Freescale Semiconductor
Application Note

AN2635
Rev. 4, 10/2005

On-Chip FLASH Programming Routines

For MC68HC908LB8, MC68HC908QL4, MC68HC908QB8, MC68HC908QB4, MC68HC908QY8, and MC68HC908QY4A Series[1]

By: Kazue Kikuchi
MCU Applications Engineering
Austin, Texas

Introduction

This application note applies to the MC68HC908LB8, MC68HC908QL4, MC68HC908QB4/QB8/QY8, and MC68HC908QY4A Series[1] FLASH-based microcontroller units (MCUs). To program, erase, and verify FLASH, these 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 user software, what is performed, and what is returned as confirmation of routine execution. The software files are available as a zip file, AN2635SW, from the Freescale Semiconductor website: www.freescale.com

NOTE

With the exception of mask set errata documents, if any other Freescale Semiconductor 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.

1. MC68HC908QY4A Series includes MC68HC908**QY4A**, MC68HC908**QY2A**, MC68HC908**QY1A**, MC68HC908**QT4A**, MC68HC908**QT2A**, MC68HC908**QT1A**

freescale™ semiconductor

FLASH Overview

The FLASH cell used on these 0.5-μ MCUs is an industry-proven split-gate cell. 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}.

With the quick byte-programming time and the organization of the FLASH array into 32-byte rows, the entire 8-Kbyte memory can be programmed in less than one-half second. This type of FLASH is specified to withstand at least 10,000 program/erase cycles and has enhanced reliability over previous technology.

Usually, split-gate FLASH is programmed on a row basis and erased on a page basis. Also, an entire specified array can be mass erased. For the target MCUs, rows are 32 bytes and pages are 64 bytes (two rows of 32 bytes each).

Routines Supported in ROM

In the ROM, six routines are supported. 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.

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 PTA0. The receiving baud rate is the same as the baud rate used in monitor mode. In the GetByte routine, the GetBit routine is called to generate baud rates required for each MCU.

PutByte

This routine is used to send a byte serially on the general-purpose I/O PTA0. The sending baud rate is the same as the baud rate specified in monitor mode.

RDVRRNG

This routine is used to perform one of two options. Using the send-out option, this routine reads FLASH locations and sends the data out serially on the general-purpose I/O PTA0. Using to verify option, this routine verifies the FLASH data against data in a specific RAM location, which is referred to as a DATA array.

PRGRNGE

This routine is used to program a contiguous range of FLASH locations that is up to 32 bytes and in the same row. Programming data is first loaded into the DATA array. PRGRNGE can be used when the internal operating frequency (f_{op}) is between 1.0 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 1.0 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 ERARNGE routine.

Variables Used in the Routines

The RDVRNGE, PRGRNGE, and ERARNGE routines require certain registers and/or RAM locations to be initialized before calling the routines in the 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
RAM – RAM+7	Reserved	8	Reserved for future use
RAM+$8	CTRLBYT	1	Control byte setting erase size
RAM+$9	CPUSPD	1	CPU speed — the nearest integer of f_{op} (in MHz) × 4; for example, if f_{op} = 2.4576 MHz, CPUSPD = 10
RAM+$A, RAM+$B	LADDR	2	Last address of a 16-bit range
RAM+$C	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

RAM

In general, RAM in Table 1 indicates the RAM start address. See Table 2 for RAM start locations for specific MCUs. For example, the RAM start address for the MC68HC908LB8 (and each MCU currently in the table) is $80.

CTRLBYT

The control byte (CTRLBYT) is located at RAM address RAM+$8 and is used for the ERARNGE routine. In the case of the MC68HC908LB8, the CTRLBYT is located at $88. Bit 6 in this location is used to specify either MASS (1) or PAGE (0) erase. The other bits must be 0. If one or more of these bits (except bit 6) is initialized with 1, the erase operation is not executed.

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 RAM+$9. In the case of the MC68HC908LB8, the CPUSPD is located at $89. The CPUSPD value is the nearest integer of f_{op} (in MHz) times 4. For example, if f_{op} is 4.2 MHz, the CPUSPD value is 17. If f_{op} is 2.1 MHz, the CPUSPD value is 8. 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 RAM+$A and RAM+$B holds the last address of a range. The addresses RAM+$A and RAM+$B are the high and low bytes of the last address, respectively. In the case of MC68HC908LB8, the LADDR is located at $8A and $8B. LADDR is used for RDVRRNG and PRGRNGE routines.

DATA

DATA is the first location of the DATA array and is located at RAM address RAM+$C. For the MC68HC908LB8, the DATA is located at $8C. 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 2 provides necessary addresses used in the on-chip FLASH routines for each MCU type. Table 3 summarizes the six routines.

Table 2. MCU Type vs. Necessary Addresses Required for On-Chip FLASH Routines

MCU Name	RAM	GetByte	PutByte	RDVRRNG	PRGRNGE	ERARNGE	DELNUS
MC68HC908**LB8**	$80	$037E	$0381	$0384	$038A	$0387	$038D
MC68HC908**QL4**	$80	$2B7E	$2B81	$2B84	$2B8A	$2B87	$2B8D
MC68HC908**QY4A** Series(1)	$80	$2800	$280F	$2803	$2809	$2806	$280C
MC68HC908**QB4/QB8,** MC68HC908**QY8**	$80	$2800	$280F	$2803	$2809	$2806	$280C

NOTES:
1. MC68HC908QY4A Series includes MC68HC908**QY4A**, MC68HC908**QY2A**, MC68HC908**QY1A**, MC68HC908**QT4A**, MC68HC908**QT2A**, MC68HC908**QT1A**

Table 3. Summary of On-Chip FLASH Support Routines

	GetByte	PutByte	RDVRRNG	PRGRNGE	ERARNGE	DELNUS
Routine Description	Get a data byte serially through PTA0	Send a data byte serially through PTA0	Read and/or verify a FLASH range	Program a FLASH range (maximum 32 bytes in a row)	Erase a PAGE or entire array	Generate delay $3 \times A \times X + 8$ (cycles)
Internal Operating Frequency (f_{op})	—	—	—	1 MHz to 8.4 MHz	1 MHz to 8.4 MHz	—
Hardware Requirement	Pullup on PTA0	Pullup on PTA0	For send-out option, pullup on PTA0	N/A	N/A	N/A
Entry Conditions	PTA0: Input (DDRA0 = 0)	PTA0: Input and 0 data bit (DDRA0 = 0, PTA0 = 0) A: data to be sent	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: the nearest integer f_{op} (in MHz) times 4 Data array: Load data to be programmed	H:X: Address within a page or an array to be erased CPUSPD: the nearest integer f_{op} (in MHz) times 4 CTRLBYT: $40 = MASS erase $00 = PAGE erase	A: Value between 4 and 255 X: Value between 1 and 255

Table 3. Summary of On-Chip FLASH Support Routines (Continued)

	GetByte	PutByte	RDVRRNG	PRGRNGE	ERARNGE	DELNUS
Exit Conditions	A: Data received through PTA0 C-bit: Framing error indicator (error: C = 0)	A, X: No change PTA0: Input and 0 data bit (DDRA0 = 0, PTA0 = 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	I bit is set	—
COP	Not Serviced	Not Serviced	Serviced	Serviced	Serviced	Not Serviced
Subroutines Called	GetBit	—	PutByte for send-out option	—	DELNUS	—
RAM Variable	—	—	LADDR (2 bytes), DATA array (no size limitation as long as in the zero page)	CPUSPD, LADDR (2 bytes), DATA array (maximum 32 bytes)	CTRLBYT, CPUSPD	—
Stack Used (Including the Routine's Call)	6 bytes	4 bytes	9 bytes for verify option 11 bytes for send-out option	9 bytes	7 bytes	3 bytes

GetByte

GetByte is a routine that receives a byte on the general-purpose I/O PTA0, and the received value is returned to the calling routine in the accumulator (A). This routine is also used in monitor mode so that it expects the same non-return-to-zero (NRZ) communication protocol and baud rates.

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. The user software is responsible for handling such errors.

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.

To provide a specific communication baud rate, GetByte calls the GetBit subroutine. In the GetByte routine, two different clock sources, internal clock and external clock, are supported. For example, the MC68HC908LB8 usually has a trimmed internal bus clock of 4 MHz and an external bus clock of 2.4576 MHz. For the MCU to distinguish which clock source is currently selected, the ECGST (external clock generator status) bit in the OSCSTAT (oscillator status register) is monitored in the GetBit subroutine. When ECGST bit is set, the external clock is selected as a clock source. When the bit is cleared, the internal clock is selected.

The baud rate is defined by f_{op} divided by a constant value, which is specified in the development support section in the device data sheet. In the case of the MC68HC908LB8, the baud rate of an internal clock source is defined by f_{op} divided by 417. If the internal bus clock is 4 MHz, the baud rate is 4 MHz/417 = 9592. Therefore, the closest PC baud rate is 9600. On the other hand, the baud rate of an external clock source is f_{op} divided by 256. When an external bus clock is 2.4576 MHz, the baud rate is 2.4576 MHz/256 = 9600.

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 development support section in the device data sheet.

Entry Condition

PTA0 — This pin must be configured as an input and pulled up in hardware.

Exit Condition

A — Contains data received from PTA0.

C bit — Usually 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  $037E          ;LB8 GetByte jump address

        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

After GetByte is called, the program will remain in this routine until a START bit (0) is detected and a complete character is received.

PutByte

PutByte is a routine that receives a byte on the general-purpose I/O PTA0. The sent value must be loaded into the accumulator (A) before calling this routine. This routine is also used in the monitor mode. Therefore, it uses the same non-return-to-zero (NRZ) communication protocol. The communication baud rates are the same as those described in GetByte.

To use this routine, some hardware setup is required. The general-purpose I/O PTA0 must be pulled up and configured as an input and the PTA0 data bit must be initialized to 0.

Interrupts are not masked and the COP is not serviced in the PutByte routine. User software should ensure that interrupts are blocked during character transmission.

Entry Condition

A — Contains data sent from PTA0

PTA0 — This pin must be configured as an input and pulled up in hardware and the PTA0 data bit must be initialized to 0.

Exit Condition

A and X — are restored with entry values.

Example 2: Sending a Byte Serially

Example 2 shows how to send a byte ($55) serially on PTA0:

```
PutByte equ  $0381          ;LB8 PutByte jump address

        bclr 0,DDRA         ;Configure port A bit 0 as an input
        bclr 0,PTA          ;Initialize data bit to zero PTA0=0
        lda  #$55           ;Load sent data $55 to A
        jsr  PutByte        ;Call PutByte routine
```

RDVRRNG

When using the RDVRRNG routine, the user must select one of the following function options:

- Send-out option — Used to read a range of FLASH locations and to send the read data to a host through PTA0 by using the PutByte routine.
- 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 PTA0. The communication baud rate is the same as the baud rate described in the PutByte 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 for each byte of FLASH and the DATA array is replaced by the data read from FLASH. If the data does not match the corresponding value, the data read from FLASH can be confirmed in the DATA array. All data in the DATA array must be in the zero page, but a range can be beyond a row size or a page size.

Carry (C) Bit and Checksum

The first 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 A upon return from the function.

Interrupts are not masked. The COP is serviced in RDVRRNG. The first COP is serviced at 23 bus cycles after this routine is called in the user software. 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 immediately 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 3: Verify Option

Example 3 shows how to use the verify option:

```
RDVRRNG equ  $0384          ;LB8 RDVRRNG jump address

        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 4: Send-Out Option

Example 4 shows how to use the send-out option:

```
RDVRRNG equ  $0384          ;LB8 RDVRRNG jump address

        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
```

PRGRNGE

PRGRNGE is used to program a range of FLASH locations with data loaded into the DATA array. The range must be less-than or equal-to 32 bytes. All bytes that will be programmed must be in the same row. 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 1.0 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 the nearest integer of f_{op} (in MHz) multiplied by 4. For example, if f_{op} is 2.4576 MHz, the CPUSPD value is 10 ($0A). If f_{op} is 8.0 MHz, the CPUSPD value is 32 ($20).

Table 4. t_{prog} vs. Bus Frequency

	Operating Bus Freq. (f_{op})	CPUSPD	t_{prog} (Cycles)	t_{prog}
Case 1	1.0 MHz $\leq f_{Bus} <$ 1.125 MHz	4	38	33.8 µs $< t_{prog} \leq$ 38.0 µs
Case 2	1.125 MHz $\leq f_{Bus} \leq$ 8.4 MHz	5 to 34	8 x CPUSPD + 5	32.1 µs $\leq t_{prog} \leq$ 40.0 µs

In PRGRNGE, the high programming voltage time is enabled for less than 125 µs when programming a single byte at any operating bus frequency between 1.0 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 the next erase. For more information, refer to the 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 perform 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

verifying operations. The RDVRRNG routine can be used to verify a programmed FLASH range against the DATA array.

Interrupts are masked and the COP is serviced in this routine. The first COP is serviced at 59 bus cycles after this routine is called in the user software.

Entry Condition

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

LADDR — Contains the last address in a range.

CPUSPD — Contains the nearest integer value of f_{op} (in MHz) times 4.

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 5:
Programming a Row

Example 5 shows how to program one full 32-byte row:

```
PRGRNGE equ  $038A          ;LB8 PRGRNGE jump address

        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  #$0A,CPUSPD    ;fop = 2.4576MHz in this example
        ldhx #$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 6:
Programming a Range Smaller than a Row

PRGRNGE can be used to program a range less than 32 bytes. Example 6 shows how to program $55 and $AA at location $E004 and $E005, respectively.

```
PRGRNGE equ  $038A          ;LB8 PRGRNGE jump address

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

        mov  #$18,CPUSPD    ;fop = 6.0MHz in this example
        ldhx #$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, CTRLBYT is used. Writing $40 to CTRLBYT selects the entire array (MASS) erase. Writing $00 to CTRLBYT selects the page erase. When other values are written to CTRLBYT, the erase operation is not executed. This routine supports an internal operating frequency between 1.0 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 the nearest integer of f_{op} (in MHz) times 4. For example if f_{op} is 3.1 MHz, the CPUSPD is 12 ($0C). If f_{op} is 4.9152 MHz, the CPUSPD is 20 ($14).

Interrupts are masked and the COP is serviced in ERARNGE. The first COP is serviced on (40+3xCPUSPD) bus cycles after this routine is called in the user software.

Entry Condition

CTRLBYT — For MASS erase, write $40. For PAGE erase, write $00.

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

CPUSPD — Contains the nearest integer value of f_{op} (in MHz) times 4.

Exit Condition

None

Example 7: Erasing an Entire Array

Example 7 shows how to erase an entire array:

```
ERARNGE equ  $0387          ;LB8 ERARNGE jump address

        mov  #$08,CPUSPD    ;fop = 2.0MHz in this example
        mov  #$40,CTRLBYT   ;Select Mass erase operation
        ldhx #$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  $0387          ;LB8 ERARNGE jump address
        mov  #$14,CPUSPD    ;fop = 4.9152MHz in this example
        mov  #$00,CTRLBYT   ;Select Page erase operation
        ldhx #$E121         ;Load any address within the
                            ; page to H:X
        jsr  ERARNGE        ;Call ERARNGE routine
```

If the FLASH locations that 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).

DELNUS

DELNUS is a delay routine used in support of the ERARNGE routine. It can, however, be called independently in the 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 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 above equation.

Interrupts are not masked 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

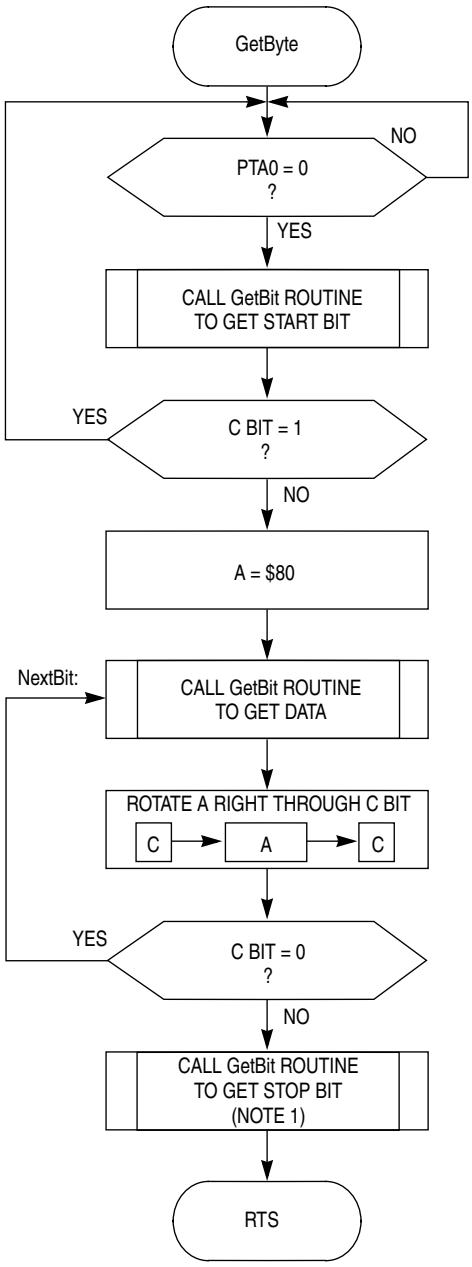
Initialized A = 16 and X = 8 to generate 100 μs delay at f_{op} = 4 MHz

```
DELNUS  equ  $038D          ;LB8 DELNUS jump address

        lda  #$10           ;[2]A=16
        ldx  #$08           ;[2]X=8
        jsr  DELNUS         ;[4]Call DELNUS routine
```

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

On-Chip Routines Flowcharts



NOTES:
1. When C bit is 0, communication has a framing error.

Figure 1. GetByte Routine

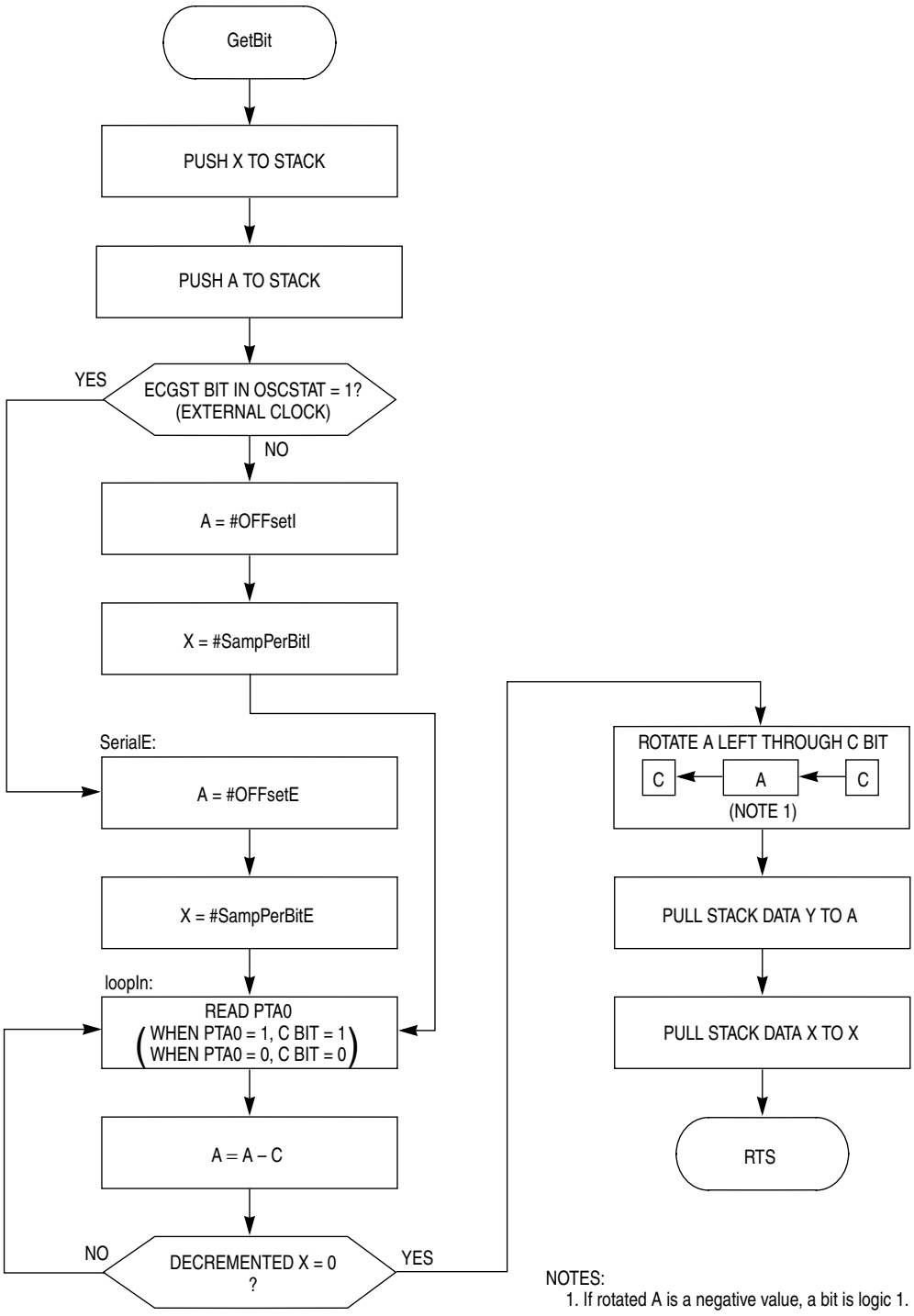


Figure 2. GetBit Routine

On-Chip Routines Flowcharts

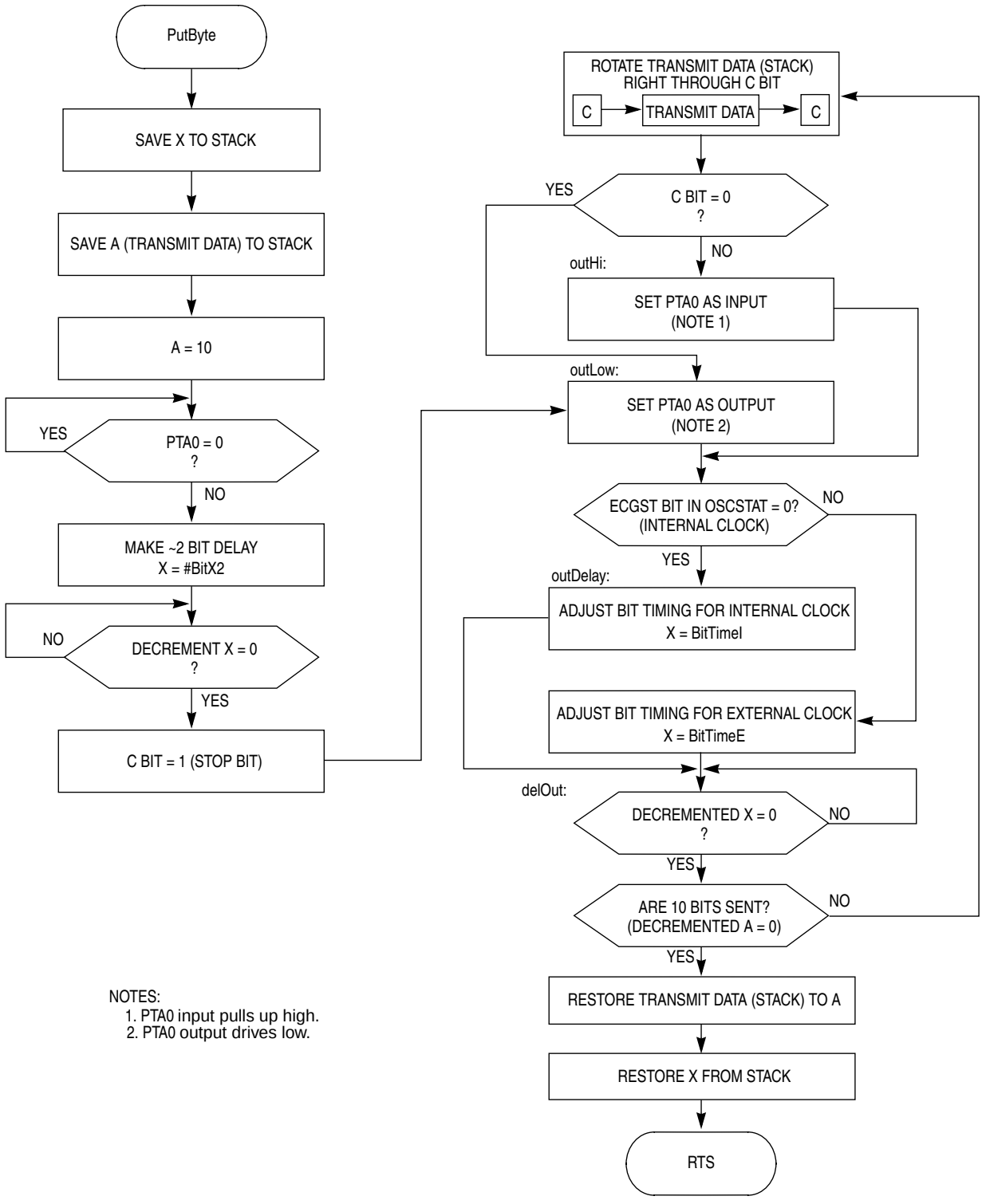


Figure 3. PutByte Routine

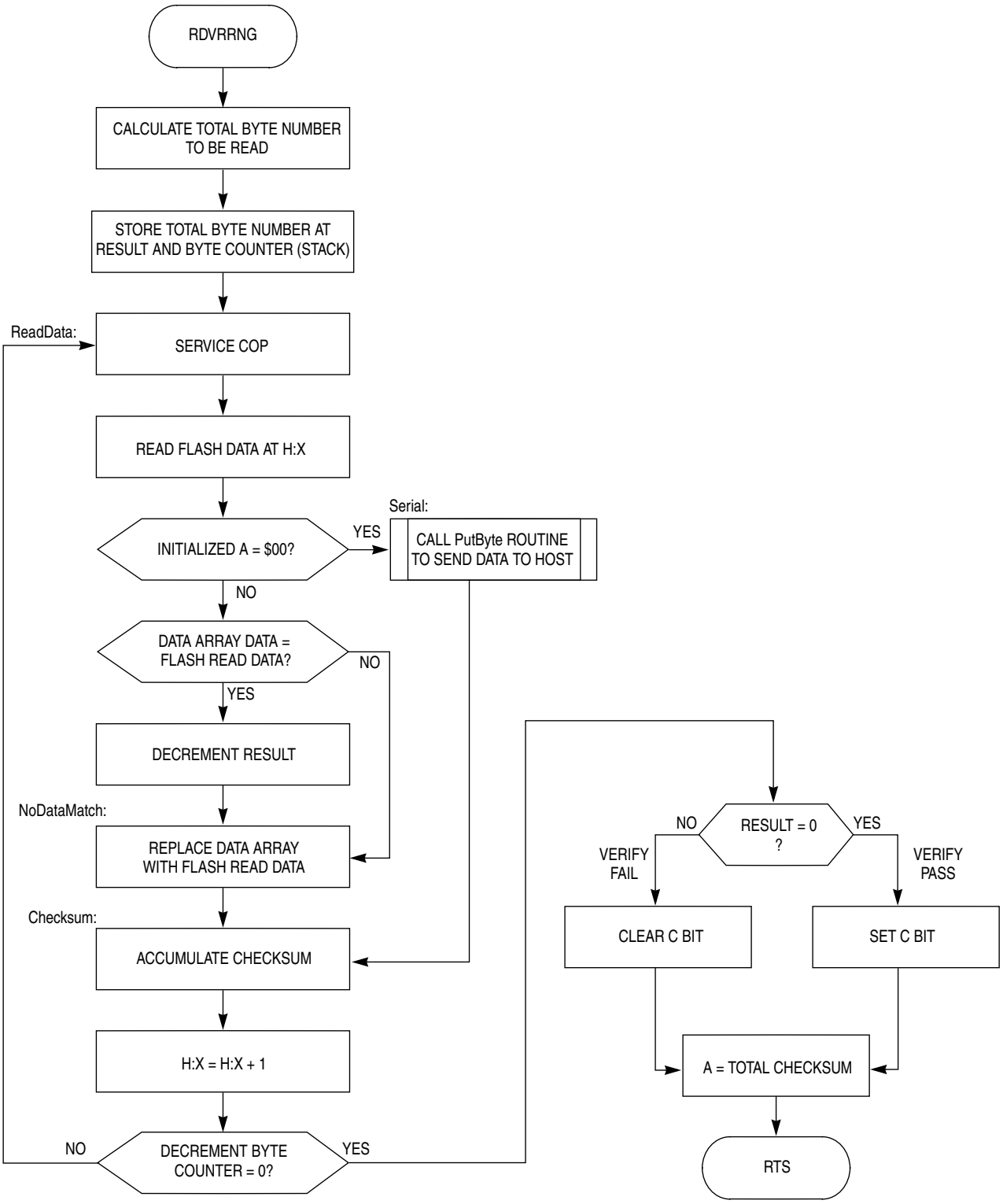


Figure 4. RDVRRNG Routine

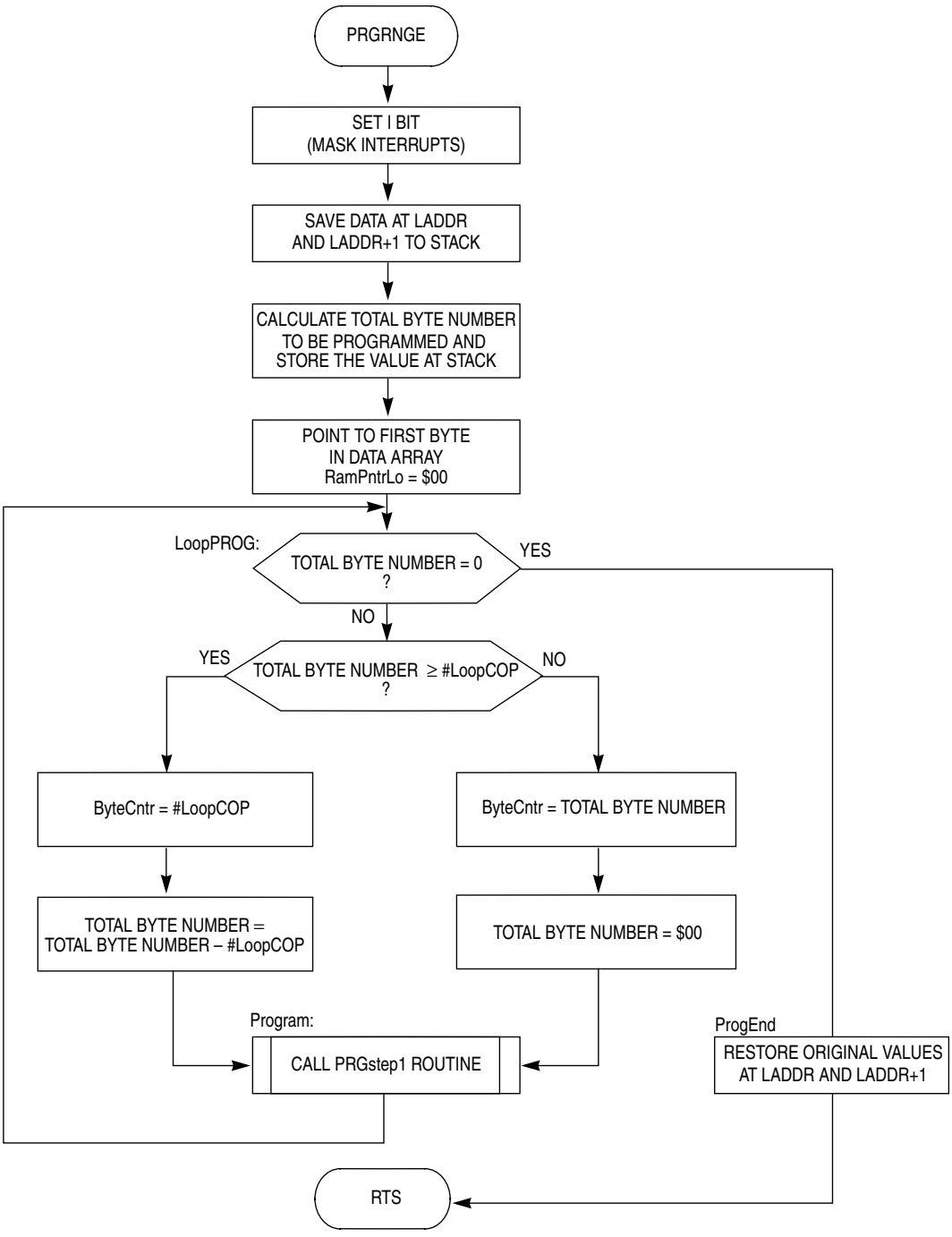


Figure 5. PRGRNGE Routine, Part 1

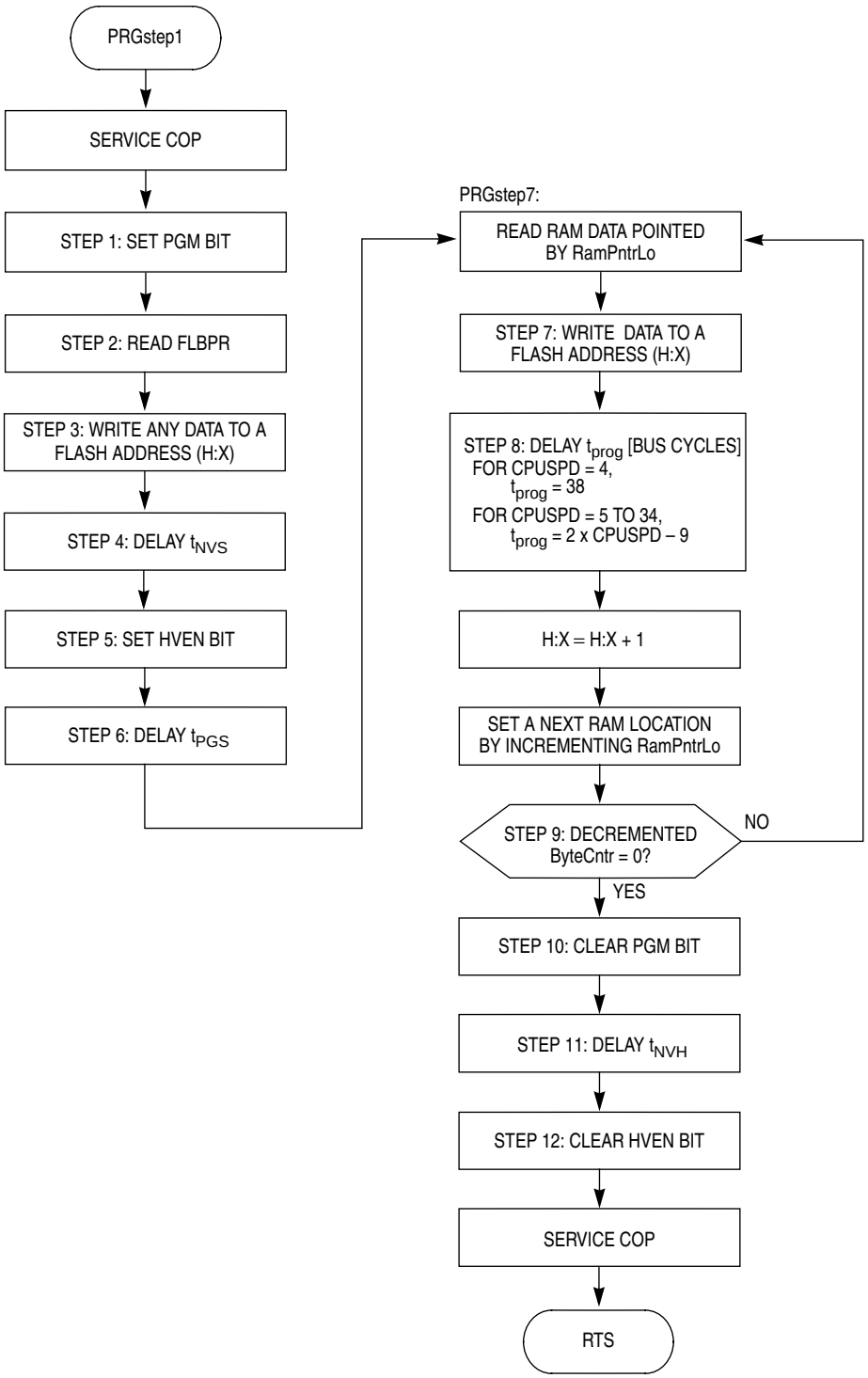


Figure 6. PRGRNGE Routine, Part 2

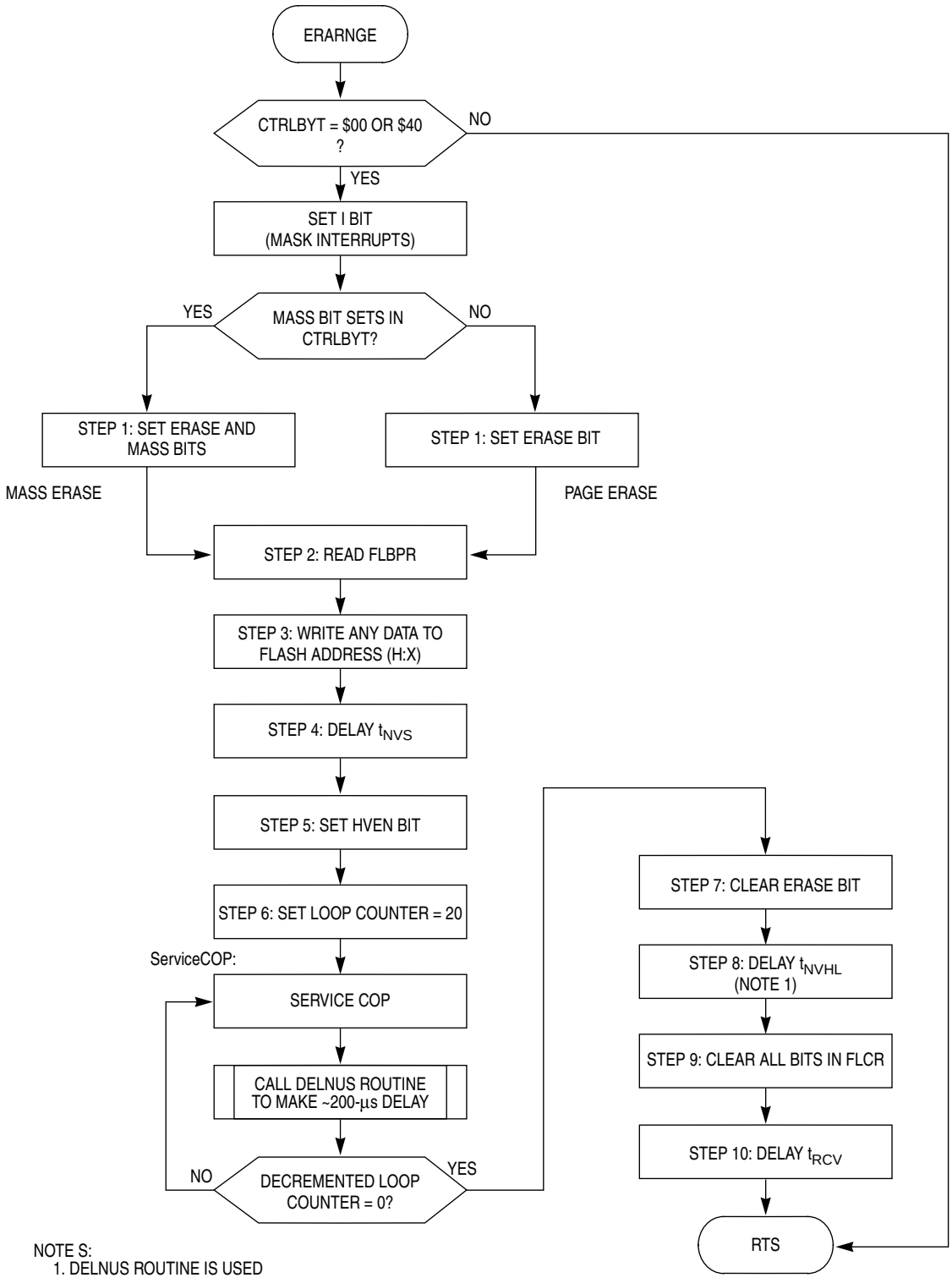


Figure 7. ERARNGE Routine

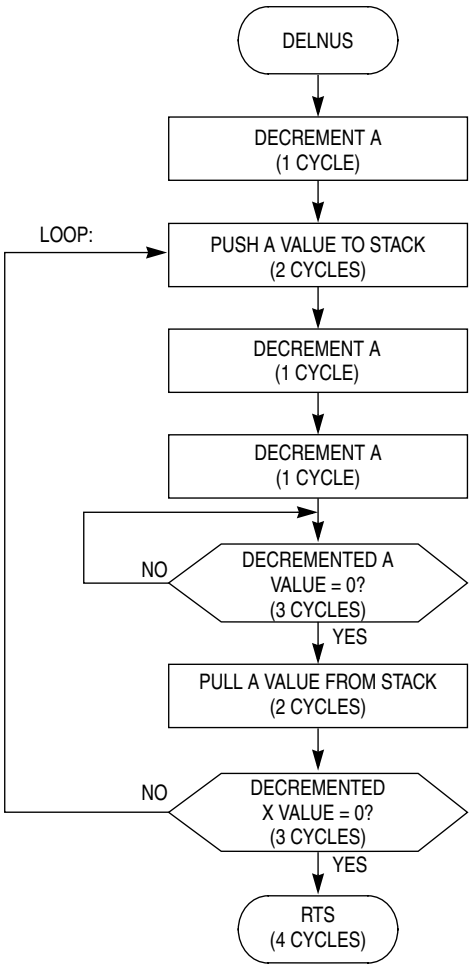


Figure 8. DELNUS Routine

On-Chip Routines Source Code

The following source code is for the MC68HC908LB8 on-chip routines. Because other MCUs support different communication baud rates, GetBit and PutByte routines are slightly different. However, these routines are built in the same manner.

```
.pagewidth   98t
;*********************************************************************
;* PURPOSE: This program has the HC908LB8 FLASH program, erase, verify
;*          routines and serial communication routines.
;*
;* TARGET DEVICE: HC908LB8
;*
;* ASSEMBLER: P&E Microsystems CASM08Z
;* VERSION: 3.16
;*
;* GENERAL CODING NOTES:
;* A standard equate file "908LB86vXrY.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 "908LB8v0r2.inc"
$LIST
;*********************************************************************
;* EQUATES for ROM Assigned Locations
;*********************************************************************
;* ROM Assigned Location
;*
JumpTable:    equ   $037E         ;jump table start address
FlashROM:     equ   JumpTable+$1B ;FLASH ROM start address

;*********************************************************************
;* EQUATES and VARIABLES for GetBit and PutByte routines
;* Constants used in the GetBit and PutByte routines:
;* NOTE: changing the following parameters will alter the baud rate!
;*  External clock (Ext) values set for 9600 baud @ 2.4576MHz bus rate
;*  Internal clock (Int) values set for 9600 baud @ 4.0MHz bus rate
;*********************************************************************
SampPerBitE: equ    22           ;samples per bit time (Ext)
SampPerBitI: equ    38           ;(Int) used in GetBit
OffsetE:     equ    15           ;~70% SampPerBit (Ext) and (Int)
OffsetI:     equ    27           ; used in GetBit
```

```
;* more than Offset samples = 1 means bit is detected as a logic 1
BitX2:       equ    210          ;delay count for ~2 bit times
;* 2 bit time is not accurate
BitTimeE:    equ    76           ;delay count for 1 bit time (Ext) and
BitTimeI:    equ    129          ; (Int) used in PutByte
BrkTimeE:    equ    232          ;delay count for 10 bit times (Ext)
BrkTimeI:    equ    123          ; and (Int) used in EchoBrk

;*****************************************************************
;* EQUATES and VARIABLES for FLASH routines
;*****************************************************************

DATSTRC:     equ     RamStart+8  ;leave 8-byte offset from start of
                                 ; RAM for future requirement
MASSBIT:     equ     6           ;MASS bit of CTRLBYT located in bit 6
ROWSIZE:     equ     32          ;FLASH ROW size

             org     DATSTRC
;* The following variables set by user
CTRLBYT:     rmb     1           ;control byte for erase operation
                                 ; selection
CPUSPD:      rmb     1           ;CPU bus speed (nearest integer of
                                 ; bus freq (in MHz) * 4)
LADDR:       rmb     2           ;last address
DATA:        rmb     ROWSIZE     ;allocation/use of this space depends
                                 ; on a device
RamPntrLo:   equ     LADDR       ;LADDR loc. reused as RAM pointer in
                                 ; PRGRNGE routine
ByteCntr:    equ     LADDR+1     ;LADDR+1 loc. reused as Byte Count in
                                 ; PRGRNGE routine

;* These times are for use by ERARNGE
LoopErase:   equ     20          ;total Terase time (~ 4ms)
                                 ; =20+(EraseLOOP*(3*CPUSPD*TERASE+26))
TERASE:      equ     17          ;FLASH erase time between COP service
                                 ; COP is serviced every ~200 us =
                                 ; 3*CPUSPD*TERASE+26 (bus cycles)
TNVHL:       equ     9           ;FLASH high-voltage hold time (>= 100us)
                                 ; = 3*SPUSPD*TNVHL+19 (bus cycles)
LoopCOP:     equ     6           ;COP is serviced when LoopCOP reaches
                                 ; to zero

;*****************************************************************
;* JUMP TABLE
;*****************************************************************
             org    JumpTable

ByteGet:     jmp    GetByte      ;receive one byte data from a host
BytePut:     jmp    PutByte      ;send one byte data to a host
RNGRDVR:     jmp    RDVRRNG      ;read/verify FLASH data
RNGEERA:     jmp    ERARNGE      ;erase FLASH
RNGEPRG:     jmp    PRGRNGE      ;program FLASH
NUSDEL:      jmp    DELNUS       ;generate delay

*****************************************************************
* ROUTINES
*****************************************************************

             org       FlashROM
```

```
;*********************************************************************
;* NAME: GetByte
;* PURPOSE:
;*   Get one byte data through PTA0 serially.  This routine supports
;*   a baud rate 9600 bps at internal 4MHz and external 2.4576MHz bus
;*   frequencies.  A clock is distinguished by the state of ECGST bit
;*   (bit 0) in OSCSTAT.  When ECGST=1, an external clock is selected.
;* 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 1
;*   PTA0 configured as an input
;* SUBROUTINES CALLED: GetBit
;* VARIABLES READ:
;* VARIABLES MODIFIED:
;* STACK USED: 6 (including the call to this routine)
;* SIZE: 18 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.
;*     When OSCSTAT[0]=0 (internal clock is selected):
;*       9+(28+10*38)= 417 cycles @ 4.0 MHZ=104.3 us=9592 bps
;*                              (closest PC baud rate 9,600 bps)
;*     When OSCSTAT[0]=1 (external clock is selected):
;*       9+(27+10*22) = 256 cycles @ 2.4576 MHZ = 104 us = 9,600 bps
;*********************************************************************
GetByte:     brset  0,PTA,GetByte ;[.r...] loop till PTA0=0 (start)
             bsr    GetBit        ;[4+GetBit] 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 C
NextBit:                          ;    top of loop to get 8 bits
             bsr    GetBit        ;[4+GetBit] sense level of next bit
             rora                 ;[1] rotate into A from left
             nop                  ;[1] pad to tune timing
             bcc    NextBit       ;[3] continue 'till 1 RORs into C
stpBit:      bsr    GetBit        ;[4+GetBit] sense level of stop bit
             rts                  ;[4]
;* GetByte DONE ******************

;*********************************************************************
;* NAME: PutByte
;* PURPOSE:
;*   Send one byte data through PTA0 serially.  This routine supports
;*   a baud rate 9600 bps at internal 4MHz and external 2.4576MHz bus
;*   frequencies. A clock is distinguished by the state of ECGST bit
;*   (bit 0) in OSCSTAT.  When ECGST=1, an external clock is selected.
;* ENTRY CONDITIONS:
;*   PTA0 configured as an input, PTA0 data bit = 0
;*   A contains data to be sent
;* EXIT CONDITIONS:
```

```
;*   A and X is restored to entry values
;*   PTA0 configured as an input (PTA0=high idle line)
;* SUBROUTINES CALLED:
;* VARIABLES READ:
;* VARIABLES MODIFIED:
;* STACK USED: 4 (including the call to this routine)
;* SIZE: 46 bytes
;* DESCRIPTION: EXECUTED OUT OF ROM
;*   After ~2 bit times delay, a character in A is sent via PTA0
;*   Once called, program will remain in PutByte until PTA0=high
;*   This routine does not service COP
;*****************************************************************
PutByte:     pshx                ;[2] save X
             psha                ;[2] temp save Tx data
             lda    #10          ;[2] start, 8 data, stop = 10 loops
             brclr  0,PTA,*      ;[.r...] wait for PTA0 high (idle)
             ldx    #BitX2       ;[2] load constant for Ext

;* delay ~2 bit times before transmitting data (time not critical)
;* Ext 2 bit is 25% longer and Int 2 bit is 23% shorter

delay:       dbnzx  delay        ;[3] loop 3 cyc * BitX2I

             sec                 ;[1] becomes stop bit after 9 RORs
             bra    outLow       ;[3] Tx a low for start bit

PutLoop:     ror    1,SP         ;[5] LSB to C-bit, Tx that level
             bcc    outLow       ;[3] if C=0 Tx low, else Tx high
outHi:       bclr   0,DDRA       ;[4] PTA0 input pulls up to high
             bra    outDelay     ;[3] go to time 1 bit delay
outLow:      bset   0,DDRA       ;[4] PTA0 output makes pin drive low
             bra    outDelay     ;[3] time 1 bit delay (match time)
outDelay:    ldx    OSCSTAT      ;[3] check for Int/Ext clk
             bne    BitE         ;[3] branch if Ext (OSCSTAT!=$00)
             ldx    #BitTimeI    ;[2] load Int bit delay
             bra    delOut       ;[3] skip to delOut
BitE:        nop                 ;[1] timing adjust
             ldx    #BitTimeE    ;[2] load Ext bit delay
delOut:      dbnzx  delOut       ;[3] loop 3~ * (value in X)
             nop                 ;[1] timing adjust
             dbnza  PutLoop      ;[3] repeat for start, 8 data, stop

             pula                ;[2] restore Tx data
             pulx                ;[2] restore X
             rts                 ;[4]
;* PutByte DONE ******************

;*****************************************************************
;* NAME: GetBit
;* PURPOSE:
;*   Receive one serial bit via PTA0 and return it in C-bit
;* ENTRY CONDITIONS:
;*   PTA0 configured as an input.
;* EXIT CONDITIONS:
;*   A and X is restored to entry values
;*   Bit level is returned to C bit in CCR
;*   PTA0 configured as an input.
;* SUBROUTINES CALLED: GetBit
;* VARIABLES READ:
```

```
;* VARIABLES MODIFIED:
;* STACK USED: 4 (including the call to this routine)
;* SIZE: 31 bytes
;* DESCRIPTION: EXECUTED OUT OF ROM
;*   Execution cycle for Internal and external is:
;*    Internal  (OSCSTAT[0]=0) = 28 + (10 x SampPerBitI)
;*    Extnernal (OSCSTAT[0]=1) = 27 + (10 x SampPerBitE)
;*****************************************************************
GetBit:      pshx                ;[2] preserve X
             psha                ;[2] preserve A
             nop                 ;[1] time padding
             brset  0,OSCSTAT,SerialE ;[5] check if int or ext clk
             lda    #OffsetI     ;[2] # of samples to detect 1 (Int)
             ldx    #SampPerBitI ;[2] # of samples per bit (Int)
             brclr  0,OSCSTAT,loopIn ;[5] time matching padding
SerialE:     lda    #OffsetE     ;[2] # of samples to detect 1 (Ext)
             ldx    #SampPerBitE ;[2] # of samples per bit (Ext)
             nop                 ;[1] time padding
             bra    loopIn       ;[3] time padding
loopIn:      brclr  0,PTA,subSamp ;[5] set/clr C based on PTA0 level
subSamp:     sbc    #0           ;[2] subtract C from offset in A
             dbnzx  loopIn       ;[3] loop SampPerBitI times
             rola                ;[1] copy MSB to C bit (1 if A neg)
;* A would be negative if # of 1 samples was > OffsetG_
;* C bit reflects detected sense of current serial bit
             pula                ;[2] restore A
             pulx                ;[2] restore X
             rts                 ;[4] return
;* GetBit DONE ******************

;*****************************************************************
;* 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 contains the data 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
;* 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
;*   The COP is serviced in this routine. The first COP is serviced on
```

```
;*   23 bus cycles after this routine is called in the user software.
;*   However, the COP timeout might still occur 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
;* RDVRRNG DONE ******************

;*********************************************************************
;* NAME: PRGRNGE
;* PURPOSE:
;*   Program a FLASH address range which is maximum 32 bytes in the
;*   same row. Bus frequency must be between 1.0MHz 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 contains the data to be programmed to the FLASH
;*     (maximum 32 bytes)
;*   CPUSPD contains a nearest integer of 4 x bus frequency (MHz)
;* EXIT CONDITIONS:
;*   H:X contains a next FLASH address; I-bit set
;* SUBROUTINES CALLED:
;* VARIABLES READ: CPUSPD, LADDR:LADDR+1, DATA array
;* VARIABLES MODIFIED: LADDR(ByteCntr):LADDR+1(RamPntrLo)
;*   The values are modified, but they are restored with original
;*   values before exiting from this routine.
;* STACK SIZE: 9 bytes (including the call to this routine)
;* SIZE: 132 bytes
;* DESCRIPTION: EXECUTED OUT OF ROM
;*   This routine can program the FLASH only in the same row.
;*   Therefore, the total programing byte No. is maximum 32 bytes.
;*   The COP is serviced in this routine. The first COP is serviced on
;*   59 bus cycles after this routine is called in the user software.
;*   However, there could still be a COP time out if the COP is not
;*   serviced within a proper period in user software.
;*********************************************************************
PRGRNGE:
            sei                  ;set I bit to mask interrupts
            lda    LADDR
            psha                 ;save LADDR(hi) to stack [A]
            lda    LADDR+1
            psha                 ;save LADDR(lo) to stack [B]
            pshx                 ;calculate total # of bytes
                                 ; to be programmed
            pula                 ;SLADDR (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                 ;[C] total remaining bytes to prog
                                 ; will decrement by LoopCOP on each
                                 ; pass through LoopPROG
StartProg:
```

```
            clr    RamPntrLo     ;start with 1st loc. in DATA array

;* Current stack frame
;*          SP+2 [C] total bytes left to program; count down to zero
;*          SP+3 [B] LADDR(lo) used to restore last addr before RTS
;*          SP+4 [A] LADDR(hi)

;*****************************************************************
;* COP is serviced before each block of LoopCOP bytes are programmed
;* LoopPROG is the top of the outer loop.  BSR PRGstep1 programs up to
;* LoopCOP bytes before return (last batch may be fewer than LoopCOP)

LoopPROG:   lda   1,sp           ;[C] total bytes remaining to prog
            beq   ProgEnd        ;if zero, programing is done
            cmp   #LoopCOP       ;bytes remaining >= LoopCOP ?
            bge   InitPROG       ;if so, skip to InitPROG
            sta   ByteCntr       ;< so make ByteCntr = BytesRemaining
            clr   1,sp           ;and clear BytesRemaining at [C]
            bra   Program        ;Go program last partial block

InitPROG:   sub   #LoopCOP       ;>= so subtract LoopCOP
            sta   1,sp           ;bytes remaining reduced by LoopCOP
            lda   #LoopCOP       ;prepare to prog LoopCOP bytes
            sta   ByteCntr       ;ByteCntr = LoopCOP

Program:    bsr   PRGstep1       ;program up to LoopCOP bytes
            bra   LoopPROG       ;repeat outer loop...check number of
                                 ;bytes remaining
ProgEnd:
            pula                 ;deallocate [C]
            pula
            sta   LADDR+1        ;restore an original value to LADDR+1
            pula
            sta   LADDR          ;restore an original value to LADDR
            rts

;*****************************************************************
;* FLASH Programming Algorithm
;*****************************************************************

PRGstep1:   sta    COPCTL        ;[4] service COP
                                 ;before programming ByteCntr bytes
            lda    #mPGM         ;[2]
            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 Flash address [H:X]
                                 ;    w/ any data  (Prog Algo Step 3)
PRGstep4:   lda    CPUSPD        ;[3] delay for time Tnvs
            dbnza  *             ;[3*CPUSPD]       (Prog Algo Step 4)

PRGstep5:   lda    #(mPGM+mHVEN) ;sets HVEN and leaves PGM set
            sta    FLCR          ;[..w.] set HVEN  (Prog Algo Step 5)

PRGstep6:   lda    CPUSPD        ;[3] delay for time Tpgs
            dbnza  *             ;[3*CPUSPD]       (Prog Algo Step 6)
```

```
;*********************************************************************
;* Step 7 and Step 8 are repeated until a value in location LADDR+1
;* reaches to zero.
;*********************************************************************
PRGstep7:    pshx                ;[2] temp flash pointer (lo) [F]
             pshh                ;[2] temp flash pointer (hi) [G]

;* Current stack frame
;*           SP+1 [G] flash pointer (hi) temp store so H:X available
;*           SP+2 [F] flash pointer (lo) temp store so H:X available
;*           SP+3 [E] PCH (return addr hi)
;*           SP+4 [D] PCL (return addr lo)
;*           SP+5 [C] bytes remaining to prog..not counting this block
;*           SP+6 [B] LADDR+1
;*           SP+7 [A] LADDR

             clrh                ;[1] clear upper half of H:X
             ldx    RamPntrLo    ;[3] get DATA array pointer (lo)
             lda    DATA,x       ;[3] read data from a DATA array
             pulh                ;[2] restore flash pointer (hi) [G]
             pulx                ;[2] restore flash pointer (lo) [F]
             sta    ,x           ;[.w] write data to Flash addr
                                 ;                 (Prog Algo Step 7)
;*********************************************************************
;* Compute Tprog based on bus speed
;*  For slowest bus speeds (CPUSPD=4), Tprog = 38 bus cycles.  For
;*  other speeds, Tprog = 8 * CPUSPD + 5 bus cycles.

PRGstep8:                        ;delay for Tprog (Prog Algo Step 8)
             lda    CPUSPD       ;[3]
             cmp    #4           ;[2] if CPUSPD=4 (bus = 1MHz),
             beq    PRGstep9     ;[3]  Tprog=38 cycles
             asla                ;[1] for other cases
             sub    #9           ;[2]  A = 2 x CPUSPD - 9

DelayPRG:    nop                 ;[1]  1~ delay
             dbnza  DelayPRG     ;[3]  Tprog = 8 * CPUSPD + 5 cycles

PRGstep9:                        ;                (Prog Algo Step 9)
             aix    #1           ;[2] point to next FLASH address
             inc    RamPntrLo    ;[4] increment DATA array pointer
             dec    ByteCntr     ;[4] decrement byte counter
             bne    PRGstep7     ;[3] loop until byte counter is = 0

             rol    1,sp         ;[5] ROL/ROR/SEI makes 12~ delay
             ror    1,sp         ;[5] to match delay to PRGstep10
             sei                 ;[2]

PRGstep10:   lda    #mHVEN       ;[2] clear PGM, leave HVEN=1
             sta     FLCR         ;[..w.]        (Prog Algo Step 10)

PRGstep11:   lda    CPUSPD       ;[3] delay for time Tnvh
             dbnza  *            ;[3*CPUSPD]      (Prog Algo Step 11)

PRGstep12:   clra                ;[1] pattern to clear HVEN
             sta    FLCR         ;[..w.] clear HVEN bit in FLCR
```

```
                                ;clr HVEN (Prog Algo Step 12)
            sta    COPCTL       ;[4] service COP
            rts                 ;[4]
;* PRGRNGE DONE ******************

;*********************************************************************
;* 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 the call to this routine)
;* SIZE: 10 bytes
;* DESCRIPTION: EXECUTED OUT OF ROM
;*   This routine is called from ERARNGE routines.
;*   For example when bus frequency = 4MHz, A=16, and X=17, the
;*   delay time is:
;*      delay time = (3 x 16 x 17) + 5 = 821 cycles (205.25us)
;*      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]
;* DELNUS DONE ******************

;*********************************************************************
;* NAME: ERARNGE
;* PURPOSE:
;*  Erase a page or a whole array in FLASH memory. A bus frequency
;*  range has to be between 1.0MHz and 8.4MHz.
;* ENTRY CONDITIONS:
;*   H:X contains an FLASH address within a page or an array to be
;*    erased
;*   CTRLBYT selects MASS erase ($40) or PAGE erase ($00)
;*    If other value is written to CTRLBYT, the erase operation
;*    will not be performed
;*   CPUSPD contains a nearest integer of 4 x bus frequency
;* EXIT CONDITIONS:
;*   The contents of H:X (address passed) is preserved; I-bit set
;* SUBROUTINES CALLED: DELNUS
;* VARIABLES READ: CTRLBYT, CPUSPD
;* VARIABLES MODIFIED:
;* STACK USED: 7 (including the call to this routine)
;* SIZE: 76 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 successful erase).  The COP is
;*   serviced in this routine. The first COP is serviced on
;*   (40+3xCPUSPD) bus cycles after this routine is called in the user
;*   software. However, there could still be COP time out if the COP
```

```
;*   is not served within a proper period in the user software.
;*********************************************************************
ERARNGE:
             lda    CTRLBYT       ;if CTRLBYT is not either $40 or
             and    #$BF          ; $00, the operation is skipped
             bne    Finish
             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:    lda    CPUSPD        ;[3] delay Tnvs  (Erase Algo Step 4)
             dbnza  *             ;[3+(3*A)]

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    #LoopErase    ;[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 200us delay
             lda    CPUSPD        ;[3]
             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 + 26)) + 15    33,180~ @8MHz (Terase=4.148mS)

             pula                 ;[2] deallocate Loop Counter
                                  ;                (Erase Algo Step 7)
             sta    COPCTL        ;[4] service COP
ERAstep7:
             lda    FLCR          ;[4]
             and    #{$FF-(mERASE+mMASS)}
                                  ;[2] clear ERASE and MASS bits
             sta    FLCR          ;[..w.]          (Erase Algo Step 8)
                                  ;[2]
ERAstep8:    ldx    #TNVHL        ;delay for time Tnvhl
             lda    CPUSPD        ;[3] Tnvhl is used for both
             bsr    DELNUS        ; page and mass erase
                                  ;[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)
            nsa                 ;[3] 3~ delay
Finish:
            rts                 ;[4] return from ERARNGE
;* ERARNGE DONE ******************
```

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

AN2635
Rev. 4, 10/2005

freescale™
semiconductor