Application Note

AN2545/D
Rev. 1, 6/2004

Using MC68HC908GR/GZ On-Chip FLASH Programming Routines

By Kazue Kikuchi
8/16 Bit MCU Applications Engineering
Austin, Texas

Introduction

The MC68HC908GZ16 (GZ16), MC68HC908GR16 (GR16), and MC68HC908GZ8 (GZ8) microcontroller units (MCUs) have FLASH memory (16 Kbytes for GZ16 and GR16; 8 Kbytes for GZ8). To program, erase, and verify FLASH, the MCUs have on-chip FLASH support routines residing in ROM (read-only memory). These routines may be accessed in either user mode or monitor mode and eliminate the need to develop separate FLASH routines for applications.

This application note describes how to call each of the routines in the user software and what is performed and returned as confirmation of routine execution.

NOTE: *With the exception of mask set errata documents, if any other Motorola document contains information that conflicts with the information in the device data sheet, the data sheet should be considered to have the most current and correct data.*

FLASH Overview

The FLASH cell used on the GZ8/16 and GR16 is an industry-proven split-gate cell available from Silicon Storage Technology® (SST) in 0.5-micron geometry. The cell uses channel hot electron injection for programming and Fowler-Nordheim tunnelling for erasing. All programming voltages are generated internally by a charge pump from a single connection to V_{DD}. More information on the FLASH cell is available at the SST website: http://www.ssti.com.

With the quick byte programming time and the organization of the FLASH array into 32-byte rows, the entire 16-Kbyte memory can be programmed in less than one second. This type of FLASH is specified to withstand at least 10,000 program/erase cycles and has enhanced reliability over previous technology.

page basis. Also, an entire specified array can be mass erased. For the GZ8/16 and GR16, rows are 32 bytes and pages are 64 bytes (two rows of 32 bytes each).

Routines Supported in ROM

In the GZ8/16 and GR16 ROM, five routines are supported. This section introduces each routine briefly. Details are discussed in later sections.

GetByte This routine is used to receive a byte serially on the general-purpose I/O port A, bit 0 (PTA0). The receiving baud rate is the same as the baud rate used in monitor mode.

RDVRRNG This routine is used to read FLASH locations and to verify the FLASH data against data in specific RAM locations, which are referred to as DATA arrays.

PRGRNGE This routine is used to program a contiguous range of FLASH locations. Programming data is first loaded into the DATA array. PRGRNGE can be used when the internal operating frequency (f_{op}) is between 2.0 MHz and 8.4 MHz.

ERARNGE This routine is used to erase either a page (64 bytes) or the whole array of FLASH. It can be used when the internal operating frequency (f_{op}) is between 2.0 MHz and 8.4 MHz.

DELNUS This routine can generate a specified delay based on the values of register X and accumulator (A) as parameters. DELNUS is used in PRGRNGE and ERARNGE routines.

NOTE: *Because the ROM has a jump table, the user does not call the routines with direct addresses. Therefore, the calling addresses will not change even when the ROM code is updated in the future.*

Variables Used in the Routines

The RDVRRNG, PRGRNGE, and ERARNGE routines require certain registers and/or RAM locations to be initialized before calling the routines in user software. Table 1 shows variables used in the routines and their locations.

Table 1. Variables and Their Locations

Location	Variable Name	Size (Bytes)	Description
$40 – $47	Reserved	8	Reserved for future use
$48	CTRLBYT	1	Control byte including MASS erase bit (bit 6)
$49	CPUSPD	1	CPU speed — f_{op} (in MHz) × 2 then rounded up to the next integer; for example, if f_{op} = 2.4576 MHz, CPUSPD = 5
$4A, $4B	LADDR	2	Last address of a 16-bit range
$4C	DATA	Varies	First location of DATA array; DATA array size must match a programming or verifying range
Registers H:X	—	2	Beginning address of a 16-bit range

CTRLBYT

The control byte (CTRLBYT) is located at RAM address $48 and is used for the ERARNGE routine. Bit 6 in this location is used to specify either MASS (1) or PAGE (0) erase.

CPUSPD

To set up proper delays used in the PRGRNGE and ERARNGE routines, a value indicating the internal operating frequency (f_{op}) must be stored at CPUSPD, which is located at RAM address $49. The value is f_{op} (in MHz) times 2 then rounded up to the next integer. For example, if f_{op} is 4.2 MHz, the CPUSPD value is 9. If f_{op} is 2.5 MHz, the CPUSPD value is 5. Setting a correct CPUSPD value is very important to program or erase the FLASH successfully.

LADDR

A range specifies the FLASH locations to be read, verified, or programmed. The 16-bit value in RAM addresses $4A and $4B holds the last address of a range. The addresses $4A and $4B are the high and low bytes of the last address, respectively. LADDR is used for RDVRRNG and PRGRNGE routines.

$4C. The array is used for loading program or verify data. The DATA array must be in the zero page and its size must match the size of the range to be programmed or verified.

Registers H:X

In the RDVRRNG and PRGRNGE routines, registers H and X are initialized with a 16-bit value representing the first address of a range. High and low bytes of the address are stored to registers H and X, respectively. In the ERARNGE routine, registers H and X are initialized with an address which is within the page or entire array to be erased.

How to Use the Routines

This section describes the details of each routine. **Table 5** summarizes the five routines.

GetByte

GetByte is a routine that receives a byte on the general-purpose I/O PTA0, and the received value is passed back to the calling routine in the accumulator (A). This routine expects the same non-return-to-zero (NRZ) communication protocol and baud rate that is used in monitor mode. A similar routine that is used by the monitor echoes each received byte before attempting to receive a new byte. It is more efficient to use this GetByte routine when user software or data is downloaded to RAM because it eliminates the time overhead in echoing back every byte that is received. If user software already has a built-in error detection scheme such as checksum, data echoing back is not necessary.

This routine detects a framing error when a STOP bit is not detected. If the carry (C) bit of the condition control register (CCR) is cleared after returning from this routine, a framing error occurred during the data receiving process. Therefore, the data in A is not reliable. User software is responsible for handling such errors.

To use this routine, some hardware setup is required. The general-purpose I/O PTA0 must be pulled up. For more information, refer to the monitor ROM section in the device data sheet.

The GZ8/16 and GR16 support different baud rates. The GZ8/16 baud rate is defined by f_{op} divided by 278; the GR16 baud rate is defined by f_{op} divided by 256. **Table 2** and **Table 3** show typical PC baud rates used for these MCUs.

Table 2. Typical Baud Rates for GZ8 and GZ16

Operation Bus Freq. (f_{op})	Calculated Baud Rate	Closest PC Baud Rate
2.0 MHz	7,194 bps	7,200 bps
4.0 MHz	14,388 bps	14,400 bps
8.0 MHz	28,777 bps	28,800 bps

Table 3. Typical Baud Rates for GR16

Operation Bus Freq. (f_{op})	Calculated Baud Rate	Closest PC Baud Rate
2.4576 MHz	9,600 bps	9,600 bps
4.9152 MHz	19,200 bps	19,200 bps

NOTE: *Interrupts are not masked (the I bit is not set) and the COP is not serviced in the GetByte routine. User software should ensure that interrupts are blocked during character reception.*

Entry Condition

In hardware, PTA0 must be pulled up.

In the user software, PTA0 must be configured as an input.

Exit Condition

A — Contains data received from PTA0.

C bit — Normally the C bit is set, indicating proper reception of the STOP bit. However, if the C bit is clear, a framing error occurred. Therefore, the received byte in A is not reliable.

Example 1: Receiving a Byte Serially

Example 1 shows how to receive a byte serially on PTA0:

```
GETBYTE equ  $1C00

        bclr 0,DDRA0        ;Configure Port A bit 0 as an input

        jsr  GETBYTE        ;Call GETBYTE routine
        bcc  FrameError     ;If C bit is clear, framing error
                            ; occurred. Take a proper action
```

NOTE: *As soon as GetByte is called, the program will remain in this routine until a START bit (0) is detected and a complete character is received.*

function options:

- Send-out option — Used to read a range of FLASH locations and send the read data to a host through communication PTA0.
- Verify option — Used to read a range of FLASH locations and to verify the read data against the DATA array.

Send-Out Option

If the accumulator (A) is initialized with $00 at the routine entry, the read data will be sent out serially through communication PTA0 (send-out option). The communication baud rate is the same as the baud rate described in the **GetByte** routine. When this option is selected, the PTA0 must be pulled up and configured as an input and the PTA0 data bit must be initialized to 0.

Verify Option

If A is initialized with a non-zero value, the read data is verified against the DATA array (verify option) for each byte of FLASH data that does not match the corresponding value in the DATA array. The value in the DATA array is replaced by the data read from FLASH. All data in the DATA array must be in the zero page and its size must match the size of a specified verify range.

Carry (C) Bit

The beginning and last addresses of the range to be read and/or verified are specified as parameters in registers H:X and LADDR, respectively. In the verify option, the carry (C) bit of the condition code register (CCR) is set if the data in the specified range is verified successfully against the data in the DATA array. However when the send-out option is selected, the status of the C bit is meaningless because this function does not include the verify operation. Both options calculate a checksum on data read in the range. This checksum, which is the LSB of the sum of all bytes in the entire data collection, is stored in the A upon return from the function.

Interrupts are masked (the I bit is set) when the send-out option is selected. The COP is serviced in RDVRRNG. However, the COP timeout might still occur in the send-out option if the COP is configured for a short timeout period.

Entry Condition

H:X — Contains the beginning address in a range.

LADDR — Contains the last address in a range.

A — When A contains $00, read data is sent out via PTA0 (send-out option is selected). When A contains a non-zero value, read data is verified against the DATA array (verify option is selected).

DATA array — contains data to be verified against FLASH data. For the send-out option, the DATA array is not used.

PTA0 — When the send-out option is selected, this pin must be configured as an input and pulled up in hardware and PTA0 must be initialized to 0.

Exit Condition

A — Contains a checksum value.

H:X — Contains the address of the next byte just after the range read.

C bit — Indicates the verify result (only applies to the verify option).

When the C bit is set, the verify succeeded.

When the C bit is cleared, the verify failed.

DATA array — Replaced with data read from FLASH when the verify option is selected.

Example 2: Verify Option

Example 2 shows how to use the verify option:

```
RDVRRNG equ  $1C03

        ldhx #$0000         ;Index offset into DATA array
        lda  #$AA           ;Initial data value to store in array
Data_load:
        coma
        sta  DATA,x         ;Fill DATA array, 32 bytes data,
                            ; to verify against programmed FLASH
        aix  #$1            ; data (In this example verifying data
        cphx #$20           ; is $55, $AA, $55, $AA....)
        bne  Data_load

        ldhx #$C01F         ;Load last address of range to
        sthx LADDR          ; LADDR
        ldhx #$C000         ;Load beginning address of range
                            ; to H:X
        lda  #$55           ;Write non-zero value to A to select
                            ; the verify option
        jsr  RDVRRNG        ;Call RDVRRNG routine
        bcc  Error          ;If bit C is cleared, verify failed
                            ; Take a proper action
                            ;A contains a checksum value
```

Example 3: Send-Out Option

Example 3 shows how to use the send-out option:

```
RDVRRNG equ  $1C03

        bclr 0,DDRA         ;Configure Port A bit 0 as an input
        bclr 0,PTA          ;Initialize data bit to zero PTA0=0
        ldhx #$C025         ;Load last address of range to
        sthx LADDR          ; LADDR
        ldhx #$C010         ;Load beginning address of range
                            ; to H:X
        clra                ;A=0 to select send-out option
        jsr  RDVRRNG        ;Call RDVRRNG routine
                            ;A contains a checksum value
```

into the DATA array. All data in the DATA array must be in the zero page, but the range size is not limited to the 32-byte row size. Programming data is passed to PRGRNGE in the DATA array. The size of the DATA array must match the size of a specified programming range. This routine supports an internal operating frequency between 2.0 MHz and 8.4 MHz.

For this split-gate FLASH, the programming algorithm requires a programming time (t_{prog}) between 30 µs and 40 µs. (Refer to the FLASH memory section in the device data sheet.) **Table 4** shows how t_{prog} is adjusted by a CPUSPD value in this routine. The CPUSPD value is f_{op} (in MHz) multiplied by 2 then rounded up to the next integer. For example, if f_{op} is 2.4576 MHz, the CPUSPD value is 5. If f_{op} is 8.0 MHz, the CPUSPD value is 16 ($10).

Table 4. t_{prog} vs. Bus Frequency

	Operating Bus Freq. (f_{op})	CPUSPD	t_{prog} (Cycles)	t_{prog}
Case 1	2.0 MHz $\leq f_{Bus} \leq$ 2.5 MHz	4, 5	75	30.00 µs $\leq t_{prog} \leq$ 37.50 µs
Case 2	2.5 MHz $< f_{Bus} \leq$ 3.0 MHz	6	90	30.00 µs $\leq t_{prog} <$ 36.00 µs
Case 3	3.0 MHz $< f_{Bus} \leq$ 4.0 MHz	7, 8	CPUSPD $\times$ 3 + 99	30.75 µs $\leq t_{prog} <$ 40.00 µs
Case 4	4.0 MHz $< f_{Bus} \leq$ 5.5 MHz	9, 10, 11	CPUSPD $\times$ 6 + 104	30.90 µs $\leq t_{prog} <$ 39.50 µs
Case 5	5.5 MHz $< f_{Bus} \leq$ 8.4 MHz	12, 13, 14, 15, 16, 17	CPUSPD $\times$ 9 + 101	30.62 µs $\leq t_{prog} <$ 38.18 µs

In PRGRNGE, the high programming voltage time is enabled for less than 125 µs when programming a single byte at any operation bus frequency between 2.0 MHz and 8.4 MHz. Therefore even when a row is programmed by 32 separate single-byte programming operations, the cumulative high voltage programming time is less than the maximum t_{HV} (4 ms). The t_{HV} is defined as the cumulative high voltage programming time to the same row before next erase. For more information, refer to memory characteristics in the electrical specifications section of the device data sheet.

This routine does not confirm that all bytes in the specified range are erased prior to programming. Nor does this routine do a verification after programming, so there is no return confirmation that programming was successful. To program data successfully, the user software is responsible for these checking operations. The RDVRRNG routine can be used to verify a programmed FLASH range against the DATA array.

Interrupts are masked (the I bit is set) and the COP is serviced in this routine.

Entry Condition

H:X — Contains the beginning address in a range.

LADDR — Contains the last address in a range.

CPUSPD — Contains an integer value equal to f_{op} (in MHz) times 2 and rounded up to the next integer.

DATA array — Contains the data values to be programmed into FLASH.

Exit Condition

H:X — Contains the address of the next byte after the range just programmed.

Example 4: Programming a Row

Example 4 shows how to program one full 32-byte row:

```
PRGRNGE equ   $1C09

        ldhx #$0000         ;Index offset into DATA array
        lda  #$AA           ;Initial data value (inverted)
Data_load:
        coma                ;Alternate between $55 and $AA
        sta  DATA,x         ;Fill DATA array, 32 bytes data,
                            ; values to program into FLASH
        aix  #$1            ; (ie. 55, AA, 55, AA....)
        cphx #$20
        bne  Data_load

        mov  #$5,CPUSPD     ;fop = 2.4576MHz in this example
        ldhx #$C01F         ;Load last address of the row
        sthx LADDR          ; to LADDR
        ldhx #$C000         ;Load beginning address of the
                            ; row to H:X
        jsr  PRGRNGE        ;Call PRGRNGE routine
```

Example 5: Programming a Page

Example 5 shows how to program one full 64-byte page:

```
PRGRNGE equ  $1C09

        ldhx #$0000         ;Index offset into DATA array
        clra                ;Initial data value (-1)
Data_load:
        inca
        sta  DATA,x         ;Fill DATA array, 64 bytes data,
                            ; values to program into FLASH
        aix  #$1            ; (ie. 01,02,03,04,...,63,64)
        cphx #$40
        bne  Data_load

        mov  #$4,CPUSPD     ;fop = 2.0MHz in this example
        ldhx #$C03F         ;Load last address of the page
        sthx LADDR          ; to LADDR
        ldhx #$C000         ;Load beginning address of the
                            ; page to H:X
        jsr  PRGRNGE        ;Call PRGRNGE routine
```

Programming a Range Smaller than a Row

shows how to program $55 and $AA at locations $E004 and $E005, respectively.

```
PRGRNGE equ   $1C09

        mov   #$55,DATA
        mov   #$AA,DATA+1

        mov   #$0C,CPUSPD     ;fop = 6.0MHz in this example
        ldhx  #$E005          ;Load last address to LADDR
        sthx  LADDR
        ldhx  #$E004          ;Load beginning address to H:X
        jsr   PRGRNGE         ;Call PRGRNGE routine
```

ERARNGE

ERARNGE can be called to erase a page (64 bytes) or a whole array of FLASH. Registers H and X can be any address within the page or array to be erased. To select erase size, the MASS bit (bit 6) in the CTRLBYT is used. Setting the MASS bit selects the entire array erase. Clearing the MASS bit selects the page erase. This routine supports an internal operating frequency between 2.0 MHz and 8.4 MHz.

In this routine, both page erase time (t_{Erase}) and mass erase time (t_{MErase}) are set between 4 ms and 5.5 ms. The CPUSPD value is equal to f_{op} (in MHz) times 2 then rounded up to the next integer. For example if f_{op} is 3.1 MHz, the CPUSPD is 7. If f_{op} is 4.9152 MHz, the CPUSPD is 10 ($A).

Interrupts are masked (the I bit is set) and the COP is serviced in ERARNGE.

Entry Condition

CTRLBYT — For MASS erase, set bit 6. For page erase, clear bit 6.

H:X — Contains an address within a desired erase page or an array.

CPUSPD — Contains an integer value equal to f_{op} (in MHz) times 2 then rounded up to the next integer.

Exit Condition

None

Example 7: Erasing an Entire Array

Example 7 shows how to erase an entire array:

```
ERARNGE equ   $1C06

        mov   #$4,CPUSPD      ;fop = 2.0MHz in this example
        bset  6,CTRLBYT       ;Select Mass erase operation
        ldhx  #$E000          ;Load any Flash address to H:X
        jsr   ERARNGE         ;Call ERARNGE routine
```

Example 8: Erasing a Page

Example 8 shows how to erase a page from $E100 through $E13F:

```
ERARNGE equ   $1C06
        mov   #$0A,CPUSPD     ;fop = 4.9152MHz in this example
        bclr  6,CTRLBYT       ;Select Page erase operation
        ldhx  #$E121          ;Load any address within the
                              ; page to H:X
        jsr   ERARNGE         ;Call ERARNGE routine
```

NOTE: *If the FLASH locations which you want to erase are protected due to the value in the FLASH block protect register (FLBPR), the erase operation will not be successful. However when a high voltage (V_{tst}) is applied to the $\overline{IRQ}$ pin, the block protection is bypassed.*

When the FLASH security check fails in the normal monitor mode, the FLASH can be re-accessed by erasing the entire FLASH array. To override the FLASH security mechanism and erase the FLASH array using this routine, registers H and X must contain the address of the FLASH block protect register (FLBPR).

routines. It can, however, be called independently in user software. DELNUS uses two parameters stored in the accumulator (A) and the X register (X). Neither of these parameters is passed as an absolute value. The total delay (cycles) resulting from this routine is:

DELNUS = 3 × (A value) × (X value) + 8 cycles

where a value of A is 4 or greater and a value of X is 1 or greater. In the PRGRNGE and ERARNGE routines, the CPUSPD value (which is a frequency parameter) is loaded into A.

Because this routine is called from a jump table, three additional cycles are included in the DELNUS equation provided above.

Interrupts are not masked (the I bit is not set) and the COP is not serviced in DELNUS.

Initialization

A — Select A value between 4 and 255

X — Select X value between 1 and 255

Exit Condition

None

Example 9: Generating a Delay

Initialized A = 8 and X = 16 to generate 100 µs delay at f_{op} = 4 MHz

```
DELNUS  equ  $1C0C

        lda  #$8            ;[2]A=8
        ldx  #$10           ;[2]X=16
        jsr  DELNUS         ;[4]Call DELNUS routine
```

In this example, the total delay time is 8 + (3 × 8 × 16 + 8) cycles = 400 cycles (100 µs).

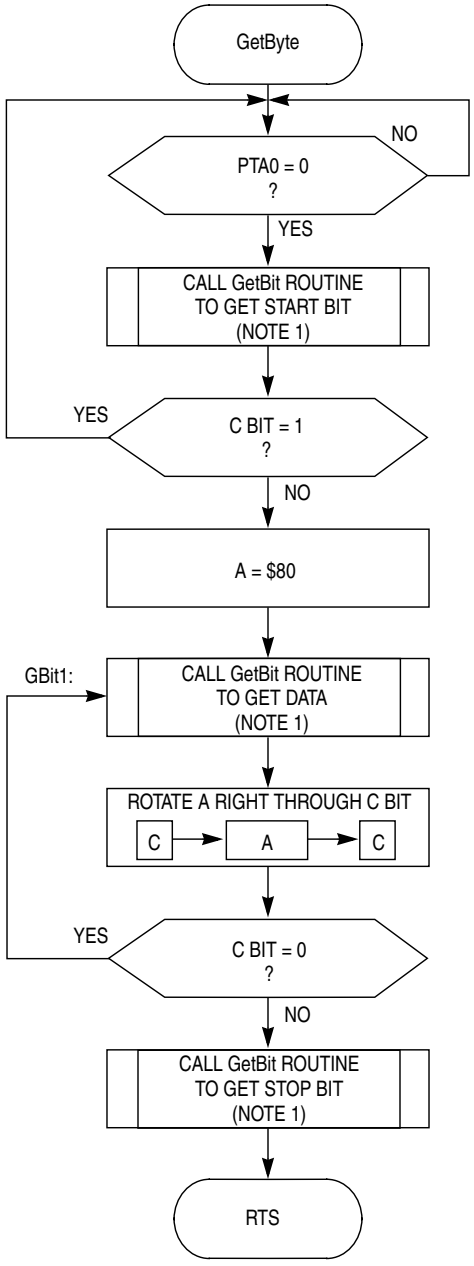
Table 5. Summary of On-Chip Flash Support Routines

	GetByte	RDVRRNG	PRGRNGE	ERARNGE	DELNUS
Routine Description	Get a byte data serially through PTA0	Read and/or verify a FLASH range	Program a FLASH range	Erase PAGE or entire array	Generate delay $3 \times A \times X + 8$ (cycles)
Call Address	$1C00	$1C03	$1C09	$1C06	$1C0C
Internal Operating Frequency (f_{op})	—	—	2 MHz – 8.4 MHz	2 MHz – 8.4 MHz	—
Hardware Requirement	Pullup on PTA0	For send-out option, pullup on PTA0	N/A	N/A	N/A
Entry Conditions	PTA0: Input (DDRA0 = 0)	H:X: First address of range LADDR: Last address of range A: A = $00 for send-out option or A ≠ $00 for verify option For send-out option PTA0: Input and 0 data bit (DDRA0=0, PTA0=0) For verify option, DATA array: Load data to be verified against FLASH read data	H:X: First address of range LADDR: Last address of range CPUSPD: f_{op} (in MHz) times 2 then rounded up to the next integer Data array: Load data to be programmed	H:X: Address within a page or an array to be erased CPUSPD: f_{op} (in MHz) times 2 then rounded up to the next integer CTRLBYT (bit 6): 1 = mass erase 0 = page erase	A: Value between 4 and 255 X: Value between 1 and 255

	GetByte	RDVRRNG	PRGRNGE	ERARNGE	DELNUS
Exit Conditions	A: Data received through PTA0 C-bit: Framing error indicator (error: c = 0)	A: Checksum H:X: Next FLASH address C-bit: Verify result indicator (success: c = 1) DATA array: Data replaced with FLASH read data (verify option)	H:X: Next FLASH address	H:X: No change	—
I Bit	—	I bit is set for send-out option	I bit is set	I bit is set	—
COP	Not Serviced	Serviced	Serviced	Serviced	Not Serviced
Subroutines Called	GetBit(1)	PutByte(1) for send-out option	DELNUS	DELNUS	—
RAM Variable	—	LADDR (2 bytes), DATA array (no size limitation, but all data must be in the zero page)	CPUSPD, LADDR (2 bytes), DATA array (no size limitation, but all data must be in the zero page)	CTRLBYT, CPUSPD	—
Stack Used (Including the Routine's Call)	6 bytes	9 bytes for verify option 11 bytes for send-out option	11 bytes	7 bytes	3 bytes

1. This routine is located in the monitor ROM.

On-Chip Routines Flowcharts



NOTES:
1. GetBit routine resides in monitor ROM

Figure 1. GetByte Routine

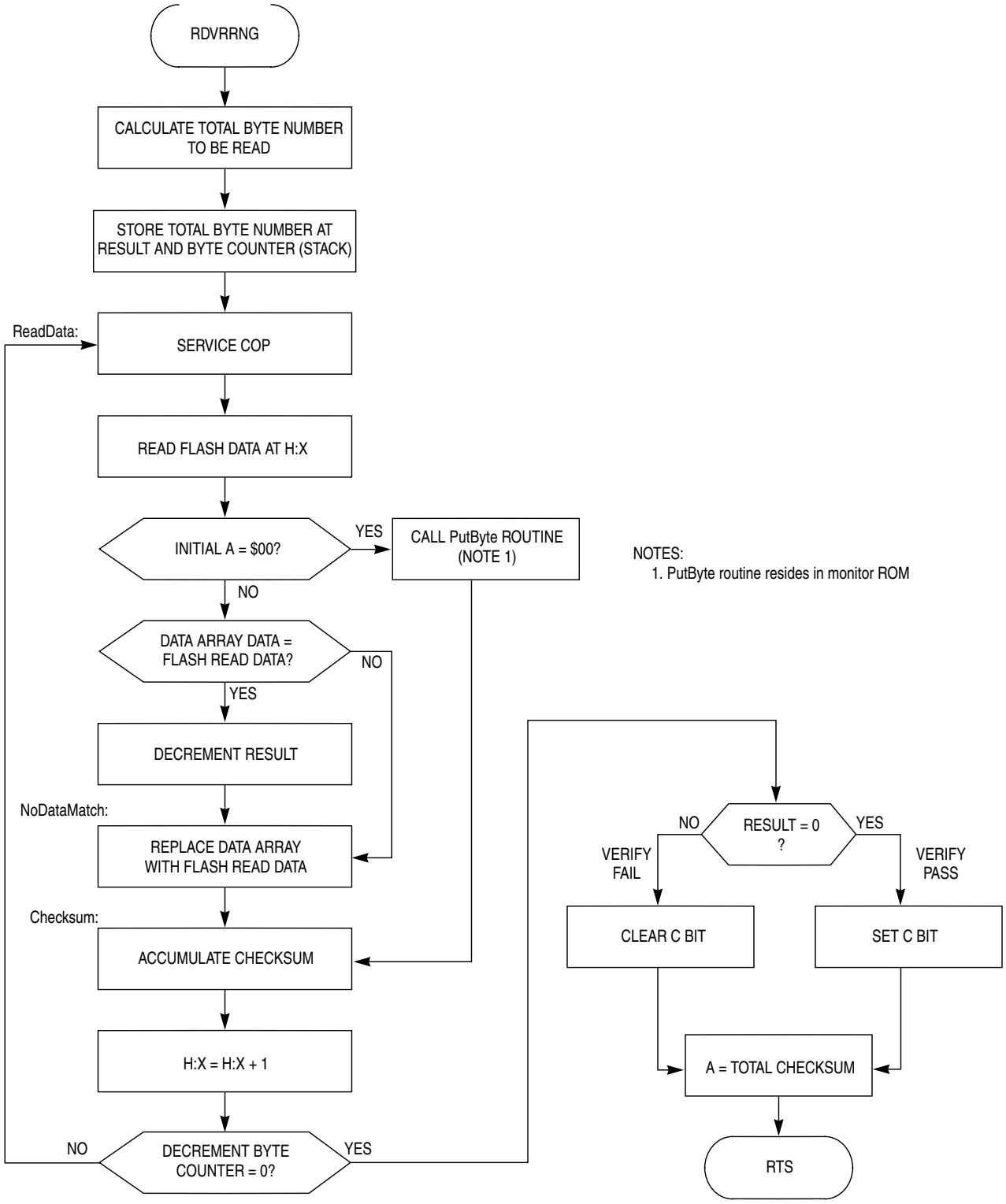
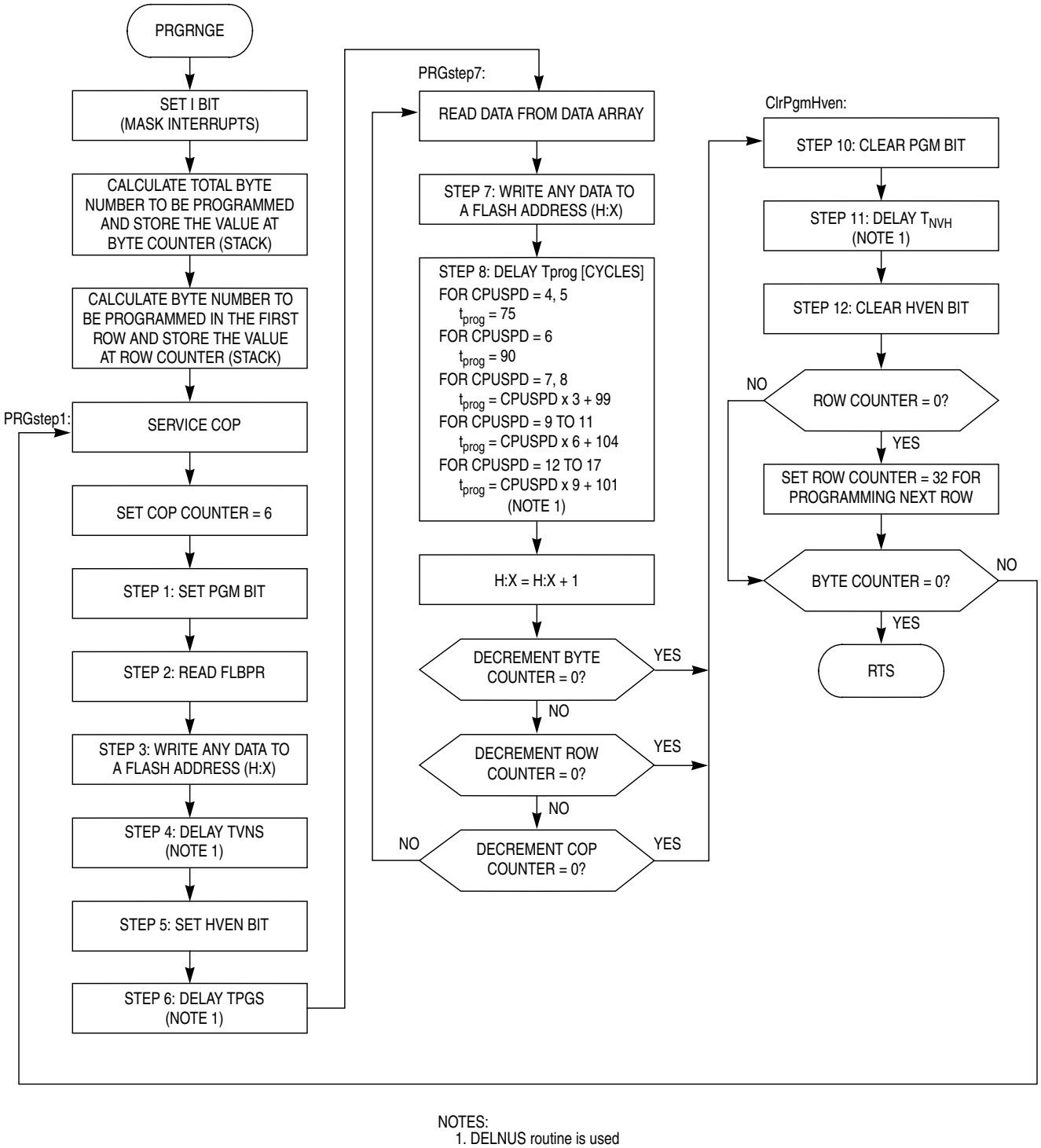
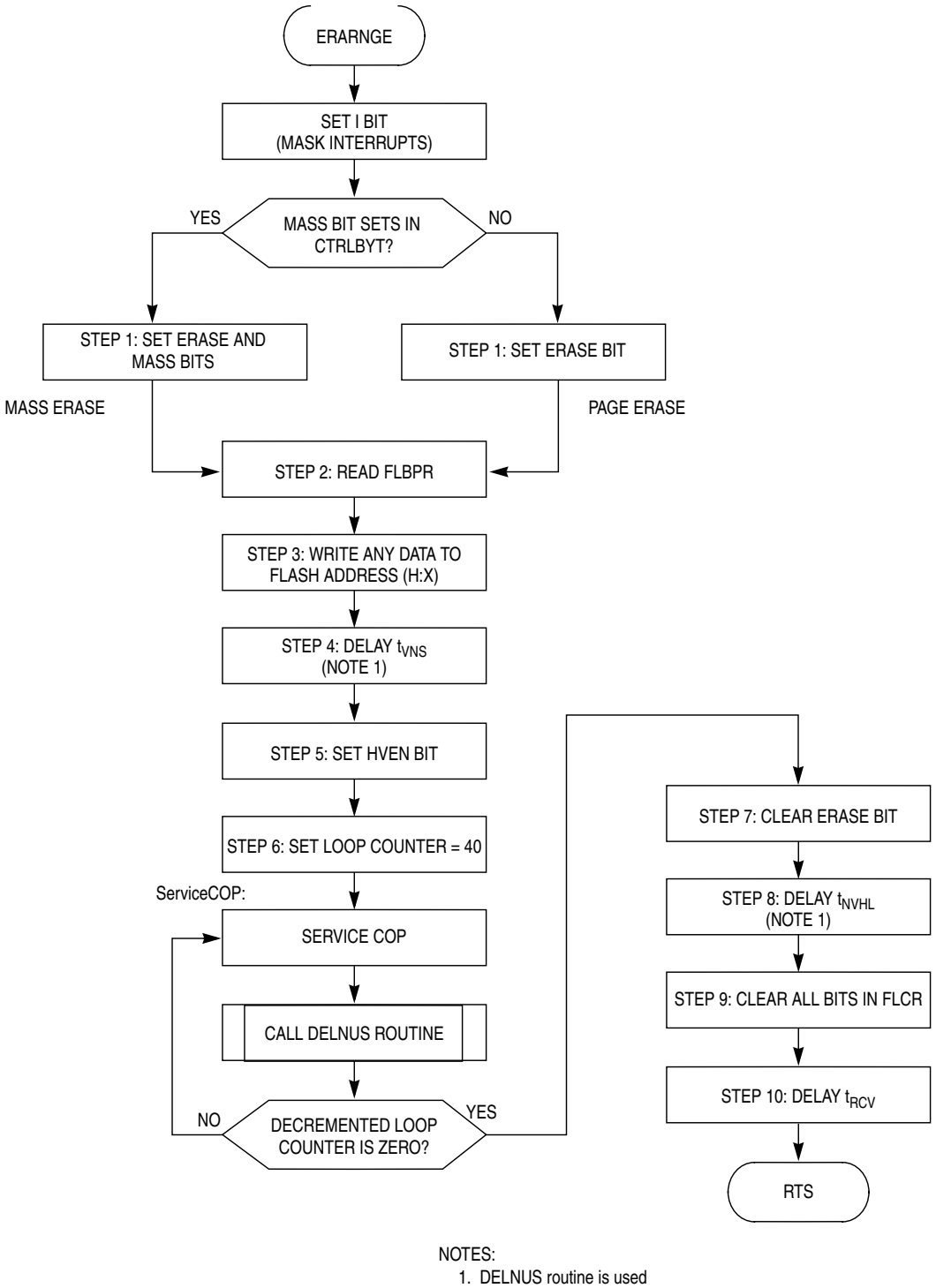


Figure 2. RDVRRNG Routine



NOTES:
1. DELNUS routine is used

Figure 3. PRGRNGE Routine



NOTES:
1. DELNUS routine is used

Figure 4. ERARNGE Routine

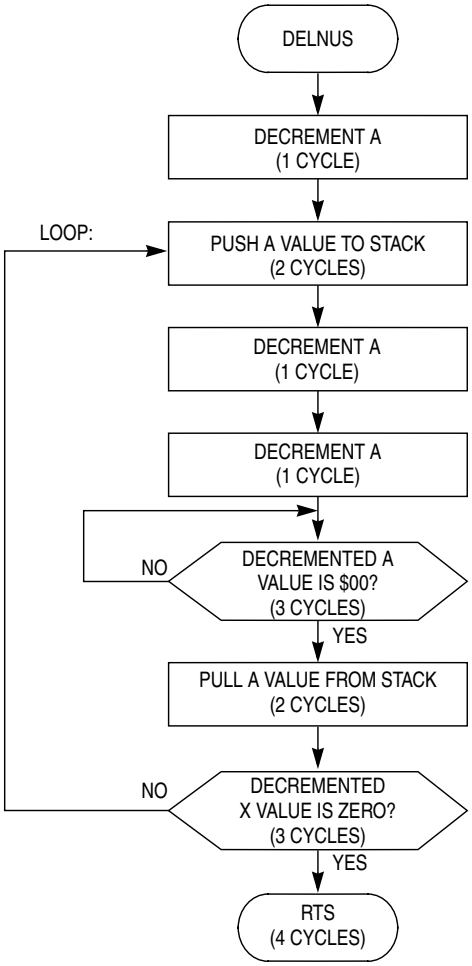


Figure 5. Flowchart of DELNUS Routine

On-Chip Routines Source Code

```
.pagewidth   98t
;*****************************************************************
;* PURPOSE: These routines are embedded into ROM to support FLASH
;*          erase, program and verify.
;*
;* TARGET DEVICE: HC908GZ8/16 and HC908GR16
;*
;* ASSEMBLER: CASM08Z
;* VERSION:   3.16
;*
;* GENERAL CODING NOTES:
;* A standard equate file "908GZ16v1r0.inc" is used to define all MCU
;* register and bit names. Bit names use all uppercase characters.
;* BCLR, BSET, BRCLR, and BRSET use the bit name alone while logical
;* instructions such as ORA use the bit name with a prefix of
;* lowercase "m" which is a bit position mask.
;*****************************************************************

;*****************************************************************
;* ASSEMBLER DIRECTIVES
;* (BASE, MACROS, SETS, CONDITIONS, ETC.)
;*****************************************************************
          base      10t      ;Change default to decimal
;*****************************************************************
;* INCLUDED FILES
;*****************************************************************

$NOLIST
          include "908GZ16v1r0.inc"
$LIST
;*****************************************************************
;* EQUATES
;*****************************************************************

MASSBIT       equ     6       ;MASS bit of CTRLBYT located in bit 6
ROWSIZE       equ     32      ;Programming ROW Size (number of bytes)

              org     RamStart+8 ;Leave 8-byte offset from start of RAM
                                 ; for future use
*FOLLOWING VARIABLES SET/ACCESSED BY USER
CTRLBYT       rmb     1       ;Control byte - bit 6 used for MASSBIT
CPUSPD        rmb     1       ;Used to indicate CPU bus speed
                              ; bus freq (in MHz) * 2 then round up
LADDR         rmb     2       ;Last address
DATA          rmb     ROWSIZE ;Allocation/Use of this space depends
                              ; on the application
* TOTAL DATA STRUCTURE BYTES:  4+ROWSIZE


;*****************************************************************
```

```
;* EQUATES
;*********************************************************************
;* The below parameters represent values that are passed to the delay
;* routine to generate delays required for the FLASH algorithm.
;* DELNUS generates a delay of (3*A*X)+5 cycles where A normally
;* holds CPUSPD and X is loaded with Txxxx from below

ELOOPS       equ     40      ;total TERASE for MASS and PAGE erase
TERASE       equ     17      ; uses DELNUS w/ TERASE, ELOOPS times
                             ; to allow COP service ~ every 100uS
TNVS         equ     2       ;FLASH PGM/ERASE to HVEN Setup Time
TPGS         equ     1       ;FLASH Program Hold Time
TNVH         equ     1       ;FLASH High-Voltage Hold Time
TNVHL        equ     17      ;FLASH High-Voltage Hold Time (MASS)
LoopCOP      equ     6       ;Service COP after every 6th byte prog'ed

;*********************************************************************
;* ROUTINES
;*********************************************************************

             org       FlashROM

;*********************************************************************
;* NAME: GetByte
;* PURPOSE:
;*   Get one byte data through PTA0 serially.  This routine supports
;*   a baud rate 14,400bps @ bus frequency 4.0MHz for GZ16 and
;*   a baud rate 9,600bps @ bus frequency 2.4576MHz for GR16
;* ENTRY CONDITIONS:
;*   PTA0 configured as an input
;* EXIT CONDITIONS:
;*   A contains a byte received when START bit is detected
;*   C-bit in CCR indicates a framing error
;*    If C-bit is cleared, a framing error is indicated because
;*    the STOP bit was detected as a 0 instead of a one
;*   PTA0 configured as an input
;* SUBROUTINES CALLED: GetBit
;* VARIABLES READ:
;* VARIABLES MODIFIED:
;* STACK USED: 6 (including the call to this routine)
;* SIZE: 20 bytes
;* DESCRIPTION: EXECUTED OUT OF ROM
;*   Once called, program will remain in GetByte until a byte is
;*   received.  Signal to start receiving a byte is a valid
;*   (low) START bit.
;*   This routine does not service COP.
;* NOTE: Cycle path for each bit reception must be kept the same to
;*   maintain a steady baud rate.
;*    bit timing for GZ16
;*     9 + (GZ GetBit) = 9 + 269 cycles = 278 cycles @ 4.0 MHZ=69.5 us
;*                     =14,388 bps (closest PC baud rate 14,400 bps)
;*    bit timing for GR16
;*     9 + (GR GetBit) = 9 + 247 cycles = 256 cycles @ 2.4576 MHZ
;*                     = 104 us = 9,600 bps
;*********************************************************************
```

```
            jsr     GetBit          ;[5+SUB] Check sense of START bit
            bcs     GetByte         ;[3] C-bit should be 0, else noise
            lda     #$80            ;[2] Rx byte done when 1 RORs into
GBit1:                              ;    C top of loop to get 8 bits
            jsr     GetBit          ;[5+SUB] Sense level of next bit
            rora                    ;[1] Rotate into A from left
            bcc     GBit1           ;[3] Continue 'till 1 RORs into C

            jsr     GetBit          ;[5+SUB] Sense level of STOP bit
            rts

;*********************************************************************
;*********************************************************************
;* NAME: RDVRRNG
;* PURPOSE: Read and/or verify a range of FLASH memory
;* ENTRY CONDITIONS:
;*   H:X contains a start address of the FLASH address range
;*   LADDR:LADDR+1 contains a last address of the FLASH address range
;*   The contents of A decides if read data is transferred serially
;*    via PTA0 (When A=0, PTA0 is used for serial transfer) or
;*    the data is verified against the DATA array in RAM
;*   DATA array must be in the zero page and its size must match the
;*    size of the range to be verified.
;*   If A=0, PTA0 is configured as an input (DDRA0=0) and
;*   data bit = 0 (PTA0=0)
;* EXIT CONDITIONS:
;*   A contains checksum
;*   C-bit in CCR indicates verify result when entry A is NOT zero
;*    If C-bit is set, the verify is successful
;*   DATA array contains read FLASH data when entry A is NOT zero
;*   H:X contains a next FLASH read address
;*   I bit for data send out operation
;* SUBROUTINES CALLED: PutByte
;* VARIABLES READ: LADDR:LADDR+1,DATA array
;* VARIABLES MODIFIED: DATA array
;* STACK USED: (include the call to this routine)
;*   9 bytes for Verify operation (entry A is NOT zero)
;*   11 bytes for data send out operation (entry A is zero)
;* SIZE: 67 bytes
;* DESCRIPTION: Executed out of ROM
;*   This routine services the COP, but there could still be a
;*   COP timeout under the following conditions:
;*   1) COP is not serviced within a proper period in user software
;*   2) COP set for short timeout and Read data is sent through PTA0
;* STACK FRAME:
;*     SP+1       [G] SADDR(hi) temp storage
;*     SP+2       [F] SADDR(lo) temp storage
;*     SP+3  SP+1 [E] ByteCount - decrements to zero
;*     SP+4  SP+2 [D] # of bad bytes - 0 on return means all were good
;*     SP+5  SP+3 [C] Checksum - sum of all data values read
;*     SP+6  SP+4 [B] Offset pointer into DATA array in RAM
;*     SP+7  SP+5 [A] Verify/Read flag - 1=verify/0=read
;*       |     |   |
;*       |     |   +--reference label in square brackets
```

```
;*       |     +---SP offset when SADDR not on stack
;*       +---------SP offset when SADDR on stack for temp storage
;*****************************************************************
RDVRRNG:     psha                ;Verify(1)/Read(0) flag to Stack [A]
             clra
             psha                ;Offset pointer into DATA array in
                                 ; RAM [B] (initially 0)
                                 ; increments from $00 to ByteCount
             psha                ;Initial Checksum to Stack [C]
                                 ;Calculate total # of bytes
             txa                 ;SADDR(lo) -> A
             sub    LADDR+1      ;SADDR(lo) - LADDR(lo) -> A
             nega                ;LADDR(lo) - SADDR(lo) -> A
             inca                ;change to 1-oriented vs 0-oriented
             psha                ;# of bytes to Stack [D] (# of bad)
                                 ; decrements to zero if all good
             psha                ;ByteCount to Stack [E]
                                 ; counter - decrements to zero
ReadData:
             sta    COPCTL       ;Service COP
             lda    ,x           ;Data from a FLASH location @ 0,X
             tst    5,sp         ;Check Read/Verify flag [A]
             beq    Serial       ;0 - send data through PTA0
                                 ;1 - verify against DATA in RAM
             pshx                ;Push SADDR(lo) to Stack [F]
             pshh                ;Push SADDR(hi) to Stack [G]
             ldx    6,sp         ;DATA array Pointer(lo) -> X
             clrh                ;H:X = 0:Pointer(lo)
             cmp    DATA,x       ;Compare FLASH data with DATA array
             bne    NoDataMatch  ;If not equal, skip decrement of [D]
             dec    4,sp         ;Data matched so decrement # of bad
NoDataMatch: sta    DATA,x       ;Replace DATA array value with
                                 ; value read from FLASH
             pulh                ;Restore SADDR(hi) pointer from [G]
             pulx                ;Now H:X = SADDR, A is FLASH data
             bra    Checksum     ;Skip serial send if in Verify mode

Serial:      jsr    PutByte      ;Read mode so send data to host

Checksum:    add    3,sp         ;FLASH data + checksum [C] -> A
             sta    3,sp         ;Update checksum [C] on stack
             inc    4,sp         ;Update offset into DATA array [B]
             aix    #1           ;Update pointer into FLASH (H:X)
             dec    1,sp         ;Decrement ByteCount [E]
             bne    ReadData     ;Loop until ByteCount=0

             pula                ;Deallocate [E]
             pula                ;# of bad [D] -> A, and deallocate
                                 ;If Verify OK, A = $00
             coma                ;$00 -> $FF if verify OK
             add    #1           ;$FF -> $00; C=1 if verify was OK
             pula                ;Checksum [C] -> A, and deallocate
             ais    #2           ;Deallocate [A] and [B]
             rts
```

```
;* NAME: PRGRNGE
;* PURPOSE:
;*   Program a FLASH address range. Bus frequency must be
;*   between 2.0MHz and 8.4MHz.
;* ENTRY CONDITIONS:
;*   H:X contains a start address of the FLASH address range
;*   LADDR:LADDR+1 contains a last address of the FLASH address range
;*   DATA array must be in the zero page and its size must match the
;*    size of the range to be programmed.
;*   CPUSPD equals bus frequency x 2 then rounded up
;* EXIT CONDITIONS:
;*   H:X contains the next address past LADDR; I-bit set
;* SUBROUTINES CALLED: DELNUS, ClrPgmHven
;* VARIABLES READ: CPUSPD, LADDR:LADDR+1, DATA array
;* STACK SIZE: 11 bytes (including this routine's call)
;* SIZE: 198 bytes (including ClrPgmHven routine)
;* DESCRIPTION: Executed out of ROM
;*   This routine allows passing of a range of addresses to PRGRNGE,
;*   which does not have to be on page boundaries, either beginning or
;*   end. i.e., passing $8010 to $8025 is valid.  This is to prevent
;*   program a non-FLASH address.  However, the total number of bytes
;*   to be programmed must be less or equal to the DATA array size.
;*   This routine services the COP, but there could still be a
;*   COP time out if the COP is not serviced within the proper period
;*   in user software.
;*****************************************************************
PRGRNGE:     sei                 ;set I bit to mask interrupts
                                 ;Calculate total # of bytes
             txa                 ;SADDR(lo) -> A
             sub    LADDR+1      ;SADDR(lo) - LADDR(lo) -> A
             nega                ;LADDR(lo) - SADDR(lo) -> A
             inca                ;change to 1-oriented vs 0-oriented
             psha                ;Byte Counter [A] (total bytes)
             clra                ;
             psha                ;DATA array index [B]
             pshx                ;temp save addr(lo)
             pshh                ;temp save addr(hi)
                                 ;Calculate total # of bytes in ROW
             txa                 ;SADDR(lo) -> A
             ldx    #ROWSIZE     ;ROWSIZE -> X
             clrh                ;H:A = 0:SADDR(lo)
             div                 ;A=H:A/X;r->H SADDR(lo)/#ROWSIZE
                                 ;remainder = # of bytes left in ROW
             pshh                ;remainder to stack for calculation
             txa                 ;ROWSIZE -> A
             sub    1,sp         ;ROWSIZE - Remainder -> A
             pulh                ;remainder not used, just deallocate
             pulh                ;recover temp addr(hi)
             pulx                ;recover temp addr(lo)
             psha                ;ROW Counter [C] bytes left in ROW
             psha                ;reserve space for COP Counter [D]

;* Current stack frame
;*           SP+1 [D] COP Counter - when 0, service COP
```

```
;*          SP+2 [C] ROW Counter - # bytes left in current row
;*          SP+3 [B] DATA array index - offset into RAM DATA array
;*          SP+4 [A] Byte Counter - # bytes left in program operation
;
;*****************************************************************
;* Loop top if 1st byte, time to service COP, or start of a new row
;* otherwise the loop top is at PRGstep7
;
PRGstep1:    sta    COPCTL         ;service COP
             lda    #LoopCOP       ;initialize COP Counter [D]
             sta    1,sp           ; counts down to 0
             lda    #mPGM
             sta    FLCR           ;[..w.] set PGM   (Prog Algo Step 1)
PRGstep2:    lda    FLBPR          ;[4] read FLBPR   (Prog Algo Step 2)

PRGstep3:    sta    ,x             ;[2] Write to a Flash address [H:X]
                                   ;    w/ any data  (Prog Algo Step 3)
             pshx                  ;[2] temp save addr(lo) to free up X
PRGstep4:    ldx    #TNVS          ;[2] Delay for time Tnvs
             lda    CPUSPD         ;[3]
             bsr    DELNUS         ;[4+(3*A*X)+5]    (Prog Algo Step 4)
PRGstep5:
             lda    #(mPGM+mHVEN) ;[2]
             sta    FLCR           ;[..w.] set HVEN  (Prog Algo Step 5)
PRGstep6:
             ldx    #TPGS          ;[2] Delay for time Tpgs
             lda    CPUSPD         ;[3]
             bsr    DELNUS         ;[4+(3*A*X)+5]    (Prog Algo Step 6)
             pulx                  ;[2] restore addr(lo)

;*****************************************************************
;* Loop top if this is not a new row and it is not time to service COP
;* PGM is already set and HVEN is already turned on
PRGstep7:    pshh                  ;[2] temp save addr(hi) [E]
             pshx                  ;[2] temp save addr(lo) [F]
;* Current stack frame
;*          SP+1 [F] Current addr (lo) temp store so H:X available
;*          SP+2 [E] Current addr (hi) temp store so H:X available
;*          SP+3 [D] COP Counter - when 0, service COP
;*          SP+4 [C] ROW Counter - # bytes left in current row
;*          SP+5 [B] DATA array index - offset into RAM DATA array
;*          SP+6 [A] Byte Counter - # bytes left in program operation
;
             clrh                  ;[1] clear upper half of H:X
             ldx    5,sp           ;[4] H:X = offset into DATA array
             lda    DATA,x         ;[3] Read data from a DATA array
             pulx                  ;[2] restore addr(lo) [F]
             pulh                  ;[2] restore addr(hi) [E]
             sta    ,x             ;[.w] write data to Flash addr
                                   ;                  (Prog Algo Step 7)
             pshh                  ;[2] temp save addr(hi) [E]
             pshx                  ;[2] temp save addr(lo) [F]
;*******************************
;* Compute Tprog based on bus speed
;*  for slowest bus speeds (5 or 6), use in-line delays rather than
```

```
PRGstep8:    ldx    #2            ;[2] initial default X value
             lda    CPUSPD        ;[3] bus speed const for comparisons
Case1:
             cmp    #5            ;[2] If CPUSPD=5, Tprog=75 cycles
             bls    SkipDELNUS    ;[3] skip to end of case
Case2:
             cmp    #6            ;[2] If CPUSPD=6, Tprog=90 cycles
             bhi    Case3         ;[3] if not, skip to Case3
             dbnzx  *             ;[6] X*3~ or 2*3~ = 6~
             nop                  ;[1] 1-cycle delay
             bra    SkipDELNUS    ;[3] skip to end of case
Case3:
             cmp    #8            ;[2] check for CPUSPD=7 or 8
             bhi    Case4         ;[3] if not, skip to Case4
             ldx    #1            ;[2]
             bra    DelayTprog    ;[3] Tprog=99+3*CPUSPD cycles
Case4:
             cmp    #11           ;[2] check for CPUSPD=9,10, or 11
             bhi    Case5         ;[3]  if not, skip to Case5
             ldx    #2            ;[2]
             bra    DelayTprog    ;[3] Tprog=104+6*CPUSPD cycles
Case5:
             ldx    #$03          ;[2] If CPUSPD=12,13,14,15 or 16
                                  ;      Tprog=101+9*CPUSPD cycles
DelayTprog:  bsr    DELNUS        ;[4+(3*A*X)+5]
;*
;*********************************
SkipDELNUS:  pulx                 ;[2] restore addr(lo) [F]
             pulh                 ;[2] restore addr(hi) [E]

;* Current stack frame
;*           SP+1 [D] COP Counter - when 0, service COP
;*           SP+2 [C] ROW Counter - # bytes left in current row
;*           SP+3 [B] DATA array index - offset into RAM DATA array
;*           SP+4 [A] Byte Counter - # bytes left in program operation
;
;*********************************
;* Byte programmed. Update pointers and counters on stack & check for
;** Done - go to RANGEstep10, turn off PGM & HVEN, cleanup stack & RTS
;** End-of-row - go PAGEstep10, turn off PGM & HVEN, loop to PRGstep1
;** More-in-row/not time for COP service - loop to PRGstep7
;** COP count=0 - turn off PGM & HVEN, loop to PRGstep1
;
;* balance timing from prev Flash write, to PGM bit clear
;                                                  (Prog Algo Step 9)
PRGstep9:    aix    #1            ;[2] point to next FLASH address
             inc    3,sp          ;[5] Increment DATA array index [B]
             dec    2,sp          ;[5] Decrement Row Counter [C]
             dec    1,sp          ;[5] Decrement COP Counter [D]
             dec    4,sp          ;[5] Decrement Byte Counter [A]

             beq    RANGEstep10   ;[3] 0 if done programming last byte

             tst    2,sp          ;[4] Row Counter=00? (end of a row)
```

```
            beq    PAGEstep10    ;[3] if so, cycle HVEN then continue

            tst    1,sp          ;[4] COP Counter=00?
            bne    PRGstep7      ;[3] If no, just continue programming
;* Bottom of loop; not done, not new row, and not time to service COP
;*********************************

;*********************************
;* time to service COP so cycle HVEN off and go to Prog Algo Step 1
            nop                  ;[1] 1~ delay to adjust timing
            sei                  ;[2] 2~ delay, I was already set
            bsr    ClrPgmHven    ;[4+11] time to write cycle that
                                 ;    clears PGM in ClrPgmHven
                                 ;ClrPgmHven clears PGM then HVEN
            jmp    PRGstep1      ;    then continue programming

;NOTE: DELNUS placed here to allow BSR instead of JSR for most calls
;*****************************************************************
;* NAME: DELNUS
;* PURPOSE: Generate delay  (3 * A * X) + 5 [cycles]
;* ENTRY CONDITIONS:
;*    A contains an integer value equal to 4 or higher
;*    X contains an integer value equal to 1 or higher
;* STACK USED: 3 bytes (including this routine's call)
;* SIZE: 10 bytes
;* DESCRIPTION: EXECUTED OUT OF ROM
;*   This routine is called from PRGRNGE and ERARNGE routines.
;*   For example when bus frequency = 8MHz, A=16, and X=17, the
;*   delay time is:
;*      delay time = (3 x 16 x 17) + 5 = 821 cycles (102.625us)
;*      remember to consider delays associated with setup and JSR/BSR
;*****************************************************************
DELNUS:      deca                ;[1] A - 1

Loop:        psha                ;[2] temp save
             deca                ;[1] original A - 2
             deca                ;[1] original A - 3
             dbnza  *            ;[3(orig A - 3)] (inner loop)
             pula                ;[2] recover original A - 1
             dbnzx  Loop         ;[3] (bottom of outer loop)
;* outer loop = (X(2+1+1+(3(A-3))+2+3)) = (X(9+(3A-9)) = 3 * X * A

             rts                 ;[4]
;*****************************************************************

RANGEstep10: nsa                 ;[3] total 7~ delay to match timing
             nsa                 ;[3]  from beq RANGEstep10 to
             nop                 ;[1]  PRGstep10

PAGEstep10:  lda    #2           ;[2] total 10~ delay -
             dbnza  *            ;[6] (2x3~) part of time Tprog
             sei                 ;[2] 2~ delay, I was already set
             bsr    ClrPgmHven   ;[4+11] time to write cycle that
                                 ;    clears PGM in ClrPgmHven
                                 ;ClrPgmHven clears PGM then HVEN
```

```
;*           SP+1 [D] COP Counter - when 0, service COP
;*           SP+2 [C] ROW Counter - # bytes left in current row
;*           SP+3 [B] DATA array index - offset into RAM DATA array
;*           SP+4 [A] Byte Counter - # bytes left in program operation
;
             lda    2,sp          ;check Row Counter=00? (end-of-row?)
             bne    CheckAddr     ;If not, check Byte Counter
             lda    #ROWSIZE      ;new row, init ROW Counter = ROWSIZE
             sta    2,sp          ;update Row Counter [C] on stack

CheckAddr:   lda    4,sp          ;check Byte Counter=00? (done?)
             bne    GoPRGstep1    ;if not done, go program next byte
;*******************************
;* Programming complete, cleanup stack and return
;
             ais    #4            ;deallocate A, B, C and D
             rts                  ;return from PRGRNGE
;* PRGRNGE DONE ******************

;*******************************
;* End of a row, go to Prog Algo Step 1 to start programming next row
;
GoPRGstep1:  jmp    PRGstep1      ;to top of loop
;*********************************************************************

;*********************************************************************
;* NAME: ClrPgmHven
;* This local sub-routine is a part of the FLASH programming
;* algorithm and called from PRGRNGE.  In this routine, PGM bit is
;* cleared, time Tnvh is waited and then HVEN bit is cleared.
;*********************************************************************
ClrPgmHven:  pshx                 ;[2] temp save Addr(lo) to free up X
PRGstep10:
             sei                  ;[2] 2~ delay, I was already set
             sei                  ;[2] 2~ delay, I was already set
             lda    #mHVEN        ;[2] clear PGM, leave HVEN=1
             sta    FLCR          ;[..w.] Clear PGM bit in FLCR
                                  ;                 (Prog Algo Step 10)
PRGstep11:   ldx    #TNVH         ;[2] delay for time Tnvh
             lda    CPUSPD        ;[3]              (Prog Algo Step 11)
             bsr    DELNUS        ;[4+(3*A*X)+5]
PRGstep12:   clra                 ;[1] pattern to clear HVEN
             sta    FLCR          ;[..w.] clr HVEN (Prog Algo Step 12)
             pulx                 ;restore Addr(lo)
             rts

;*********************************************************************
;* NAME: ERARNGE
;* PURPOSE:
;*  Erase a page or a whole array in FLASH memory. The bus frequency
;*  range must be between 2.0MHz and 8.4MHz.
;* ENTRY CONDITIONS:
;*   H:X contains a FLASH address within a page or an array to be
;*    erased
```

```
;*   Bit 6 in CTRLBYT selects MASS erase (1) or PAGE erase (0)
;*   CPUSPD equals bus frequency x 2 then rounded up
;* EXIT CONDITIONS:
;*   The contents of H:X (address passed) is preserved; I-bit set
;* SUBROUTINES CALLED: DELNUS
;* VARIABLES READ: CTRLBYT, CPUSPD
;* STACK USED: 7 (including the call to this routine)
;* SIZE: 70 bytes
;* DESCRIPTION: Executed out of ROM
;*   Does not check for a blank range before (to see if erase is
;*   necessary) or after (to see if erase was successful). This
;*   routine services the COP, (but the COP could still time out if
;*   it is not serviced correctly in the user software)
;*****************************************************************
ERARNGE:     sei                  ;block interrupts during erase
             pshx                 ;temp save addr(lo) to free up X

ERAstep1:    lda    #mERASE
             brclr  MASSBIT,CTRLBYT,PageErase
                                  ;if MASSBIT is set in the CTRLBYT,
             ora    #mMASS        ; sets MASS and ERASE bits in A
PageErase:   sta    FLCR          ;[..w.]          (Erase Algo Step 1)
                                  ; set ERASE only, or MASS and ERASE

ERAstep2:    lda    FLBPR         ;[4]             (Erase Algo Step 2)

ERAstep3:    sta    ,x            ;[.w]            (Erase Algo Step 3)
                                  ;latch addr for Flash page or block

ERAstep4:    ldx    #TNVS         ;[2] delay Tnvs  (Erase Algo Step 4)
             lda    CPUSPD        ;[4]
             bsr    DELNUS        ;[4+(3*A*X)+5)]

ERAstep5:    lda    FLCR          ;[4] leave MASS and ERASE as is
             ora    #mHVEN        ;[2] set HVEN
             sta    FLCR          ;[..w.]          (Erase Algo Step 5)

ERAstep6:                         ;delay Terase    (Erase Algo Step 6)
                                  ;slit up to allow COP service
             lda    #ELOOPS       ;[2] initialize Loop Counter
             psha                 ;[2] Loop Count on stack for calcs
                                  ;    using ' dec  1,sp' instruction

ServiceCOP:  sta    COPCTL        ;[4] service COP
             ldx    #TERASE       ;[2] about 100us delay
             lda    CPUSPD        ;[4]
             bsr    DELNUS        ;[4+(3*A*X)+5)]
             dec    1,sp          ;[5] decrement Loop Counter
             bne    ServiceCOP    ;[3] loop if Loop Count not zero
;* bottom of COP service loop
;* total Terase time = setup from HVEN=1 + loop + overhead to ERASE=0
;* = 5 + (ELOOPS(3*A*X + 27)) + 11    33,739~ @8MHz (Terase=4.217mS)

             pula                 ;[2] deallocate Loop Counter
ERAstep7:                         ;                (Erase Algo Step 7)
```

```
            and     #($FF-mERASE) ;[2] clear ERASE bit only
            sta     FLCR          ;[..w.]

ERAstep8:                         ;               (Erase Algo Step 8)
            ldx     #TNVHL        ;delay for time Tnvhl
            lda     CPUSPD        ; Tnvhl is used for both
            jsr     DELNUS        ;[4+(3*A*X)+5)] page and mass erase

ERAstep9:                         ;               (Erase Algo Step 9)
            clra                  ;[1] clear all bits in FLCR
            sta     FLCR          ;[..w.] next 3 instructions
                                  ; including last cycle of this
                                  ; instruction make at least 1us
                                  ; delay for Trcv
ERAstep10:                        ;              (Erase Algo Step 10)
            pulx                  ;[2] recover original addr(lo)
            sei                   ;[2] 2~ delay, I was already set
            rts                   ;[4] return from ERARNGE
;* ERARNGE DONE ******************
```

Software

This application note has a companion software file, AN2545SW.zip, available from the Motorola semiconductor website, http://motorola.com/sps.

This page is intentionally blank.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com