NEC

Application Note

Self-Programming

32-/16-bit Single-Chip Microcontroller

Self-Programming Library for embedded Dual Power FLASH

Document No. U15352EE3V0AN00
Date Published June 2003

NOTES FOR CMOS DEVICES

① PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

Note:

Strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it once, when it has occurred. Environmental control must be adequate. When it is dry, humidifier should be used. It is recommended to avoid using insulators that easily build static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work bench and floor should be grounded. The operator should be grounded using wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with semiconductor devices on it.

② HANDLING OF UNUSED INPUT PINS FOR CMOS

Note:

No connection for CMOS device inputs can be cause of malfunction. If no connection is provided to the input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND with a resistor, if it is considered to have a possibility of being an output pin. All handling related to the unused pins must be judged device by device and related specifications governing the devices.

③ STATUS BEFORE INITIALIZATION OF MOS DEVICES

Note:

Power-on does not necessarily define initial status of MOS device. Production process of MOS does not define the initial operation status of the device. Immediately after the power source is turned ON, the devices with reset function have not yet been initialized. Hence, power-on does not guarantee out-pin levels, I/O settings or contents of registers. Device is not initialized until the reset signal is received. Reset operation must be executed immediately after power-on for devices having reset function.

MS-DOS and MS-Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
PC/AT and PC DOS are trademarks of IBM Corp.

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

The export of this product from Japan is regulated by the Japanese government. To export this product may be prohibited without governmental license, the need for which must be judged by th customer. The export or re-export of this product from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

M5 2000.03

Regional Information

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

J99.1

Chapter 1 Introduction

NEC Flash devices may have different basic Flash technologies. Two major options have to be considered. These options differ regarding the Flash features and regarding self-programming:

- Dual voltage Flash
  This Flash needs a dedicated Flash programming voltage (usually called Vpp). Dual voltage Flash is covered by this document.

- Single voltage Flash
  No dedicated programming voltage is required. A similar document covering self-programming of single voltage Flash is available on the internet too.

A ***SelfLib Device Reference List*** is available on the internet. This list contains information about the Flash devices, Flash technology, library versions, supported special functions, development environments and operating conditions.
Please check this list in order to get the correct library and application note.

All necessary information including self-programming libraries, application notes and the reference list can be downloaded from the following URL:
http://www.nec.de/Products/Micro/Tools/Download/V850/Index.html

Chapter 2 Overview

NEC Flash devices provide two different methods to program the on-chip Flash:

- Programming the Flash using the Flash programmer interface
  The programmer interface is used to program the device without any user software using a standardized programming tool (e.g. NEC flashMASTER).

- Self-Programming
  Self-Programming allows the user to define an own interface for data transmission, i.e. CAN, and reprogram the device using a specific self-programming library.

This document describes the usage of the self-programming libraries, also called ***SelfLib***.

As the libraries use device internal functions and hardware, they are individually made for one device and cannot be used for any other device.

All required functions, including erase, blank check, write and verify, are part of the libraries. Furthermore special functions like for secure reprogramming are part of the libraries when supported by the device.

The libraries are pre-compiled and therefore linked to the development environment. Basically the development environments of the companies Green Hills (GHS) **Note** and IAR **Note** can be supported.

Note: All other product, brand, or trade names used in this publication are the trademarks or registered trademarks of their respective trademark owners.

The libraries contain functions, which directly operate on the Flash (Write, Erase,...). These functions are called ***Flash Access Functions*** in the following.
During execution of these functions, the device is switched into the self-programming mode. In this mode no instruction or data fetch from the Flash is possible. So these functions have to be executed outside the internal Flash.

A *SelfLib*, which can be downloaded from the internet, contains the following elements:

- SelfLib.a
  This is the library module which has to be linked to the user program

- SelfLib.h
  This is the header file with the function (pointer) prototypes

- DemoSelfLib
  This directory contains a short sample program. It does not contain a correct reprogramming sequence but shows the usage of the library functions.

2.1 Library Handling

2.1.1 Peripherals

Some peripherals are used by the library. Please refer to the *SelfLib Device Reference List*. After initialization of the library the setup of these peripherals may not be changed as long as library functions are used.
Furthermore the original setup of the peripherals is not restored automatically after using the library. This has to be done by the user program.

2.1.2 Interrupts

User interrupt functions cannot be served during execution of the *Flash Access Functions* as the interrupt entry address (In the Flash) is not accessible at that time.
Nevertheless some interrupts are internally used, e.g. as a time base. Therefore the concerning interrupts are unmasked and globally enabled (The ID Flag in the PSW of the device is cleared).
To prevent user interrupts from being acknowledged at that time, please mask them before calling any *Flash Access Function*.

After leaving a *Flash Access Function* the ID flag is set to globally disable the interrupts. In order to prevent unexpected acknowledgment of invalid interrupts, the setup of the peripherals used by the *Flash Access Functions* including the interrupt registers has to be restored before clearing the ID flag in the user program again.
Please refer to the *SelfLib Device Reference List* for the used peripherals.

As some of the *Flash Access Functions* require a long execution time (up to more than 20s for the erase function, depending on the device), real time operations (Like Watchdog handling) cannot be realized by interrupt functions. Therefore, libraries of some devices support watchdog handler routines, which can poll an interrupt flag and serve short handler routines (See “Watchdog-Handling” on page 27).

2.1.3 Code Sections

The library functions are placed into separate sections, that will be handled differently:

SelfLib_ROM
This section contains functions which will be used in the initialization phase of the library only and therefore may be executed in the internal Flash.
These functions are called directly (All functions called by SelfLib_..., see “Library Functions” on page 32) and need no special handling.

SelfLib_ToRAM
This section contains the *Flash Access Functions*.
Two different possibilities have to be considered:

- The section is located in the internal Flash. This is the most common option.
  Before execution of any of these functions the section needs to be copied to internal or external RAM.
  This is done automatically by the library initialization when requested (See “Library Initialization” on page 32).

- The section is located anywhere else but the internal Flash.
  The functions can be executed on this location so that the section need not to be copied.

The section may not only contain the *SelfLib* functions but also user functions may be placed there. In the case (1) they are copied with the complete section to the new destination.

As the location of the SelfLib_ToRAM section may change, the functions in this section cannot directly be called.
Therefore the library contains declarations of function pointers to these functions. The pointers are set to the correct address in the initialization phase (“Library Initialization” on page 32). Library functions which are called by the pointers begin with fpSelfLib_... (See “Library Functions” on page 32, fp means “function pointer”).

When the SelfLib_ToRAM section location changes, it will be necessary to call user functions located in this section by function pointers, too. To do so the user program has to declare function pointers. To set the pointers to the correct new address, the library provides a function to calculate the new address (“New Function Address” on page 34).
Calculation of the required RAM space for copying the section SelfLib_ToRAM is easy, because the size is calculated by the linker and mentioned in the map file after linking the project.

SelfLib_RAM
This Section contains the global variables and function pointers used by the *SelfLib*. It has to be linked to an internal or external RAM location

2.1.4 Function Handling

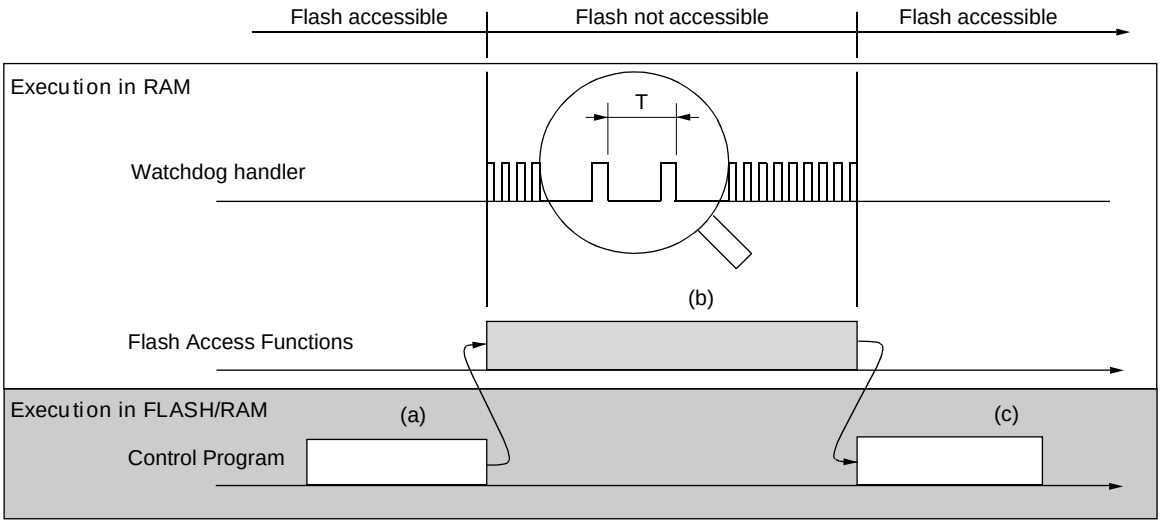
When no external memory is available, the easiest way to handle self-programming is to place the complete reprogramming program including reprogramming control functions and communication functions into the SelfLib_ToRAM section.
Then, all functions are copied into the internal RAM (if necessary, see “Library Handling” on page 19). After calling a control function (user program) the execution is done entirely in the RAM.

Furthermore it is possible to access the Flash outside the *Flash Access Functions*. The control program and the communication functions can also be located in a Flash area that is not reprogrammed and only the SelfLib_ToRAM section must be located in the RAM or external memory.

If an external watchdog is being used, a watchdog handler is called by the *Flash Access Functions*. It must still be available, even when the Flash is not accessible. Therefore the SelfLib_ToRAM section is the best section to place the handler function.

When any user functions are placed in the SelfLib_ToRAM section and this section is copied to a different location, it is necessary that the functions are compiled with special options to avoid absolute function calls (See “Compiler Options” on page 21).

Figure 2-1: Flash Library Call Sequence



2.1.5 Compiler Options

For correct operation of the *SelfLib* and all functions which are in the SelfLib_ToRAM section (When copied into the RAM) certain compiler options are necessary to avoid absolute jumps or calls of library functions. The following table lists up the compiler options used for the SelfLib_ToRAM section.

Table 2-1: Special Compiler Options (Command Line or IDE options alternatively)

IDE	Command Line Options	IDE Options
GHS	-pic -inline_prologue	Project -> CPU Options -> Position Independent Code -> '+' Project -> CPU Options -> Inline Prologue -> '+'
IAR		Project -> Options -> ICCV850 -> Code -> Function Inlining -> check

Notes: 1. Projects comiled with GHS may have code and memory models differing from the library. IAR projects need the same models for all files. Please choose the memory model of the project according to the library. Please refer to the *Device Reference List* for the libraries memory model.

2. IAR tools sometimes use small IAR library functions to shorten the code. As the SelfLib_ToRAM section is normally copied into the internal RAM or external memory it is not possible to call these library functions from there.
Although the SelfLib part itself does not call any library functions it might happen, that a part of the user control program in the SelfLib_ToRam section does so. So please check carefully all user code parts in the SelfLib_ToRAM section for those calls (e.g. using the debugger).

2.2 Reprogramming the Device

Please refer to the *SelfLib Device Reference List* to check, if your device supports secure reprogramming and which sequence you should use.
If the device does not support secure reprogramming, you may reprogram the Flash areas according to the chapter “Area Reprogramming” on page 22.
If your device supports secure reprogramming you will embed the area reprogramming sequence into the sequence according to the chapter “Secure Device Reprogramming - 1” on page 22 or “Secure Device Reprogramming - 2” on page 24.

The following principle steps have to be executed:

Table 2-2: Basic Steps

Step	Function	Description
1	Setup the user program	Setup communication Interface, watchdog timer, Vpp supply, ...
2	Switch Vpp on	Vpp has to be applied to the Vpp pin (External hardware controlled by a user function)
3	SelfLib_Init	*SelfLib* (And all code in the SelfLib_ToRAM section) is copied into the device RAM. *SelfLib* is initialized
4	Get new function addresses(s)	Use SelfLib_FktRamAddress function to get the new address of the functions in the SelfLib_ToRAM Section which have to be called from the user program in the Flash (E.g. A Self-Programming control function)
5	Reprogram the Flash	Take care to keep the correct sequence for reprogramming single Flash areas (“Area Reprogramming” on page 22) and for secure Flash reprogramming (“Secure Device Reprogramming - 1” on page 22, “Secure Device Reprogramming - 2” on page 24)
6	Switch Vpp off	Vpp has to be switched off (External hardware controlled by a user function)

2.2.1 Area Reprogramming

To ensure, that the data is written correctly and the data retention is given it is important to keep the following sequence for area reprogramming. Whenever a function returns an error, you may not continue the sequence because the next steps may not work correctly and the result is undefined.

Table 2-3: Area reprogramming sequence

Step	Flash Lib. Function	Description
1	Blank Check	Check, if the area is blank. If yes, step 2 can be skipped
2	Area Erase	The Flash is erased
3	Write	Data is written to the Flash
4	Internal Verify	Internal comparison of the data on the read level and on verify level. This step is **necessary** for the specified data retention

2.2.2 Secure Device Reprogramming - 1

The basic device feature allowing this way of secure reprogramming is that on device start-up the internal firmware checks, if the upper area contains a valid signature on a certain address. Please refer to *SelfLib Device Reference List*.

If a signature could be found, the firmware jumps on the first address of the upper area. If the signature could not be found, the firmware jumps to the address 0x00000000.

By this feature and the correct reprogramming sequence (see below) it can be assured, that always a valid boot program is started to continue the reprogramming sequence in case that the reprogramming sequence was interrupted (Power fail, accidental reset).

The boot program (Also called Boot loader or boot block) must be able to control the complete reprogramming sequence including data transfer of the new Flash contents. To do so it has to contain the sections SelfLib_ROM and SelfLib_ToRAM.
The size is limited by the area size and the rest of the application program only.

Table 2-4: Secure Reprogramming Sequence - 1 (1/2)

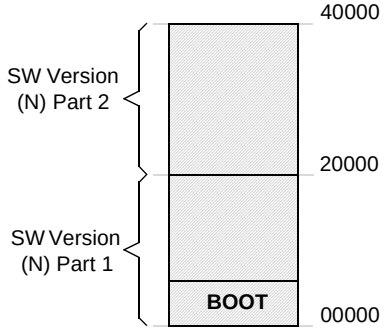
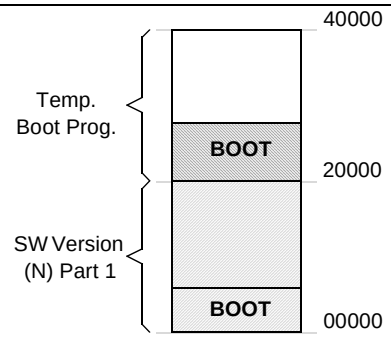
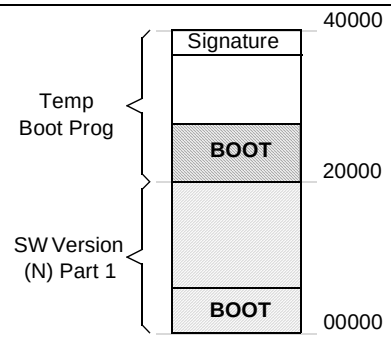
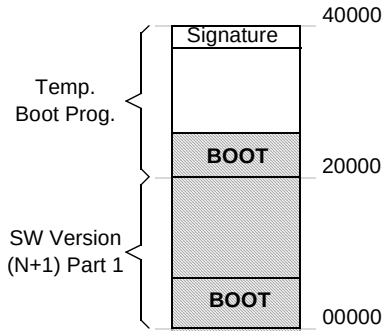
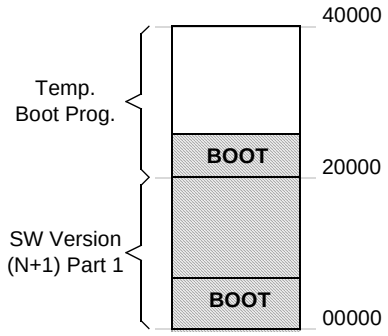
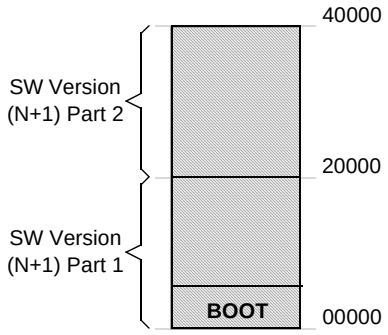
Step	Flash Lib. Function	Description	Sample Configuration (2 areas with 128kByte each)
1	Initial Status	Boot Program in the lower area	
2	Reprogram the upper area	A boot loader needs to be programmed into the upper area. This boot loader is only temporary (Use the sequence according to “Area Reprogramming” on page 22)	
3	Write the signature to the upper area	Writing the signature to the upper area ensures, that the device will restart in the upper area in case of a power failure or accidental reset. After writing the signature, the lower area may safely be erased because the upper area contains a valid boot program	

Table 2-4: Secure Reprogramming Sequence - 1 (2/2)

Step	Flash Lib. Function	Description	Sample Configuration (2 areas with 128kByte each)
4	Reprogram the lower area	The new code is being written into the lower area. It must contain the new boot program and interrupt table (Use the sequence according to “Area Reprogramming” on page 22).	
5	Delete the signature	The signature has to be deleted before reprogramming the upper area. Otherwise it might happen that when reprogramming the upper block fails (Power fail, accidental reset) the signature is still valid although the data is partly changed. Then the device starts in a partly reprogrammed upper area. After deleting the signature the device would start in the correct programmed lower area.	
6	Reprogram the upper area	The new code is being written into the upper area (Use the sequence according to “Area Reprogramming” on page 22)	

2.2.3 Secure Device Reprogramming - 2

The basic feature allowing this way of secure reprogramming is, that on device start-up the internal firmware checks, which of two areas contains a valid signature. Please refer to the *SelfLib Device Reference List*.

In case that the upper area contains a valid signature and the lower area not, the areas are swapped before starting the user program. Then the upper area is addressed from 0x00000000 onwards and the lower area can be accessed by using the original addresses of the upper area.
By this feature and the correct reprogramming sequence (see below) it can be assured, that always a valid boot program is started to continue the reprogramming sequence in case that the reprogramming algorithm was interrupted (Power fail, accidental reset).

The boot program (Also called Boot loader or boot block) must be able to control the complete reprogramming sequence including data transfer of the new Flash contents. To do so it has to contain the sections SelfLib_ROM and SelfLib_ToRAM.
The size is limited by the area size and the rest of the application program only.

Table 2-5: Secure Reprogramming Sequence - 2 (1/2)

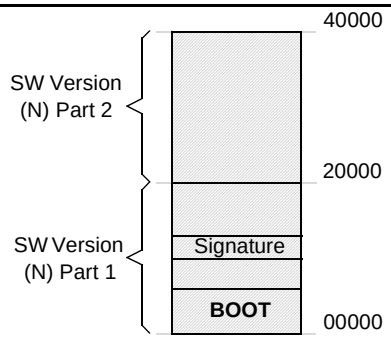
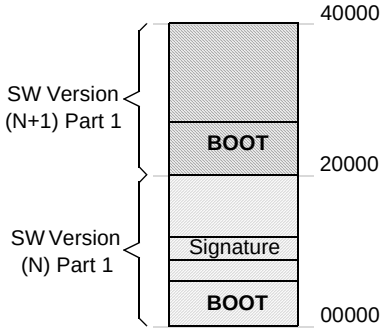
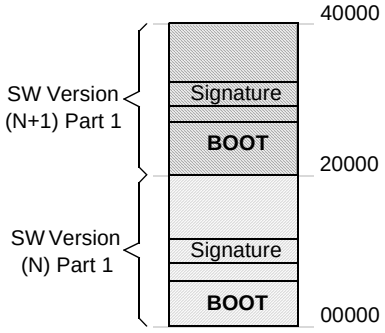
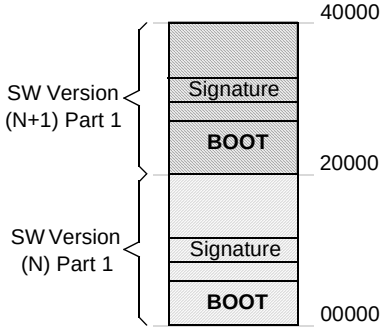
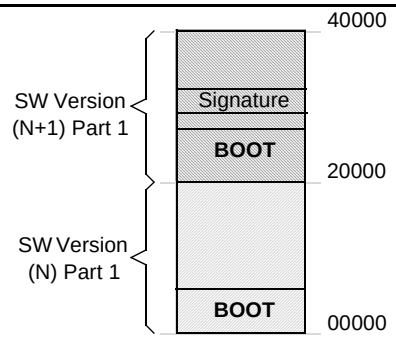
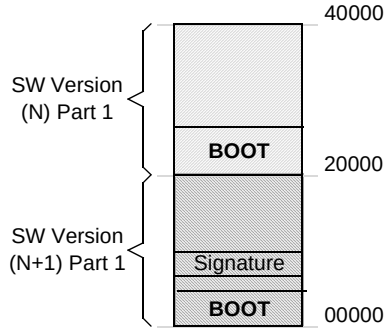
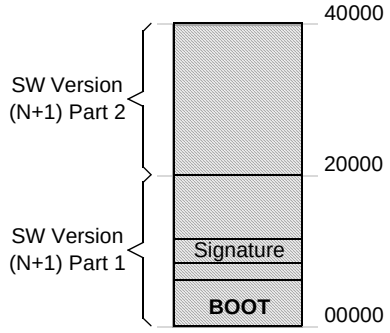
Step	Flash Lib. Function	Description	Sample Configuration (2 areas with 128kByte each)
1	Initial Status	Boot Program in the lower area	
2	Reprogram the upper area	The programmed data shall contain the code which shall be located on the address 0x0000000 later on. So the boot loader and interrupt table will be in this part of the code (Use the sequence according to “Area Reprogramming” on page 22, but the internal verify may be skipped as this is done in step 4)	
3	Write the signature to the upper area	Writing the signature to the upper area is the first step to ensure that the upper area will be swapped to the address 0x0000000 in case of a power failure or accidental reset.	
4	Internal verify on upper area	After writing data (The signature is treated as data too) in the upper area an internal verify on this area has to be executed to ensure the data retention.	

Table 2-5: Secure Reprogramming Sequence - 2 (2/2)

Step	Flash Lib. Function	Description	Sample Configuration (2 areas with 128kByte each)
5	Delete the signature in the lower area	Deleting the signature to the lower area is the second step to ensure that the upper area will be swapped to the address 0x0000000 in case of a power failure or reset. The lower area may now safely be reprogrammed.	
6	Swap areas	After swapping the areas the program code in the lower area is at the correct address and can be executed. Either a hardware reset or the area swap function (See “Swap Area” on page 38) can be used to swap the areas.	
7	Reprogram the upper area	The upper area is being written. (Use the sequence according to “Area Reprogramming” on page 22)	

2.3 Watchdog-Handling

Please check in the *SelfLib Device Reference List* whether the handling/triggering of an external watchdog (WD) is supported by the library or not.
To use this feature a timer can be set up in the user program. Instead of the timer any other internal or external hardware generating periodical interrupts of the correct frequency can be used.
The timer interrupt (or other interrupt) register address and the function pointer to the WD handler is passed to the library init function (“Library Initialization” on page 32). The library monitors the interrupt request flag. When set, it will be reset and a WD handler function will be called.

Note: When SelfLib_ToRAM is copied into the RAM, the WD handler function should be located in the SelfLib_ToRAM to be copied there too and to be able to pass the function address directly without any modification (The address offset between SelfLib and WD handler will be the same before and after copying into the RAM).

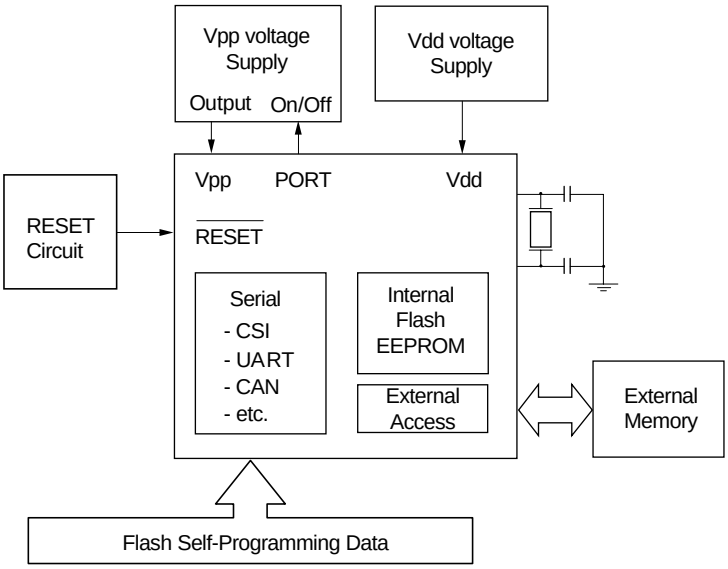
2.4 Used Rescources

2.4.4 External

The device requires a correct programming voltage (Vpp) which has to be provided by the application. So please take care for the correct programming voltage according the device specification.
Furthermore the voltage may only be applied to the Vpp pin during reprogramming and therefore needs to be switched on and off by the device.
Especially on device start-up Vpp must be off. Otherwise it will not start in the normal operation mode and will not execute any user program.

A communication interface is needed to transmit the new program code for reprogramming. This may be any serial or other interface as the reprogramming control program including the communication interface handling is not part of the *SelfLib* and therefore has to be programmed by the user.

Figure 2-2: Hardware Requirements Overview



2.4.5 Device Internal

During the reprogramming procedure the *SelfLib* is copied to the internal RAM and executed there. During execution, following items are required:

- Internal or external RAM for the *SelfLib* code (If copying the library into the RAM is requested). The size is equal to the size of the section SelfLib_ToRAM
- Less than 100 bytes up to 240 bytes of stack, depending on the library.
- 70~170 Bytes internal or external RAM for the global library variables (SelfLib_RAM section), the size is depending on the library version
- All general-purpose registers are used.
- A timer as timebase for the *SelfLib* (Device internal firmware and library dependent)
- A Timer (And Timer interrupt) for the watchdog handling time base (Optional, has to be configured by the user).

2.5 Signatures

NEC Flash devices contain ECC (Error correction code) bits. By this feature one bit failure in a word is automatically corrected.
A signature has the size of one word (32bits + ECC bits). By creating a signature on a certain address a special combination of data bits (0x00000000) and ECC bits is written. This combination is not a valid combination for normal data or program code. So it is not possible to generate a signature by writing normal code to the signature address. This can only be done by a special command (“Create Signature” on page 37).

Signatures may have different purposes depending on the device:

- Secure reprogramming (See “Secure Device Reprogramming - 1” on page 22 and “Secure Device Reprogramming - 2” on page 24).

- Protection of the external programmer interface. Writing the Protection signature(s) disables most commands which can be sent by external Flash programmers (e.g. *flash*MASTER), like bootstrap, write, erase,...
  To enable execution of these commands again, at first the “chip prewrite” command has to be executed successfully. This command writes 0x00000000 to all Flash addresses and so deletes all Flash contents (Code and data). This command is implemented in the *flash*MASTER (Command “cpw”, *flash*MASTER firmware V3.09 onwards).

Depending on the purpose and the device, they may have to be created on more than one different address. Please refer to the *SelfLib Device Reference List* for the signature addresses.

Notes: 1. Whenever the signatures according to a signature number are created or deleted (See “Create Signature” on page 37 and “Delete Signature” on page 37), all areas in which the signatures are located have to be checked by an internal verify (See “Internal Verify” on page 36) to ensure the data retention.

2. Secure reprogramming of NEC devices is very save. Nevertheless the reprogramming sequence may be interrupted by an accidental reset or power failure. If this happens during a erase procedure or a write on a signature address, there is a theoretical chance that at the instant of the power failure or reset a valid signature is incidentally resulting (When some cells in the word are already erased/written, others still not) and the reprogramming flow is not continued correctly.
Although the risk of this event is extremely low, it shall be mentioned here.

2.6 Programming granularity

Erasing a FLASH brings all FLASH cells to a ’1’ level, that means result of reading these cells is a ’1’. Writing dedicated FLASH cells results in a ’0’ level.

Flash is always erased on area level, that means that a complete area is erased at once. So erasing only parts of an area is not possible at all.

V850 Flash is word (4 bytes) oriented. So also smallest writing unit is one word.

2.6.1 Overwriting Flash words

The V850 Flash contains additional bits for error correction per one word. So, when writing one word, 32 data bits + 6 ECC bits are written.
Following that, writing bits of an already written word from ’0’ to ’1’ is not possible because write can only write from ’1’ to ’0’.
But also writing bits of an already written word from ’1’ to ’0’ is not possible, because ECC bits can change from ’0’ to ’1’ when data bits change from ’1’ to ’0’ due to the ECC algorithm.

The only exception is writing a 0x00000000 because in that case also the error correction bits are all written with 0. Please consider, that this feature may only be used in a single shot programming sequence. That means, that the complete area reprogramming (See “Area Reprogramming” on page 22ff) must be executed in one continous sequence. Long times between the different write steps, like in EEPROM emulation, are not specified. Furthermore, an internal verify is necessary after the last write operation (See “Internal Verify” on page 36).

[MEMO]

Chapter 3 Library Functions

Library functions according to the chapter “Initialization Phase” on page 32 and “Basic Flash Functions” on page 35 are supported by all devices, while the chapter “Special Flash Functions” on page 37 describes functions, which are only supported by certain devices.
Functions in chapter “Initialization Phase” on page 32 are called directly while all other functions are called by function pointers to be independent from the location of the SelfLib_ToRAM section (See “Code Sections” on page 19).

3.1 Initialization Phase

3.1.1 Library Initialization

Library call:

SelfLib_Init (void *vpAddSelfLib,
unsigned int uiFrequency,
unsigned char *ucpWDAddSfrInt,
void *vpWDHandler,
int iCopy)

Required parameters:

vpAddSelfLib	Destination address of the section SelfLib_ToRAM Values (See “Library Handling” on page 19): 0: The *SelfLib_ToRAM* destination location is the same address as in the linker file. So the function pointers to the *Flash Access Functions* point to the original function addresses >0 (Int. RAM or ext. Memory): New address of the SelfLib_ToRAM section. The function pointers of the *Flash Access Functions* are set based on this address
uiFrequency	The operation frequency of the device (Hz) e.g.: crystal 5 MHz, internal PLL with the factor 10: uiFrequency=50,000,000
ucpWDAddSfrInt	Address of the interrupt register, that shall be polled for starting the execution of a Watchdog trigger handler. If Watchdog handling is not supported by your device or shall not be used, please set the parameter to 0.
vpWDHandler	Address of the Watchdog handler, where it shall be executed. The handler should be located in the *SelfLib_ToRAM* section to be copied into the device RAM (See “Watchdog-Handling” on page 27) If Watchdog handling is not supported by your device or shall not be used, please set the parameter to 0.
iCopy	0: The SelfLib_ToRam section is on the correct location and need not be copied 1: The SelfLib_ToRam section is copied to the address defined by ’vpAddSelfLib’. If iCopy is 0, vpAddSelfLib must also be set to 0, to ensure that the correct function pointers are returned.

Returned value:

-

Function description:

(1)Copies the *Selflib_ToRAM* section into the RAM if requested (see above).
(2)The pointers to call *Flash Access Functions* are set to the correct function addresses. So the functions can be called by the pointers.

Function call sample:

The following function call samples shall explain the usage of the SelfLib_Init function. Depending on the application one of the samples may be used. The parameters have to be adapted to the user application.

(1) The program to control the reprogramming (e.g. boot loader or monitor program) including the library is located in the device internal flash. To do Flash programming the *Selflib_ToRAM* section, possibly including the reprogramming control part of the program, needs to be copied to internal RAM or external memory.
A Watchdog handler routine is not used.

```
SelfLib_Init(   SELFLIB_DEST,     /* The copy destination address */
                FREQUENCY,        /* Operation frequency */
                0,                /* No WD handler used */
                0,                /* No WD handler used */
                1 );              /* Copy SelfLib_ToRAM to the RAM location */
```

(2) Same condition as (1), but a Watchdog handler routine is used.

```
SelfLib_Init(   SELFLIB_DEST,     /* The copy destination address */
                FREQUENCY,        /* Operation frequency */
                INT_CONT_REG,     /* Interrupt control register address */
                &WDHandler,       /* Execution Address of the WD handler routine */
                1 );              /* Copy SelfLib_ToRAM to the RAM location */
```

(3) The program to control the reprogramming (e.g. boot loader or monitor program) including the library is already located in the device internal RAM or external memory. The program code must have been linked to this location by the linker. Functions in the SelfLib_ToRAM section can be executed directly on their original location and need not be copied.
A Watchdog handler routine is not used.

```
SelfLib_Init(   0,                /* The copy destination address */
                FREQUENCY,        /* Operation frequency */
                0,                /* No WD handler used */
                0,                /* No WD handler used */
                0 );              /* Do not copy SelfLib_ToRAM to the RAM location */
```

(4) Same condition as (3), but a Watchdog handler routine is used.

```
SelfLib_Init(   0,                /* The copy destination address */
                FREQUENCY,        /* Operation frequency */
                INT_CONT_REG,     /* Interrupt control register address */
                &WDHandler,       /* Execution Address of the WD handler routine */
                0 );              /* Do not copy SelfLib_ToRAM to the RAM location */
```

Sample definitions (Depending on the application and the device):

```
#define INT_CONT_REG (unsigned char *)&CM4IC0  /* e.g. V850/IA1, Timer 4 compare interrupt */
#define SELFLIB_DEST 0xFFFFE000                /* e.g. Destination address 0xFFFFE000 */
#define FREQUENCY    16000000                  /* e.g. System operation frequency 16MHz */
```

3.1.2 New Function Address

Library call:

```
SelfLib_FktNewAddress (   void    *vpAddFkt,
                          void    *vpAddSelfLibRam)
```

Required parameters:

vpAddSelfLibRam	Destination address of the section SelfLib_ToRAM Values see “Library Initialization” on page 32
vpAddFkt	Original Function address.

Returned value:

Address of the function after the SelfLib_ToRAM section has been copied to the destination address

Function description:

If a user function is located in the SelfLib_ToRAM section and the section is copied to a new location in the internal or external RAM, the function has to be called by a function pointer (See “Code Sections” on page 19). To set-up the function pointer SelfLib_FktRamAddress returns the new address of the function, when the section has been copied by SelfLib_Init.

3.1.3 Library Version

Library call:

SelfLib_Version ()

Required parameters:

-

Returned value:

Address of the version string

Function description:

Returns the address of the version string of the *SelfLib*
Sample version string: "01.02.03"

3.1.4 Vpp Check

Library call:

SelfLib_VppCheck ()

Required parameters:

-

Returned value:

SELFLIB_VPP:	Vpp detected on the Vpp pin
SELFLIB_NO_VPP:	No Vpp detected on the Vpp pin

Function description:

Check, if Vpp is applied to the Vpp pin of the device

3.2 Basic Flash Functions

3.2.1 Area Blank Check

Library call:

fpSelfLib_BlankCheck (	unsigned int uiArea)

Required parameters:

uiArea	Area number (Not Area address, but number of the flash area)

Returned value:

SELFLIB_OK	Area is blank
SELFLIB_UNDERERASE	Area is not blank
SELFLIB_ERR_AREAINFO	Area does not exist

Function description:

This function checks, if the area is blank or not

3.2.2 Area Erase

Library call:

fpSelfLib_AreaErase (	unsigned int uiArea)

Required parameters:

uiArea	Area number (Not Area address, but number of the flash area)

Returned value:

SELFLIB_OK	Area successfully erased
SELFLIB_ERR_WRITE	Prewrite error
SELFLIB_ERR_UNDERERASE	Area could not be erased
SELFLIB_ERR_OVERERASE	Flash cells are overerased
SELFLIB_ERR_AREAINFO	Area does not exist

Function description:

This function erases a Flash area.
The following erase sequence is implemented:

(1)Prewrite	All Flash cells in the area are written with ’0’
(2)Erase	The area is erased for constant time (Erase Time)
(3)Blank Check	Check, if the area is completely erased. If not, step (2) is executed again
(4)Write Back	Writeback is executed if required

3.2.3 Write

Library call:

fpSelfLib_Write (	unsigned int uiAddSource,
	unsigned int uiAddDest,
	unsigned int uiLength)

Required parameters:

uiAddSource	Address of the source data that has to be written into the Flash
uiAddDest	Destination where the data has to be written (Word aligned address!)
uiLength	Number of WORDS (32bit) to be written

Returned value:

SELFLIB_OK	Data successfully written
SELFLIB_ERR_WRITE	Write error

Function description:

Data is written to the Flash. This is done word wise

3.2.4 Internal Verify

Library call:

fpSelfLib_IVerify (	unsigned int uiArea)

Required parameters:

uiArea	Area number (Not Area address, but number of the flash area)

Returned value:

SELFLIB_OK	Internal Verify successful
SELFLIB_ERR_VERIFY	Internal verify error
SELFLIB_ERR_AREAINFO	Area is not available

Function description:

This function compares the Flash contents on read level regarding the verify level. Internal verify of an area is **necessary** after the last write procedure in the concerning area to ensure the data retention.

3.3 Special Flash Functions

The functions explained in this chapter are not available on all devices. Please refer to the *SelfLib Device Reference List* to check whether your device supports the functions.

3.3.1 Create Signature

Library call:
fpSelfLib_CreateSig (unsigned int uiNo)
Required parameters:
uiNo Number of the Signature to be created

Returned value:
SELFLIB_OK Signature written successfully
SELFLIB_ERR_WRITE Signature write error

Function description:
This function writes the Signature(s) according to the signature number to the specified address(es). The signature numbers and the related addresses are listed up the *SelfLib Device Reference List*, which can be loaded from the same internet address as this document. Please refer to “Signatures” on page 29 for detailed explanation of the signatures

3.3.2 Delete Signature

Library call:
fpSelfLib_DeleteSig (unsigned int uiNo)
Required parameters:
uiNo Number of the Signature(s) to be deleted

Returned value:
SELFLIB_OK Signature deleted successfully
SELFLIB_ERR_WRITE Signature write error

Function description:
This function deletes the Signature(s) according to the signature number on the specified address(es). This is done by overwriting them with 0x00000000.
The signature numbers and the related addresses are listed up the *SelfLib Device Reference List*, which can be loaded from the same internet address as this document. Please refer to “Signatures” on page 29 for detailed explanation of the signatures

3.3.3 Check Signature

Library call:
fpSelfLib_CheckSig (unsigned int uiNo)
Required parameters:
uiNo Number of the Signature to be checked

Returned value:
SELFLIB_SIG Signature available
SELFLIB_NOSIG Signature is not available

Function description:
This function checks, whether the Signature(s) according to the signature number are present or not. The signature numbers and the related addresses are listed up the *SelfLib Device Reference List*, which can be loaded from the same internet address as this document. Please refer to “Signatures” on page 29 for detailed explanation of the signatures.

3.3.4 Check Area

Library call:
fpSelfLib_CheckArea ()

Required parameters:
-

Returned value:
0: The area 0 is located at the address 0x00000000
1: The other swappable area is located at the address 0x00000000

Function description:
This function checks, which area is located at the address 0x00000000 (Reset vector).

3.3.5 Swap Area

Library call:

fpSelfLib_SwapArea (	unsigned int uiArea)

Required parameters:

uiArea	0: The area 0 shall be located at the address 0x00000000 1: The other swappable area shall be located 0x00000000

Returned value:
-

Function description:
This function swaps the area related to 'uiArea' to the address 0x0000000. This is only valid until the device has restarted after a Reset in normal mode. Then the internal firmware selects the area to be located at 0x0000000 according to the found signature(s).

Sample routine to exchange the location of the two swappable areas:

```
uiArea = fpSelfLib_CheckArea ();        /*Check, which area is currently located at 0x00000000*/
uiArea = uiArea^1;                      /* XOR with 1 to invert the area flag */
fpSelfLib_SwapArea( uiArea );           /* Exchange the location of the two areas */
```

3.3.6 Read Back

Library call:

fpSelfLib_Read (	unsigned int uiAddSource, unsigned int uiAddDest, unsigned int uiLength)

Required parameters:

uiAddSource	Flash read address
uiAddDest	Destination address in the RAM
uiLength	Number of WORDS to be read out

Returned value:
-

Function description:
Data is read from the Flash and written into the RAM.

[MEMO]

Chapter 4 Revision History

V1.0	Initial Version	
V1.1	Chapter 1:	Sub-chapter ’Compiler Options’ extended
	Chapter 2:	Function call parameters and return values corrected
		Function call samples added for fpSelfLib_Init and fpSelfLib_SwapArea
		Function descriptions extended
V3.0	Chapter 1:	Stacksize adjusted
		Compiler options extended
	Chapter 2:	Function call parameter explanation replenished

[MEMO]

NEC

Facsimile Message

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

From:

Name

Company

Tel. FAX

Address

Thank you for your kind support.

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

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

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

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

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

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

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

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

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

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

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

CS 99.1

[MEMO]