Using Macro commands to erase and program Flash EEPROM

Larry Chen

In BDM software version 3.1 for parallel port and 4.1 for serial port, macro commands are incorporated to allow users to erase and program flash EEPROM through macro file. In this way, users have the full control how they are going to perform the operation. This application note explains the use of these commands.

The idea behind the operations is to write codes that perform erase or program operation in assembly language. The compiled code is loaded in to the device internal RAM. The code is then executed to perform the required function. For erase operation, data is generally not needed, so that the assembly code does not interact with the BDM. However, the interaction between assembly code and BDM software is necessary for programming operation. The BDM needs to prepare data to be programmed from S19 file, load the data into device memory. Then the assembly code works on the data and programs it into device flash memory. Then it informs BDM software that programming is finished, and the programming outcome, i.e., either data is programmed successfully, or an error is encountered in the process. BDM can access this error information to inform user. Therefore, BDM needs to perform the following function:

1. Load the assembly code into device internal memory;
2. Depending the function performed, BDM software may need to prepare data for the assembly code;
3. Execute the assembly code;
4. Probe the outcome when the process is finished.

These procedure may be repeated until the end of the data file.

According to these requirements, the following commands are provided:

LOAD	Load a file into device internal memory
ERASE	Execute an erase assembly code
PROG	Execute a program assembly code

The command formats are discussed in the following text.

LOAD

Command format
LOAD FILE_NAME

The full path may be necessary if the file is not in the directory where the BDM software starts. The file must be in S19 format, which may have S1 or S2 record. For example:
LOAD C:\PROGRAM\FEPROG.S19
Will load FEPROG.S19 into device memory. The file is loaded with the current setup, which include device, linear address or paged address, etc.

PROG

Command format
PROG FILE_NAME PROG_ADDR DATA_ADDR MASK

FILE_NAME: data file name in S19 format. For S2 record, BDM will send the long address in high word and low word. It is up to the program code to interpret the meaning of the address.

PROG_ADDR: The start address of the assembly code that performs the programming function (or other functions that require data).

DATA_ADDR: The address BDM uses to store information required by the assembly code. It must have the format below:

```
            ORG     $2100
Page        dc.w    0           ; page
Address     dc.w    0           ; start address in Flash
NumWords    dc.w    0           ; number of words
ErrorFlag   dc.w    0
DATA        dc.w    0           ; max of 64 words start here
```

Page:	memory where BDM stores the high word of the address. The page can be linear or paged. It will be zero if S1 record is used. The software accepts any kind of S-record. S-record can be mixed in one S19 file. It is up to the assembly program to interpret the contents of the address (paged address, linear address, or not-paged address).
Address:	memory where BDM stores the low word of the address.
NumWords:	memory where BDM stores the number of **bytes** of data.
ErrorFlag:	memory where assembly code stores the error code for BDM software to read.
DATA:	Start memory where data is stored. The total number of byte is indicated by NumWords.

MASK: a 16-bit mask value. This value is ANDed with the error code read back from device. If the result is not zero, a message will pop up to indicate that an error is encountered in programming. And programming function will stop here. If an error does not matter, a 0 mask can be used.

PROG_ADDR, DATA_ADDR, and MASK all must be in hex-decimal format.
The ORG directive in this case indicates that the DATA_ADDR should be 2100 hex.
The BDM will read the S19 file to be programmed line by line. It always tries to store even number of bytes into the device since the programming algorithm of some devices requires word format. Therefore, if a line contains even number of bytes, the whole data is sent to the device. If a line contains odd number of bytes, the last byte will not be sent. This byte will be combined with the next line if the addresses of the two lines are continuous. If the addresses are not continuous, only one byte is sent as the high byte of the word. The number of byte in this case is 1. The code may need to handle this value specially.

Example:
S110C03E0B30215DCC0030CD102B6B40CD38
S111C04B102C6940CC000CCD102D6B40180BCA
S110C06E00181813C32014B7463A6B40F6FB

The first line starts at address $C03E, and contains odd number of data. Therefore, 0B30 215D CC00 30CD 102B 6B40 are first programmed. CD will be programmed next time. In the second line, the address starts at C04B. This is continuous address from the first line. So CD is combined with this line. CD10 2C69 40CC 000C CD10 2D6B 4018 are then programmed. The remaining 0B will be combined with the third line. The start address of the third line is C06E, which is not a continuous address from the second line. Thus, only one byte 0B is programmed the third time. The fourth time will send data start at address $C06E.

Since BDM always sends even number of bytes each time except the last single byte in a continuous block, it is suggested that each data block start on even address.

ERASE

Command format
ERASE PROG_ADDR DATA_ADDR MASK

The PROG_ADDR, DATA_ADDR, and MASK all have the same means as those explained in the PROG command. The only difference is the format of the DATA_ADDR:

```
ErrorFlag   dc.w    0
```

Only ErrorFlag is present. No data is necessary for erasing.

Example

The following example uses very simple programs to illustrate the uses of the ERASE and PROG commands. The code is intended to run on 912B32. For other processor, the address must be changed accordingly.

The first program simulates the erase procedure. It contains a wait loop to delay for a short while. Then the code clears RAM locations from B00 to BFF to prepare for "programming". Thus the code is called MacWait.asm, which is listed in the following:

```
;  MacWait.ASM: Simple program to simulate flash erase code

            ORG     $0800           ; start of internal ram
MAIN:
            ldd     #$FFFF          ; load counter
AGAIN:
            subd    #1              ; decrement counter
            bne     AGAIN

            ldaa    #$80
            staa    ErrorFlag+1
            ldd     #0
            ldx     #$b00
AGAIN1
            std     2,x+
            dec     ErrorFlag+1
            bne     AGAIN1

            ldaa    #1
            staa    ErrorFlag+1
            bgnd

            ORG     $A00
ErrorFlag   dc.w    0

            END
```

Note that the code starts at memory $800. The data for error flag is $A00.

The next piece of code simulates the programming of flash EEPROM. It only stores the data in s19 file in RAM starting $B00.

```
;  MacTest.ASM: Simple program to simulate flash programming

            ORG     $0800           ; start of internal ram
MAIN:
            ldx     Address         ; Destination
            ldy     #DATA           ; source
AGAIN:
            ldaa    1,y+            ; get word & change point
            staa    1,x+            ; store word & change point
            dec     NumWords+1      ; decrement counter
            bne     AGAIN
            ldaa    #0
            staa    ErrorFlag+1
```

```
                bgnd

                ORG     $A00
Page            dc.w    0               ; page
Address         dc.w    0               ; start address in Flash
NumWords        dc.w    0               ; number of words
ErrorFlag       dc.w    0
DATA            dc.w    0               ; max of 64 words start here

                END
```

As in code MacWait.asm , the code starts address $800, and data starts at $A00. Code then moves the data sent by BDM to other RAM locations.

Note: For both erase and program code, a BGND command must be placed at the end of the program for BDM to sense if the program execution is finished.

The Macro file for MacWait and MacProg is listed below:

```
; File Name: LOAD.MAC: Test to load and program S19 file using MACRO file

RESET
LOAD C:\Userdata\HC12BGND\MacTest\MacWait.S19
erase 0800 0a00 ffff

LOAD C:\Userdata\HC12BGND\MacTest\MacProg.S19
prog c:\userdata\hc12bgnd\MacTest\EMPTY.s19 0800 0a00 ffff

LOAD C:\Userdata\HC12BGND\MacTest\MacProg.S19
prog c:\userdata\hc12bgnd\MacTest\Test1.s19 0800 0a00 ffff

LOAD C:\Userdata\HC12BGND\MacTest\MacProg.S19
prog c:\userdata\hc12bgnd\MacTest\Test2.s19 0800 0a00 ffff
```

MacWait is executed only once. However, MacProg is executed three times, with different data file each time. The first one just shows the response of an empty data file. The other two files contain different formats of S19 records. Note that all these commands can be placed in one Mcaro file to be executed once. The BDM will interpret these commands one by one and execute each command interpreted.

Since a value of 1 is loaded in the ErrorFlag in MacWait.asm, an error is always encountered in simulating erase procedure. A window will pop up to show the error code. In order to avoid this pop-up window, use a 0 to replace the mask ffff to mask of any error.

Appendix

Assembly code example

The following assembly code programs D256 flash EEPROM. Note that this code only program page FF. For other pages, page register must be considered.

```
; flash program registers
FCLKDIV         equ     $100            ; clock divider
FSEC            equ     $101
FCNFG           equ     $103
FPROT           equ     $104            ; protection register
FSTAT           equ     $105            ; status register
```

```
FCMD            equ     $106            ; command register
; bit definition
CBEIF           equ     $80             ; command buffer empty
CCIF            equ     $40             ; command complete flag
PVIOL           equ     $20             ; protection violation
ACCERR          equ     $10             ; Access error occurred
BLANK           equ     $04             ;

* Flash/EEprom programming commands...
ERASE           equ     $40             ; erase
PROGRAM         equ     $20             ; Program word
ERVER           equ     $04             ; Erase verify
MASS            equ     $01             ; Mass erase

VERIFY          equ     $05             ; Verify erased
S_ERASE         equ     $40             ; Sector erase
M_ERASE         equ     $41             ; Mass erase
S_MOD           equ     $60             ; sector modify, EEprom only

PPAGE           equ     $30

                org     $2000
; PC: start address of this code in internal RAM, = 2000
; X: points to destination in Flash
; PPAGE:
; NumWords: number of words to be programmed
; Address: start address in Flash to be programmed
; FCNFG: Block selection register
; FCLKDIV: clock divider, only need to init once for all programming

; SP: not used

; Y: points to data in internal RAM, always the same = #DATA
                movb    #$4a,FCLKDIV    ; Clock divider
                ldy     #DATA           ; points to start data
                ldx     Address         ; point address in flash
                ldab    NumWords+1      ; init count, +1 since a word
                lsrb                    ; shift 1 bits in word
                bne     EvenBytes
                ldab    #1              ; only one byte
EvenBytes       stab    NumWords+1      ; store back
                stab    count           ; also store in temp count
ProgLoop:
                ldd     2,y+            ; get word & change point
                std     2,x+            ; store word & change point
                ldab    #PROGRAM        ; program command
                stab    FCMD            ; write to command register
                ldab    #CBEIF          ; / start command
                stab    FSTAT           ; \ by write 1 to CBEIF
                ldab    FSTAT           ; check status
                bitb    #PVIOL+ACCERR           ;
                bne     ErrCmd          ; command error
                brclr   FSTAT,#CBEIF,*          ; loop if command buffer empty
                dec     count           ; any more word?
                bne     ProgLoop        ; do again
                brclr   FSTAT,#CCIF,*   ; loop till all CMDs finish
```

```
; start to verify
                ldab    NumWords+1      ; get number of words
                stab    count           ; save in count
                ldx     Address         ; load start address
                ldy     #DATA           ; load data pointer
VeriLoop        ldd     2,y+            ; load data
                cpd     2,x+            ; same as programmed?
                bne     ErrProg         ; error programming
                dec     count           ; decrease count
                bne     VeriLoop        ; again if not done
                ldd     #0              ; clear error flag
                std     ErrorFlag
                bgnd                    ; stop
ErrCmd          ldd     #1              ; program command error
                std     ErrorFlag
                bgnd
ErrProg         ldd     #2              ; not programmed correctly
                std     ErrorFlag
                bgnd

                org     $2100
count           dc.w    0               ; temp counter

Page            dc.w    0               ; page
Address         dc.w    0               ; start address in Flash
NumWords        dc.w    0               ; number of words
ErrorFlag       dc.w    0
DATA            dc.w    0               ; max of 64 words start here
```

From the code segment, PROG_ADDR is 2000, and DATA_ADDR is 2102.

FERASE.ASM: Erase DP256 internal flash eeprom

```
* EEprom status bits...
CBIEF           equ     $80             ; command buffer empty
CCIF            equ     $40             ; command complete flag
PVIOL           equ     $20             ; protection violation
ACCERR          equ     $10             ; Access error occurred
BLANK           equ     $04             ;

* Flash/EEprom programming commands...
EVRFY           equ     $05             ; Verify erased
PRGRM           equ     $20             ; Program word
S_ERASE         equ     $40             ; Sector erase
M_ERASE         equ     $41             ; Mass erase
S_MOD           equ     $60             ; sector modify, EEprom only

REGBS   equ     $0000
#include dp256reg.asm

        org     $1000

        movb    #$4A,FCLKDIV            ; set Flash clock divider
        lds     #$2000                  ; set stack
```

```
; Erase Internal Flash EEPROM
; Receive high order address to determine which block to erase.

Bulk:
;       jsr     (RecvHi-UTL_START)+RAMSTRT ; get the flash block address
EraseAll:
        jsr     CLRERR                  ; clear error flags if any
        clra                            ; bulk all, start with block 0
EraseLoop:
        psha                            ; save current block
        bsr     Erase                   ; do block erase
        pula                            ; get block number back
        bcs     EraseErr;
        tsta                            ; check if block 0 erased
        bne     EraseLp1                ; skif not block 0
        bsr     F_SEC                   ; if 0, reset security
EraseLp1:
        inca                            ; next block
        cmpa    #4                      ; test if done
        bne     EraseLoop               ; loop for all blocks
        ldab    #$A5                    ; load pass code
        bra     EraseEnd

EraseErr:
        ldab    #$80                    ; load error code
EraseEnd
;       jsr     (SendByte-UTL_START)+RAMSTRT ; Send code
        bgnd

; flash block mass erase routine..
Erase:
        staa    FCNFG                   ; set flash block, interrupts off
        coma                            ; change up for page compute
        anda    #$03                    ; mask block bits
        lsla                            ; shift for page ID
        lsla                            ;
        oraa    #$30                    ; pages = $30, $34, $38, $3c
        staa    PPAGE                   ; set page
        ldaa    #PVIOL+ACCERR           ; clear old fail flags,if any
        staa    FSTAT                   ;
Erase1:
        ldaa    FSTAT                   ; check if command buffer ready
        bpl     Erase1                  ; wait till buffer empty
        bita    #CCIF                   ; test command done
        beq     Erase1                  ; wait till last command complete
        std     $8000                   ; write to page address
* perform erase...
        movb    #M_ERASE,FCMD           ; mass erase block
        ldaa    #CBIEF                  ; start command
        staa    FSTAT                   ;
Eralp2:
        ldab    FSTAT                   ; check if error
        bitb    #PVIOL+ACCERR           ; test for error
        bne     EraseFail               ; if error, quit this one
        ldab    FSTAT                   ; check if command buffer ready
        bpl     Eralp2                  ; wait till buffer empty
```

```
        bitb    #CCIF               ; test command done
        beq     Eralp2              ; wait till last command complete

; here if part was erased successfully
ErasePass
        clc
        rts
; here if part cannot be erased
EraseFail
        sec
        rts

; clear error flags...
CLRERR:
        ldab    #PVIOL+ACCERR       ; clear old fail flags,if any
        clra                        ; prep for loop
CLRERRLP:
        staa    FCNFG               ; set block number
        stab    FSTAT               ; clear flags
        inca
        bita    #$04                ; test for done
        beq     CLRERRLP            ; loop if not
        rts

; DP256 version...
; REset security byte to valid value...
F_SEC:
        ldd     #$FFFE              ; program new security
        std     $FF0E               ;
        ldaa    #PRGRM              ; get command
        staa    FCMD
        ldaa    #CBIEF              ; start command
        staa    FSTAT               ;
F_SEC1:
        ldab    FSTAT               ; check if error
        bitb    #PVIOL+ACCERR       ; test for error
        bne     F_SECX                      ; if error, quit this one
        ldab    FSTAT               ; get status again
        bpl     F_SEC1              ; wait till buffer empty
        bitb    #CCIF               ; test command done
        beq     F_SEC1              ; wait till last command complete
        clra
F_SECX:
        rts                         ; done
```