AP-627

APPLICATION NOTE

Byte-Wide FlashFile™ Memory Family Software Drivers

BRIAN DIPERT
MCD MARKETING
APPLICATIONS

KEN MCKEE
TECHNICAL MARKETING
ENGINEER

April 1996

Order Number: 292182-001

CONTENTS

REVISION HISTORY

Number	Description
-001	Original version

1.0 INTRODUCTION

This application note provides example software for controlling Intel's byte-wide FlashFile™ memory family which includes the 28F004SC, 28F008SA, 28F008SC, and 28F016SC. Two programming languages are provided: ASM86 assembly and high-level "C" for multi-platform support. In many cases, the driver routines can be inserted "as is" into the main body of code being developed by the software engineer. Each routine includes extensive comments to facilitate adapting the code to specific applications.

The devices' internal automation makes software timing loops unnecessary and results in platform-independent code. This software executes in any type of memory and with all processor clock rates. "C" code ports easily to many microprocessors, while ASM86 assembly code provides a solution optimized for Intel microprocessors and embedded controllers.

Below is a list of software driver assumptions.

- Pointers (in "C") or EDI offsets (in ASM86) are four bytes long, providing a flat addressing space over the entire memory space.
- A "char" is 8 bits, "int" is 16 bits and "long" is 32 bits in "C."
- A "set_pin" function controls high voltage on RP#. The function enables in-system hardware block locking.
- Writes and reads are to one device. Minor modifications are needed for a device pair.

2.0 SOFTWARE COMPATIBILITY

The 28F008SA is software compatible with the 28F004SC, 28F008SC, and 28F016SC. These components also share a common status register definition. The SmartVoltage FlashFile memory devices, the 28F004SC, 28F008SC, and 28F016SC, provide a superset of software commands to enable new and enhanced features. Table 1 highlights the common and new commands.

Table 1. Software Compatibility Checklist

Procedure	28F008SA	28F004SC 28F008SC 28F016SC
block_erase	✓	✓
byte_write	✓	✓
erase_suspend_to_read	✓	✓
erase_suspend_to_write		✓
write_suspend_to_read		✓
set_block_lock_bit		✓
set_master_lock_bit		✓
clear_block_lock_bits		✓
read_identifier_codes	✓	✓
SR_full_status_check	✓	✓

Because of density differences and software enhancements, the devices do not share the same device code. This difference allows for software component identification. System software can read the device code and select the appropriate algorithms for the given component.

Table 2. Device Codes for the Byte-Wide FlashFile™ Memory Family

Device	Device Code (Hex)
28F004SC/SC-L	A7
28F008SA-L	A1
28F008SA	A2
28F008SC/SC-L	A6
28F016SC/SC-L	AA

Table 3. Status Register Definition

Bit	Description	Status
SR.7	WSM Status	1 = Ready 0 = Busy
SR.6	Erase Suspend Status	1 = Erase Suspended 0 = Erase in Progress/Completed
SR.5	Erase and Clear Lock-Bits Status	1 = Error in Block Erase or Clear Block Lock-Bits 0 = Successful Block Erase or Clear Block Lock-Bits
SR.4	Byte Write and Set Lock-Bit Status	1 = Error in Byte Write or Set Block/Master Lock-Bit 0 = Successful Byte Write or Set Block/Master Lock-Bit
SR.3	V_{PP} Status	1 = V_{PP} Low Detect, Operation Abort 0 = V_{PP} OK
SR.2	Byte Write Suspend Status	1 = Byte Write Suspended 0 = Byte Write in Progress/Completed
SR.1	Device Protect Status	1 = Block Lock-Bit, Master Lock-Bit and/or RP# Lock Detected, Operation Abort 0 = Unlock
SR.0	Reserved for Future Use	

NOTE:
Underlined text ONLY applies to the 28F004SC, 28F008SC, and 28F016SC.

The example code makes use of bit masking when reading information from the status register. Table 3 defines the meaning of status register bits for the 28F008SA and byte-wide SmartVoltage FlashFile memories. Note that bits SR.2 and SR.1 provide additional system feedback for the 28F004SC, 28F008SC, and 28F016SC. These bits were added to support the new features and previously reserved for future use in the 28F008SA status register definition. Code written for the 28F008SA should mask these two bits when polling the status register.

3.0 CONCLUSION

This application note provides example code for the Intel byte-wide FlashFile memory family. This information facilitates software development. For further information about these components, consult reference documentation in Appendix C.

APPENDIX A
"C" DRIVERS

```
/****************************************************************************************/
/* Header file for "C" drivers for Intel's byte-wide FlashFile memory family              */
/****************************************************************************************/

/****************************************************************************************/
/* Copyright Intel Corporation, 1996                                                      */
/* File: stddefs.h                                                                        */
/* Standard definitions for C Drivers for Intel's byte-wide FlashFile memory family       */
/* Author: Ken McKee                                                                      */
/* Revision 1.0, January 1, 1996                                                          */
/****************************************************************************************/

/****************************************************************************************/
/* error codes                                                                            */
/****************************************************************************************/
#define NO_ERROR                    0
#define VPP_ERROR                   1
#define WRITE_ERROR                 2
#define ERASE_ERROR         3
#define BLOCK_PROTECTION_ERROR      4       /* ONLY valid for the 28F0xxSC.          */
#define COMMAND_SEQ_ERROR           5

/****************************************************************************************/
/* bit mask                                                                               */
/****************************************************************************************/
#define BIT_0                       0x01
#define BIT_1                       0x02
#define BIT_2                       0x04
#define BIT_3                       0x08
#define BIT_4                       0x10
#define BIT_5                       0x20
#define BIT_6                       0x40
#define BIT_7                       0x80
```

```
/*****************************************************************************************/
/* "C" drivers for Intel's byte-wide FlashFile memory family                              */
/*****************************************************************************************/

/*****************************************************************************************/
/* Copyright Intel Corporation, 1996                                                      */
/* Example C Routines for Intel's byte-wide FlashFile memory family                       */
/* File: ff_drv.c                                                                         */
/* Author: Ken McKee                                                                      */
/* Revision 1.0, January 1, 1996                                                          */
/*****************************************************************************************/

#include <stdio.h>
#include "stddefs.h"

void set_pin(int level)               /* Controls RP# voltage.                               */
{
     /* An implementation-dependent function that controls RP# high voltage (2 8F0xxSC ONLY).  */
}

char block_erase(char *address)              /* Works for the 28F008SA and 28F0xxSC.         */
{
     /* This procedure erases a 64-Kbyte block.                                               */

     char SR;                                /* SR variable returns content  of SR.          */

     *address = 0x20;                        /* Block Erase command.                          */
     *address = 0xD0;                        /* Confirm command.                              */
     while(!(BIT_7 & *address))              /* Poll SR until SR.7 = 1.                       */
     {
            /* Erase may be suspended here to  read or write to a different block.            */
     };
     SR = *address;                          /* Save SR before clearing it.                   */
     *address = 0x50;                        /* Clear SR command and place device in read mode. */
     return(SR);                             /* Return SR to be checked for status of operation. */
}

char byte_write(char *address, char data)    /* Works for the 28F008SA and 28F0xxSC.         */
{
     /* This procedure writes a byte.                                                         */

     char SR;                                /* SR variable returns content of SR.            */

     *address = 0x40;                        /* Byte Write command.                           */
     *address = data;                        /* Actual data write to flash address.           */
     while(!(BIT_7 & *address))              /* Poll SR until SR.7 = 1.                       */
     {
          /* Byte write may be suspended here to read from a different location (28F0xxSC ONLY).*/
     };
     SR = *address;                          /* Save SR before clearing it.                   */
     *address = 0x50;                        /* Clear SR command and plac e device in read mode. */
     return(SR);                             /* Return SR to be checked  for status of operation. */
}
```

```
void erase_suspend_to_read(char *address, char *result)
                                        /* Works for the 28F008SA and 28F0xxSC.              */
{
    /* This procedure suspends an erase operation to do a read. It assumes erase is underway.   */
    /* The procedure works equally well for the 28F008SA and 28F0xxSC.                       */

    *address = 0xB0;                    /* Block Erase Suspend command.                       */
    while(!(BIT_7 & *address));         /* Poll SR until SR.7 = 1.                            */
    *address = 0xFF;                    /* Read Flash Array command.                          */
    *result = *address;                 /* Do the actual read. Any number of reads can be     */
                                        /* done here.                                         */
    *address = 0x70;                    /* Read SR command.                                   */
    if (BIT_6 & *address)               /* If SR.6 = 1 (erase incomplete).                    */
         *address = 0xD0;               /* Block Erase Resume command                         */
}

void erase_suspend_to_write(char *address, char data)
                                        /* Works for the 28F0xxSC ONLY.                       */
{
    /* This procedure suspends an erase operation to do a byte write. It assumes erase is underway.   */
    /* The procedure ONLY applies to the 28F0xxSC.                                            */

    *address = 0xB0;                    /* Block Erase Suspend command.                       */
    while(!(BIT_7 & *address));         /* Poll SR until SR.7 = 1.                            */
    *address = 0x40;                    /* Byte Write command.                                */
    *address = data;                    /* Actual data write to flash address.                */
    while(!(BIT_7 & *address))          /* Poll SR until SR.7 = 1.                            */
    {
         /* Byte write may be suspended here to read from a different location (28F0xxSC ONLY).   */
    };
    if (BIT_6 & *address)               /* If SR.6 = 1 (erase incomplete).              */
         *address = 0xD0;               /* Block Erase Resume command.                        */
}

void byte_suspend_to_read(char *address, char *result)
                                        /* Works for the 28F0xxSC ONLY.                       */
{
    /* This procedure suspends an byte write operation to do a read. It assumes write is underway.   */
    /* The procedure ONLY applies to the 28F0xxSC.                                            */

    *address = 0xB0;                    /* Block Erase Suspend command.                       */
    while(!(BIT_7 & *address));         /* Poll SR until SR.7 = 1.                            */
    *address = 0xFF;                    /* Read Flash Array command                           */
    *result = *address;                 /* Do the actual read. Any number of reads can be     */
                                        /* done here.                                         */
    *address = 0x70;                    /* Read SR command.                                   */
    if (BIT_2 & *address)               /* If SR.2 = 1 (byte write incomplete).               */
         *address = 0xD0;               /* Erase Resume command.                        */
}
```

```
char set_block_lock_bit(char *lock_address)       /* Works for the 28F0xxSC ONLY.                    */
{
        /* This procedure sets a block lock-bit on the 28F0xxSC.                                       */

        char SR;                                  /* SR variable returns content  of SR.             */

        /* If the master lock-bit is set, RP# = Vhh                                                    */
        /* set_pin(1);          */                /* Enable high voltage on to RP                    */

        *lock_address = 0x60;                     /* Set Block Lock-Bit command.                     */
        *lock_address = 0x01;                     /* Set block lock-bit confirmation.                */
        while (!(BIT_7 & *lock_address);          /* Poll SR until SR.7 = 1.                         */

        /* If the Master lock-bit is set return RP# to Vih.                                            */
        /* set_pin(0);          */                /* Disable high voltage on to RP#.                 */

        SR = *lock_address;                       /* Save SR before clearing it.                     */
        *lock_address = 0x50;                     /* Clear SR command and place device in read mode. */
        return(SR);                               /* Return SR to be checked for status of operation.*/
}

char set_master_lock_bit(char *lock_address)      /* Works for the 28F0xxSC ONLY.                    */
{
        /* This procedure sets the master lock-bit on the 28F0xxSC.                                    */

        char SR;                                  /* SR variable returns content of SR.              */

        set_pin(1);                               /* Enable high voltage on to RP#.                  */
        *lock_address = 0x60;                     /* Set Master Lock-Bit command.                    */
        *lock_address = 0xF1;                     /* Set master lock-bit confirmation.               */
        while (!(BIT_7 & *lock_address);          /* Poll SR until SR.7 = 1.                         */
        set_pin(0);                               /* Disable high voltage on to RP#.                 */
        SR = *lock_address;                       /* Save SR before clearing it.                     */
        *lock_address = 0x50;                     /* Clear SR command and place device in read mode. */
        return(SR);                               /* Return SR to be checked for status of operati on.*/
}
```

```
char clear_block_lock_bits(char *lock_address)  /* Works for the 28F0xxSC ONLY.                        */
{
        /* This procedure clears all block lock-bits on the 28F0xxSC.                                 */

        char SR;                                /* SR variable returns content of SR.                 */

        /* If the Master lock-bit is set, RP# = Vhh                                                   */
        /* set_pin(1);          */              /* Enable high voltage on to RP#.                     */

        *lock_address = 0x60;                   /* Clear Block Lock-Bits command.                     */
        *lock_address = 0xD0;                   /* Clear block lock-bits confirmation.                */
        while (!(BIT_7 & *lock_address);        /* Poll SR until SR.7 = 1.                            */

        /* If the Master lock-bit is set return RP# to Vih.                                           */
        /* set_pin(0);          */              /* Disable high voltage on to RP#.                    */

        SR = *lock_address;                     /* Save SR before clearing it.                        */
        *lock_address = 0x50;                   /* Clear SR command and place device in read mode.    */
        return(SR);                             /* Return SR to be checked for status of operation.   */
}

char read_identifier_codes(char *address)       /* Works for the 28F008SA and 28F0xxSC.               */
{
        /* This procedure provides access to the 28F0xxSC's Manufacture Code, Device Code,            */
        /* Master/Block Lock Configuration Code. As well, this procedure can provide access to the    */
        /* 28F008SA's Manufacture Code and Device Code.                                               */

        char code;                              /* ID code variable returned                          */

        *address = 0x90;                        /* Read Identifier Codes command.                     */
        code = *address;                        /* Store code.                                        */
        *address = 0xFF;                        /* Read Flash Array command                           */
        return(code);                           /* Return value.                                      */
}
```

```
char SR_full_status_check(char SR)          /* Works for the 28F008SA and 28F0xxSC.                  */
{
        /* This procedure performs a full SR check. It is valid for byte write, block erase, lock-bet set and  */
        /* block lock-bit reset operations.                                                          */
        /*                                                                                           */
        /* Note:    This procedure assumes that SR data resides in SR. This information is placed in the     */
        /*          variable after the completion of each operation. If an error is detected, the previous   */
        /*          operation should be executed again.                                              */

        char error_code;                    /* returns error code.                                  */

        if (SR & BIT_3)                     /* Vpp range error check.                               */
                error_code = VPP_ERROR;     /* Set error code.                                      */
        else if (SR & BIT_1)                /* Device protection error check.                       */
                error_code = BLOCK_PROTECTION_ERROR;
                                            /* Set error code. ONLY valid for the 28F00SC. This     */
                                            /* check should be remove for the 28F008SA.             */
        else if (SR & BIT_4){               /* Byte write error check.                              */
                if (SR & BIT_5)     /* Command sequence error check.                        */
                        error_code = COMMAND_SEQ_ERROR;
                                            /* Set error code.                                      */
                else
                        error_code = BYTE_WRITE_ERROR;     /* Set byte write error code.            */
        }
        else if (SR & BIT_5)                /* Block erase error check.                             */
                error_code = ERASE_ERROR;
                                            /* Set error code.                                      */
        else                                /* No error detected.                                   */
                error_code = NO_ERROR;      /* Set error code.                                      */
        return(error_code);                 /* Return error code.                                   */
}
```

APPENDIX B
ASM86 DRIVERS

```
;===========================================================================
; ASM86 assembly language drivers for Intel's byte-wide FlashFile memory family
;===========================================================================

;===========================================================================
; Copyright Intel Corporation, 1996
; EXAMPLE ASM86 Drivers for Intel's byte-wide FlashFile memory family
; Author: Ken McKee, Intel Corporation
; Revision 1.0, January 1, 1996
;
; NOTE: The code assumes 32-bit flat model protected mode for simplicity. i.e. ES contains 0 and EDI
;          accesses the entire memory space.
;===========================================================================

TEXT    segment byte public  'CODE'
        assume  cs:TEXT

     ; Following is the structure by which all parameters are passed.

params STRUCT
     erase_addr        DD      ?        ; base of block or device to erase.
     write_addr        DD      ?        ; address to write to.
     write_base        DD      ?        ; base address of block written to.
     read_addr         DD      ?        ; address to read from.
     read_base         DD      ?        ; base address of block read from.
     read_id           DD      ?        ; address to read device code from.
     block_lock_addr   DD      ?        ; base address of block to lock.
     master_addr       DD      ?        ; base address for the Master Lock-Bit
     data_addr         DD      ?        ; address of data to write.
     data              DB      ?        ; data byte to write.
params ENDS

;===========================================================================
; Defines
;===========================================================================

;===========================================================================
; Error Codes
;===========================================================================
NO_ERROR                DW   0
VPP_ERROR               DW   1
WRITE_ERROR                     DW    2
ERASE_ERROR                     DW    3
BLOCK_PROTECTION_ERROR    DW    4              ; ONLY valid for the 28F0xxSC.
COMMAND_SEQ_ERROR               DW    5
```

```
;============================================================================
; MACRO          set_pin
; This macro pushes parameters needed for the set_pin routine, calls pin_control, and then pops those
; parameters. set_pin is an implementation-dependent function which enables high voltage on the RP# pin.
; This macro is used in the Master and Block Lock-Bit procedures which are ONLY valid for the 28F0xxSC.
;
; Data needed at the beginning of this macro:
; level:         level to set pin
; Trashes:       CL
;============================================================================
set_pin   MACRO   level
        push    level                       ; Push logic level of pin
        call    near ptr pin_control        ; Call pin_control
        pop     CL                          ; Pop off parameters
ENDM

;============================================================================
; PROCEDURE   block_erase
; This procedure erases a 64K byte block on the 28F008SA and 28F0xxSC.
;
; Param fields needed:
;                erase_address: offset of base block to erase
; Output         AL: holds SR information
;============================================================================
block_erase  proc   near
        mov     EDI,params.erase_addr
        mov     BYTE PTR ES:[EDI],020H      ; Block Erase command
        mov     ES:[EDI],0D0H               ; Block erase confirm

        ; Note that it is not strictly necessary to write an erase command to the base of a block any
        ; address within the block will do.

WSM_busy:
        mov     AL,ES:[EDI]                 ; Read SR.
        test    AL,080H                     ; If SR.7 = 0, test sets ZF.

        ; Erase may be suspended here to read/write from/to a different block.

        jz      short WSM_busy              ; Loop while ZF is set.
        mov     BYTE PTR ES:[EDI],050H      ; Clear Status Registers command and place into read mode.
        ret                                 ; Return to calling routine.
block_erase  endp
```

```
;============================================================================
; PROCEDURE     byte_write
; This procedure writes a byte to the 28F008SA and 28F0xxSC.
;
; Param fields needed:
;                   params.data: data word to be written
;                   params.write_addr: offset address to write
; Output:           AL: holds SR information
;============================================================================
byte_write   proc    near
        mov     EDI,params.write_addr
        mov     BYTE PTR ES:[EDI],040H      ; Write To Flash command
        mov     ES:[EDI],params.data        ; Write data to 28F0xxSC.
WSM_busy1:
        mov     AL,ES:[EDI]                 ; Read SR
        test    AL,080H                     ; If SR.7 = 0, test sets ZF.

        ; Byte write may be suspended here to read from a different block.

        jz      short WSM_busy1             ; Loop while ZF is set.
        mov     BYTE PTR ES:[EDI],050H      ; Clear Status Registers command and place into read mode.
        ret                                 ; Return to calling routine.
byte_write   endp

;============================================================================
; PROCEDURE   erase_suspend_to_read
; This procedure suspends an erase operation to do a read. The procedure assumes that an erase
; operation is underway. This procedure is valid for the 28F008SA and 28F0xxSC.
;
; Param fields needed:
;                   params.read_addr: offset address to read
; Output:           AL: holds SR information
;                   CL: data read from the address in params.read_addr
;============================================================================
erase_suspend_to_read   proc    near
        mov     EDI,params.read_addr        ; Set up offset of erase address.
        mov     BYTE PTR ES:[EDI],0B0H      ; Erase Suspend command
WSM_busy2:
        mov     AL,ES:[EDI]                 ; Read SR from any address.
        test    AL,080H                     ; If SR.7 = 0, test sets ZF.
        jz      short WSM_busy2             ; Loop while ZF is set.
        mov     BYTE PTR ES:[EDI],0FFH      ; Read Flash command
        mov     CL,ES:[EDI]                 ; Do actual read; put result in CL.

        ; Arbitrary number of reads can be done here.

        mov     BYTE PTR ES:[EDI],070H      ; Read SR command
        mov     AL,ES:[EDI]                 ; Read SR from any address.
        test    AL,040H                     ; If SR.6 = 0, indicating that there is no erase suspended.
        jz      short continue              ; Jump to continue if ZF is set.
        mov     BYTE PTR ES:[EDI],0D0H      ; Erase Resume command
continue:
        ret                                 ; Return to calling routine.
erase_suspend_to_read   endp
```

```
;============================================================================
; PROCEDURE     erase_suspend_to_write
; This procedure suspends an erase operation to do a byte write. The procedure assumes that an erase
; operation is underway. ONLY valid for the 28F0xxSC.
;
; Param fields needed:
;                       params.write_addr: offset block to erase
;                       params.params.data: offset address to read
; Output:               AL: holds SR information
;============================================================================
erase_suspend_to_write   proc    near
        mov     EDI,params.write_addr           ; Set up offset of erase address.
        mov     BYTE PTR ES:[EDI],0B0H          ; Erase Suspend command
WSM_busy3:
        mov     AL,ES:[EDI]                     ; Read SR from any address.
        test    AL,080H                         ; If SR.7 = 0, test sets ZF.
        jz      short WSM_busy3                 ; Loop while ZF is set.
        mov     BYTE PTR ES:[EDI],040H          ; Byte Write command
        mov     ES:[EDI],params.data            ; Write data.
WSM_busy4:
        mov     AL,ES:[EDI]                     ; Read SR
        test    AL,080H                         ; If SR.7 = 0, test sets ZF.
        jz      short WSM_busy4                 ; Loop while ZF is set.

        ; Arbitrary number of writes can be done.

        mov     AL,ES:[EDI]                     ; Read SR from any address.
        test    AL,040H                         ; If SR.6 = 0, indicating that there is no erase suspended.
        jz      short continue1                 ; Jump to continue if ZF is set.
        mov     BYTE PTR ES:[EDI],0D0H          ; Erase Resume command
continue1:
        ret                                     ; Return to calling routine.
erase_suspend_to_write   endp
```

```
;============================================================================
; PROCEDURE     write_suspend_to_read
; This procedure suspends a byte write operation to do a read. The procedure assumes that a
; byte write operation is underway. ONLY valid for the 28F0xxSC.
;
; Param fields needed:
;                    params.read_addr: offset address to read
; Output:            AL: holds SR information
;                    CL: data read from the address in params.read_addr
;============================================================================
write_suspend_to_read   proc    near
        mov     EDI,params.read_addr        ; Set up offset of erase address.
        mov     BYTE PTR ES:[EDI],0B0H      ; Erase Suspend command
WSM_busy5:
        mov     AL,ES:[EDI]                 ; Read SR from any address.
        test    AL,080H                     ; If SR.7 = 0, test sets ZF.
        jz      short WSM_busy5             ; Loop while ZF is set.
        mov     BYTE PTR ES:[EDI],0FFH      ; Read Flash command
        mov     CL,ES:[EDI]                 ; Do actual read; put result in CL.

        ; Arbitrary number of reads can be done here.

        mov     BYTE PTR ES:[EDI],070H      ; Read SR command
        mov     AL,ES:[EDI]                 ; Read SR from any address.
        test    AL,004H                     ; If SR.2 = 0, indicating that there is no erase suspended.
        jz      short continue2             ; Jump to continue if ZF is set.
        mov     BYTE PTR ES:[EDI],0D0H      ; Erase Resume command
continue2:
        ret                                 ; Return to calling routine.
write_suspend_to_read   endp
```

```
;===========================================================================
; PROCEDURE     set_block_lock_bit
; This procedure sets a block lock-bit on the 28F0xxSC. ONLY valid for the 28F0xxSC.
;
; Param fields needed:
;               params.block_lock_addr: offset of base block to lock
; Output:       AL: holds SR information
;===========================================================================
set_block_lock_bit      proc    near
        mov     EDI,params.block_lock_addr      ; Set up offset of address.

        ; If Master lock-bit is set, RP = Vhh.
        ; set_pin       1                       ; Enable high voltage on to RP#.

        mov     BYTE PTR ES:[EDI],060H          ; Set Block Lock-Bit command.
        mov     BYTE PTR ES:[EDI],001H          ; Block lock-bit confirmation command.
WSM_busy6:
        mov     AL,ES:[EDI]                     ; Read SR from any address.
        test    AL,080H                         ; If SR.7 = 0, test sets ZF.
        jz      short WSM_busy6                 ; Loop while ZF is set

        ; If Master lock-bit was set return RP to Vih.
        ; set_pin       0                       ; Disable high voltage on to RP#.

        mov     BYTE PTR ES:[EDI],050H          ; Clear Status Registers command and place into read mode.
        ret                                     ; Return to calling routine.
set_block_lock_bit      endp

;===========================================================================
; PROCEDURE     set_master_lock_bit
; This procedure sets the master lock-bit on the 28F0xxSC. ONLY valid for the 28F0xxSC.
;
; Param fields needed:
;               params.master_addr: offset of base Master Lock-Bit
; Output:       AL: holds SR information
;===========================================================================
set_master_lock_block   proc    near
        mov     EDI,params.master_addr          ; Set up offset of address.
        set_pin 1                               ; Enable high voltage on to RP#.
        mov     BYTE PTR ES:[EDI],060H          ; Set Master Lock-Bit command.
        mov     BYTE PTR ES:[EDI],0F1H          ; Master lock-bit confirmation command.
WSM_busy7:
        mov     AL,ES:[EDI]                     ; Read SR from any address.
        test    AL,080H                         ; If SR.7 = 0, test sets ZF.
        jz      short WSM_busy7                 ; Loop while ZF is set
        set_pin 0                               ; Disable high voltage on to RP#.
        mov     BYTE PTR ES:[EDI],050H          ; Clear Status Registers command and place into read mode.
        ret                                     ; Return to calling routine.
set_master_lock_bit             endp
```

```
;============================ =============================================
; PROCEDURE     clear_block_lock_bits
; This procedure resets the block lock-bits on the 28F0xxSC. ONLY valid for the 28F0xxSC.

;
; Param fields needed:
;                       params.block_lock_addr: offset of base block
; Output:               AL: holds SR information
;==========================================================================
clear_block_lock_bits      proc    near
        mov         EDI,params.block_lock_addr    ; Set up offset of address.

        ; If Master lock-bit is set, RP = Vhh.
        ; set_pin   1                             ; Drive high voltage on to RP#.

        mov         BYTE PTR ES:[EDI],060H        ; Clear Block Lock-Bits command
        mov         BYTE PTR ES:[EDI],0D0H        ; Confirmation command
WSM_busy8:
        mov         AL,ES:[EDI]                   ; Read SR from any address.
        test        AL,080H                       ; If SR.7 = 0, test sets ZF.
        jz          short WSM_busy8               ; Loop while ZF is set.

        ; If Master lock-bit was set return RP to Vih.
        ; set_pin   0                             ; Drive high voltage on to RP#.

        mov         BYTE PTR ES:[EDI],050H        ; Clear Status Registers command and place into read mode.
        ret                                       ; Return to calling routine.
clear_block_lock_bits                endp

;==========================================================================
; PROCEDURE     read_identifier_codes
; This procedure provides access to device codes. This procedure is valid for the 28F008SA and 28F0xxSC.
;
; Param fields needed:
;                       params.read_id: offset of base id
; Output:               CL: data read from the device
;==========================================================================
read_identifier_codes      proc    near
        mov         EDI,params.read_id            ; Set up offset of address.
        mov         BYTE PTR ES:[EDI],090H        ; Read Identifier Codes command.
        mov         CL,ES:[EDI]                   ; Device code data.
        mov         BYTE PTR ES:[EDI],0FFH        ; Read Array command.
        ret                                       ; Return to calling routine.
read_identifier_codes      endp
```

```
;==========================================================================
; PROCEDURE     SR_full_status_check
; This procedure performs a full status register check. It is valid for byte write, block erase, block-bit set
; and block-bit reset operations. This procedure is valid for the 28F008SA and 28F0xxSC.
;
; Note: This procedure assumes the status register data reside in the AL. This information is place in AL
; after the completion of each operation. If an error is detected, the previous operation should be
; executed again.
;
; Output:       CL : error code
;==========================================================================
SR_full_status_check      proc    near
vpp_check:
        test      AL,008H                       ; If SR.3 = 0, test sets ZF.
        jz        device_protection_check       ; Next check if ZF is set.
        mov       CL,VPP_ERROR                  ; Place error code in CL.
        jmp       continue3                     ; Jump to end of SR check.
device_protection_check:
        test      AL,002H                       ; If SR.1 = 0, test sets ZF.
        jz        command_seq_check             ; Next check if ZF is set.
        mov       CL,BLOCK_PROTECTION_ERROR
                                                ; Place error code in CL. ONLY valid for the 28F0xxSC
        jmp       continue3                     ; Jump to end of SR check.
command_seq_check:
        test      AL,010H                       ; If SR.4 = 0, test sets ZF.
        jz        block_erase_check             ; Next check if ZF is set.
        test      AL,020H                       ; If SR.5 = 0, test sets ZF.
        jz        write_check                   ; Next check if ZF is set.
        mov       CL,COMMAND_SEQ_ERROR
                                                ; Place error code in CL.
        jmp       continue3                     ; Jump to end of SR check.
block_erase_check:
        test      AL,020H                       ; If SR.5 = 0, test sets ZF.
        jz        no_error_detected             ; No error detected if ZF is set.
        mov       CL,ERASE_ERROR               ; Place error code in CL.
        jmp        continue3                    ; Jump to end of SR check.
write_check:
        mov       CL,BYTE_WRITE_ERROR    ; Place error code in CL.
        jmp       continue3                     ; Jump to end of SR check.
no_error_detected:
        mov       CL,NO_ERROR                   ; Place error code in CL.
continue3:
        ret                                     ; Return to calling routine.
SR_full_status_check      endp
```

APPENDIX C
ADDITIONAL INFORMATION

RELATED INFORMATION (1,2)

Order Number	Document/Tool
290592	*28F004SC/28F004SC-L 4-Mbit (512 KB x 8) SmartVoltage FlashFile™ Memory Datasheet*
290577	*28F008SC 8-Mbit (1 MB x 8) SmartVoltage FlashFile™ MemoryDatasheet*
290576	*28F008SC-L 8-Mbit (1 MB x 8) SmartVoltage FlashFile™ MemoryDatasheet*
290429	*28F008SA 8-Mbit (1-Mbit x 8) FlashFile™ Memory Datasheet*
290435	*28F008SA-L (1-Mbit x 8) FlashFile™ Memory Datasheet*
290593	*28F016SC/28F016SC-L 16-Mbit (2 MB x 8) SmartVoltage FlashFile™ Memory Datasheet*
292094	*AP-359 28F008SA Hardware Interfacing*
292099	*AP-364 28F008SA Automation and Algorithms*
292180	*AP-625 28F008SC Compatibility with 28F008SA*
292183	*AB-64 4-, 8-, 16-Mbit Byte-Wide FlashFile™ Memory Family Overview*
297647	Flash SOFTWAREBuilder
Contact Intel/Distribution Sales Office	TimingDesigner* Files for Intel's Byte-Wide FlashFile™ Memory Family
Contact Intel/Distribution Sales Office(2)	Schematic Symbols for Intel's Byte-Wide FlashFile™ Memory Family
Contact Intel/Distribution Sales Office(2)	VHDL and Verilog Models for Intel's Byte-Wide FlashFile™ Memory Family
Contact Intel/Distribution Sales Office(2)	iBIS Models for Intel's Byte-Wide FlashFile™ Memory Family

NOTE:

1. Please call the Intel Literature Center at (800) 548-4725 to request Intel documentation. International customers should contact their local Intel or distribution sales office.
2. Visit Intel's World Wide Web home page at http://www.Intel.com for technical documentation and tools.