EPSON
EXCEED YOUR VISION

CMOS 8-BIT SINGLE CHIP MICROCOMPUTER

S5U1C8F626Y4 Manual

(S1C8F626 Self-Programming Library)

SEIKO EPSON CORPORATION

Configuration of product number

Devices

S1 C 88104 F 0A01 00

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]

Specification

Package
[D: die form; F: QFP, B: BGA]

Model number

Model name
[C: microcomputer, digital products]

Product classification
[S1: semiconductor]

Development tools

S5U1 C 88348 D1 1 00

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 –

1 Overview

The S5U1C8F626Y4 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.

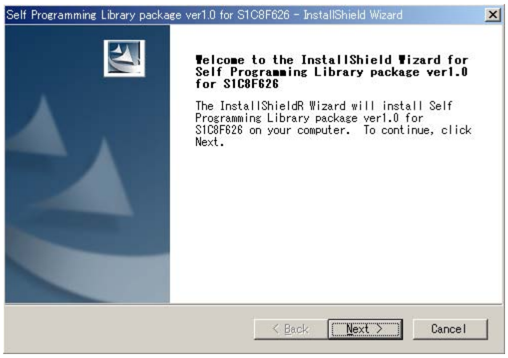
2.3 How to Install the Library

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.



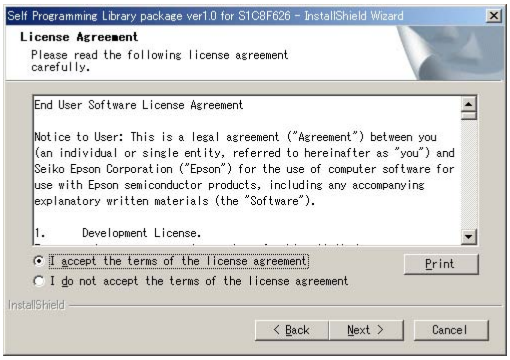
Self Programming Library package ver1.0 for S1C8F626 - InstallShield Wizard

Welcome to the InstallShield Wizard for Self Programming Library package ver1.0 for S1C8F626

The InstallShieldR Wizard will install Self Programming Library package ver1.0 for S1C8F626 on your computer. To continue, click Next.

You will see the install wizard start screen.

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



Self Programming Library package ver1.0 for S1C8F626 - InstallShield Wizard

License Agreement

Please read the following license agreement carefully.

End User Software License Agreement

Notice to User: This is a legal agreement ("Agreement") between you (an individual or single entity, referred to hereinafter as "you") and Seiko Epson Corporation ("Epson") for the use of computer software for use with Epson semiconductor products, including any accompanying explanatory written materials (the "Software").

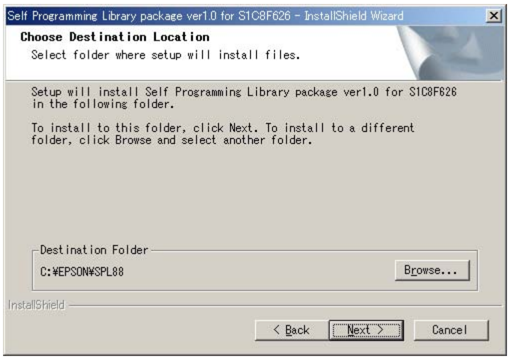
1. Development License.

● I accept the terms of the license agreement

○ I do not accept the terms of the license agreement

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.



Self Programming Library package ver1.0 for S1C8F626 - InstallShield Wizard

Choose Destination Location

Select folder where setup will install files.

Setup will install Self Programming Library package ver1.0 for S1C8F626 in the following folder.

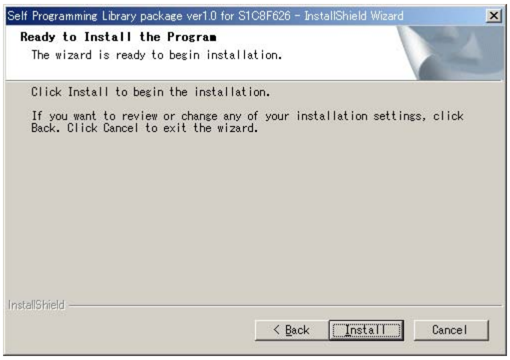
To install to this folder, click Next. To install to a different folder, click Browse and select another folder.

Destination Folder: C:¥EPSON¥SPL88

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.



Self Programming Library package ver1.0 for S1C8F626 - InstallShield Wizard

Ready to Install the Program

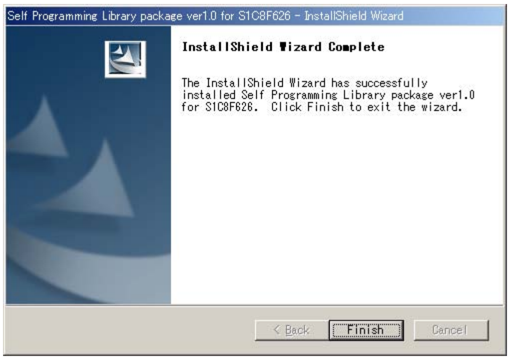
The wizard is ready to begin installation.

Click Install to begin the installation.

If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

This is the install start screen.

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



Self Programming Library package ver1.0 for S1C8F626 - InstallShield Wizard

InstallShield Wizard Complete

The InstallShield Wizard has successfully installed Self Programming Library package ver1.0 for S1C8F626. Click Finish to exit the wizard.

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		Library directory for large 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)
		\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

Size/Number of cycles		**Library memory model**	
		Small, compact code	**Compact data, large**
Object file size		834 bytes	981 bytes
Library work area size (RAM)	Stack	34 bytes	34 bytes
	Error structure	6 bytes	6 bytes
	Data buffer	Max. 4,096 bytes	Max. 4,096 bytes
Number of command processing cycles (per 1 sector)	Write	151,836 cycles	168,250 cycles
	Erase	227 cycles	249 cycles
	Verify	73,999 cycles	106,797 cycles
	Blank check	61,626 cycles	94,416 cycles

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.)

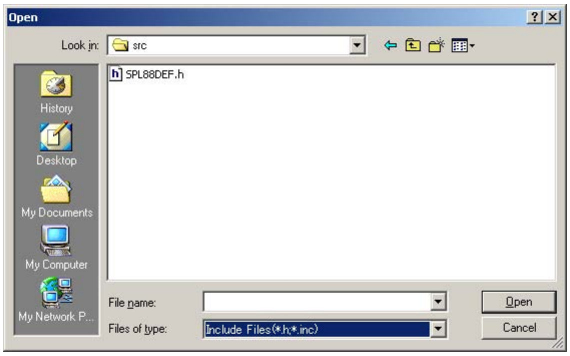
(2) Specifying the include file

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.

A dialog box for selecting an include file is displayed.



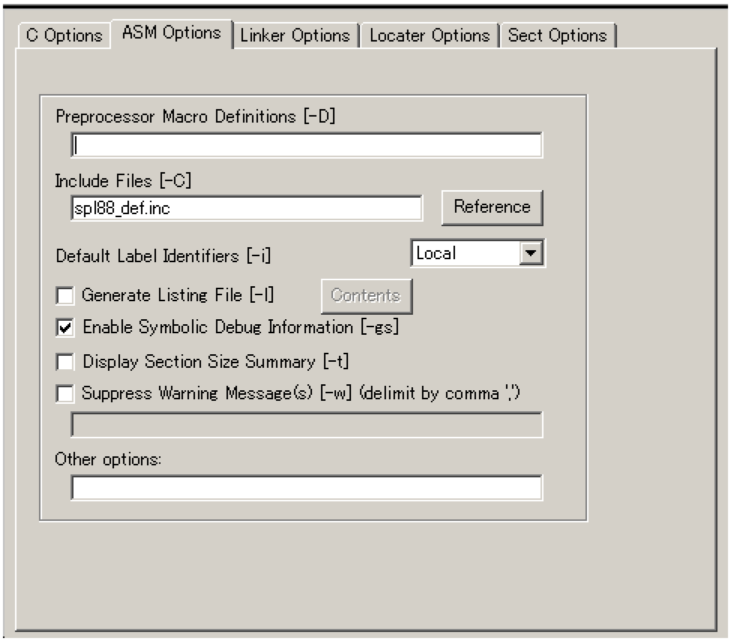
Look in: src

Files of type: Include Files(*.h;*.inc)

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.



Preprocessor Macro Definitions [-D]

Include Files [-C]: spl88_def.inc

Default Label Identifiers [-i]: Local

☐ Generate Listing File [-l]

☑ Enable Symbolic Debug Information [-gs]

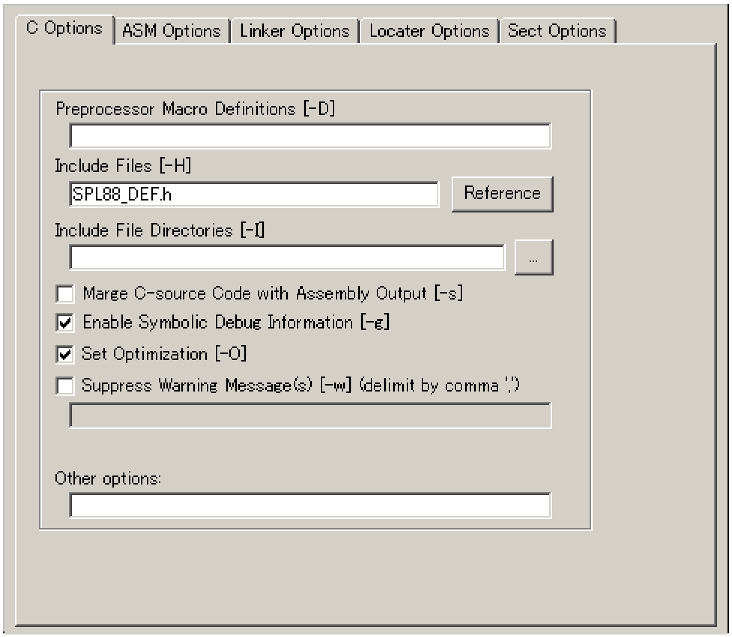
☐ Display Section Size Summary [-t]

☐ Suppress Warning Message(s) [-w] (delimit by comma ',')

Other options:

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.



Preprocessor Macro Definitions [-D]

Include Files [-H]: SPL88_DEF.h

Include File Directories [-I]

☐ Marge C-source Code with Assembly Output [-s]

☑ Enable Symbolic Debug Information [-g]

☑ Set Optimization [-O]

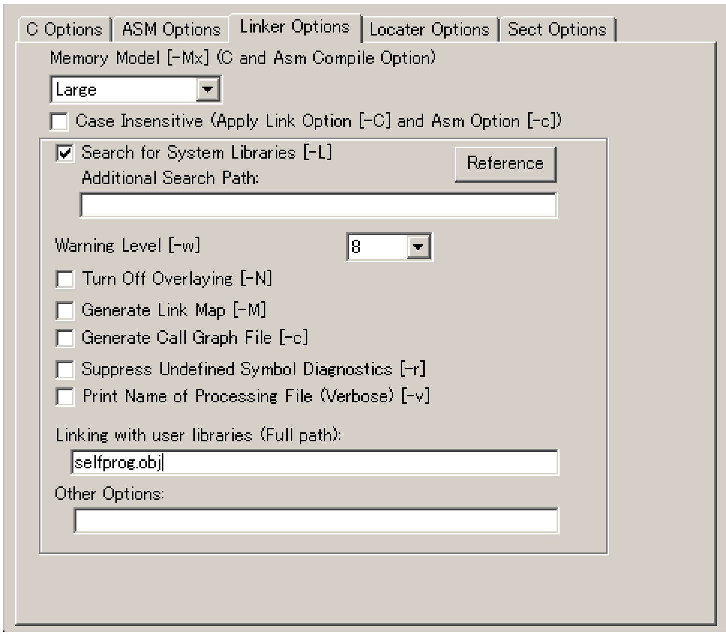
☐ Suppress Warning Message(s) [-w] (delimit by comma ',')

Other options:

(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.
3. Enter “selfprog.obj” in the [Linking with user libraries] field.



Memory Model [-Mx] (C and Asm Compile Option): Large

☐ Case Insensitive (Apply Link Option [-C] and Asm Option [-c])

☑ Search for System Libraries [-L]

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

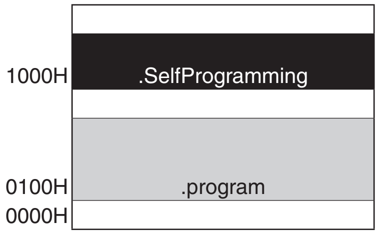
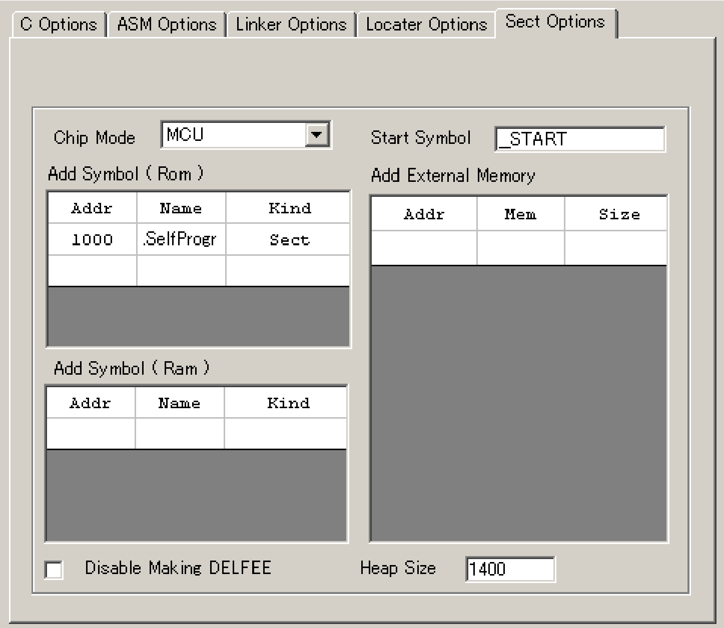


Figure 4.2.1 Example of Memory Layout

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.



Chip Mode: MCU

Start Symbol: _START

Add Symbol (Rom)

Addr	Name	Kind
1000	.SelfProgr	Sect

Add External Memory

Addr	Mem	Size

Add Symbol (Ram)

Addr	Name	Kind

☐ Disable Making DELFEE

Heap Size: 1400

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.

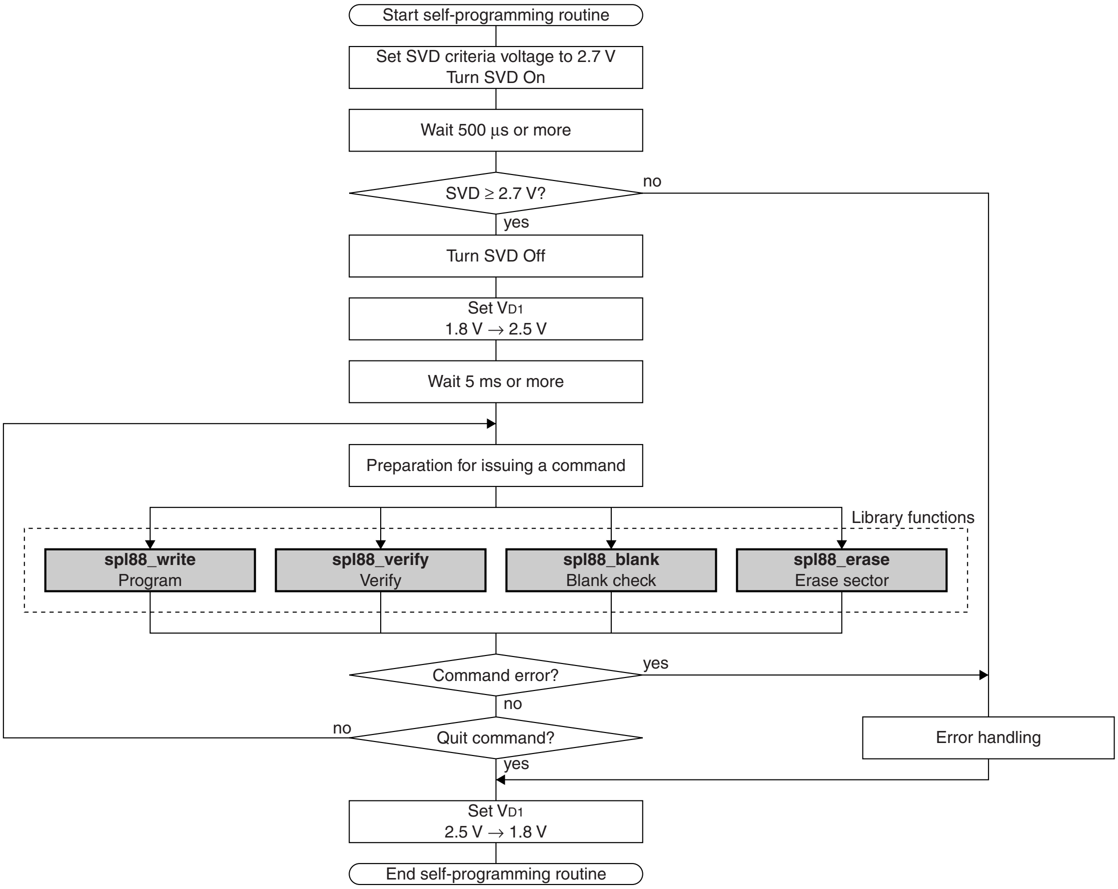


Figure 5.1.1 Self-Program Processing Flow

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.

```
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 5.4.1 List of Return Values from the Functions

Defined name	Value	Description
SPL88_ERR_NON	0	Terminated normally
SPL88_ERR_SCTNUM	1	Sector number error The specified sector number is 0CH, 0DH, 0EH, 0FH, or a 40H or more value.
SPL88_ERR_DATSIZ	2	Data size error The specified data size is 0 or a value more than 1000H.
SPL88_ERR_BLANK	3	Blank error An error has occurred in the blank check.
SPL88_ERR_VERIFY	4	Verify error An error has occurred in the verify check.
SPL88_ERR_VD1	5	V_{D1} error The currently set V_{D1} voltage is 1.8 V.

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 5.4.2 Default Values for Specifying Sectors

Defined name	Value	Description
SPL88_DAT_SIZ	1000H	Write or verify data size
SPL88_START_SCTNUM	4	Start sector number for erasing, blank check, writing or verification
SPL88_END_SCTNUM	5	End sector number for erasing, blank check, writing or verification

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 5.5.1 Compiler Memory Model

Compiler memory model	Code size	Data size	CPU mode	Bus mode
Small model	Code < 64K bytes	Data < 64K bytes	Minimum	Single chip mode (MCU) Extended 64K mode (MPU)
Compact code model	Code < 64K bytes	Data ≥ 64K bytes	Minimum	Extended 512K minimum mode
Compact data model	Code ≥ 64K bytes	Data < 64K bytes	Maximum	Extended 512K maximum mode
Large model	Code ≥ 64K bytes	Data ≥ 64K bytes	Maximum	Extended 512K maximum mode

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.

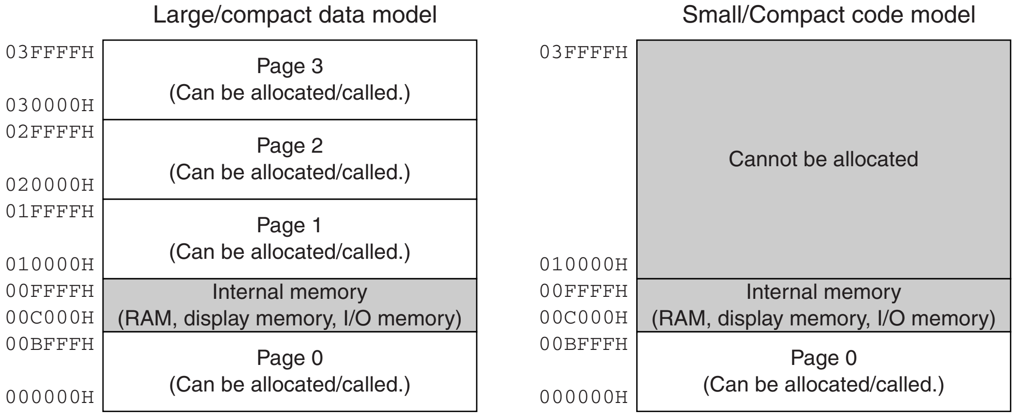


Figure 5.5.1 Library Allocatable Area

(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 5.5.2 Relationship between Sector Numbers and Addresses

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

* 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 `stat = spl88_erase(sectornum);`

Assembler `CARL _spl88_erase`

6.1 Erase Sector Function (spl88_erase)

Function	unsigned char spl88_erase(unsigned int sector_num);	
Description	Erases a specified sector in the S1C8F626 Flash EEPROM. While this function is being executed, the watchdog timer and all interrupts are disabled.	
Argument	BA register (unsigned int sector_num)	Sector number (see Table 5.5.2.) 00H–0BH, 10H–3FH
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_VD1: V_{D1} error
Output data	None	
Usage example	[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.
```

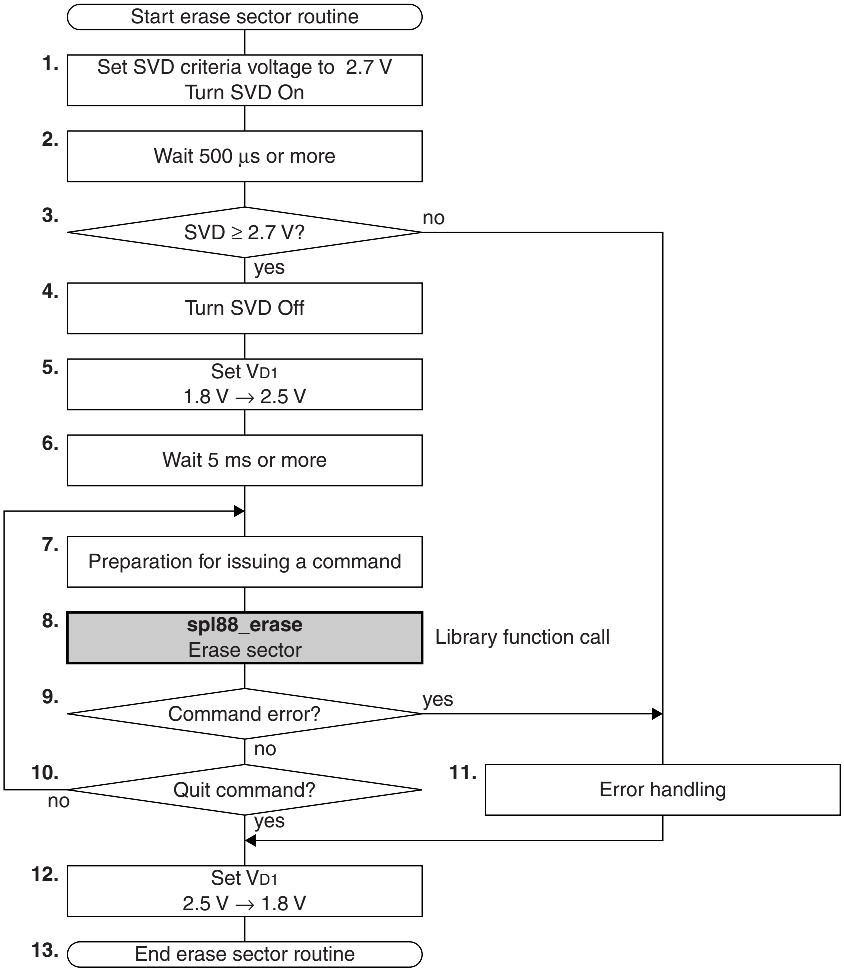
[C]

```
unsigned char stat;                 // Status (= A)
unsigned int sectornum;             // Sector No. (= BA)

sectornum = 0x1;                    // Sets the sector No. to be erased (sector No. = 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 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 V_{D1} voltage from 1.8 V to 2.5 V.
6. Wait 5 ms or more before calling the `spl88_erase` function after switching the V_{D1} 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 V_{D1} voltage from 2.5 V to 1.8 V.
13. Terminate the erase sector processing routine.

6.2 Program Function (spl88_write)

Function	unsigned char spl88_write(unsigned char* pdata, unsigned int sector_num, unsigned int size);	
Description	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.	
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: V_{D1} error
Output data	None	
Usage example	[ASM] / [C]	

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.
```

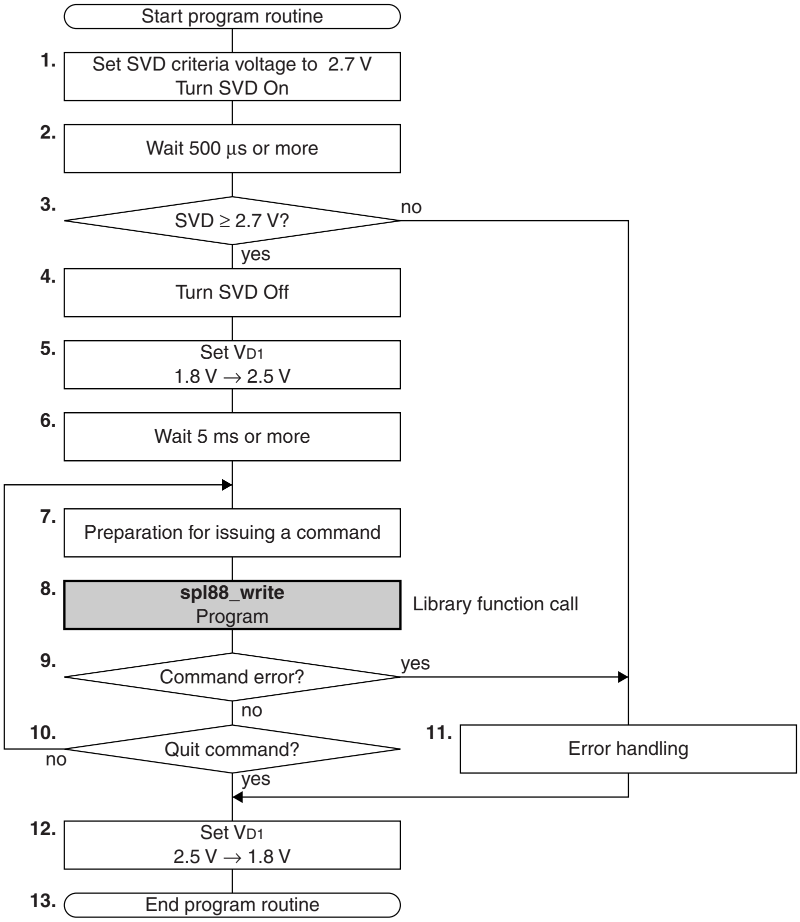
[C]

```
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.
  ...                                               //  Sets data to the 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.
  ...                                               //  Error handling
}
free(pdat);                                         //  Deallocates the write data area.
```

Program processing flow

The following shows a procedure to write data.



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 V_{D1} voltage from 1.8 V to 2.5 V.
6. Wait 5 ms or more before calling the `spl88_write` function after switching the V_{D1} 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 V_{D1} voltage from 2.5 V to 1.8 V.
13. Terminate the erase sector processing routine.

6.3 Verify Function (spl88_verify)

Function	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	YP and IY registers (unsigned char* pdata)	Pointer to the original data for comparison (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)	Verification size 1–4096
	IX register (spl88_err_str* pSpl88_err_str)	Pointer to the error structure (spl88_err_str)
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_VERIFY: Verify error SPL88_ERR_VD1: V_{D1} error
Output data	(unsigned long) spl88_err_str.spl88_err_adr	Address where an error has occurred (Flash EEPROM)
	(unsigned char) spl88_err_str.spl88_org_dat	Original data for comparison (RAM)
	(unsigned char) spl88_err_str.spl88_err_dat	Data in which an error has occurred (Flash EEPROM)

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).
CARL  _spl88_verify            ; Calls the verify function.
CP    A,#000H                  ; Checks the status bits.
POP   IX
```

Usage example

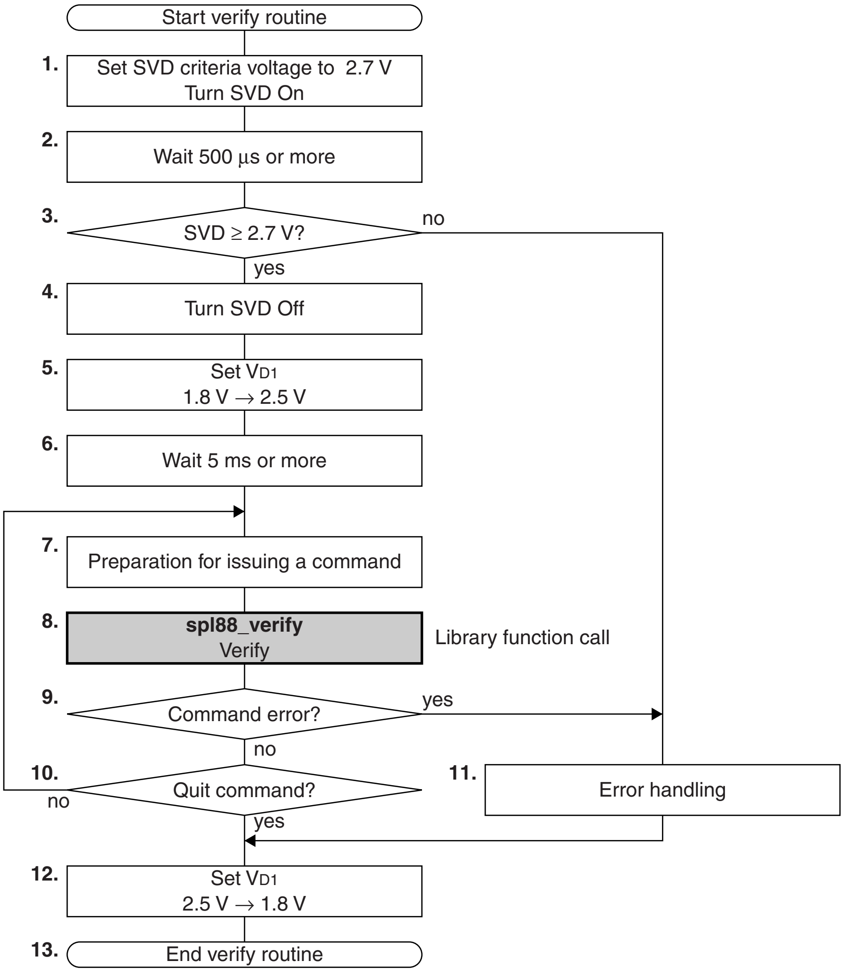
[C]

```
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 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 V_{D1} voltage from 1.8 V to 2.5 V.
6. Wait 5 ms or more before calling the `spl88_verify` function after switching the V_{D1} 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 V_{D1} voltage from 2.5 V to 1.8 V.
13. Terminate the erase sector processing routine.

6.4 Blank Check Function (spl88_blank)

Function	unsigned char spl88_blank(unsigned int sector_num, spl88_err_str* pSpl88_err_str);	
Description	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.	
Argument	BA register (unsigned int sector_num)	Sector number (see Table 5.5.2.) 00H–0BH, 10H–3FH
	IY register (spl88_err_str* pSpl88_err_str)	Pointer to the error structure (spl88_err_str)
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_BLANK: Blank error SPL88_ERR_VD1: V_{D1} error
Output data	(unsigned long) spl88_err_str.spl88_err_adr	Address where an error has occurred (Flash EEPROM)
	(unsigned char) spl88_err_str.spl88_org_dat	Original data (0FFH)
	(unsigned char) spl88_err_str.spl88_err_dat	Data in which an error has occurred (Flash EEPROM)

Usage example

[ASM]

```
LD    YP,#@DPAG(spl88_err_str) ; Sets the page address of the structure pointer.
LD    IY,#@DOFF(spl88_err_str) ; Sets the address of the structure pointer.
PUSH  IY
LD    IY,SP                    ; Stack pointer (structure pointer on the stack)
LD    BA,#001H                 ; Sets the blank check sector No. (BA). Sector No. = 1
CARL  _spl88_blank             ; Calls the blank check function.
CP    A,#000H                  ; Checks the status bits.
POP   IY
```

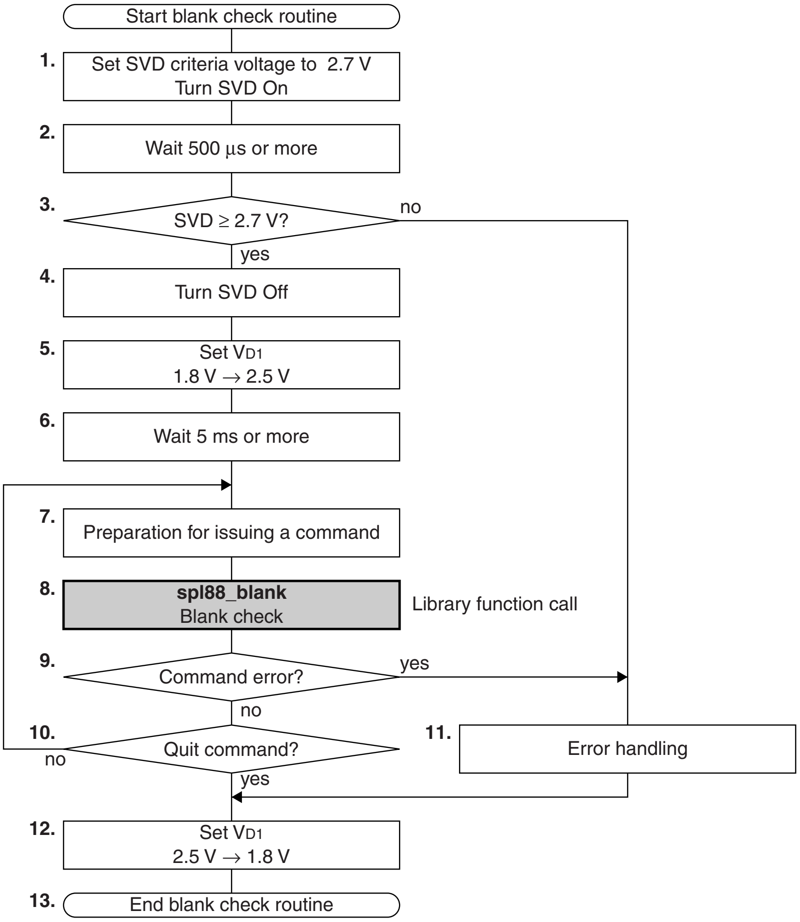
[C]

```
unsigned char stat;                     // Status (= A)
unsigned int sectornum;                 // Blank check sector No. (= BA)
spl88_err_str* pSpl88errstr;            // Error structure (= IY)

pSpl88errstr = (spl88_err_str*) malloc(sizeof(spl88_err_str));
                                        // Allocates an area for the error structure.
sectornum = 0x1;                        // Sets the blank check sector No. (1).
stat = spl88_blank(sectornum, (spl88_err_str*) &pSpl88errstr);
                                        // Calls the blank check function.
if(stat != 0){                          // Checks the status bits.
  ...                                   // Error handling
}
free(pSpl88errstr);                     // Deallocates the error structure area.
```

Blank check processing flow

The following shows a procedure for blank check.



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 V_{D1} voltage from 1.8 V to 2.5 V.
6. Wait 5 ms or more before calling the `spl88_blank` function after switching the V_{D1} voltage.
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 V_{D1} 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

Compiler memory model	CPU mode	
	Minimum mode	Maximum mode
Small model	○	×
Compact code model	○	×
Compact data model	×	○
Large model	×	○

(○: 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 V_{D1} 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_{D1} 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
```

\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

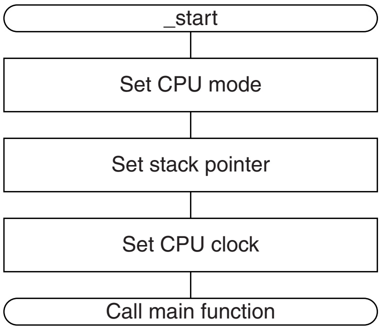
A.2 Functions in the Sample Program

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



Small or compact code model
Set the CPU mode to minimum mode.
Compact data or large model
Set the CPU mode to maximum mode.

Select OSC3 as the CPU clock.

Figure A.2.1.1 _start Flowchart

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 V_{D1}. 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 V_{D1} to 2.5 V for Flash programming. After waiting for stabilization of the V_{D1} 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-programming 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

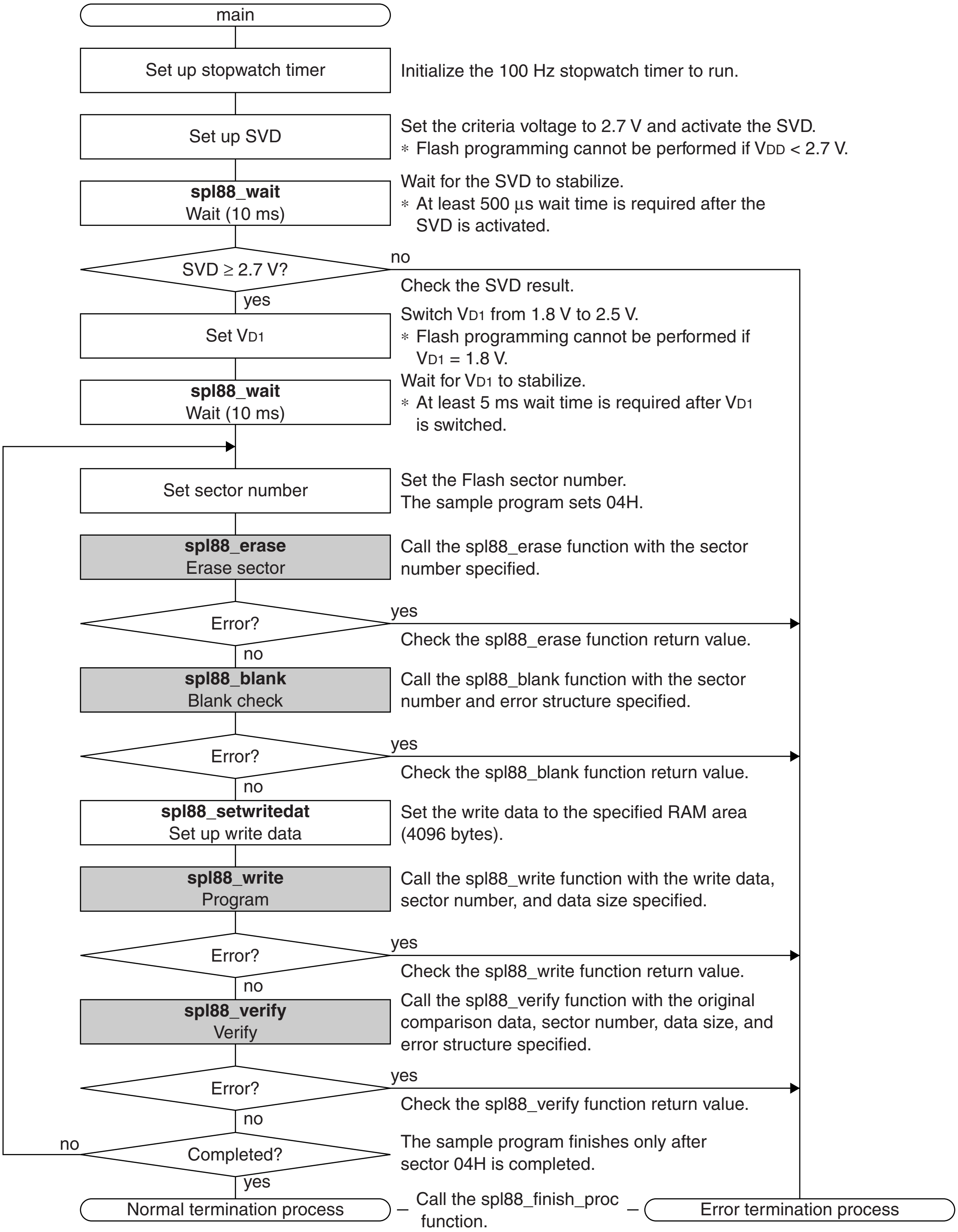
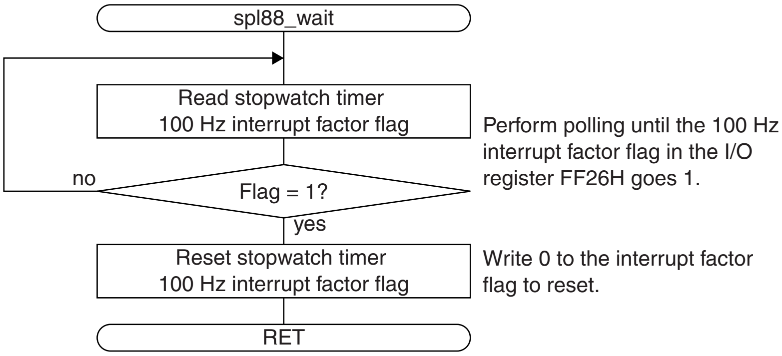


Figure A.2.2.1 main Flowchart

A.2.3 spl88_wait (Wait Function)

Function	void spl88_wait(void);
Description	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.
Arguments	None
Return value	None
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 V_{D1} is switched. Generate appropriate wait times using a method possible in the application system.



Perform polling until the 100 Hz interrupt factor flag in the I/O register FF26H goes 1.

Write 0 to the interrupt factor flag to reset.

Figure A.2.3.1 spl88_wait Flowchart

A.2.4 spl88_setwritedat (Write Data Setup Function)

Function	void spl88_setwritedat(unsigned char* spl88_rxp_dat, unsigned int sectornum);	
Description	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.	
Arguments	YP and IY registers (unsigned char* spl88_rxp_dat)	Pointer to data buffer (RAM) YP: one high-order byte of address IY: two low-order bytes of address
	BA register (unsigned int sectornum)	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.	

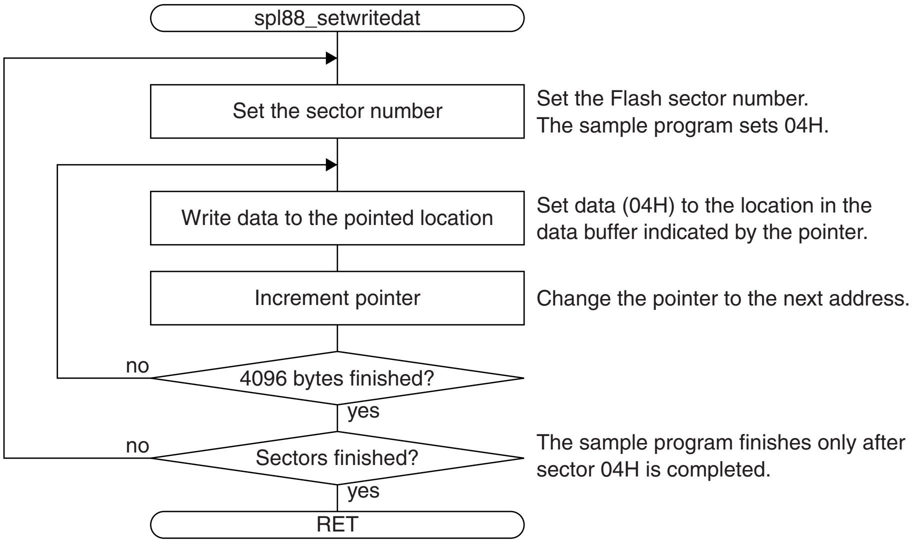


Figure A.2.4.1 spl88_setwritedat Flowchart

A.2.5 spl88_finish_proc (Termination Process Function)

Function	void spl88_finish_proc(unsigned char* spl88_rxp_dat, spl88_err_str* pSpl88_err_str);	
Description	This function performs processing for termination after the library functions are executed. It disables SVD and returns V_{D1} 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.	
Arguments	YP and IY registers (unsigned char* spl88_rxp_dat)	Pointer to data buffer (RAM) YP: one high-order byte of address IY: two low-order bytes of address
	XP and IX registers (spl88_err_str* pSpl88_err_str)	Pointer to the error structure (spl88_err_str) XP: one high-order byte of address IX: two low-order bytes of address
Return value	None	

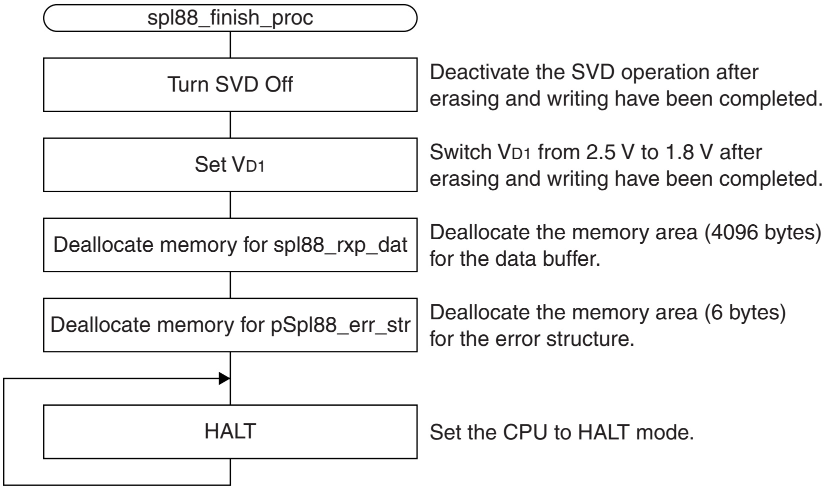


Figure A.2.5.1 spl88_finish_proc Flowchart

EPSON

International Sales Operations

AMERICA

EPSON ELECTRONICS AMERICA, INC.

HEADQUARTERS
2580 Orchard Parkway
San Jose, CA 95131, U.S.A.
Phone: +1-800-228-3964 Fax: +1-408-922-0238

SALES OFFICE
Northeast
301 Edgewater Place, Suite 210
Wakefield, MA 01880, U.S.A.
Phone: +1-800-922-7667 Fax: +1-781-246-5443

EUROPE

EPSON EUROPE ELECTRONICS GmbH

HEADQUARTERS
Riesstrasse 15
80992 Munich, GERMANY
Phone: +49-89-14005-0 Fax: +49-89-14005-110

ASIA

EPSON (CHINA) CO., LTD.
23F, Beijing Silver Tower 2# North RD DongSanHuan
ChaoYang District, Beijing, CHINA
Phone: +86-10-6410-6655 Fax: +86-10-6410-7320

SHANGHAI BRANCH
7F, High-Tech Bldg., 900, Yishan Road
Shanghai 200233, CHINA
Phone: +86-21-5423-5522 Fax: +86-21-5423-5512

EPSON HONG KONG LTD.
20/F, Harbour Centre, 25 Harbour Road
Wanchai, Hong Kong
Phone: +852-2585-4600 Fax: +852-2827-4346
Telex: 65542 EPSCO HX

EPSON Electronic Technology Development (Shenzhen) LTD.
12/F, Dawning Mansion, Keji South 12th Road
Hi- Tech Park, Shenzhen
Phone: +86-755-2699-3828 Fax: +86-755-2699-3838

EPSON TAIWAN TECHNOLOGY & TRADING LTD.
14F, No. 7, Song Ren Road
Taipei 110
Phone: +886-2-8786-6688 Fax: +886-2-8786-6660

EPSON SINGAPORE PTE., LTD.
1 HarbourFront Place
#03-02 HarbourFront Tower One, Singapore 098633
Phone: +65-6586-5500 Fax: +65-6271-3182

SEIKO EPSON CORPORATION
KOREA OFFICE
50F, KLI 63 Bldg., 60 Yoido-dong
Youngdeungpo-Ku, Seoul, 150-763, KOREA
Phone: +82-2-784-6027 Fax: +82-2-767-3677

GUMI OFFICE
2F, Grand B/D, 457-4 Songjeong-dong
Gumi-City, KOREA
Phone: +82-54-454-6027 Fax: +82-54-454-6093

SEIKO EPSON CORPORATION
SEMICONDUCTOR OPERATIONS DIVISION

IC Sales Dept.
IC International Sales Group
421-8, Hino, Hino-shi, Tokyo 191-8501, JAPAN
Phone: +81-42-587-5814 Fax: +81-42-587-5117

S5U1C8F626Y4 Manual
(S1C8F626 Self-Programming Library)

SEIKO EPSON CORPORATION
SEMICONDUCTOR OPERATIONS DIVISION

■ **EPSON Electronic Devices Website**

http://www.epson.jp/device/semicon_e

Document code: 411090300
Issue July, 2007
Printed in Japan Ⓛ A