

Programming and Erasing FLASH and EEPROM Memories on the MC68HC912DT128A/DG128A/D60A

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

Introduction

Motorola has released two new microcontrollers (MCU), the MC68HC912DT128A (DT128A) and the MC68HC912DG128A (DG128A), as new products in the M68HC12 Family of devices.

NOTE: *The MC68HC912D60A is not referenced specifically in this application note, but the memory technology and algorithms apply to this part as well.*

The DT128A and DG128A offer many features, including:

- 16-bit central processor unit (CPU)
- 128 Kbytes of FLASH memory
- FLASH boot code protection
- 2 Kbytes of EEPROM
- 8 Kbytes of on-chip RAM

This application note explains how to use the FLASH and EEPROM on the MC68HC912DT128A/DG128A and provides example software for program and erase operations. These algorithms are written in M68HC12 assembly code.

This code is available for download from <http://www.motorola.com/semiconductor/> which is Motorola's Semiconductor Product Sector's Web site.

The topics covered in this application note include:

- **Reference Documents**
- **MC68HC912DT128A/DG128A vs. MC68HC912DG128**
- **FLASH Functional Description**
- **FLASH Memory Mapping**
- **FLASH Control Registers**
- **FLASH Erase Operation**
- **FLASH Program Operation**
- **EEPROM Functional Description**
- **EEPROM Control Registers**
- **EEPROM Block Protection**
- **Timebase Initialization and SHADOW Word**
- **EEPROM Erase Operation**
- **EEPROM Program Operation**
- **Selective Bit Programming**
- **Evaluating Delay Times for the Sample Code**
- **FLASH Frequently Asked Questions**
- **EEPROM Frequently Asked Questions**
- **Sample Code**

Reference Documents

For complete information on the MC68HC912DT128A/DG128A devices, the user should reference *MC68HC912DT128A and MC68HC912DG128A Technical Data*, Motorola document number MC68HC912DT128A/D.

For information on the MC68HC912D60A, the user should reference *MC68HC912D60A Technical Data*, Motorola document number MC68HC912D60A/D.

The memory cells used in the MC68HC912DT128A/DG128A devices are split gate cells from Silicon Storage Technology (SST) in 0.5-micron geometry. The SST Web site contains a detailed description of these cells. Refer to <http://www.ssti.com>.

MC68HC912DT128A/DG128A vs. MC68HC912DG128

In general, the MC68HC912DT128A/DG128A devices discussed in this application note are a technology shrink from the MC68HC912DG128 device.

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

- Using a new FLASH and EEPROM technology
- A new FLASH programming algorithm. The new algorithm programs on a per-row basis and is faster and simpler than before.
- Availability of a new, faster EEPROM programming algorithm. This new algorithm automatically turns off erasing/programming voltage when the operation is completed.
- Addition of an internal charge pump for the FLASH to supply programming and erasing voltage. There is no need to provide a high voltage supply on the VFP pin.

WARNING: *Do not apply 12 volts to the VFP pin on the "A" Family devices. This may damage the device! For safety, the user may connect this pin to V_{SS} or V_{DD} .*

- Adding a required constant timebase source to the new EEPROM. The timebase is the external clock input, EXTAL, divided by the value programmed into the EEDIVH and EEDIVL registers.

The devices are compared in detail in [Table 1](#) and [Table 2](#).

Table 1. FLASH Comparison

	MC68HC912DG128	MC68HC912DT128A/ DG128A
Type	UDR (1.5T)	UDR (SST)
FLASH control register	ERAS is bit 2, LATCH control is bit 1	ERAS is bit 1; bit 2 has no function.
Programming voltage	External programming and erasing voltage VFP must be provided	No external high voltage needed; internal charge pump
Algorithm	Multiple pulses using margin read verification	One fixed pulse; erase and programming times fixed
Bit-erased state	"1"	"1"
Programming minimum size	1 byte	1 word (2 bytes)
Erasing minimum size	bulk (32 Kbyte array)	bulk (32 Kbyte array)
128 Kbytes programming time	Typical 8 s	Minimum 2 s
Erase time	Minimum 100 ms for all sizes: 64 bytes, 512 bytes, 16 Kbytes, 32 Kbytes	Bulk erase: minimum 8 ms

Table 2. EEPROM Comparison

	MC68HC912DG128	MC68HC912DT128A/ DG128A
Minimum programming clock frequency	1.0 MHz	250 kHz
Bit-erased state	"1"	"1"
Algorithm	Fixed delays	Two modes: Standard mode: Can utilize same routine as MC68HC912DG128 AUTO mode: Program/erase cycle is terminated by the internal timer
Registers	4-byte block from \$00F0	6 byte block from \$00EE EEDIVH = \$00EE EEDIVL = \$00EF All other registers same
Erase sizes supported	1 byte, 1 word (2 bytes), 1 row (32 bytes), bulk	1 byte, 1 word (2 bytes), 1 row (32 bytes), bulk
Program sizes supported	1 byte or 1 word (2 bytes)	1 byte or 1 word (2 bytes)
Charge pump clock	EERC controls clock source. EERC = 0 = system clock; EERC = 1 = internal RC oscillator	Clock source is the oscillator clock (EXTAL) User specifies the divide ratio
SHADOW size and location	1 byte at \$0FC0	1 word (2 bytes) at \$0FC0-\$0FC1
SHADOW mapping	At reset, SHADOW byte is loaded into EEMCR	At reset, SHADOW word is loaded into EEMCR, EEDIVH, and EEDIVL
SHADOW disable bit	NOSHB, bit 6 in EEMCR	NOSHW, bit 6 in EEMCR
Successive programming	Successive programming is allowed	Erase operation is required before programming. The same byte may be successively programmed only if selective bit programming is used.
Programming time	Minimum 10 ms	Standard mode: minimum 10 ms AUTO mode: maximum 500 μ s
Erasing time	Minimum 10 ms	Standard mode: minimum 10 ms Auto mode: maximum 10 ms

FLASH Functional Description

The MC68HC912DT128A/DG128A devices contain 128 Kbytes of FLASH memory. The memory is divided into four arrays of 32 Kbytes each. Each array consists of windows, which are software selectable to be one page (16 Kbytes) or two pages (32 Kbytes) each.

While the memory is subdivided into four arrays, there are restrictions on how the arrays can be used to hold program/erase execution code. In the 32-Kbyte window configuration, such code cannot be located in FLASH. In the 16-Kbyte window configuration, code can be executed out of FLASH to program/erase the other arrays. But, to do this, the code must be in direct-addressable FLASH, \$4000–\$7FFF or \$C000–\$FFFF. This code cannot program or erase addresses in either of those ranges.

An erased bit in the FLASH reads as a logic 1 and a programmed bit reads as a logic 0. The algorithm programs one row, 32 words (64 bytes), at a time. The erase operation will erase the entire active FLASH array (32 Kbytes). See [FLASH Memory Mapping](#) for a description of the FLASH window ranges.

Program and erase operations are facilitated through control bits in memory-mapped registers. Details for these operations appear later in this application note.

The FLASH memory module and associated registers on the MC68HC912DT128A/DG128A are:

- In 16-K window configuration:
 - \$4000–\$7FFF and \$C000–\$FFFF, direct accessible FLASH space
 - \$8000–\$BFFF, page-addressable FLASH space

NOTE: *The ROMHM bit in the MISC register must be cleared for access to addresses \$4000–\$7FFF. The direct addressable space means that the addresses can be read without using the PPAGE register. However, program and erase operations still require paging.*

- In 32-K window configuration:
 - \$8000–\$FFFF, page-addressable FLASH space

- \$0013, miscellaneous mapping control register, MISC
- \$00F4, FLASH lock control register, FEELCK
- \$00F5, FLASH module configuration register, FEEMCR
- \$00F7, FLASH control register, FEECTL

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

FLASH Memory Mapping

The FLASH memory on the MC68HC912DT128A/DG128A can be configured to different window sizes. As mentioned earlier, four physical arrays of FLASH memory are on the device. A special bit controls whether each array is divided into 16-Kbyte or 32-Kbyte windows.

This function is controlled in a special mapping register which is critical in FLASH operation. This is known as the miscellaneous mapping control register, and it is located at \$0013.

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	ROMTST	NDRF	RFSTR1	RFSTR	EXSTR1	EXSTR0	ROMHM	ROMON
Write:								
Reset:	0	0	0	0	1	1	0	1

Figure 1. Miscellaneous Mapping Control Register (MISC)

Only the bits that relate to FLASH operation will be discussed here. Refer to *MC68HC912DT128A and MC68HC912DG128A Technical Data* for information on device modes and other bits.

ROMTST — FLASH EEPROM Test Mode

This bit determines the memory window configuration for the FLASH. In normal modes, this bit is clear upon reset, but may be changed with a write to the MISC register.

- 0 = FLASH is divided into 16-Kbyte windows from \$8000–\$BFFF.
- 1 = FLASH is divided into 32-Kbyte windows from \$8000–\$FFFF.

ROMHM — FLASH EEPROM Half Map

This bit has no meaning if the ROMON bit is clear. The function of this bit changes depending on the value of the ROMTST bit.

If ROMTST = 0, when ROMHM = :

0 = Page 6 can be accessed at locations \$4000–\$7FFF

1 = Page 6 can NOT be accessed at locations \$4000–\$7FFF.

Page 6 can be accessed at addresses \$8000–\$BFFF using the PPAGE register set to 6.

If ROMTST = 1, when ROMHM = :

0 = There are four distinct page-addressable FLASH memory arrays.

1 = The four FLASH arrays overlap each other. A write or erase to one array will write or erase all four arrays.

ROMON — FLASH EEPROM Enable Bit

This bit is used to enable/disable the FLASH arrays.

0 = FLASH arrays are disabled.

1 = FLASH arrays are enabled.

Table 3 summarizes the effect of these bits on the address ranges and page values for each memory window configuration.

Table 3. Memory Window Ranges

Array	Page	PPAGE	FLASH Control Register Locations	ROMTST Bit in MISC = 0 Windows are 16 K Each See Figure 2 .	ROMTST Bit in MISC = 1 Windows are 32 K Each ⁽¹⁾ See Figure 3 .
00FEE32K	0	000	\$00F4– \$00F7 ⁽²⁾	\$8000–\$BFFF	\$8000–\$FFFF Boot block = \$E000–\$FFFF ⁽³⁾
	1	001		\$8000–\$BFFF Boot block = \$A000–\$BFFF	
01FEE32K	2	010	\$00F4– \$00F7 ⁽²⁾	\$8000–\$BFFF	\$8000–\$FFFF Boot block = \$E000–\$FFFF ⁽³⁾
	3	011		\$8000–\$BFFF Boot block = \$A000–\$BFFF	
10FEE32K	4	100	\$00F4– \$00F7 ⁽²⁾	\$8000–\$BFFF	\$8000–\$FFFF Boot block = \$E000–\$FFFF ⁽³⁾
	5	101		\$8000–\$BFFF Boot block = \$A000–\$BFFF	
11FEE32K	6	110	\$00F4– \$00F7 ⁽²⁾	\$8000–\$BFFF or \$4000–\$7FFF ⁽⁴⁾	\$8000–\$FFFF Boot block = \$E000–\$FFFF ⁽³⁾
	7	111		\$8000–\$BFFF Boot block = \$A000–\$BFFF or \$C000–\$FFFF Boot block = \$E000–\$FFFF	

- Addresses \$4000–\$7FFF are not accessible in this configuration. Also, If both the ROMTST and the ROMHM bits are set, the four memory arrays overlap. A write/erase to any location will be duplicated over all four arrays.
- Each FLASH array has one set of registers. Therefore, you can access the registers with either page value in the PPAGE register.
- If the ROMTST bit is set, then the array can be programmed/erased with the PPAGE register set to either page.
- This memory window only exists if the ROMHM bit in the MISC register is clear. The bit is cleared on reset. In this case, these addresses can be read directly or can be programmed or erased using the PPAGE register.

Application Note

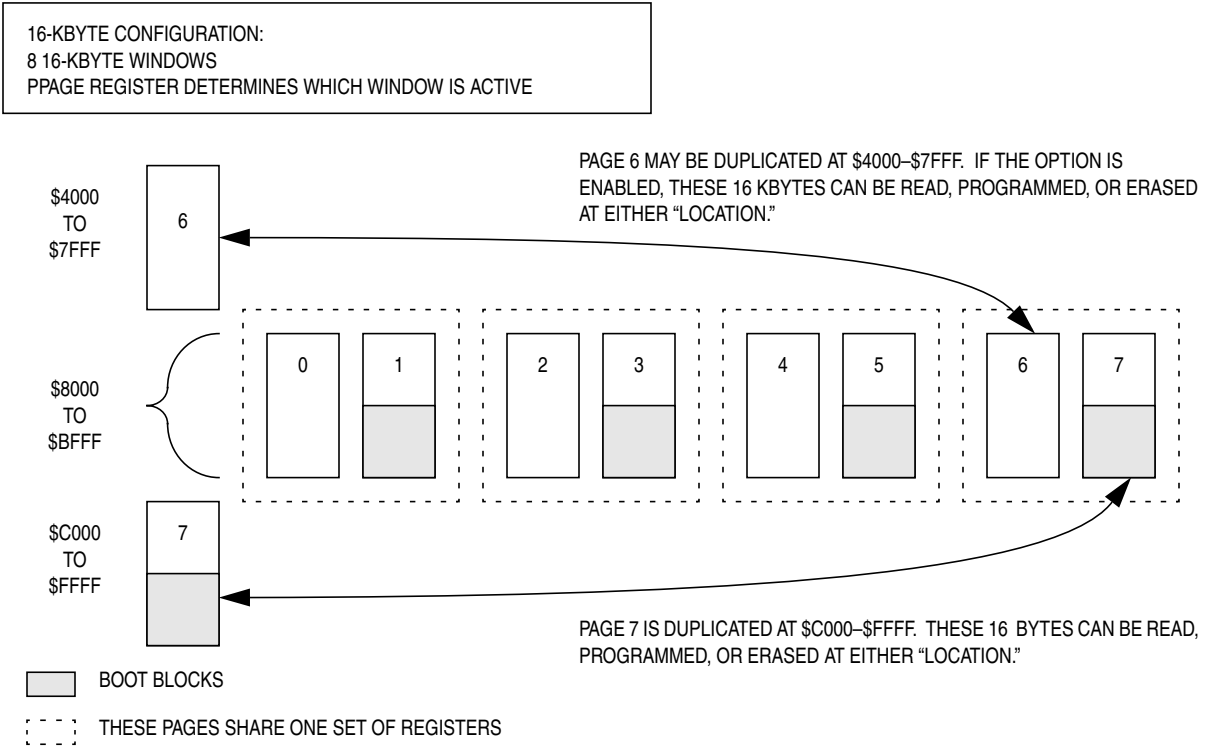


Figure 2. 16-Kbyte Configuration

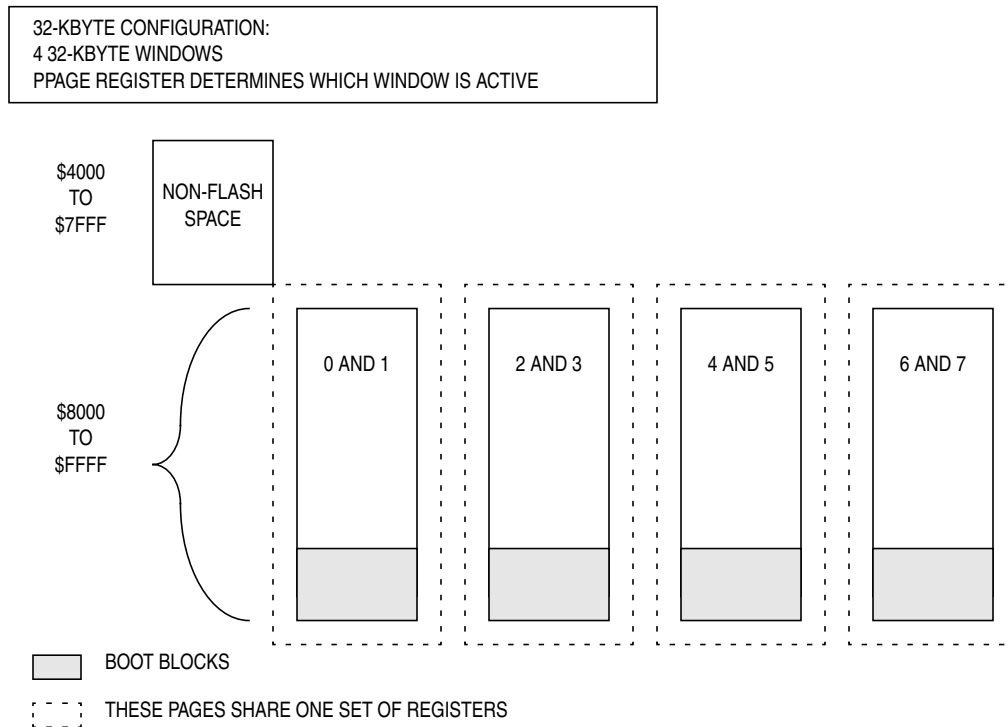


Figure 3. 32-Kbyte Configuration

FLASH Block Protection

Four memory ranges of 8 Kbytes each can be protected from being inadvertently programmed or erased. The protected blocks are located at page addressable addresses of \$A000–\$BFFF or \$E000–\$FFFF depending on how the memory windows are configured.

Table 3 summarizes the protected boot blocks on these devices.

Block protection is controlled by the FLASH module configuration register, FEEMCR, located at \$00F5. See *MC68HC912DT128A and MC68HC912DG128A Technical Data*, for more information on this register.

FLASH Control Registers

Each of the four FLASH arrays has three registers that control its operation:

- \$00F4, FLASH lock control register (FEELCK)
- \$00F5, FLASH module configuration register (FEEMCR)
- \$00F7, FLASH control register (FEECTL)

The PPAGE register must be used to select the correct register set. The user should ensure that the correct FLASH control registers are configured for each memory array. See [Table 3](#) for more information.

NOTE: *There are essentially 12 distinct FLASH control registers.*

These registers are described in detail in *MC68HC912DT128A and MC68HC912DG128A Technical Data*.

Word Alignment

The programming and erasing algorithms include a write to a random address within the array. This write allows the memory to pinpoint the proper physical location for the actual erasing or programming. While the actual data written can be any word, the address written to must be an aligned word.

An aligned word is defined as any 2-byte space starting with an address where the final digit is an even number. That means words starting with an address of the form \$xxx0, \$xxx2, \$xxx4, ..., \$xxxE. A misaligned word has an odd final digit.

If the user specifies a misaligned word in the write step of the erasing algorithm, the memory erase function will not be successful. This restriction also holds true for the programming algorithm, both for the row-selection write (step 2) and also for the actual address being programmed.

FLASH Erase Operation

On the MC68HC912DT128A/DG128A, the entire active FLASH array of 32 Kbytes is erased at once. The protected locations will not be erased unless the BOOTP bit in the FLASH module configuration register (FEEMCR) is cleared first. See [Table 3](#) for boot block locations.

[Figure 4](#) shows the flowchart for the erase operation.

NOTE: *The user should make sure that the proper window is selected in the MISC and PPAGE registers, or other FLASH addresses may be accidentally erased.*

1. Set the ERAS bit in the FLASH control register (FEECTL).
ERAS = 1 configures the FLASH memory for an erase operation.
2. Write any word to any word-aligned FLASH address within the window.
The data written and the address written to are not important.
3. Wait for a time, t_{NVS} .
Internal high voltage is charged.
4. Set the HVEN bit.
Internal high voltage is applied to the window.
5. Wait for a time, t_{ERAS} .
 t_{ERAS} is the erase time.
6. Clear the ERAS bit.
The erase operation is disabled.
7. Wait for a time, t_{NVHL} .
This is the time required for internal high voltage to discharge from the window.
8. Clear the HVEN bit.
Disable the internal high voltage.
9. Wait for a time, t_{RCV} .
After a time, t_{RCV} , the memory can be accessed in normal read mode.

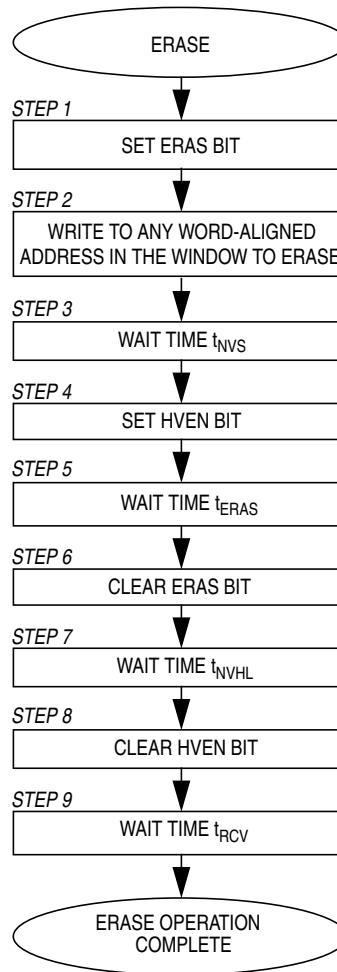


Figure 4. FLASH Erase Operation Flowchart

FLASH Program Operation

On the MC68HC912DT128A/DG128A, programming of the FLASH memory is done on a row-by-row basis, with each programming cycle writing a word (2 bytes).

A row consists of 32 consecutive words (64 bytes) with the following boundaries:

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

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

WARNING: *The FLASH attempts to program in rows. The user should ensure that all of the programmed data fits in one row, or some FLASH addresses may be unintentionally programmed!*

The programming algorithm outlined next specifies some delay times. Take care that exact delay times are used. Excessive program time can result in a program disturb condition, in which case an erased bit on the row being programmed may become unintentionally programmed.

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

Figure 5 shows the flowchart for the programming algorithm.

NOTE: *The user should make sure that the proper window is selected in the MISC and PPAGE registers, or the proper FLASH addresses may not be programmed.*

1. Set the PGM bit in the FLASH control register (FEECTL).
PGM = 1 configures the FLASH memory for a program operation.
2. Write a word of data to any word-aligned FLASH address within the row address range desired.
3. Wait for a time, t_{NVS} .
Internal high voltage is charged.
4. Set the HVEN bit.
Internal high voltage is applied to programming row.
5. Wait for a time, t_{PGS} .
 t_{PGS} is program hold time.
6. Write one data word (2 bytes) to a word-aligned FLASH address to be programmed.
If the BOOTP bit in the FLASH module control register (FEEMCR) for this range is set, an attempt to program the location will be ignored.
7. Wait for a time, t_{FPGM} .
 t_{FPGM} is the 1-word programming time. t_{FPGM} actually includes the total time from step 6 (A on the flowchart) back to step 6 (B on the flowchart) for additional word programming, or from step 6 (A) to step 9 (C on the flowchart) for the last word. This total time must be between 30 and 40 μ s in both cases. Refer to [Figure 5](#).
8. Repeat steps 6 and 7 until all the bytes within the row are programmed.
9. Clear the PGM bit.
Disable the programming operation.
10. Wait for a time, t_{NVH} .
Internal high voltage is discharged from the row.
11. Clear the HVEN bit.
Internal high voltage is disabled.
12. Wait for a time, t_{RCV} .
After a time, t_{RCV} , the memory can be accessed in normal read mode.

While these operations must be performed in the order shown, other unrelated operations may occur between the steps.

Do not exceed t_{FPGM} maximum or t_{HV} maximum. t_{FPGM} is defined in step 7. t_{HV} is defined as the cumulative time that high voltage is applied to the same row before an erase.

$$t_{HV} = t_{NVS} + (t_{FPGM} * (\text{Number of Words Programmed}))$$

This routine assumes that the row being programmed was initially erased.

Note: t_{FPGM} is the total time from **A** to **B** or from **A** to **C**. This time must be between 30 and 40 μs .

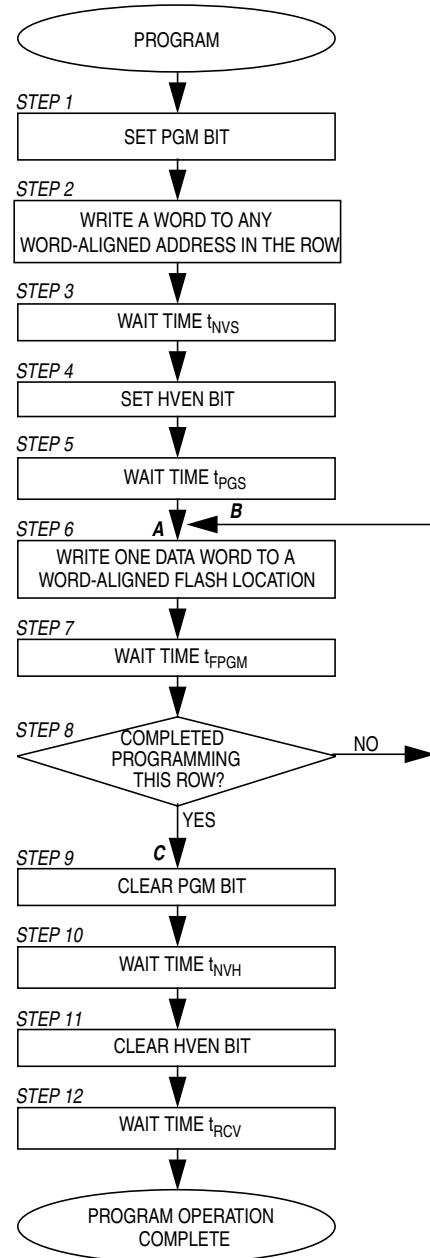


Figure 5. FLASH Program Operation Flowchart

EEPROM Functional Description

The MC68HC912DT128A/DG128A devices contain 2 Kbytes of EEPROM memory. An erased bit in the EEPROM reads as a logic 1 and a programmed bit reads as a logic 0.

The algorithm can program one byte or an aligned word (2 bytes) at a time. The erase operation can be used to erase the entire EEPROM (2 Kbytes), a row (32 bytes), a word (2 bytes), or a single byte at a time. Program and erase operations are facilitated through control bits in memory-mapped registers. Details for these operations appear later in this application note.

The EEPROM memory module and associated registers on the MC68HC912DT128A/DG128A are:

- \$0800–\$0FFF, EEPROM array, 2 Kbytes
- \$00EE, EEPROM modulus divider high register, EEDIVH
- \$00EF, EEPROM modulus divider low register, EEDIVL
- \$00F0, EEPROM module configuration register, EEMCR
- \$00F1, EEPROM block protect register, EEPROT
- \$00F3, EEPROM control register, EEPROG

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

EEPROM Control Registers

The EEPROM register set consists of the five registers listed previously.

The modulus divider registers are used to set the timebase for the EEPROM clock. The module configuration register controls several modes of the device as well as block protection and the SHADOW word. See [Timebase Initialization and SHADOW Word](#) for more information on these registers.

The block protect register will be discussed further in [EEPROM Block Protection](#), and the EEPROM control register is used directly in the erasing and programming algorithms.

These registers are described in detail in *MC68HC912DT128A and MC68HC912DG128A Technical Data*.

EEPROM Block Protection

The block protection feature exists to protect the EEPROM from being inadvertently programmed or erased. The user may specify which regions of the EEPROM to protect using the EEPROM block protect register, EEPROT.

Table 4 summarizes the protection size and addresses for each bit of the EEPROT register.

Table 4. EEPROM Protected Ranges

Bit Name	Protected Addresses	Protected Size
BPROT5	\$0800–\$0BFF	1024 bytes
BPROT4	\$0C00– \$0DFF	512 bytes
BPROT3	\$0E00–\$0EFF	256 bytes
BPROT2	\$0F00–\$0F7F	128 bytes
BPROT1	\$0F80–\$0FBF	64 bytes
BPROT0	\$0FC0–\$0FFF	64 bytes

Once the user has set up block protection, the contents of the block protection register can be locked. To do this, set the PROTLCK bit in the EEPROM module configuration register, EEMCR.

Timebase Initialization and SHADOW Word

To function properly, this new version of EEPROM requires a steady internal clock of 35 μs ($\pm 2 \mu\text{s}$). This clock is divided down from the oscillator clock (EXTAL) by the value in the EEPROM modulus divider registers, EEDIVH and EEDIVL.

Use the following formula to determine the proper divide value.

$$\text{EEDIV} = \text{INT} [\text{EXTAL (Hz)} \times (35 \times 10^{-6}) + 0.5]$$

NOTE: *INT [A] denotes the round-down integer value of A.*

The EEPROM contains a special word location called the SHADOW word (\$0FC0–\$0FC1) which can be used to set up the required timebase. At reset, the value programmed into the SHADOW word is automatically loaded into the EEDIVH:EEDIVL registers. Once the timebase has been set up in the SHADOW word, the user will be able to use the EEPROM without considering clocking. Other than this timebase step, code is backward compatible with the MC68HC912DG128.

To set up the timebase, follow these steps:

1. Calculate the EEDIV value.
2. Write EEDIV into the EEDIVH:EEDIVL registers.
3. Program the SHADOW word to reflect the EEDIV value and the desired values for the EEPROM module configuration register, EEMCR.

WARNING: *When the EEDIV value is set to 0, the EEPGM bit cannot be set.*

The SHADOW word maps into the upper four bits of the EEMCR and 10 bits of the EEDIV registers. See [Table 5](#) for information on SHADOW word mapping.

Table 5. SHADOW Word Mapping

High Byte \$0FC0								
Register	EEMCR	EEMCR	EEMCR	EEMCR	NA	NA	EEDIVH	EEDIVH
Bit	NOBDML	NOSHW	Bit 5	Bit 4	NA	NA	EEDIV9	EEDIV8
Low Byte \$0FC1								
Register	EEDIVL	EEDIVL	EEDIVL	EEDIVL	EEDIVL	EEDIVL	EEDIVL	EEDIVL
Bit	EEDIV7	EEDIV6	EEDIV5	EEDIV4	EEDIV3	EEDIV2	EEDIV1	EEDIV0

Once the timebase and module information has been defined and programmed using the SHADOW word, this location can be protected from unintended programming or erasing. This feature is controlled by the SHPROT bit in the EEPROM block protect register, EEPROT.

EEPROM Erase Operation

On the MC68HC912DT128A/DG128A, erasing can be done on a per-byte, per-word (2 bytes), per-row (32 bytes) or bulk array (2 Kbytes) basis. Word erasing requires the start address to be an aligned word. For more information on word alignment, see [Word Alignment](#).

If some protected locations are included in the erase area, those bytes will not be affected and only the unprotected locations will be erased. Refer to [Table 4](#) for block protection locations.

There are two ways to erase the EEPROM. This can be done using the standard mode erasing algorithm which includes defined delays, or the AUTO mode erasing algorithm can be employed. In the AUTO mode, there are no fixed delays, and instead the EEPGM bit is polled. When the bit is cleared, the erasing has been completed.

EEPROM Standard Mode Erasing Algorithm

Figure 6 shows the flowchart for the standard mode erasing algorithm.

1. Write the BULKP, BYTE, and ROW bits in the EEPROM control register (EEPREG) to specify the erase size. Set the ERASE bit to specify erasing operation. Set the EELAT bit to control erasing latches.

Table 6. Erase Size Selection

BULKP	BYTE	ROW	Block Size
0	0	0	Bulk erase (2 Kbytes)
0	0	1	Row erase (32 bytes)
X ⁽¹⁾	1	X ⁽¹⁾	1 byte or 1 word erase ⁽²⁾

1. X = Don't care

2. If BYTE = 1, then the value of the ROW and BULKP bits are not important. If the value written in step 2 is 1 byte, then the operation will erase 1 byte. If the value written is one word, then the erase operation will erase one word.

2. Write a byte of data to an EEPROM address OR write a word of data to a word-aligned EEPROM address.

If the erase operation is not erasing the entire array or a full row, then this write determines whether a single byte or a word will be erased. Therefore, the address written to must be within the desired erase block.

3. Set the EEPGM bit.

Apply erasing voltage to the EEPROM.

NOTE: *If the value stored in the EEDIV registers is a zero, then the EEPGM bit will not be set.*

4. Wait t_{ERASE} .

t_{ERASE} is the high voltage hold time for erasing.

5. Clear the EEPGM bit.

Disable the erasing voltage from the array.

6. Clear the EELAT bit.

Set the EEPROM into the normal mode.

EEPROM AUTO Mode Erasing Algorithm

Figure 6 shows the flowchart for the AUTO mode erasing algorithm.

1. Write the BULKP, BYTE, and ROW bits in the EEPROM control register (EERPROG) to specify the erase size. Set the ERASE bit to specify erasing operation. Set the EELAT bit to control erasing latches. Set the AUTO bit for automatic erasing time termination. See **Table 6** for a description of the BULKP, BYTE, and ROW bits.
2. Write a byte of data to an EEPROM address or write a word of data to a word-aligned EEPROM address.

If the erase operation is not erasing the entire array or a full row, then this write determines whether a single byte or a word will be erased. Therefore, the address written to must be within the desired erase block.

3. Set the EEPGM bit.

Apply erasing voltage to the EEPROM.

NOTE: *If the value stored in the EEDIV registers is a zero, then the EEPGM bit will not be set.*

4. Poll the EEPGM bit until it is cleared by the internal timer.
5. Clear the EELAT bit.

Set the EEPROM into the normal mode.

NOTE: *In AUTO mode, be careful about erase attempts on protected areas. If the erase area (byte, word, row) is a protected area, the erasing will not be successful and the EEPGM bit will never clear. The user may include a step to verify that the addresses in question are not protected, or include a timer to ensure that the software does not get trapped in that step. A bulk erase, even if some of the memory areas are protected, WILL result in unprotected memory areas being erased.*

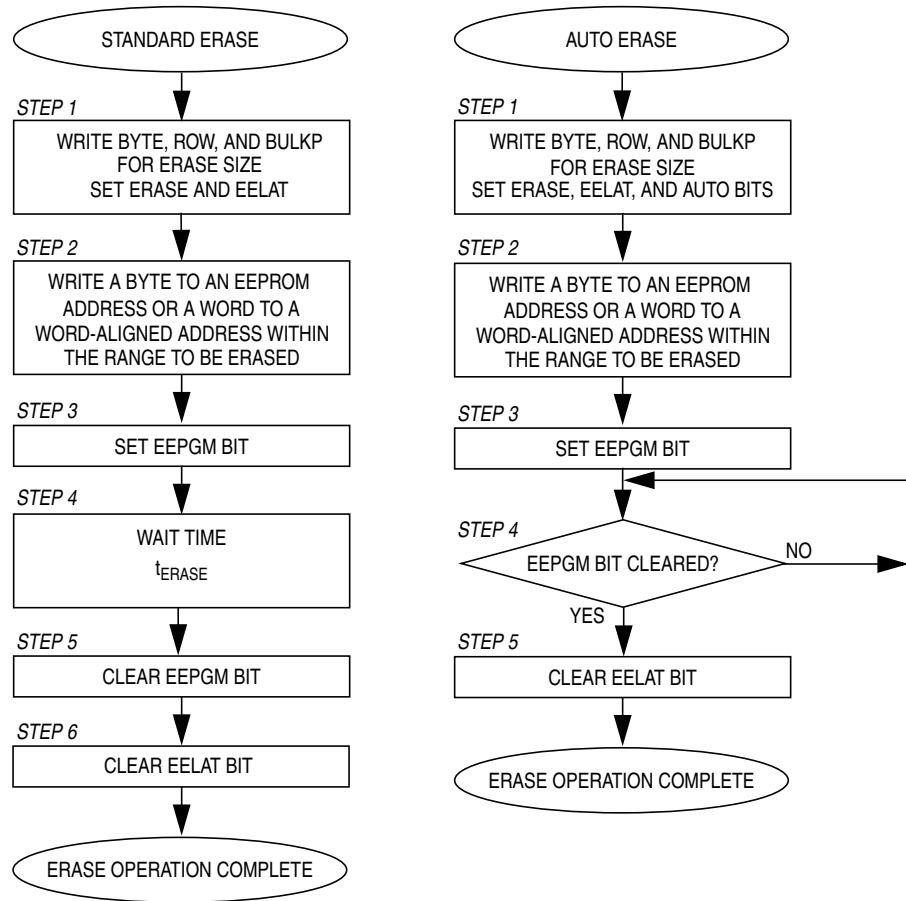


Figure 6. EEPROM Erasing Algorithm Flowcharts

EEPROM Program Operation

On the MC68HC912DT128A/DG128A, programming can be done on a per-byte or per-aligned-word (2 bytes) basis. For more information on aligned words, see [Word Alignment](#).

If some protected locations are included in the program area, those bytes will not be affected and only the unprotected locations will be altered. See [Table 4](#) for block protection locations.

There are two ways to program the EEPROM. This can be done using the standard mode programming algorithm which includes defined delays, or the AUTO mode programming algorithm can be employed. In the AUTO mode, there are no fixed delays, and instead the EEPGM bit is polled. When the bit is cleared, the programming has been completed.

EEPROM Standard Mode Programming Algorithm

Figure 7 shows the flowchart for the standard mode programming algorithm.

1. Clear the ERASE bit to specify programming operation. Set the EELAT bit to control programming latches.
2. Write a byte of data to an EEPROM address or write a word of data to a word-aligned EEPROM address.

The size of this write determines whether a single byte or a word will be programmed. Make sure that the address is a byte or an aligned word.

3. Set the EEPGM bit.

Apply programming voltage to the EEPROM.

4. Wait t_{PROG} .

t_{PROG} is the high voltage hold time for programming.

5. Clear the EEPGM bit.

Disable the programming voltage from the array.

6. Clear the EELAT bit.

Set the EEPROM into the normal mode.

EEPROM AUTO Mode Programming Algorithm

Figure 7 shows the flowchart for the AUTO mode programming algorithm.

1. Clear the ERASE bit to specify programming operation. Set the EELAT bit to control programming latches. Set the AUTO bit for automatic programming time termination.
2. Write a byte of data to an EEPROM address or write a word of data to a word-aligned EEPROM address.

The size of this write determines whether a single byte or a word will be programmed. Make sure that the address is a byte or an aligned word.

3. Set the EEPGM bit.
Apply programming voltage to the EEPROM.
4. Poll the EEPGM bit until it is cleared by the internal timer.
5. Clear the EELAT bit.

Set the EEPROM into the normal mode.

NOTE: *In AUTO mode, if the programming block is a protected area, the programming will not be successful and the EEPGM bit will never clear. The user may include a step to verify that the addresses in question are not protected, or include a timer to ensure that the software does not get trapped in that step.*

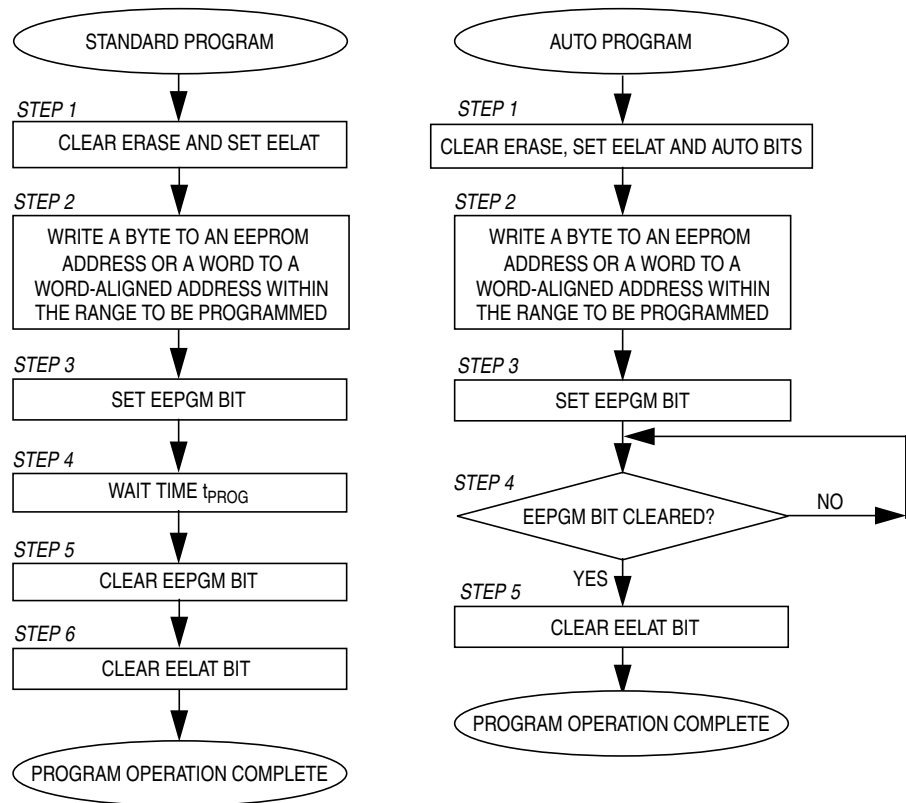


Figure 7. EEPROM Programming Algorithm Flowcharts

Selective Bit Programming

On each programming cycle, one to eight bits of the EEPROM memory may be programmed. It is possible to program multiple bits at the same time. However, the same bit may not be programmed twice unless the entire byte has been erased first. This means that the same byte location may be programmed eight times as long as an individual bit is not written to more than once. This is referred to as selective bit programming. The acceptable sequence in [Table 7](#) shows how the same byte may be used for eight program cycles without an erase.

Table 7. Selective Bit Programming

Acceptable Sequence			Unacceptable Sequence		
Operation	Program Data	Value in Memory	Operation	Program Data	Value in Memory
Erase	NA	1111:1111	Erase	NA	1111:1111
Write \$FE	1111:1110	1111:1110	Write \$FE	1111:1110	1111:1110
Write \$FD	1111:1101	1111:1100	Write \$F9	1111:1001	1111:1000
Write \$FB	1111:1011	1111:1000	Write \$EF	1110:1111	1110:1000
Write \$F7	1111:0111	1111:0000	Write \$D8	1101:1000	Unknown
Write \$EF	1110:1111	1110:0000			
Write \$DF	1101:1111	1100:0000			
Write \$BF	1011:1111	1000:0000			
Write \$7F	0111:1111	0000:0000			
Erase	NA	1111:1111			

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

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

Practical Considerations for Programming/Erasing

To ensure successful programming and erasing of the FLASH and/or EEPROM on the MC68HC912DT128A/DG128A, the user should consider the following suggestions:

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

Evaluating Delay Times for the Sample Code

The FLASH algorithm include specific delay steps. These delay times are defined in the MC68HC912DT128A/DG128A specification and must be considered when utilizing the algorithm.

To ensure that each delay step meets the specification, the delay times have to be evaluated. All delays in the sample code provided in this application note were evaluated and confirmed to meet the specification using a general I/O port pin. The port pin was initialized with a high output signal. Just before entering a delay, the port pin was toggled low and held low until the end of the delay.

The actual delay times shown in [Table 8](#) and [Table 9](#) were the low periods as measured on an oscilloscope. To toggle the port pin, instructions BSET, BCLR, and COM were used. The sample code also includes the instructions used for the delay time evaluation.

Table 8. FLASH Erase Delay Times

Name of Delay	Specified Duration	Calculated Delay Time ⁽¹⁾	Actual Delay Time	Delta ⁽²⁾
t _{NVS}	> 10 μs	10.25 μs	10.8 μs (S1 to E1) ⁽³⁾	0.55 μs
t _{ERAS}	> 8 ms	8.0 ms	8.0 ms (S2 to E2) ⁽³⁾	0 μs
t _{NVHL}	> 100 μs	100.25 μs	101.2 μs (S3 to E3) ⁽³⁾	0.95 μs
t _{RCV}	> 1 μs	1.25 μs	1.76 μs (S4 to E4) ⁽³⁾	0.51 μs

1. Delay time calculated by dividing the number of cycles in the delay by the bus speed of 8 MHz.
2. Most delta times are around 0.5 μs. Since the instructions BSET and COM require four internal bus cycles, this additional time comes from the execution time of the instruction.

$$\frac{1}{8.0 \text{ MHz}} \times 4 \text{ cycles} = 0.5 \mu\text{s}$$
3. These times refer to measured delays based on running the attached sample code. In that code, the points at which measurements were taken are defined by these markers.

Table 9. FLASH Program Delay Times

Name of Delay	Specified Duration	Calculated Delay Time ⁽¹⁾	Actual Delay Time	Delta ⁽²⁾
t _{NVS}	> 10 μs	10.25 μs	10.76 μs (S5 to E5) ⁽³⁾	0.51 μs
t _{PGS}	> 5 μs	5.37 μs	5.88 μs (S6 to E6) ⁽³⁾	0.505 μs
t _{FPGM}	30 μs–40 μs	30.6 μs ⁽⁴⁾	31.20 μs (S7 to E7) ⁽³⁾	0.6 μs
		30.25 μs ⁽⁵⁾	30.80 μs (S8 to E8) ⁽³⁾	0.55 μs
t _{NVH}	> 5 μs	5.375 μs	5.88 μs (S9 to E9) ⁽³⁾	0.505 μs
t _{RCV}	> 1 μs	1.25 μs	1.76 μs (S10 to E10) ⁽³⁾	0.51 μs

1. Delay time calculated by dividing the number of cycles in the delay by the bus speed of 8 MHz.
2. Most delta times are around 0.5 μs. Since the instructions BSET and COM require four internal bus cycles, this additional time comes from the execution time of the instruction.

$$\frac{1}{8.0 \text{ MHz}} \times 4 \text{ cycles} = 0.5 \mu\text{s}$$
3. These times refer to measured delays based on running the attached sample code. In that code, the points at which measurements were taken are defined by these markers.
4. The word programmed is NOT the last word of the row being programmed (A to B on [Figure 3](#)).
5. The word programmed IS the last word of the row being programmed (A to C on [Figure 3](#)).

FLASH Frequently Asked Questions

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

Question 1

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

Answer 1

Check the following:

- *Are the FLASH arrays enabled?*

The FLASH is enabled/disabled by the ROMON bit in the miscellaneous system control register, MISC. Ensure that the ROMON bit is set to write to these addresses.

- *Is each step of the programming algorithm (or erasing algorithm) performed in the right order?*

The sequence of the program and erase operations is interlocked in hardware so only the prescribed order of these operations will allow erase/program operations. However, other non-FLASH operations may occur between the steps shown.

- *Is the memory block where you want to program/erase unprotected?*

The block protect feature of the FLASH is present to prevent unintentional programming or erasing. The block protect bits must be cleared in the right page such that the memory to be erased or programmed is unprotected.

- *Are delay times (t_{PGS} , t_{ERAS} , t_{NVHL} , t_{NVS} , t_{RCV} , t_{NVH} , t_{FPGM}) 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. Motorola does not guarantee FLASH performance if all timing requirements are not being adhered to.

- *Is the correct FLASH register being written to enable erase or program?*

The MC68HC912DT128A/DG128A has four FLASH arrays and four separate sets of control and block protect registers. Make sure the appropriate register set is being addressed in the PPAGE register. Refer to [FLASH Control Registers](#).

- *Is the COP enabled?*

The COP is the computer operating properly timer that periodically checks the device for proper operation. If the COP is enabled, the COPRST register has to be written periodically (with the values \$55 then \$AA) to prevent a COP reset. To avoid issues, make sure that the selected COP period is long enough that the COP feeding process is not performed during the program/erase operation or disable the COP entirely during this operation.

NOTE: *The COP is always enabled out of RESET in normal modes and can be disabled by modifying the COPCTL register.*

- *Was Motorola's recommended programming algorithm (or erasing algorithm) used in your code?*

The recommended programming algorithm ensures that the FLASH is programmed for sufficient data retention with a minimum program time. Not following this algorithm can lead to overprogramming, which risks program disturb.

Question 2

I wanted to erase only vector locations using the page erase operation, but memory addresses were also erased. Did I do anything wrong?

Answer 2

No. The vector locations (\$FF00–\$FFFF) are located in the FLASH. On the MC68HC912DT128A/DG128A, the minimum erase size is 32 Kbytes. In 16-Kbyte configuration, memory addresses \$4000 to \$7FFF and \$C000–\$FFFF will all be erased at one time. In 32-Kbyte configuration, all bytes from \$8000–\$FFFF in the active page will be erased.

Question 3 What is the FLASH charge pump?

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

Question 4 The MC68HC912DT128A/DG128A FLASH programs one row (64 bytes) at a time. Do I always have to program the entire row?

Answer 4 No, it is not necessary to program the entire row. If you include a test to make sure that programming stops when the row boundary ends, then addresses which are not programmed are left as they were before the row programming was started. Be careful to not allow the code to attempt a program beyond the end of the row; the other addresses may be overwritten. Also note that, before reprogramming any additional bytes in this row, the entire page must be erased.

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

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

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

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

Answer 6 No. See question 7.

Application Note

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

Answer 7 Yes. The MC68HC912DT128A/DG128A has four FLASH memory arrays. IF the memory is configured into 16-Kbyte windows (ROMTST bit = 0), array 11FEE32K can be used to program or erase code in the other arrays. The code must be in addresses \$4000–\$7FFF or \$C000–\$FFFF.

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

Answer 8 Yes and no. While each array does have a separate charge pump, the address decode logic does not allow more than one row to be programmed at a time. However, in the 32-Kbyte window configuration, the memory can be set such that the four memory arrays overlap. This is done by setting the ROMHM bit in the MISC register. In this case, all memory arrays are programmed at the same time with the same data. For more information on this feature, refer to the Resource Mapping Section of *MC68HC912DT128A and MC68HC912DG128A Technical Data*.

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

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

Question 10 I'm sending external data into the MC68HC912DT128A/DG128A for programming. How can I speed up this programming process?

Answer 10 Excluding data download time, it takes about two seconds to program 128 Kbytes of FLASH. If data is transferred at a higher communication baud rate or in a parallel manner, the overall programming time can be reduced. The ideal situation would be a fast parallel transfer of data during the delay times associated with the programming algorithm.

Question 11 Do I need to confirm the memory contents after programming the FLASH?

Answer 11 It is recommended that the code used to program the FLASH also include a verification step to ensure the integrity of the data programmed into the FLASH.

Question 12 A block of memory in the FLASH array is protected by programming the block protect register. When I execute an erase operation, will the unprotected block be erased?

Answer 12 Yes. When a FLASH array is partially protected, the erase operation erases all non-protected bytes, and leaves the protected bytes as they were before erasing.

Question 13 What is the expected lifetime of FLASH memory?

Answer 13 The minimum program/erase endurance and data retention lifetime of the FLASH memory for all conditions is found in *MC68HC912DT128A and MC68HC912DG128A Technical Data*.

Question 14 What steps can I take to prolong the life of the FLASH memory?

Answer 14 The FLASH memory has a finite program/erase durability and a finite data retention lifetime. However, the specification shows the minimum lifetime considering the worst case set of conditions applied to the part. In general, the FLASH will last longer if it is used at moderate temperatures (0–70°C) and the program/erase cycles are kept to a minimum.

Application Note

Question 15 The programming algorithm includes two write steps. Do both writes need to use aligned word addresses?

Answer 15 Yes. The programming algorithm requires writes to occur to aligned words. This restriction applies to both step 2 and step 6 of the programming algorithm.

Question 16 What modes of operation cause the most noise?

Answer 16 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.

Question 17 The memory is configured to use 32-Kbyte windows. What value should I use in the PPAGE register to program/erase the array?

Answer 17 There are two pages per memory array. You can set the PPAGE register to either page value to program/erase the associated array block when in 32-K configuration. See [Table 3](#) for more information on the memory mapping.

Question 18 I am using the memory in 16-Kbyte window configuration, but I want to write to address \$4000–\$7FFF (page 6) directly. This function is not working. Why?

Answer 18 Direct access to address locations \$4000–\$7FFF is controlled by the ROMHM bit in the miscellaneous system control register, MISC. Ensure that the ROMHM bit is clear to write to these addresses. Note that programming and erasing these locations requires setting the PPAGE register to 6.

Question 19 In 32-Kbyte configuration, there are four memory arrays and four sets of registers that must be considered. In 16 Kbyte configuration, are there eight sets of registers?

Answer 19 No. There are always four physical arrays of FLASH on these devices, and each array has a set of registers. In 16-Kbyte configuration, the registers are shared between Pages 0&1, 2&3, 4&5, and 6&7. A write to the registers will modify their contents with either page being set in the PPAGE register. See [FLASH Memory Mapping](#) for more information.

Question 20 I attempted to program/erase a section of memory, but a different section of memory was programmed/erased instead. What did I do wrong?

Answer 20 The MC68HC912DT128A/DG128A has 128 Kbytes of FLASH memory, but they require page accessing to program. Not only does the proper address have to be used in the programming or erasing algorithm, but the PPAGE register must be pointing to the desired page. See [Table 3](#) for more information.

Question 21 The memory is configured to 16-Kbyte windows. Will the erase function erase the active 16-Kbyte window, or will it erase the entire 32-Kbyte array that the active window is in?

Answer 21 The entire 32-Kbyte window will be erased, unless the boot block protection bit is set. In that case, the non-protected bits, the top 24 Kbytes of the array, will be erased.

Question 22 The erasing algorithm uses a write to determine which array to erase. Does it matter which address I use for that write?

Answer 22 As long as the address written to is an aligned word, it can be any address within the array. See [Word Alignment](#) for more information.

EEPROM Frequently Asked Questions

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

Answer 1 Check the following:

- *Is the EEPROM array enabled?*

The EEPROM is enabled/disabled by the EEON bit in the Initialization of Internal EEPROM position register, INITEE. Ensure that the EEON bit is set to write to these addresses.

- *Did you program correct divider values in the EEDIV registers (EEDIVH and EEDIVL)?*

The EEDIVH and EEDIVL registers are loaded from the SHADOW word (\$0FC0–\$0FC1) at reset. You must first calculate the desired EEDIV value for the crystal frequency being used. After that, you can either write that value to EEDIVH:EEDIVL for temporary operation, or program that value into the SHADOW register and reset the part. All of this must be done before executing the erase or program operation. In normal mode, the EEDIV registers are write-once registers. See [Timebase Initialization and SHADOW Word](#) for more information.

- *Did you use our recommended programming and erasing algorithms in your code?*

The EEPROM consists of FLASH memory surrounded by a logic state machine. Motorola's recommended programming algorithm ensures that the EEPROM is programmed for sufficient data retention and in a minimum program time. Motorola does not guarantee the performance of the EEPROM if the recommended algorithms are not followed.

- *Is each step of the programming and erasing algorithms performed in the right order?*

The sequence of the program and erase operations are interlocked in hardware so only the prescribed order of these operations can occur. However, other non-EEPROM operations may occur between the steps shown.

- *Is the memory block where you want to program/erase unprotected?*

The block protect feature of the EEPROM is present to prevent unintentional programming or erasing. The block protect bits must be cleared such that the memory to be erased or programmed is unprotected.

- *Are delay times (t_{PROG} , t_{ERASE}) within the specification?*

In standard mode, timing is critical to ensure proper EEPROM operation. Delay times that are too long or too short can alter the EEPROM performance to the point where it does not work or is not reliable. Motorola does not guarantee EEPROM performance if the wrong delay times are used.

- *Is the array mapped where you expect it?*

The EEPROM may be mapped to any 4-Kbyte boundary within the address space. The upper four bits of the initialization of internal EEPROM position register, INITEE, sets up the EEPROM address block. These bits are clear on reset.

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

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

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

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

Application Note

- Question 4** During a program/erase process, can I execute an interrupt service or include additional steps?
- Answer 4* Unrelated (non-EEPROM) steps may be included between steps of the program/erase algorithms as long as the sequence of the steps remains consistent. However, interrupt service routines can cause errors in the program or erase timing and lead to corrupt or missing data in the EEPROM. Motorola does not guarantee performance of the EEPROM if interrupts are not masked during the program or erase operation.
- Question 5** Can I program each bit in the same EEPROM location successively?
- Answer 5* No. However, the same byte location can be successively programmed using selective bit programming. Refer to [Selective Bit Programming](#).
- Question 6** Is one charge pump used for both EEPROM and the FLASH arrays?
- Answer 6* No. Each FLASH array has a separate charge pump, and the EEPROM has its own charge pump.
- Question 7** Do I need to confirm the memory contents after programming the EEPROM?
- Answer 7* It is recommended that the code used to program the EEPROM also include a verification step to ensure the integrity of the data programmed.
- Question 8** What is the expected lifetime of EEPROM memory?
- Answer 8* The minimum program/erase endurance and data retention lifetime of the EEPROM memory for all conditions is found in *MC68HC912DT128A and MC68HC912DG128A Technical Data*.

Question 9 What steps can I take to prolong the life of the EEPROM memory?

Answer 9 The EEPROM memory has a finite program/erase durability and a finite data retention lifetime. However, the specification quotes the minimum guaranteed lifetime considering the worst case set of conditions applied to the part. In general, the EEPROM array will last longer if the program/erase cycling is kept to a minimum and the temperature is kept at a nominal level (0–70°C).

Question 10 Can I program/erase the EEPROM at the maximum temperature limits for the specified life of the part?

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

Question 11 I have calculated the EEDIV value and tried to program it in the SHADOW word locations. Why is the value not programming into this location?

Answer 11 This may have several causes. For instance:

- What is the value of the SHPROT bit in the EEPROM block protect register, EEPROT? If this bit is set, the SHADOW word is protected from programming and erasing.
- Have you reset the part after programming the SHADOW word? The SHADOW word is loaded into the EEDIV register only on reset.
- Are you trying to read the SHADOW word at locations \$0FC0–\$0FC1? These EEPROM locations are mapped to the SHADOW word ONLY when the NOSHW bit in the EEPROM module configuration register, EEMCR, is clear.

- Question 12** I cleared the EEPROM block protection register, EEPROT. Why am I still not able to program all the EEPROM addresses?
- Answer 12* While you may have executed an erase on the register, the erase function may not have been successful. First check the value of the PROTLCK bit in the EEPROM module configuration register, EEMCR. If the PROTLCK bit is set, then the EEPROM block protect register cannot be changed.
- Question 13** Can I ignore the SHADOW word and directly program the EEDIV value into the EEDIVH and EEDIVL registers?
- Answer 13* Yes. In normal modes, these are write-once registers (the EEPROM latch control, EELAT, must be off). In special modes, these registers may be written at any time, with the same condition as mentioned earlier.
- Question 14** Can the AUTO bit be set at the same time as other bits in the EEPROM control register, EEPROG?
- Answer 14* Yes. The sample code sets the AUTO bit at the same time that the BULKP, BYTE, ROW, ERASE, and EELAT bits are written. EEPGM can NOT be set at the same time.
- Question 15** What happens if you try to erase/program a protected range of EEPROM?
- Answer 15* The unprotected areas change, but the protected areas are unaffected.
- Question 16** What happens when the SHADOW word is not used? What happens to locations \$0FC0 and \$0FC1? How do you program the divider value?
- Answer 16* First of all, the SHADOW word is a distinct place on the device. When the SHADOW word is enabled via the NOSHW bit in the EEPROM module configuration register, the EEPROM addresses \$0FC0 and \$0FC1 map to the SHADOW word. When the SHADOW word is disabled, the locations are normal EEPROM locations.

Whether the location is enabled or not, reset still loads the SHADOW word into the EEDIVH:EEDIVL registers. For temporary operation, change the divider value by writing directly to those registers. For long-term operation, if you want to use these locations as normal EEPROM, then you will need to execute a sequence where the SHADOW word is enabled, programmed, and disabled.

Question 17 Even if the SHADOW word is enabled, can I still just write the divider value to the EEDIVH:EEDIVL registers once the part is out of reset?

Answer 17 Yes. In normal modes, these registers are once-write registers, but in special modes, they may be written at any time.

FLASH Assembly Source Code Flowcharts

The main routine in `SSTflash.mrt` initializes the device for erasing and programming operations before calling the subroutines themselves. The routine starts by setting up the FLASH as desired and filling a RAM data buffer with values to program into the array. It then calls the `FlashErase` subroutine which follows the algorithm listed in this application note. After erasing an array of data, the main routine calls `ProgRow` to program a row (64 bytes). `SSTflash.mrt` also includes a verification step after the programming is completed.

The `FlashErase` and the `ProgRow` subroutines follow the flowcharts shown in [Figure 4](#) and [Figure 5](#) closely. A flowchart is also included for the `ms_delay` subroutine which generates delays greater than 1 millisecond.

NOTE: *These routines must be executed at an 8-MHz bus frequency to meet expected delay times.*

The flowcharts for `SSTflash.mrt`, `FlashErase`, `ProgRow`, and `ms_delay` are [Figure 8](#), [Figure 9](#), [Figure 10](#), and [Figure 11](#), respectively.

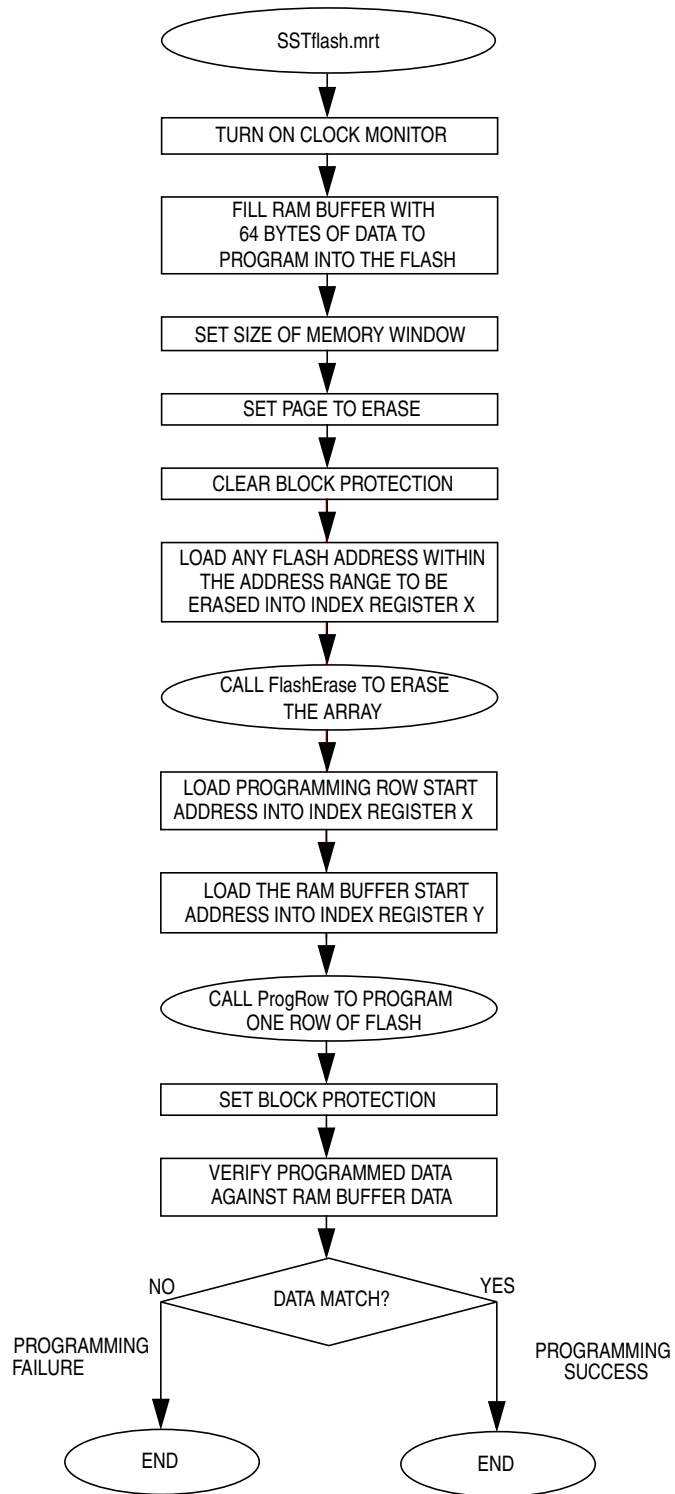


Figure 8. FLASH Main Routine Flowchart

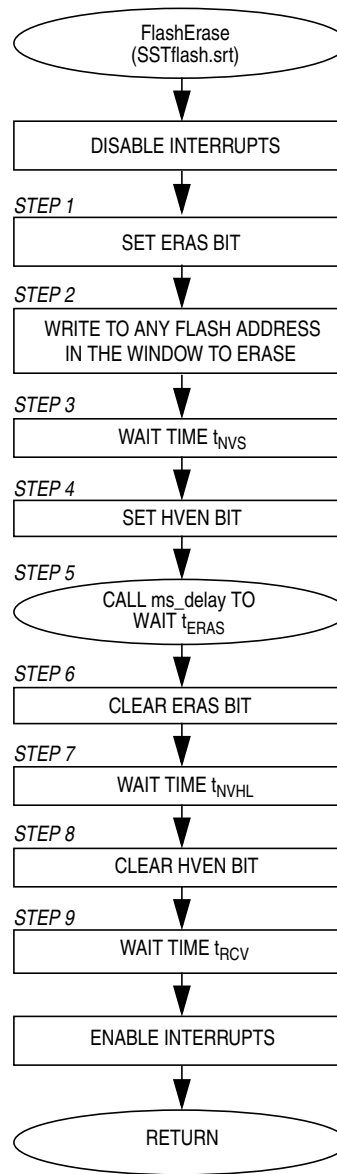


Figure 9. Subroutine FlashErase Flowchart

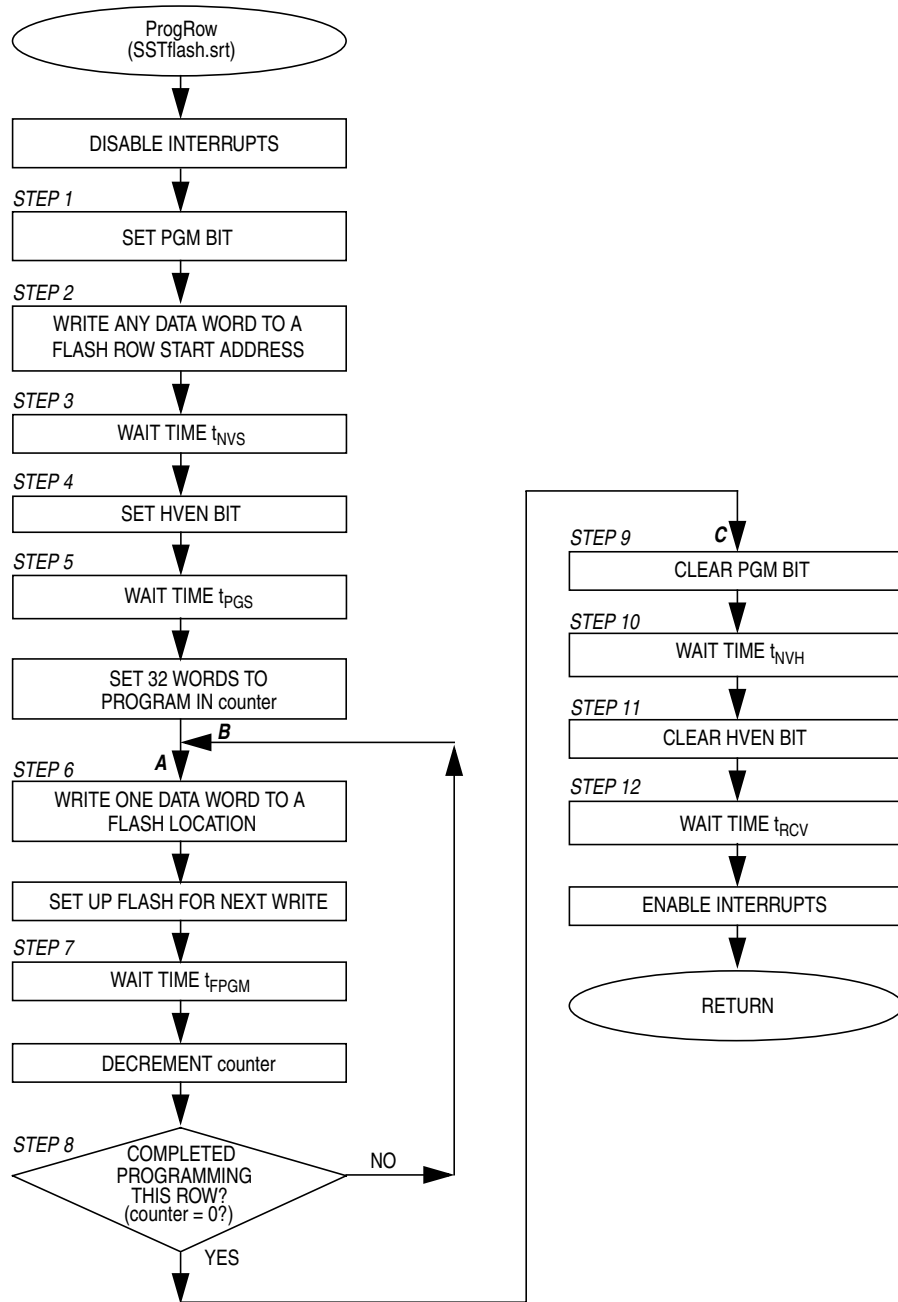


Figure 10. Subroutine ProgRow Flowchart

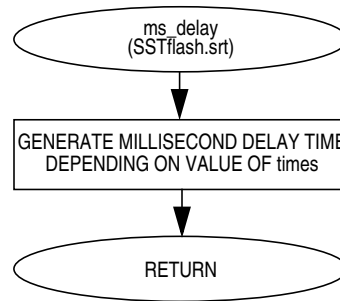


Figure 11. Subroutine Delay Flowchart

FLASH Assembly Source Code

```

*****
*****
*
* SST FLASH Memory Programming and Erasing on the MC68HC912DT128A/DG128A *
*
*****
* File Name: SSTflash.mrt Copyright (c) Motorola 2001 *
*
* Current Revision: 1.0 *
* Current Release Level: RP *
* Current Revision Release Date: July 6, 2001 *
*
* Current Release Written By: Kazue Kikuchi and Darci Ernst *
* Motorola Applications Engineering - Austin, TX *
*
* Assembled Under: CASM012Z (P&E Microcomputer Systems, Inc.) *
* Ver.: 3.11 *
*
* Part Family Software Routine Works With: HC12 0.5u Flash Memory *
*
* Routine Size (Bytes): 204 *
* Stack Space Used (Bytes): 4 *
* RAM Used (Bytes): 66 *
* Global Variables Used: DATA *
* Subroutine Called: FlashErase, ProgRow *
*
* Full Functional Description Of Routine Design: *
* SSTflash.mrt is the main routine for the Flash programming and *
* erasing operations on the MC68HC912DT128A and the MC68HC912DG128A. *
* This main routine calls on programming and erasing algorithms in *
* the SSTflash.srt file. This SST programming algorithm minimizes *
* the amount of time needed to program a row of FLASH memory. One *
* row consists of 64 consecutive bytes of FLASH memory within *
* specified address ranges. Before the programming operation is *
* executed, the 64 bytes of programming data are stored in the RAM *

```

Application Note

```
*      buffer. *
*      Note: This routine must be executed at a bus frequency of 8MHz *
*      because each delay time in the algorithm was calculated *
*      with this bus frequency. *
*****
* Motorola reserves the right to make changes without further notice to *
* any product herein. Motorola makes no warranty, representation or *
* guarantee regarding the suitability of its products for any particular *
* purpose, nor does Motorola 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.*
* Motorola does not convey any license under its patent rights nor the *
* rights of others. Motorola 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 *
* Motorola product could create a situation where personal injury or death*
* may occur. Should Buyer purchase or use Motorola products for any such *
* intended or unauthorized application, Buyer shall indemnify and hold *
* Motorola 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 Motorola was negligent*
* regarding the design or manufacture of the part. Motorola and the *
* Motorola symbol are registered trademarks of Motorola, Inc. Motorola, *
* Inc. is an Equal Opportunity/Affirmative Action Employer. *
*****
*****
*****      Include Files      *****
*****
NOLIST
$INCLUDE      "912DG128A_memory.frk"      ;Equates for the MC68HC912DG/DT128A
; registers and bits used in this
; routine
      org      RAM
$INCLUDE      "SSTflash.var"      ;RAM variable definitions
LIST
*****
*****      Main Routine      *****
*****
      org      RAM+$50      ;The code start address
;The RAM below this address is used
Start:      ; as the FLASH data buffer and spare data
; storage

      lds      #$4000      ;Set Stack Pointer

      movb     #$8F,COPCTL      ;Enable Clock Monitor Function
; If a loss of clock is detected, take
; appropriate action
```



```

        ldx    #$00
        ldaa   #$1                ;Fill the RAM buffer with 64 bytes
Data_load:                               ; data to program into FLASH
        staa  DATA,x            ; (ie. 01,02,03,.....,3E,3F,40)
        inca
        inx
        cmpa  #!65
        bne   Data_load

;      movb   #$81,MISC           ;Select a 32K Window
                                       ; Select either a 16K Window or a
                                       ; 32K Window. Default is 16K. NOTE: This
                                       ; register also controls narrow data bus
                                       ; and stretch bits.

        movb   #$01,PPAGE        ;Select PPAGE=1
                                       ; Select a desired page

        bclr  FEEMCR,BOOTP.      ;If Boot Block in this page is
                                       ; protected, clear BOOTP bit.
                                       ; NOTE: FEELCK ($00F4) must = 0.

        ldx   #$8002             ;Load Index Register X with any address
                                       ; within the page. The address with an
                                       ; aligned word has to be selected

        jsr   FlashErase        ;Erase the whole selected page

        ldx   #$BF40            ;Load Index Register X with a programming
                                       ; Row start address ($xx00, $0xx40, $0080,
                                       ; or $xxC0)

        ldy   #DATA             ;Load Index Register Y with a RAM buffer
                                       ; start address
        jsr   ProgRow           ;Program a Row (32 words)

        bset  FEEMCR,BOOTP.     ;Set BOOTP bit to protect the Boot Block

Verify:
        movb  #!32,COUNTER      ;After the desired block is programmed,
                                       ; it is recommended that programmed data
                                       ; be verified

        ldy   #DATA             ;Load Index Register Y with a RAM buffer
                                       ; start address

        ldx   #$BF40           ;Load Index Register X with a verifying
                                       ; Row start address

Verify_Loop:
        ldd   ,X                ;Read data from a FLASH location
        cpd   ,Y                ;Compare the read data with data in the
        bne   Error             ; RAM Buffer
        dec   COUNTER           ;When the verify fails, branch to Error
        beq   Success
        inx
        inx
        iny

```

Application Note

```
    iny
    bra   Verify_Loop

Success:
    bra   *                ; ** Programming Successful **
                          ; End of program
Error:
    bra   *                ; ** Programming Failed **
                          ; Take appropriate action

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

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
*****
*****
*
* SST FLASH Memory Programming and Erasing on the MC68HC912DT128A/DG128A *
*
* File Name: SSTflash.var              Copyright (c) Motorola 2001 *
*
* Current Revision: 1.0                *
* Current Release Level: RP            *
* Current Revision Release Date: July 6, 2001 *
*
* Current Release Written By: Kazue Kikuchi and Darci Ernst *
*                               Motorola Applications Engineering - Austin, TX *
*
* Assembled Under: CASM12Z (P&E Microcomputer Systems, Inc.) *
*                               Ver.: 3.11 *
*
* Part Family Software Routine Works With: HC12 0.5u Flash Memory *
*
* RAM Used (Bytes): 66                *
*
* Description:
*   RAM variable definitions for the main routine SSTflash.mrt. *
*****
*****           RAM Variables           *****
*****
DATA          rmb   !64           ;64 data bytes that will be programmed
COUNTER       rmb   $1           ;1 byte storage of the word number
                          ; contained in a row
TIMES         rmb   $1           ;1 byte in which the delay time will be
                          ; determined

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

```

* SST FLASH Memory Program & Erase Subroutines on the MC68HC912DT/DG128A *
*
*****
* File Name: SSTflash.srt                      Copyright (c) Motorola 2001 *
*
* Current Revision: 1.0                        *
* Current Release Level: RP                    *
* Current Revision Release Date: July 6, 2001 *
*
* Current Release Written By: Kazue Kikuchi and Darci Ernst *
*                               Motorola Applications Engineering - Austin, TX *
*
* Assembled Under: CASM12Z (P&E Microcomputer Systems, Inc.) *
*                               Ver.: 3.11 *
*
* Part Family Software Routine Works With: HC12 0.5u Flash Memory *
*
*
* Module Size (Bytes):          FlashErase      43 *
*                               ProgRow         62 *
*                               ms_delay        18 *
* Stack Space Used (Bytes):     FlashErase      2 *
*                               ProgRow         2 *
*                               WriteFLCR       0 *
*                               ms_delay        0 *
* RAM Used (Bytes):            FlashErase      1 *
*                               ProgRow         66 *
*                               ms_delay        1 *
* Global Variable(s) Used:     FlashErase      None *
*                               ProgROW         DATA *
*                               ms_delay        None *
* Submodule(s) Called:        EraseRoutine    ms_delay *
*                               ProgRow         ms_delay *
*                               ms_delay        None *
* Calling Sequence:           JSR FlashErase, JSR ProgRow *
*                               JSR ms_delay *
* Entry Label:                 FlashErase, ProgRow, ms_delay *
*
* Entry Conditions:           FlashErase      Flash address defined at *
*                               Index Register X *
*                               ProgRow         Flash address defined at *
*                               Index Register X *
*                               RAM Buffer address defined *
*                               at Index Register Y *
*                               64 programming bytes *
*                               located at variables DATA *
*                               ms_delay        Delay variable passed in *
*                               TIMES *
* Number of Exit Points:      3 *
*   Exit Label:               FlashErase      FlashErase_End *
*                               ProgRow         ProgRow_End *
*                               ms_delay        ms_delay_End *
*   Exit Conditions:         FlashErase      None *
*                               ProgRow         None *
*                               ms_delay        None *

```

Application Note

```
*
* Full Functional Description Of Subroutine:
* SSTflash.srt consists of two primary subroutines called FlashErase*
* and ProgRow. These routines demonstrate SST FLASH erasing and *
* programming algorithms, respectively. The routines also call *
* another subroutine ms_delay. Since delay times must be met *
* precisely for successful FLASH programming and erasing, additional*
* software was added to measure the delay times. This code is *
* included as comments throughout the file. It is recommended that *
* the user verify all delay times before using this software for *
* production.
* Note: Each delay time related to SST FLASH program and erase *
* operations was calculated with a bus frequency of 8MHz. *
*****
*****
*****          SST FLASH Erase Subroutine          *****
*****
FlashErase:
;-----;
; Delay Time Evaluation
; Initialize Port A bit 0 as output high
; bset  PORTA,PA0.      ;Set Port A bit 0
; bset  DDRA,DDRA0.    ;Select output for Port A bit 0
;-----;
sei                      ;Disable maskable interrupts

bset  FEECTL,ERAS.      ;Step 1 - Set the ERAS bit

std  ,X                 ;Step 2 - Write to any FLASH address with
                        ; any aligned word within the page to be
                        ; erased

;-----;
; Delay Time tNVS Evaluation (Time between points S1 and E1)
; Measure low level period on Port A bit 0 pin using a scope
;-----;
; Delay Evaluation: Point S1
; bclr  PORTA,PA0.      ;Clear Port A bit 0

ldaa  #$1B              ;Step 3 - Wait for time tNVS
dbne  A,*               ; 1 + (3 x 27) cycles = 82 cycles
                        ; (10.25us)

; Delay Evaluation: Point E1
; bset  PORTA,PA0.      ;Set Port A bit 0

bset  FEECTL,HVEN.      ;Step 4 - Set the HVEN bit

;-----;
; Delay Time tERAS Evaluation (Time between points S2 and E2)
; Measure low level period on Port A bit 0 pin using a scope
;-----;
; Delay Evaluation: Point S2
; bclr  PORTA,PA0.      ;Clear Port A bit 0

movb  #!8,TIMES         ;Step 5 - Wait for time tERAS (8.0ms)
```

```

        jsr    ms_delay

; Delay Evaluation: Point E2
;   bset  PORTA,PA0.          ;Set Port A bit 0

        bclr  FEECTL,ERAS.    ;Step 6 - Clear the ERAS bit

;-----;
; Delay Time tNVHL Evaluation (Time between points S3 and E3)      ;
;   Measure low level period on Port A bit 0 pin using a scope    ;
;-----;
; Delay Evaluation: Point S3
;   bclr  PORTA,PA0.          ;Clear Port A bit 0

        ldd   #$010B          ;Step 7 - Wait for time tNVHL
        dbne D,*              ; 1 + (3 x 267) cycles = 802 cycles
                                ; (100.25us)

; Delay Evaluation: Point E3
;   bset  PORTA,PA0.          ;Set Port A bit 0

        bclr  FEECTL,HVEN.    ;Step 8 - Clear the HVEN bit

;-----;
; Delay Time tRCV Evaluation (Time between points S4 and E4)      ;
;   Measure low level period on Port A bit 0 pin using a scope    ;
;-----;
; Delay Evaluation: Point S4
;   bclr  PORTA,PA0.          ;Clear Port A bit 0

        ldaa  #$03            ;Step 9 - Wait for time tRCV
        dbne A,*              ; 1 + (3 X 3) cycles = 10 cycles
                                ; (1.25us)

; Delay Evaluation: Point E4
;   bset  PORTA,PA0.          ;Set Port A bit 0

        cli                               ;Enable maskable interrupts
FlashErase_End:
        rts
*****
*****          SST FLASH Programming Subroutine          *****
*****
ProgRow:
;-----;
; Delay Time Evaluation                                          ;
; Initialize Port A bit 0 as output high                          ;
;   bset  PORTA,PA0.          ;Set Port A bit 0                  ;
;   bset  DDRA,DDRA0.        ;Select output for Port A bit 0    ;
;-----;
        sei                               ;Disable maskable interrupts

        bset  FEECTL,PGM.      ;Step 1 - Set the PGM bit

        std  ,X                ;Step 2 - Write to a ROW start address

```

Application Note

```

; with any word data
;-----;
; Delay Time tNVS Evaluation (Time between points S5 and E5) ;
; Measure low level period on Port A bit 0 pin using a scope ;
;-----;
; Delay Evaluation: Point S5
; bclr PORTA,PA0. ;Clear Port A bit 0

ldaa #$1B ;Step 3 - Wait for time tNVS
dbne A,* ; 1 + (3 x 27) cycles = 82 cycles
; (10.25us)

; Delay Evaluation: Point E5
; bset PORTA,PA0. ;Set Port A bit 0

bset FEECTL,HVEN. ;Step 4 - Set the HVEN bit

;-----;
; Delay Time tPGS Evaluation (Time between points S6 and E6) ;
; Measure low level period on Port A bit 0 pin using a scope ;
;-----;
; Delay Evaluation: Point S6
; bclr PORTA,PA0. ;Clear Port A bit 0

ldaa #$0E ;Step 5 - Wait for time tPGS
dbne A,* ; 1 + (3 x 14) cycles = 43 cycles
; (5.375us)

; Delay Evaluation: Point E6
; bset PORTA,PA0. ;Set Port A bit 0
;-----;
;- tFPGM is defined as the total time from writing one data -
;- word to writing the next data word. (labelled "A" below). -
;- For the last word programmed, tFPGM is defined as the -
;- time from writing the data word ("A") to clearing the PGM -
;- bit (in the WriteFLCR routine). Both of these loops -
;- should be executed in a time between 30 and 40 us. -
;- -
;- The word-to-next-word time is 245 cycles (30.6 us). -
;- The word-to-PGM time is 242 cycles (30.25 us). -
;-----;
movb #!32,COUNTER ;Write the total word number per
; ROW, 32, to COUNTER
```

Copy_Loop:

```

movw 0,Y,0,X ;Step 6 - Copy one word data from the
; RAM buffer to the appropriate FLASH
; location ("A")
;-----;
; Delay Time word-to-next-word Evaluation (one loop period ;
; starting from point S7 to point E7) ;
; Measure low level period on Port A bit 0 pin using a scope ;
;-----;
; Delay Evaluation: Point S7, E7
; com PORTA ;Complement Port A bit 0
```

```

;-----;
; Delay Time word-to-PGM Evaluation (Time between points S8 and ;
; E8) ;
; Measure low level period on Port A bit 0 pin using a scope ;
;-----;
; Delay Evaluation: Point S8
; bclr PORTA,PA0. ;Clear Port A bit 0

inx ;Set next Flash programming location
inx
iny ;Set next RAM Buffer location
iny

ldaa #$4C ;Step 7 - Delay, part of tFPGM
dbne A,* ; 1 + (3 x 76) cycles = 229 cycles
; (28.625us)

dec COUNTER ;Step 8 - Repeat step 6 through 8
bne Copy_Loop ; until all the bytes within the row
; are programmed

bclr FEECTL,PGM. ;Step 9 - Clear the PGM bit

; Delay Evaluation: Point E8
; bset PORTA,PA0.
;-----;
; Delay Time tNVH Evaluation (Time between points S9 and E9) ;
; Measure low level period on Port A bit 0 pin using a scope ;
;-----;
; Delay Evaluation: Point S9
; bclr PORTA,PA0. ;Clear Port A bit 0

ldaa #$0E ;Step 10 - Wait for time tNVH
dbne A,* ; 1 + (3 x 14) cycles = 43 cycles
; (5.375us)

; Delay Evaluation: Point E9
; bset PORTA,PA0. ;Set Port A bit 0

bclr FEECTL,HVEN. ;Step 11 - Clear the HVEN bit
;-----;
; Delay Time tRCV Evaluation (Time between points S10 and E10) ;
; Measure low level period on Port A bit 0 pin using a scope ;
;-----;
; Delay Evaluation: Point S10
; bclr PORTA,PA0. ;Clear Port A bit 0

ldaa #$03 ;Step 12 - Wait for time tRCV
dbne A,* ; 1 + (3 x 3) cycles = 10 cycles
; (1.25us)

; Delay Evaluation: Point E10
; bset PORTA,PA0. ;Set Port A bit 0

```

Application Note

```
cli                ;Enable maskable interrupts
ProgRow_End:
    rts
*****
*****          Delay Routine          *****
*****
* This routine generates unit millisecond delay depending on the value in *
* "TIMES". For example if times=1, the delay time is 1ms.                *
* Delay = [{1 + (2 + 3) * 1597 + 2 + 1 + 3 + 1 + 4 + 3} * (TIMES - 1) *
*          {1 + (2 + 3) * 1597 + 2 + 1 + 3 + 1 + 4 + 1 +5}
*          / Bus Frequency
*          = (8000 * TIMES + 3) / 8MHz
* Initializations required:
* - Set a value in "TIMES"
* Values returned:
* - None
*****
ms_delay:
    ldd    #$063E    ;1 cyc.

ms_loop:
    subd   #$01      ;2 cyc.
    bne   ms_loop   ;If the branch is taken, 3 cyc. If the branch
                    ; is not taken, 1 cyc.
    tst   TIMES     ;3 cyc
    nop   ;1 cyc
    dec   TIMES     ;4 cyc.
    bne   ms_delay  ;If the branch is taken, 3 cyc. If the branch
                    ; is not taken, 1 cyc.

ms_delay_End:
    rts            ;5 cyc.
```

EEPROM AUTO Mode Source Code Flowcharts

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

The `AutoRoutine` subroutine follows the flowcharts shown in [Figure 6](#) and [Figure 7](#) closely. The flowcharts for `AutoEEPROM.mrt` and `AutoRoutine` are [Figure 12](#) and [Figure 13](#), respectively.

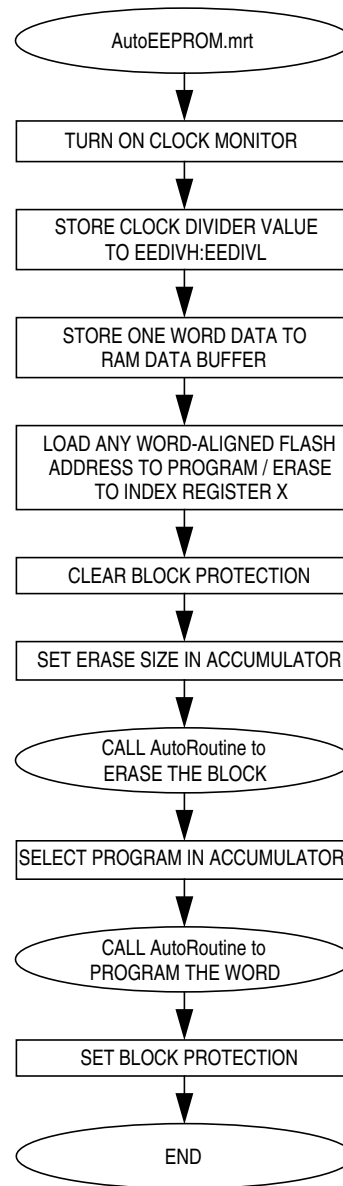


Figure 12. EEPROM AUTO Mode Main Routine

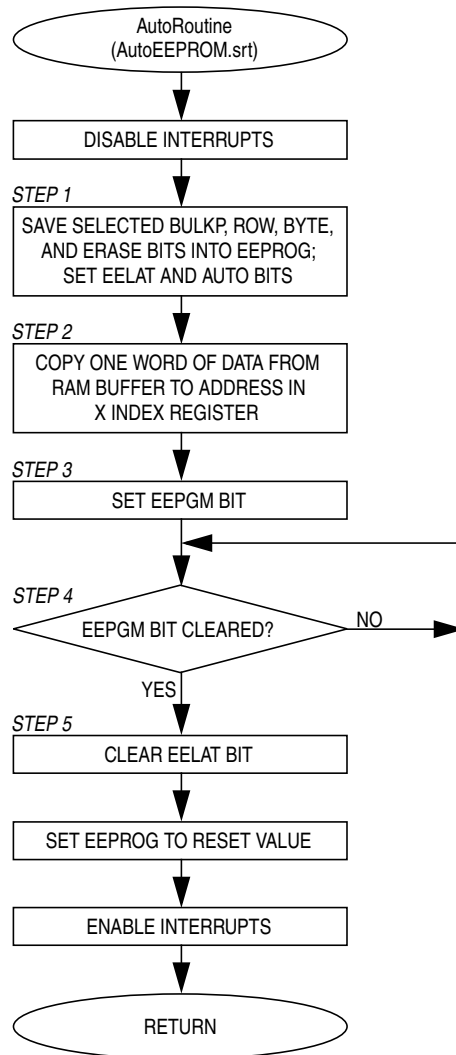


Figure 13. Subroutine AutoEEPROM Flowchart

EEPROM AUTO Mode Assembly Source Code

```
*****
*****
*
*   EEPROM AUTO Programming and Erasing on the MC68HC912DT128A/DG128A
*
*****
* File Name: AutoEEPROM.mrt           Copyright (c) Motorola 2001
*
* Current Revision: 1.0
* Current Release Level: RP
* Current Revision Release Date: July 3, 2001
*
* Current Release Written By: Kazue Kikuchi and Darci Ernst
*                               Motorola Applications Engineering - Austin, TX
*
* Assembled Under: CASM12Z (P&E Microcomputer Systems, Inc.)
*                               Ver.: 3.11
*
* Part Family Software Routine Works With: HC12 0.5u EEPROM Memory
*
* Routine Size (Bytes):          68
* Stack Space Used (Bytes):      2
* RAM Used (Bytes):              2
* Global Variables Used:        DATA
* Subroutine Called:             AutoRoutine
*
* Full Functional Description Of Routine Design:
*   AutoEEPROM.mrt is the main routine for the EEPROM programming and
*   erasing operations using the AUTO mode. It demonstrates AUTO
*   programming and erasing for the MC68HC912DT128A and
*   MC68HC912DT128A.
*   Note: The EEDIV value was calculated with an oscillator frequency of
*   16MHz. The user must recalculate the EEDIV value to use a
*   different oscillator frequency.
*****
*****
*****          Program Specific Equates          *****
*****
auto_BULKerase.   equ    %00100110  ;Select BULK Erase: set AUTO, ERASE,
                                ; EELAT bits, and clear BULKP bit
auto_ROWerase.   equ    %00101110  ;Select ROW Erase: set AUTO, ROW,
                                ; ERASE, EELAT bits, and clear BULKP
                                ; bit
auto_WORDerase.  equ    %10110110  ;Select WORD Erase: set AUTO, BYTE
                                ; ERASE and EELAT bits
auto_WORDprogram. equ    %10100010  ;Select WORD Program: set AUTO and
                                ; EELAT bits
*****
*****          Include Files          *****
*****
```

Application Note

```
NOLIST
$INCLUDE      "912DG128A_memory.frk"    ;Equates for the MC68HC912DG/DT128A
                                           ; registers and bits used in this
                                           ; routine

      org     RAM
$INCLUDE      "AutoEEPROM.var"          ;RAM variable definitions
LIST
*****
*****                               Main Routine                               *****
*****
      org     RAM+$50                   ;The code start address
                                           ;The RAM below this address is used
Start:                                               ; as the EEPROM data buffer and spare
                                           ; data storage

      lds     #$4000                     ;Set Stack Pointer

      movb   #$8F,COPCTL                 ;Enable Clock Monitor
                                           ; If a loss of clock is detected, take
                                           ; appropriate action. Other bits may be
                                           ; set/cleared for user application.

      movw   #$0230,EEDIVH               ;If SHADOW word does not set a constant
                                           ; timebase of 35us, write $02 AND $30
                                           ; to EEDIVH and EEDIVL, respectively
                                           ; since the oscillator frequency is
                                           ; 16MHz
                                           ; <CAUTION> When EEDIVH:EEDIVL=00:00, the
                                           ; EEPROM bit is NOT automatically cleared
                                           ; These registers are "write once"
                                           ; in normal mode

      movw   #$55AA,DATA                 ;Write one word data $55AA to RAM buffer

      ldx     #$0E02                     ;Load Index Register X with address of
                                           ; where the aligned word should be
                                           ; erased and programmed

      bclr   EEPROT,BPROT3.              ;Unprotect the block which will be
                                           ; erased and programmed
                                           ; <CAUTION> When the programming or
                                           ; erasing location is protected (except
                                           ; with bulk erase), the EEPROM bit is
                                           ; NOT automatically cleared

      ldaa   #auto_WORDerase.            ;Select Bulk, Row or Word Erase
      jsr   AutoRoutine                  ;Erase the selected EEPROM size using
                                           ; AUTO Mode

      ldaa   #auto_WORDprogram.          ;Select Word Program
      jsr   AutoRoutine                  ;Program one word using AUTO Mode

      bset   EEPROT,BPROT3.              ;Protect the programmed block

      bra    *
```


Application Note

```

*                               Ver.: 3.11                               *
*                                                                           *
* Part Family Software Routine Works With: HC12 0.5u EEPROM Memory      *
*                                                                           *
* Module Size (Bytes):          AutoRoutine      27                       *
* Stack Space Used (Bytes):     AutoRoutine      0                       *
* RAM Used (Bytes):             AutoRoutine      2                       *
* Global Variable(s) Used:     AutoRoutine      DATA                   *
* Calling Sequence:             JSR AutoRoutine                                     *
* Entry Label:                  AutoRoutine                                     *
* Entry Conditions:             AutoRoutine      1 byte setup (BULKP, AUTO, *
*                               BYTE, ROW, ERASE, EELAT                       *
*                               bits) defined at                             *
*                               accumulator A *                               *
*                               2 bytes address defined at                   *
*                               Index Register X                             *
*                               1 programming word located                   *
*                               at variable DATA (the                       *
*                               erasing operation is not                     *
*                               required)                                   *
* Number of Exit Points:        1                                             *
*   Exit Label:                 AutoRoutine      AutoRoutine_End           *
*   Exit Conditions:            AutoRoutine      None                       *
*                                                                           *
* Full Functional Description Of Subroutine:                               *
*   AutoEEPROM.srt contains one primary subroutine called AutoRoutine.    *
*   This demonstrates EEPROM erasing and programming in the AUTO mode.    *
*   ****                                                                     *
*   ****      EEPROM AUTO Program and Erase Subroutine      ****          *
*   ****                                                                     *
AutoRoutine:
    sei                ;Disable maskable interrupts
    staa  EEPROG       ;Step 1 - Select BULKP, BYTE, ROW, and
                      ; ERASE bits, and set EELAT and AUTO bits

    ldd  DATA         ;Step 2 - Copy one word of data from RAM
    std  ,X            ; buffer to address specified in X
                      ; index register

    bset  EEPROG,EEPGM. ;Step 3 - Set EEPGM bit

Clear_EEPGM:          ;Step 4 - Wait until EEPGM bit is cleared
    brset EEPROG,EEPGM.,Clear_EEPGM

                      ;<CAUTION> When EEDIVH:EEDIVL=00:00 or a
                      ; programming/erasing location is
                      ; protected (except for bulk erase),
                      ; the EEPGM bit is NOT automatically cleared

    bclr  EEPROG,EELAT. ;Step 5 - Clear EELAT bit


    movb  #$80,EEPROG  ;Write a reset value to EEPROG register

    cli                ;Enable maskable interrupts

AutoRoutine_End:
    rts

```


Application Note

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola 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 Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola 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 Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution; P.O. Box 5405, Denver, Colorado 80217. 1-303-675-2140 or 1-800-441-2447

JAPAN: Motorola Japan Ltd.; SPS, Technical Information Center, 3-20-1, Minami-Azabu, Minato-ku, Tokyo 106-8573 Japan. 81-3-3440-3569

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong. 852-26668334

Technical Information Center: 1-800-521-6274

HOME PAGE: <http://www.motorola.com/semiconductors/>



MOTOROLA

© Motorola, Inc., 2001

AN2166/D