Interfacing the NM29N16 in a Microcontroller Environment

National Semiconductor
Application Note 910
Cliff Zitlaw
Rob Frizzell
September 1993

INTRODUCTION

The NM29N16 is a 2Mbyte NAND Flash EEPROM memory that operates from a single 5V supply. This device does not have the parallel data, address, and control bus interfaces traditionally found on memory devices. The NM29N16 uses a byte wide serial interface with internal address, data, and control registers. The serial interface dramatically reduces the number of pins required to interface to the NM29N16. While the interface is nontraditional, it can easily be interfaced to standard microcontrollers. This application note describes how the NM29N16 can be interfaced to the Motorola 68HC11 microcontroller.

68HC11 INTERFACE

The NM29N16 can be interfaced to a microcontroller using the data bus, control bus, and a few I/O port bits. *Figure 1* shows the NM29N16 interfaced to a minimal 68HC11 system. The 68HC11 is configured in the expanded multiplexed mode which allows access to external memory devices. Most microcontrollers offer a mode that allows access to external memory and the NM29N16 should fit easily into all of these environments.

The I/Os of the NM29N16 were connected directly to the 68HC11 data bus. The NM29N16 occupies addresses C000H to DFFFH in the 68HC11 memory map due to the use of a three to eight (74HCT138) address decoder. While 8Kbytes of memory is taken in this design, the NM29N16 only requires a single address (C000H) out of that block. Due to timing constraints, the $\overline{RE}$ (Read Enable) and $\overline{WE}$ (Write Enable) signals must be ORed with the C000H address decode signal. $\overline{CE}$ (Chip Enable), CLE (Command Latch Enable), and ALE (Address Latch Enable) are controlled directly from three 68HC11 I/O port bits. The R/$\overline{B}$ (Ready/Busy) status output of the NM29N16 is polled by one I/O port bit.

A MAX707 μP supervisory chip is used to drive the $\overline{RESET}$ input of the 68HC11. The MAX707 forces its $\overline{RESET}$ output low until V_{CC} reaches 4.75V. Once V_{CC} exceeds 4.75V the $\overline{RESET}$ output remains low for an additional 200 ms before going high. This $\overline{RESET}$ output is also used to drive the $\overline{WP}$ (Write Protect) input of the NM29N16 to insure against inadvertent writes when V_{CC} is below 4.75V.

68HC11 TO NM29N16 COMMUNICATION

Information is transferred back and forth with a series of read and write operations that access the NM29N16 data, address and control registers. Loading the address register is accomplished by bringing $\overline{CE}$ low, ALE high and then loading data through the data bus with write operations to address C000H. Control register access is performed in a similar manner except that CLE is brought high instead of ALE. Data register access is performed when both ALE and CLE are low.

The EEPROM array in the NM29N16 is not directly accessible from the controller. An intermediate data register is used to transfer a page (264 bytes) of information back and forth between the EEPROM memory and the external controller. There are three basic forms of data transfers; erase operations that operate on a 16 page block, program operations that alter the contents of a single page, and read operations. During these three operations the R/$\overline{B}$ output goes low until the transfer or erase has completed.

A read operation is performed with a four step sequence. The command register is first loaded with the read instruction. The address register is then loaded with the page and byte address to access. At this point an internal recall operation is performed to transfer the contents of an EEPROM page to the 264 byte data register. After the recall has completed the accessed data is finally accessible by reading the contents of the data register. This is accomplished by pulsing $\overline{RE}$ low to read out sequential bytes.

Erase and program operations are performed similarly by accessing the data, control, and address registers. The simple access to these registers allow software routines that are as simple as that required to interface with a traditional parallel memory device.

SOFTWARE DRIVERS FOR A 68HC11 TO NM29N16 INTERFACE

A software listing is provided to demonstrate several features of the NM29N16. Different subroutines were developed that perform the basic read and write functions. These routines can be used with only minor modifications to interface the NM29N16 to any microcontroller.

 TL/D/11925 RRD-B30M105/Printed in U. S. A.

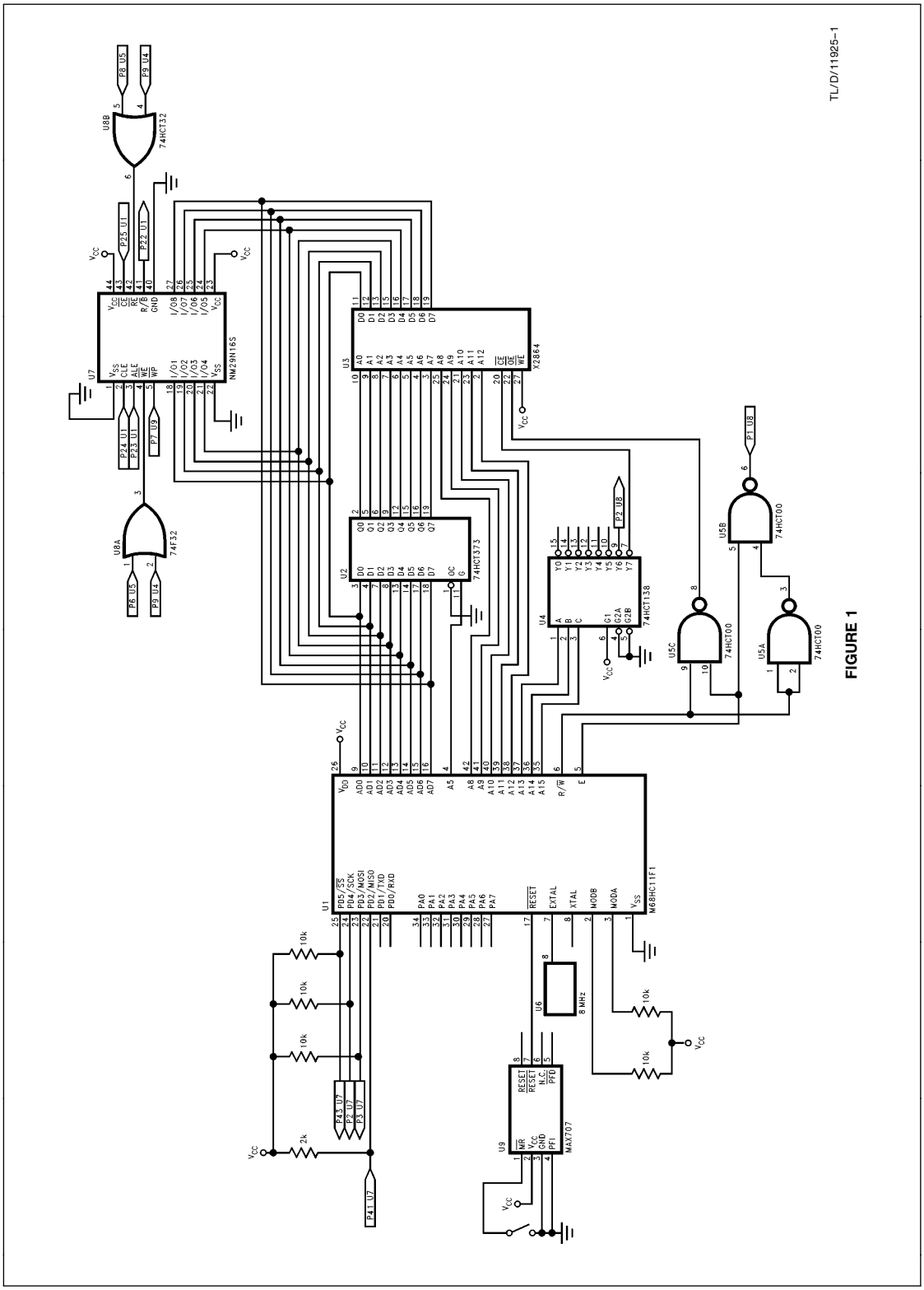

FIGURE 1

```
*************************************************************************
* This code was developed to demonstrate how the NM29N16 EEPROM can be *
* interfaced to the MC68HC11 microcontroller.  The software includes   *
* several subroutines that perform various interface functions.  The   *
* subroutines include:                                                 *
*                                                                      *
*    RDPAG  : Read a page of information (264 bytes) out of the NM29N16*
*    RDDAT1 : Read a byte from data memory                             *
*    RDRED1 : Read a byte from redundancy memory                       *
*    PGMRED : Program a byte in redundancy memory                      *
*    PGMDAT : Program a byte in data memory                            *
*    PGMPAG : Program an entire page (256 bytes data, 8 redundancy)    *
*    ERASE1 : Erase a block (16 pages)                                 *
*    STATUS : Read the Status Register                                 *
*    READID : Read the manufacturer code and the device code           *
*    INIT   : Tag blocks that are not fully functional                 *
*                                                                      *
* The 68HC11 interfaces to the NM29N16 by using the data bus, control  *
* bus, and a few I/O port lines.  The NM29N16 requires only one address*
* location when configured in this manner.  The data bus is directly   *
* connected to the EEPROM.  Three I/O lines drive CLE, ALE, and CE.    *
* One I/O line is used to monitor the R/B output.                      *
*                                                                      *
* The mainline was used to test the functionality of the subroutines.  *
* The subroutines can be copied directly into a customer's program and *
* be expected to operate as described.  The final mainline only        *
* performs a block erase and verify.                                   *
*************************************************************************

****************************
* ADDRESS LOCATION EQUATES *
****************************

DDRD    EQU     $09                     port D direction register = $1009
PORTD   EQU     $08                     port D data register      = $1008
FLASH   EQU     $C000                   NM29N16 = $C000 to $DFFF

************************
* BIT POSITION EQUATES *
************************

CEBIT   EQU     $20                     CE position in port D  = bit 5
CLEBIT  EQU     $10                     CLE position in port D = bit 4
ALEBIT  EQU     $08                     ALE position in port D = bit 3

****************************
* VARIABLE ADDRESS EQUATES *
****************************

HIPG    EQU     $0180                   high order page pointer
LOPG    EQU     $0181                   low order page pointer
ADD     EQU     $0182                   byte pointer within a page
DATVAL  EQU     $0183                   data transfer register
BLOCK   EQU     $0184
BLOCKL  EQU     $0185

****************
* RESET VECTOR *
****************
```

TL/D/11925–2

```
        ORG     $FFFE                   reset vector to $E000
        FDB     $E000

****************************
* PROGRAM STARTING LOCATION *
****************************

        ORG     $E000                   program execution begins at $E000
BEGIN:  LDS     #$01FF                  initialize stack pointer
        LDX     #$1000                  initialize "address index register"
        LDAA    #$FF                    initialize I/O ports
        STAA    PORTD,X
        LDAA    #$3B
        STAA    DDRD,X
        BCLR    PORTD,X #CLEBIT
        BCLR    PORTD,X #ALEBIT         CE=1 CLE=0 ALE=0 initially

************
* MAINLINE *
************

        LDAA    #$00
        STAA    LOPG
        STAA    HIPG
        JSR     ERASE1
        JSR     STATUS

LOOP:   BRA     LOOP                    wait until reset loop

*************************************************************************
* RDPAGE copies a page from the NM29N16 into SRAM memory on the 68HC11.*
* All 264 bytes (data and redundancy) are copied into the SRAM buffer. *
* The page number to be transfered is passed in the variables LOPG and *
* HIPG.  The EEPROM data is copied into the 68HC11 SRAM between        *
* addresses 0000H and 0107H.                                           *
*************************************************************************

RDPAGE: BSET    PORTD,X #CLEBIT
        BCLR    PORTD,X #CEBIT
        LDAA    #$00
        STAA    FLASH                   load READ(1) instruction into
        BSET    PORTD,X #CEBIT           the command register
        BCLR    PORTD,X #CLEBIT
        BSET    PORTD,X #ALEBIT
        BCLR    PORTD,X #CEBIT
        LDAA    #$00                    load the byte pointer in the address
        STAA    FLASH                    register with 00H (start of page)
        LDAA    LOPG
        STAA    FLASH                   load low order page number
        LDAA    HIPG
        STAA    FLASH                   load high order page number
        BCLR    PORTD,X #ALEBIT
        JSR     WAIT                    wait for recall into data register
        LDY     #$0000
NEXTR:  LDAA    FLASH                   read data from EEPROM and fill SRAM
        STAA    $00,Y                    buffer with data register contents
        INY
        CPY     #$0108                  loop until all 264 bytes have
        BNE     NEXTR                    been transfered
        BSET    PORTD,X #CEBIT
```

TL/D/11925–3

```
        JSR     WAIT                    pause to assure EEPROM is idle
        RTS

*************************************************************************
* RDDAT1 and RDRED1 are used to read the contents of a single address *
* in the EEPROM array.  Data can be read from either the DATA portion *
* of the array (RDDAT1) or the REDUNDANT portion (RDRED1).  The       *
* location to be accessed is defined in the variables ADD, LOPG, and  *
* HIPAG.  LOPG and HIPG define the page to be accessed and ADD        *
* indicates a position within the page.  ADD can range between 0 and  *
* 255 for DATA accesses or between 0 and 7 for REDUNDANT accesses.    *
* The value in the chosen location is returned in the variable DATVAL.*
*************************************************************************

RDDAT1: LDAA    #$00                    READ(1) command
        BRA     RDJMP
RDRED1: LDAA    #$50                    READ(2) command
RDJMP:  BSET    PORTD,X #CLEBIT
        BCLR    PORTD,X #CEBIT
        STAA    FLASH                   load appropriate READ command into
        BSET    PORTD,X #CEBIT           the command retister
        BCLR    PORTD,X #CLEBIT
        BSET    PORTD,X #ALEBIT
        BCLR    PORTD,X #CEBIT
        LDAA    ADD                     load the byte address into the
        STAA    FLASH                    address register
        LDAA    LOPG
        STAA    FLASH                   load the low order page number
        LDAA    HIPG
        STAA    FLASH                   load the high order page number
        BCLR    PORTD,X #ALEBIT
        JSR     WAIT                    wait for recall to data register
        LDAA    FLASH                   load the value from the chosen

        LDAA    FLASH

        STAA    DATVAL                   address and save the result in DATVAL
        BSET    PORTD,X #CEBIT
        JSR     WAIT                    pause until EEPROM is idle
        RTS

*************************************************************************
* PGMRED, PGMDAT, and PGMPAG are used to program either a single byte   *
* or an entire page.  During program operations the entire data register*
* must be loaded and then the contents transfered to an EEPROM page.    *
* EEPROM bits can only be flipped from a one (erased state) to a zero   *
* (programmed state) during a program operation.  To program a single   *
* byte the entire data register must be filled with FFH except for the  *
* byte that is to be programmed.  During the programming cycle bits     *
* that are zero in the data register will force the corresponding bits  *
* in the chosen page to the zero state, other bits will remain          *
* unchanged.                                                            *
*                                                                       *
* These routines use a SRAM data array located on the 68HC11 between    *
* address 0000H and 0107H.  This array is transfered byte for byte into *
* the NM29N16 data register during the data load portion of the         *
* programming cycle.  If single byte is to be altered the location      *
* in the SRAM array corresponding to the address to be programmed is    *
* loaded with the new data and all other addresses in the array are     *
* filled with FFH.                                                      *
```

TL/D/11925–4

```
*                                                                          *
* The routines PGMRED and PGMDAT are used to program a single byte in      *
* redundant or data memory respectively.  The data value to be updated     *
* is contained in the variable DATVAL, the page number is contained in     *
* PGLO and PGHI, and the byte position within the page is contained in     *
* ADD.                                                                     *
*                                                                          *
* The routine PGMPAG assumes that the SRAM array already has the data      *
* that will be programmed into the EEPROM.  PGLO and PGHI contain the      *
* page number to be programmed.                                            *
****************************************************************************

PGMRED: JSR     FILLFF                  fill SRAM array with FFH
        LDY     #$0100                  load Y with redundant memory offset
        BRA     PGMB
PGMDAT: JSR     FILLFF                  fill SRAM array with FFH
        LDY     #$0000                  load Y with data memory offset
PGMB:   LDAB    ADD                     data or redundant address to alter
        ABY                             calculate absolute address in page
        LDAA    DATVAL
        STAA    $00,Y                   write new data byte into SRAM array
PGMPAG: BSET    PORTD,X #CLEBIT
        BCLR    PORTD,X #CEBIT
        LDAA    #$80                    load command register with
        STAA    FLASH                    data input command
        BCLR    PORTD,X #CLEBIT
        BSET    PORTD,X #ALEBIT
        LDAA    #$00                    load address register with
        STAA    FLASH                    start of page
        LDAA    LOPG
        STAA    FLASH                   load low order page number
        LDAA    HIPG
        STAA    FLASH                   load high order page number
        BCLR    PORTD,X #ALEBIT
        LDY     #$0000
LOADB:  LDAA    $00,Y                   transfer data from SRAM array
        STAA    FLASH                    into the NM29N16 data register
        INY
        CPY     #$0108                  loop until all 264 bytes have
        BNE     LOADB                    been transfered
        BSET    PORTD,X #CLEBIT
        LDAA    #$10                    load command register with
        STAA    FLASH                    start program command
        BSET    PORTD,X #CEBIT
        BCLR    PORTD,X #CLEBIT
        JSR     WAIT                    pause to make sure EEPROM idle
        RTS

FILLFF: LDY     #$0000                  fill SRAM addresses 0000H to
        LDAA    #$FF                     0107H with FFH
LOOPF:  STAA    $00,Y
        INY
        CPY     #$0108
        BNE     LOOPF
        RTS

****************************************************************************
* ERASE1 performs an erase operation on a single block (16 pages).  The    *
* block to be erased is specified in the variables LOPG and HIPG.  The     *
* lower 4 bits of LOPG are not used so that the least significant bit of*
```

TL/D/11925–5

```
* the block number is the 5th bit of LOPG.  The block number is         *
* specified in LOPG (bits 4-7) and HIPG (bits 0-4).                     *
*************************************************************************

ERASE1: BSET    PORTD,X #CLEBIT
        BCLR    PORTD,X #CEBIT
        LDAA    #$60                    load command register with
        STAA    FLASH                    block erase command
        BCLR    PORTD,X #CLEBIT
        BSET    PORTD,X #ALEBIT
        LDAA    LOPG                    load low order block number
        STAA    FLASH                    (XXXX0123)
        LDAA    HIPG                    load high order block number
        STAA    FLASH                    (45678XXX)
        BCLR    PORTD,X #ALEBIT
        BSET    PORTD,X #CLEBIT
        LDAA    #$D0                    load command register with erase
        STAA    FLASH                    execution command
        BCLR    PORTD,X #CLEBIT
        BSET    PORTD,X #CEBIT
        JSR     WAIT                    pause until EEPROM is idle
        RTS

*************************************************************************
* STATUS is used to read the NM29N16 status register.  This command can *
* be used after erase and program cycles to determine if the results    *
* were successfull.  The contents of the status register are returned in*
* the variable DATVAL.                                                  *
*************************************************************************

STATUS: BSET    PORTD,X #CLEBIT
        BCLR    PORTD,X #CEBIT
        LDAA    #$70                    load command register with
        STAA    FLASH                    status read command
        BSET    PORTD,X #CEBIT
        BCLR    PORTD,X #CLEBIT
        BCLR    PORTD,X #CEBIT
        LDAA    FLASH                   read status register
        STAA    DATVAL                  save results
        BSET    PORTD,X #CEBIT
        RTS

*********************************************************************
* READID is used to read the NM29N16 device and manufacturer codes. *
* The manufacturer code is returned in the A register and the device*
* code is returned in the B register.                               *
*********************************************************************

READID: BSET    PORTD,X #CLEBIT
        BCLR    PORTD,X #CEBIT
        LDAA    #$90                    load the command register with
        STAA    FLASH                    the ID read command
        BSET    PORTD,X #CEBIT
        BCLR    PORTD,X #CLEBIT
        BSET    PORTD,X #ALEBIT
        BCLR    PORTD,X #CEBIT
        LDAA    #$00                    load the address register with
        STAA    FLASH                    address 0
        BCLR    PORTD,X #ALEBIT
        LDAA    FLASH                   read the manufacturer code
```

TL/D/11925–6

```
        LDAB    FLASH                   read the device code
        BSET    PORTD,X #CEBIT
        RTS

*********************************************************************
* WAIT is used to pause until the NM29N16 returns to the ready mode. *
* The routine polls the R/B (ready/busy) pin until it returns high.  *
*********************************************************************

WAIT:   LDAA    PORTD,X                 check bit 2 of port D (R/B line)
        ANDA    #$04
        CMPA    #$00
        BEQ     WAIT                    loop until R/B returns high
        RTS

*************************************************************************
* INIT is used to determine which blocks in the NM29N16 are usable.  The *
* sequence is as follows:                                                *
*                                                                        *
*  1      Start with block 0                                             *
*  2      Erase and verify block                                         *
*  3      Write $AA to each page in the block and verify                 *
*  4      Erase and verify block                                         *
*  5      Write $55 to each page in the block and verify                 *
*  6      Erase and verify block                                         *
*  7      If steps 1-5 were successful tag the good block with data $F0  *
*          in address 0 of the redundancy memory in page 0 of the block  *
*          just verified                                                 *
*  8      Step through all 512 blocks                                    *
*************************************************************************

INIT:   LDY     #$0000                  start with block 0
        STY     BLOCK
LOOP1:  LDY     BLOCK
        STY     HIPG
        JSR     ERASE1                  erase block
        JSR     STATUS                  see if erase was successful
        LDAA    DATVAL
        ANDA    #$01
        BEQ     NXTSTP                  jump to BADBLK if erase unsuccessful
        JMP     BADBLK
NXTSTP: LDAA    #$AA                    verify that $AA can be written
        JSR     FILLXX                   to all pages in the block
        LDY     BLOCK
        LDAB    #$0F                    start with page 15 and work down
        CLC                              to page 0
        ABY
        STY     HIPG
        LDAA    #$00
        STAA    ADD
LOOPAA: JSR     PGMPAG                  program page with $AA
        JSR     STATUS                  see if programming is successful
        LDAA    DATVAL
        ANDA    #$01
        BNE     BADBLK                  jump to BADBLK if bad page found
        LDAA    LOPG
        ANDA    #$0F                    loop until all pages have been
        BEQ     DONEAA                   tested
        DEC     LOPG                    step to next page in the block
        BRA     LOOPAA                   being verified
```

TL/D/11925–7

```
DONEAA: JSR     ERASE1                  erase block
        JSR     STATUS                  see if erase was successful
        LDAA    DATVAL
        ANDA    #01
        BNE     BADBLK                  jump to BADBLK if erase unsuccessful
        LDAA    #$55                    verify that $55 can be written to
        JSR     FILLXX                   all pages in the block
        LDY     BLOCK
        LDAB    #$0F                    start with page 15
        CLC
        ABY
        STY     HIPG
        LDAA    #$00
        STAA    ADD
LOOP55: JSR     PGMPAG                  program page with $55
        JSR     STATUS                  see if programming is successful
        LDAA    DATVAL
        ANDA    #$01
        BNE     BADBLK                  jump to BADBLK if bad page found
        LDAA    LOPG
        ANDA    #$0F                    loop until all pages in block
        BEQ     DONE55                   have been verified
        DEC     LOPG                    step to next page
        BRA     LOOP55
DONE55: JSR     ERASE1                  erase block
        JSR     STATUS                  see if erase is successful
        LDAA    DATVAL                  jump to DATVAL if bad block found
        ANDA    #01
        BNE     BADBLK                  jump to BADBLK if erase unsuccessful
        LDAA    #$F0                    tag good block by writing $F0 into
        STAA    DATVAL                   byte 0, page 0 (redundancy memory)
        LDY     BLOCK                    of the block just verified
        STY     HIPG
        LDAA    #$00
        STAA    ADD
        JSR     PGMRED
BADBLK: LDY     BLOCK                   exit routine if all 512 blocks
        CPY     #$1FF0                   have been tested
        BEQ     DONE
        CLC                             step to next block (16 pages)
        LDAB    #$10
        ABY
        STY     BLOCK
        JMP     LOOP1                   go verify next block
DONE:   RTS

FILLXX: LDY     #$0000                  fill SRAM addresses 0000H to
LOOPF2: STAA    $00,Y                    0107H with value in A reg
        INY
        CPY     #$0108
        BNE     LOOPF2
        RTS
```

TL/D/11925–8

SUMMARY

The NM29N16 provides an extremely flexible interface for many systems. By not utilizing address lines, the device gives designers the ability to incorporate multiple megabytes of memory without the use of an expensive processor or system bus. The application described here is only one example of this. With this architecture, the NM29N16 should enable new types of portable systems to be developed.

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: cnjwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408