AN1827

Programming and Erasing FLASH Memory on the MC68HC908AS60

By Kim Keating, Adeela Gill, and Kazue Kikuchi
Body Electronics and Occupant Safety
Matt Rutledge
Non-Volatile Memory Technology Center

Introduction

Freescale has released an innovative type of FLASH non-volatile memory (NVM) for its 8-bit M68HC08 Family of microcontrollers. This FLASH technology allows in-circuit reprogrammability over the entire automotive specification range.

In-circuit reprogrammability offers these advantages:

- In-system code revision
- EPROM (erasable programmable read-only memory) replacement as a reusable code development platform
- Quick time to market with one chip for code development and production
- No obsolete inventory as with ROM parts
- Allows for last-minute code changes without waiting for new ROM code lots

This application note explains how to use the FLASH on the MC68HC908AS60 and provides example software for program and erase operations. The reprogramming algorithms are written in both M68HC08 assembly code and in C code.

This code is available for download from Motorola's Semiconductor Product Sector's Web site at http://mot-sps.com.

The FLASH topics covered in this application note include:

- Features
- Implementation on Motorola's M68HC08 microcontrollers
- Functional description
- Control and block protect registers
- Charge pump
- Block protection
- Erase operation
- Page program/margin read algorithm
- Frequently asked questions
- Hardware schematic
- Assembly source code
- C source code

Features

The benefits of FLASH on the MC68HC908AS60 include:

- *Single V_{DD} power supply is utilized for program/erase.*
  This feature simplifies program and erase with respect to EPROM (no high voltage power supply or UV (ultraviolet) oven required), reduces program and erase cycle time, and enables in-circuit reprogrammability.

- *The FLASH manufacturing process is fully compatible with the EEPROM (electrically erasable, programmable read-only memory) process.*
  This process compatibility allows the functionality of both FLASH and EEPROM non-volatile memories on the same chip.
- *Meets automotive specifications*
  Unlike many competing microcontrollers with FLASH, this FLASH can operate and meet reliability requirements for the automotive space. The FLASH on the MC68HC908AS60 will read, program, and erase over the –40°C to 125°C temperature range. The specified program/erase endurance and data retention lifetime are valid over the entire temperature range.
- *Multiple arrays*
  Multiple arrays on the MC68HC908AS60 allow code execution out of one array while programming or erasing the other array.
- *Smart programming algorithm*
  Use of the smart programming algorithm ensures minimum program time while still guaranteeing automotive environment operation and data retention.

Implementation on Motorola's M68HC08 Microcontrollers

The specific FLASH technology found on the MC68HC908AS60 is known as FLASH 2TS, in reference to its 2-transistor source-select bit cell. FLASH 2TS is commonly found on Motorola's M68HC08 Family of microcontrollers, but it is not the exclusive FLASH technology for the HC08 core. The FLASH 2TS technology discussed in this application note is referred to generically as FLASH.

This FLASH technology is available in array sizes between 2 Kbytes and 32 Kbytes. For parts requiring more than 32 Kbytes, multiple arrays of any size between 2 Kbytes and 32 Kbytes, with 2-Kbyte boundaries, can be placed on a chip. Typically, only one charge pump is used on parts with multiple arrays. This constrains program or erase operations to one array at a time, but conserves die area.

The FLASH bit cell consists of two transistors in series, referred to as the select-gate and control-gate transistors. The floating gate is associated with the control gate transistor and stores charges which represent the two different data states of the memory. The high threshold condition of the bit cell is the erased state and the low threshold condition is the programmed state. The select gate prevents bit cell leakage of unselected wordlines during read operations when the floating gate is programmed.

Although the size and shape of the array is mostly transparent to the user, it does help when determining the "cared addresses" during the erase algorithm. The term and function of "cared addresses" are explained in the **Erase Operation** of this application note. For now, it is important to know that the cared addresses determine exactly which block will erase during the erase operation.

More importantly, the size and shape of the memory array may alter the size of the programming page. The term page refers to the number of consecutive bytes that are programmed during a page program/margin read operation. On larger memory arrays, like the two arrays found on the MC68HC908AS60, one page equals eight bytes. As the array is scaled down, the page is proportionally scaled to either four, two, or one byte(s). This will affect the programming algorithm that appears later in this application note.

The only other obvious difference in implementation of the FLASH array is the size of the blocks that can be protected against an undesired program or erase operation. Again, this depends on the size of the memory on the microcontroller. Check the appropriate documentation for each specific microcontroller to determine the size of the memory array, the page program size, and the erase block sizes.

Functional Description

The FLASH memory on the MC68HC908AS60 physically consists of two independent arrays with two bytes of block protection and additional bytes of user vectors. An erased bit reads as a logic 0 and a programmed bit reads as a logic 1. 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 pages within rows. There are eight pages of memory per row with eight bytes per page. The minimum erase block size is a single row, 64 bytes. Programming is performed on a per-page basis, eight 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 MC68HC908AS60 consists of:

- $0450–$05FF, FLASH-2 array, 432 bytes
- $0E00–$7FFF, FLASH-2 array, 29,184 bytes
- $8000–$FDFF, FLASH-1 array, 32,256 bytes
- $FE0B, FLASH-1 control register, FLCR1
- $FE11, FLASH-2 control register, FLCR2
- $FF80, FLASH-1 block protect register, FLBPR1
- $FF81, FLASH-2 block protect register, FLBPR2
- $FFDA–$FFFF, FLASH-1 vector space, 38 bytes

To program the FLASH, each page must be erased before it is programmed. The erase block sizes are found in **Erase Operation**.

The four 64-byte row address boundaries for the MC68HC908AS60 are:

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

When programming the FLASH, exact program time must be used to program a page. 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 an iterative program and margin read technique known as the smart programming algorithm. The smart programming algorithm is required whenever programming the FLASH. See **Page Program/Margin Read Algorithm**.

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.

Control and Block Protect Registers

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

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	FDIV1	FDIV0	BLK1	BLK0	HVEN	MARGIN	ERASE	PGM
Reset:	0	0	0	0	0	0	0	0

Figure 1. FLASH Control Register (FLCR)

There are two FLASH control registers, FLCR1 and FLCR2, for FLASH-1 and FLASH-2 arrays, respectively.

- $FE0B — FLASH-1 control register (FLCR1)
- $FE11 — FLASH-2 control register (FLCR2)

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

FDIV1 — Frequency Divide Control Bit

This read/write bit together with FDIV0 selects the factor by which the charge pump clock is divided from the bus clock. See **Charge Pump**.

FDIV0 — Frequency Divide Control Bit

This read/write bit together with FDIV1 selects the factor by which the charge pump clock is divided from the bus clock. See **Charge Pump**.

BLK1 — Block Erase Control Bit

This read/write bit together with BLK0 allows erasing of blocks of varying sizes. See **Erase Operation** for a description of available block sizes.

BLK0 — Block Erase Control Bit

This read/write bit together with BLK1 allows erasing of blocks of varying sizes. See **Erase Operation** for a description of available block sizes.

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 erase or page program/margin read is followed.

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

MARGIN — Margin Read Control Bit

This read/write bit configures the memory for the margin read operation. MARGIN cannot be set if HVEN = 1. MARGIN will automatically clear (MARGIN = 0) if asserted when HVEN = 1.

1 = Margin read operation selected
0 = Margin read operation unselected

ERASE — Erase Control Bit

This read/write bit configures the memory for the 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 the 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:	DC	DC	DC	DC	BPR3	BPR2	BPR1	BPR0
Reset:	NV	NV	NV	NV	NV	NV	NV	NV

NV — Non-volatile, 1 if programmed, 0 if erased
DC — Don't care

Figure 2. FLASH Block Protect Register (FLBPR)

There are two FLASH block protect registers, FLBPR1 and FLBPR2, for FLASH-1 and FLASH-2 arrays, respectively.

- $FF80 — FLASH-1 block protect register (FLBPR1)
- $FF81 — FLASH-2 block protect register (FLBPR2)

BPR3 — Block Protect Register Bit 3

This bit protects the memory contents in the address range:
FLASH-1 $C000 to $FFFF or FLASH-2 $4000 to $7FFF.

1 = Address range protected from erase or program
0 = Address range open to erase or program

BPR2 — Block Protect Register Bit 2

This bit protects the memory contents in the address range:
FLASH-1 $A000 to $FFFF or FLASH-2 $2000 to $7FFF.
1 = Address range protected from erase or program
0 = Address range open to erase or program

BPR1 — Block Protect Register Bit 1

This bit protects the memory contents in the address range:
FLASH-1 $9000 to $FFFF or FLASH-2 $1000 to $7FFF.
1 = Address range protected from erase or program
0 = Address range open to erase or program

BPR0 — Block Protect Register Bit 0

This bit protects the memory contents in the address range:
FLASH-1 $8000 to $FFFF or FLASH-2 $0450 to $7FFF.
1 = Address range protected from erase or program
0 = Address range open to erase or program

Charge Pump

The internal charge pump is required for program, margin read, and erase operations of the FLASH.

The charge pump is a dynamic circuit that uses a specific clocking sequence of capacitors and switches to generate voltages higher in magnitude than V_{DD}. This charge pump design requires a clock frequency range between 1.8 MHz and 2.5 MHz to operate the FLASH correctly. The charge pump clock is derived from the bus clock. The FDIV1 and FDIV0 bits in the FLASH control register are able to divide the internal bus clock by 1, 2, or 4 to generate the charge pump clock. These divide ratios allow enough tolerance for several commonly available crystal frequencies.

See **Table 1** for common divide ratios based upon internal bus frequency.

NOTE: *When FLASH memory is programmed/erased with the PLL on or in monitor mode, bus frequency is not always the same as the external clock frequency divided by four. Since the charge pump frequency is derived from the bus frequency, confirm the bus frequency being used.*

Table 1. Bus Frequency Divide Ratios for Charge Pump Clock

f_{Bus} (MHz)	FDIV1	FDIV0	Division	f_{Pump} (MHz)
2.000	0	0	1	2.000
2.4576	0	0	1	2.4576
4.000	0	1	2	2.000
4.9152	0	1	2	2.4576
8.000	1	1	4	2.000
8.400	1	1	4	2.100

NOTE: *If the charge pump frequency is not between 1.8 MHz and 2.5 MHz, Freescale does not guarantee the operation, electrical, or reliability specifications of the FLASH.*

The HVEN bit in the FLASH control register enables the charge pump to generate high voltages for program and erase modes. The charge pump also generates a regulated voltage for the margin read mode in the smart programming algorithm. During programming, the HVEN bit should be asserted only for 1 ms to 1.2 ms at a time. (See **Figure 8**.) Asserting HVEN for longer than 1.2 ms at a time risks program disturb, where an erased bit on the same row becomes unintentionally programmed. Program disturb is a common soft fault and can be recovered by erasing the row and reprogramming using the smart programming algorithm.

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 non-volatile bytes within the FLASH-1 array, with one byte per FLASH array. Once the block protect bits are set in the FLBPR registers, the defined address ranges are protected from being programmed or erased. See **Control and Block Protect Registers** for a description of address ranges.

The FLBPR register itself can be erased or programmed only with an external voltage V_{HI} on the IRQ pin. V_{HI} is defined as a voltage between V_{DD} + 2 V and V_{DD} + 4 V. Use of the block protect register is an additional measure to prevent inadvertent programming or erasing of FLASH contents in an application.

NOTE: *To implement in-system program or erase for a protected area of FLASH, a high voltage signal must be routed to the IRQ pin.*

Erase Operation

To erase a FLASH array, follow this 9-step procedure. **Figure 3** shows a flowchart of this procedure.

1. Set ERASE = 1, and set the BLK bits and FDIV bits.

   ERASE = 1 configures the FLASH memory for an erase operation. The BLK bits determine the erase block size: whole array, half array, 512 bytes or 64 bytes. The FDIV bits determine the charge pump frequency. The frequency should be selected within the range between 1.8 MHz and 2.5 MHz. Refer to **Charge Pump**.

2. Read the FLASH block protect register.

   The block protect registers 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 block address range desired.

   The "cared" bits for the FLASH address are latched and used to determine the address range that will be erased. The details are discussed later in this section.

4. Set HVEN = 1.

   a. Internal high voltage is applied for erasing.

5. Wait for a time, t_{Erase}.

   t_{Erase} is the block erase time.

6. Set HVEN = 0.

   Internal high voltage is disabled.

7. Wait for a time, t_{Kill}.

   This allows the high voltage to be discharged completely.

8. Set ERASE = 0.

   Disable the erase operation.

9. Wait for a time, t_{HVD}.

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

NOTE: *If bulk erase is attempted on a FLASH array where either part or all of the array is block protected, then none of the FLASH memory in that array is erased.*

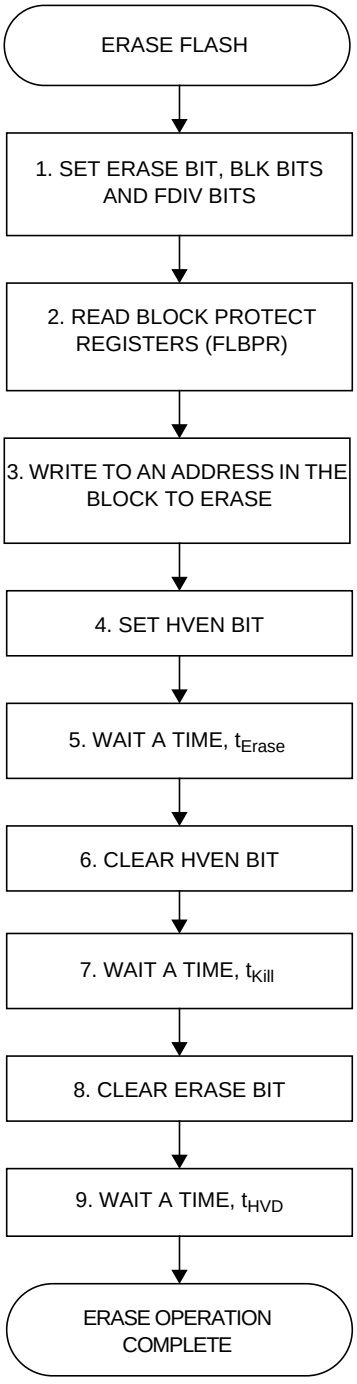


Figure 3. FLASH Erase Operation Flowchart

Although the overall procedure is relatively simple, step 3 could use some clarification. Since the specified address is any address within the block to erase, the microcontroller must somehow know exactly which memory range to erase. This is where the "cared address" becomes important.

In step 2, the size of the block to erase is set by writing to the BLK bits in the FLCR. **Table 2** shows the various block sizes which can be erased in one erase operation.

Table 2. Erase Block Sizes

BLK1	BLK0	Block Size	Cared Addresses
0	0	Full array: 32 Kbytes	A15
0	1	One-half array: 16 Kbytes	A15–A14
1	0	Eight rows: 512 bytes	A15–A9
1	1	Single row: 64 bytes	A15–A6

When an address is specified in step 3, certain address lines are latched pertaining to this block size, and they establish the start and end addresses of the block. The larger the erase block, the smaller the size of the cared address.

For example, if the BLK bits are set such that the erase block size is a single row (BLK0 = BLK1 = 1), and the address $9AF0 is specified in step 3, then bits 15–6 of $9AF0 are the "cared addresses." Therefore, the values of these address bits are fixed.

Hex	9				A				F				0			
Bit No.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Binary	1	0	0	1	1	0	1	0	1	1	1	1	0	0	0	0

Cared Addresses

Figure 4. Cared Address Example $9AF0

As a result, the beginning address of the block which will be erased is:

Hex	9				A				C				0			
Bit No.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Binary	1	0	0	1	1	0	1	0	1	1	0	0	0	0	0	0

Cared Addresses

Figure 5. Erase Beginning Address for Example

The end address of the block which will be erased is:

Hex	9				A				F				F			
Bit No.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Binary	1	0	0	1	1	0	1	0	1	1	1	1	1	1	1	1

Cared Addresses

Figure 6. Erase End Address for Example

This results in erasing 64 bytes from address $9AC0 to $9AFF.

NOTE: *All memory arrays are shaped differently, so one row may not equal 64 bytes like on the MC68HC908AS60. Refer to the appropriate documentation for the pertinent device and apply these same principles.*

Page Program/Margin Read Algorithm

In the MC68HC908AS60, programming of the FLASH memory is done on a page-by-page basis. A page consists of eight bytes, from addresses $XXX0 to $XXX7 or from $XXX8 to $XXXF. Therefore, the addresses of the first byte in a page must be $XXX0 or $XXX8.

This FLASH memory requires the smart programming algorithm. The smart programming algorithm is defined as an iterative program and margin read sequence. Every page program operation is followed by a margin read until the data is programmed successfully. The margin read step of the smart programming algorithm is used to ensure programmed bits are programmed to sufficient margin for data retention over the device's lifetime.

The smart programming algorithm steps are shown here and in the **Figure 7** flowchart.

1. Initialize attempt counter.

   The sequence will be attempted until the count reaches fls_{Pulses}.

2. Set PGM = 1 and set FDIV bits.

   PGM = 1 configures the FLASH memory for a program operation and enables the latching of the address and data for programming. The FDIV bits determine the charge pump frequency. The frequency should be selected within the range between 1.8 MHz and 2.5 MHz. Refer to **Charge Pump**.

3. Read the FLASH block protect register.

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

4. Write data to the page being programmed (typically 8 bytes).

   This requires separate write operations for each byte and the addresses of the page must be $XXX0 to $XXX7, or $XXX8 to $XXXF.

5. Set HVEN = 1.

   Internal high voltage is applied for programming.

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

   t_{Step} is the time high voltage is applied for every program pulse.

7. Set HVEN = 0.

   Internal high voltage is disabled.

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

   Wait for programming voltages to dissipate before margin reading.

9. Set MARGIN = 1.

   This configures the FLASH memory for margin read operation.

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

    Time to discharge the margin read voltage.

11. Set PGM = 0.

    This step disables the programming operation.

12. Wait for a time, t_{HVD}.

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

13. Read programmed data (margin read process).

    This requires separate read operations for each byte.

14. Compare margin read data with data written in Step 4.

    This requires separate read operations for each byte.

15. Clear the MARGIN bit.

    Disable the margin read operation.

16. Increment attempt counter since programming was not successful.

17. If any byte of programmed data does not match the margin read data, then there are two options. If the count is less than the maximum (fls_{Pulses}(); return to step 2 to repeat programming of the same page. If the attempt to program count has reached fls_{Pulses}, the programming operation has failed.

Notes:
1) This algorithm is mandatory for programming the FLASH.
2) This page program algorithm assumes the page(s) to be programmed are initially erased.

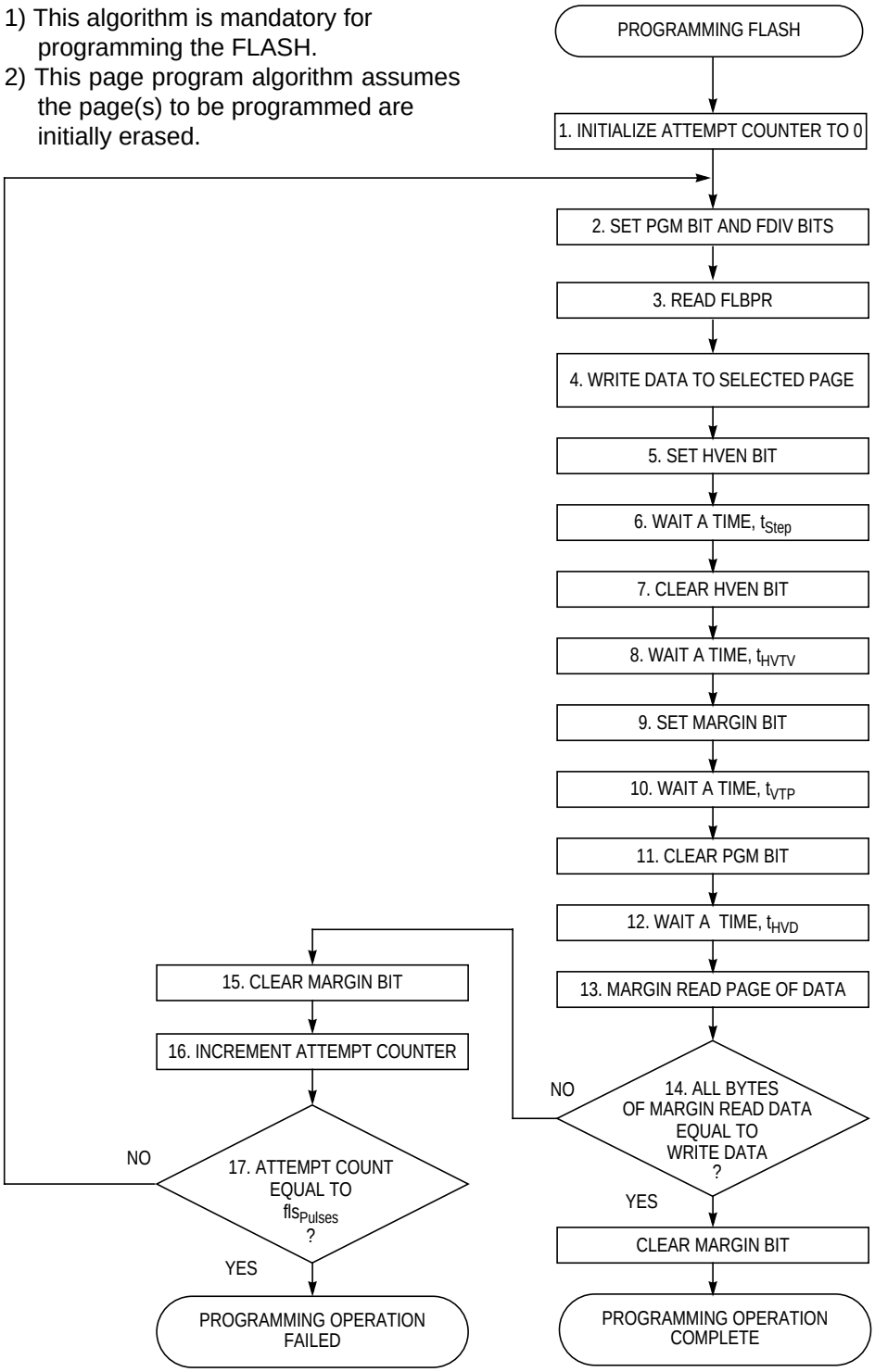


Figure 7. FLASH Smart Programming Algorithm Flowchart

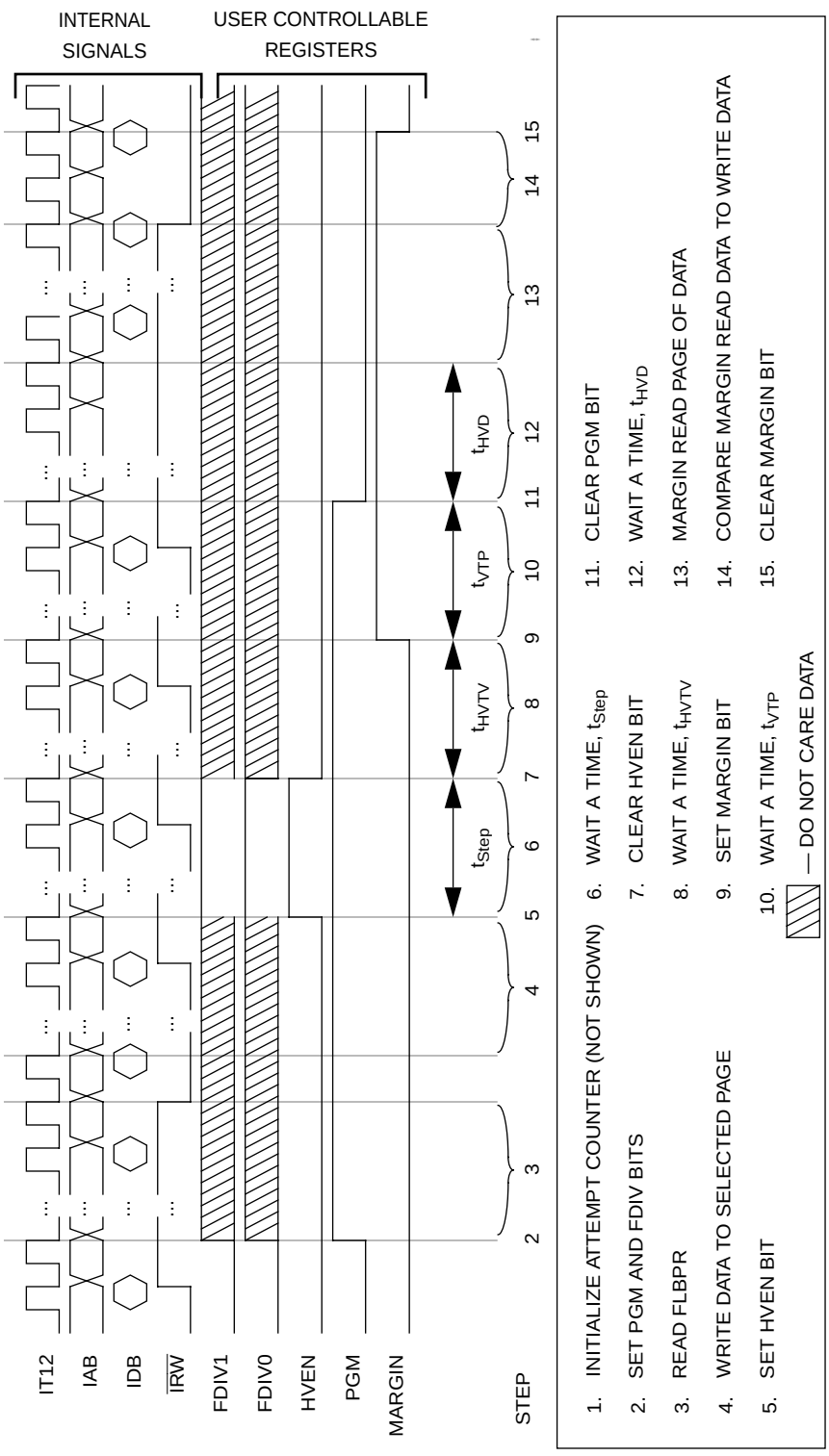


Figure 8. Timing Diagram of One Page Program-Margin Read Step in the Smart Programming Algorithm

The smart programming algorithm ensures programming data retention and minimum program time and reduces the possibility of program disturb. The margin read operation imposes a more stringent read condition on the bit cell so that long-term data retention is ensured. The operation is the same as the ordinary read operation except the margin bit is set (MARGIN = 1). However, when the margin read operation is executed, data is read automatically with seven additional bus cycles per byte. This additional settling time allows sensing of the lower bit cell current.

The smart programming algorithm also uses multiple short program pulses instead of using one long program pulse. Therefore, whenever the margin read is successful, the page programming is completed even if the program pulses do not reach the maximum. If the program pulses reach the maximum, it means that programming operation has failed.

NOTE: *When the COP is enabled, the seven additional cycles of the margin read operation must be considered. Since the COP counter continues to run during the additional cycles, the additional cycles need to be added to the COP feed loop.*

NOTE: *When the block protect bits in the FLASH block protect register are set, a portion of the memory will be locked so that no further erase or program operation may be performed. However, when high voltage is applied on the IRQ pin, the whole FLASH memory is unprotected. The details are described in the* ***Control and Block Protect Registers****.*

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:

- *Did you use the smart programming algorithm in your programming code?*
  The smart programming algorithm ensures that the FLASH is programmed for sufficient data retention and in minimum program time. Furthermore, not following this algorithm can lead to overprogramming, which risks program disturb. The use of the smart programming algorithm is highly recommended. (Refer to **Page Program/Margin Read Algorithm**.)
- *Is each step of the smart programming algorithm (or erase algorithm) 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-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 set such that the memory to be erased or programmed is unprotected. The only way to override the block protect bits is to apply voltage V_{HI} on IRQ during the erase and program algorithms. (Refer to **Block Protection**.)
- *Are delay times such as t_{Step}, t_{Erase}, t_{HVD}, 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 the wrong delay times are used.

- *Is the correct FLASH register being written to enable erase or program?*
  The MC68HC908AS60 has two FLASH arrays with two separate sets of control and block protect registers. Make sure the appropriate register is being addressed. Refer to **Control and Block Protect Registers**.
- *Is the maximum pulse value (fls_{Pulses}) set correctly according to specification?*
  Usually, the FLASH will program in fewer than the maximum specified number of program pulses allowed. However, the specification is chosen to ensure that even the worst-case (slowest) bits program by allowing enough programming time. Setting this value lower than the specification may not work all the time. Refer to the electrical specifications in *MC68HC908AS60 Advance Information*, Freescale document order number MC68HC908AS60/D.
- *Is the charge pump frequency correct?*
  The charge pump frequency has to be set between 1.8 MHz and 2.5 MHz. If the bus speed is not between 1.8 MHz and 2.5 MHz, you must set FDIV bits to generate a suitable charge pump frequency. Refer to **Charge Pump.**
- *Is the COP enabled?*
  If the COP is enabled, make sure that the COP bit is cleared before the COP reset occurs. Remember that in margin read mode, every byte requires seven additional cycles for sensing. Refer to the **Page Program/Margin Read Algorithm**.

Question 2 What is the FLASH charge pump?

Answer 2 The charge pump is a dynamic (clocked) circuit which generates high voltages internally in the FLASH to program and erase the non-volatile memory.

Question 3 The MC68HC908AS60 FLASH programs one page (eight bytes) at a time. Do I always have to program the entire page?

Answer 3 No, it is not necessary to program the entire page. Addresses which are not programmed are left as they were before the page programming was started. If one page includes reserved bytes, these bytes should not be programmed.

Question 4 Do I have to use the smart programming algorithm?

Answer 4 The use of smart programming algorithm is required. Freescale does not guarantee the performance of the FLASH if this algorithm is not followed. Refer to **Page Program/Margin Read Algorithm**.

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 can cause errors in the program or erase timing and lead to corrupt or missing data in the FLASH. Freescale does not guarantee performance of the FLASH if interrupts are not masked during the program or erase operations.

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

Answer 7 Yes. The MC68HC908AS60 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. The charge pump is shared between both arrays to minimize silicon area and cost. Therefore, only one high voltage FLASH operation (either program or erase) can occur at a time on the MC68HC908AS60.

Question 9 The FLASH specification states that a maximum of eight page program cycles can be done per row between erase cycles. What does this mean?

Answer 9 This specification states that a row (64 bytes of eight pages) should not be programmed more than eight times before erasing. Programming the row in excess of eight times risks inadvertent programming of erased bits. (This type of fault is known as program disturb.) Programming is done on a per-page basis where eight smart programming cycles are used typically to program eight pages, programming the entire row of 64 bytes. If further programming is required on a row after eight program cycles, the row must be erased first before it is programmed again.

Question 10 When writing eight bytes of data to one page of FLASH memory for programming, does the order of written data matter?

Answer 10 No, as long as the bytes are written within a page, the data is latched for the programming operation.

Question 11 I cannot program/erase the FLASH block protect register (FLBPR).

Answer 11 To program or erase the FLBPR, you must apply V_{Hi} on the IRQ pin. Refer to **Control and Block Protect Registers**.

Question 12 Does bus frequency affect the programming time? For example, is programming time using 8-MHz bus frequency shorter than using 2-MHz bus frequency?

Answer 12 FLASH program times using an 8-MHz bus verses a 2-MHz bus have minimal difference.

Question 13 I'm sending external data serially into the MC68HC908AS60 for programming. How can I speed up this process?

Answer 13 If you run the MC68HC908AS60 at a higher bus speed, you can improve the non-FLASH overhead during programming.

Question 14 If I program FLASH with 2-MHz bus frequency, can I read the FLASH with 8-MHz bus frequency without any problems?

Answer 14 Yes. The FLASH will meet all specifications, including data retention performance, if the FLASH is programmed/erased and used within specification limits.

Question 15 Why is the maximum number of program attempts (fls_{Pulses}) so high?

Answer 15 This limit is set high to account for the worst case manufacturing process variations, ensuring that the slowest FLASH bit will still program. On average, the page program times are much faster than this worst case limit.

Question 16 The program/erase operation is not successful in the monitor mode.

Answer 16 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 the *MC68HC908AS60 Advance Information*, Freescale document order number MC68HC908AS60/D.

Question 17 I have failed to break security in monitor mode. Can I execute a bulk erase?

Answer 17 Yes. Bulk erase is the only FLASH operation to attempt when failing to break security in monitor mode. Make sure the block protect feature is not asserted or override it to bulk erase the device.

Question 18 In the monitor mode, how I can tell if the break security has been successful?

Answer 18 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 19 Do I need to confirm the memory contents after programming the FLASH?

Answer 19 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. Some sort of error flag should be set if the data in the FLASH does not agree with what was programmed.

Question 20 A block of memory in the FLASH array is protected by programming the block protect register. When I execute bulk erase without applying high voltage on the IRQ pin, will all of the array, except for the protected block, be erased?

Answer 20 No. If any part of the array being bulk erased is protected, the bulk erase operation is defeated unless high voltage is placed on IRQ.

Question 21 What is the expected lifetime of FLASH memory?

Answer 21 The minimum program/erase endurance and data retention lifetime of the FLASH memory for all conditions is found in *MC68HC908AS60 Advance Information*, Freescale document order number MC68HC908AS60/D.

Question 22 What steps can I take to prolong the life of the FLASH memory?

Answer 22 The FLASH memory has a finite program/erase and 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 temperatures much lower than the maximum specified, such as 25°C. The program/erase endurance and data retention of this FLASH memory is worst at 125°C.

Question 23 Can I program/erase/read the FLASH at the maximum temperature limits continuously for the specified lifetime of the part?

Answer 23 Yes.

Question 24 What modes of operation cause the most noise?

Answer 24 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.

Schematic

Figure 9 shows the hardware schematic for the FLASH 2TS.

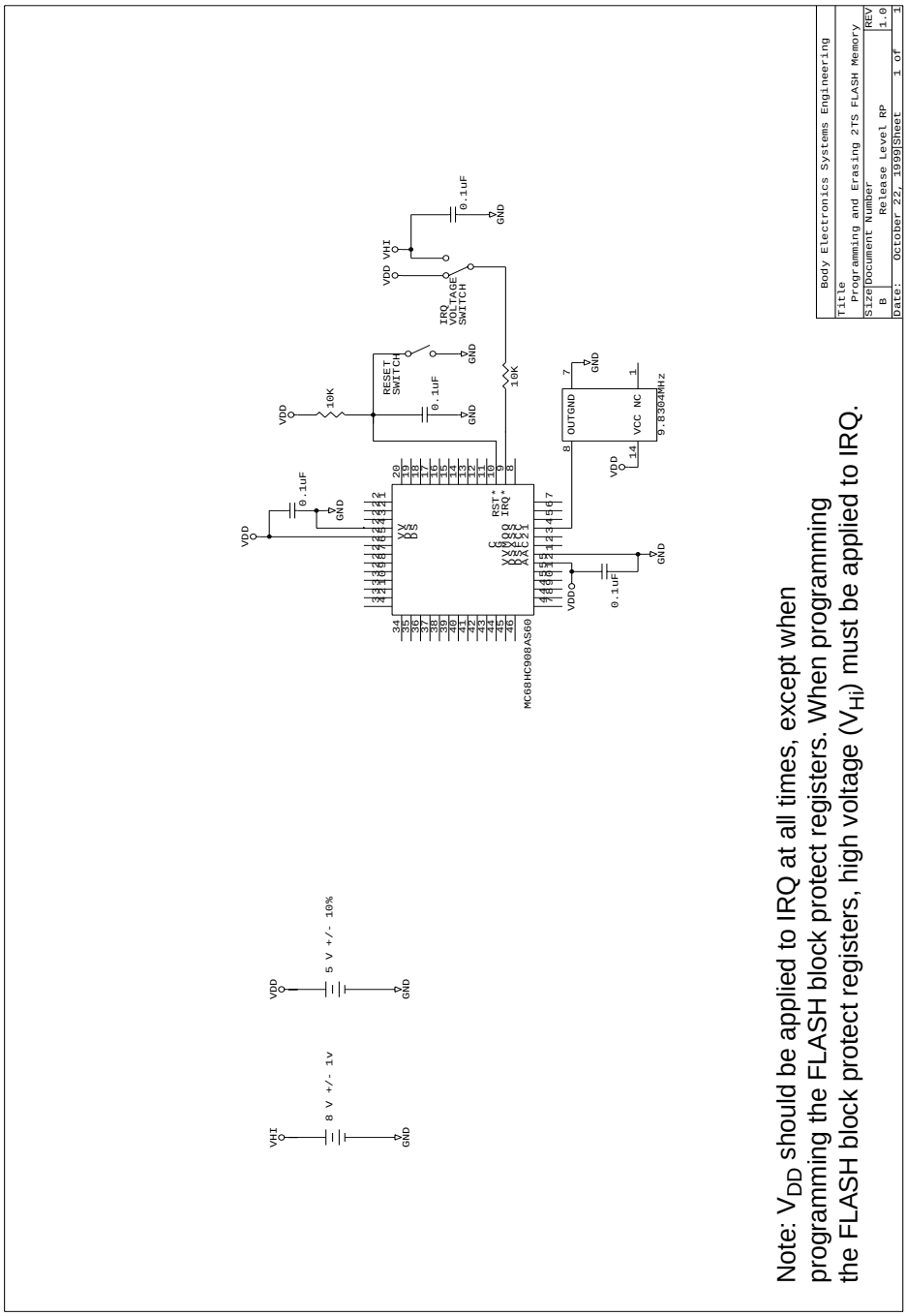


Figure 9. Hardware Schematic for Programming and Erasing FLASH 2TS on the MC68HC908AS60

Assembly Source Code

Sample assembly source code for FLASH programming and erasing are included in this section.

The routines `Erase.mrt` and `Program.mrt` are the respective main routines for erasing and programming. Both disable the COP and initialize the charge pump clocks. The main routines also set up the parameters required by the subroutines such as `FLASH_addr` and the size of the erase block or the data bytes to be programmed. `Erase.mrt` can erase a 64-byte, 512-byte, 16-Kbyte, or a 32-Kbyte block of FLASH. `Program.mrt` programs one page (eight bytes) of FLASH with the data 01, 02, 03, 04, 05, 06, 07, 08. **Figure 10** and **Figure 11** are the flowcharts for `Erase.mrt` and `Program.mrt`.

The subroutine `EraseRoutine` includes the erasing flowchart and is called from `Erase.mrt`. It calls subroutine `WriteFLCR` to set or clear various bits in the FLCR register and subroutine `Delay` to generate the required delays between steps.

`Prog8Bytes` is the smart programming algorithm subroutine called from `Program.mrt`. It also uses the `WriteFLCR` and `Delay` subroutines. Note that `Prog8Bytes` can make multiple attempts (up to the value set in fls_{Pulses}) to program one page.

The flowcharts for `EraseRoutine`, `Prog8Bytes`, `WriteFLCR` and `Delay` are **Figure 12**, **Figure 13**, **Figure 14**, and **Figure 15**, respectively.

NOTE: *V_{DD} should be applied to IRQ at all times, except when programming the FLASH block protect registers. When programming the FLASH block protect registers, flip the switch such that high voltage is applied to IRQ.*

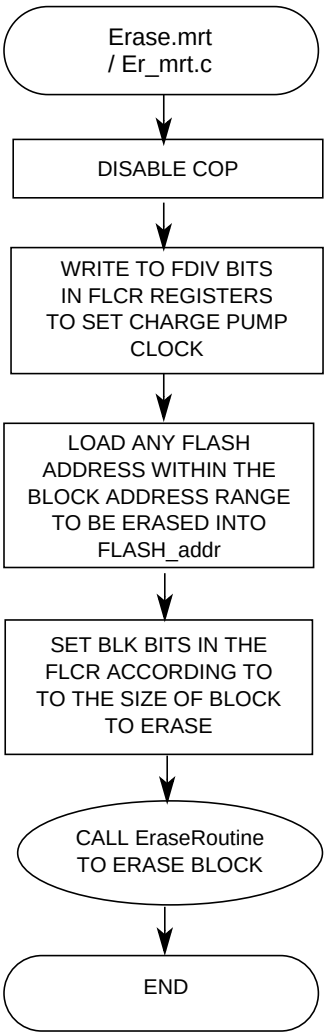


Figure 10. Erasing Main Routine Flowchart

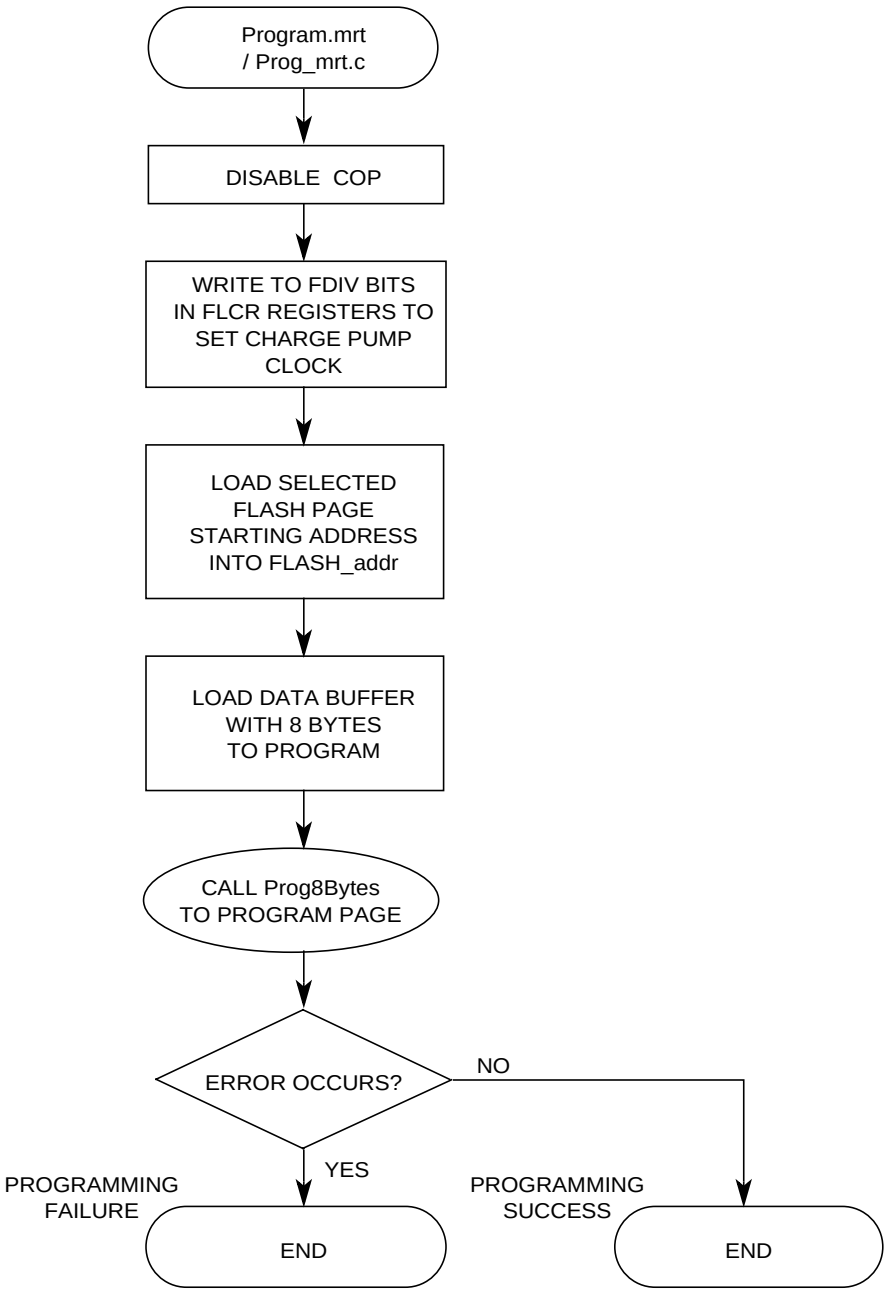


Figure 11. Programming Main Routine Flowchart

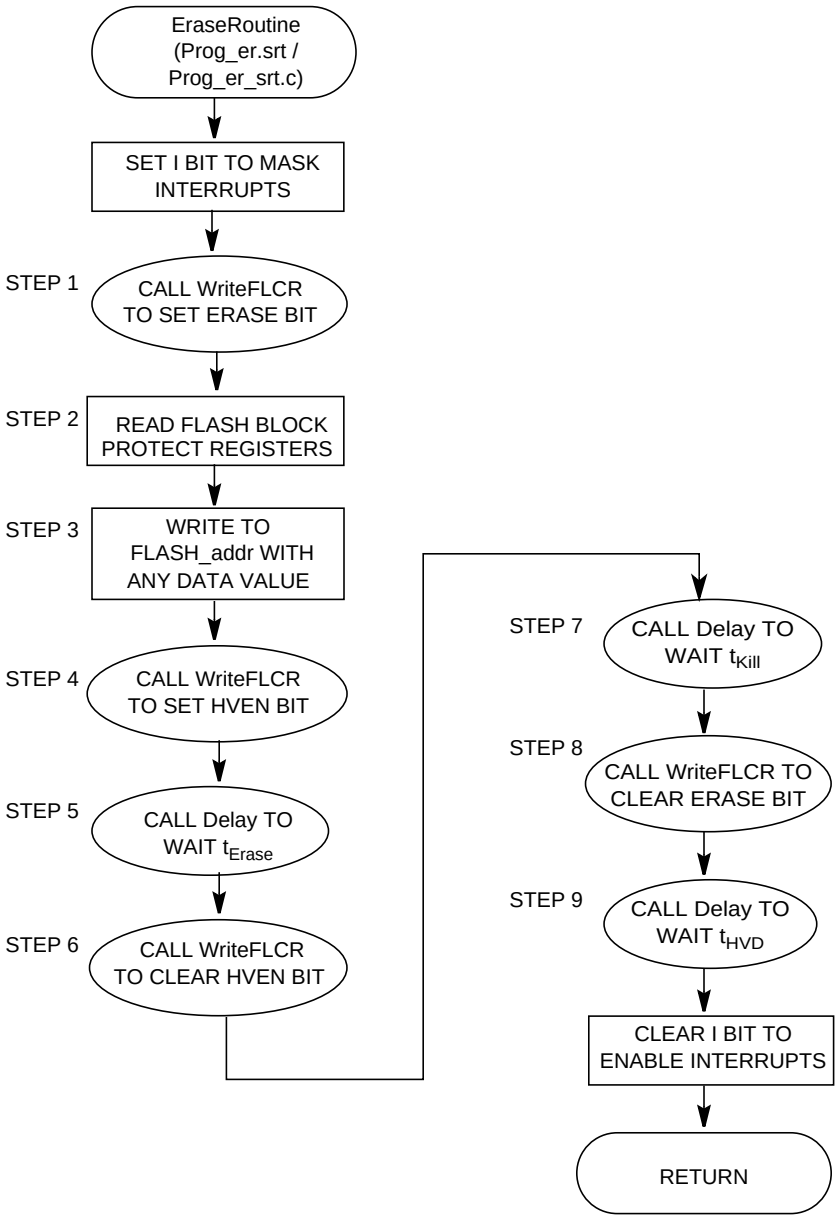


Figure 12. Subroutine EraseRoutine Flowchart

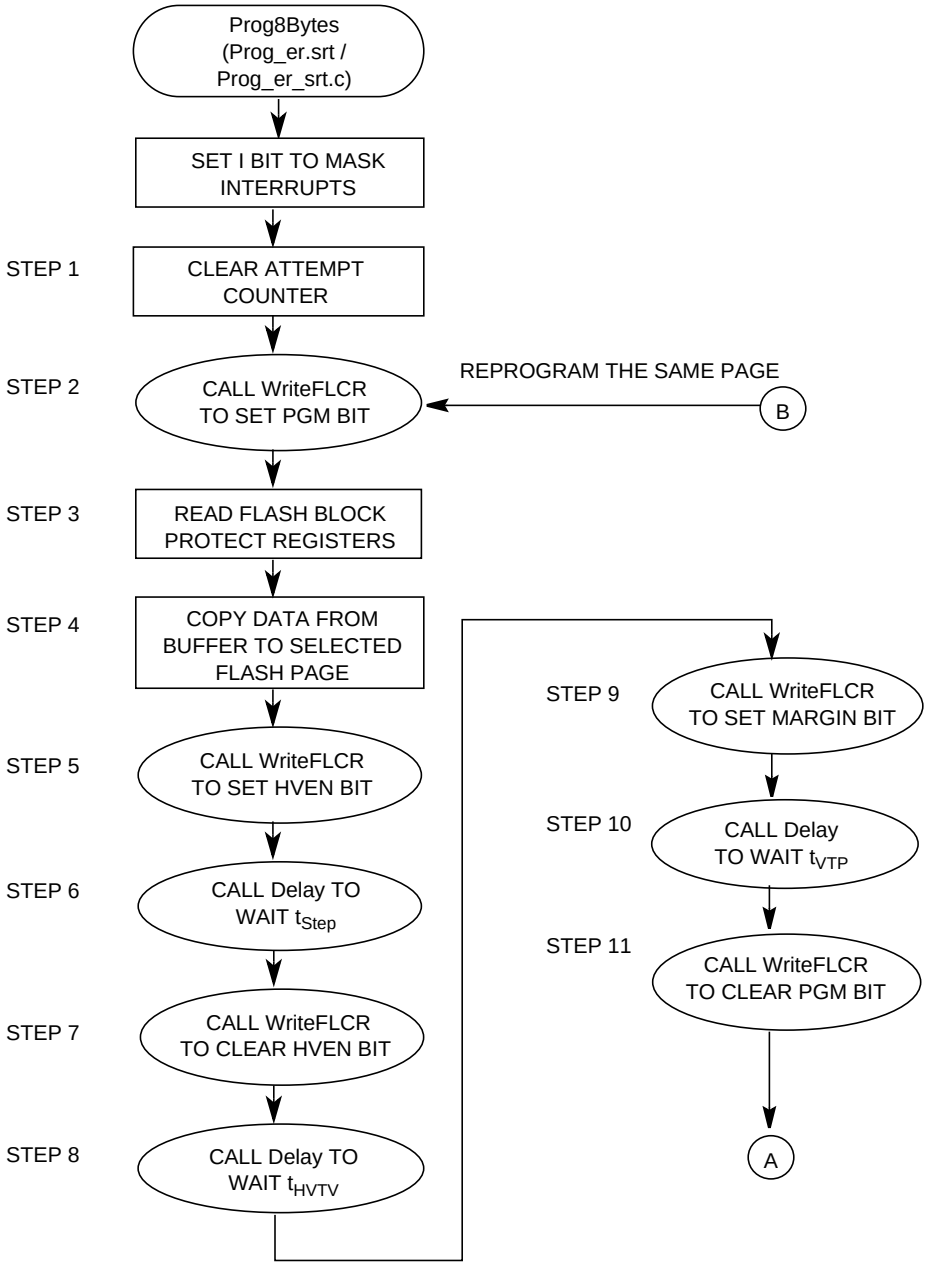
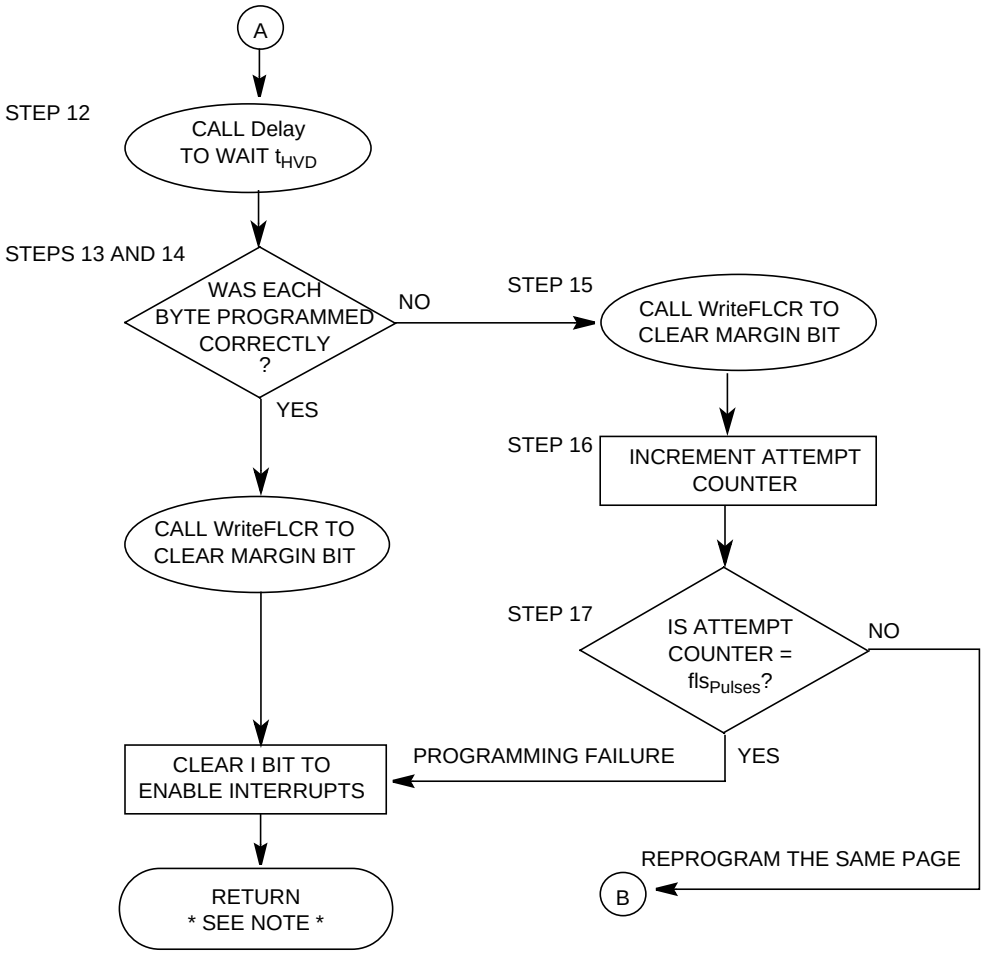


Figure 13. Subroutine Prog8Bytes Flowchart



Note: ACCUMULATOR VALUE (OR RETURNED VALUE) INDICATES THE PROGRAMMING RESULT.
ZERO VALUE = PROGRAMMING SUCCESS
NON-ZERO VALUE = PROGRAMMING FAILURE

Figure 13. Subroutine Prog8Bytes Flowchart (Continued)

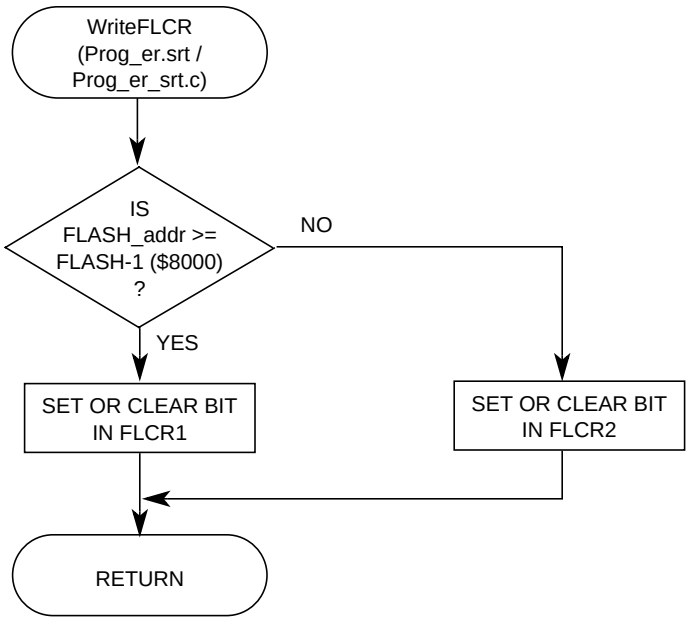


Figure 14. Subroutine WriteFLCR Flowchart

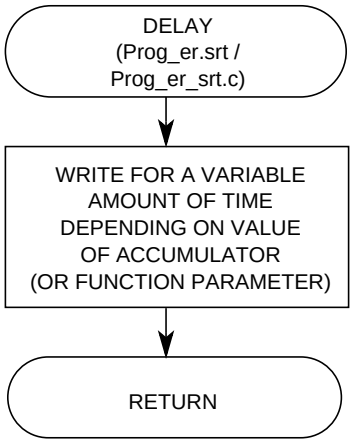


Figure 15. Subroutine Delay Flowchart

Assembly Source Code

```
*****************************************************************************
*****************************************************************************
*                                                                           *
*                  Erase FLASH 2TS Memory on the MC68HC908AS60              *
*                                                                           *
*****************************************************************************
* File Name: Erase.mrt                        Copyright (c)             *
*                                                                           *
* Current Revision: 1.0                                                     *
* Current Release Level: RP                                                 *
* Current Revision Release Date: 9/6/99                                     *
*                                                                           *
* Current Release Written By:  Kim Sparks                                   *
*                              Freescale Systems Engineering - Austin, TX      *
*                                                                           *
* Assembled Under: CASM08 (P&E Micro Inc.)     Ver.: 3.06 SLD               *
*                                                                           *
* Project Folder Name: FLASH_2TS                                            *
*                                                                           *
* Part Family Software Routine Works With: HC08                             *
* Part Module(s) Software Routine Works With: fls32k_a01                    *
*                                                                           *
* Routine Size (Bytes):      270                                            *
* Stack Space Used (Bytes):  6                                              *
* RAM Used (Bytes):          2                                              *
* Global Variables Used:     FLASH_addr                                     *
* Subroutine Called:         EraseRoutine, WriteFLCR                        *
*                                                                           *
* Full Functional Description Of Routine Design:                            *
*  Erase.mrt is the main routine that demonstrates how to erase different   *
*  size blocks of FLASH 2TS memory on the MC68HC908AS60.                    *
*                                                                           *
*****************************************************************************
* 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 is an Equal Opportunity/Affirmative Action Employer.                     *
*************************************************************************************
*************************************************************************************
*****                              Include Files                                  *****
*************************************************************************************
NOLIST
$INCLUDE          "H908as60.frk"           ;Equates for all registers and bits in
                                           ; the MC68HC908AS60

*************************************************************************************
*****                          Program Specific Equates                           *****
*************************************************************************************
eraseallrows.     equ       %00000000      ;Full array: 32 Kbytes (A15)
erasehalfrows.    equ       %00010000      ;One-half array: 16 Kbytes
                                           ; (A15 & A14)
erase8rows.       equ       %00100000      ;Eight rows: 512 bytes (A15-A9)
erase1row.        equ       %00110000      ;Single row: 64 bytes (A15-A6)

*************************************************************************************
*****                        RAM Variable Declarations                            *****
*************************************************************************************
       org        ram1
$INCLUDE          "Prog_er.var"            ;RAM variable definitions
LIST

*************************************************************************************
*****                             Erase Main Routine                              *****
*************************************************************************************
* This routine initializes the 908AS60 before calling the erasing routine,          *
* EraseRoutine. If a user plans to incorporate EraseRoutine into his/her            *
* program, make sure the FLASH control registers and the FLASH_addr are             *
* initialized before calling EraseRoutine.                                          *
*                                                                                   *
* After the initialization, EraseRoutine is called to erase a data block            *
* of 64 bytes, 512 bytes, 16 Kbytes or 32 bytes depending on the value in           *
* the control registers. FLASH_addr is any address within the block that            *
* will be erased. This program does not verify that the operation was               *
* successful.                                                                       *
*************************************************************************************
```

```
        org         flash-1

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

        lda         #$00                    ;Set up the frequency divide control
        sta         flcr1                   ; bits in flcr1 & flcr2 for an
        sta         flcr2                   ; appropriate charge pump clock

        ldhx        #$4001                  ;Any address within the block that will
                                            ; be erased

        sthx        FLASH_addr              ;Set address of erase block

        lda         #eraseallrows.          ;Set BLK1 and BLK0 in appropriate
        jsr         WriteFLCR               ; FLASH control register to specify the
                                            ; size of block to erase

        jsr         EraseRoutine            ;Erase the block of FLASH including the
                                            ; specified address
        bra         *

*****************************************************************************
*****                     Subroutine Body Includes Section                  *****
*****************************************************************************
$INCLUDE            "prog_er.srt"           ;WriteFLCR and Delay subroutines

*****************************************************************************
*****                             Reset Vectors                             *****
*****************************************************************************
        org         reset
        fdb         Start
*****************************************************************************
*****************************************************************************
*                                                                           *
*      Program / Margin Read FLASH 2TS Memory on the MC68HC908AS60          *
*                                                                           *
*****************************************************************************
* File Name: Program.mrt                  Copyright (c)              *
*                                                                           *
* Current Revision: 1.0                                                     *
* Current Release Level: RP                                                 *
* Current Revision Release Date: 9/6/99                                     *
*                                                                           *
* Current Release Written By: Kim Sparks                                    *
*                             Freescale Systems Engineering - Austin, TX      *
*                                                                           *
* Assembled Under: CASM08 (P&E Micro Inc.)    Ver.: 3.06 SLD                *
*                                                                           *
* Project Folder Name: FLASH_2TS                                            *
*                                                                           *
```

```
* Part Family Software Routine Works With: HC08                                *
* Part Module(s) Software Routine Works With: fls32k_a01                        *
*                                                                               *
* Routine Size (Bytes):      278                                                *
* Stack Space Used (Bytes):  5                                                  *
* RAM Used (Bytes):          11                                                 *
* Global Variables Used:     FLASH_addr, data, attempt                          *
* Subroutine Called:         Prog8Bytes                                         *
*                                                                               *
* Full Functional Description Of Routine Design:                                *
*  Program.mrt is the main routine for the programming operation. It            *
*  demonstrates a smart programming algorithm that minimizes the amount         *
*  of time needed to program a page of FLASH 2TS memory on the                  *
*  MC68HC908AS60. One page consists of eight consecutive bytes of FLASH         *
*  memory starting at either address $xxx0 or $xxx8.                            *
*                                                                               *
*********************************************************************************
*****                              Include Files                            *****
*********************************************************************************
NOLIST
$INCLUDE          "H908as60.frk"          ;Equates for all registers and bits in
                                          ; the MC68HC908AS60
       org        ram1
$INCLUDE          "Prog_er.var"           ;RAM variable definitions
LIST

*********************************************************************************
*****                       Program/Margin Read Main Routine                *****
*********************************************************************************
* This routine initializes the 908AS60 before calling the programming           *
* routine, Prog8Bytes. If a user plans to incorporate Prog8Bytes into           *
* his/her program, make sure the FLASH control registers, FLASH_addr and        *
* data buffer are initialized before calling Prog8Bytes.                        *
*                                                                               *
* After the initialization, Prog8Bytes is called to program 8 data bytes,       *
* or one page of FLASH memory. If programming was successful, then the          *
* program will jump to the branch always statement in "No_Error." If            *
* programming failed, then the program will remain at the branch always         *
* statement in "Load."                                                          *
*********************************************************************************
       org        flash-2

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

       lda        #$00                    ;Set up the frequency divide control
       sta        flcr1                   ; bits in flcr1 & 2 for the appropriate
       sta        flcr2                   ; charge pump clock

       ldhx       #$8000                  ;Load FLASH_addr with address of where
       sthx       FLASH_addr              ; the 8 bytes should start being
```

```
                                        ; programmed. Must be XXX0 or XXX8

       clrh                             ;Fill the RAM buffer, data, with values
       ldx         #8                   ; to program into FLASH.
Load:  stx         data-1,X             ; (ie. 01,02,03,04,05,06,07,08)
       decx
       bne         Load

       jsr         Prog8Bytes           ;Program the 8 data bytes

       cbeqa       #0,No_Error          ;Check if a programming error occurred

       bra         *                    ; **Programming failed**
                                        ; Take appropriate action
No_Error:
       bra         *                    ; **Programming successful**
                                        ; End of program

*******************************************************************************
*****                    Subroutine Body Includes Section                 *****
*******************************************************************************
$INCLUDE           "prog_er.srt"        ;Prog8Bytes, WriteFLCR and Delay
                                        ; subroutines
*******************************************************************************
*****                              Vectors                                *****
*******************************************************************************
       org         reset
       fdb         Start
*******************************************************************************
*******************************************************************************
*                                                                             *
*         Programming and Erasing FLASH 2TS Memory on the MC68HC908AS60       *
*                                                                             *
*******************************************************************************
* File Name: Prog_er.var                Copyright (c)                 *
*                                                                             *
* Current Revision: 1.0                                                       *
* Current Release Level: RP                                                   *
* Current Revision Release Date: 9/6/99                                       *
*                                                                             *
* Current Release Written By: Kim Sparks                                      *
*                            Freescale Systems Engineering - Austin, TX                    *
*                                                                             *
* Assembled Under: CASM08 (P&E Micro Inc.)    Ver.: 3.06 SLD                  *
*                                                                             *
* Project Folder Name: FLASH_2TS                                              *
*                                                                             *
* Part Family Software Routine Works With: HC08                               *
*                                                                             *
* RAM Used (Bytes):     11                                                    *
*                                                                             *
```

```
* Description:                                                                 *
*  RAM variable definitions for Program.mrt and Erase.mrt                     *
********************************************************************************
*****                              RAM Variables                           *****
********************************************************************************
FLASH_addr         rmb  $2                  ;16 bit Address of FLASH memory to
                                            ; erase or program
data:              rmb  $8                  ;Eight data bytes that will be
                                            ; programmed
attempt:           rmb  $1                  ;Counts number of attempts to program
                                            ; a page
********************************************************************************
*                                                                              *
*           Program / Margin Read and Erase FLASH 2TS Memory Subroutines       *
*                                                                              *
********************************************************************************
* File Name: Prog_er.srt                    Copyright (c)            *
*                                                                              *
* Current Revision: 1.0                                                        *
* Current Release Level: RP                                                    *
* Current Revision Release Date: 9/6/99                                        *
*                                                                              *
* Current Release Written By:   Kim Sparks                                     *
*                               Freescale Systems Engineering - Austin, TX                *
*                                                                              *
* Assembled Under: CASM08 (P&E Micro Inc.)     Ver.: 3.06 SLD                  *
*                                                                              *
* Project Folder Name: FLASH_2TS                                               *
*                                                                              *
* Part Family Software Routine Works With: HC08                                *
* Part Module(s) Software Routine Works With: fls32k_a01                       *
*                                                                              *
*                                                                              *
*  Module Size (Bytes):          EraseRoutine    53                            *
*                                Prog8Bytes      160                           *
*                                WriteFLCR       20                            *
*                                Delay           9                             *
*  Stack Space Used (Bytes):     EraseRoutine    4                             *
*                                Prog8Bytes      3                             *
*                                WriteFLCR       0                             *
*                                Delay           1                             *
*  RAM Used (Bytes):             EraseRoutine    2                             *
*                                Prog8Bytes      11                            *
*                                WriteFLCR       0                             *
*                                Delay           0                             *
*  Global Variable(s) Used:      EraseRoutine    FLASH_addr                    *
*                                Prog8Bytes      FLASH_addr, data, attempt     *
*                                WriteFLCR       FLASH_addr                    *
*                                Delay           None                          *
*  Submodule(s) Called:          EraseRoutine    WriteFLCR, Delay              *
*                                Prog8Bytes      WriteFLCR, Delay              *
```

```
*                                WriteFLCR       None                          *
*                                Delay           None                          *
*  Calling Sequence:             JSR EraseRoutine, JSR Prog8Byte               *
*                                JSR WriteFLCR, JSR Delay                      *
*  Entry Label:                  EraseRoutine, Prog8Bytes, WriteFLCR           *
*                                Delay                                         *
*  Entry Conditions:             EraseRoutine    2 bytes address defined at    *
*                                                  FLASH_addr                  *
*                                Prog8Bytes      2 bytes address defined at    *
*                                                  FLASH_addr                  *
*                                                8 programming bytes located   *
*                                                  at variables data           *
*                                WriteFLCR       2 bytes address defined at    *
*                                                  FLASH_addr                  *
*                                                FLCR (FLASH Control           *
*                                                  Register) bit definition    *
*                                                  passed in accumulator       *
*                                Delay           Delay variable passed in      *
*                                                  accumulator                 *
*  Number of Exit Points:        4                                             *
*  Exit Label:                   EraseRoutine   Erase990                       *
*                                Prog8Bytes      Prog990                       *
*                                WriteFLCR       FLCR990                       *
*                                Delay           Delay990                      *
*      Exit Conditions:          EraseRoutine    None                          *
*                                Prog8Bytes      pass/fail result in           *
*                                                accumulator                   *
*                                WriteFLCR       None                          *
*                                Delay           None                          *
*                                                                              *
* Full Functional Description Of Subroutine:                                   *
*  Prog_er.srt consists of two primary subroutines called EraseRoutine         *
*  and Prog8Bytes. These demonstrate 2TS flash erasing and programming         *
*  algorithms, respectively. The routines also call other subroutines          *
*  WriteFLCR and Delay.                                                        *
*                                                                              *
********************************************************************************
*****                          FLASH Delay Equates                         *****
********************************************************************************
*  Each delay time related to FLASH 2TS program and erase operations is        *
*  calculated with bus speed 2.4576 MHz.                                       *
********************************************************************************
tERASE: equ       $F6                    ;FLASH 2TS block/bulk erase time
                                         ; (~102 ms)
tKILL:  equ       $0A                    ;FLASH 2TS high-voltage kill time
                                         ; (~209 µs)
tHVD:   equ       $03                    ;FLASH 2TS return to read time
                                         ; (~66 µs)
tSTEP:  equ       $31                    ;FLASH 2TS page program step size
                                         ; (~1 ms)
tHVTV:  equ       $03                    ;FLASH 2TS HVEN low to MARGIN high
```

```
                                        ; time (~66 µs)
tVTP:           equ     $08             ;FLASH 2TS MARGIN high to PGM low
                                        ; time (~168 µs)
flsPULSES:      equ     $64             ;FLASH 2TS maximum page program pulses
                                        ; (100 pulses)
*************************************************************************************
*****                  Erase a Block of FLASH Memory Subroutine                 *****
*************************************************************************************
* This routine erases a block of FLASH of the size specified in the FLASH           *
* Control Registers and at the location specified by FLASH_addr.                    *
* FLASH_addr is any address within the block to be erased. The routine              *
* follows the basic 9 step sequence of the FLASH erase operation.                  *
*                                                                                   *
* Initializations required:                                                         *
*       Set charge pump clock and block erase size in FLCR1 and FLCR2;              *
*       FLASH_addr set                                                              *
* Values returned:                                                                  *
*       none                                                                        *
*************************************************************************************
EraseRoutine:
        sei                             ;Disable interrupts
        ldhx        FLASH_addr          ;Load the starting address of the block
                                        ; to be erased

        lda         #erase.             ;Step 1 - Set the ERASE bit
        jsr         WriteFLCR

        lda         flbpr1              ;Step 2 - Read from block protect
        lda         flbpr2              ; registers

        sta         ,X                  ;Step 3 - Write to any FLASH address
                                        ; within the block address range to be
                                        ; erased with any data value
                                        ;(indexed, 16 bit offset=H:X+$00)

        lda         #hven.              ;Step 4 - Set the HVEN bit
        jsr         WriteFLCR

        lda         #terase             ;Step 5 - Wait for time tERASE
        pshx
        ldx         #$14
Again:
        jsr         Delay
        dbnzx       Again
        pulx

        lda         #hven.              ;Step 6 - Clear the HVEN bit
        jsr         WriteFLCR

        lda         #tkill              ;Step 7 - Wait for time tKILL
        jsr         Delay
```

```
        lda         #erase.                 ;Step 8 - Clear the ERASE bit
        jsr         WriteFLCR

        lda         #thvd                   ;Step 9 - Wait for time tHVD
        jsr         DELAY

        cli                                 ;Enable interrupts

Erase990:
        rts
*******************************************************************************
*****           Program/Margin Read 8 Bytes of FLASH Memory Subroutine        *****
*******************************************************************************
* This routine programs the 8 bytes of data from the RAM data buffer to       *
* the FLASH memory starting at FLASH_addr. The programming is done in         *
* ascending order, ie. data[0] -> FLASH_addr[0], data[1] -> FLASH_addr[1],    *
* etc.                                                                        *
*                                                                             *
* The routine is structured into a loop for smart programming so that the     *
* minimum amount of high voltage is applied to the FLASH. Each time           *
* through the loop is called a "pulse." Each pulse consists of all 17         *
* steps shown in the application note, with the programming time, tSTEP,      *
* equal to approximately 1 ms.                                                *
*                                                                             *
* If all 8 bytes program correctly in the approximately 1 ms time period,     *
* then the routine exits and returns control to the calling program. If       *
* the 8 bytes do not program correctly, another attempt(pulse) is made.       *
* This continues for the maximum number of pulses (flsPULSES). If the 8       *
* bytes are not programmed successfully after a maximum number of pulses,     *
* then the programming error flag is incremented, the routine is exited       *
* and the error value (any non-zero value) is returned in the accumulator.    *
*                                                                             *
*  Initializations required:                                                  *
*       Set charge pump clock in FLCR1 and FLCR2; FLASH_addr set; data        *
*       buffer filled                                                         *
*  Values returned:                                                           *
*       A = Program Success/Error Flag                                        *
*******************************************************************************
Prog8Bytes:
        sei                                 ;No interrupts allowed during
                                            ; programming

        clr         attempt                 ;Step 1 - Clear attempt counter

NextAttempt:
        ldhx        FLASH_addr              ;Load the address of the page to be
                                            ; programmed in the HX register

        lda         #pgm.                   ;Step 2 - Set the PGM bit
        jsr         WriteFLCR
```

```
lda         flbpr1                  ;Step 3 - Read from the block protect
lda         flbpr2                  ; registers

lda         data                    ;Step 4 - Copy the 8 bytes of data
sta         ,x                      ; from the RAM buffer to the
lda         data+1                  ; appropriate FLASH locations
sta         1,x
lda         data+2
sta         2,x
lda         data+3
sta         3,x
lda         data+4
sta         4,x
lda         data+5
sta         5,x
lda         data+6
sta         6,x
lda         data+7
sta         7,x

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

lda         #tSTEP                  ;Step 6 - Wait for time tSTEP
jsr         Delay

lda         #hven.                  ;Step 7 - Clear the HVEN bit
jsr         WriteFLCR

lda         #tHVTV                  ;Step 8 - Wait for time tHVTV
jsr         Delay

lda         #margin.                ;Step 9 - Set the MARGIN bit
jsr         WriteFLCR

lda         #tVTP                   ;Step 10 - Wait for time tVTP
jsr         Delay

lda         #pgm.                   ;Step 11 - Clear the PGM bit
jsr         WriteFLCR

lda         #tHVD                   ;Step 12 - Wait for time tHVD
jsr         Delay

lda         ,x                      ;Step 13 and 14 - Check to see if the
cmp         data                    ; correct data was programmed in the
bne         Repeat                  ; FLASH page. If not, jump to Repeat to
lda         1,x                     ; do another attempt. If so, jump to
cmp         data+1                  ; Complete.
bne         Repeat
```

```
        lda         2,x
        cmp         data+2
        bne         Repeat
        lda         3,x
        cmp         data+3
        bne         Repeat
        lda         4,x
        cmp         data+4
        bne         Repeat
        lda         5,x
        cmp         data+5
        bne         Repeat
        lda         6,x
        cmp         data+6
        bne         Repeat
        lda         7,x
        cmp         data+7
        bne         Repeat
        jmp         Complete

Repeat:
        lda         #margin.                ;  ** Program Failure **
        jsr         WriteFLCR               ;Step 15 - Clear the MARGIN bit
        inc         attempt                 ;Step 16 - Increment attempt counter
        lda         attempt

        cbeqa       #flsPULSES,Return       ;Step 17 - If attempt is less than
        jmp         NextAttempt             ; flsPULSES, go back to NextAttempt
                                            ; (Step 2)

Complete:                                   ;  ** Program Success **
        lda         #margin.                ;Clear the MARGIN bit
        jsr         WriteFLCR
        clra

Return:
        cli                                 ;Clear the interrupt mask bit and
                                            ; return

Prog990:                                    ;If programming was unsuccessful in
        rts                                 ; flsPULSES, the accumulator has a
                                            ; non-zero value
```

```
*****************************************************************************
*****                        Write to a FLASH Control Register          *****
*****************************************************************************
* This routine determines whether flcr1 or flcr2 should be written to       *
* based on the FLASH address specified by FLASH_addr.                       *
*                                                                           *
*                                                                           *
* Initializations required:                                                 *
*       H:X = FLASH_addr                                                    *
*                                                                           *
* Values returned:                                                          *
*       None                                                                *
*****************************************************************************
WriteFLCR:
        cphx        #flash-1                ;If FLASH_addr is in FLASH-1 array,
        bhs         Array1                  ; jump to Array1

        eor         flcr2                   ;Write to flcr2 register
        sta         flcr2
        bra         FLCR990
Array1:
        eor         flcr1                   ;Write to flcr1 register
        sta         flcr1
FLCR990:
        rts
*****************************************************************************
*****                              Delay Routine                        *****
*****************************************************************************
* This routine delays for a variable amount of time depending on the value  *
* passed into the routine by the accumulator (A).                           *
* Delay = (2 + (2 + 45 + 3) * A + 2 + 4) / Bus Freq                         *
*                                                                           *
* Initializations required:                                                 *
*       A = delay variable                                                  *
* Values returned:                                                          *
*       None                                                                *
*****************************************************************************
Delay:
         pshx                               ;2 cyc., Store the lower address of
                                            ; FLASH_addr on the stack
Loop:    ldx        #$0F                    ;2 cyc., Initialize X for inner loop
         dbnzx      *                       ;3 cyc., 15 * 3 = 45 cycles
         dbnza      Loop                    ;3 cyc., Decrement value passed in by
                                            ; accumulator and repeat if necessary
         pulx                               ;2 cyc., Restore the lower address
Delay990:
         rts                                ;4 cyc., Return
```

C Source Code

The sample assembly source code contained in the previous section is written in C language in this section.

In C, the main routines are called *Er_mrt.c* and *Prog_mrt.c*. The same flowcharts apply.

```
/*****************************************************************************/
/*****************************************************************************/
/*                                                                           */
/*               Erase FLASH 2TS Memory on the MC68HC908AS60                 */
/*                                                                           */
/*****************************************************************************/
/* File Name: Er_mrt.c                          Copyright (c)       */
/*                                                                           */
/* Current Revision: 1.0                                                     */
/* Current Release Level: RP                                                 */
/* Current Revision Release Date: 10/02/99                                   */
/*                                                                           */
/* Current Release Written By:   Adeela Gill                                 */
/*                               Freescale Systems Engineering                       */
/*                               Austin, Texas                               */
/*                                                                           */
/* Compiled Under: HiCross HC08 (HiWare)   Ver.: 5.0.5                       */
/*                                                                           */
/* Project Folder Name: FLASH_2TS                                            */
/*                                                                           */
/* Part Family Software Routine Works With: HC08                             */
/* Part Module(s) Software Routine Works With: fls32k_a01                    */
/*                                                                           */
/* Routine Size (Bytes):     346                                             */
/* Stack Space Used (Bytes): 9                                               */
/* RAM Used (Bytes):         2                                               */
/* Global Variables Used:    FLASH_addr                                      */
/* Subroutine(s) Called:     EraseRoutine, WriteFLCR                         */
/*                                                                           */
/* Full Functional Description Of Routine Design:                            */
/*     Er_main.c is the main routine that demonstrates how to erase          */
/*     different size blocks of FLASH 2TS memory on the MC68HC908AS60.       */
/*                                                                           */
/*****************************************************************************/
/* 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.                         */
/*********************************************************************************/
/*********************************************************************************/
/*****                              Include Files                              *****/
/*********************************************************************************/
#include <as60_flash_frk.c>                 /* Equates for the registers and bits  */
                                            /* of the HC908AS60 that are used      */
#include <prog_er_var.c>                    /* RAM variable definitions            */
/*********************************************************************************/
/*****                         Program-Specific Defines                        *****/
/*********************************************************************************/
#define   eraseallrows   0x00               /* full array erase: 32 Kbytes         */
                                            /*      cared addresses A15            */
#define   erasehalfrows  0x10               /* one-half array erase: 16 Kbytes     */
                                            /*     cared addresses A15-A14         */
#define   erase8rows     0x20               /* eight row erase:  512 Bytes         */
                                            /*     cared addresses A15-A9          */
#define   erase1row      0x30               /* single row erase: 64 Bytes          */
                                            /*     cared addresses A15-A6          */
/*********************************************************************************/
/*****                           Function Definitions                          *****/
/*********************************************************************************/
extern void   EraseRoutine  (void);
extern void   writeFLCR     (unsigned char);

/*********************************************************************************/
/*****                            Erase Main Routine                           *****/
/*********************************************************************************/
/* This routine initializes the 908AS60 before calling the erasing                */
/* routine, EraseRoutine. If a user plans to incorporate EraseRoutine             */
/* into his/her program, make sure the FLASH control registers and the            */
/* variable FLASH_addr are initialized before calling EraseRoutine.               */
/* After the initialization, EraseRoutine is called to erase a data               */
```

```
/* block of 64 bytes, 512 bytes, 16 Kbytes or 32bytes depending on the            */
/* value in the control registers. FLASH_addr is any address within the          */
/* block that will be erased. This program does not verify that the              */
/* operation was successful.                                                     */
/*********************************************************************************/
void main ()
{
       CONFIG1 = 0x71;                        /*Turn off the COP, but leave the      */
                                              /*  LVI on                             */

       FLCR1 = 0x00;                          /*Set up the frequency divide          */
       FLCR2 = 0x00;                          /*  control bits in flcr1 & flcr2      */
                                              /*  for an appropriate charge pump     */
                                              /*  clock                              */
       FLASH_addr=(unsigned char *)0x4001;
                                              /*Set address of erase block. This     */
                                              /*  is any address within the block    */
                                              /*  that will be erased                */
       writeFLCR(eraseallrows);               /*Set BLK1 and BLK0 in appropriate     */
                                              /*  FLASH control register to specify  */
                                              /*  the size of block to erase         */
       EraseRoutine ();                       /*Erase the block of FLASH including   */
                                              /*  the specified address              */
        while (1);

}
/*********************************************************************************/
/****                          Subroutine Body Includes                       ****/
/*********************************************************************************/
#include      <prog_er_srt.c>                 /*Prog8Bytes, WriteFLCR and Delay      */
                                              /*  Subroutines                        */
/*********************************************************************************/
/*********************************************************************************/
/*                                                                               */
/*     Program / Margin Read FLASH 2TS Memory on the MC68HC908AS60               */
/*                                                                               */
/*********************************************************************************/
/* File Name: Prog_mrt.c                    Copyright (c)           */
/*                                                                               */
/* Current Revision: 1.0                                                         */
/* Current Release Level: RP                                                     */
/* Current Revision Release Date:  10/02/99                                      */
/*                                                                               */
/* Current Release Written By: Adeela Gill                                       */
/*                             Freescale Systems Engineering                      */
/*                             Austin, Texas                                     */
/*                                                                               */
/* Compiled Under: HiCross HC08 (HiWare)   Ver.: 5.0.5                           */
/*                                                                               */
/* Project Folder Name: FLASH_2TS                                                */
/*                                                                               */
/* Part Family Software Routine Works With: HC08                                 */
/* Part Module(s) Software Routine Works With: fls32k_a01                        */
```

```
/*                                                                          */
/* Routine Size (Bytes):     368                                            */
/* Stack Space Used (Bytes): 12                                             */
/* RAM Used (Bytes):         11                                             */
/* Global Variables Used:    FLASH_addr, data, attempt                      */
/* Subroutine(s) Called:     Prog8Bytes                                     */
/*                                                                          */
/* Full Functional Description Of Routine Design:                           */
/*   Prog_mrt.c is the main routine for the programming operation. It       */
/*   demonstrates a smart programming algorithm that minimizes the          */
/*   amount of time needed to program a page of 2TS FLASH memory on the     */
/*   MC68HC908AS60. One page consists of eight consecutive bytes of         */
/*   FLASH memory starting at either address $xxx0 or $xxx8.                */
/*                                                                          */
/**************************************************************************/
/**************************************************************************/
/*****                          Include Files                         *****/
/**************************************************************************/
#include <as60_flash_frk.c>                 /* Equates for the registers and bits */
                                            /*   of the HC908AS60 that are used   */
#include <prog_er_var.c>                    /* RAM variable definitions           */
/**************************************************************************/
/*****                        Function Definitions                    *****/
/**************************************************************************/
extern void           writeFLCR             (unsigned char);
extern unsigned char  Prog8Bytes            (void);
/**************************************************************************/
/*****                    Program/Margin Read Main Routine            *****/
/**************************************************************************/
/* This routine initializes the 908AS60 before calling the programming      */
/* routine, Prog8Bytes. If a user plans to incorporate Prog8Bytes into      */
/* his/her program, make sure the FLASH control registers, FLASH_addr       */
/* and data buffer are initialized before calling Prog8Bytes.               */
/* After the initialization, Prog8Bytes is called to program 8 data         */
/* bytes, or one page of FLASH memory. If programming was successful,       */
/* then the program will jump to the while(1) statement in the "if"         */
/* expression. If programming failed, then the program will remain at       */
/* the while(1) statement in the "else" expression.                         */
/**************************************************************************/
void main ()
{
       unsigned char   count;               /* Variable to count down bits  */
       unsigned char   pgmsuccess;          /* Return value from Prog8Bytes */

       CONFIG1 = 0x71;                      /*Turn off the COP, but leave the */
                                            /*  LVI on                        */

       FLCR1 = 0x00;                        /*Set up the frequency divide     */
       FLCR2 = 0x00;                        /*  control bits in flcr1 & 2 for */
                                            /*  the appropriate charge pump   */
                                            /*  clock                         */
```

```
        FLASH_addr = (unsigned char *)0x8000;
                                            /*Load FLASH_addr with address of     */
                                            /*  where the 8 bytes should start    */
                                            /*  being programmed. Must be XXX0    */
                                            /*  or XXX8                           */

        for (count = 8; count != 0; count--)
                data[count-1]=count;        /* Fill the RAM array, data, with      */
                                            /*  values to program into FLASH.     */
                                            /*  (ie. 01,02,03,04,05,06,07,08)     */
        pgmsuccess=Prog8Bytes();            /* Program the 8 data bytes            */

        if (pgmsuccess!=0)                  /* Check if a programming error        */
                                            /*  occurred                          */

               while (1);                   /* -- Programming failed --            */
                                            /*  Take appropriate action           */

        else while (1);                     /* -- Programming successful --        */
                                            /*  End of program                    */
}
/*****************************************************************************/
/*****                        Subroutine Body Includes Section              *****/
/*****************************************************************************/
#include      <prog_er_srt.c>               /* Prog8Bytes, WriteFLCR and Delay   */
                                            /* Subroutines                       */
/*****************************************************************************/
/*****************************************************************************/
/*                                                                           */
/*                           HC908AS60 FLASH Framework                       */
/*                                                                           */
/*****************************************************************************/
/* File Name: as60_flash_frk.c          Copyright (c)                */
/*                                                                           */
/* Current Revision: 1.0                                                     */
/* Current Release Level: RP                                                 */
/* Current Revision Release Date: 10/02/99                                   */
/*                                                                           */
/* Current Release Written By:  Adeela Gill                                  */
/*                               Freescale Systems Engineering                                */
/*                               Austin, Texas                               */
/*                                                                           */
/* Compiled Under: HiCross HC08 (HiWare)   Ver.: 5.0.5                       */
/*                                                                           */
/* Project Folder Name: FLASH_2TS                                            */
/*                                                                           */
/* Part Family Software Routine Works With: HC08                             */
/* Part Module(s) Software Routine Works With: fls32k_a01                    */
/*                                                                           */
/* Framework Description:                                                    */
/*     This framework was generated using the MC68HC908AT60 General         */
/*     Release Specification as a reference.                                 */
/*****************************************************************************/
```

```
/*****************************************************************************/
/*****                          Register Equates                         *****/
/*****************************************************************************/
#define CONFIG1 (*((volatile unsigned char *)0x001F))
                                        /* Configuration Register 1          */

#define FLCR1   (*((volatile unsigned char *)0xFE0B))
                                        /* FLASH Control Register 1          */
#define FLCR2   (*((volatile unsigned char *)0xFE11))
                                        /* FLASH Control Register 2          */

#define FLBPR1  (*((volatile unsigned char *)0xFF80))
                                        /* FLASH Block Protect Register 1    */
#define FLBPR2  (*((volatile unsigned char *)0xFF81))
                                        /* FLASH Block Protect Register 2    */

#define FLASH1  (*((volatile unsigned char *)0x8000))
                                        /* Start Address of FLASH Array 1    */
#define FLASH2  (*((volatile unsigned char *)0x0450))
                                        /* Start Address of FLASH Array 2    */
/*****************************************************************************/
/*****                            Bit Equates                            *****/
/*****************************************************************************/
#define HVEN          0x08              /* Bit 3: High-Voltage Enable Bit    */
#define MARGIN        0x04              /* Bit 2: Margin Read Control Bit    */
#define ERASE         0x02              /* Bit 1: Erase Control Bit          */
#define PGM           0x01              /* Bit 0: Program Control Bit        */
/*****************************************************************************/
/*****************************************************************************/
/*                                                                           */
/*           Programming and Erasing FLASH 2TS Memory on the MC68HC908AS60   */
/*                                                                           */
/*****************************************************************************/
/* File Name: Prog_er_var.c                Copyright (c)          */
/*                                                                           */
/* Current Revision: 1.0                                                     */
/* Current Release Level: RP                                                 */
/* Current Revision Release Date: 10/02/99                                   */
/*                                                                           */
/* Current Release Written By:  Adeela Gill                                  */
/*                              Freescale Systems Engineering                                */
/*                              Austin, Texas                                */
/*                                                                           */
/* Compiled Under: HiCross HC08 (HiWare)   Ver.: 5.0.5                       */
/*                                                                           */
/* Project Folder Name: FLASH_2TS                                            */
/*                                                                           */
/* Part Family Software Routine Works With: HC08                             */
/* Part Module(s) Software Routine Works With: fls32k_a01                    */
/*                                                                           */
/* RAM Used (Bytes):          11                                             */
/*                                                                           */
/* Description:                                                              */
```

```
/*   RAM variable definitions for Prog_mrt.c and Er_main.c                    */
/*                                                                            */
/******************************************************************************/

/******************************************************************************/
/****                              RAM Variables                           ****/
/******************************************************************************/
unsigned char * FLASH_addr;                 /* 16 bit address of FLASH memory to  */
                                            /*   erase or program                 */
unsigned char  data[8];                     /* Eight data bytes that will be      */
                                            /*   programmed                       */
unsigned char  attempt;                     /* Counts number of attempts to       */
                                            /*   program a page                   */
/******************************************************************************/
/******************************************************************************/
/*                                                                            */
/*         Program / Margin Read and Erase FLASH 2TS Memory Subroutines       */
/*                                                                            */
/******************************************************************************/
/* File Name: Prog_er_srt.c                Copyright (c)           */
/*                                                                            */
/* Current Revision: 1.0                                                      */
/* Current Release Level: RP                                                  */
/* Current Revision Release Date: 10/02/99                                    */
/*                                                                            */
/* Current Release Written By: Adeela Gill                                    */
/*                             Freescale Systems Engineering                   */
/*                             Austin, Texas                                  */
/*                                                                            */
/* Compiled Under: HiCross HC08 (HiWare)   Ver.: 5.0.5                        */
/*                                                                            */
/* Project Folder Name: FLASH_2TS                                             */
/*                                                                            */
/* Part Family Software Routine Works With: HC08                              */
/* Part Module(s) Software Routine Works With: fls32k_a01                     */
/*                                                                            */
/* Routine Sizes (Bytes):    composite with called routines:                  */
/*                              Prog8Bytes     187 bytes                      */
/*                              EraseRoutine   75 bytes                       */
/*                              Delay          24 bytes                       */
/*                              WriteFLCR      30 bytes                       */
/* Stack Space Used (Bytes): composite with called routines:                  */
/*                              Prog8Bytes     8 bytes                        */
/*                              EraseRoutine   9 bytes                        */
/*                              Delay          2 bytes                        */
/*                              WriteFLCR      1 bytes                        */
/* RAM Used (Bytes):         composite with called routines:                  */
/*                              Prog8Bytes    11 bytes                        */
/*                              EraseRoutine  2 bytes                         */
/*                              Delay         0 bytes                         */
/*                              WriteFLCR     2 bytes                         */
/*                                                                            */
/* Global Variable(s) Used:  Prog8Bytes      FLASH_addr, data, attempt        */
```

```
/*                          EraseRoutine    FLASH_addr                      */
/*                          Delay           None                            */
/*                          WriteFLCR       FLASH_addr                      */
/*                                                                          */
/* Submodule(s) Called:     Prog8Bytes      WriteFLCR, Delay                */
/*                          EraseRoutine    WriteFLCR, Delay                */
/*                          Delay           None                            */
/*                          WriteFLCR       None                            */
/*                                                                          */
/* Calling Sequence:   EraseRoutine ()                                      */
/*                          int=Prog8Bytes ()                               */
/*                          WriteFLCR (unsigned char)                       */
/*                          Delay (unsigned char)                           */
/*                                                                          */
/* Entry Label:        void EraseRoutine(),                                 */
/*                          unsigned char Prog8Bytes()                      */
/*                          void WriteFLCR (unsigned char)                  */
/*                          void Delay (unsigned char)                      */
/*                                                                          */
/* Entry Conditions:   EraseRoutine  2 byte address defined at              */
/*                                     FLASH_addr                           */
/*                          Prog8Bytes      2 bytes address defined at      */
/*                                     FLASH_addr                           */
/*                                     8 programming bytes located          */
/*                                     at variable data                     */
/*                          WriteFLCR       2 bytes address defined at      */
/*                                     FLASH_addr                           */
/*                                     1 byte FLCR bit definition           */
/*                                     passed in                            */
/*                          Delay           1 byte delay value passed in    */
/*                                                                          */
/* Number of Exit Points:   4                                               */
/*     Exit Label:          EraseRoutine    Erase990                        */
/*                          Prog8Bytes      Prog990                         */
/*                          WriteFLCR       FLCR990                         */
/*                          Delay           Delay990                        */
/*                                                                          */
/*     Exit Conditions:     EraseRoutine    None                            */
/*                          Prog8Bytes      pass/fail result returned       */
/*                          WriteFLCR       None                            */
/*                          Delay           None                            */
/*                                                                          */
/* Full Functional Description Of Subroutine:                               */
/*   Prog_er_srt.c consists of two primary subroutines called               */
/*   EraseRoutine and Prog8Bytes. These demonstrate 2TS flash erasing       */
/*   and programming algorithms, respectively. The routines also call       */
/*   other subroutines WriteFLCR and Delay.                                 */
/*                                                                          */
```

```
/*****************************************************************************/
/*****************************************************************************/
/*****                          FLASH Delay Equates                          *****/
/*****************************************************************************/
/* Each delay time related to FLASH 2TS program and erase operations is      */
/* calculated with bus speed 2.4576 MHz.                                     */
/*****************************************************************************/
#define tERASE     0xF0                    /* FLASH 2TS block/bulk erase time    */
                                            /*   (~101 ms)                        */
#define tKILL      0x0A                    /* FLASH 2TS high-voltage kill time   */
                                            /*   (~210 µs)                        */
#define tHVD       0x02                    /* FLASH 2TS return to read time      */
                                            /*   (~51 µs)                         */
#define tSTEP      0x32                    /* FLASH 2TS page program step size   */
                                            /*   (~1 ms)                          */
#define tHVTV      0x02                    /* FLASH 2TS HVEN low to MARGIN high  */
                                            /*   time (~51 µs)                    */
#define tVTP       0x07                    /* FLASH 2TS MARGIN high to PGM low   */
                                            /*   time (~151 µs)                   */
#define flsPULSES  0x64                    /* FLASH 2TS maximum page program     */
                                            /*  pulses (100 pulses)               */
/*****************************************************************************/
/*****              Necessary Assembly-Level Commands Defines                *****/
/*****************************************************************************/
#define EnableInterrupts   {asm CLI;}
#define DisableInterrupts  {asm SEI;}
/*****************************************************************************/
/*****                          Function Initializations                     *****/
/*****************************************************************************/
void                  EraseRoutine         (void);
unsigned char         Prog8Bytes           (void);
void                  writeFLCR            (unsigned char);
void                  Delay                (unsigned char);
/*****************************************************************************/
/****                   Erase a Block of FLASH Memory Subroutine             *****/
/*****************************************************************************/
/* This routine erases a block of FLASH of the size specified in the         */
/* FLASH Control Registers and at the location specified by FLASH_addr.      */
/* FLASH_addr is any address within the block to be erased. The routine      */
/* follows the basic 9 step sequence of the FLASH erase operation.           */
/*                                                                           */
/* Initializations required:                                                 */
/*     Set charge pump clock and block erase size in FLCR1 and FLCR2;        */
/*     FLASH_addr set                                                        */
/* Values returned:                                                          */
/*     none                                                                  */
/*****************************************************************************/
void EraseRoutine ()
{
       unsigned char Blk_Protect1;         /* set up local variables         */
       unsigned char Blk_Protect2;
       unsigned char loop;
```

```
        DisableInterrupts;                  /* Disable interrupts                */

        writeFLCR (ERASE);                  /* Step 1 - Set the ERASE bit        */

        Blk_Protect1 = FLBPR1;              /* Step 2 - Read from the block      */
        Blk_Protect2 = FLBPR2;              /*   protect registers               */

        *FLASH_addr = 0xFF;                 /* Step 3 - Write to any FLASH       */
                                            /*   address within the block address */
                                            /*   range with any data value       */

        writeFLCR (HVEN);                   /* Step 4 - Set the HVEN bit         */

        for (loop=0x15; loop !=0; loop--)
              Delay (tERASE);               /* Step 5 - Wait for time tERASE     */

        writeFLCR (HVEN);                   /* Step 6 - Clear the HVEN bit       */

        Delay (tKILL);                      /* Step 7 - Wait for time tKILL      */

        writeFLCR (ERASE);                  /* Step 8 - Clear the ERASE bit      */

        Delay (tHVD);                       /* Step 9 - Wait for time tHVD       */

        EnableInterrupts;                   /* Enable Interrupts                 */

Erase990:
        return;
}
/*****************************************************************************/
/****              Program/Margin Read 8 Bytes of FLASH Memory Subroutine   *****/
/*****************************************************************************/
/* This routine programs the 8 bytes of data from the RAM data array to      */
/* the FLASH memory starting at FLASH_addr. The programming is done in       */
/* ascending order, ie. data[0] -> FLASH_addr[0], data[1] ->                 */
/* FLASH_addr[1], etc.                                                       */
/*                                                                           */
/* The routine is structured into a loop for smart programming so that       */
/* the minimum amount of high voltage is applied to the FLASH. Each time     */
/* through the loop is called a "pulse". Each pulse consists of all 17       */
/* steps shown in the application note, with the programming time, tSTEP,    */
/* equal to approximately 1 ms.                                              */
/*                                                                           */
/* If all 8 bytes program correctly in the approximately 1ms time period,    */
/* then the routine exits and returns control to the calling program. If     */
/* the 8 bytes do not program correctly, another attempt(pulse) is made.     */
/* This continues for the maximum number of pulses (flsPULSES). If the 8     */
/* bytes are not programmed successfully after a maximum number of           */
/* pulses, then the programming error flag is incremented, the routine is    */
/* exited and the error value (any non-zero value) is returned.              */
/*                                                                           */
```

```
/*  Initializations required:                                                 */
/*    Set charge pump clock in FLCR1 and FLCR2; FLASH_addr set; data           */
/*     buffer filled                                                          */
/*  Values returned:                                                          */
/*      Program Success/Error Flag                                            */
/*************************************************************************/
unsigned char Prog8Bytes ()
{
       unsigned char Blk_Protect1;          /* set up local variables          */
       unsigned char Blk_Protect2;
       unsigned char byte,status;

       DisableInterrupts;                   /* No interrupts allowed during    */
                                            /*   programming                   */

       attempt=0;                           /* Step 1 - Clear attempt counter  */

        do {                                /* Set up Repeat Loop              */

              writeFLCR (PGM);              /* Step 2 - Set the PGM bit        */

              Blk_Protect1 = FLBPR1;        /* Step 3 - Read from the block    */
              Blk_Protect2 = FLBPR2;        /*   protect registers             */

                                            /* Step 4 - Copy the 8 bytes of data */
                                            /*   from the RAM buffer to the    */
                                            /*  appropriate FLASH locations    */
              for (byte=0;byte<8;byte++)
                      *(FLASH_addr+byte)=data[byte];

              writeFLCR (HVEN);             /* Step 5 - Set the HVEN bit       */

              Delay (tSTEP);                /* Step 6 - Wait for time tSTEP    */

              writeFLCR (HVEN);             /* Step 7 - Clear the HVEN bit     */

              Delay (tHVTV);                /* Step 8 - Wait for time tHVTV    */

              status=0;

              writeFLCR (MARGIN);           /* Step 9 - Set the MARGIN bit     */

              Delay (tVTP);                 /* Step 10 - Wait for time tVTP    */

              writeFLCR (PGM);              /* Step 11 - Clear the PGM bit     */

              Delay (tHVD);                 /* Step 12 - Wait for time tHVD    */
```

```
            for (byte=0; byte<8; byte++)
                 if (*(FLASH_addr+byte) == data[byte]) status++;

                                        /* Step 13 and 14 - Check to see if     */
                                        /*   the correct data was programmed    */
                                        /*   in the FLASH page. If not,         */
                                        /*   proceed with the bracketed         */
                                        /*   expressions. If so, skip to next   */
                                        /*   if statement.                      */
            if (status != 8) {
                                        /* -- Program Failure --                */
                                        /* Step 15 - Clear the MARGIN bit       */
                    writeFLCR (MARGIN);
                    attempt++;          /* Step 16 - Increment attempt          */
                                        /* counter                              */
                     }

            if (status == 8)           /* -- Program Success --                */
                    writeFLCR (MARGIN);
                                        /* Clear the MARGIN bit                 */

} while ((status!=8) && (attempt<flsPULSES));
                                        /* Step 17 - If attempt is less than    */
                                        /*   flsPULSES, go back to step 2       */

       EnableInterrupts;                /* Clear the interrupt mask bit         */

Prog990:
       return (8-status);               /* If programming was unsuccessful in   */
                                        /*   flsPULSES, a non-zero value is     */
                                        /*   returned                           */
}
/*****************************************************************************/
/****                  Write to a FLASH Control Register                  *****/
/*****************************************************************************/
/* This routine determines whether flcr1 or flcr2 should be written to       */
/* based on the FLASH address specified by FLASH_addr.                       */
/*                                                                           */
/* Initializations required:                                                 */
/*      set FLASH_addr                                                       */
/* Values returned:                                                          */
/*      None                                                                 */
/*****************************************************************************/
void writeFLCR (unsigned char A)
{
                                        /* FLASH-1=$8000, FLASH-2=$0450         */
       if (FLASH_addr >= &FLASH1)       /*If FLASH_addr is in FLASH-1 array     */
              FLCR1 = FLCR1 ^ A;        /* write to FLCR1 Register              */

       else
              FLCR2 = FLCR2 ^ A;        /* else write to FLCR2 Register         */

FLCR990:
       return;
```

```
/*****************************************************************************/
/****                          Delay Routine                                ****/
/*****************************************************************************/
/* This routine delays for a variable amount of time depending on the        */
/* value passed into the routine as variable A.                              */
/* Delay time = (12 + (5 + (11+72+11)*A + 11) + 8) / Bus Freq                */
/* This equation includes the bus cycles from calling the delay routine      */
/* until hitting the next line of code. This allows the user to              */
/* accurately predict the delay between two steps in the code.               */
/*                                                                           */
/* Initializations required:                                                 */
/*     Pass in variable A = Delay Value                                      */
/* Values returned:                                                          */
/*     None                                                                  */
/*****************************************************************************/
void Delay (unsigned char A)
{
       unsigned char   count;

       while (A!=0) {
              for (count=0x03;count!=0;count--);
              A--;
       }

Delay990:
       return;
}
```