SGS-THOMSON MICROELECTRONICS

APPENDIX D

APPLICATION NOTE

PROGRAMMING FLASH MEMORY OF THE ST10F166

by S. Fruhauf, G. Petrosino

INTRODUCTION

The ST10F166 high end microcontroller with on-chip Flash Memory fulfills the requirements of applications requiring an update to a part or all the program code. The block erase capability is also of use during the application development stage or for program updating. For data acquisition, the ST10F166 allows the programming of 16 or 32 bits data independently.

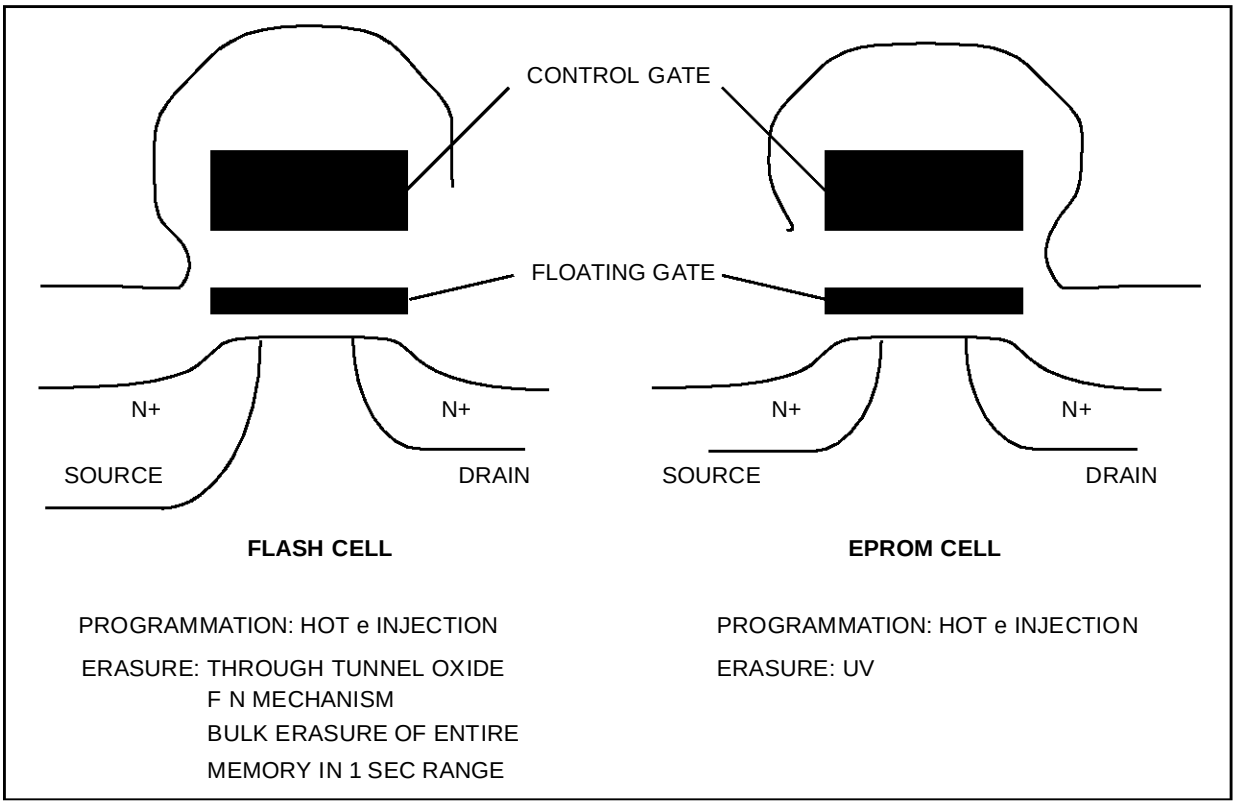
Operations on the Flash memory are under software control. Erasure or programming is a simple procedure, however precautions must be taken to prevent damage to the ST10F166.

This application note describes the basic characteristics of the Flash memory cell, and the different algorithms used for erasure and programming.

FUNDAMENTALS OF FLASH MEMORY

The Flash memory included in the ST10F166 combines the EPROM programming mechanism with electrical erasability (like EEPROM) to create a highly reliable and cost effective memory. A Flash memory cell consists of a single transistor with a floating gate for charge storage like EPROM, the main difference being that Flash memory uses a thinner gate oxide.

Figure 1. SGS-THOMSON Flash Cell VS Eprom Cell



FUNDAMENTALS OF FLASH MEMORY (Cont'd)

The programming mechanism of a cell is based on hot electron injection. This means that the cell control gate and drain are set to a high voltage and the cell source is grounded. The high voltage on the drain generates "hot" electrons through the channel, and the high voltage on the control gate traps the free electrons into the floating gate.

The cell erase mechanism is based on "Fowler-Nordheim" tunnelling. This means that the cell control gate is grounded, the cell drain is disconnected and the high voltage is applied to the cell source. The high electric field between the floating gate and the source removes electrons from the floating gate.

Unlike standard EEPROM memory, where individual bytes can be erased, the Flash memory of the ST10F166 performs erase on blocks where the high voltage is applied to all cells simultaneously.

Figure 2. Flash Memory Cell Programming Mechanism

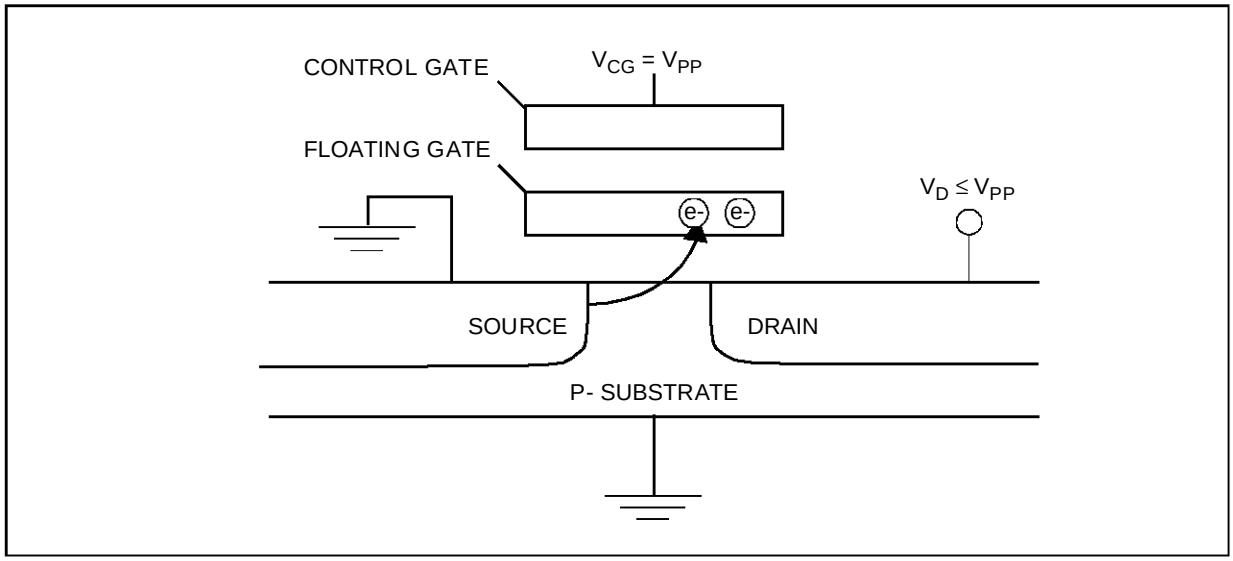
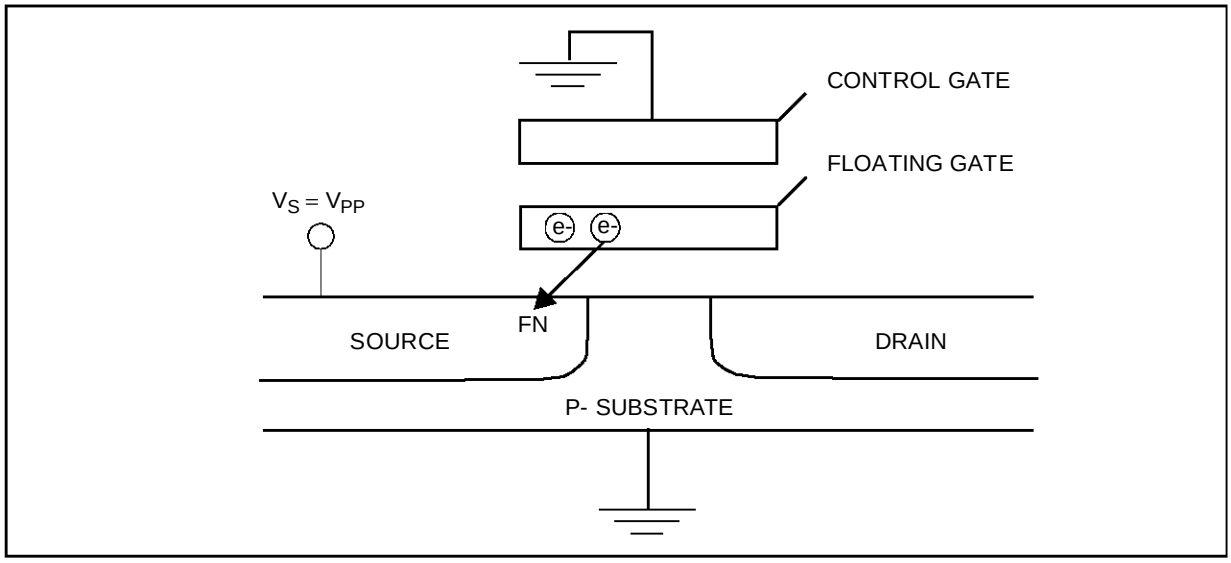


Figure 3. Flash Memory Cell Erase Mechanism



FUNDAMENTALS OF FLASH MEMORY (Cont'd)

A difficulty with Flash memory concerns the requirement to set all the cells of a block to a minimum threshold level suitable for programming and erase operations. Applying a new erasing pulse to a block with a different storage level on each cell (a different threshold level), can be very dangerous for the functionality of the Flash memory.

A fast erasing cell may have a threshold voltage too low or negative, in this case the transistor is always on and is read at "one". This has the effect of leakage on other cells placed on the same array column. Thus all cells of the column will be read at "one" instead of "zero".

To avoid this, the user must equalize the amount of charge on each cell by performing a programming operation before every erasure.

For increased reliability, the SGS-THOMSON Flash memory technology, combined with the use of the Erase-verify PRESTO F algorithm, provides a tight erase threshold voltage distribution, generating sufficient margin to the faster erasing cell and the minimum threshold level required to read a "one" data value.

ERASE & PROGRAMMING CONTROL

To simplify control of the Flash operation modes, the ST10F166 Flash memory includes a Flash Control Register (FCR) used for all programming or erase operations. Mapped virtually into the Flash address space, FCR is not accessible during normal memory access modes and must be unlocked by a special instruction sequence.

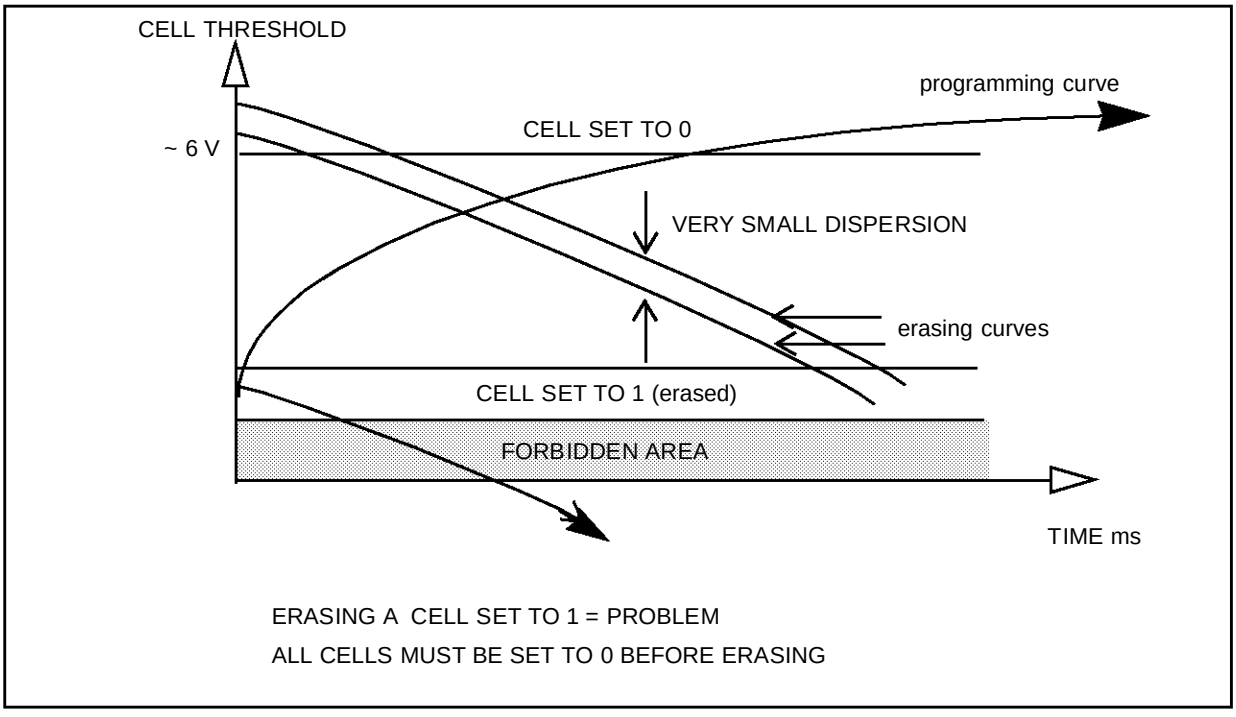
To avoid unpredictable programming or erase operation on the Flash memory, the ST10F166 provides several levels of security:

First level: the user must perform a special sequence to enable the FCR and to enter into the program mode.

Second level: to operate on the Flash memory, two steps are necessary. First the user must set up the FCR in the desired configuration, second the operation begins ONLY with the appropriate command.

Third level: during the program mode, two bits of FCR (VPPRIV & FCVPP) indicate to the user the status of VPP (the high voltage) before and during an operation. It is advisable for the user to test them in the erase or programming routine.

Figure 4. Flash Erasure



THE PRESTO F PROGRAM WRITE ALGORITHM

The following section explains the Presto F Program Write Algorithm shown in figure 5 for a better understanding of the user. For high reliability, it is necessary to follow this algorithm to program the Flash memory.

It is considered that the EBC1/VPP pin has been switched to the VPP supply after reset, and the program mode has been unlocked.

Before performing the unlock sequence, remember that the interrupts should be disabled, bit IEN of PSW cleared. After exiting the write mode, bit IEN should be set, to enable the interrupts again.

– READ VPPRIV

After setting the writing mode, a delay of 10 µs must be inserted to allow the device to set its internal high voltage signals. Then, before starting the proper programming operation, the VPP level must be checked. VPPRIV is at the "one" level if VPP is correct. If it is not the programming algorithm must be held until VPP reaches its correct value or until the VPP supply is set correctly.

```
mov  fcrrd,  FCR                ; read FCR
jnb  vppriv, vpp_fail           ; test if VPP is high
```

– PCOUNT = 0

Initialization of PCOUNT variable to zero. The Presto F Program Write algorithm consist of applying several pulses to each word until a correct verify occurs. The maximum number of programming pulses is fixed and depends on the CPU clock. The maximum cumulated programming time is 2.5 ms for the ST10F166B. If this limit is reached the word will never be programmed.

In case of several words to program, an Address variable can be initialized.

```
mov lpcnt,  #ALL0               ; reset algo. loop counter
```

– Write Programming Setup command into FCR

First step for programming:

Set FCR with the desired value.

Set FWE bit to enable programming operation.

Clear CKCTL0 & CKCTL1 bits to define the programming pulse width: 6.4 µs at 20MHz CPU clock.

Choose the configuration:

Set WDWW bit for double word programming.

Clear WDWW bit for word programming.

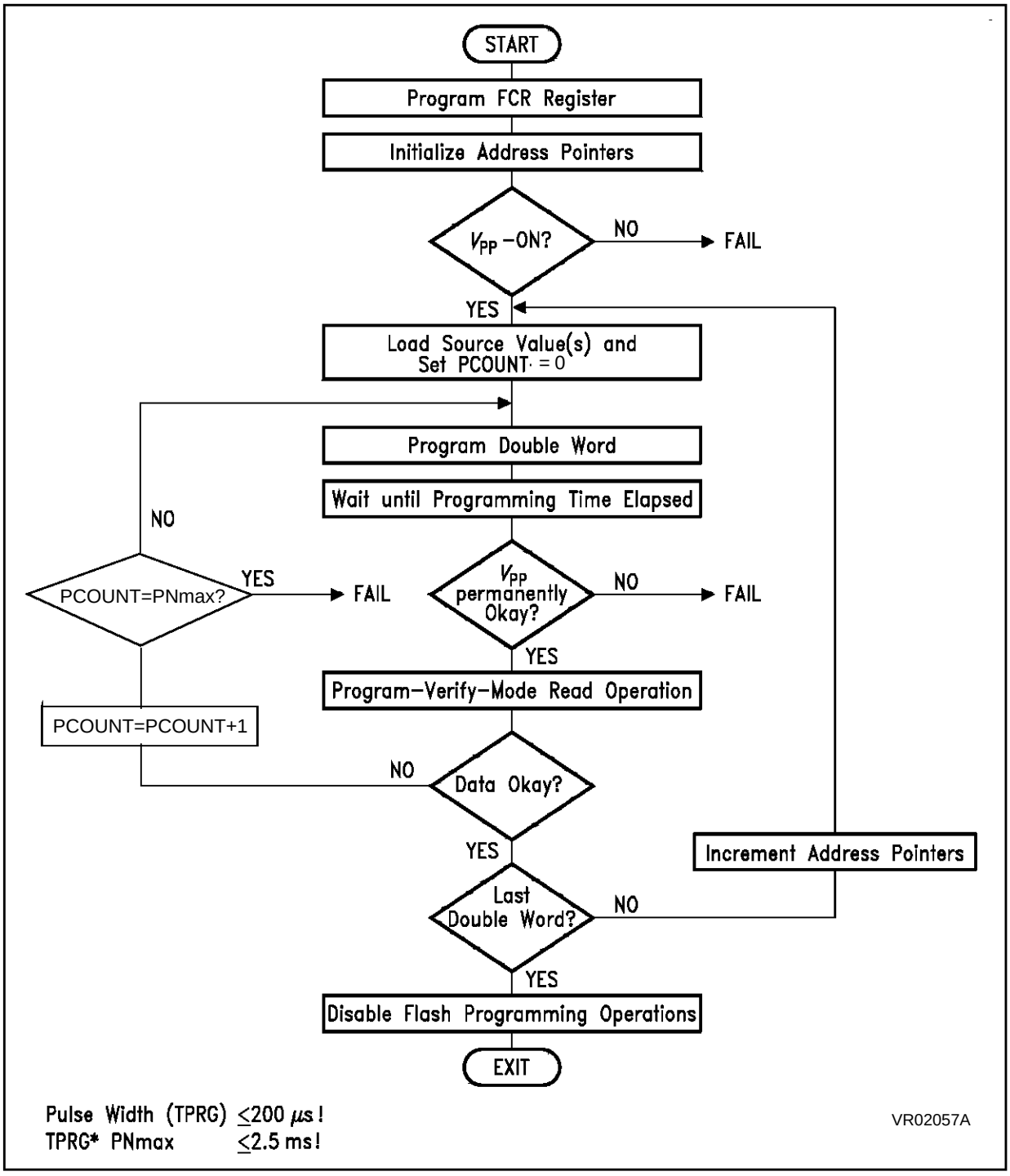
Set FWMSET bit for program mode.

Take care at this point as this step prepares the device for programming but does not activate the process.

```
mov    fcrval, #ALL0            ; reset FCR data value
bset   fwe                      ; FWE=1 define programming operation
bclr   ckctl0                   ; CKCTL0=0 )
bclr   ckctl1                   ; CKCTL1=0 ) define the pulse width
bset   wdww                     ; WDWW=1 define 32-bit configuration
bset   fwmset                   ; FWMSET=1 confirm write mode
mov    FCR,   fcrval            ; load FCR with the desired value
```

THE PRESTO F PROGRAM WRITE ALGORITHM (Cont'd)

Figure 5. PRESTO F Program Write Algorithm



THE PRESTO F PROGRAM WRITE ALGORITHM (Cont'd)

– Write valid data address

The following command starts automatically the programming process.

For word programming:

```
mov    [addrev],datal          ; programming command
```

For double word programming:

```
mov    [addrev],datal          ; programming command, even word
mov    [addrev],datah          ; programming command, odd word
```

– WAIT PT

The programming time (PT) depends on the bits CKCTL0 & CKCTL1 of FCR (see setting of FCR). The end of programming can be detected by polling on the FBUSY bit of FCR.

FBUSY set to "1" indicates programming is in progress.

FBUSY cleared indicates programming has ended

```
waitpr:  mov   fcrrd,  FCR       ; read FCR
         jb     busy,   waitpr   ; jump if programming is not ended
```

– FCVPP = "0" ?

To have a well programmed word, it is important to check if VPP was at the correct value during programming. This is indicated by the status of the FCVPP bit of FCR.

If FCVPP = "0" there was no problem, continue with the algorithm.

If FCVPP = "1" VPP was not enough high during programming, jump to the user defined VPP-fail routine. An example of this routine could be a reset of FCR, then a new test of the VPPRIV bit and, if all is correct, redo a programming operation, otherwise exit the programming routine.

```
jb     fcvpp,  vpp_fail          ; jump if FCVPP is set
```

– PROGRAM VERIFY READ

To check if the word is correctly programmed, a comparison must be performed with the data expected. A Program Verify Read will check the cell margin of the word.

Perform twice the same reading instruction separated by a time of 4 µs.

This sequence must be made to get a correct reading of the word. This time corresponds to an internal switching of signals.

THE PRESTO F PROGRAM WRITE ALGORITHM (Cont'd)

– COMPARE WITH DATA EXPECTED

This step can be merged with the Program Verify Read step as the comparison instruction is a read instruction. If the data programmed at the address given is different from the data expected, an extra programming operation must be performed (the next step).

```
cmp    datal,  [addrev]        ; first instruction for PVM (even)
calla   cc_UC,  wait4          ; 4µs
cmp    datal,  [addrev]        ; second instruction for PVM
jmpr   cc_NZ,  prog            ; jump if the word is not correctly
                               ; programmed, restart programming
cmp    datah,  [addrod]        ; first instruction for PVM (odd)
calla   cc_UC,  wait4          ; 4µs
cmp    datah,  [addrod]        ; second instruction for PVM
jmpr   cc_NZ,  prog            ; jump if the word is not correctly
                               ; programmed, restart programming
```

– PCOUNT = PN max

For each new programming operation the PCOUNT variable must be incremented; at this point, it must be tested to verify whether the PN max limit has been reached or not. If yes, the word will never be programmed and the algorithm should be exited from. In this case a possible solution is to change the address of the word to program.

```
add    lpcnt,  #01h            ; increment the algo. loop counter
cmp    lpcnt,  #MAXLOOP1       ; compare to the limit
jmpr   cc_Z,   prg_fail        ; jump if limit has been reached
```

– LAST ADDRESS

In case of consecutives words to program, check the address variable to know if the last address has been reached. If not, increment the address variable and start another programming operation from the beginning of the algorithm.

– WRITE FWE = "0"

All the words are programmed, exit the presto F program Write algorithm. All programming or Program Verify Read operation are stopped by a reset of FCR register (especially FWE bit cleared). Normal reading of the Flash memory can be performed only after this step.

```
mov   fcrval,    #ALLO
mov    FCR,      fcrval        ; reset FCR and exit program mode
```

THE PRESTO F ERASE ALGORITHM

The following section explains the Presto F Erase Algorithm shown in figure 6 but all parts already described in the previous section will not be explained again. Note that an entire block will be erased instead of one or two words as programming.

– ALL WORDS AT 0000h

Prior to erasure, program all block addresses to 0000h. This step equalizes the charge on each memory cell of the block. Erasure removes charge from all memory cells regardless of their previous state, and not performing this programming will drive cells previously at a "one" to be stuck at "one" (as explained in the Fundamentals of Flash memory section).

The Presto F Program Write Algorithm must be used for this block programming. (refer to the previous section).

– VARIABLE INITIALIZATION

Initialize two variables:

PCOUNT = 0 for the pulse count, and the address variable to the first address of the block. N can be incremented from 0 to EN max. The maximum cumulated erase time is 30s.

Note: with each pulse, all the block will be erased.

– WRITE ERASE SETUP COMMAND INTO FCR

As for programming, this step only prepares the device for erasure.

Set FWE,FEE bits to enable erasure.

Clear CKCTL0 & set CKCTL1 bits to define a the erasing pulse width: 1.64ms at 20MHz CPU clock.

Choose the block configuration for erasure (BE0,BE1).

Clear WDWW bit.

Set FWMSET bit for write mode.

– WRITE ERASE COMMAND

Perform the specific instruction to start automatically the erase process.

```
mov    [fl_scan],fl_scan    ; erase command, erasure start
```

– WAIT ET

The erasing time (ET) depends on the bits CKCTL0 & CKCTL1 of FCR (see setting of FCR). The end of erasure can be detected by polling on the FBUSY bit of FCR.

FBUSY set to "1" indicates erase is in progress.

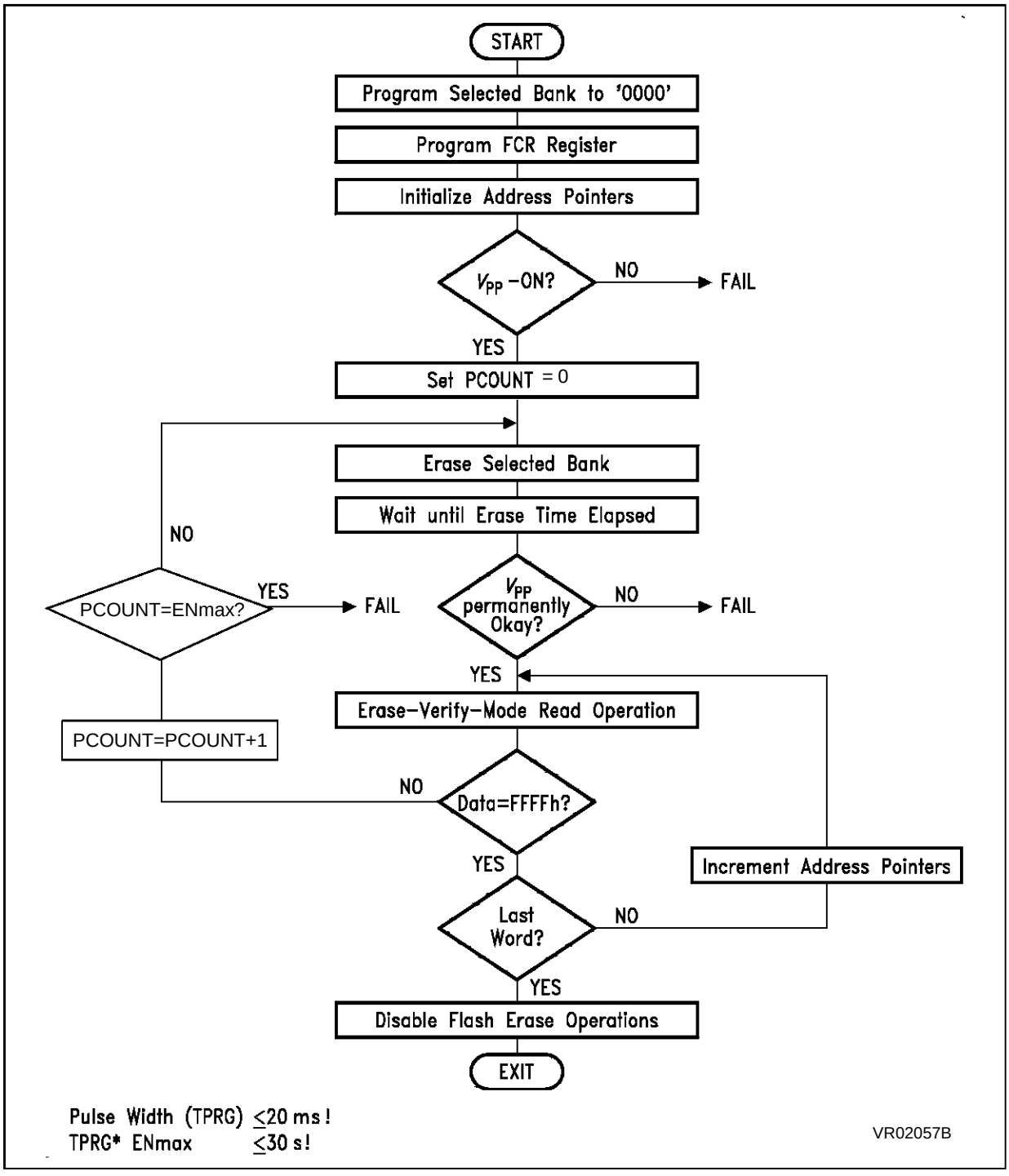
FBUSY cleared indicates erase has ended.

– FCVPP = "0" ?

Test VPP to detect any discontinuity in VPP during erasure (see previous section).

THE PRESTO F ERASE ALGORITHM (Cont'd)

Figure 6. PRESTO F Erase algorithm



THE PRESTO F ERASE ALGORITHM (Cont'd)

– ERASE VERIFY READ

This mode, equivalent to the Program Verify Read, guarantees a improved cell margin of a word.

Read the data at the address given by the address variable twice with the same instruction separated by a time of 4 µs.

– COMPARE DATA = FFFFh

Compare the data read to FFFFh. If it equals FFFFh, this address has been erased; continue verification until the last address of the block has been verified. If not, increment PCOUNT variable. Apply a new erasing pulse to the block, and continue until the data is correctly checked or the maximum erasing pulse count has been reached.

```
read_ff: cmp   all1,  [fl_scan]        ; first instruction for EVM
calla  cc_UC,  wait4                   ; 4µs
cmp    all1,  [fl_scan]                ; second instruction for EVM
jmpr   cc_NZ,  erase                   ; jump if the word is not erased
```

– LAST ADDRESS

Check the address variable to see if the last address of the block has been reached. If not, increment the address variable and start another

Erase Verify Read.

```
add    fl_scan,#02h                    ; increment the bank pointer
cmp    fl_scan,#FL_SIZE                ; compare to the last bank address
jmpr   cc_NZ,  read_ff                 ; jump to verify the next address
```

– WRITE FWE = "0"

All the block is erased, exit the Presto F Erase algorithm stopping all erasure or Erase Verify Read operations with a reset of FCR register (especially FWE, FEE bits cleared)

Normal reading of Flash memory can be performed only after this step.

RULES FOR USING THE FLASH MEMORY

– Follow the Presto F Algorithm and verify its correct implementation. This will ensure that all the block has been programmed before erasure to minimize internal stresses on the memory cells, and to perform writing operation in a fast and reliable way.

– Verify VPP status before and after every writing operation.

BASIC ROUTINES FOR ERASURE AND PROGRAMMING

This section describes basic routines which can be helpful for the user.

Erasure, 32-bit programming and 16-bit programming routines are written as subroutines to allow easy inclusion in a user program.

The following routines are written in a way to clarify the operations as well as possible.

The initial conditions are described at the head of the routine, if needed.

Table 1. Recommended CKCTL values depending on the CPU clock used

F_{CPU}	CKCTL		TPRG		N_{MAX}	
	PROG.	ERASE	PROG.	ERASE	PROG.	ERASE
1MHz	00	01	128µs	2.05ms	19	14648
10 MHz	00	10	12.8µs	3.28ms	195	9157
16 MHz	00	10	8µs	2.05ms	312	14648
20 MHz	00	10	6.4µs	1.64ms	390	18315

```
;  VARIABLE DEFINITIONS FOR THE FLASH MEMORY ROUTINES

      ALL0            equ    00000h  ;constant 0
      ALL1            equ    0FFFFh  ;constant FFFF
      BLK_START       equ    03000h  ;first address of bank 1
      FL_SIZE         equ    03000h  ;size of bank 1
      FCR             equ    07FFEh  ;dummy address chosen for FCR
      ADDREV          equ    0000Ch  ;address even (least significant bit)
      ADDROD          equ    0000Eh  ;address odd (most significant bit)
      DATAH           equ    09753h  ;data to program to odd address
      DATAL           equ    08642h  ;data to program to even address
      MAXLOOP1        equ    00186h  ;limit of the programming loop
      MAXLOOP2        equ    0478Bh  ;limit of the erase loop
      UNLOCK          equ    01000h  ;data to unlock the program mode
      WAIT4           equ    0000Bh  ;loop 4 µs
      WAIT10          equ    0001Fh  ;loop 10 µs
      addrev          LIT    'R0'    ;even address pointer
      fcrval          LIT    'R1'    ;register for FCR writing
      addrod          LIT    'R2'    ;odd address pointer
      datal           LIT    'R3'    ;register with first data
      datah           LIT    'R4'    ;register with second data
      lpcnt           LIT    'R5'    ;algorithm loop counter
      all1            LIT    'R6'    ;register used in EVM
      unlock          LIT    'R7'    ;register used to unlock
      val10u          LIT    'R8'    ;counter 10µs
      val4u           LIT    'R9'    ;counter 4µs
      wait_cnt        LIT    'R10'   ;register to control wait loop
      fl_scan         LIT    'R13'   ;bank address pointer
      fcrrd           LIT    'R15'   ;register for FCR reading

      fwe             LIT    'R1.0'  ;FCR FWE bit
      fee             LIT    'R1.1'  ;FCR FEE bit
      ckctl0          LIT    'R1.5'  ;FCR CKCTL0 bit
      ckctl1          LIT    'R1.6'  ;FCR CKCTL1 bit
      wdww            LIT    'R1.7'  ;FCR WDWW bit
      be0             LIT    'R1.8'  ;FCR BE0 bit
      be1             LIT    'R1.9'  ;FCR BE1 bit
      busy            LIT    'R15.2' ;FCR BUSY bit
      fcvpp           LIT    'R15.3' ;FCR FCVPP bit
      vppriv          LIT    'R15.4' ;FCR VPPRIV bit
```

```
;ERASE ROUTINE: erasure of bank 1, this routine assumes that the bank
;_______________ was previously programmed to 0000h before erasure

;************* INITIAL CONDITIONS: ************************************
;
;   ALL WORDS IN BANK 1 HAVE TO BE PROGRAMMED AT ”ZERO”
;   WITH THE PRESTO F PROGRAM WRITE ALGORITHM
;
;************************************** ******************************
f_erase:
      ;
      ; REGISTERS INITIALIZATION
      ;
      mov    lpcnt,  #ALL0            ; reset algo. loop counter
      mov    fcrval, #ALL0            ; reset FCR data value
      mov    unlock, #UNLOCK          ; load unlock data
      mov    val10u, #WAIT10          ; load 10µs loop data
      mov    val4u,  #WAIT4           ; load  4µs loop data
      mov    wait_cnt,#ALL0           ; reset wait loop counter
      mov    all1,   #ALL1            ; set R2 to FFFFh
      mov    fl_scan,#BLK_START       ; load first bank address
      ;
      ; UNLOCK SEQUENCE FOR ENTERING IN THE PROGRAM MODE
      ;
      mov    FCR,    unlock           ; first instruction
      mov    [unlock],unlock          ; second instruction of unlock
                                      ; sequence to enter in the program mode
      calla   cc_UC,  wait10          ; time out 10 µs to set internal signals
      ;
      ; FCR SET UP FOR ERASURE
      ;
      bset   fwe                      ; FWE=1 ) these two instructions
      bset   fee                      ; FEE=1 ) define the erasure
      bclr   ckctl0                   ; CKCTL0=0 )
      bset   ckctl1                   ; CKCTL1=1 ) define the pulse
      bclr   wdww                     ; WDWW=0
      bset   be0                      ; BE0=1 )
      bclr   be1                      ; BE1=0 ) select bank 1
      bset   fwmset                   ; FWMSET=1 enable program mode
      mov    FCR,    fcrval           ; load FCR set up
```

```
      ;
      ; TEST VPP
      ;
      mov    fcrrd,  FCR              ; read FCR
      jnb    vppriv, vpp_fail         ; test if VPP is high
      ;
      ; FLASH ERASURE
      ;
erase:
      add    lpcnt,  #01h             ; increment the algo. loop counter
      cmp    lpcnt,  #MAXLOOP2        ; compare to the limit
      jmpr   cc_Z,   eras_fail        ; jump if limit has been reached
      mov    [fl_scan],fl_scan        ; erase command, erasure start
waiter: mov    fcrrd,  FCR            ; read FCR
      jb     busy,   waiter           ; jump if erasure is not ended
      ;
      ; TEST VPP
      ;
      jb     fcvpp,  vpp_fail         ; jump if FCVPP is set, to know if
                                      ; a fail occured because VPP did not
                                      ; have the correct value during
                                      ; erasure
      ;
      ; ERASE VERIFY MODE
      ;
read_ff:cmp    all1,   [fl_scan]      ; first instruction for EVM
      calla  cc_UC,  wait4            ; time out 4µs
      cmp    all1,   [fl_scan]        ; second instruction for EVM
      jmpr   cc_NZ,  erase            ; jump if the word is not erased
      add    fl_scan,#02h             ; increment the bank pointer
      cmp    fl_scan,#FL_SIZE         ; compare to the last bank address
      jmpr   cc_NZ,  read_ff          ; jump to verify the next address
      ;
      ; EXIT OF PROGRAM MODE
      ;
      mov    FCR,    #ALL0            ; reset FCR and exit program mode
      ret                             ; return to main program
```

```
;32-BIT PROGRAMMING ROUTINE:  programming of address 0000Ch with 08642h
;_____________________________ and address 0000Eh with 09753h
bit32prg:
      ;
      ; REGISTER INITIALIZATION
      ;
      mov    lpcnt,  #ALL0            ; reset algo. loop counter
      mov    fcrval, #ALL0            ; reset FCR data value
      mov    unlock, #UNLOCK          ; load unlock data
      mov    val10u, #WAIT10          ; load 10µs loop data
      mov    val4u,  #WAIT4           ; load  4µs loop data
      mov    wait_cnt,#ALL0           ; reset wait loop counter
      mov    all1,   #ALL1            ; set R2 to FFFF
      mov    datal,  #DATAL           ; load data for even address
      mov    datah,  #DATAH           ; load data for odd address
      mov    addrev, #ADDREV          ; load even address
      mov    addrod, #ADDROD          ; load odd address
      ;
      ; UNLOCK SEQUENCE FOR ENTERING IN THE PROGRAM MODE
      ;
      mov    FCR,    unlock           ; first instruction
      mov    [unlock],unlock          ; second instruction of unlock
                                      ; sequence to enter in the program mode
      calla   cc_UC,  wait10          ; time out 10 µs to set internal signals
      ;
      ; FCR SET UP FOR PROGRAMMING
      ;
      bset   fwe                      ; FWE=1 define programming operation
      bclr   ckctl0                   ; CKCTL0=0 )
      bclr   ckctl1                   ; CKCTL1=0 ) define the pulse width
      bset   wdww                     ; WDWW=1 define 32-bit configuration
      bset   fwmset                   ; FWMSET=1 confirm program mode
      mov    FCR,    fcrval           ; load FCR set up
      ;
      ; TEST VPP
      ;
      mov    fcrrd,  FCR              ; read FCR
      jnb    vppriv, vpp_fail         ; test if VPP is high
```

```
      ; FLASH PROGRAMMING
      ;
prog:
      add    lpcnt,  #01h             ; increment the algo. loop counter
      cmp    lpcnt,  #MAXLOOP1        ; compare to the limit
      jmpr   cc_Z,   prg_fail         ; jump if limit has been reached
      mov    [addrev],datal           ; programming command, even word
      mov    [addrev],datah           ; programming command, odd word
waitpr:mov    fcrrd,  FCR              ; read FCR
      jb     busy,   waitpr           ; jump if programming is not ended
      ;
      ; TEST VPP
      ;
      jb     fcvpp,  vpp_fail         ; jump if FCVPP is set, to know if
                                      ; a fail occured because VPP did not
                                      ; have the correct value during
                                      ; programming
      ;
      ; PROGRAM VERIFY MODE
      ;
      cmp    datal,  [addrev]         ; first instruction for PVM (even)
      calla  cc_UC,  wait4            ; time out 4µs
      cmp    datal,  [addrev]         ; second instruction for PVM
      jmpr   cc_NZ,  prog             ; jump if the word is not correctly
                                      ; programmed, restart programming
      cmp    datah,  [addrod]         ; first instruction for PVM (odd)
      calla  cc_UC,  wait4            ; time out 4µs
      cmp    datah,  [addrod]         ; second instruction for PVM
      jmpr   cc_NZ,  prog             ; jump if the word is not correctly
                                      ; programmed, restart programming
      ;
      ; EXIT OF PROGRAM MODE
      ;
      mov    FCR,    #ALL0            ; reset FCR and exit program mode
      ret                             ; return to main program
```

```
;16-BIT PROGRAMMING ROUTINE:  programming of address 0000Ch with 08642h
;_____________________________

bit16prg:
      ;
      ; REGISTERS INITIALIZATION
      ;
      mov    lpcnt,  #ALL0            ; reset algo. loop counter
      mov    fcrval, #ALL0            ; reset FCR data value
      mov    unlock, #UNLOCK          ; load unlock data
      mov    val10u, #WAIT10          ; load 10µs loop data
      mov    val4u,  #WAIT4           ; load  4µs loop data
      mov    wait_cnt,#ALL0           ; reset wait loop counter
      mov    all1,   #ALL1            ; set R2 to FFFF
      mov    datal,  #DATAL           ; load data
      mov    addrev, #ADDREV          ; load address
      ;
      ; UNLOCK SEQUENCE FOR ENTERING IN THE PROGRAM MODE
      ;
      mov    FCR,   unlock            ; first instruction
      mov    [unlock],unlock          ; second instruction of unlock
                                      ; sequence to enter into the program mode
      calla  cc_UC,  wait10           ; time out 10 µs to set internal signals
      ;
      ; FCR SET UP FOR PROGRAMMING
      ;
      bset     fwe                    ; FWE=1 define programming operation
      bclr     ckctl0                 ; CKCTL0=0 )
      bclr     ckctl1                 ; CKCTL1=0 ) define the pulse width
      bclr     wdww                   ; WDWW=0 define 16-bit configuration
      bset     fwmset                 ; FWMSET=1 confirm program mode
      mov      FCR,   fcrval          ; load FCR set up
      ;
      ; TEST VPP
      ;
      mov    fcrrd,  FCR              ; read FCR
      jnb    vppriv, vpp_fail         ; test if VPP is high
```

```
      ; FLASH PROGRAMMING
      ;

progw:
      add    lpcnt,  #01h             ; increment the algo. loop counter
      cmp    lpcnt,  #MAXLOOP1        ; compare to the limit
      jmpr   cc_Z,   prg_fail         ; jump if limit has been reached
      mov    [addrev], datal          ; programming command
waitprw:mov    fcrrd,  FCR            ; read FCR
      jb     busy,   waitprw          ; jump if programming is not ended
      ;
      ; TEST VPP
      ;
      jb     fcvpp,  vpp_fail         ; jump if FCVPP is set, to know if
                                      ; a fail occured because VPP did not
                                      ; have the correct value during
                                      ; programming
      ;
      ; PROGRAM VERIFY MODE
      ;
      cmp    datal,  [addrev]         ; first instruction for PVM
      calla   cc_UC,  wait4           ; time out 4µs
      cmp    datal,  [addrev]         ; second instruction for PVM
      jmpr   cc_NZ,  progw            ; jump if the word is not correctly
                                      ; programmed, restart programming
      ;
      ; EXIT OF PROGRAM MODE
      ;
      mov    FCR,    #ALL0            ; reset FCR and exit program mode
      ret                             ; return to main program
```

SUBROUTINES USED IN WRITING OPERATION

```
wait4:add    wait_cnt,#01h          ; increment counter
      cmp    wait_cnt,val4u         ; compare with final value
      jmpr   cc_NZ,  wait4          ; jump if not equal
      mov    wait_cnt,#ALL0         ; reset counter
      ret

wait10:add    wait_cnt,#01h         ; increment counter
      cmp    wait_cnt,val10u        ; compare with final value
      jmpr   cc_NZ,  wait10         ; jump if not equal
      mov    wait_cnt,#ALL0         ; reset counter
      ret

vpp_fail:
      ; VPP FAIL ROUTINE DEFINED BY THE USER
prg_fail:
      ; PROGRAM FAIL ROUTINE DEFINED BY THE USER
eras_fail:
      ; ERASE FAIL ROUTINE DEFINED BY THE USER
```

THE SOFTWARE INCLUDED IN THIS NOTE IS FOR GUIDANCE ONLY. SGS-THOMSON SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM USE OF THE SOFTWARE.