linker Options Page]

Options | ASM Options | Linker Options | Locator Options | Sect Options
---|---|---|---|---
Memory Model [SY] (C and Asm Compile Option)
Large 
Case Insensitive (Apply Link Option [-C] and Asm Option [-c])

Search for System Libraries [IL]
Additional Search Path:

Warning Level [-w]
8 

Turn Off Overlaying [-N]
Generate Link Map [-M]
Generate Call Graph File [-c]
Suppress Undefined Symbol Diagnostics [-r]
Print Name of Processing File (Verbose) [-v]

Linking with user libraries (Full path):
selfprog.obj

Other Options:
4.2 Locating the Library Object in the Memory

Memory location to place the library object must be defined in a locator description file (*.dsc). The following shows how to define the object location into a locator description file using WB88:

Example: when locating the S1C8F626 Self-Programming Library module beginning at address 1000H

1. Open the [Sect Options] page in the WB88 option view.
2. In the [Add Symbol (Rom)] field, click the [Addr] cell in a blank line and enter the address (e.g. 1000).
3. Enter “.SelfProgramming” in the [Name] field.
4. Click the [Kind] cell to display a pull-down list and select “Sect” from the list.
5. Enter other symbols for the application program as necessary.

The WB88 will generate a locator description file and send it to the locator.

The self-programming library module has been designed so that it can be placed at any location in the S1C8F626 internal memory. Note, however, that the area where the library can actually be located depends on the CPU mode to be used. For more information, see Section 5.5, “Programming Notes.”
5 Creating a Program

5.1 Self-Program Processing Flow

Figure 5.1.1 shows a flowchart for the self-programming routine to be created in the application program. For program examples, open the sample programs included in the library package (\SPL88\sample directory) or see Appendix.

Start self-programming routine

Set SVD criteria voltage to 2.7 V
Turn SVD On

Wait 500 µs or more

SVD ≥ 2.7 V?

Turn SVD Off

Set VDD
1.8 V → 2.5 V

Wait 5 ms or more

Preparation for issuing a command

Command error?

no

Quit command?

no

yes

Library functions

library functions

spl88_write
Program

spl88_verify
Verify

spl88_blank
Blank check

spl88_erase
Erase sector

Error handling

Set VDD
2.5 V → 1.8 V

End self-programming routine

Figure 5.1.1  Self-Program Processing Flow
5.2 Data Buffer

The application program must allocate a RAM area for the data buffer (max. 4,096 bytes) that is used to pass the code and data to be written to the Flash EEPROM to the library function. The data buffer is also used for storing the original data to be compared with the Flash EEPROM data during verification. It can be placed at any location in the RAM. Pass the start address to the library function as an argument when calling the function.

5.3 Error Structure spl88_err_str

If an error occurs during verification or blank check, the library function will write the error information to an error structure spl88_err_str. The structure members are shown below.

```c
struct spl88_err_str{
    unsigned long  spl88_err_adr;  // Address where an error has occurred
    unsigned char  spl88_org_dat;   // Original data to be compared
    unsigned char  spl88_err_dat;   // Data in which an error has occurred
};
```

5.4 Constant Definitions

The constants shown below have been defined in the header files and they can be used in user programs.

Return values from the functions

The library functions return their execution results as an unsigned char type return value. In assembler programs, they can be read from the A register. The return values from the functions have been defined as below.

<table>
<thead>
<tr>
<th>Defined name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPL88_ERR_NON</td>
<td>0</td>
<td>Terminated normally</td>
</tr>
<tr>
<td>SPL88_ERR_SCTNUM</td>
<td>1</td>
<td>Sector number error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The specified sector number is 0CH, 0DH, 0EH, 0FH, or a 40H or more value.</td>
</tr>
<tr>
<td>SPL88_ERR_DATSIZ</td>
<td>2</td>
<td>Data size error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The specified data size is 0 or a value more than 1000H.</td>
</tr>
<tr>
<td>SPL88_ERR_BLANK</td>
<td>3</td>
<td>Blank error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An error has occurred in the blank check.</td>
</tr>
<tr>
<td>SPL88_ERR_VERIFY</td>
<td>4</td>
<td>Verify error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An error has occurred in the verify check.</td>
</tr>
<tr>
<td>SPL88_ERR_VD1</td>
<td>5</td>
<td>VD1 error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The currently set VD1 voltage is 1.8 V.</td>
</tr>
</tbody>
</table>

Constants for specifying sectors

The default write/verify data size and the start and end sector numbers within the default sector range are defined as below.

<table>
<thead>
<tr>
<th>Defined name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPL88_DAT_SIZ</td>
<td>1000H</td>
<td>Write or verify data size</td>
</tr>
<tr>
<td>SPL88_START_SCTNUM</td>
<td>4</td>
<td>Start sector number for erasing, blank check, writing or verification</td>
</tr>
<tr>
<td>SPL88_END_SCTNUM</td>
<td>5</td>
<td>End sector number for erasing, blank check, writing or verification</td>
</tr>
</tbody>
</table>
5 CREATING A PROGRAM

5.5 Programming Notes

When creating a self-programming routine, take the notes below into consideration.

(1) Reserved word
The S1C8F626 Self-Programming Library uses the section name and global label/function names listed below. These names cannot be used in the user program.

Section name: .SelfProgramming
Global label names (assembler): _spl88_erase, _spl88_write, _spl88_verify, _spl88_blank
Global function names (C): spl88_erase, spl88_write, spl88_verify, spl88_blank

(2) Code efficiency
The code size and execution speed of the assembled object generated from a C source using the C compiler and assembler is about two (small model) to four times (large model) larger and slower than the code generated from an assembler source using the assembler only. Therefore, assembler program development is recommended to achieve a higher execution speed or compact code size. (The comparison result above is an index of performance, as code size depends on processing.)

(3) Combination of compiler memory model and CPU mode
In the S1C88 system, the code memory size and data memory size that can be accessed vary depending on the CPU mode and bus mode settings. The C compiler provides four memory models to support these modes.

<table>
<thead>
<tr>
<th>Compiler Memory Model</th>
<th>Code size</th>
<th>Data size</th>
<th>CPU mode</th>
<th>Bus mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small model</td>
<td>Code &lt; 64K bytes</td>
<td>Data &lt; 64K bytes</td>
<td>Minimum</td>
<td>Single chip mode (MCU)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Extended 64K mode (MPU)</td>
</tr>
<tr>
<td>Compact code model</td>
<td>Code &lt; 64K bytes</td>
<td>Data ≥ 64K bytes</td>
<td>Minimum</td>
<td>Extended 512K minimum mode</td>
</tr>
<tr>
<td>Compact data model</td>
<td>Code ≥ 64K bytes</td>
<td>Data &lt; 64K bytes</td>
<td>Maximum</td>
<td>Extended 512K maximum mode</td>
</tr>
<tr>
<td>Large model</td>
<td>Code ≥ 64K bytes</td>
<td>Data ≥ 64K bytes</td>
<td>Maximum</td>
<td>Extended 512K maximum mode</td>
</tr>
</tbody>
</table>

When minimum mode is set as the CPU mode, for example, the CARL instruction pushes a two-byte return address onto the stack. In maximum mode, the CARL instruction must push a three-byte return address. Therefore, an appropriate compiler memory model must be selected according to the CPU mode and data memory size. Do not use a combination other than one listed in the table.

The self-programming library provides four object files corresponding to each compiler memory model. Use an appropriate object file according to the application system, not only in C programming but also in assembler programming.

(4) Library allocatable area and area from which the function can be called
The pages in which the library can be allocated and pages from which the library functions can be called depend on the compiler memory model used. The small or compact code model does not allow the application to allocate the program code outside page 0.

<table>
<thead>
<tr>
<th>Large/compact data model</th>
<th>Small/Compact code model</th>
</tr>
</thead>
<tbody>
<tr>
<td>03FFFFH</td>
<td>03FFFFH</td>
</tr>
<tr>
<td>Page 3</td>
<td>Cannot be allocated</td>
</tr>
<tr>
<td>030000H</td>
<td></td>
</tr>
<tr>
<td>Page 2</td>
<td></td>
</tr>
<tr>
<td>020000H</td>
<td></td>
</tr>
<tr>
<td>Page 1</td>
<td></td>
</tr>
<tr>
<td>010000H</td>
<td></td>
</tr>
<tr>
<td>Internal memory</td>
<td></td>
</tr>
<tr>
<td>000000H</td>
<td>Page 0</td>
</tr>
<tr>
<td>(RAM, display memory, I/O memory)</td>
<td>(Can be allocated/called.)</td>
</tr>
</tbody>
</table>

Figure 5.5.1 Library Allocatable Area
## (5) Relationship between sector numbers and addresses

Table 5.5.2 lists the relationship between sector numbers and addresses. Sectors 0CH to 0FH (0C000H–0FFFFH) and Sector 40H and subsequent sectors (40000H–) cannot be specified.

<table>
<thead>
<tr>
<th>Sector number</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H</td>
<td>00000H–00FFFH</td>
</tr>
<tr>
<td>01H</td>
<td>01000H–01FFFH</td>
</tr>
<tr>
<td>02H</td>
<td>02000H–02FFFH</td>
</tr>
<tr>
<td>03H</td>
<td>03000H–03FFFH</td>
</tr>
<tr>
<td>04H</td>
<td>04000H–04FFFH</td>
</tr>
<tr>
<td>05H</td>
<td>05000H–05FFFH</td>
</tr>
<tr>
<td>06H</td>
<td>06000H–06FFFH</td>
</tr>
<tr>
<td>07H</td>
<td>07000H–07FFFH</td>
</tr>
<tr>
<td>08H</td>
<td>08000H–08FFFH</td>
</tr>
<tr>
<td>09H</td>
<td>09000H–09FFFH</td>
</tr>
<tr>
<td>0AH</td>
<td>0A000H–0AFFFH</td>
</tr>
<tr>
<td>0BH</td>
<td>0B000H–0BFFFH</td>
</tr>
<tr>
<td>(0CH)*</td>
<td>0C000H–0CFFFH</td>
</tr>
<tr>
<td>(0DH)*</td>
<td>0D000H–0DFFH</td>
</tr>
<tr>
<td>(0EH)*</td>
<td>0E000H–0EFFFH</td>
</tr>
<tr>
<td>(0FH)*</td>
<td>0F000H–0FFFFH</td>
</tr>
<tr>
<td>10H</td>
<td>10000H–10FFFH</td>
</tr>
<tr>
<td>11H</td>
<td>11000H–11FFFH</td>
</tr>
<tr>
<td>12H</td>
<td>12000H–12FFFH</td>
</tr>
<tr>
<td>13H</td>
<td>13000H–13FFFH</td>
</tr>
<tr>
<td>14H</td>
<td>14000H–14FFFH</td>
</tr>
<tr>
<td>15H</td>
<td>15000H–15FFFH</td>
</tr>
<tr>
<td>16H</td>
<td>16000H–16FFFH</td>
</tr>
<tr>
<td>17H</td>
<td>17000H–17FFFH</td>
</tr>
<tr>
<td>18H</td>
<td>18000H–18FFFH</td>
</tr>
<tr>
<td>19H</td>
<td>19000H–19FFFH</td>
</tr>
<tr>
<td>1AH</td>
<td>1A000H–1AFFFH</td>
</tr>
<tr>
<td>1BH</td>
<td>1B000H–1BFFFH</td>
</tr>
<tr>
<td>1CH</td>
<td>1C000H–1CFFFH</td>
</tr>
<tr>
<td>1DH</td>
<td>1D000H–1DFFH</td>
</tr>
<tr>
<td>1EH</td>
<td>1E000H–1EFFFH</td>
</tr>
<tr>
<td>1FH</td>
<td>1F000H–1FFFFH</td>
</tr>
<tr>
<td>20H</td>
<td>20000H–20FFFH</td>
</tr>
<tr>
<td>21H</td>
<td>21000H–21FFFH</td>
</tr>
<tr>
<td>22H</td>
<td>22000H–22FFFH</td>
</tr>
<tr>
<td>23H</td>
<td>23000H–23FFFH</td>
</tr>
<tr>
<td>24H</td>
<td>24000H–24FFFH</td>
</tr>
<tr>
<td>25H</td>
<td>25000H–25FFFH</td>
</tr>
<tr>
<td>26H</td>
<td>26000H–26FFFH</td>
</tr>
<tr>
<td>27H</td>
<td>27000H–27FFFH</td>
</tr>
<tr>
<td>28H</td>
<td>28000H–28FFFH</td>
</tr>
<tr>
<td>29H</td>
<td>29000H–29FFFH</td>
</tr>
<tr>
<td>2AH</td>
<td>2A000H–2AFFFH</td>
</tr>
<tr>
<td>2BH</td>
<td>2B000H–2BFFFH</td>
</tr>
<tr>
<td>2CH</td>
<td>2C000H–2CFFFH</td>
</tr>
<tr>
<td>2DH</td>
<td>2D000H–2DFFH</td>
</tr>
<tr>
<td>2EH</td>
<td>2E000H–2EFFFH</td>
</tr>
<tr>
<td>2FH</td>
<td>2F000H–2FFFFH</td>
</tr>
<tr>
<td>30H</td>
<td>30000H–30FFFH</td>
</tr>
<tr>
<td>31H</td>
<td>31000H–31FFFH</td>
</tr>
<tr>
<td>32H</td>
<td>32000H–32FFFH</td>
</tr>
<tr>
<td>33H</td>
<td>33000H–33FFFH</td>
</tr>
<tr>
<td>34H</td>
<td>34000H–34FFFH</td>
</tr>
<tr>
<td>35H</td>
<td>35000H–35FFFH</td>
</tr>
<tr>
<td>36H</td>
<td>36000H–36FFFH</td>
</tr>
<tr>
<td>37H</td>
<td>37000H–37FFFH</td>
</tr>
<tr>
<td>38H</td>
<td>38000H–38FFFH</td>
</tr>
<tr>
<td>39H</td>
<td>39000H–39FFFH</td>
</tr>
<tr>
<td>3AH</td>
<td>3A000H–3AFFFH</td>
</tr>
<tr>
<td>3BH</td>
<td>3B000H–3BFFFH</td>
</tr>
<tr>
<td>3CH</td>
<td>3C000H–3CFFFH</td>
</tr>
<tr>
<td>3DH</td>
<td>3D000H–3DFFH</td>
</tr>
<tr>
<td>3EH</td>
<td>3E000H–3EFFFH</td>
</tr>
<tr>
<td>3FH</td>
<td>3F000H–3FFFFH</td>
</tr>
</tbody>
</table>

* Cannot be specified.
6 Library Functions

This chapter explains each library function.

Note: This chapter describes the function names in the C format. When writing them in assembler sources, '_' must be prefixed to the function names.

Example: C

```c
unsigned char stat = spl88_erase(sector_num);
```

Assembler

```
CARL _spl88_erase
```

6.1 Erase Sector Function (spl88_erase)

<table>
<thead>
<tr>
<th>Function</th>
<th>unsigned char spl88_erase(unsigned int sector_num);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Erases a specified sector in the S1C8F626 Flash EEPROM. While this function is being executed, the watchdog timer and all interrupts are disabled.</td>
</tr>
<tr>
<td>Argument</td>
<td>BA register (unsigned int sector_num) Sector number (see Table 5.5.2.) 00H–0BH, 10H–3FH</td>
</tr>
<tr>
<td>Return value</td>
<td>A register (unsigned char) Status (see Table 5.4.1.) SPL88_ERR_NON: Terminated normally SPL88_ERR_SCTNUM: Sector number error SPL88_ERR_VD1: VD1 error</td>
</tr>
<tr>
<td>Output data</td>
<td>None</td>
</tr>
</tbody>
</table>
| Usage example | [ASM]

```asm
LD BA,#001H ; Sets the sector No. to be erased (BA). Sector No. = 1
CARL _spl88_erase ; Calls the erase sector function.
CP A,#000H ; Checks the status bits.
```

[C]

```c
unsigned char stat; // Status (= A)
unsigned int sectornum; // Sector No. (= BA)
sectornum = 0x1; // Sets the sector No. to be erased (sectornum = 1).
stat = spl88_erase(sectornum); // Calls the erase sector function.
if(stat != 0){ // Checks the status bits.
```
Erase sector processing flow

The following shows a procedure to erase a sector.

1. Set SVD criteria voltage to 2.7 V
   Turn SVD On

2. Wait 500 µs or more

3. SVD ≥ 2.7 V?
   yes
   no

4. Turn SVD Off

5. Set VD1
   1.8 V → 2.5 V

6. Wait 5 ms or more

7. Preparation for issuing a command

8. Library function call
   spl88_erase
   Erase sector

9. Command error?
   yes
   no

10. Quit command?
    yes
    no

11. Error handling

12. Set VD1
    2.5 V → 1.8 V

13. End erase sector routine

1. Set the SVD criteria voltage to 2.7 V and turn the SVD circuit on to check the supply voltage.
2. Wait 500 µs or more.
3. Check if the supply voltage is 2.7 V or more using the SVD circuit. Branch to Step 4 when the supply voltage is 2.7 V or more, or branch to Step 11 if it is less than 2.7 V.
4. Turn the SVD circuit off.
5. Switch the VD1 voltage from 1.8 V to 2.5 V.
6. Wait 5 ms or more before calling the spl88_erase function after switching the VD1 voltage.
7. In an assembler source, set the sector number to be erased to the BA register.
   In a C source, declare the (unsigned int)sectornum variable and substitute the sector number to be erased for it.
8. In an assembler source, call _spl88_erase.
   In a C source, call the spl88_erase function with sectornum as its argument.
   The called function starts processing to erase the specified sector.
9. In an assembler source, check the results of the erase sector processing by reading the A register.
   In a C source, check the return value from the spl88_erase function.
   Branch to Step 10 when the function has terminated normally, or branch to Step 11 if an error has occurred.
10. Branch to Step 12 to terminate the command processing, or branch to Step 7 to continue.
11. Perform an error handling.
12. Switch the VD1 voltage from 2.5 V to 1.8 V.
13. Terminate the erase sector processing routine.
6.2 Program Function (spl88_write)

<table>
<thead>
<tr>
<th>Function</th>
<th>unsigned char spl88_write(unsigned char* pdata, unsigned int sector_num, unsigned int size);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Writes data specified with a pointer to the specified sector in the S1C8F626 Flash EEPROM. Data size from 1 byte to 4096 bytes can be specified. While this function is being executed, the watchdog timer and all interrupts are disabled.</td>
</tr>
</tbody>
</table>
| Arguments | YP and IY registers (unsigned char* pdata) | Pointer to data to be written (RAM)  
YP: one high-order byte of address  
IY: two low-order bytes of address |
| | BA register (unsigned int sector_num) | Sector number (see Table 5.5.2.)  
00H–0BH, 10H–3FH |
| | HL register (unsigned int size) | Data size to be written  
1–4096 |
| Return value | A register (unsigned char) | Status (see Table 5.4.1.)  
SPL88_ERR_NON: Terminated normally  
SPL88_ERR_SCTNUM: Sector number error  
SPL88_ERR_DATSIZ: Data size error  
SPL88_ERR_VD1: VD1 error |
| Output data | None |
| Usage example | [ASM]  
LD YP,#@DPAG(spl88_rxp_dat);  
Sets one high-order byte of the pointer to the write data (YP).  
LD IY,#@DOFF(spl88_rxp_dat);  
Sets two low-order bytes of the pointer to the write data (IY).  
LD BA,#001H;  
Sets the write sector No. (BA). Sector No. = 1  
LD HL,#01000H;  
Sets the write data size (HL). 4096 bytes  
CARL _spl88_write;  
Calls the program function.  
CP A,#000H;  
Checks the status bits.  
[
C
]  
unsigned char stat;  
// Status (= A)  
unsigned char* pdat;  
// Pointer to the write data (= YP-IY)  
unsigned int sectornum;  
// Write sector No. (= BA)  
unsigned int datasize;  
// Write data size (= HL)  
pdat = (unsigned char*)malloc(0x1000);  
// Allocates a write data area.  
...  
sectornum = 0x1;  
// Sets the write sector No. (1).  
datasize = 0x1000;  
// Sets the write data size.  
stat = spl88_write(pdat, sectornum, datasize);  
// Calls the program function.  
if(stat != 0){  
// Checks the status bits.  
...  
}  
free(pdat);  
// Deallocates the write data area. |
Program processing flow

The following shows a procedure to write data.

1. Set SVD criteria voltage to 2.7 V and turn the SVD circuit on to check the supply voltage.
2. Wait 500 µs or more.
3. Check if the supply voltage is 2.7 V or more using the SVD circuit. Branch to Step 4 when the supply voltage is 2.7 V or more, or branch to Step 11 if it is less than 2.7 V.
4. Turn the SVD circuit off.
5. Switch the VD1 voltage from 1.8 V to 2.5 V.
6. Wait 5 ms or more before calling the spl88_write function after switching the VD1 voltage.
7. In an assembler source, set the pointer (start address) to the write data to the YP and IY registers, the write sector number to the BA register, and the write data size to the HL register.
   In a C source, set the pointer to the write data, the write sector number, and the write data size to the (unsigned char*)pdat, (unsigned int)sectornum, and (unsigned int)datasize variables, respectively.
8. In an assembler source, call _spl88_write.
   In a C source, call the spl88_write function with pdat, sectornum, and datasize as its arguments.
   The called function starts processing to write data.
9. In an assembler source, check the results of the program processing by reading the A register.
   In a C source, check the return value from the spl88_write function.
   Branch to Step 10 when the function has terminated normally, or branch to Step 11 if an error has occurred.
10. Branch to Step 12 to terminate the command processing, or branch to Step 7 to continue.
11. Perform an error handling.
12. Switch the VD1 voltage from 2.5 V to 1.8 V.
13. Terminate the erase sector processing routine.
### 6.3 Verify Function (spl88_verify)

**Function**

```c
unsigned char spl88_verify(unsigned char* pdata, unsigned int sector_num, unsigned int size,
                        spl88_err_str* pSpl88_err_str);
```

**Description**

Compares S1C8F626 Flash EEPROM data in the specified sector with data specified with a pointer to verify the data that has been programmed. Verification size from 1 byte to 4096 bytes can be specified. While this function is being executed, the watchdog timer and all interrupts are disabled.

**Arguments**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YP and IY registers (unsigned char* pdata)</td>
<td>Pointer to the original data for comparison (RAM)</td>
<td>YP: one high-order byte of address IY: two low-order bytes of address</td>
</tr>
<tr>
<td>BA register (unsigned int sector_num)</td>
<td>Sector number (see Table 5.5.2.)</td>
<td>00H–0BH, 10H–3FH</td>
</tr>
<tr>
<td>HL register (unsigned int size)</td>
<td>Verification size</td>
<td>1–4096</td>
</tr>
<tr>
<td>IX register (spl88_err_str* pSpl88_err_str)</td>
<td>Pointer to the error structure (spl88_err_str)</td>
<td></td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A register (unsigned char)</td>
<td>Status (see Table 5.4.1.)</td>
<td>SPL88_ERR_NON: Terminated normally SPL88_ERR_SCTNUM: Sector number error SPL88_ERR_DATSIZ: Data size error SPL88_ERR_VERIFY: Verify error SPL88_ERR_VD1: Vd1 error</td>
</tr>
</tbody>
</table>

**Output data**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>spl88_err_str.spl88_err_adr</td>
<td>Address where an error has occurred (Flash EEPROM)</td>
<td></td>
</tr>
<tr>
<td>spl88_err_str.spl88_org_dat</td>
<td>Original data for comparison (RAM)</td>
<td></td>
</tr>
<tr>
<td>spl88_err_str.spl88_err_dat</td>
<td>Data in which an error has occurred (Flash EEPROM)</td>
<td></td>
</tr>
</tbody>
</table>

**Usage example**

```asm
LD XP,#@DPAG(spl88_err_str) ; Sets the page address of the structure pointer.
LD IX,#@DOFF(spl88_err_str) ; Sets the address of the structure pointer.
PUSH IX
LD IX,SP ; Stack pointer (structure pointer on the stack)
LD YP,#@DPAG(spl88_rxp_dat) ; Sets one high-order byte of the pointer to the original comparison data (YP).
LD IY,#@DOFF(spl88_rxp_dat) ; Sets two low-order bytes of the pointer to the original comparison data (IY).
LD BA,#001H ; Sets the verification sector No. (BA). Sector No. = 1
LD HL,#01000H ; Sets the verification size (HL).
CALL _spl88_verify ; Calls the verify function.
CP A,#000H ; Checks the status bits.
POP IX
```
6 LIBRARY FUNCTIONS

Usage example

```c
unsigned char stat; // Status (= A)
unsigned char* pdat; // Pointer to the original data (= YP-IY)
unsigned int sectornum; // Verification sector No. (= BA)
unsigned int datasize; // Verification size (= HL)
spl88_err_str* pSpl88errstr; // Error structure (= IX)
pdat = (unsigned char*) malloc(0x1000); // Allocates an original data area.
... // Sets data to the area.
pSpl88errstr = (spl88_err_str*) malloc(sizeof(spl88_err_str)); // Allocates an area for the error structure.
sectornum = 0x1; // Sets the verification sector No. (1).
datasize = 0x1000; // Sets the verification size.
stat = spl88_verify(pdat, sectornum, datasize, (spl88_err_str*) &pSpl88errstr); // Calls the verify function.
if(stat != 0){ // Checks the status bits.
  ...
  // Error handling
}
free(pdat); // Deallocates the original data area.
free(pSpl88errstr); // Deallocates the error structure area.
```

Verify processing flow

The following shows a procedure to verify data.

1. Set SVD criteria voltage to 2.7 V
   → Turn SVD On
2. Wait 500 µs or more
3. SVD ≥ 2.7 V?
   yes
   → Turn SVD Off
4. Set VD1
   → 1.8 V → 2.5 V
5. Wait 5 ms or more
6. Preparation for issuing a command
7. Library function call
   spl88_verify
   Verify
8. Command error?
   yes
   → Quit command?
   no
9. no
10. yes
11. Error handling
12. Set VD1
   → 2.5 V → 1.8 V
13. End verify routine
6 LIBRARY FUNCTIONS

1. Set the SVD criteria voltage to 2.7 V and turn the SVD circuit on to check the supply voltage.
2. Wait 500 µs or more.
3. Check if the supply voltage is 2.7 V or more using the SVD circuit. Branch to Step 4 when the supply voltage is 2.7 V or more, or branch to Step 11 if it is less than 2.7 V.
4. Turn the SVD circuit off.
5. Switch the VDI voltage from 1.8 V to 2.5 V.
6. Wait 5 ms or more before calling the spl88_verify function after switching the VDI voltage.
7. In an assembler source, set the pointer (start address) to the original comparison data to the YP and IY registers, the verification sector number to the BA register, the verification size to the HL register, and the pointer to the error structure to the IX register.
   In a C source, set the pointer to the original comparison data, the verification sector number, the verification size, and the pointer to the error structure to the (unsigned char*)pdat, (unsigned int)sectornum, (unsigned int)datasize, and (spl88_err_str*)pSpl88errstr variables, respectively.
8. In an assembler source, call _spl88_verify.
   In a C source, call the spl88_verify function with pdat, sectornum, datasize, and pSpl88errstr as its arguments.
   The called function starts processing to verify data.
9. In an assembler source, check the results of the verify processing by reading the A register.
   In a C source, check the return value from the spl88_verify function.
   Branch to Step 10 when the function has terminated normally, or branch to Step 11 if an error has occurred.
10. Branch to Step 12 to terminate the command processing, or branch to Step 7 to continue.
11. Perform an error handling.
12. Switch the VDI voltage from 2.5 V to 1.8 V.
13. Terminate the erase sector processing routine.
### 6.4 Blank Check Function (spl88_blank)

<table>
<thead>
<tr>
<th>Function</th>
<th><code>unsigned char spl88_blank(unsigned int sector_num, spl88_err_str* pSpl88_err_str);</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Performs a blank check (checks if data is 0FFH) of the specified sector (4096 bytes) in the S1C8F626 Flash EEPROM. While this function is being executed, the watchdog timer and all interrupts are disabled.</td>
</tr>
<tr>
<td>Argument</td>
<td><strong>BA register</strong> (unsigned int sector_num)</td>
</tr>
<tr>
<td></td>
<td>Sector number (see Table 5.5.2.) 00H–0BH, 10H–3FH</td>
</tr>
<tr>
<td></td>
<td><strong>IY register</strong> (spl88_err_str* pSpl88_err_str)</td>
</tr>
<tr>
<td></td>
<td>Pointer to the error structure (spl88_err_str)</td>
</tr>
<tr>
<td>Return value</td>
<td><strong>A register</strong> (unsigned char)</td>
</tr>
<tr>
<td></td>
<td>Status (see Table 5.4.1.)</td>
</tr>
<tr>
<td></td>
<td>SPL88_ERR_NON: Terminated normally</td>
</tr>
<tr>
<td></td>
<td>SPL88_ERR_SCTNUM: Sector number error</td>
</tr>
<tr>
<td></td>
<td>SPL88_ERR_BLANK: Blank error</td>
</tr>
<tr>
<td></td>
<td>SPL88_ERR_VD1: VD1 error</td>
</tr>
<tr>
<td>Output data</td>
<td><strong>(unsigned long)</strong> spl88_err_str.spl88_err adr</td>
</tr>
<tr>
<td></td>
<td>Address where an error has occurred (Flash EEPROM)</td>
</tr>
<tr>
<td></td>
<td><strong>(unsigned char)</strong> spl88_err_str.spl88_org_dat</td>
</tr>
<tr>
<td></td>
<td>Original data (0FFH)</td>
</tr>
<tr>
<td></td>
<td><strong>(unsigned char)</strong> spl88_err_str.spl88_err_dat</td>
</tr>
<tr>
<td></td>
<td>Data in which an error has occurred (Flash EEPROM)</td>
</tr>
<tr>
<td>Usage example</td>
<td><strong>[ASM]</strong></td>
</tr>
<tr>
<td></td>
<td><code>LD YP,#@DPAG(spl88_err_str) ;</code> Sets the page address of the structure pointer.</td>
</tr>
<tr>
<td></td>
<td><code>LD IY,#@DOFF(spl88_err_str) ;</code> Sets the address of the structure pointer.</td>
</tr>
<tr>
<td></td>
<td><code>PUSH IY ;</code> Stack pointer (structure pointer on the stack)</td>
</tr>
<tr>
<td></td>
<td><code>LD BA,#001H ;</code> Sets the blank check sector No. (BA). Sector No. = 1</td>
</tr>
<tr>
<td></td>
<td><code>CALL spl88_blank ;</code> Calls the blank check function.</td>
</tr>
<tr>
<td></td>
<td><code>CP A,#000H ;</code> Checks the status bits.</td>
</tr>
<tr>
<td></td>
<td><code>POP IY</code></td>
</tr>
<tr>
<td></td>
<td><strong>[C]</strong></td>
</tr>
<tr>
<td></td>
<td><code>unsigned char stat; // Status (= A)</code></td>
</tr>
<tr>
<td></td>
<td><code>unsigned int sectornum; // Blank check sector No. (= BA)</code></td>
</tr>
<tr>
<td></td>
<td><code>spl88_err_str* pSpl88errstr; // Error structure (= IY)</code></td>
</tr>
<tr>
<td></td>
<td><code>pSpl88errstr = (spl88_err_str*) malloc(sizeof(spl88_err_str)); // Allocates an area for the error structure.</code></td>
</tr>
<tr>
<td></td>
<td><code>sectornum = 0x1; // Sets the blank check sector No. (1).</code></td>
</tr>
<tr>
<td></td>
<td><code>stat = spl88_blank(sectornum, (spl88_err_str*) &amp;pSpl88errstr); // Calls the blank check function.</code></td>
</tr>
<tr>
<td></td>
<td><code>if(stat != 0){ // Checks the status bits.</code></td>
</tr>
<tr>
<td></td>
<td><code>...</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
</tr>
<tr>
<td></td>
<td><code>free(pSpl88errstr); // Deallocates the error structure area.</code></td>
</tr>
</tbody>
</table>
6 LIBRARY FUNCTIONS

Blank check processing flow

The following shows a procedure for blank check.

1. Set SVD criteria voltage to 2.7 V and turn SVD On.
2. Wait 500 µs or more.
3. Check if SVD voltage is ≥2.7 V.
   - If yes, go to Step 4.
   - If no, go to Step 1.
4. Turn SVD Off.
5. Set VD1 voltage from 1.8 V to 2.5 V.
6. Wait 5 ms or more before calling the spl88_blank function.
7. In an assembler source, set the sector number of the Flash EEPROM to be blank checked to the BA register and the pointer to the error structure to the IY register.
   In a C source, set the blank check sector number and the pointer to the error structure to the (unsigned int)sectornum and (spl88_err_str*)pSpl88errstr variables, respectively.
8. In an assembler source, call _spl88_blank.
   In a C source, call the spl88_blank function with sectornum and pSpl88errstr as its arguments.
   The called function starts blank check processing.
9. In an assembler source, check the results of the blank check processing by reading the A register.
   In a C source, check the return value from the spl88_blank function.
   Branch to Step 10 when the function has terminated normally, or branch to Step 11 if an error has occurred.
10. Branch to Step 12 to terminate the command processing, or branch to Step 7 to continue.
11. Perform an error handling.
12. Switch the VD1 voltage from 2.5 V to 1.8 V.
13. Terminate the erase sector processing routine.
7 Precautions on Debugging

Take the following precautions when debugging the program in which the self-programming library is linked.

- Edit the “Internal ROM” parameters in the parameter file (8F626.par) as follows before debugging the program:
  
  Map0=000000 00BFFF U W → Map0=000000 00BFFF U
  Map1=010000 03FFFF U W → Map1=010000 03FFFF U

- The erase sector and program functions in the library will always be executed without any prompt even if the self-programming library, C library, or user code is located in the specified sector. Make sure that the correct sector is specified when calling the erase sector or program function.

- The library functions do not run if the supply voltage is less than 2.7 V. Evaluate the program using an actual application system in addition to debugging with development tools.

- The library uses 34 bytes in the stack area. Note that the library functions will not be executed normally if this area is overwritten.

- Note that all the interrupts and the watchdog timer are disabled while the library function is being executed.
8 Restrictions

The library functions cannot be run in a built-in Flash EEPROM processors other than the S1C8F626.

The library functions may not operate normally if the S1C8F626 CPU mode and compiler memory model are not matched correctly.

---

Table 8.1 Combination of Compiler Memory Model and CPU Mode

<table>
<thead>
<tr>
<th>Compiler memory model</th>
<th>Minimum mode</th>
<th>Maximum mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small model</td>
<td>○</td>
<td>×</td>
</tr>
<tr>
<td>Compact code model</td>
<td>○</td>
<td>×</td>
</tr>
<tr>
<td>Compact data model</td>
<td>×</td>
<td>○</td>
</tr>
<tr>
<td>Large model</td>
<td>×</td>
<td>○</td>
</tr>
</tbody>
</table>

(○: can be used, ×: cannot be used)

• The watchdog timer is disabled while the library function is being executed.

• All interrupts are disabled while the library function is being executed.

• When creating the self-programming module in assembler, the library functions use and overwrite the general-purpose registers. Therefore, be sure to save the general-purpose register values before calling the library functions.

• Do not switch the VDD level (1.8 V → 2.5 V, 2.5 V → 1.8 V) every time the sector to be programmed is changed.

• A 2.7 V or more supply voltage is required to execute the library functions. Refer to the “S1C8F626 Technical Manual” for more information.

• The self-programming library supports only writing the code and data stored in the S1C8F626 RAM to the Flash EEPROM, and it does not support loading code and data from a PC to the RAM. Prepare a user program and circuits to transfer code and data from a PC if it is required.
Appendix  Sample Programs

The S1C8F626 Self-Programming Library Package includes sample programs that perform the processing listed below.

1. Controlling SVD (check if the supply voltage is 2.7 V or more)
2. Controlling the V\textsubscript{DD} voltage (set it to 2.5 V during self-programming)
3. Erasing a sector (address range to be erased: 4000H–4FFFH)
4. Blank check for a sector (blank check address range: 4000H–4FFFH)
5. Programming a sector (program address range: 4000H–4FFFH, write data: 04H)
6. Verify check for a sector (verification address range 4000H–4FFFH)
7. Function error handling and termination processing

A.1 List of Sample Programs

The sample programs are copied into the C:\EPSON\SPL88\sample directory (default) during installation of the library. The sample directory contains subdirectories for different source language and memory models as shown below. Each program located in the subdirectories has the same facilities.

C:\EPSON
  \SPL88
  \sample
  \ASM Assembler sample directory
  \Small Assembler sample program for small model
  \SRC
    boot.asm Startup routine source file
    sample.asm Main routine source file
    spl88_def.inc External symbol declaration/definition file
  \OBJ
    selfprog.obj Self-programming library object file
  \CompactCode Assembler sample program for compact code model
  \SRC
    boot.asm Startup routine source file
    sample.asm Main routine source file
    spl88_def.inc External symbol declaration/definition file
  \OBJ
    selfprog.obj Self-programming library object file
  \CompactData Assembler sample program for compact data model
  \SRC
    boot.asm Startup routine source file
    sample.asm Main routine source file
    spl88_def.inc External symbol declaration/definition file
  \OBJ
    selfprog.obj Self-programming library object file
  \Large Assembler sample program for large model
  \SRC
    boot.asm Startup routine source file
    sample.asm Main routine source file
    spl88_def.inc External symbol declaration/definition file
  \OBJ
    selfprog.obj Self-programming library object file
APPENDIX SAMPLE PROGRAMS

\C C sample directory
\Small C sample program for small model
\SRC
  cstart.s Startup routine source file
  sample.c Main routine source file
  spl88_def.h External symbol declaration/definition file
\OBJ
  selfprog.obj Self-programming library object file

\CompactCode C sample program for compact code model
\SRC
  cstart.s Startup routine source file
  sample.c Main routine source file
  spl88_def.h External symbol declaration/definition file
\OBJ
  selfprog.obj Self-programming library object file

\CompactData C sample program for compact data model
\SRC
  cstart.s Startup routine source file
  sample.c Main routine source file
  spl88_def.h External symbol declaration/definition file
\OBJ
  selfprog.obj Self-programming library object file

\Large C sample program for large model
\SRC
  cstart.s Startup routine source file
  sample.c Main routine source file
  spl88_def.h External symbol declaration/definition file
\OBJ
  selfprog.obj Self-programming library object file
A.2 Functions in the Sample Program

The sample program contains the functions shown below.

(1) _start Initialize function
(2) main Main function
(3) spl88_wait Wait function
(4) spl88_setwritedat Write data setup function
(5) spl88_finish_proc Termination process function

A.2.1 _start (Initialize Function)

Function void _start(void);

Description This function is executed by the CPU after an initial reset to initialize some I/O registers.
First, the function sets the CPU mode through I/O address FF00H. The sample program for the small
or compact code model sets the CPU to minimum mode. The sample program for the compact data or
large model sets the CPU to maximum mode.
Next, the function sets the stack page to 0 through I/O address FF01H, and then sets the stack pointer.
Finally, it sets the CPU clock to OSC3 through I/O address FF02H and calls the main function.

Arguments None

Return value None

A.2.2 main (Main Function)

Function void main(void);

Description This is the main routine of the sample program and is executed in the ROM.
First, this function sets up the stopwatch timer to generate wait times for SVD and VD1. Next, it controls
the SVD circuit to check if a 2.7 V or more power voltage is supplied. If the supply voltage is less than
2.7 V, this function calls the spl88_finish_proc function to terminate the self-programming routine with
an error. When the supply voltage is 2.7 V or more, it switches VD1 to 2.5 V for Flash programming. After
waiting for stabilization of the VD1 voltage, it performs erasing, blank check, writing data (04H), and a
verify check for Flash sector 4 (4000H–4FFFH) sequentially.
If an error occurs during processing, it calls the spl88_finish_proc function to terminate the self-pro-
gramming routine with an error.
When the verify check is completed normally, it calls the spl88_finish_proc function to terminate the
self-programming routine with no error.

Arguments None

Return value None
APPENDIX SAMPLE PROGRAMS

main

- Set up stopwatch timer: Initialize the 100 Hz stopwatch timer to run.
- Set up SVD: Set the criteria voltage to 2.7 V and activate the SVD.
  - Flash programming cannot be performed if VDD < 2.7 V.
  - Wait for the SVD to stabilize.
  - At least 500 µs wait time is required after the SVD is activated.

- spl88_wait: Wait (10 ms)
  - SVD ≥ 2.7 V?
    - yes: Check the SVD result.
      - Switch VD1 from 1.8 V to 2.5 V.
      - Flash programming cannot be performed if VD1 = 1.8 V.
    - no: Wait for VD1 to stabilize.
      - At least 5 ms wait time is required after VD1 is switched.

- spl88_wait: Wait (10 ms)

- Set sector number: Set the Flash sector number. The sample program sets 04H.

- spl88_erase: Erase sector
  - Error?
    - yes: Check the spl88_erase function return value.
    - no: Call the spl88_blank function with the sector number and error structure specified.

- spl88_blank: Blank check
  - Error?
    - yes: Check the spl88_blank function return value.
    - no: Set the write data to the specified RAM area (4096 bytes).

- spl88_setwritedat: Set up write data

- spl88_write: Program
  - Error?
    - yes: Call the spl88_write function with the write data, sector number, and data size specified.
    - no: Call the spl88_verify function with the original comparison data, sector number, data size, and error structure specified.

- spl88_verify: Verify
  - Error?
    - yes: Check the spl88_verify function return value.
    - no: Completed?
      - yes: The sample program finishes only after sector 04H is completed.
      - no: Normal termination process: Call the spl88_finish_proc function.

Figure A.2.2.1 main Flowchart
A.2.3  spl88_wait (Wait Function)

<table>
<thead>
<tr>
<th>Function</th>
<th>void spl88_wait(void);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This function is called by the main routine to wait until the SVD or VD1 operation has stabilized using the stopwatch timer. This function returns to the caller function after the 100 Hz stopwatch timer has counted up for about 10 ms.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return value</td>
<td>None</td>
</tr>
</tbody>
</table>
| Note | • Before this function can be used, the 100 Hz stopwatch timer must be set up (refer to the source of the main function).  
  • It is not necessary to use the stopwatch timer to generate wait times. However, a wait time that exceeds the stability time is required when the SVD circuit activates or VD1 is switched. Generate appropriate wait times using a method possible in the application system. |

```
spl88_wait
  Read stopwatch timer
  100 Hz interrupt factor flag
  no
  Flag = 1?  
  yes
  Reset stopwatch timer
  100 Hz interrupt factor flag
  Write 0 to the interrupt factor flag to reset.
  RET
```

Figure A.2.3.1 spl88_wait Flowchart

A.2.4  spl88_setwritedat (Write Data Setup Function)

<table>
<thead>
<tr>
<th>Function</th>
<th>void spl88_setwritedat(unsigned char* spl88_rxp_dat, unsigned int sectornum);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This function sets data in the 4096-byte data buffer (RAM area) specified with the pointer. The sample program fills the 4096-byte area with 04H.</td>
</tr>
</tbody>
</table>
| Arguments | YP and IY registers  
( unsigned char* spl88_rxp_dat)  
BA register  
( unsigned int sectornum)  
Pointer to data buffer (RAM)  
YP: one high-order byte of address  
IY: two low-order bytes of address  
Sector number |
| Return value | None |
| Note | This function is created just for the sample program use and it sets a fixed value to a RAM area. Note that a feature that can be used for applications is not implemented. |
APPENDIX  SAMPLE PROGRAMS

Figure A.2.4.1 spl88_setwritedat Flowchart

A.2.5 spl88_finish_proc (Termination Process Function)

<table>
<thead>
<tr>
<th>Function</th>
<th>void spl88_finish_proc(unsigned char* spl88_rxp_dat, spl88_err_str* pSpl88_err_str);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This function performs processing for termination after the library functions are executed. It disables SVD and returns VDD to 1.8 V for normal mode. Also it deallocates the memory areas for the 4096-byte data buffer and the error structure. Finally, it sets the CPU to HALT mode.</td>
</tr>
</tbody>
</table>
| Arguments | YP and IY registers (unsigned char* spl88_rxp_dat)  \( \text{YP: one high-order byte of address} \) \( \text{IY: two low-order bytes of address} \)  
| | XP and IX registers (spl88_err_str* pSpl88_err_str)  \( \text{XP: one high-order byte of address} \) \( \text{IX: two low-order bytes of address} \) |
| Return value | None |

Figure A.2.5.1 spl88_finish_proc Flowchart