NEC

Customer Notification

μPD78F9234 Subseries

8-Bit Single-Chip Microcontrollers

Operating Precautions

μPD78F9232
μPD78F9234

Document No. U18765EE1V0IF00
Date Published May 2007

DISCLAIMER

The related documents in this customer notification may include preliminary versions. However, preliminary versions may not have been marked as such.

The information in this customer notification is current as of its date of publication. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC's data sheets or data books, etc., for the most up-to-date specifications of NEC PRODUCT(S). Not all PRODUCT(S) and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.

No part of this customer notification may be copied or reproduced in any form or by any means without prior written consent of NEC. NEC assumes no responsibility for any errors that may appear in this customer notification. NEC does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC PRODUCT(S) listed in this customer notification or any other liability arising from the use of such PRODUCT(S).

No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC or others. Descriptions of circuits, software and other related information in this customer notification are provided for illustrative purposes of PRODUCT(S) operation and/or application examples only. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.

While wherever feasible, NEC endeavors to enhance the quality, reliability and safe operation of PRODUCT(S) the customer agree and acknowledge that the possibility of defects and/or erroneous thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects and/or errors in PRODUCT(S) the customer must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.

The customer agrees to indemnify NEC against and hold NEC harmless from any and all consequences of any and all claims, suits, actions or demands asserted against NEC made by a third party for damages caused by one or more of the items listed in the enclosed table of content of this customer notification for PRODUCT(S) supplied after the date of publication.

Applicable Law:

The law of the Federal Republic of Germany applies to all information provided by NEC to the Customer under this Operating Precaution document without the possibility of recourse to the Conflicts Law or the law of 5th July 1989 relating to the UN Convention on Contracts for the International Sale of Goods (the Vienna CISG agreement).

Düsseldorf is the court of jurisdiction for all legal disputes arising directly or indirectly from this information. NEC is also entitled to make a claim against the Customer at his general court of jurisdiction.

If the supplied goods/information are subject to German, European and/or North American export controls, the Customer shall comply with the relevant export control regulations in the event that the goods are exported and/or re-exported. If deliveries are exported without payment of duty at the request of the Customer, the Customer accepts liability for any subsequent customs administration claims with respect to NEC.

Notes: (1) "**NEC**" as used in this statement means NEC Corporation and also includes its direct or indirect owned or controlled subsidiaries.

(2) "**PRODUCT(S)**" means 'NEC semiconductor products' *(NEC semiconductor products* means any semiconductor product developed or manufactured by or for NEC) and/or 'TOOLS' *(TOOLS* means 'hardware and/or software development tools' for NEC semiconductor products' developed, manufactured and supplied by 'NEC' and/or 'hardware and/or software development tools' supplied by NEC but developed and/or manufactured by independent 3rd Party vendors worldwide as their own product or on contract from NEC)

Table of Contents

µPD78F9234 Subseries

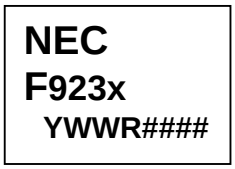
(A) Table of Operating Precautions

No.	Outline		µPD78F923x			
		Rev.	all			
		Rank Note	all			
1	Restriction on using flash self-programming (Direction of use)		✗			

✓: Not applicable
✗: applicable

Note: The rank is indicated by the letter appearing at the 4th position from the left in the lot number, marked on each product.

Marking of the devices



Y: Year Code
WW: Week Code
R: Rank
###: Inhouse Code

(B) Description of Operating Precautions

No. 1	Restriction on using flash self-programming (Direction of use)

Details

If the standby function performed by the HALT instruction and flash self-programming are used together using the procedure shown below, the subsequent operation becomes unexpected.

Workaround

When using flash self-programming, clear the FLCMD register to 0 immediately before shifting to normal mode or self-programming mode. In addition, execute NOP and HALT instructions after specific sequence processing to shift to self-programming mode.

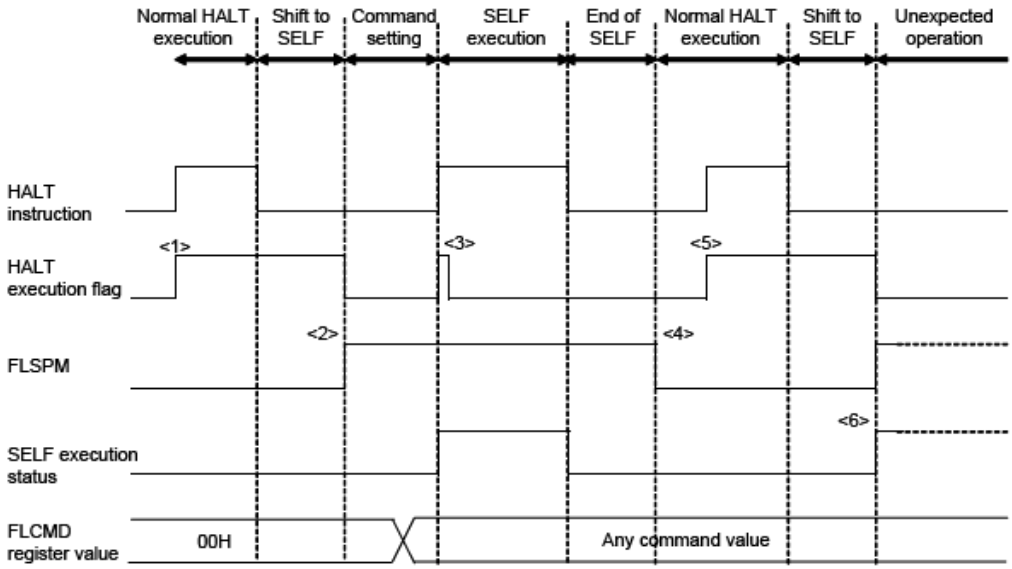
Detailed explanation

The following two modes are available in these products.

- Normal mode:
  The state in which normal operation is executed. Operation enters into a standby state after execution of the HALT instruction.

- Self-programming mode:
  The state in which self-programming commands are executable. After setting commands,addresses and write data and executing the HALT instruction, self-programming is executed. The specific sequence described in this document is referring to the register manipulation to switch these two modes.

Process Leading up to Unexpected Operation



<1> An ordinary HALT instruction is executed and the internal HALT execution flag is set. Self-programming is executed by setting the FLSPM bit while the HALT execution flag is set.
<2> The specific sequence is executed and the operation then enters into self-programming mode. At this time, the FLSPM bit changes to indicate that self-programming is now executable. However, self-programming commands are not executed at this time, because the FLCMD register has been initialized to 00H.
<3> Once a command value is set to the FLCMD register and the HALT instruction is executed, the self-programming command is executed. The HALT execution flag is cleared just as the self-programming command is executed.
<4> Execution of the self-programming command is completed, the specific sequence is executed again, and operation enters into normal mode.
<5> The HALT instruction is executed again, operation enters into standby, and the HALT execution flag is set.
<6> After the standby state is released, the specific sequence is executed to shift to self-programming mode. If the command value set to the FLCMD register has not been initialized at this time, the command still set to the FLCMD register is reexecuted when the FLSPM bit is set. Self-programming is subsequently executed during CPU operation and the CPU fetches an incorrect instruction from the flash memory, resulting in an unexpected operation.

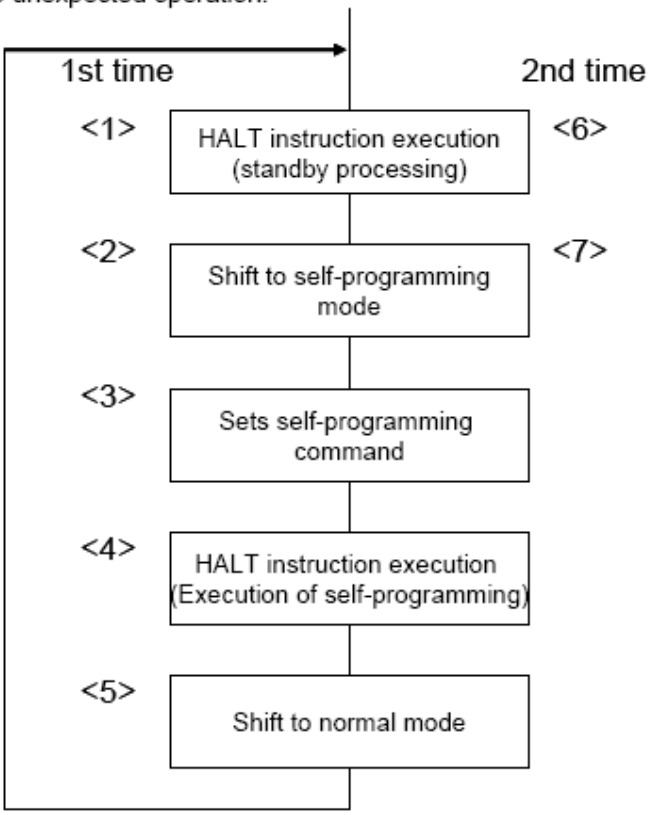
Remark The same situation occurs when flash self-programming is executed before <1>.

Workaround:

When using flash self-programming, clear the FLCMD register to 0 immediately before shifting to normal mode or self-programming mode; this prevents execution of illegal commands immediately after the mode is shifted. In addition, execute NOP and HALT instructions after specific sequence processing to shift to self-programming mode; this controls the execution timing between the CPU and the flash memory control block.

The flowcharts and source code examples for the operation restriction and its workaround implementation are described on the following pages.

Flowchart leading up to unexpected operation:



<1> An ordinary HALT instruction is executed to shift to standby. After that, the standby state is released by a standby release signal, such as an interrupt.
<2> The specific sequence is executed to shift to self-programming mode.
<3> A self-programming command (block erase in the source code example) is set to the FLCMD register.
<4> The self-programming command is executed by executing the HALT instruction.
<5> After self-programming processing specified in <3> is completed, the specific sequence is executed to shift to normal mode. This example presumes that processes from <1> to <5> are performed repeatedly.
<6> An ordinary HALT instruction is executed to generate a standby release signal, and the standby state is released.
<7> A self-programming command (block erase in the source code example) is executed immediately after the specific sequence is executed to shift to self-programming mode for the second time. Consequently, microcontroller operation becomes unexpected.

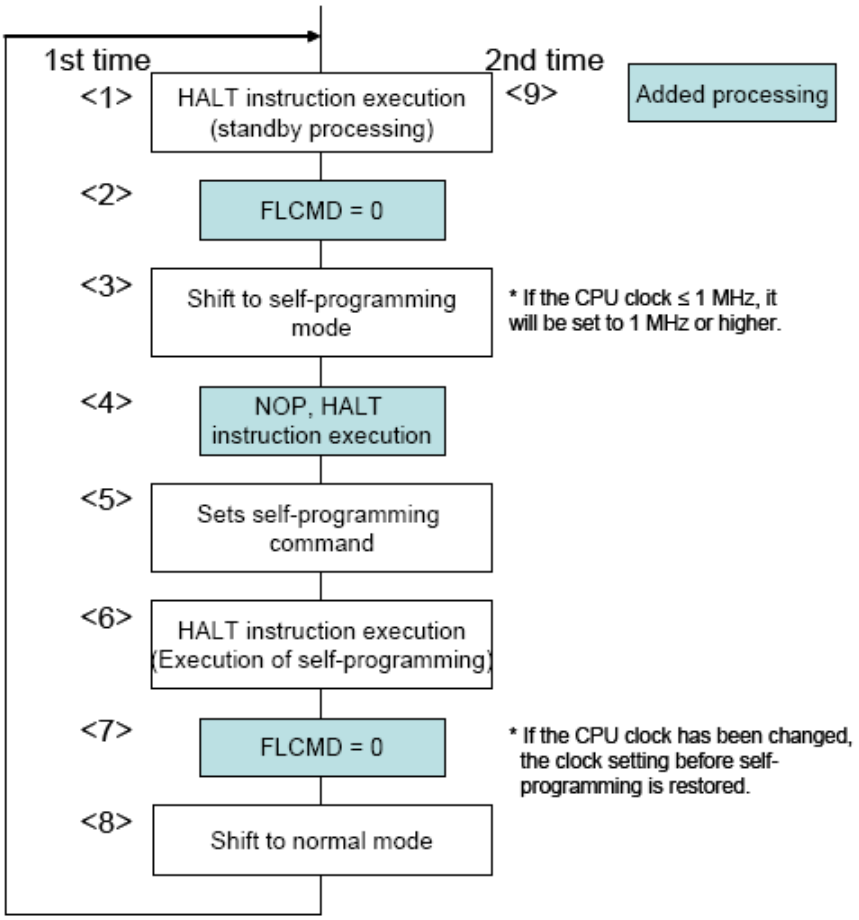
Example source code causing unexpected operation (assembly language):

```
MAINLOOP:
        ; Executes HALT to shift to standby state - <1> and <6> in flowchart
        HALT

        ; Saves the interrupt mask setting before executing self-programming.
        DI                      ; Disables interrupts
        MOV     A,MK0           ; Saves interrupt mask setting
        XCH     A,X
        MOV     A,MK1           ; Saves only MK0 in KU1+ and KY1+
        PUSH    AX
        MOV     MK0,#0FFH
        MOV     MK1,#0FFH

        ; Executes the specific sequence to shift to self-programming mode - <2> and <7> in flowchart
ModeOnLoop:
        MOV     PFCMD,#0A5H     ; Controls PFCMD register
        MOV     FLPMC,#01H      ; Controls FLPMC register (set value)
        MOV     FLPMC,#0FEH     ; Controls FLPMC register (inverted set value)
        MOV     FLPMC,#01H      ; Sets self-programming mode
        ;When using a clock generated by an external resonator or external input clock,insert a 16 υs wait.

        ; Operation becomes unexpected when entered into self-programming mode for the second time.
        BT      PFS.0,$ModeOnLoop       ; Confirms completion of mode shift

        ; Performs command settings - <3> in flowchart
        MOV     A, #0FH
        MOV     FLAPH,A         ; Sets number of block to be erased
        MOV     FLAPHC,A        ; Sets compare number for block to be erased (value set to FLAPH)
        MOV     FLCMD,#03H      ; Sets flash control command (block erase)
        MOV     PFS,#00H        ; Clears flash status register
        MOV     WDTE,#0ACH      ; Clears and starts WDT

        ; Executes erase command - <4> in flowchart
        HALT ; Executes self-programming

        ; Executes the specific sequence to shift to normal mode - <5> in flowchart
ModeOffLoop:
        MOV     PFS,#00H
        MOV     PFCMD,#0A5H     ; Controls PFCMD register
        MOV     FLPMC,#00H      ; Controls FLPMC register (set value)
        MOV     FLPMC,#0FFH     ; Controls FLPMC register (inverted set value)
        MOV     FLPMC,#00H      ; Sets normal mode
        BT      PFS.0,$ModeOffLoop      ; Confirms completion of mode shift
        POP     AX              ; Restores interrupt mask setting
        MOV     MK1,X
        XCH     A,X
        MOV     MK0,A
        BR      MAINLOOP
```

Example source code causing unexpected operation (C language):

```
while(1){
        /* Executes HALT to shift to standby state - <1> and <6> in flowchart */
        HALT();

        /* Saves interrupt mask settings */
        DI();                                   // Disables interrupts
        ch_mask_bak0 = MK0;                     // Saves only MK0 in KU1+ and KY1+
        ch_mask_bak1 = MK1;                     // ch_mask_bak0/1 are variables for saving

        /* Shifts to self-programming mode - <2> and <7> in flowchart */
        do{
                PFS = 0;                        // Clears flash status register
                PFCMD = 0xA5;                   // Controls PFCMD register
                FLPMC = 0x01;                   // Controls FLPMC register (set value)
                FLPMC = 0xFE;                   // Controls FLPMC register (inverted set value)
                FLPMC = 0x01;                   // Sets self-programming mode

                /* When using a clock generated by an external resonator or external input clock,
                insert a 16 ⌈s wait.*/

                /* Operation becomes unexpected when entered into self-programming mode
                for the second time. */

        }while(PFS.0 == 1);                     // Confirms completion of mode shift

        /* Performs command settings - <3> in flowchart */
        FLAPH = FLAPHC = 0x0F;                  // Specifies block to be erased
        FLCMD = 0x03;                           // Specifies erase command
        PFS = 0x00;                             // Clears flash status register
        WDTE = 0xAC;                            // Clears WDT counter

        /* Executes erase command - <4> in flowchart */
        HALT();                                 // Executes erase command

        /* Shifts to normal mode - <5> in flowchart */
        do{
                PFS = 0;                        // Clears flash status register
                PFCMD = 0xA5;                   // Controls PFCMD register
                FLPMC = 0x00;                   // Controls FLPMC register (set value)
                FLPMC = 0xFF;                   // Controls FLPMC register (inverted set value)
                FLPMC = 0x00;                   // Sets normal mode
        }while(PFS.0 == 1);                     // Confirms completion of mode shift

        /* Restores interrupt mask settings */
        MK0 = ch_mask_bak0;
        MK1 = ch_mask_bak1;
    }
```

Flowchart of workaround implementation:



<1>An ordinary HALT instruction is executed to shift to standby. After that, the standby state is released by a standby release signal, such as an interrupt.
<2>The FLCMD register is cleared to 0 before executing the specific sequence to shift to self-programming mode. The CPU clock is set to 1 MHz or higher.
<3>The specific sequence is executed to shift to self-programming mode.
<4>After the specific sequence (1 assigned to FLPMC for the second time), NOP and HALT instructions are executed. It takes at most 10 ⌈s until the HALT instruction is released.
<5>A self-programming command (such as write or erase) is set to the FLCMD register.
<6>The self-programming command is executed by executing the HALT instruction.
<7>The FLCMD register is cleared to 0 before the specific sequence is executed to shift to normal mode.
<8>Execute the specific sequence to shift to normal mode. If the CPU clock has been changed, the clock setting before self-programming is restored at this time.
<9>This restriction is avoided by adding the above processes <2>, <4>, <7> and <8>.

Example of source code to which workaround is implemented (assembly language):

```
MAINLOOP:
        ; Executes HALT to shift to standby state - <1> and <9> in flowchart
        HALT

        ; Saves the interrupt mask setting before executing self-programming.
        DI                      ; Disables interrupts
        MOV A,MK0               ; Saves interrupt mask setting
        XCH A,X
        MOV A,MK1               ; Saves only MK0 in KU1+ and KY1+
        PUSH AX
        MOV MK0, #0FFH
        MOV MK1, #0FFH

        ; Initializes FLCMD register - <2> in flowchart
        MOV FLCMD, #00H
        ; If CPU clock δ 1 MHz, sets CPU clock to 1 MHz or higher.

        ; Executes the specific sequence to shift to self-programming mode - <3> in flowchart
ModeOnLoop:
        MOV PFCMD,#0A5H         ; Controls PFCMD register
        MOV FLPMC,#01H ; Controls FLPMC register (set value)
        MOV FLPMC,#0FEH         ; Controls FLPMC register (inverted set value)
        MOV FLPMC,#01H ; Sets self-programming mode
        BT PFS.0,$ModeOnLoop                    ; Confirms completion of mode shift

        ; Executes NOP and HALT instructions - <4> in flowchart
        NOP
        HALT
        ;When using a clock generated by an external resonator or external input clock, insert an 8us wait.

        ; Performs command settings - <5> in flowchart
        MOV A, #0FH
        MOV FLAPH,A             ; Sets number of block to be erased
        MOV FLAPHC,A            ; Sets compare number for block to be erased (value set to FLAPH)
        MOV FLCMD,#03H ; Sets flash control command (block erase)
        MOV PFS,#00H            ; Clears flash status register
        MOV WDTE,#0ACH          ; Clears and starts WDT
        ; Executes erase command - <6> in flowchart
        HALT                    ; Executes self-programming

        ; Initializes FLCMD register - <7> in flowchart
        MOV FLCMD, #00H
        ; If the CPU clock has been changed, the setting before self-programming is restored.

        ; Executes the specific sequence to shift to normal mode - <8> in flowchart
ModeOffLoop:
        MOV PFS,#00H
        MOV PFCMD,#0A5H         ; Controls PFCMD register
        MOV FLPMC,#00H ; Controls FLPMC register (set value)
        MOV FLPMC,#0FFH         ; Controls FLPMC register (inverted set value)
        MOV FLPMC,#00H ; Sets normal mode
        BT PFS.0,$ModeOffLoop   ; Confirms completion of mode shift
        POP AX          ; Restores interrupt mask setting
        MOV MK1,X
        XCH A,X
        MOV MK0,A
        BR MAINLOOP
```

Example of source code to which workaround is implemented (C language):

```
while(1){
        /* Executes HALT to shift to standby state - <1> and <9> in flowchart */
        HALT();

        /* Saves interrupt mask settings */
        DI();                   // Disables interrupts
        ch_mask_bak0 = MK0;     // Saves only MK0 in KU1+ and KY1+
        ch_mask_bak1 = MK1;     // ch_mask_bak0/1 are variables for saving

        /* Initializes FLCMD register - <2> in flowchart */
        FLCMD = 0;
        /* If CPU clock δ 1 MHz, sets CPU clock to 1 MHz or higher */

        /* Enters into self-programming mode - <3> in flowchart */
        do{
                PFS = 0;            // Clears flash status register
                PFCMD = 0xA5;       // Controls PFCMD register
                FLPMC = 0x01;       // Controls FLPMC register (set value)
                FLPMC = 0xFE;       // Controls FLPMC register (inverted set value)
                FLPMC = 0x01;       // Sets self-programming mode
        }while(PFS.0 == 1);         // Confirms completion of mode shift

        /* Executes NOP and HALT instructions - <4> in flowchart */
        NOP();
        HALT();
        /* When using a clock generated by an ext. resonator or external input clock, insert an 8us wait.*/

        /* Performs command settings - <5> in flowchart */
        FLAPH = FLAPHC = 0x0F; // Specifies block to be erased
        FLCMD = 0x03;           // Specifies erase command
        PFS = 0x00;             // Clears flash status register
        WDTE = 0xAC;            // Clears WDT counter

        /* Executes erase command - <6> in flowchart */
        HALT();                 // Executes erase command

        /* Initializes FLCMD register - <7> in flowchart */
        FLCMD = 0;

        /* If the CPU clock has been changed, the setting before self-programming is restored. */

        /* Shifts to normal mode - <8> in flowchart */
        do{
                PFS = 0;            // Clears flash status register
                PFCMD = 0xA5;       // Controls PFCMD register
                FLPMC = 0x00;       // Controls FLPMC register (set value)
                FLPMC = 0xFF;       // Controls FLPMC register (inverted set value)
                FLPMC = 0x00;       // Sets normal mode
        }while(PFS.0 == 1);         // Confirms completion of mode shift

        /* Restores interrupt mask settings */
        MK0 = ch_mask_bak0;
        MK1 = ch_mask_bak1;
        }
```

(C) Valid Specification

Item	Date published	Document No.	Document Title
1	Jan 2007 or later	U17446EJ…	User's Manual

(D) Revision History

Item	Date published	Document No.	Comment
1	May, 2007	U18765EE1V0IF00	1st Release
2			