Semiconductor Products Sector
Application Note

AN2156

Programming and Erasing FLASH and EEPROM Memories on the MC68HC908AS60A/AZ60A

By Adeela Gill and Kazue Kikuchi
Transportation and Standard Products Group
Austin, Texas

Introduction

Freescale has released two new microcontrollers (MCU), the MC68HC908AS60A (AS60A) and the MC68HC908AZ60A (AZ60A), as new products in the M68HC08 Family of low-cost, high-performance devices.

The AS60A and AZ60A offer many features, including:

- 8.4-MHz internal bus frequency
- 60 Kbytes of FLASH memory
- FLASH data security
- One Kbyte of EEPROM with security option
- Two Kbytes of on-chip RAM
- On-chip charge pump

This application note explains how to use the FLASH and EEPROM on the MC68HC908AS60A/AZ60A and provides example software for program and erase operations. These algorithms are written in M68HC08 assembly code.

This code is available for download from http://www.freescale.com/mcu which is Freescale's Semiconductor Product Sector's microcontroller Web site.

The topics covered in this application note include:

- Memory Overview
- MC68HC908AS60A/AZ60A vs. MC68HC908AS60/AZ60
- FLASH Functional Description
- FLASH Block Protection
- FLASH Control and Block Protect Registers
- FLASH Erase Operations
- FLASH Program Operation
- EEPROM Functional Description
- Configuration Register 2 and EEPROM Timebase Divider Control Registers
- EEPROM Timebase Divider Initialization
- EEPROM Block Protection and Security
- Standard EEPROM Erase Operation
- AUTO Mode EEPROM Erase Operation
- Standard EEPROM Program Operation
- AUTO Mode EEPROM Program Operation
- Selective Bit Programming
- Practical Considerations for Programming/Erasing
- Evaluating Delay Times for the Sample Code
- FLASH Frequently Asked Questions
- EEPROM Frequently Asked Questions
- FLASH Assembly Source Code
- EEPROM AUTO Mode Assembly Source Code

Memory Overview

The FLASH cell utilized on the MC68HC908AS60A/AZ60A is an industry proven split gate cell available from Silicon Storage Technology (SST) in a 0.5-micron geometry. More information on the FLASH cell is available at http://www.ssti.com which is SST's Web site.

The cell uses channel hot electron injection for programming and Fowler-Nordheim tunneling for erasing. All programming voltages are generated internally by a charge pump from a single connection to V_{DD}.

With the quick bit programming time and the organization of the FLASH array into 64-byte rows, the entire 60-Kbyte memory can be programmed in less than two seconds. The cell has field-proven reliability and endurance. Freescale already has four devices with this technology that have been MC qualified per AEC-Q1000 Stress Test.

On the MC68HC908AS60A/AZ60A, the 60-Kbyte memory is separated into two arrays, each having its own charge pump. Decode logic in the programming circuitry allows only one array to be programmed at a time.

The EEPROM is a simple extension of the FLASH technology.Freescale has added a logic state machine around the EEPROM to make the programming and erasing code of older (0.65-µ technology) M68HC08 Family devices compatible with these new devices (0.5-µ technology).

However, a new, faster technique is available for programming/erasing the EEPROM. By using the AUTO bit in the EEPROM control register, fixed delays are eliminated. Instead, the software polls for a program/erase complete flag. A complete description of this technique is included in **AUTO Mode EEPROM Erase Operation** and **AUTO Mode EEPROM Program Operation**.

MC68HC908AS60A/AZ60A vs. MC68HC908AS60/AZ60

The MC68HC908AS60A/AZ60A devices discussed in this application note are a technology shrink from the previous MC68HC908AS60/AZ60 devices.

In addition to the transistor size reduction, a few changes were made on the devices. These include:

- Using a new FLASH technology
- Adding a special program/erase protection option on the EEPROM.
- Relocating several registers on the new devices

Table 1 lists the registers that were moved on the new parts. Also, the divider EEPROM registers discussed in **Configuration Register 2 and EEPROM Timebase Divider Control Registers** are all new registers specific to the new parts.

Table 1. Register Address Differences

	MC68HC908AS60/ AZ60	MC68HC908AS60A/ AZ60A
FLASH-1 control register (FL1CR)	$FE0B	$FF88
FLASH-2 control register (FL2CR)	$FE11	$FE08
EEPROM-2 control register (EE2CR)	$FE19	$FF7D
EEPROM-2 array con gur ation register (EE2ACR)	$FE1B	$FF7F
EEPROM-2 nonvolatile register (EE2NVR)	$FE18	$FF7C

Additional differences between the devices are listed in detail in **Table 2** and **Table 3**.

Table 2. FLASH Differences

	MC68HC908AS60/AZ60	MC68HC908AS60A/AZ60A
Technology	UDR (2TS)	UDR (SST)
Bus clock frequency requirement	The bus clock must be such that, when divided by a prescaler of 1, 2, or 4, yields a frequency of 1.8 MHz–2.5 MHz	Minimum 1 MHz Maximum 8.4 MHz
Algorithm	Multiple pulses using margin read	One x ed pulse
Bit-erased state	“0”	“1”
Programming minimum size	8 bytes	64 bytes
64 bytes programming time	Typical 80 ms	Minimum 1.94 ms
60 Kbytes programming time	Typical 77.5 s	Minimum 1.85 s
Erase size and time	Minimum 100 ms for all sizes: 64 bytes, 512 bytes, 16 Kbytes, 32 Kbytes	128 bytes: minimum 1 ms 32 Kbytes: minimum 4 ms
Erase/program the block protect register (Note: Does not refer to erasing/programming the array)	High voltage is required on IRQ pin	High voltage is NOT required on IRQ pin
Block protect size	16 K, 24 K, 28 K, 32 K	128-byte increments except that locations $7F00– $7FFF and $FF00–$FFFF (255 bytes each) are protected as one block
Row erase endurance	Minimum 100 cycles	Minimum 10,000 cycles
Row program endurance	Minimum 100 cycles	Minimum 10,000 cycles
Data retention	Minimum 10 years	Minimum 10 years

Table 3. EEPROM Differences

	MC68HC908AS60/AZ60	MC68HC908AS60A/AZ60A
EExDIVHNVR and EExDIVLNVR registers	N/A	The value programmed in these registers is automatically loaded into the EExDIVH/L registers on reset.
EExDIVH and EExDIVL registers	N/A	These registers can either be loaded from the EExDIVH/LNVR registers or must be written to directly to set up a constant 35-µs timebase.
EEDIVCLK bit in CONFIG-2	N/A	Timebase reference clock selection (bus clock or CGMXCLK)
EEBCLK bit in EEPROM control register	EEPROM bus clock selection (bus clock or internal RC oscillator)	N/A
Timebase reference clock frequency	N/A	250 kHz–16 MHz
Algorithm	Fixed delays	— The MC68HC908AS60/AZ60 EEPROM algorithm can be used. However, the EEDIV initialization is required to set proper prescaler value. — Program/erase cycle is terminated by the internal time in auto mode.
Bit-erased state	“1”	“1”
Programming time per byte	Minimum 10 ms	Minimum 10 ms for standard mode
Erasing time per byte	Minimum 10 ms	Minimum 10 ms for standard mode
Erasing time per block	Minimum 10 ms	Minimum 10 ms for standard mode
Erasing time for bulk	Minimum 10 ms	Minimum 10 ms for standard mode
Successive programming	Erase operation is not necessary before programming.	Erase operation is required before programming. The same byte may be successively programmed only if selective bit programming is used.
Recovery time for lower power consumption	Recovery time is required to stabilize after clearing EEOFF bit or recovering from stop mode.	Recovery time is not required.
Write/erase cycles	Minimum 10,000 cycles	Minimum 10,000 cycles
Data retention	Minimum 10 years	Minimum 10 years

FLASH Functional Description

The FLASH memory on the MC68HC908AS60A/AZ60A physically consists of two independent arrays each with two bytes of block protection. An erased bit reads as a logic 1 and a programmed bit reads as a logic 0. Program and erase operations are facilitated through control bits in memory-mapped registers. Details for these operations appear later in this application note.

Memory in the FLASH array is organized into 128 byte pages, and each page is subdivided into two rows of 64 bytes each. The minimum erase block size is a single page of 128 bytes. Programming is performed on a per-row basis, 64 bytes at a time. The address ranges for the user memory, control registers, block protect registers, and vectors are listed here.

The FLASH memory map on the MC68HC908AS60A/AZ60A consists of:

- $0450–$05FF, FLASH-2 array, 432 bytes (See Note)
- $0E00–$7FFF, FLASH-2 array, 29,184 bytes
- $8000–$FDFF, FLASH-1 array, 32,256 bytes
- $FE08, FLASH-2 control register, FL2CR
- $FF80, FLASH-1 block protect register, FL1BPR
- $FF81, FLASH-2 block protect register, FL2BPR
- $FF88, FLASH-1 control register, FL1CR
- $FFD2–$FFD3, $FFDA–$FFFF, vector space; subset of FLASH-1 area, 40 bytes (See Note)

NOTE: *The memory map of the MC68HC908AZ60A differs slightly from the MC68HC908AS60A. On the MC68HC908AZ60A:*

- *The FLASH-2 array is not continuous. It is divided into three sections: $0450–$04FF (176 bytes), $0580–$05FF (128 bytes), and $0E00–$7FFF (29,184 bytes)*
- *$FFCC–$FFD1 and $FFD4–$FFD9 are also defined vector addresses (total vector size is 52 bytes).*

The 64-byte row address boundaries for the MC68HC908AS60A/AZ60A are:

- $xx00–$xx3F
- $xx40–$xx7F
- $xx80–$xxBF
- $xxC0–$xxFF

When programming the FLASH, exact programming time must be used to program a row. Excessive program time can result in a program disturb condition, in which case an erased bit on the row being programmed becomes unintentionally programmed. Program disturb is avoided by using the recommended algorithm. See **FLASH Program Operation**.

NOTE: *A security feature prevents viewing of the FLASH contents.*[1]

Programming tools are available from Freescale. Contact a local Freescale representative for more information.

FLASH Block Protection

To protect the contents in the FLASH array from being inadvertently programmed or erased by run-away code in the user application, the FLASH block protect register option was implemented. This register is composed of two nonvolatile bytes within the FLASH-1 array, with one byte per FLASH array. Once the block protect bits are set in the FLxBPR registers, the defined address ranges are protected from being programmed or erased. See **FLASH Block Protection** for a description of address ranges.

However, If a protected memory block needs to be programmed or erased, the block protection can be overridden by applying the voltage V_{HI} on the IRQ pin during the erase and program operations.

NOTE: *The vector locations and FLxBPR registers are located in the same page. The FLxPBR registers are not protected with special hardware or software. Therefore, if the page is not protected by the FL1BPR and the vector locations are erased by either page or mass erase operation, both FL1BPR and FL2BPR registers are also erased.*

1. No security feature is absolutely secure. However, Motorola's strategy is to make reading or copying the FLASH difficult for unauthorized users.

FLASH Control and Block Protect Registers

Each FLASH array has two registers that control its operation, the FLASH control register (FLxCR) and the FLASH block protect register (FLxBPR). See **Figure 1** and **Figure 2**.

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	0	0	0	0	HVEN	MASS	ERASE	PGM
Write:								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 1. FLASH Control Register (FLxCR)

Two FLASH control registers, FL1CR and FL2CR, are for FLASH-1 and FLASH-2 arrays, respectively.

- $FF88 — FLASH-1 control register (FL1CR)
- $FE08 — FLASH-2 control register (FL2CR)

HVEN — High-Voltage Enable Bit

This read/write bit enables the charge pump to drive high voltages for program and erase operations in the array. HVEN can be set only if either PGM = 1 or ERASE = 1 and the proper sequence for program or erase is followed.

1 = High voltage enabled to array and charge pump on
0 = High voltage disabled to array and charge pump off

MASS — Mass Erase Control Bit

This read/write bit configures the memory for mass or page erase operations.

1 = Mass erase operation selected
0 = Page erase operation selected

ERASE — Erase Control Bit

This read/write bit configures the memory for erase operation. ERASE is interlocked with the PGM bit such that both bits cannot be set at the same time.

1 = Erase operation selected
0 = Erase operation unselected

PGM — Program Control Bit

This read/write bit configures the memory for program operation. PGM is interlocked with the ERASE bit such that both bits cannot be set at the same time.

1 = Program operation selected
0 = Program operation unselected

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	BPR7	BPR6	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0

Figure 2. FLASH Block Protect Register (FLxBPR)

Two FLASH block protect registers, FL1BPR and FL2BPR, are for FLASH-1 and FLASH-2 arrays, respectively.

- $FF80 — FLASH-1 block protect register (FL1BPR)
- $FF81 — FLASH-2 block protect register (FL2BPR)

BPR[7:0] — Block Protect Register Bit 7 to Bit 0

These eight bits specify the most significant bits of a 32-K FLASH memory location. Memory is protected against program and erase operations starting from this address to the end of the FLASH array.

The protected range is:

FLxBPR Value		FLASH-1	FLASH-2
$FF	=	No protection	No protection
$FE	=	$FF00–$FFFF	$7F00–$7FFF
$FD	=	$FE80–$FFFF	$7E80–$7FFF
		↓	↓
$0B	=	$8580–$FFFF	$0580–$7FFF
$0A	=	$8500–$FFFF	$0500–$7FFF
$09	=	$8480–$FFFF	$0480–$7FFF
$08	=	$8400–$FFFF	$0450–$7FFF
		↓	↓
$04	=	$8200–$FFFF	$0450–$7FFF
$03	=	$8180–$FFFF	$0450–$7FFF
$02	=	$8100–$FFFF	$0450–$7FFF
$01	=	$8080–$FFFF	$0450–$7FFF
$00	=	$8000–$FFFF	$0450–$7FFF

Decreasing the value in the FLxBPR by one increases the protected range by one page (128 bytes). However, programming the block protect register with $FE protects a range twice that size, 256 bytes, in the corresponding array. $FE means that locations $FF00–$FFFF or $7F00–$7FFF are protected in FLASH-1 or FLASH-2, respectively.

The FLASH memory does not exist at some locations. The block protection range is unaffected if FLASH memory does not exist in that range. Refer to the memory map and make sure that the desired locations are protected.

FLASH Erase Operations

On the MC68HC908AS60A/AZ60A, the FLASH arrays can be erased one page (128 bytes) at a time or an entire array can be erased with one routine (mass erase.) The pages are from addresses $xx00 to $xx7F and from $xx80 to $xxFF. **Figure 3** shows mass erase and page erase flowcharts.

FLASH Mass Erase Algorithm

NOTE: *Each FLASH array has separate FLASH control and block protect registers. Make sure to set the bits in the appropriate register.*

1. Set the ERASE bit and the MASS bit in the FLASH control register (FLxCR).

   ERASE = 1 configures the FLASH memory for an erase operation.
   MASS = 1 sets the MASS erase operation.

2. Read the FLASH block protect register (FLxBPR).

   The block protect register must be read before high voltage can be enabled. If the desired address set in step 3 is in a protected block, erase will fail.

3. Write to any FLASH address within the address range to be erased. For mass erase, this is any address in that FLASH array.

   The address is used to determine which array will be erased.

4. Wait for a time, t_{NVS}.

   Internal high voltage is charged.

5. Set the HVEN bit.

   Internal high voltage is applied to the array.

6. Wait for a time, t_{MERASE}.

   t_{MERASE} is the mass erase time.

7. Clear the ERASE bit.

   The erase operation is disabled.

8. Wait for a time, t_{NVHL}.

   This is the time required for internal high voltage to discharge from the array.

9. Clear the HVEN bit.

   Disable the internal high voltage.

10. Wait for a time, t_{RCV}.

    After a time, t_{RCV}, the memory can be accessed in normal read mode.

FLASH Page Erase Algorithm

NOTE: *Each FLASH array has separate FLASH control and block protect registers. Make sure to set the bits in the appropriate register.*

1. Set the ERASE bit and clear the MASS bit in the FLASH control register (FLxCR).

   ERASE = 1 configures the FLASH memory for an erase operation. MASS = 0 sets the PAGE erase operation.

2. Read the FLASH block protect register (FLxBPR).

   The block protect register must be read before high voltage can be enabled. If the desired address set in step 3 is in a protected block, erase will fail.

3. Write to any FLASH address within the address range to be erased. For page erase, this is any address within that 128-byte block.

   The address is used to determine the address range that will be erased.

4. Wait for a time, t_{NVS}.

   Internal high voltage is charged.

5. Set the HVEN bit.

   Internal high voltage is applied to the page.

6. Wait for a time, t_{ERASE}.

   t_{ERASE} is the page erase time.

7. Clear the ERASE bit.

   The erase operation is disabled.

8. Wait for a time, t_{NVH}.

   This is the time required for internal high voltage to discharge from the page.

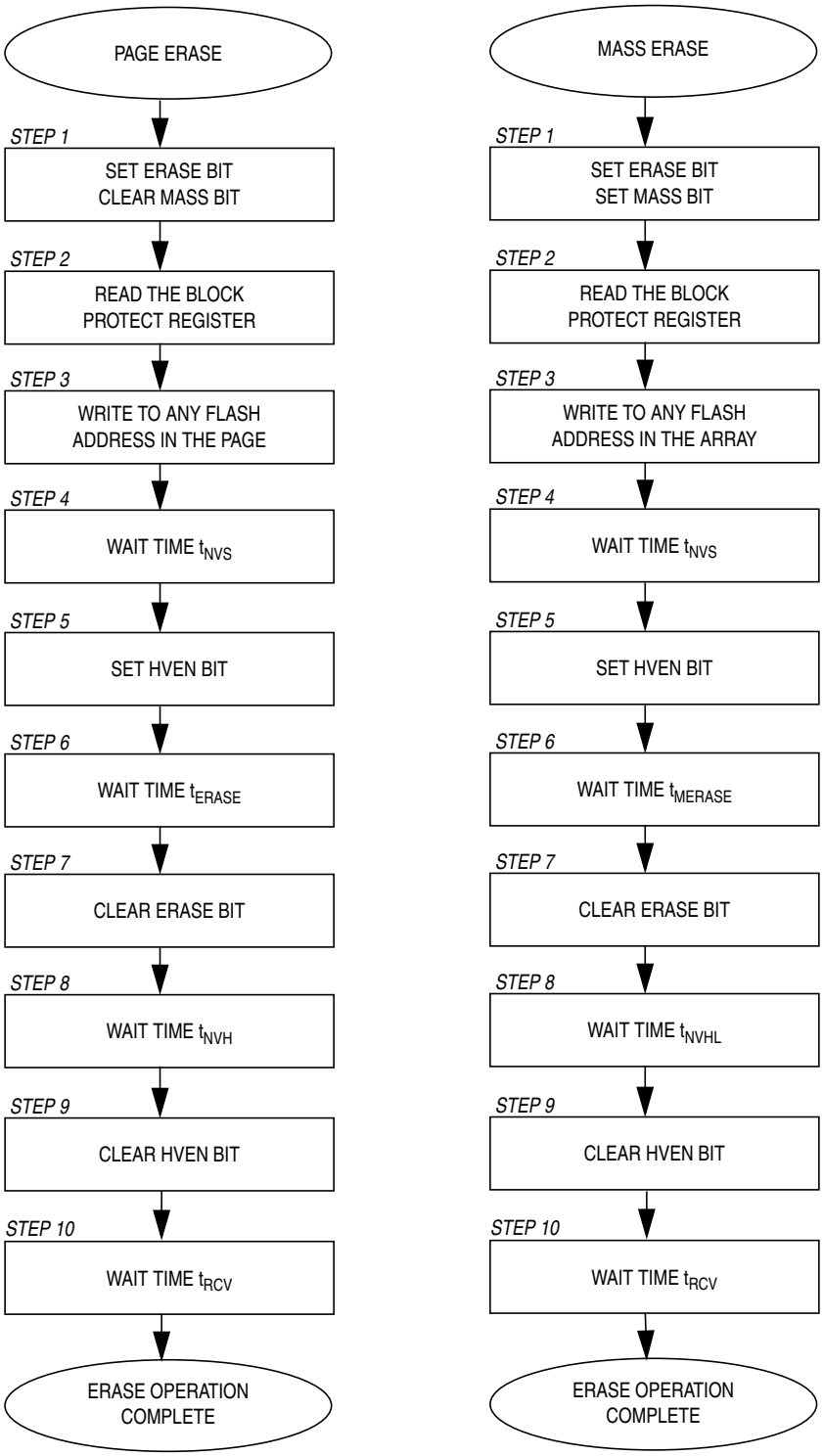
9. Clear the HVEN bit.

   Internal high voltage is disabled.

10. Wait for a time, t_{RCV}.

    After a time, t_{RCV}, the memory can be accessed in normal read mode.

NOTE: *Programming and erasing of FLASH locations cannot be performed by code being executed from the same FLASH array. However, code can be executed out of one FLASH array when programming/erasing locations in the other array.*



Note: To erase multiple pages, repeat steps 1–10.

Figure 3. FLASH Erase Operation Flowcharts

FLASH Program Operation

On the MC68HC908AS60A/AZ60A, programming of the FLASH memory is done on a row-by-row basis. A row consists of 64 bytes, with address ranges as follows:

- $xx00 to $xx3F
- $xx40 to $xx7F
- $xx80 to $xxBF
- $xxC0 to $xxFF

During a programming cycle, make sure that all addresses are being written to fit within one of the ranges specified. Attempts to program addresses in different row ranges in one programming cycle will fail. For example, programming from addresses $xx30 to $xx6F will not be successful because addresses $xx30–$xx3F and $xx40–$xx6F are in different rows.

Also take care that the initial address written makes logical sense. The programming algorithm includes a step (step 3 that follows) where the row to be programmed is identified by writing to any address in that row with any data. For most rows, this address can be chosen haphazardly since the row consists of 64 bytes. However, there are a few areas where the entire row does not consist of FLASH space. In these rows, when programming the array, ensure that a non-FLASH location is not used as the row specifier or written to during the sequence.

- First row of FLASH-2 ($0450–$047F) = 48 bytes
- Vector area on the MC68HC908AS60A ($FFD2–$FFD3 and $FFDA–$FFFF) = 40 bytes
- Vector area on the MC68HC908AZ60A ($FFCC–$FFFF) = 52 bytes

NOTE: *To avoid program disturbs, the row must be erased before any byte on that row is programmed.*

The programming flowchart is shown in **Figure 4**.

1. Set the PGM bit in the FLASH control register (FLxCR).

   PGM = 1 configures the FLASH memory for a program operation.

2. Read the FLASH block protect register.

   The block protect register must be read before high voltage can be enabled. If the desired address set in step 3 is in a protected block, programming will fail.

3. Write to any FLASH address within the row address range desired with any data.

4. Wait for a time, t_{NVS}.

   Internal high voltage is charged.

5. Set the HVEN bit.

   Internal high voltage is applied to programming row.

6. Wait for a time, t_{PGS}.

7. Write one data byte to a FLASH address to be programmed.

8. Wait for a time, t_{PROG}.

   t_{PROG} is the 1-byte programming time. t_{PROG} actually includes the total time from step 7 (A on the flowchart) back to step 7 (B on the flowchart) for additional byte programming, or from step 7 (A) to step 10 (C on the flowchart) for the last byte. This total time must be between 30 and 40 µs in both cases. Refer to **Figure 4**.

9. Repeat steps 7 and 8 until all the bytes within the row are programmed.

10. Clear the PGM bit.

    Disable the programming operation.

11. Wait for a time, t_{NVH}.

    Internal high voltage is discharged from the row.

12. Clear the HVEN bit.

    Internal high voltage is disabled.

13. Wait for a time, t_{RCV}.

    After a time, t_{RCV}, the memory can be accessed in normal read mode.

NOTE: *Programming and erasing of FLASH locations cannot be performed by code being executed from the same FLASH array. However, code can be executed out of one FLASH array to program/erase locations in the other array.*

While these operations must be performed in the order shown, other unrelated operations may occur between the steps. Do not exceed t_{PROG} maximum or t_{HV} maximum. t_{PROG} is defined in step 8. t_{HV} is defined as the cumulative time that high voltage is applied to the same row before an erase. $t_{NVS} + t_{NVH} + (t_{PROG} \times 64) \leq t_{HV}$.

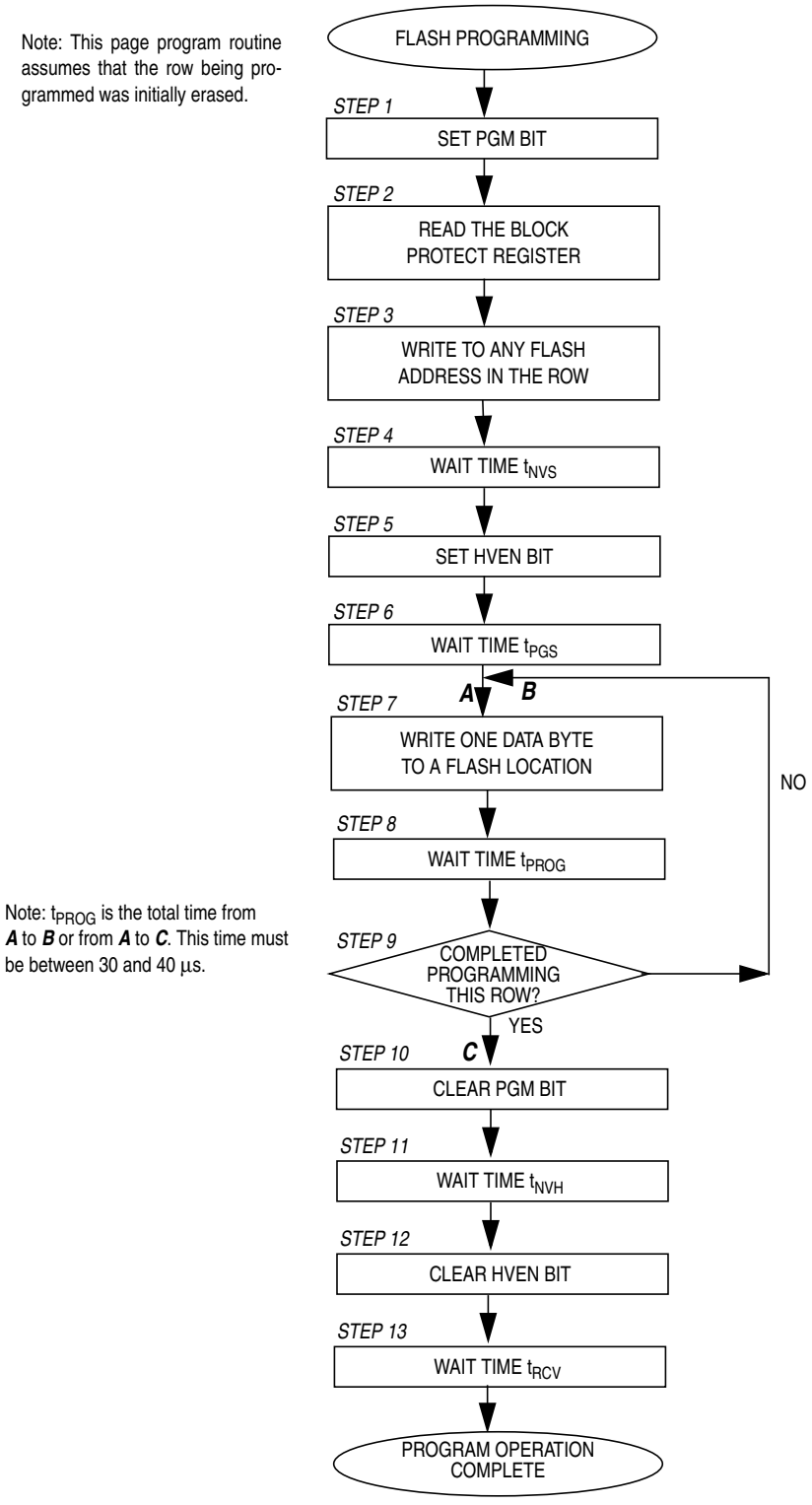


Figure 4. FLASH Programming Algorithm Flowchart

EEPROM Functional Description

The EEPROM memory on the MC68HC908AS60A/AZ60A physically consists of two independent 512-byte arrays each with two bytes of block protection and four bytes of timebase dividers. An erased bit reads as a logic 1 and a programmed bit reads as a logic 0. Program and erase operations are facilitated through control bits in memory mapped registers. Details for these operations appear later in this application note.

Memory in the EEPROM array can be programmed only one byte (8 bits) at a time. However, an entire array (512 bytes), a block (128 bytes), or a single byte can be erased at one time.

The EEPROM memory map on the MC68HC908AS60A/AZ60A, which includes registers related to EEPROM operation, consists of:

- $0600–$07FF, EEPROM-2 array, 512 bytes
- $0800–$09FF, EEPROM-1 array, 512 bytes
- $FE09, configuration write-once register, CONFIG-2
- $FE10, EEPROM-1 nonvolatile register, EE1DIVHNVR
- $FE11, EEPROM-1 nonvolatile register, EE1DIVLNVR
- $FE1A, EEPROM-1 divider high register, EE1DIVH
- $FE1B, EEPROM-1 divider low register, EE1DIVL
- $FE1C, EEPROM-1 nonvolatile register, EE1NVR
- $FE1D, EEPROM-1 control register, EE1CR
- $FE1F, EEPROM-1 array configuration register, EE1ACR
- $FF70, EEPROM-2 nonvolatile register, EE2DIVHNVR
- $FF71, EEPROM-2 nonvolatile register, EE2DIVLNVR
- $FF7A, EEPROM-2 divider high register, EE2DIVH
- $FF7B, EEPROM-2 divider low register, EE2DIVL
- $FF7C, EEPROM-2 nonvolatile register, EE2NVR
- $FF7D, EEPROM-2 control register, EE2CR
- $FF7F, EEPROM-2 array configuration register, EE2ACR

Configuration Register 2 and EEPROM Timebase Divider Control Registers

Five registers are related to the EEPROM timebase divider, which is discussed in **EEPROM Timebase Divider Initialization**.

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	EEDIV CLK	0	0	MSCAND	AT60A	0	0	AZxx
Write:								
Reset:	0	0	0	1	1	0	0	0

= Unimplemented

Figure 5. Configuration Register (CONFIG-2)

There is one byte configuration write-once register, CONFIG-2.

- $FE09, CONFIG-2

The EEDIVCLK bit in the CONFIG-2 register must be configured for the EEPROM to be erased or programmed properly.

EEDIVCLK — EEPROM Timebase Divider Clock Select Bit

This bit selects the reference clock source for the EEPROM-1 and EEPROM-2 timebase divider modules.

1 = EExDIV clock input is driven by internal bus clock.
0 = EExDIV clock input is driven by CGMXCLK.

NOTE: *Once the register is written, further writes will not have an effect until a reset occurs and the AZxx bit will not change except by a power-on reset.*

The EEPROM requires a 35-µs timebase. The next four registers set up the EEPROM timebase divider to provide this constant clock. See **EEPROM Timebase Divider Initialization**.

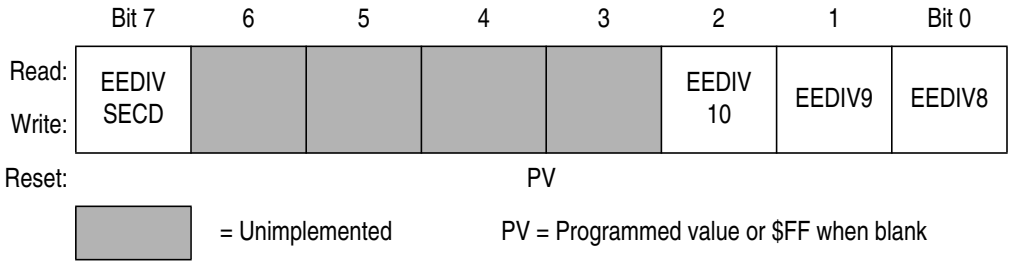


Figure 6. EExDIV Nonvolatile High Register (EExDIVHNVR)

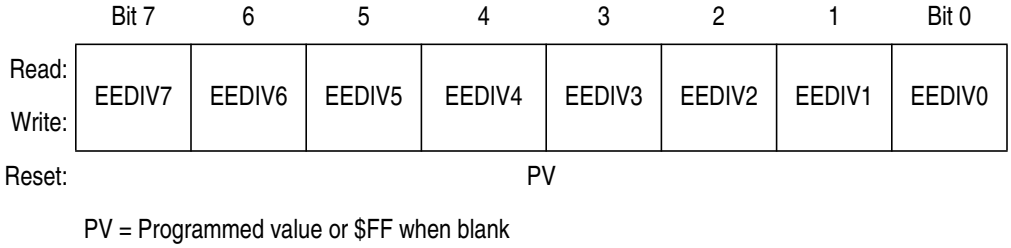


Figure 7. EExDIV Nonvolatile Low Register (EExDIVLNVR)

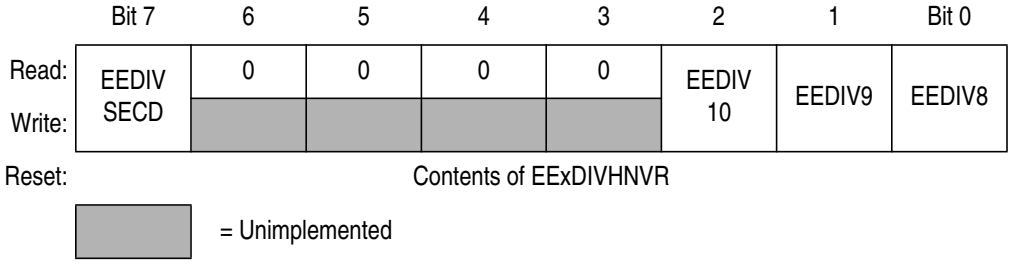


Figure 8. EExDIV Divider High Register (EExDIVH)

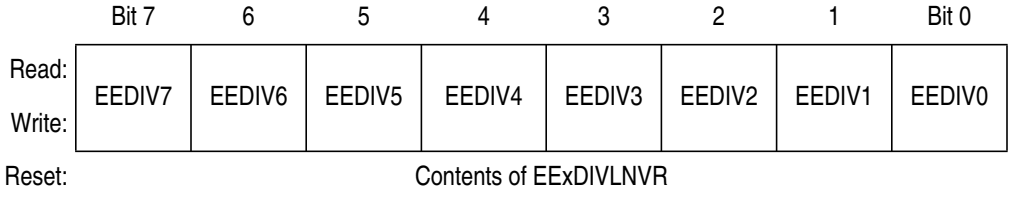


Figure 9. EExDIV Divider Low Register (EExDIVL)

There are two sets of registers to set up proper constant timebase for EEPROM-1 and EEPROM-2 arrays. They are:

- $FE10, EEPROM-1 nonvolatile register, EE1DIVHNVR
- $FE11, EEPROM-1 nonvolatile register, EE1DIVLNVR
- $FE1A, EEPROM-1 divider high register, EE1DIVH

- $FE1B, EEPROM-1 divider low register, EE1DIVL
- $FF70, EEPROM-2 nonvolatile register, EE2DIVHNVR
- $FF71, EEPROM-2 nonvolatile register, EE2DIVLNVR
- $FF7A, EEPROM-2 divider high register, EE2DIVH
- $FF7B, EEPROM-2 divider low register, EE2DIVL

NOTE: *The EExDIVHNVR and EExDIVLNVR registers consist of nonvolatile memory. Therefore, programming or erasing these locations requires the normal EEPROM program/erase sequences, which are described in later sections. On the other hand, the EExDIVH and EExDIVL do not consist of nonvolatile memory. Whenever reset occurs, values in the EExDIVHNVR and EExDIVLNVR are automatically loaded into the EExDIVH and EExDIVL, respectively. These registers are also writable in software.*

EEDIVSECD — EEPROM Divider Security Disable Bit

This bit enables/disables the security feature of the EExDIV registers. When the EExDIV security feature is enabled (EEDIVSECD is programmed to logic 0 in the EExDIVHNVR), the registers EExDIVH, EExDIVL, EExDIVHNVR, and EExDIVHNVR are locked permanently.

1 = EExDIV security feature disabled
0 = EExDIV security feature enabled

NOTE: *This security feature is enabled when EEDIVSECD in the EExDIVHNVR is programmed to 0 and then the system is reset.*

EEDIV10–EEDIV0 — EEPROM Timebase Prescaler Bits

These prescaler bits store the value of EExDIV which is used as the divisor to derive a timebase of 35 μs from the selected reference clock source (CGMXCLK or bus clock) for the EEPROM related internal timer and circuits. EEDIV10–EEDIV0 are readable at any time.

NOTE: *The EExDIVH and EExDIVL registers are writable anytime that EELAT is cleared and EEDIVSECD is set. However, modified values are held only until the next reset.*

The EExDIV value is calculated by using this formula:

$$EExDIV = INT\ [Reference\ Frequency\ (Hz) \times (35 \times 10^{-6}) + 0.5\ ]$$

EEPROM Control and Configuration Registers

Each EEPROM array has three registers that control its operation:

- EEPROM control register (EExCR)
- EEPROM nonvolatile register (EExNVR)
- EEPROM array configuration register (EExACR)

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	UN USED	0	EEOFF	EERAS1	EERAS0	EELAT	AUTO	EEPGM
Write:								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 10. EEPROM Control Register (EExCR)

There are two EEPROM control registers, EE1CR and EE2CR, for EEPROM-1 and EEPROM-2 arrays, respectively.

- $FE1D — EEPROM-1 control register (EE1CR)
- $FF7D — EEPROM-2 control register (EE2CR)

Bit 7 — Unused Bit

EEOFF — EEPROM Power-Down Bit

This read/write bit disables the EEPROM module for lower power consumption. Any attempts to access the array will give unpredictable results. Reset clears this bit.

1 = Disable EEPROM array
0 = Enable EEPROM array

EERAS1 and EERAS0 — Erase Bits

These read/write bits set the erase modes. Reset clears these bits.

Table 4. EEPROM Program/Erase Mode Select

EEBPx	EERAS1	EERAS0	MODE
0	0	0	Byte program
0	0	1	Byte erase
0	1	0	Block erase
0	1	1	Bulk erase
1	X	X	No erase/program

X = don't care

EELAT — EEPROM Latch Control Bit

This read/write bit latches the address and data buses for programming the EEPROM array. EELAT cannot be cleared if EEPGM is still set. Reset clears this bit.

1 = Buses configured for EEPROM programming/erasing
0 = Buses configured for normal read operation

AUTO — Indication for Automatic Termination of Program/Erase Cycle

When AUTO is set, EEPGM is cleared automatically after the program/erase cycle is terminated by the internal timer.

1 = Automatic clear of EEPGM is enabled.
0 = Automatic clear of EEPGM is disabled.

EEPGM — EEPROM Program/Erase Enable Bit

This read/write bit enables the internal charge pump. If the EELAT bit is set and a write to a valid EEPROM location has occurred, then the programming/erasing voltage is applied to the EEPROM array. Reset clears the EEPGM bit.

1 = EEPROM programming/erasing power switched on
0 = EEPROM programming/erasing power switched off

NOTE: *Writing logic 0s to both the EELAT and EEPGM bits with a single instruction will only clear the EEPGM bit to allow time for the high voltage to dissipate.*

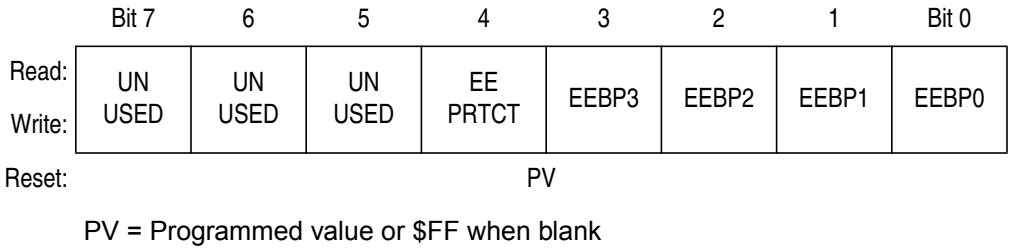


Figure 11. EEPROM Nonvolatile Register (EExNVR)

NOTE: *The EExNVR registers are factory programmed to $F0 so that the full array is available and unprotected.*

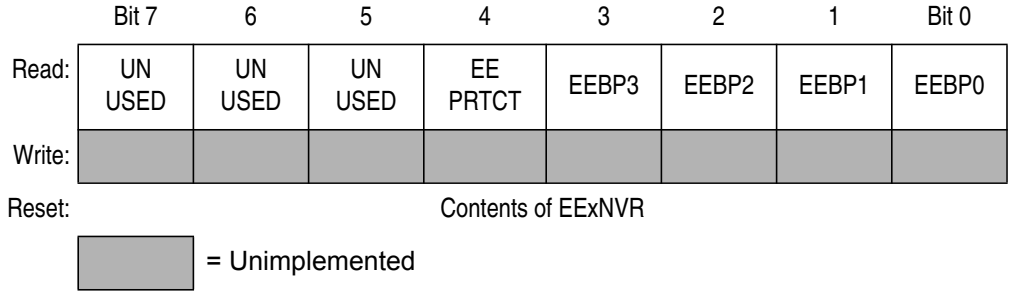


Figure 12. EEPROM Array Configuration Register (EExACR)

The registers in **Figure 11** and **Figure 12** determine a memory protection block. There are two sets of registers for both EEPROM-1 and EEPROM-2 arrays. They are:

- $FE1C, EEPROM-1 nonvolatile register, EE1NVR
- $FE1F, EEPROM-1 array configuration-volatile register, EE1ACR
- $FF7C, EEPROM-2 nonvolatile register, EE2NVR
- $FF7F, EEPROM-2 array configuration-volatile register, EE2ACR

NOTE: *The EExNVR registers are nonvolatile memory locations. Therefore, programming or erasing the registers requires the normal EEPROM program/erase sequences, which are described in later sections. On the other hand, the EExACR registers are not nonvolatile memory. On reset, the values in the EExNVRs are automatically loaded into the EExACRs. Thereafter, all reads to the EExNVRs will result in the EExACRs being reloaded.*

Bits 7:5 — Unused Bits

EEPRTCT — EEPROM Program/Erase Protection Bit

This one-time programmable bit can be used to protect 16 bytes (EEPROM-1: $8F0–$8FF and EEPROM-2: $6F0–$6FF) from being erased or programmed.

1 = EEPROM program/erase protection disabled
0 = EEPROM program/erase protection enabled

NOTE: *This feature is a write-once feature. Once the protection is enabled, it cannot be disabled.*

EEBP3–EEBP0 — EEPROM Block Protection Bits

These read/write bits select blocks of EEPROM array from being programmed or erased. Reset loads EEBP[3:0] from EExNVR to EExACR.

1 = EEPROM array block is protected.
0 = EEPROM array block is unprotected.

	EEPROM-1	**EEPROM-2**
EEBP0 = 1	$0800–$087F	$0600–$067F
EEBP1 = 1	$0880–$08FF	$0680–$06FF
EEBP2 =1	$0900–$097F	$0700–$077F
EEBP3 =1	$0980–$09FF	$0780–$07FF

NOTE: *The configuration in the EExNVRs can be changed by programming or erasing. The new configuration will take effect with a reset or a read of the EExNVR.*

EEPROM Timebase Divider Initialization

The EEPROM is actually a FLASH cell surrounded by a logic state machine. The state machine requires a constant timebase with a period of 35 μs to apply high voltage with the right timing during the erase and program operations. If the constant timebase is not set properly, the memory cells may be over-programmed and permanently damaged. Freescale cannot guarantee EEPROM program and erase operations in this case.

To generate this constant timebase, a clock divider value (EExDIV value) has to be set in the EEPROM timebase divider registers (EExDIVH and EExDIVL).

Follow these steps to initialize a timebase divider value:

1. The reference clock source has to be selected. The EEDIVCLK bit in the CONFIG-2 register determines the clock source. (Refer to **Configuration Register 2 and EEPROM Timebase Divider Control Registers**). The two options for clock source are:
   - CGMXCLK (output of the crystal oscillator circuit)
   - System bus clock

NOTE: *The recommended frequency range of the reference clock is 250 kHz to 16 MHz.*

2. The EExDIV has to be calculated to derive a EExDIV value that matches the user's application. Use the formula:

   EExDIV = INT [Reference Frequency(Hz) x (35×10^{-6}) +0.5]

   - Example 1: When the reference clock is 4.9152 MHz, the EExDIV value is 172 (hex $AC).
   - Example 2: When the reference clock is 8 MHz, the EExDIV value is 280 (hex $118).

NOTE: *Make sure that the EExDIV is calculated with the reference clock source being used.*

3. The calculated EExDIV value must be set in the EExDIVH and EExDIVL using one of two methods:
   – Program the EExDIV value to EExDIVHNVR and EExDIVLNVR and then reset the microcontroller. In this method, the programmed value is loaded to the EExDIVH and EExDIVL on reset.
   – Write the calculated EExDIV value to the EExDIVH and EExDIVL. In this method, the written value is held in the registers until the next reset occurs.

EEPROM Block Protection and Security

The MC68HC908AS60A/AZ60A has a special program/erase protection feature which prevents program/erase access to certain memory locations. This is called block protection and is controlled by the EEBPx bits in the EExNVR registers.

There is also a special security feature of this EEPROM which designates 16 bytes that can be permanently secured. These are in addresses $06F0–$06FF and $08F0–$08FF. Once the EEPRTCT bit is cleared for the first time, the values in these bytes are stored permanently. They can be read, but not programmed or erased.

When the EEPRTCT bit is programmed to 0:

- Programming and erasing of secured locations (EEPROM-2: $06F0–$06FF and EEPROM-1: $08F0–$08FF) are disabled permanently.
- The protected locations can be read as normal.
- Programming and erasing of the EExNVR is disabled permanently.
- For unprotected locations (EEPROM-2: $0600–$06EF, $0700–$07FF and EEPROM-1: $0800–$09EF, $0900–$09FF), bulk and block erase modes are disabled. Single byte erase and program modes are available. However, if the locations are partially or entirely protected by EEPBx bits, the protected block cannot be erased or programmed.

Table 5. EEPROM Protection and Security Features

Address Range Covered (EEPROM-1 / EEPROM-2)		EEPRTCT = 1	EEPRTCT = 0
$0800–$087F $0600–$067F	EEBP0 = 0	Byte programming available Bulk, block, and byte erasing available	Byte programming available Only byte erasing available
	EEBP0 = 1	Protected	Protected
$0880–$08EF $0680–$06EF	EEBP1 = 0	Byte programming available Bulk, block, and byte erasing available	Byte programming available Only byte erasing available
	EEBP1 = 1	Protected	Protected
$08F0–$08FF $06F0–$06FF	EEBP1 = 0	Byte programming available Bulk, block, and byte erasing available	Secure (no erasing or programming)
	EEBP1 = 1	Protected	
$0900–$097F $0700–$077F	EEBP2 = 0	Byte programming available Bulk, block, and byte erasing available	Byte programming available Only byte erasing available
	EEBP2 = 1	Protected	Protected
$0980–$09FF $0780–$07FF	EEBP3 = 0	Byte programming available Bulk, block, and byte erasing available	Byte programming available Only byte erasing available
	EEBP3 = 1	Protected	Protected

Standard EEPROM Erase Operation

The EEPROM can be erased on a byte, block (128 bytes), or bulk array (512 bytes) basis. If the block protection register shows the desired erase region to be protected, erasing will fail. Therefore, first ensure that the block protection feature is disabled for the relevant addresses.

This EEPROM erase operation is compatible with the MC68HC908AS60/AZ60 EEPROM erase operation.

The flowchart for the standard EEPROM erase operation is included as **Figure 13**.

1. Clear/set EERAS1 and EERAS0 to select byte/block/bulk erase, and set EELAT in EExCR.

2. Write any data to an EEPROM location. For byte erase, this should be the address to erase. For block or bulk erase, this is any address within the block or array to erase.
3. Set the EEPGM bit.
4. Wait for a time, $t_{EEBYTE}/t_{EEBLOCK}/t_{EEBULK}$, to erase a byte/block/whole array.
5. Clear the EEPGM bit.
6. Wait for a time, t_{EEFPV}, for the erasing voltage to fall.
7. Clear the EELAT bit.
8. Repeat steps 1 to 7 for more EEPROM byte/block/bulk erasing.

AUTO Mode EEPROM Erase Operation

The EEPROM can be erased on a byte, block (128 bytes), or bulk array (512 bytes) basis. If the block protection register shows the desired erase region to be protected, erasing will fail. Therefore, first ensure that the block protection feature is disabled for the relevant addresses.

On this EEPROM memory, another erase operation mode is provided. It is called AUTO mode. When AUTO mode is selected, the erase cycle is terminated by the internal timer, which polls the EEPGM bit in the EExCR register. Therefore, fixed delay times are not required in the AUTO mode.

The flowchart for the AUTO mode EEPROM erase operation is included as **Figure 13**.

1. Clear/set EERAS1 and EERAS0 to select byte/block/bulk erase, and set EELAT and AUTO in the EExCR.
2. Write any data to an EEPROM location. For byte erase, this should be the address to erase. For block or bulk erase, this is any address within the block or array to erase.
3. Set the EEPGM bit.
4. Poll the EEPGM bit until it is cleared by the internal timer.
5. Clear the EELAT bit.
6. Repeat steps 1 to 5 for more EEPROM byte/block/bulk erasing.

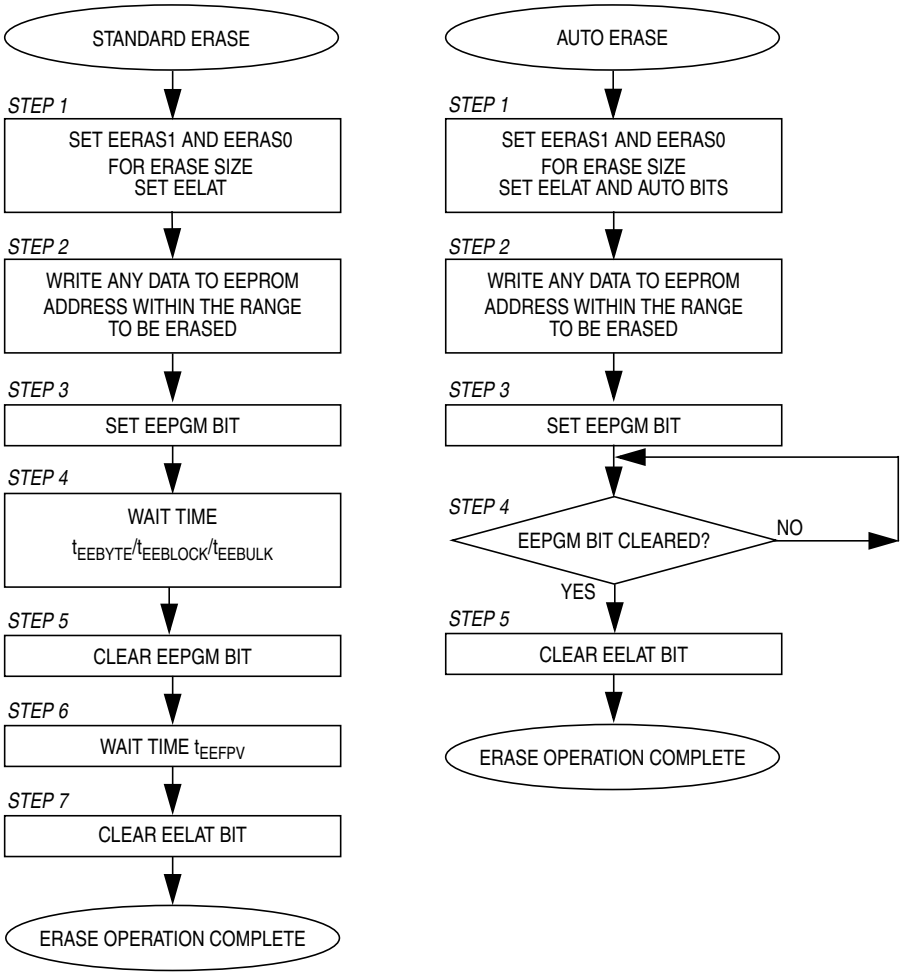


Figure 13. EEPROM Erasing Algorithm Flowcharts

Standard EEPROM Program Operation

Follow the next procedure to program one byte of EEPROM. Ensure that the block protection feature is disabled for the byte to be programmed.

This EEPROM program operation is compatible with the MC68HC908AS60/AZ60 EEPROM program operation.

The flowchart for this programming algorithm is shown in **Figure 14**.

NOTE: *The byte must be erased before programming.*

1. Clear EERAS1 and EERAS0 and set EELAT in the EExCR.
2. Write the desired data byte to the desired EEPROM address.
3. Set the EEPGM bit.
4. Wait for a time, t_{EEPGM}, to program the byte.
5. Clear the EEPGM bit.
6. Wait for a time, t_{EEFPV}, for the programming voltage to fall.
7. Clear the EELAT bit.
8. Repeat steps 1 to 7 to program additional bytes in the EEPROM arrays.

AUTO Mode EEPROM Program Operation

Follow the next procedure to program one byte of EEPROM using the AUTO feature. Ensure that the block protection feature is disabled for the byte to be programmed.

On this EEPROM memory, another program operation is provided. It is called AUTO mode. When AUTO mode is selected, the program cycle is terminated by the internal timer, which polls the EEPGM bit in the EExCR register. Therefore, fixed delays are not required with the AUTO mode operation.

The flowchart for this programming algorithm is shown in **Figure 14**.

NOTE: *The byte must be erased before programming.*

1. Clear EERAS1 and EERAS0 and set EELAT and AUTO in the EExCR.
2. Write the desired data byte to the desired EEPROM address.
3. Set the EEPGM bit.
4. Poll the EEPGM bit until it is cleared by the internal timer.
5. Clear EELAT bits.
6. Repeat steps 1 to 5 to program additional bytes in the EEPROM array.

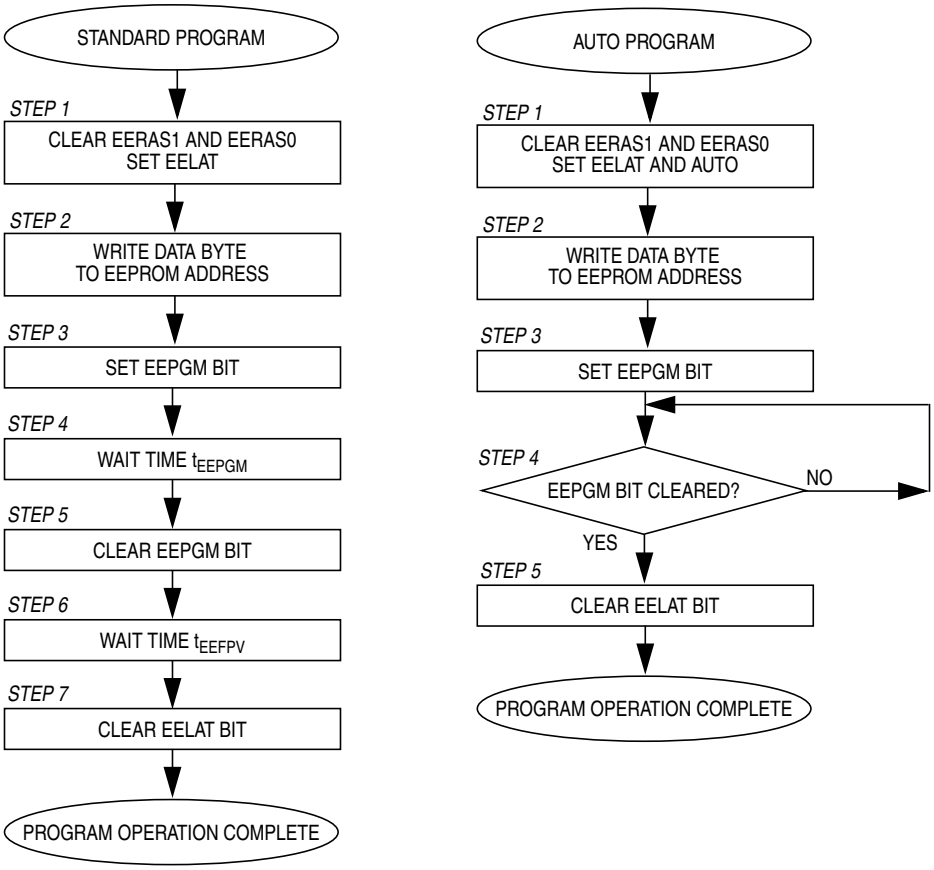


Figure 14. EEPROM Programming Algorithm Flowcharts

Selective Bit Programming

On each programming cycle, one to eight bits of the EEPROM memory may be programmed. It is possible to program multiple bits at the same time. However, the same bit may not be programmed twice unless the entire byte has been erased first. This means that the same byte location may be programmed eight times as long as an individual bit is not written to more than once. This is referred to as selective bit programming. The acceptable sequence in **Table 6** shows how the same byte may be used for eight program cycles without an erase.

Table 6. Selective Bit Programming

Acceptable Sequence			Unacceptable Sequence		
Operation	**Program Data**	**Value in Memory**	**Operation**	**Program Data**	**Value in Memory**
Erase	NA	1111:1111	Erase	NA	1111:1111
Write $FE	1111:1110	1111:1110	Write $FE	1111:111**0**	1111:1110
Write $FD	1111:1101	1111:1100	Write $F9	1111:1**00**1	1111:1000
Write $FB	1111:1011	1111:1000	Write $EF	1110:1111	1110:1000
Write $F7	1111:0111	1111:0000	Write $D8	1101:1**000**	Unknown
Write $EF	1110:1111	1110:0000			
Write $DF	1101:1111	1100:0000			
Write $BF	1011:1111	1000:0000			
Write $7F	0111:1111	0000:0000			
Erase	NA	1111:1111			

EEPROM memory lifetime is guaranteed for 10-K read/write cycles. However, using selective bit programming extends the life cycle of the memory 8-fold since each bit is only programmed 10-K times. This allows the user to program a single byte up to 80-K times.

If a bit is programmed more than once before the byte is erased, Freescale cannot guarantee proper operation of the EEPROM array.

Practical Considerations for Programming/Erasing

To ensure successful programming and erasing of the FLASH and/or EEPROM on the MC68HC908AS60A/AZ60A, the user should consider the following suggestions:

- Provide a good ground.
- Provide a clean and constant clock during the program and erase operations.
- Filter all signals leaving a noisy environment.
- If a microcontroller is programmed or erased in a socket, ensure that all pins are making good electrical contact.
- Provide an electrically noise-free environment for the MCU. The V_{DD} supply should be filtered and within the specification limits. Decoupling capacitors should be placed very close to the V_{DD}/V_{SS} pin pairs. Any high current switching activity on the PCB or in the general vicinity should be disabled during programming.

Evaluating Delay Times for the Sample Code

The FLASH and EEPROM algorithms include specific wait steps. These delay times are defined in the MC68HC908AS60A/AZ60A specification and must be considered when utilizing the algorithms.

To ensure that each wait step meets the specification, the delay times have to be evaluated. All delays in the sample code provided in this application note were evaluated and confirmed to meet the specification using a general I/O port pin. The port pin was initialized with a high output signal. Just before entering a delay, the port pin was toggled low and held low until the end of the delay. The actual delay times shown in **Table 7**, **Table 8**, and **Table 9** were the low periods as measured on an oscilloscope. To toggle the port pin, instructions BSET, BCLR, and COM were used. The sample code also includes the instructions used for the delay time evaluation.

Table 7. FLASH Erase Delay Times

Name of Delay	Specified Duration	Calculated Delay Time(1)	Actual Delay Time	Delta(2)
t_{NVS}	> 10 µs	10.6 µs	12.24 µs (S1 to E1)(3)	1.64 µs
t_{ERASE}	> 1 ms	1.01 ms	1.012 ms (S2 to E2)	2 µs
t_{NVH}	> 5 µs	5.7 µs	7.32 µs (S3 to E3)	1.62 µs
t_{MERASE}	> 4 ms	4.01 ms	4.02 ms (S4 to E4)	10 µs
t_{NVHL}	> 100 µs	101 µs	102.8 µs (S5 to E5)	1.6 µs
t_{RCV}	> 1 µs	2.0 µs	3.66 µs (S6 to E6)	1.66 µs

1. Delay time calculated by dividing the number of cycles in the delay by the bus speed of 2.4576 MHz.
2. Most delta times are around 1.6 µs. Since the instructions BSET and COM require four internal bus cycles, this additional time comes from the execution time of the instruction.
$$\frac{1}{2.4576\ \text{MHz}} \times 4\ \text{cycles} = 1.627\ \mu\text{s}$$
3. These times refer to measured delays based on running the attached sample code. In that code, the points at which measurements were taken are defined by these markers.

Table 8. FLASH Program Delay Times

Name of Delay	Specified Duration	Calculated Delay Time(1)	Actual Delay Time	Delta(2)
t_{NVS}	> 10 µs	10.6 µs	12.24 µs (S7 to E7)(3)	1.64 µs
t_{PGS}	> 5 µs	5.7 µs	7.34 µs (S8 to E8)	1.64 µs
t_{PROG}	30 µs–40 µs	30.5 µs(4)	32.2 µs (S9 to E9)	1.7 µs
		31.3 µs(5)	32.9 µs (S10 to E10_1)	1.6 µs
		31.3 µs(6)	32.8 µs (S10 to E10_2)	1.5 µs
t_{NVH}	> 5 µs	5.7 µs	7.32 µs (S11 to E11)	1.62 µs
t_{RCV}	> 1 µs	2.4 µs	4.0 µs (S12 to E12)	1.60 µs

1. Delay time calculated by dividing the number of cycles in the delay by the bus speed of 2.4576 MHz.
2. Most delta times are around 1.6 µs. Since the instructions BSET and COM have require four internal bus cycles, this additional time comes from the execution time of the instruction. $\frac{1}{2.4576\ \text{MHz}} \times 4\ \text{cycles} = 1.627\ \mu\text{s}$
3. These times refer to measured delays based on running the attached sample code. In that code, the points at which measurements were taken are defined by these markers.
4. The byte programmed is NOT the last byte of the row being programmed.
5. For the FLASH array-1, the byte programmed IS the last byte of the row being programmed.
6. For the FLASH array-2, the byte programmed IS the last byte of the row being programmed.

Table 9. EEPROM Standard Sequence Delay Times

Name of Delay	Specified Duration	Calculated Delay Time(1)	Actual Delay Time	Delta
t_{EEBYTE} $t_{EEBLOCK}$ t_{EEBULK} t_{EEPGM}	> 10 ms	10.01 ms	10.4 ms (S1 to E1)	0.39 ms
t_{EEFPV}	> 100 µs	101 µs	102.8 µs (S2 to E2)	1.8 µs

1. Delay time calculated by dividing the number of cycles in the delay by the bus speed of 2.4576 MHz.

FLASH Frequently Asked Questions

These questions and answers are designed to help the user with frequent concerns.

Question 1 I cannot program/erase FLASH memory at all. What should I consider to make my program/erase code work?

Answer 1 Check the following:

- *Is each step of the programming algorithm (or erasing algorithm) performed in the right order?*
  The sequence of the program and erase operations is interlocked in hardware so only the prescribed order of these operations will allow erase/program operations. However, other non-FLASH operations may occur between the steps shown.
- *Is the memory block where you want to program/erase unprotected?*
  The block protect feature of the FLASH is present to prevent unintentional programming or erasing. The block protect bits must be cleared such that the memory to be erased or programmed is unprotected. The only way to override the block protection is to apply voltage V_{HI} on IRQ during the erase and program algorithms. (Refer to **FLASH Erase Operations**.)
- *Are delay times such as t_{PROG}, t_{ERASE}, t_{MERASE}, t_{NVS}, etc., within the specification?*
  Timing is critical to ensure proper FLASH operation. Delay times that are too long or too short can alter the FLASH performance to the point where it does not work or is not reliable. Freescale does not guarantee FLASH performance if all timing requirements are not being adhered to.
- *Is the correct FLASH register being written to enable erase or program?*
  The MC68HC908AS60A/AZ60A has two FLASH arrays with two separate sets of control and block protect registers. Make sure the appropriate register is being addressed. Refer to **FLASH Block Protection**.

- *Is the COP enabled?*
  If the COP is enabled, address $FFFF has to be written periodically (with any value) to prevent a COP reset. However, this COP feeding process during the program/erase operation can cause unintentional FLASH programming or erasing because address $FFFF exists in the FLASH. To avoid issues, make sure that the selected COP period is long enough that the COP feeding process is not performed during the program/erase operation or disable the COP entirely during this operation.

NOTE: *The COP is always enabled out of RESET and can be disabled by modifying the "write-once" CONFIG-1 register. Refer to the* ***FLASH Program Operation****.*

- *Was Motorola's recommended programming algorithm (or erasing algorithm) used in your code?*
  The recommended programming algorithm ensures that the FLASH is programmed for sufficient data retention with a minimum program time. Not following this algorithm can lead to overprogramming, which risks program disturb.

Question 2 I wanted to erase only vector locations using the page erase operation, but the block protection registers were also erased. Did I do anything wrong?

Answer 2 No. The block protect registers and vector locations are in the same page. On the MC68HC908AS60A/AZ60A, programming and erasing the block protect registers is not protected. (On the MC68HC908AS60/AZ60, this operation required a high voltage on the IRQ pin). Therefore, the block protect registers are also erased when the vector locations are erased. If other FLASH locations must be protected, you have to reprogram the block protect register after the vector erasing is completed.

Question 3 What is the FLASH charge pump?

Answer 3 The charge pump is a dynamic (clocked) circuit which generates high voltages internally in the FLASH to program and erase the nonvolatile memory. Users do not have access to these voltages.

Question 4 The MC68HC908AS60A/AZ60A FLASH programs one row (64 bytes) at a time. Do I always have to program the entire row?

Answer 4 No, it is not necessary to program the entire row. Addresses which are not programmed are left as they were before the row programming was started. However, before reprogramming any additional bytes in this row, the entire page must be erased.

Question 5 During a program/erase process, can I execute an interrupt service or include additional steps?

Answer 5 Unrelated (non-FLASH) steps may be included between steps of the program/erase algorithms as long as the sequence of the steps remains consistent. However, interrupt service routines may cause errors in the program or erase timing and lead to corrupt or missing data in the FLASH. Freescale does not recommend the use of interrupts during the program or erase operations.

NOTE: *Make certain not to enter stop or wait mode during a program or erase operation. High voltage may be exposed to bit cells for an extended period and may cause permanent damage.*

Question 6 I am executing program/erase code out of one of the memory arrays. Can the same array be programmed/erased?

Answer 6 No.

Question 7 While running the program/erase code in one of the memory arrays, can the other memory array be programmed/erased?

Answer 7 Yes. The MC68HC908AS60A/AZ60A has two FLASH memory arrays. One array can be used for executing code while programming/erasing the other.

Question 8 Can I program/erase both FLASH arrays at the same time?

Answer 8 No. Each array does have a separate charge pump but the address decode logic does not allow more than one row to be programmed at a time.

Question 9 When writing 64 bytes of data to one row of FLASH memory for programming, does the order of written data matter?

Answer 9 No, as long as the bytes are written within a row, the data is latched for the programming operation.

Question 10 Does programming or erasing the FLASH block protect registers (FLxBPRs) require V_{HI} on the IRQ pin?

Answer 10 No, The FLxBPRs can be programed or erased without V_{HI} on the IRQ pin. Refer to **FLASH Block Protection**.

Question 11 I want to perform both page and mass erase operations using the minimum mass erase time (t_{MERASE}) and minimum mass high voltage hold time (t_{NVHL}) since these delays cover the page delays (t_{ERASE} and t_{NVH}). Does the page operation cause any problems using the mass delays?

Answer 11 No, there is no erase disturb. However, it may reduce the endurance of the FLASH memory. Freescale recommends using the minimum time allowed for both page and mass erase operations.

Question 12 I'm sending external data into the MC68HC908AS60A/AZ60A for programming. How can I speed up this programming process?

Answer 12 Excluding data download time, it takes about two seconds to program 60 Kbytes of FLASH. If data is transferred at a higher communication baud rate or in a parallel manner, the overall programming time can be reduced. The ideal situation would be a fast parallel transfer of data during the delay times associated with the programming algorithm.

Question 13 If I program FLASH with 2-MHz bus frequency, can I read the FLASH with 8-MHz bus frequency without any problems?

Answer 13 Yes. The FLASH will meet all specifications, including data retention performance, if the FLASH is programmed/erased and used within specification limits.

Question 14 The program/erase operation is not successful in the monitor mode.

Answer 14 The FLASH memory is protected in the monitor mode to make it difficult for unauthorized users to view the memory contents. Before programming/erasing FLASH, the security feature on the part must be “broken” to view the FLASH contents. When an attempt to break security fails, the FLASH is not addressed during reads and invalid data will be observed. Refer to the monitor ROM section, which describes how to break security, in *MC68HC908AZ60A and MC68HC908AS60A Technical Data*, Freescale document order number MC68HC908AZ60A/D.

Question 15 I have failed to break security in monitor mode. Can I execute a mass erase to erase the security?

Answer 15 Yes. Mass erase is the only FLASH operation allowed after failing to break security in monitor mode. When the mass erase operation is executed, the address specified in step 3 of the sequence (refer to **FLASH Mass Erase Algorithm**) must be the address of the FLASH block protect register. Make sure the block protect feature is not asserted or override it to mass erase the device.

Question 16 In monitor mode, how can I tell if the break security has been successful?

Answer 16 If the security check was unsuccessful, memory reads will return the same data for every byte read instead of the code or data expected.

Question 17 Do I need to confirm the memory contents after programming the FLASH?

Answer 17 It is recommended that the code used to program the FLASH also include a verification step to ensure the integrity of the data programmed into the FLASH.

Question 18 A block of memory in the FLASH array is protected by programming the block protect register. When I execute a mass erase operation without applying high voltage on the IRQ pin, can an unprotected block be erased?

Answer 18 No. When a FLASH array is partially protected, the mass erase cannot be performed unless high voltage is placed on IRQ.

NOTE: *When a high voltage is placed on the IRQ pin, the block protection register is ignored and the entire memory array will be erased if a mass erase operation is executed.*

Question 19 What is the expected lifetime of FLASH memory?

Answer 19 The minimum program/erase endurance and data retention lifetime of the FLASH memory for all conditions is found in the *MC68HC908AZ60A and MC68HC908AS60A Technical Data* book, Freescale document order number MC68HC908AZ60A/D.

Question 20 What steps can I take to prolong the life of the FLASH memory?

Answer 20 The FLASH memory has a finite program/erase durability and a finite data retention lifetime. However, the specification shows the minimum lifetime considering the worst case set of conditions applied to the part. In general, the FLASH will last longer if it is used at moderate temperatures (0–70^{o}C) and the program/erase cycles are kept to a minimum.

Question 21 Can I program/erase the FLASH at the maximum temperature limits continuously for the specified lifetime of the part?

Answer 21 Yes. Row program and erase cycle durability is specified to be 10-K maximum for each. However, exceeding that value is not recommended. The FLASH can be read continuously over the life of the device.

Question 22 What modes of operation cause the most noise?

Answer 22 Program and erase modes cause a significant amount of EMI (electromagnetic interference) and power supply noise due to the high transient current demand of the charge pump. High accuracy ADC (analog-to-digital) conversions may not be possible while the FLASH is programming or erasing.

EEPROM Frequently Asked Questions

These questions and answers are designed to help the user with frequent concerns.

Question 1 I cannot program/erase EEPROM memory at all. What should I consider to make my program/erase code work?

Answer 1 Check the following:

- *Did you program correct divider values at both EExDIV nonvolatile high register (EExDIVHNVR) and EExDIV nonvolatile low register (EExDIVLNVR)?*
  If you did not program them, did you write correct divider values at both EExDIV divider high register (EExDIVH) and EExDIV divider low register (EExDIVL) before executing the erase or program algorithm operation?
  To program and erase the EEPROM, the EEPROM control requires a constant timebase of 35 µs to drive its internal timer. If proper divider values are not initialized at the divider registers, the EEPROM erase and/or program operation is not performed properly. The correct divider value is calculated by the formula in **Configuration Register 2 and EEPROM Timebase Divider Control Registers**. There are two ways to set correct divider values. One of them is to program the divider value at the EExDIVHNVR and EExDIVLNVR registers. The other is to write the value at EExDIVH and EExDIVL registers temporarily before programming and/or erasing operations.
- *Did you select a correct EEPROM reference clock source?*
  The EEPROM reference clock source may be either CGMXCLK or the system bus clock. The selection of this reference clock source is defined by the EEDIVCLK bit in the configuration register (CONFIG-2). The divider value will be changed depending on which reference clock source you select. Make sure the divider value is calculated using a correct reference clock speed.

NOTE: *CONFIG-2 is a “write-once” register and has EEDIVCLK = CGMXCLK out of reset.)*

- *Did you use our recommended programming and erasing algorithms in your code?*
  The EEPROM consists of FLASH memory surrounded by a logic state machine. Freescale's recommended programming algorithm ensures that the EEPROM is programmed for sufficient data retention and in a minimum program time. Freescale does not guarantee the performance of the EEPROM if the recommended algorithms are not followed.
- *Is each step of the programming and erasing algorithms performed in the right order?*
  The sequence of the program and erase operations are interlocked in hardware so only the prescribed order of these operations can occur. However, other non-EEPROM operations may occur between the steps shown.
- *Is the memory block where you want to program/erase unprotected?*
  The block protect feature of the EEPROM is present to prevent unintentional programming or erasing. The block protect bits must be cleared such that the memory to be erased or programmed is unprotected.
- *Are delay times such as t_{EEBYTE}, t_{EEBULK}, $t_{EEBLOCK}$, t_{EEFPV}, etc., within the specification?*
  Timing is critical to ensure proper EEPROM operation. Delay times that are too long or too short can alter the EEPROM performance to the point where it does not work or is not reliable. Freescale does not guarantee EEPROM performance if the wrong delay times are used.
- *Is the correct EEPROM register being written to enable erase or program?*
  The MC68HC908AS60A/AZ60A has two EEPROM arrays with two separate sets of control and block protect registers. Make sure the appropriate register is being addressed.

Question 2 Why do I need to set up constant timebase?

Answer 2 The EEPROM is actually a FLASH cell surrounded by a logic state machine. The state machine requires an accurate clock source for applying high voltage during the erase and program operations.

Question 3 What is the benefit of using the AUTO mode?

Answer 3 When you use the AUTO EEPROM programming and erasing algorithms, the programming and erasing time is much less than when you use the standard EEPROM algorithms. Whenever programming or erasing is done, the EEPGM bit is automatically cleared, eliminating the wait for a fixed delay time. Furthermore, since the delay time is not necessary, the delay routine is not required in your code.

Question 4 During a program/erase process, can I execute an interrupt service or include additional steps?

Answer 4 Unrelated (non-EEPROM) steps may be included between steps of the program/erase algorithms as long as the sequence of the steps remains consistent. However, interrupt service routines can cause errors in the program or erase timing and lead to corrupt or missing data in the EEPROM.Freescale does not guarantee performance of the EEPROM if interrupts are not masked during the program or erase operation.

Question 5 Can I program each bit in the same EEPROM location successively?

Answer 5 No. However, the same byte location can be successively programmed using selective bit programming. Refer to **Selective Bit Programming**.

Question 6 I programmed the EExNVR register to un-protect EEPROM blocks, so why are the blocks are still protected?

Answer 6 Did you reset the part or read the EExNVR after the register was programmed? Since a new configuration value is not loaded automatically to the EExACR register, resetting or reading the EExNVR is required to set up the new configuration.

Question 7 When the EEPRTCT bit in the EEPROM nonvolatile register (EExNVR) is programmed with zero, how does it affect unsecured locations? (Secured locations are EEPROM-1: $800–$8EF and EEPROM2: $900–$9EF.)

Answer 7 Once the EEPRTCT bit is cleared, the block and bulk erase operations are disabled. However, byte erase and byte program operations are still enabled. Therefore, you can erase and/or program unsecured locations if the locations are not protected in the block protect register.

Question 8 Is one charge pump used for both EEPROM arrays? Is the charge pump also shared with FLASH?

Answer 8 No. Each array has a separate charge pump but the address decode logic does not allow the other array to be programmed or erased at a time. The FLASH has its own charge pump.

Question 9 Do I need to confirm the memory contents after programming the EEPROM?

Answer 9 It is recommended that the code used to program the EEPROM also include a verification step to ensure the integrity of the data programmed.

Question 10 What is the expected lifetime of EEPROM memory?

Answer 10 The minimum program/erase endurance and data retention lifetime of the EEPROM memory for all conditions is found in *MC68HC908AZ60A and MC68HC908AS60A Technical Data*, Freescale document order number MC68HC908AZ60A/D.

Question 11 What steps can I take to prolong the life of the EEPROM memory?

Answer 11 The EEPROM memory has a finite program/erase durability and a finite data retention lifetime. However, the specification quotes the minimum guaranteed lifetime considering the worst case set of conditions applied to the part. In general, the EEPROM array will last longer if the program/erase cycling is kept to a minimum and the temperature is kept at a nominal level (0–70^{o}C).

Question 12 Can I program/erase the EEPROM at the maximum temperature limits for the specified life of the part?

Answer 12 Yes. Program / erase cycle durability is specified to be 10-K minimum. However, exceeding that value is not recommended. Reading the EEPROM can occur continuously over the life of the product.

FLASH Assembly Source Code Flowcharts

The main routines `SSTerase.mrt` and `SSTprog.mrt` initialize the device for erasing and programming operations. `SSTerase.mrt` specifies the size and the location of the erase block before calling the `FlashErase` subroutine which follows the algorithm listed in this application note. `SSTprog.mrt` fills a RAM data buffer with values to program, specifies the location to start programming, and defines the number of bytes of the row to program before jumping to the `ProgRow` subroutine. `SSTprog.mrt` also includes a verification step after the programming is completed.

The `FlashErase` and the `ProgRow` subroutines follow the flowcharts shown in **Figure 3** and **Figure 4** closely. Flowcharts are also included for the `WriteFLCR` subroutine which is used to set or clear various bits in the FLxCR registers and the `ms_delay` subroutine which generates delays greater than 1 millisecond.

The flowcharts for `SSTerase.mrt`, `SSTprog.mrt`, `FlashErase`, `ProgRow`, `WriteFLCR`, and `ms_delay` are **Figure 15**, **Figure 16**, **Figure 17**, **Figure 18**, **Figure 19**, and **Figure 20**, respectively.

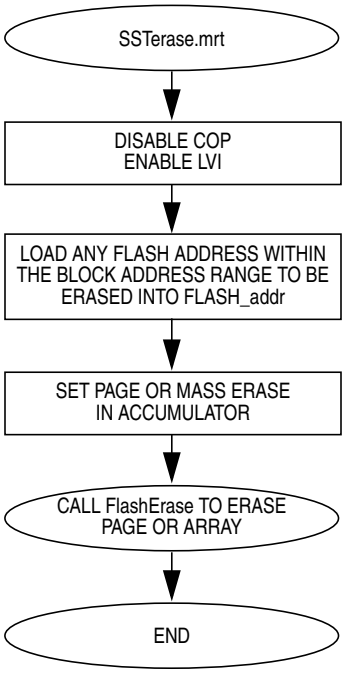


Figure 15. FLASH Erasing Main Routine Flowchart

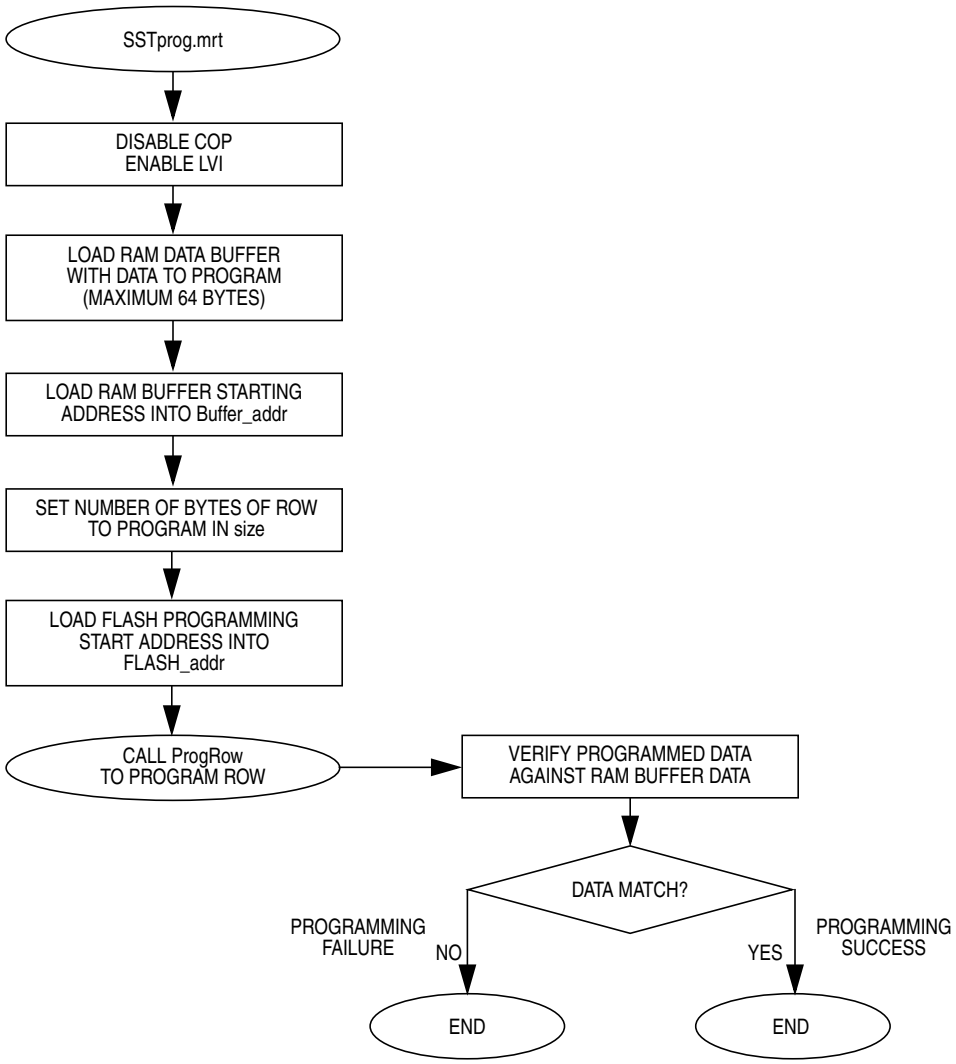


Figure 16. FLASH Programming Main Routine Flowchart

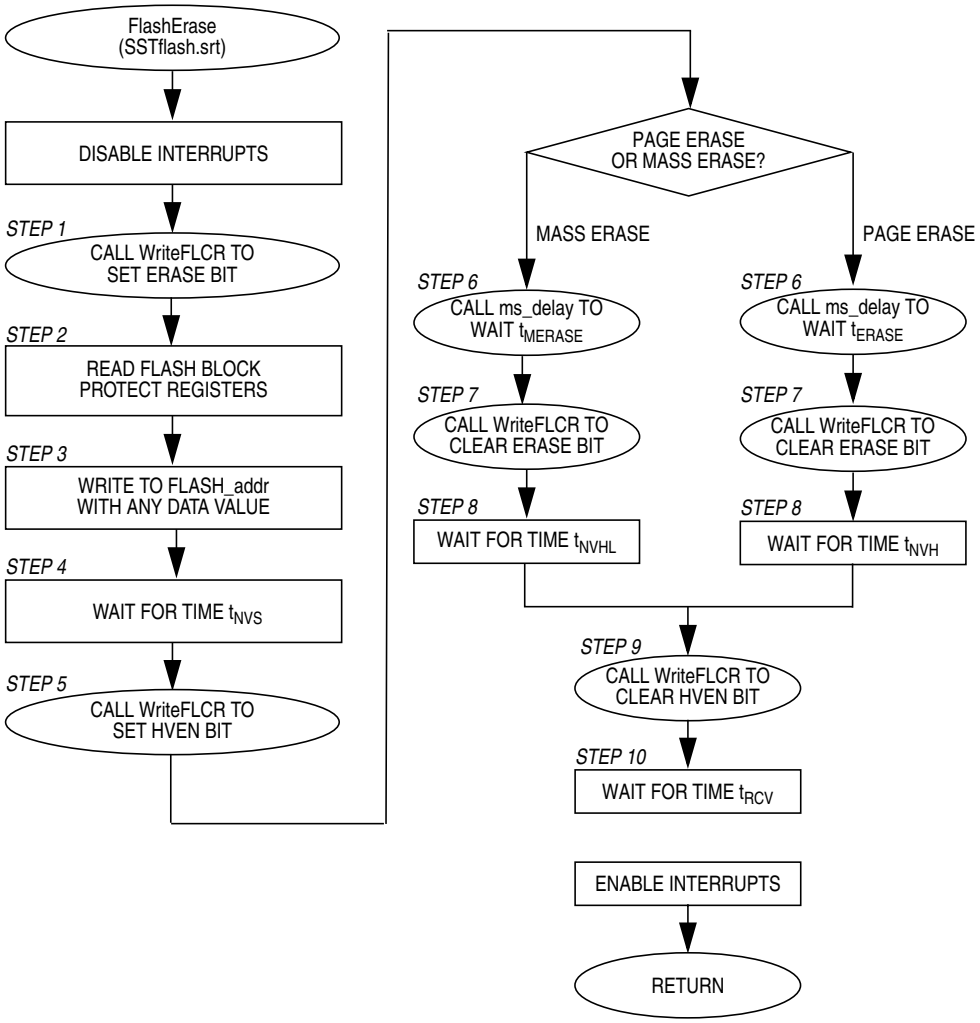


Figure 17. Subroutine FlashErase Flowchart

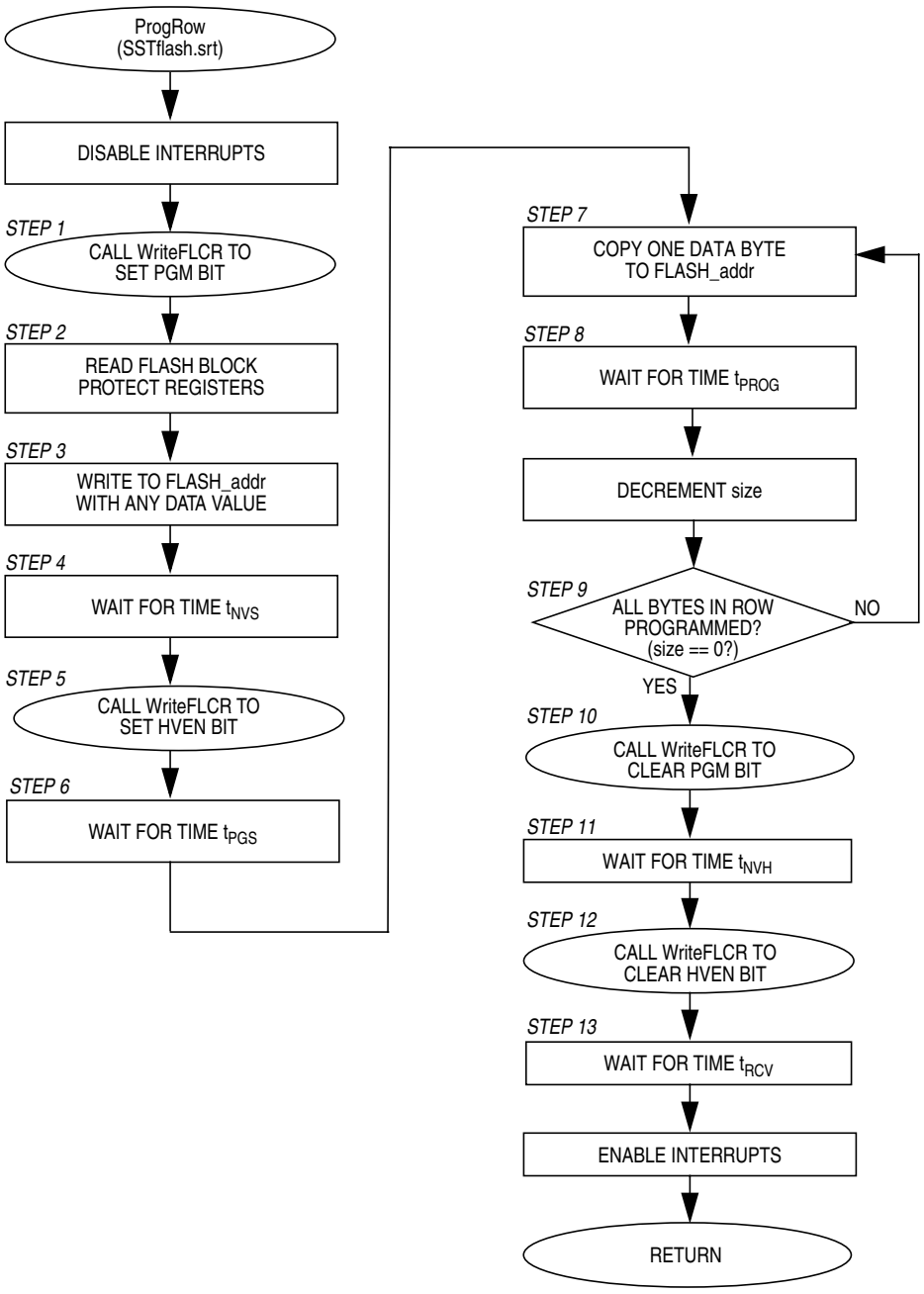


Figure 18. Subroutine ProgRow Flowchart

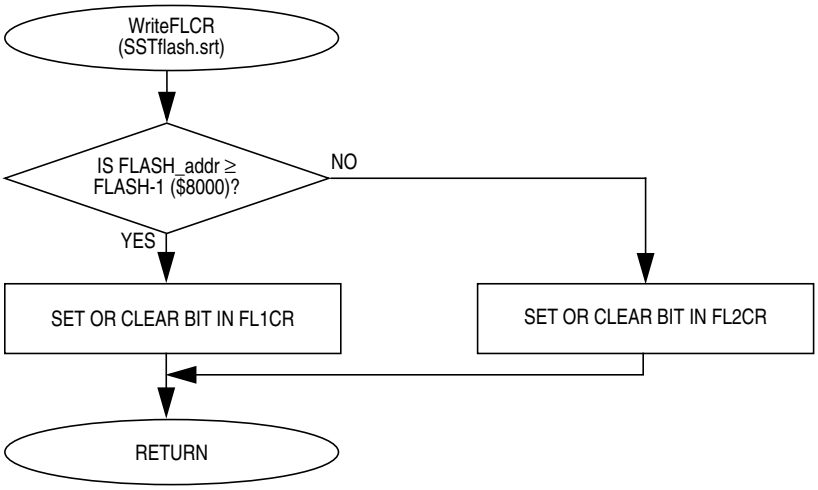


Figure 19. Subroutine WriteFLCR Flowchart

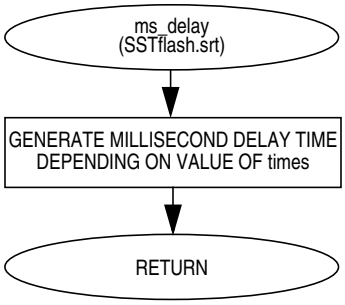


Figure 20. Subroutine Delay Flowchart

FLASH Assembly Source Code

```
*****************************************************************************
*****************************************************************************
*                                                                           *
*           SST FLASH Memory Erasing on the MC68HC908AS60A/AZ60A            *
*                                                                           *
*****************************************************************************
* File Name: SSTerase.mrt                     Copyright (c)  *
*                                                                           *
* Current Revision: 1.0                                                     *
* Current Release Level: RP                                                 *
* Current Revision Release Date: 06/21/2001                                 *
*                                                                           *
* Current Release Written By: Adeela Gill and Kazue Kikuchi                 *
*                        Freescale Applications Engineering - Austin, TX    *
*                                                                           *
* Assembled Under: CASM08Z (P&E Microcomputer Systems, Inc.)                *
*                       Ver.: 3.16                                          *
*                                                                           *
* Part Family Software Routine Works With: HC08                             *
*                                                                           *
* Routine Size (Bytes):       209                                           *
* Stack Space Used (Bytes):   4                                             *
* RAM Used (Bytes):           5                                             *
* Global Variables Used:      FLASH_addr                                    *
* Subroutine Called:          FlashErase                                    *
*                                                                           *
* Full Functional Description Of Routine Design:                            *
*       SSTerase.mrt is the main routine that demonstrates how to erase     *
*       SST FLASH memory on the MC68HC908AS60A and MC68HC908AZ60A.  The     *
*       memory may be erased on a per-page (128 bytes) basis or the         *
*       entire array may be erased at one time (mass erase.)                *
*****************************************************************************
* Freescale reserves the right to make changes without further notice to    *
* any product herein.  Freescale makes no warranty, representation or       *
* guarantee regarding the suitability of its products for any particular    *
* purpose, nor does Freescale assume any liability arising out of the       *
* application or use of any product, circuit, and specifically disclaims    *
* any and all liability, including without limitation consequential or      *
* incidental damages.  "Typical" parameters can and do vary in different    *
* applications.  All operating parameters, including "Typicals" must be     *
* validated for each customer application by customer's technical experts.  *
* Freescale does not convey any license under its patent rights nor the     *
* rights of others.  Freescale products are not designed, intended, or      *
* authorized for use as components in systems intended for surgical         *
* implant into the body, or other applications intended to support or       *
* sustain life, or for any other application in which the failure of the    *
* Freescale product could create a situation where personal injury or death *
* may occur. Should Buyer purchase or use Freescale products for any such   *
* intended or unauthorized application, Buyer shall indemnify and hold      *
* Freescale and its officers, employees, subsidiaries, affiliates, and      *
* distributors harmless against all claims, costs, damages, and expenses,   *
```

```
* and reasonable attorney fees arising out of, directly or indirectly, any*
* claim of personal injury or death associated with such unintended or    *
* unauthorized use, even if such claim alleges that Freescale was negligent*
* regarding the design or manufacture of the part.  Freescale and the      *
* Freescale symbol are registered trademarks of Freescale Semiconductor, Inc.  *
* Freescale Semiconductor, Inc. is an Equal Opportunity/Affirmative Action Employer.                *
*****************************************************************************
*****************************************************************************
*****                        Program Specific Equates                   *****
*****************************************************************************
masserase. equ     %00000100
pageerase. equ     %00000000
*****************************************************************************
*****                         Include Files                             *****
*****************************************************************************
NOLIST
$INCLUDE      "H908AS60A.frk"   ;Equates for all registers and bits in
                                ; the MC68HC908AS60A
        org   ram1
$INCLUDE      "SSTflash.var"    ;RAM variable definitions
LIST
*****************************************************************************
*****                          Main Routine                             *****
*****************************************************************************
        org   ram2
Start:
        mov   #$71,config-1     ;Turn off the COP, but leave the LVI on

        ldhx  #$8045            ;Any address within the page or the array
                                ; will be erased
        sthx  FLASH_addr        ;Load the address to FLASH_addr

        lda   #pageerase.       ;Select "PAGE" or "MASS" erase with
                                ; Mass bit
        jsr   FlashErase        ;Erase the area of FLASH including the
                                ; specified address

        bra   *
*****************************************************************************
*****               Subroutine Body Includes Section                    *****
*****************************************************************************
$INCLUDE      "SSTflash.srt"    ;SST FLASH subroutines


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
*****************************************************************************
*****************************************************************************
*                                                                           *
*         SST FLASH Memory Programming on the MC68HC908AS60A/AZ60A          *
*                                                                           *
*****************************************************************************
* File Name: SSTprog.mrt                    Copyright (c)  *
*                                                                           *
* Current Revision: 1.0                                                     *
* Current Release Level: RP                                                 *
```

```
* Current Revision Release Date: 6/21/2001                                  *
*                                                                           *
* Current Release Written By: Adeela Gill and Kazue Kikuchi                 *
*                        Freescale Applications Engineering - Austin, TX    *
*                                                                           *
* Assembled Under: CASM08Z (P&E Microcomputer Systems, Inc.)                *
*                       Ver.: 3.16                                          *
*                                                                           *
* Part Family Software Routine Works With: HC08                             *
*                                                                           *
* Routine Size (Bytes):       269                                           *
* Stack Space Used (Bytes):   4                                             *
* RAM Used (Bytes):           69                                            *
* Global Variables Used:      FLASH_addr, Buffer_addr, data, size           *
* Subroutine Called:          ProgRow                                       *
*                                                                           *
* Full Functional Description Of Routine Design:                            *
*       SSTprog.mrt is the main routine for the programming operation. It   *
*       demonstrates an SST programming algorithm that minimizes the        *
*       amount of time needed to program a row of FLASH memory on the       *
*       MC68HC908AS60A or the MC68HC908AZ60A. One row consists of 64        *
*       consecutive bytes of FLASH memory within specified address ranges.*
*****************************************************************************
*****************************************************************************
*****                           Include Files                           *****
*****************************************************************************
NOLIST
$INCLUDE      "H908AS60A.frk"   ;Equates for all registers and bits in
                                ; the MC68HC908AS60A
        org   ram1
$INCLUDE      "SSTflash.var"    ;RAM variable definitions
LIST
*****************************************************************************
*****                          Main Routine                             *****
*****************************************************************************
        org   ram2
Start:
        mov   #$71,config-1     ;Turn off the COP, but leave the LVI on
        ldhx  #$0000
        lda   #$1
Data_load:
        sta   data,x            ;Fill the RAM buffer, 64 bytes data,
        inca                    ; values to program into FLASH
        aix   #$1               ; (ie. 01,02,03,.........,3E,3F,40)
        cphx  #!64
        bne   Data_load

        ldhx  #data             ;Load Buffer_addr with the start address
        sthx  Buffer_addr       ; of the RAM buffer
        lda   #!64              ;Set the number of bytes to program
        sta   size
        ldhx  #$8040            ;Load FLASH_addr with programming start
        sthx  FLASH_addr        ; address

        jsr   ProgRow           ;Program a row
```

```
Verify:                         ;After the desired block is programmed,
        lda   #!64              ; the verify of programmed data is highly
        sta   size              ; recommended
        ldhx  #data
        sthx  Buffer_addr
        ldhx  #$8040
        sthx  FLASH_addr
Verify_Loop:
        lda   ,x                ;Read data from a FLASH location
        aix   #$1
        sthx  FLASH_addr
        ldhx  Buffer_addr
        cmp   ,x                ;Compare the data with data in the RAM
                                ; buffer
        bne   Error             ;When the data is not correct, branch
                                ; to Error
        dec   size
        beq   Success           ;If all bytes in the row are programmed
        aix   #$1               ; correctly, branch to Success
        sthx  Buffer_addr
        ldhx  FLASH_addr
        bra   Verify_Loop
Success:
        bra   *                 ; ** Programming Successful **
                                ; End of program
Error:                          ; ** Programming Failed **
        bra   *                 ; Take appropriate action
*************************************************************************
*****               Subroutine Body Includes Section                *****
*************************************************************************
$INCLUDE      "SSTflash.srt"    ;SST FLASH subroutine


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
*************************************************************************
*************************************************************************
*                                                                       *
*  SST FLASH Memory Programming and Erasing on the MC68HC908AS60A/AZ60A   *
*                                                                       *
*************************************************************************
* File Name: SSTflash.var                Copyright (c)      *
*                                                                       *
* Current Revision: 1.0                                                 *
* Current Release Level: RP                                             *
* Current Revision Release Date: 6/21/2000                              *
*                                                                       *
* Current Release Written By: Adeela Gill and Kazue Kikuchi             *
*                       Freescale Applications Engineering - Austin, TX   *
*                                                                       *
* Assembled Under: CASM08Z (P&E Microcomputer Systems, Inc.)            *
*                      Ver.: 3.16                                       *
*                                                                       *
* Part Family Software Routine Works With: HC08                         *
*                                                                       *
* RAM Used (Bytes): 71                                                  *
```

```
*                                                                             *
* Description:                                                                *
*       RAM variable definitions for SSTprog.mrt and SSTerase.mrt             *
*******************************************************************************
*******************************************************************************
*****                          RAM Variables                              *****
*******************************************************************************
FLASH_addr    rmb    $2         ;16 bit Address of FLASH memory to erase
                                ; or program
Buffer_addr   rmb    $2         ;16 bit Address of RAM data buffer
data          rmb    !64        ;64 data bytes that will be programmed
size          rmb    $1         ;1 byte storage of the byte number
                                ; contained in a row
times         rmb    $1         ;1 byte in which the delay time will be
                                ; determined
temp          rmb    $1         ;1 byte temporary storage


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
*******************************************************************************
*******************************************************************************
*                                                                             *
* SST FLASH Memory Program & Erase Subroutines on the MC68HC908AS60A/AZ60A*
*                                                                             *
*******************************************************************************
* File Name: SSTflash.srt                  Copyright (c)   *
*                                                                             *
* Current Revision: 1.1                                                       *
* Current Release Level: RP                                                   *
* Current Revision Release Date: 6/21/2001                                    *
*                                                                             *
* Current Release Written By: Adeela Gill and Kazue Kikuchi                   *
*                       Freescale Applications Engineering - Austin, TX   *
*                                                                             *
* Assembled Under: CASM08Z (P&E Microcomputer Systems, Inc.)                  *
*                     Ver.: 3.16                                              *
*                                                                             *
* Part Family Software Routine Works With: HC08                               *
*                                                                             *
* Revision History:                                                           *
*   Rev. 1.1    Delay time evaluation code added                              *
*                                                                             *
*  Module Size (Bytes):          FlashErase     82                           *
*                                ProgRow        76                           *
*                                WriteFLCR      20                           *
*                                ms_delay       16                           *
*  Stack Space Used (Bytes):     FlashErase     2                            *
*                                ProgRow        2                            *
*                                WriteFLCR      0                            *
*                                ms_delay       0                            *
*  RAM Used (Bytes):             FlashErase     4                            *
*                                ProgRow        70                           *
*                                WriteFLCR      0                            *
*                                ms_delay       1                            *
*  Global Variable(s) Used:      FlashErase     FLASH_addr                   *
```

```
*                               ProgROW         FLASH_addr, Buffer_addr   *
*                                               data, size                *
*                               WriteFLCR       FLASH_addr                *
*                               ms_delay        None                      *
*  Submodule(s) Called:         EraseRoutine    WriteFLCR, ms_delay       *
*                               Prog8Bytes      WriteFLCR                 *
*                               WriteFLCR       None                      *
*                               ms_delay        None                      *
*  Calling Sequence:            JSR FlashErase, JSR ProgRow               *
*                               JSR WriteFLCR, JSR ms_delay               *
*  Entry Label:                 FlashErase, ProgRow, WriteFLCR, ms_delay  *
*                                                                         *
*  Entry Conditions:            FlashErase    2 bytes address defined at  *
*                                              FLASH_addr                 *
*                                             Mass bit definition passed  *
*                                              in accumulator             *
*                               ProgRow       2 bytes address defined at  *
*                                              FLASH_addr                 *
*                                             2 bytes address defined at  *
*                                              Buffer_addr                *
*                                             Maximum 64 programming      *
*                                              bytes located at           *
*                                              variables data             *
*                                             1 byte defined at SIZE      *
*                               WriteFLCR     2 bytes address defined at  *
*                                              FLASH_addr                 *
*                                             FLCR (FLASH Control         *
*                                              Register) bit definition   *
*                                              passed in accumulator      *
*                               ms_delay      Delay valiable passed in    *
*                                              times                      *
*  Number of Exit Points:          4                                      *
*     Exit Label:               FlashErase     FlashErase_End             *
*                               ProgRow        ProgRow_End                *
*                               WriteFLCR      WriteFLCR_End              *
*                               ms_delay       ms_delay_End               *
*     Exit Conditions:          FlashErase     None                       *
*                               ProgRow        None                       *
*                               WriteFLCR      None                       *
*                               ms_delay       None                       *
*                                                                         *
* Full Functional Description Of Subroutine:                              *
*       SSTflash.srt consists of two primary subroutines called           *
*       FlashErase and ProgRow. These demonstrate SST FLASH erasing and   *
*       programming algorithms, respectively. The routines also call      *
*       other subroutines WriteFLCR and ms_delay. Since delay times must  *
*       be met precisely for successful FLASH programming and erasing,    *
*       additional software was added to measure the delay times. This    *
*       code is included as comments throughout the file. It is           *
*       recommended that the user verify all delay times before using     *
*       this software for production.                                     *
*       Note: Each delay time related to SST FLASH program and erase      *
*             operations was calculated with a bus speed of 2.4576MHz.    *
***************************************************************************
```

```
*****************************************************************************
*****                    SST FLASH Erase Subroutine                     *****
*****************************************************************************
FlashErase:
    ;--------------------------------------------------------------------;
    ; Delay Time Evaluation                                             ;
    ;  Initialize Port D bit 3 as output high                           ;
    ;   bset  3,PTD             ;Set Port D bit 3                       ;
    ;   bset  3,DDRD            ;Select output for Port D bit 3         ;
    ;--------------------------------------------------------------------;
        sei                     ;Disable interrupts
        ldhx  FLASH_addr        ;Load the starting address of the area
                                ; to be erased in the HX registers
        sta   temp              ;Store a value in accumulator to temp
        ora   #erase.           ;Step 1 - Set the ERASE bit
        jsr   WriteFLCR         ; If MASS bit is set, the MASS erase will
                                ; be performed
                                ; If MASS bit is clear, the PAGE erase
                                ; will be performed
        lda   fl1bpr            ;Step 2 - Read from block protect
        lda   fl2bpr            ; registers
        sta   ,X                ;Step 3 - Write to any FLASH address
                                ; within the area address range to be
                                ; erased with any data value
    ;--------------------------------------------------------------------;
    ; Delay Time tNVS Evaluation  (Time between points S1 and E1)       ;
    ;   Measure low level period on Port D bit 3 pin using a scope      ;
    ;--------------------------------------------------------------------;
    ; Delay Evaluation: Point S1
    ;   bclr  3,PTD             ;Clear Port D bit 3

        lda   #$8               ;Step 4 - Wait for time tNVS
        dbnza *                 ; 2 + (3 x 8) cycles = 26 cycles (10.6us)

    ; Delay Evaluation: Point E1
    ;   bset  3,PTD             ;Set Port D bit 3

        lda   #hven.            ;Step 5 - Set the HVEN bit
        jsr   WriteFLCR

        brset MASS,temp,MASS_Erase
                                ;If MASS erase, jump to MASS_Erase
PAGE_Erase:                     ;PAGE Erase
    ;--------------------------------------------------------------------;
    ; Delay Time tERASE Evaluation  (Time between points S2 and E2)     ;
    ;   Measure low level period on Port D bit 3 pin using a scope      ;
    ;--------------------------------------------------------------------;
    ; Delay Evaluation: Point S2
    ;   bclr  3,PTD             ;Clear Port D bit 3

        lda   #!1               ;Step 6 - Wait for time tERASE (1.0ms)
        sta   times
        jsr   ms_delay

    ; Delay Evaluation: Point E2
```

```
    ;   bset  3,PTD             ;Set Port D bit 3

        lda   #erase.           ;Step 7 - Clear the ERASE bit
        jsr   WriteFLCR
    ;---------------------------------------------------------------;
    ; Delay Time tNVH Evaluation  (Time between points S3 and E3)   ;
    ;   Measure low level period on Port D bit 3 pin using a scope  ;
    ;---------------------------------------------------------------;
    ; Delay Evaluation: Point S3
    ;   bclr  3,PTD             ;Clear Port D bit 3

        lda   #$4               ;Step 8 - Wait for time tNVH
        dbnza *                 ; 2 + (3 x 4) cycles = 14 cycles (5.7us)

    ; Delay Evaluation: Point E3
    ;   bset  3,PTD             ;Set Port D bit 3

        bra   Step9
MASS_Erase:                     ;MASS Erase
    ;---------------------------------------------------------------;
    ; Delay Time tMERASE Evaluation  (Time between points S4 and E4)  ;
    ;   Measure low level period on Port D bit 3 pin using a scope  ;
    ;---------------------------------------------------------------;
    ; Delay Evaluation: Point S4
    ;   bclr  3,PTD             ;Clear Port D bit 3

        lda   #!4               ;Step 6 - Wait for time tMERASE (4.0ms)
        sta   times
        jsr   ms_delay

    ; Delay Evaluation: Point E4
    ;   bset  3,PTD             ;Set Port D bit 3

        lda   #erase.           ;Step 7 - Clear the ERASE bit
        jsr   WriteFLCR
    ;---------------------------------------------------------------;
    ; Delay Time tNVHL Evaluation  (Time between points S5 and E5)  ;
    ;   Measure low level period on Port D bit 3 pin using a scope  ;
    ;---------------------------------------------------------------;
    ; Delay Evaluation: Point S5
    ;   bclr  3,PTD             ;Clear Port D bit 3

        lda   #$52              ;Step 8 - Wait for time tNVHL
        dbnza *                 ; 2 + (3 x 82) cycles = 248 cycles
                                ; (101us)
    ; Delay Evaluation: Point E5
    ;   bset  3,PTD             ;Set Port D bit 3

Step9:
        lda   #hven.            ;Step 9 - Clear the HVEN bit
        jsr   WriteFLCR
    ;---------------------------------------------------------------;
    ; Delay Time tRCV Evaluation  (Time between points S6 and E6)   ;
    ;   Measure low level period on Port D bit 3 pin using a scope  ;
    ;---------------------------------------------------------------;
```

```
    ; Delay Evaluation: Point S6
    ;   bclr  3,PTD             ;Clear Port D bit 3

        lda   #$1               ;Step 10 - Wait for time tRCV
        dbnza *                 ; 2 + 3 cycles = 5 cycles (2.0us)

    ; Delay Evaluation: Point E6
    ;   bset  3,PTD             ;Set Port D bit 3

        lda   #$00
        sta   fl1cr             ;Clear all bits in Flash Control
        sta   fl2cr             ; registers
        cli                     ;Enable interrupts
FlashErase_End:
        rts
*****************************************************************************
*****              SST FLASH Programming Subroutine                     *****
*****************************************************************************
ProgRow:
    ;-------------------------------------------------------------------;
    ; Delay Time Evaluation                                             ;
    ;  Initialize Port D bit 3 as output high                           ;
    ;   bset  3,PTD             ;Set Port D bit 3                       ;
    ;   bset  3,DDRD            ;Select output for Port D bit 3         ;
    ;-------------------------------------------------------------------;
        sei                     ;No interrupts allowed during programming
        ldhx  FLASH_addr        ;Load the address of the page to be
                                ; programmed in the HX registers
        lda   #pgm.             ;Step 1 - Set the PGM bit
        jsr   WriteFLCR
        lda   fl1bpr            ;Step 2 - Read from the block protect
        lda   fl2bpr            ; registers
        sta   ,x                ;Step 3 - Write to any FLASH address with
                                ; any data within the row address range
                                ; desired.
    ;-------------------------------------------------------------------;
    ; Delay Time tNVS Evaluation  (Time between points S7 and E7)       ;
    ;   Measure low level period on Port D bit 3 pin using a scope      ;
    ;-------------------------------------------------------------------;
    ; Delay Evaluation: Point S7
    ;   bclr  3,PTD             ;Clear Port D bit 3

        lda   #$8               ;Step 4 - Wait for time tNVS
        dbnza *                 ; 2 + (3 x 8) cycles = 26 cycles (10.6us)

    ; Delay Evaluation: Point E7
    ;   bset  3,PTD             ;Set Port D bit 3

        lda   #hven.            ;Step 5 - Set the HVEN bit
        jsr   WriteFLCR
    ;-------------------------------------------------------------------;
    ; Delay Time tPGS Evaluation  (Time between points S8 and E8)       ;
    ;   Measure low level period on Port D bit 3 pin using a scope      ;
    ;-------------------------------------------------------------------;
    ; Delay Evaluation: Point S8
```

```
    ;   bclr  3,PTD             ;Clear Port D bit 3

        lda   #$4               ;Step 6 - Wait for time tPGS
        dbnza *                 ; 2 + (3 x 4) cycles = 14 cycles (5.7us)

    ; Delay Evaluation: Point E8
    ;   bset  3,PTD             ;Set Port D bit 3

Copy_Loop:
        ldhx  Buffer_addr       ;Step 7 - Copy one byte data from the
        lda   ,x                ; RAM buffer to the appropriate FLASH
                                ; location
        aix   #$1               ;Increment Buffer_addr for next byte
        sthx  Buffer_addr       ; write
        ldhx  FLASH_addr
    ;-------------------------------------------------------------
    ;- tPROG is defined as the total time from writing one data  -
    ;- byte to writing the next data byte. (labelled "A" below). -
    ;- For the last byte programmed, tPROG is defined as the     -
    ;- time from writing the data byte ("A") to clearing the PGM -
    ;- bit (in the WriteFLCR routine). Both of these loops       -
    ;- should be executed in a time between 30 and 40 us.        -
    ;-                                                           -
    ;- The byte-to-next-byte time is 75 cycles (30.5 us).        -
    ;- The byte-to-PGM time is 77 cycles (31.3 us).              -
    ;-------------------------------------------------------------
        sta   ,x                ;Write the Data Byte ("A")
    ;-------------------------------------------------------------------;
    ; Delay Time byte-to-next-byte Evaluation  (one loop period       ;
    ; starting from point S9 to point E9)                             ;
    ;   Measure low level period on Port D bit 3 pin using a scope    ;
    ;-------------------------------------------------------------------;
    ; Delay Evaluation: Point S9, E9
    ;   com   PTD               ;Complement Port D bit 3
    ;-------------------------------------------------------------------;
    ; Delay Time byte-to-PGM Evaluation  (Time between points S10 and ;
    ; E10_1 & time between points S10 and E10_2)                      ;
    ;   Measure low level period on Port D bit 3 pin using a scope    ;
    ;-------------------------------------------------------------------;
    ; Delay Evaluation: Point S10
    ;   bclr  3,PTD             ;Clear Port D bit 3

        lda   #$0d              ;Step 8 - Delay, part of tPROG
        dbnza *

        dec   size              ;Step 9 - Repeat step 7 and 8 until all
        beq   Copy_End          ; the bytes within the row are programmed

        aix   #$1               ;Increment FLASH_addr for next byte
        sthx  FLASH_addr        ; write

        bra   Copy_Loop
Copy_End:
        lda   #$2               ;Step 8 - Delay, part of tPROG
        dbnza *
```

```
        lda   #pgm.             ;Step 10 - Clear the PGM bit
        jsr   WriteFLCR
    ;--------------------------------------------------------------;
    ; Delay Time tNVH Evaluation  (Time between points S11 and E11)   ;
    ;   Measure low level period on Port D bit 3 pin using a scope    ;
    ;--------------------------------------------------------------;
    ; Delay Evaluation: Point S11
    ;   bclr  3,PTD             ;Clear Port D bit 3

        lda   #$4               ;Step 11 - Wait for time tNVH
        dbnza *                 ; 2 + (3 x 4) cycles = 14 cycles (5.7us)

    ; Delay Evaluation: Point E11
    ;   bset  3,PTD             ;Set Port D bit 3

        lda   #hven.            ;Step 12 - Clear the HVEN bit
        jsr   WriteFLCR
    ;--------------------------------------------------------------;
    ; Delay Time tRCV Evaluation  (Time between points S12 and E12)   ;
    ;   Measure low level period on Port D bit 3 pin using a scope    ;
    ;--------------------------------------------------------------;
    ; Delay Evaluation: Point S12
    ;   bclr  3,PTD             ;Clear Port D bit 3

        aix   #$1               ;Step 13 - Wait for time tRCV
        sthx  FLASH_addr        ; 2 + 4 cycles = 6 cycles (2.4us)
                                ;Point the next address in FLASH_addr

    ; Delay Evaluation: Point E12
    ;   bset  3,PTD             ;Set Port D bit 3

        cli                     ;Clear the interrupt mask bit and return
ProgRow_End:
        rts
*************************************************************************
*****                Write to FLASH Control Register                *****
*************************************************************************
* This routine determines whether flcr1 or flcr2 should be written to and *
* which bit(s) in the flcr is set or clear based on the accumulator value *
* and the Flash address specified by FLASH_addr.                          *
* Initializations required:                                               *
*   - Set bit(s) in accumulator for making bit(s) in the flcr set or      *
*     clear                                                               *
*   - Load FLASH_addr to HX registers                                     *
* Values returned:                                                        *
*   - None                                                                *
*************************************************************************
WriteFLCR:
        cphx  #flash-1          ;If FLASH_addr is in Flash-1 array,
        bhs   Array1            ; jump to Array1

        eor   fl2cr             ;Write to fl2cr register
        sta   fl2cr

    ; Delay Evaluation: Point E10_2
```

```
    ;   bset  3,PTD             ;Set Port D bit 3

        bra   WriteFLCR_End
Array1:
        eor   fl1cr             ;Write to fl1cr register
        sta   fl1cr
WriteFLCR_End:

    ; Delay Evaluation: Point E10_1
    ;   bset  3,PTD             ;Set Port D bit 3

        rts
*************************************************************************
*****                        Delay Routine                          *****
*************************************************************************
* This routine generates unit millisecond delay depending on the value in *
* "times".  For example if times=1, the delay time is 1ms.                *
*   Delay = [{2 + (1 + 2 + 2 + 2 + 3) * 245 + 4 + 3} * times +4]          *
*                          / Bus Frequency                                *
*         = (2459 * times + 4) / 2.4578MHz                                *
* Initializations required:                                               *
*   - Set a value in "times"                                              *
* Values returned:                                                        *
*   - None                                                                *
*************************************************************************
ms_delay:
        lda   #!245             ;2 cyc.
ms_loop:
        deca                    ;1 cyc.
        and   #$FF              ;2 cyc.
        and   #$FF              ;2 cyc.
        and   #$FF              ;2 cyc.
        bne   ms_loop           ;3 cyc.
        dec   times             ;4 cyc.
        bne   ms_delay          ;3 cyc.
ms_delay_End:
        rts                     ;4 cyc.
```

Standard EEPROM Assembly Source Code Flowcharts

The main routine `EEPROM.mrt` initializes the device for erasing and programming operations. It sets up the clock source and timebase divider for the EEPROM memory and specifies the value and the location to be programmed. The routine then performs the EEPROM erase and program operations by calling `EEPROMroutine` twice.

The `EEPROMroutine` subroutine follows the flowcharts that are shown in **Figure 13** and **Figure 14** closely. Flowcharts are also included for the `WriteEECR` subroutine which is used to set or clear various bits in the EExCR registers and `ms_delay` subroutine which generates delays greater than 1 millisecond.

The flowcharts for `EEPROM.mrt`, `EEPROMroutine.mrt`, `WriteEECR` and `ms_delay` are **Figure 21**, **Figure 22**, **Figure 23**, and **Figure 24**, respectively.

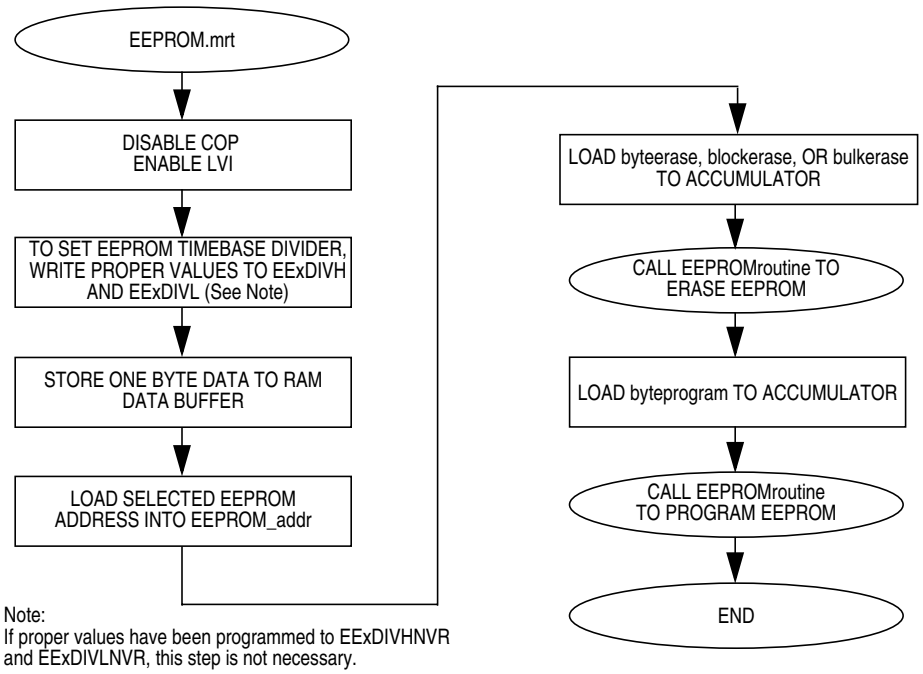


Figure 21. Standard EEPROM Main Routine Flowchart

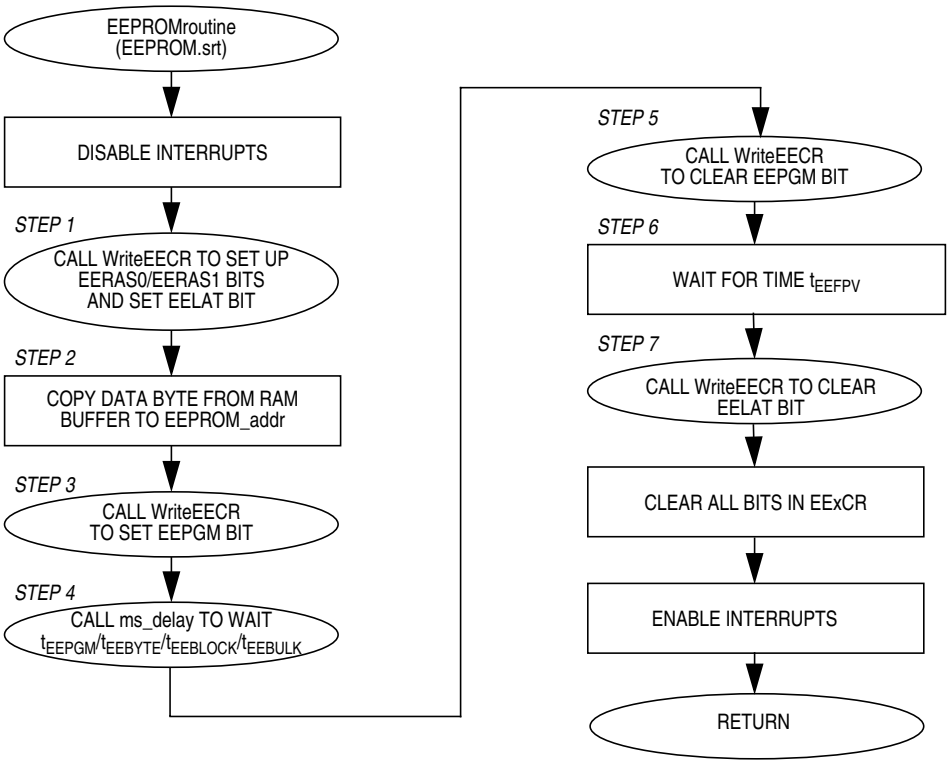


Figure 22. Subroutine EEPROMroutine Flowchart

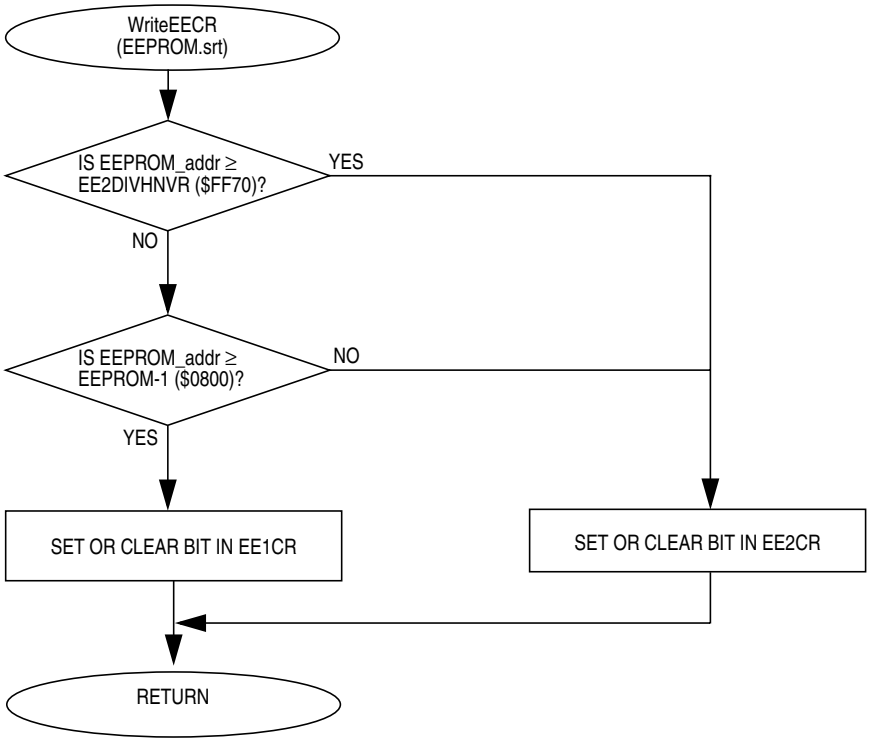


Figure 23. Subroutine WriteEECR Flowchart

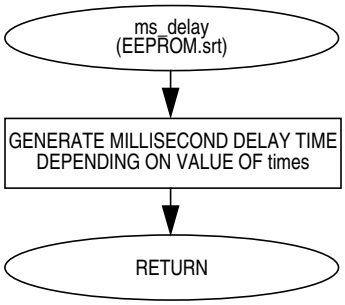


Figure 24. Subroutine ms_delay Flowchart

Standard EEPROM Assembly Source Code

```
*************************************************************************
*************************************************************************
*                                                                       *
*        EEPROM Programming and Erasing on the MC68HC908AS60A/AZ60A     *
*                                                                       *
*************************************************************************
* File Name: EEPROM.mrt                      Copyright (c)  *
*                                                                       *
* Current Revision: 1.0                                                 *
* Current Release Level: RP                                             *
* Current Revision Release Date: 6/21/2001                              *
*                                                                       *
* Current Release Written By: Adeela Gill and Kazue Kikuchi             *
*                        Freescale Applications Engineering - Austin, TX   *
*                                                                       *
* Assembled Under: CASM08Z (P&E Microcomputer Systems, Inc.)            *
*                      Ver.: 3.16                                       *
*                                                                       *
* Part Family Software Routine Works With: HC08                         *
*                                                                       *
* Routine Size (Bytes):       130                                       *
* Stack Space Used (Bytes):   4                                         *
* RAM Used (Bytes):           4                                         *
* Global Variables Used:      EEPROM_addr, data                         *
* Subroutine Called:          EEPROMroutine                             *
*                                                                       *
* Full Functional Description Of Routine Design:                        *
*       EEPROM.mrt is the main routine for the EEPROM programming and   *
*       erasing operations.  It demonstrates programming and erasing    *
*       algorithms for the MC68HC908AS60A and MC68HC908AZ60A.           *
*       Note: In this code, CGMXCLK is used as the EEPROM reference clock *
*             source and the frequency is 4.9152MHz                     *
*************************************************************************
*************************************************************************
*****                    Program Specific Equates                   *****
*************************************************************************
byteprogram. equ   %00000100    ;Select byte program and set EELAT bit
byteerase.   equ   %00001100    ;Select byte erase and set EELAT bit
blockerase.  equ   %00010100    ;Select block erase and set EELAT bit
bulkerase.   equ   %00011100    ;Select bulk erase and set EELAT bit
*************************************************************************
*****                       Include Files                           *****
*************************************************************************
NOLIST
$INCLUDE      "H908AS60A.frk"   ;Equates for all registers and
                                ; bits in the MC68HC908AS60A
        org   ram1
$INCLUDE      "EEPROM.var"      ;RAM variable definitions
LIST
*************************************************************************
*****                          Main Routine                         *****
*************************************************************************
        org   ram2
```

```
Start:
        mov   #$71,config-1     ;Turn off the COP, but leave the LVI on

        lda   #$80              ;For setting a constant timebase of 35us
        sta   EE1DIVH           ; write $80 and $AC to EExDIVH and
        sta   EE2DIVH           ; EExDIVL, respectively
        lda   #$AC              ; Note: If the EExDIVHNVR and EExDIVLNVR
        sta   EE1DIVL           ;   registers are programmed with proper
        sta   EE2DIVL           ;   values, this step is not necessary

        lda   #$AA              ;Write data $AA to RAM buffer
        sta   data
        ldhx  #$0676            ;Load EEPROM_addr with address of where
        sthx  EEPROM_addr       ; the byte should be erased and programmed

        lda   #byteerase.       ;Select Byte, Block or Bulk Erase
        jsr   EEPROMroutine     ;Erase selected erase size

        lda   #byteprogram.     ;Select Byte Program
        jsr   EEPROMroutine     ;Program one byte

        bra   *
*************************************************************************
*****               Subroutine Body Includes Section                *****
*************************************************************************
$INCLUDE      "EEPROM.srt"      ;EEPROM subroutines


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
*************************************************************************
*************************************************************************
*                                                                       *
*       EEPROM Programming and Erasing on the MC68HC908AS60A/AZ60A      *
*                                                                       *
*************************************************************************
* File Name: EEPROM.var                   Copyright (c)      *
*                                                                       *
* Current Revision: 1.0                                                 *
* Current Release Level: RP                                             *
* Current Revision Release Date: 6/21/2001                              *
*                                                                       *
* Current Release Written By: Adeela Gill and Kazue Kikuchi             *
*                        Freescale Applications Engineering - Austin, TX   *
*                                                                       *
* Assembled Under: CASM08Z (P&E Microcomputer Systems, Inc.)            *
*                       Ver.: 3.16                                      *
*                                                                       *
* Part Family Software Routine Works With: HC08                         *
*                                                                       *
* RAM Used (Bytes): 4                                                   *
*                                                                       *
* Description:                                                          *
*       RAM variable definitions for EEPROM.mrt.                        *
*************************************************************************
```

```
*****************************************************************************
*****                          RAM Variables                            *****
*****************************************************************************
EEPROM_addr     rmb     $2        ;16 bit Address of EEPROM memory to erase
                                  ; or program
data            rmb     $1        ;data byte that will be programmed
times           rmb     $1        ;1 byte in which the delay time will be
                                  ; specified


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
*****************************************************************************
*****************************************************************************
*                                                                           *
*     EEPROM Program and Erase Subroutine on the MC68HC908AS60A/AZ60A       *
*                                                                           *
*****************************************************************************
* File Name: EEPROM.srt                     Copyright (c)    *
*                                                                           *
* Current Revision: 1.1                                                     *
* Current Release Level: RP                                                 *
* Current Revision Release Date: 5/21/2001                                  *
*                                                                           *
* Current Release Written By: Adeela Gill and Kazue Kikuchi                 *
*                       Freescale Applications Engineering - Austin, TX   *
*                                                                           *
* Assembled Under: CASM08Z (P&E Microcomputer Systems, Inc.)                *
*                      Ver.: 3.16                                           *
*                                                                           *
* Part Family Software Routine Works With: HC08                             *
*                                                                           *
* Revision History:                                                         *
*   Rev. 1.1    Delay time evaluation code added                            *
*                                                                           *
*  Module Size (Bytes):          EEPROMroutine 45                         *
*                                WriteEECR     25                         *
*                                ms_delay      16                         *
*  Stack Space Used (Bytes):     EEPROMroutine 4                          *
*                                WriteEECR     0                          *
*                                ms_delay      0                          *
*  RAM Used (Bytes):             EEPROMroutine 4                          *
*                                WriteEECR     0                          *
*                                ms_delay      1                          *
*  Global Variable(s) Used:      EEPROMroutine EEPROM_addr, data          *
*                                WriteEECR     EEPROM_addr                *
*                                ms_delay      None                       *
*  Submodule(s) Called:          EEPROMroutine WriteEECR, ms_delay        *
*                                WriteEECR     None                       *
*                                ms_delay      None                       *
*  Calling Sequence:             JSR EEPROMroutine, JSR WriteEECR         *
*  Entry Label:                  EEPROMroutine, WriteEECR, ms_delay       *
*  Entry Conditions:             EEPROMroutine 2 bytes address defined at *
*                                               EEPROM_addr               *
*                                             1 programming byte located  *
*                                              at variable data (the      *
```

```
*                                          erasing operation is not  *
*                                          required)                 *
*                               WriteEECR    2 bytes address defined at *
*                                          EEPROM_addr               *
*                                          EECR (EEPROM Control      *
*                                          Register) bit definition  *
*                                          passed in accumulator     *
*                               ms_delay     Delay valiable passed in  *
*                                          times                     *
*  Number of Exit Points:          3                                 *
*     Exit Label:               EEPROMroutine  EEPROMroutine_End     *
*                               WriteEECR      WriteEECR_End         *
*                               ms_delay       ms_delay              *
*     Exit Conditions:          EEPROMroutine  None                  *
*                               WriteEECR      None                  *
*                               ms_delay       None                  *
*                                                                    *
* Full Functional Description Of Subroutine:                         *
*       EEPROM.srt consists of three subroutines:  EEPROMroutine,    *
*       WriteEECR and ms_delay. EEPROMroutine demonstrates EEPROM erasing *
*       and programming. WriteEECR allows the user to set and clear bits  *
*       in the EExCR registers. Since delay times must be met precisely   *
*       for successful EEPROM programming and erasing, additional         *
*       software was added to measure the delay times. This code is       *
*       included as comments throughout the file. It is recommended that  *
*       the user verify all delay times before using this software for    *
*       production.                                                  *
**********************************************************************
**********************************************************************
*****                EEPROM Program and Erase Subroutine              *****
**********************************************************************
EEPROMroutine:
    ;--------------------------------------------------------------;
    ; Delay Time Evaluation                                        ;
    ;  Initialize Port D bit 3 as output high                      ;
    ;   bset  3,PTD             ;Set Port D bit 3                  ;
    ;   bset  3,DDRD            ;Select output for Port D bit 3    ;
    ;--------------------------------------------------------------;
        sei                     ;Disable interrupts
        jsr   WriteEECR         ;Step 1 - Set up EERAS0, EERAS1 for a
                                ; desired operation and set EELAT
        lda   data              ;Step 2 - Write the desired data to
        ldhx  EEPROM_addr       ; the appropriate EEPROM location
        sta   ,X
        lda   #eepgm.           ;Step 3 - Set EEPGM bit
        jsr   WriteEECR
    ;--------------------------------------------------------------;
    ; Delay Time tEEBYTE/tEEBULK/tEEBLOCK/tEEPGM Evaluation  (Time ;
    ; between points S1 and E1)                                    ;
    ;   Measure low level period on Port D bit 3 pin using a scope ;
    ;--------------------------------------------------------------;
    ; Delay Evaluation: Point S1
    ;   bclr  3,PTD             ;Clear Port D bit 3

        lda   #!10              ;Step 4 - Wait for time
```

```
        sta   times             ; tEEBYTE/tEEBULK/tEEBLOCK/tEEPGM
        jsr   ms_delay          ; (all are 10 ms)

    ; Delay Evaluation: Point E1
    ;   bset  3,PTD             ;Set Port D bit 3

        lda   #eepgm.           ;Step 5 - Clear EEPGM bit
        jsr   WriteEECR
    ;---------------------------------------------------------------;
    ; Delay Time tEEFPV Evaluation  (Time between points S2 and E2)   ;
    ;   Measure low level period on Port D bit 3 pin using a scope    ;
    ;---------------------------------------------------------------;
    ; Delay Evaluation: Point S2
    ;   bclr  3,PTD             ;Clear Port D bit 3

        lda   #$52              ;Step 6 - Wait for time tEEFPV (101us)
        dbnza *                 ; 2 + (3 x 82) cycles = 248 cycles

    ; Delay Evaluation: Point E2
    ;   bset  3,PTD             ;Set Port D bit 3

        lda   #eelat.           ;Step 7 - Clear EELAT bit
        jsr   WriteEECR
        lda   #$00              ;Clear all bits in the EExCR
        sta   EE1CR
        sta   EE2CR
        cli                     ;Enable interrupts
EEPROMroutine_End:
        rts
*************************************************************************
*****                 Write to EEPROM Control Register              *****
*************************************************************************
* This routine determines whether EE1CR or EE2CR should be written to and *
* which bit(s) in the EExCR is set or clear based on the accumulator      *
* value and the EEPROM address specified by EEPROM_addr.                  *
* Initializations required:                                               *
*   - Set bit(s) in accumulator for making bit(s) in the EExCR set or     *
*     clear                                                               *
*   - Load EEPROM_addr to HX registers                                    *
* Values returned:                                                        *
*   - None                                                                *
*************************************************************************
WriteEECR:
        cphx  #EE2DIVHNVR       ;If address >= $FF70, branch to EEPROM2
        bhs   EEPROM2
        cphx  #eeprom-1         ;If address >= $0800, write to EEPROM1
        bhs   EEPROM1
EEPROM2:
        eor   EE2CR             ;Write to EE2CR register
        sta   EE2CR
        bra   WriteEECR_End
EEPROM1:
        eor   EE1CR             ;Write to EE1CR register
        sta   EE1CR
WriteEECR_End:
        rts
```

```
*****************************************************************************
*****                          Delay Routine                            *****
*****************************************************************************
* This routine generates unit millisecond delay depending on the value in *
* "times".  For example if times=1, the delay time is 1ms.                *
*   Delay = [{2 + (1 + 2 + 2 + 2 + 3) * 245 + 4 + 3} * times +4]          *
*                          / Bus Frequency                                *
*         = (2459 * times + 4) / 2.4578MHz                                *
* Initializations required:                                               *
*   - Set a value in "times"                                              *
* Values returned:                                                        *
*   - None                                                                *
*****************************************************************************
ms_delay:
        lda   #!245             ;2 cyc.
ms_loop:
        deca                    ;1 cyc.
        and   #$FF              ;2 cyc.
        and   #$FF              ;2 cyc.
        and   #$FF              ;2 cyc.
        bne   ms_loop           ;3 cyc.
        dec   times             ;4 cyc.
        bne   ms_delay          ;3 cyc.
ms_delay_End:
        rts                     ;4 cyc.
```

EEPROM AUTO Mode Source Code Flowcharts

The main routine `AutoEEPROM.mrt` initializes the device for erasing and programming operations. It sets up the clock source and timebase divider for the EEPROM memory and specifies the value and the location to be programmed. The routine then performs the EEPROM erase and program operations by calling `AUTOroutine` twice.

The `AUTOroutine` subroutine follow the flowcharts shown in **Figure 13** and **Figure 14** closely. Flowcharts are also included for the `WriteEECR` subroutine which is used to set or clear various bits in the EExCR registers.

The flowcharts for `AutoEEPROM.mrt`, `AUTOroutine.mrt`, and `WriteEECR` are **Figure 25**, **Figure 26**, and **Figure 27**, respectively.

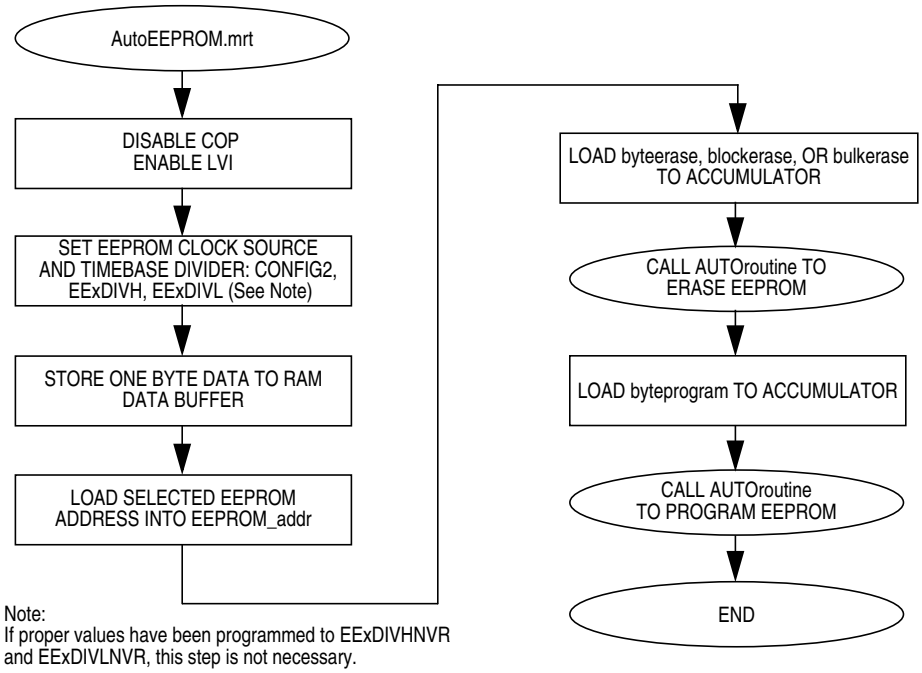


Figure 25. EEPROM AUTO Mode Main Routine

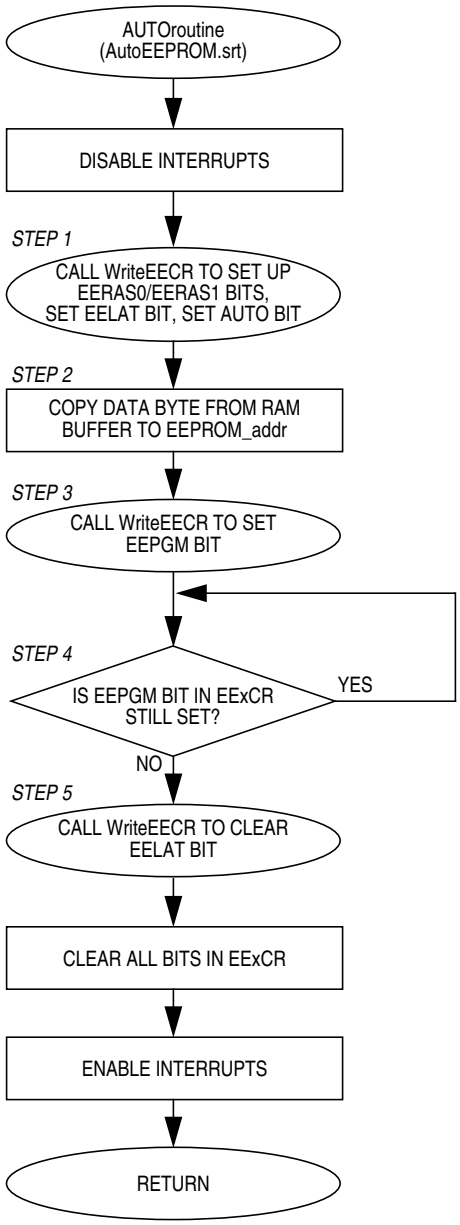


Figure 26. Subroutine AutoEEPROM Flowchart

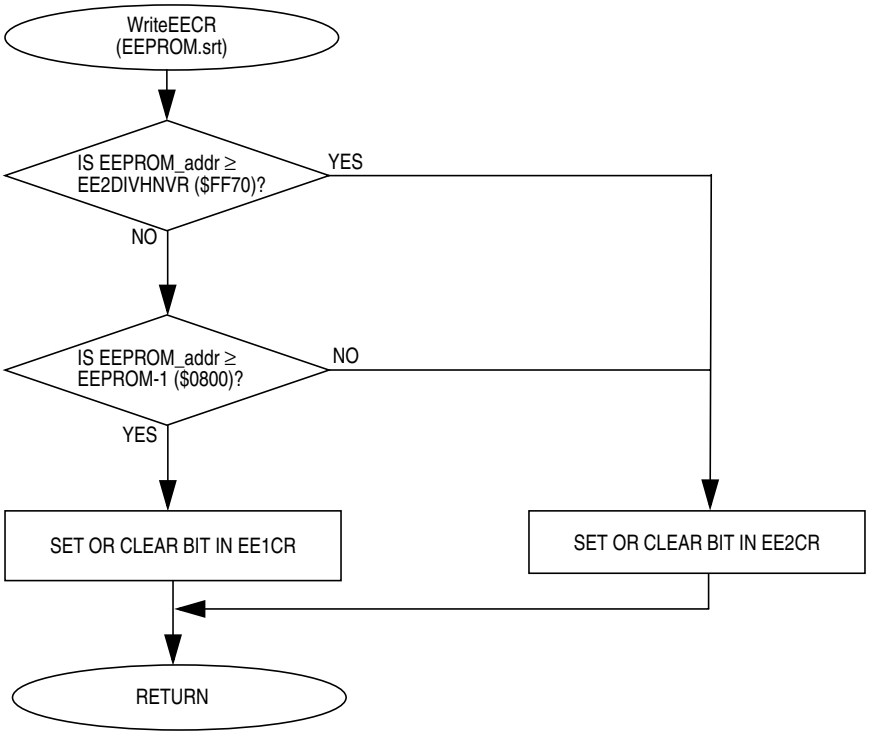


Figure 27. Subroutine WriteEECR Flowchart

EEPROM AUTO Mode Assembly Source Code

```
*****************************************************************************
*****************************************************************************
*                                                                           *
*     EEPROM AUTO Programming and Erasing on the MC68HC908AS60A/AZ60A       *
*                                                                           *
*****************************************************************************
* File Name: AutoEEPROM.mrt                  Copyright (c)   *
*                                                                           *
* Current Revision: 1.1                                                     *
* Current Release Level: RP                                                 *
* Current Revision Release Date: 6/21/2001                                  *
*                                                                           *
* Current Release Written By: Adeela Gill and Kazue Kikuchi                 *
*                        Freescale Applications Engineering - Austin, TX   *
*                                                                           *
* Assembled Under: CASM08Z (P&E Microcomputer Systems, Inc.)                *
*                      Ver.: 3.16                                           *
*                                                                           *
* Part Family Software Routine Works With: HC08                             *
*                                                                           *
* Revision History:                                                         *
*   Rev. 1.1    EELAT and AUTO bits are set at the same timing              *
*                                                                           *
* Routine Size (Bytes):       113                                           *
* Stack Space Used (Bytes):   4                                             *
* RAM Used (Bytes):           4                                             *
* Global Variables Used:      EEPROM_addr, data                             *
* Subroutine Called:          AutoRoutine                                   *
*                                                                           *
* Full Functional Description Of Routine Design:                            *
*       AutoEEPROM.mrt is the main routine for the EEPROM programming and   *
*       erasing operations using the AUTO mode.  It demonstrates AUTO       *
*       programming and erasing for the MC68HC908AS60A and MC68HC908AZ60A.  *
*       Note: In this code, the internal bus clock is used as the EEPROM    *
*             reference clock source and the frequency is 2.4576MHz.        *
*****************************************************************************
*****************************************************************************
*****                    Program Specific Equates                       *****
*****************************************************************************
auto_byteprogram. equ   %00000110    ;Select byte program, and set EELAT
                                     ; and AUTO bits
auto_byteerase.   equ   %00001110    ;Select byte erase, and set EELAT
                                     ; and AUTO bits
auto_blockerase.  equ   %00010110    ;Select block erase, and set EELAT
                                     ; and AUTO bits
auto_bulkerase.   equ   %00011110    ;Select bulk erase, and set EELAT
                                     ; and AUTO bits
*****************************************************************************
*****                        Include Files                              *****
*****************************************************************************
NOLIST
$INCLUDE      "H908AS60A.frk"   ;Equates for all registers and
                                ; bits in the MC68HC908AS60A
```

```
        org   ram1
$INCLUDE      "AutoEEPROM.var"  ;RAM variable definitions
LIST
*****************************************************************************
*****                            Main Routine                           *****
*****************************************************************************
        org   ram2
Start:
        mov   #$71,config-1     ;Turn off the COP, but leave the LVI on

        lda   #$98              ;Select bus clock as reference clock
        sta   config-2          ; source

        lda   #$80              ;For setting a constant timebase of 35us
        sta   EE1DIVH           ; write $80 AND $56 to EExDIVH and
        sta   EE2DIVH           ; EExDIVL, respectively
        lda   #$56              ; Note: If the EExDIVHNVR and EExDIVLNVR
        sta   EE1DIVL           ;   registers are programmed with proper
        sta   EE2DIVL           ;   values, this step is not necessary

        lda   #$55              ;Write data $55 to RAM buffer
        sta   data

        ldhx  #$0634            ;Load EEPROM_addr with address of where
        sthx  EEPROM_addr       ; the byte should be erased and programmed

        lda   #auto_bulkerase.  ;Select Bulk, Block or Byte Erase
        jsr   AutoRoutine       ;Erase the selected EEPROM size using AUTO
                                ; Mode

        lda   #auto_byteprogram.
                                ;Select Byte Program
        jsr   AutoRoutine       ;Program one byte using AUTO Mode

        bra   *
*****************************************************************************
*****               Subroutine Body Includes Section                    *****
*****************************************************************************
$INCLUDE      "AutoEEPROM.srt"      ;Auto EEPROM subroutines


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
*******************************************************************************
*****************************************************************************
*                                                                           *
*     EEPROM AUTO Programming and Erasing on the MC68HC908AS60A/AZ60A       *
*                                                                           *
*****************************************************************************
* File Name: AutoEEPROM.var              Copyright (c)     *
*                                                                           *
* Current Revision: 1.0                                                     *
* Current Release Level: RP                                                 *
* Current Revision Release Date: 6/21/2000                                  *
*                                                                           *
* Current Release Written By: Adeela Gill and Kazue Kikuchi                 *
```

```
*                       Freescale Applications Engineering - Austin, TX   *
*                                                                         *
* Assembled Under: CASM08Z (P&E Microcomputer Systems, Inc.)              *
*                       Ver.: 3.16                                        *
*                                                                         *
* Part Family Software Routine Works With: HC08                           *
*                                                                         *
* RAM Used (Bytes): 3                                                     *
*                                                                         *
* Description:                                                            *
*       RAM variable definitions for AutoEEPROM.mrt.                      *
***************************************************************************
***************************************************************************
*****                           RAM Variables                         *****
***************************************************************************
EEPROM_addr     rmb     $2        ;16 bit Address of EEPROM memory to erase
                                  ; or program
data            rmb     $1        ;one data byte that will be programmed


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
***************************************************************************
***************************************************************************
*                                                                         *
*  EEPROM AUTO Program and Erase Subroutine on the MC68HC908AS60A/AZ60A   *
*                                                                         *
***************************************************************************
* File Name: AutoEEPROM.srt                Copyright (c)    *
*                                                                         *
* Current Revision: 1.1                                                   *
* Current Release Level: RP                                               *
* Current Revision Release Date: 6/21/2001                                *
*                                                                         *
* Current Release Written By: Adeela Gill and Kazue Kikuchi               *
*                       Freescale Applications Engineering - Austin, TX   *
*                                                                         *
* Assembled Under: CASM08Z (P&E Microcomputer Systems, Inc.)              *
*                       Ver.: 3.16                                        *
*                                                                         *
* Part Family Software Routine Works With: HC08                           *
*                                                                         *
* Revision History:                                                       *
*   Rev. 1.1    AUTO bit is set in Step 1                                 *
*                                                                         *
*  Module Size (Bytes):          AutoRoutine    43                       *
*                                WriteEECR      25                       *
*  Stack Space Used (Bytes):     AutoRoutine    2                        *
*                                WriteEECR      0                        *
*  RAM Used (Bytes):             AutoRoutine    3                        *
*                                WriteEECR      0                        *
*  Global Variable(s) Used:      AutoRoutine    EEPROM_addr, data        *
*                                WriteEECR      EEPROM_addr              *
*  Submodule(s) Called:          AutoRoutine    WriteEECR                *
*                                WriteFLCR      None                     *
*  Calling Sequence:             JSR AutoRoutine, JSR writeEECR          *
```

```
*  Entry Label:                 AutoRoutine, WriteEECR                  *
*  Entry Conditions:            AutoRoutine   2 bytes address defined at *
*                                              EEPROM_addr               *
*                                             1 programming byte located *
*                                              at variable data (the     *
*                                              erasing operation is not  *
*                                              required)                 *
*                               WriteEECR     2 bytes address defined at *
*                                              EEPROM_addr               *
*                                             EECR (EEPROM Control       *
*                                              Register) bit definition  *
*                                              passed in accumulator     *
*  Number of Exit Points:          2                                     *
*     Exit Label:               AutoRoutine    AutoRoutine_End           *
*                               WriteEECR      WriteEECR_End             *
*     Exit Conditions:          AutoRoutine    None                      *
*                               WriteEECR      None                      *
*                                                                        *
* Full Functional Description Of Subroutine:                            *
*       AutoEEPROM.srt contains one primary subroutine called AutoRoutine.*
*       This demonstrates EEPROM erasing and programming in the AUTO mode.*
*       The routine also calls another subroutine WriteEECR.             *
**************************************************************************
**************************************************************************
*****             EEPROM AUTO Program and Erase Subroutine            *****
**************************************************************************
AutoRoutine:
        sei                     ;Disable interrupts
        jsr   WriteEECR         ;Step 1 - Set up EERAS0, EERAS1 for a
                                ; desired operation, and set EELAT and
                                ; AUTO
        lda   data              ;Step 2 - For programming, copy one byte
        ldhx  EEPROM_addr       ; data from the RAM buffer to the
        sta   ,X                ; appropriate EEPROM location

        lda   #eepgm.           ;Step 3 - Set EEPGM bit
        jsr   WriteEECR
Clear_EEPGM1:
        lda   EE1CR             ;Step 4 - Wait until EEPGM bit is cleared
        and   #$01              ; Checks included for both EEPGM registers
        bne   Clear_EEPGM1
Clear_EEPGM2:                   ; since the programmed byte could be in
        lda   EE2CR             ; either array
        and   #$01
        bne   Clear_EEPGM2

        lda   #eelat.           ;Step 5 - Clear EELAT bit
        jsr   WriteEECR

        lda   #$00              ;Clear all bits in the EExCR
        sta   EE1CR
        sta   EE2CR
        cli                     ;Enable interrupts
AutoRoutine_End:
        rts
```

```
*****************************************************************************
*****                  Write to EEPROM Control Register                 *****
*****************************************************************************
* This routine determines whether EE1CR or EE2CR should be written to and *
* which bit(s) in the EExCR is set or clear based on the accumulator      *
* value and the EEPROM address specified by EEPROM_addr.                  *
* Initializations required:                                               *
*   - Set bit(s) in accumulator for making bit(s) in the EExCR set or     *
*     clear                                                               *
*   - Load EEPROM_addr to HX registers                                    *
* Values returned:                                                        *
*   - None                                                                *
*****************************************************************************
WriteEECR:
        cphx  #EE2DIVHNVR       ;If address >= $FF70, branch to EEPROM2
        bhs   EEPROM2

        cphx  #eeprom-1         ;If address >= $0800, write to EEPROM1
        bhs   EEPROM1

EEPROM2:
        eor   EE2CR             ;Write to EE2CR register
        sta   EE2CR
        bra   WriteEECR_End
EEPROM1:
        eor   EE1CR             ;Write to EE1CR register
        sta   EE1CR

WriteEECR_End:
        rts
```